Rapport de projet de fin d'études
Création d'un système domotique sans fil

Soutenance finale

Thomas Maurice
Benoit Maliar

École Polytechnique Universitaire de Lille
Département IMA

Thomas.Maurice@polytech-lille.net
Benoit.Maliar@polytech-lille.net

24 février 2015

POLYTECH LILLE

Université Lille1 Sciences et Technologies

Table des matières

Introduction

Dans le cadre de notre projet de fin d'études de cinquième année en Informatique, Microélectronique, Automatique, nous avons été amenés à travailler sur la création d'un système domotique sans fil.

L'objectif est de créer ce système à l'aide d'une raspberry pi et de petits systèmes embarqués à base de microprocesseur et d'antenne pour permettre une production *low cost*, reproductible chez soi et modifiable à souhait.

Ce projet se découpe en deux parties faisant appel à nos compétences en Informatique et en Électronique. La première se focalise sur la construction de la carte électronique et la deuxième sur la création et l'implémentation du système d'exploitation dans le microprocesseur et de l'interface d'administration des données.

Dans un premier temps, nous allons effectuer un rappel du cahier des charges puis détailler nos réalisations matérielles et logicielles. Enfin, nous allons procéder à une analyse des difficultés que nous avons rencontrées.

1 Rappel du cahier des charges

1.1 Idée générale

Le but du projet est de réaliser un réseau de capteurs distribués à base de microprocesseurs MSP430 qui feront remonter des informations fournies par différents senseurs (gaz, fumée, température, luminosité) à une Raspberry Pi. Cette dernière traitera ces données et permettra à un utilisateur de les consulter. Le but est que l'importe quel capteur peut être branché à la carte et qu'elle s'y adapte.

1.2 Partie capteur

Chaque carte doit permettre l'envoi de données à l'aide de composants radio sur deux bandes de fréquences afin de pouvoir utiliser une autre bande si celle actuellement utilisée est surchargée ou parasitée. Le but étant d'avoir un système fonctionnel même si un éventuel cambrioleur décide de brouiller la bande de communication par défaut.

Nous avons choisi d'utiliser pour chaque carte un CC1101 et un CC2520. Le premier permet de communiquer sur la bande 868/915 MHz et le second sur la bande 2.4 GHz (voisine du WiFi). La bande la plus basse offre l'avantage d'une meilleure portée, au détriment du débit d'information.

Ces deux modules radio seront connectés à un microprocesseur MSP430 sur lequel sera installé un système d'exploitation pour connecter la carte sur un réseau (Contiki). La carte doit être capable d'identifier les différents senseurs qui lui seront branchés, afin de se reflasher de manière dynamique des parties du MSP430 de manière à ce qu'il dispose des drivers nécessaires à piloter le nouveau senseur. Les drivers seraient stockés sur la Raspberry Pi.

1.3 Partie serveur

Les données envoyées par les capteurs seront récupérées par une carte identique aux autres, mais reliée à une Raspberry Pi en SPI qui collectera toutes les informations dans une base de données PostgreSQL et qui offrira, via une interface de visualisation/administration, la possibilité d'interagir avec elles.

2 Ce qui a été réalisé

2.1 Partie Electronique

2.1.1 PCB de la carte v1

Nous avons réalisé complètement le PCB de la carte à partir de rien. Nous avons donc adapté les reference designs de chacun des composants principaux du projet de manière à les intégrer aux mieux. Ces composants principaux sont notamment :

- Le MSP430F5418A
- Un CC1101 pour la communication à 838MHz
- un CC2520 pour la communication à 2.4GHz
- Une LED RGB
- Les connecteurs HE10
- Un header SPI pour la communication avec la Raspberry Pi

Pour communiquer avec les cartes filles, nous sommes partis sur un bus universel en 9 signaux, comprenant alimentation, masse, port série, deux chips select, une entrée tout ou rien et une entrée analogique. De cette sorte la carte mère sera capable de traiter tout type de capteur.

En plus de tous ces composants, nous avons intégré tous les composants annexes nécessaires au bon fonctionnement de l'ensemble, tels que des baluns pour la partie RF, des oscillateurs pour le microcontrôleur et diverses capacités de découplages et résistances de limitation de courant.

Ce grand nombre de composants implique que la détermination du schéma électrique de l'ensemble a été assez longue et chaotique, les étapes de validation des diverses versions du schéma par les encadrants prenant du temps. De la même manière, l'étape de routage n'a pas été aisée et s'est déroulée d'octobre à décembre. En effet, il nous a fallu prendre en compte des contraintes de placement telles que la proximité des oscillateurs avec leurs composants liés, la proximité des capacités de découplages et l'éloignement au maximum des antennes des composants radio du reste du circuit de manière à ne pas subir ni émettre trop d'interférences.

Une fois la carte conçue et routée, il a fallu la faire produire, pour ce faire nous avons eu recours au laboratoire IRCICA. Le technicien de l'IRCICA nous a donc tiré la première version de la carte, et nous a soudé quelques composants dessus (le MSP430, les CCXXXX et un balun (le mauvais)). Nous avons soudé quelques autres composants de manière à pouvoir faire nos tests (capacités de découplage, port USB, résistances et LED). Malheureusement, cette carte souffrait de quelques problèmes de conception (notamment au niveau de l'alimentation), mais surtout un incident s'est produit lors du tirage, et un des bus d'alimentation n'a pas été détouré et était donc fondu dans le plan de masse. Il a donc fallu opérer la carte à coups de cutter et de Dremel de manière à essayer de la rendre exploitable.

Malheureusement, ce genre de modifications peu chirurgicales de la carte n'ont pas eu les effets escomptés et nous n'avons jamais été capables d'en faire quoi que ce soit d'un point de vue de la programmation. En revanche, d'un point de vue électronique nous avons été capables de repérer un grand nombre d'erreurs de conception et de routage, parfois liées aux limitations

de la machine. Entre autres certains vias trop fins pour être percés, des pistes trop fines, des capacités de découplages et des quartz trop loin de leur composant... Ces erreurs et manques d'optimisations ont été en partie corrigés dans la version suivante de la carte.

2.1.2 PCB de la carte v2

Ayant pris note de ce qui n'avait potentiellement pas fonctionné dans la v1, nous avons commencé à élaborer une version 2 de la carte, en corrigeant notamment les erreurs de conception de l'alimentation et les quelques défauts de routage. Cette version deux de la carte a été également confiée aux bons soins de l'IRCICA. Malheureusement, comme pour la version une de la carte, la même piste d'alimentation que la première version a subi un problème de détourage, et s'est donc une nouvelle fois retrouvée fondue dans le plan de masse. Il a donc été une fois de plus été nécessaire de re détourer cette piste au cutter et à la Dremel de manière à essayer de la rendre exploitable.

Cette carte n'a jamais été fonctionnelle non plus. Malgré la semaine de débug intensive que nous y avons passé, comme il sera détaillé en dernière partie de ce présent rapport.

2.2 Cartes de test

2.2.1 Carte mère de test

Malheureusement ces deux cartes tirées par l'IRCICA n'étaient pas fonctionnelles, comme il sera détaillé en dernière partie, nous avons donc dû en concevoir une version *dégradée* pour nos tests. Le principe de la carte dégradée est d'offrir une interface de programmation simple via un port USB et de déporter la totalité des pins du MSP430 à l'extérieur, de manière à pouvoir enficher d'éventuelles cartes filles dessus, une fois que le fonctionnement de la carte sera validé.

L'intérêt de procéder de la sorte est de se doter d'une plateforme de développement relativement générique, de manière à pouvoir venir y brancher petit à petit les différents composants du projet, notamment les cartes RF, et de s'assurer que chaque partie fonctionne comme elle doit fonctionner avant de continuer dans le projet.

Cela permet également dans le futur d'utiliser cette carte pour d'autres projets puisqu'elle est extrêmement générique.

Le schematic de cette carte est disponible en annexe de ce rapport.

2.2.2 Carte fille RF à base de CC2520

Dans le même temps, nous avons élaboré une carte fille à la manière d'un *shield* Arduino comportant le composant radio CC2520 que l'on pourra enficher sur le MSP430 une fois que le fonctionnement de la carte aura été validé (i.e. quand nous aurons réussi à flasher avec succès un programme dessus). Cette approche plus modulaire de la conception nous permet d'évaluer au cours du développement à quels endroits les éléments ne fonctionnent pas, plutôt que de se retrouver avec une carte complète, mais non fonctionnelle.

Le schematic de cette carte est disponible en annexe de ce rapport.

2.3 Partie Informatique

2.3.1 Choix technologiques

Le SGBD choisi est PostgreSQL, puisqu'il est entièrement Open Source (par opposition à MySQL propriété d'Oracle). Le serveur applicatif pour lequel nous avons opté est Node.js, qui permet d'exécuter du code JavaScript côté serveur.

Node.js est basé sur le moteur V8 de Google (utilisé notamment par Google Chrome) et permet une exécution rapide et asynchrone de code Javascript. C'est notamment pour cette capacité à exécuter le code de manière asynchrone, et donc optimisée d'un point de vue du temps de traitement que nous avons opté pour Node.js. En effet, la Raspberry Pi étant monocoeur, nous ne pouvions pas opter pour une solution basée par exemple sur PHP, et allouer plusieurs processus à un PHP-FPM pour en augmenter les performances.

Un autre avantage de Node.js est qu'un très grand nombre de modules est disponible via le gestionnaire de paquetages *npm* et permet de simplifier de manière drastique le développement en fournissant des modules pour gérer les identifications, les templates, les connexions à la base de données... La contrainte étant d'adopter un style de programmation clair et strict et de ne pas céder à la tentation de faire n'importe quoi, comme la syntaxe de JavaScript le permet.

Pour prendre en charge la partie graphique et design, nous avons opté pour le toolkit graphique Twitter Bootstrap, qui est objectivement beau et facile à intégrer et à prendre en main. Il se distingue également des autres (INK) par son poids très léger et par la qualité de sa documentation.

Au niveau du système de construction de l'application, nous avons opté pour la suite classique en développement Node.js : npm, bower et Grunt en tant que système de build.

2.3.2 Fonctionnalités de l'application Web

L'application sur la Raspberry Pi permet à l'utilisateur de visualiser les données des capteurs de manière simple et intuitive. Elle permet de restreindre l'accès aux fonctionnalités du site aux utilisateurs identifiés et propose des fonctions basiques de gestion de compte.

Le rafraîchissement de la page de présentation est prévu pour se faire à la demande de l'utilisateur (alt+r). Cela est géré par jQuery et des appels à une API REST qui nous permet d'aller récupérer les valeurs des différents capteurs de la base de données. De cette manière on a une interface dynamique et une facilité d'utilisation accrue pour l'utilisateur. Il est également possible à l'utilisateur d'afficher les détails d'un capteur individuellement. On obtient ainsi de jolis graphes permettant de se rendre compte de l'évolution de telle ou telle métrique.

Les métriques sont identifiées en base de données par des trigrammes, ainsi quand l'application trouve une métrique "TMP" ou "HUM" en base, il sait que ce n'est pas la même. De cette manière, l'application n'a absolument pas besoin de savoir quels types de métriques seront entrés à l'avance, elle s'adaptera simplement en fonction des trigrammes. De même il est possible de spécifier des manières de *rendre* visuellement les trigrammes au niveau de l'application Web en le spécifiant dans le fichier de configuration `app.json` de l'application. Ainsi si l'on prend la valeur 33, si elle est interprétée comme une métrique TMP elle sera rendue : `La`

`température est de 33°C`, alors que dans le cas d'une métrique HUM on aura quelque chose comme `L'humidité est de 33%`

Dans sa version finale, l'application propose les graphes des différentes métriques remontées en fonction du temps,et des statistiques (valeurs moyennes).

2.3.3 Interface et ergonomie

Après concertation avec Pierrick Buret, le design de l'interface a été un petit peu repensé, notamment en ce qui concerne l'affichage des mesures. En effet, sur les bandeaux déroulants nous affichons en plus du nom de la pièce, une moyenne effectuée sur les trois ou quatre dernières valeurs de chaque métrique remontée, ce qui permet d'avoir une bonne idée de l'état de chaque pièce, ainsi que de pallier à d'éventuelles erreurs de mesure en les moyennant. Notons que les données de la figure 1 ci-jointe sont *générées aléatoirement en base* pour le moment et servent ici à titre d'illustration. Le point important est qu'au login l'utilisateur peut d'un simple balayage de l'oeil vérifier que tout va bien ou pas à un instant t. Dans un second temps, s'il le souhaite il peut descendre dans le détail plus finement en dépliant le bloc correspondant à la pièce qui l'intéresse et en affichant le graphique qui l'intéresse, comme sur la figure 2.

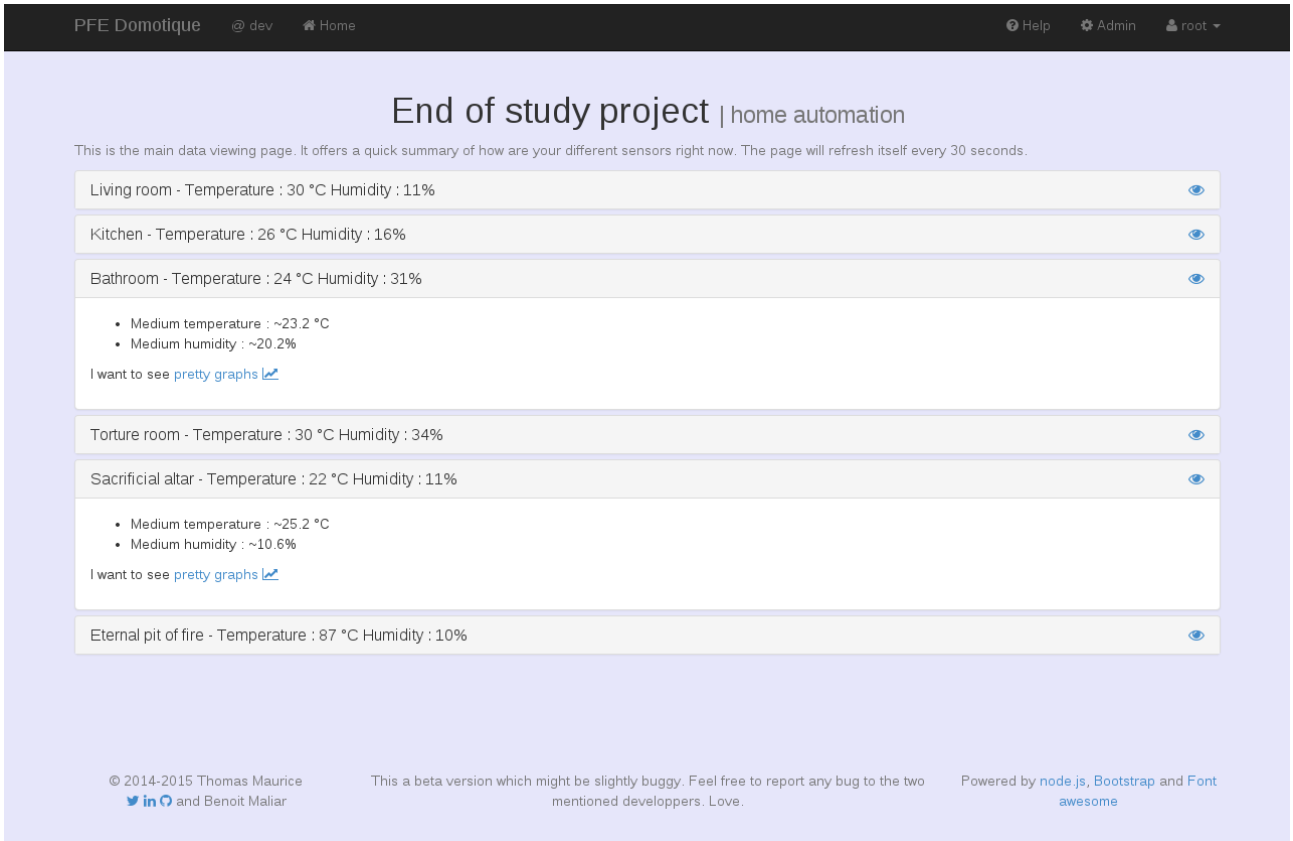


Figure 1 – Aperçu de l'interface

2.3.4 Fonctionnalités disponibles le jour de la soutenance

Les fonctionnalités suivantes sont implémentées
— Gestion des comptes utilisateurs et administrateur

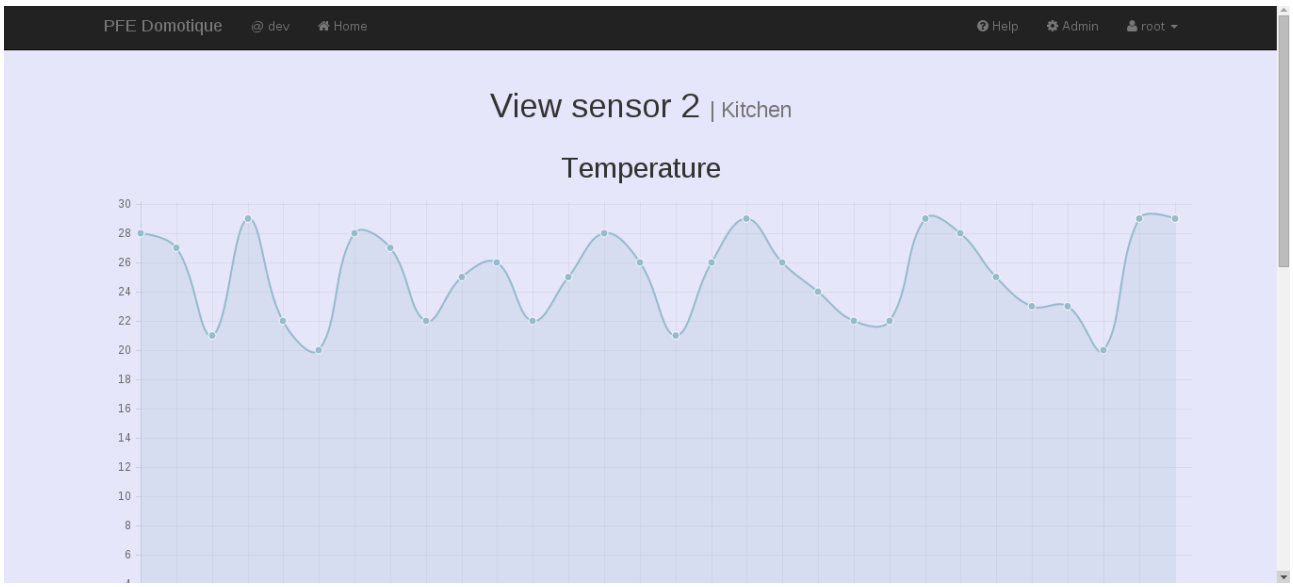


Figure 2 – Aperçu de la visualisation d'un capteur

— Ajout/suppression de capteurs
— Renommage des capteurs et spécification de leurs IDs
— Affichage des valeurs récentes en page d'accueil
— Rafraîchissement automatique et manuel de ces valeurs
— Affichage de graphiques par capteurs et par métrique
— Calcul de moyennes sur les dernières valeurs
— Affichage de la progression du chargement des valeurs

Les améliorations par rapport à la soutenance de mi-projet concernent surtout des fonctionnalités de l'interface utilisateur, à la demande de Pierrick Buret. Son retour a été très important pour nous dans la mesure ou il ne nous était pas évident de savoir précisément ce qu'il est pertinent d'afficher ou non, et ses conseils nous ont permis d'optimiser notre interface existante pour en améliorer l'ergonomie et la fonctionnalité. De plus des améliorations de stabilités ont été apportées (les crashs un peu aléatoires ont été réparés) et le code a été intégralement commenté, ce qui n'avait pas été le cas au début du projet par manque de temps.

2.3.5 Méthode de développement

Pour développer l'application Web, comme nous avons fait pour la carte, ainsi que pour le présent rapport d'ailleurs, nous utilisons le système de versionnement Git afin de synchroniser notre progression l'un avec l'autre. Le dépôt est hébergé sur l'un de nos serveurs.

Le développement s'articule autour d'une liste de fonctionnalités que nous traitons de manière séquentielle, et nous réalisons des tests fonctionnels à chaque étape du développement, de manière à nous assurer que chaque fonctionnalité fonctionne comme elle le doit, avant de s'attaquer à la suivante. Cela nous assure un code relativement robuste et un peu éprouvé.

Une documentation de l'application Web a bien évidemment été rédigée, et est disponible en annexe de ce rapport. Il a également été réalisé une image sur carte SD du système fonctionnel et prêt à être déployé sur une Raspberry Pi, l'application et les serveurs divers se lançant

automatiquement au démarrage de la Raspberry.

2.3.6 Portage de Contiki sur le MSP430F5418A

La seconde tâche que nous avons dû accomplir d'un point de vue informatique est le portage du système d'exploitation libre Contiki sur notre plateforme électronique. L'intérêt d'utiliser Contiki dans notre projet est que le système supporte nativement un certain nombre de choses intéressantes, telles que l'IPv6, le protocole 6loWPAN, et un certain nombre de composants radio tels que notre CC2520. Il permet entre autres de faire des applications coexistant au sein du même microprocesseur en pseudo-multithreadé via des protothreads.

Il était donc très intéressant pour nous de l'utiliser puisque la maintenabilité de notre code en serait théoriquement accrue. En nous inspirant des fichiers de portages de la plateforme TMote, assez similaire à la nôtre (modulo le microcontrôleur qui n'est pas exactement le même). Nous avons réussi à compiler quelques programmes d'exemple avec notre portage via le cross-compilateur msp430-gcc, mais nous n'avons malheureusement pas pu le tester dans la mesure ou nos deux premières cartes de développement n'étaient pas fonctionnelles. Le travail que nous avons fait sur Contiki s'est donc arrêté très tôt, faute de plateforme matérielle disponible pour tester ce que nous avions programmé.

Le détail des problèmes que nous avons eus avec Contiki sera donné en dernière partie de ce rapport de projet.

3 Problèmes rencontrés et solutions/enseignements tirés

Cette section a pour but d'expliquer le manque de réalisations techniques tangibles et de montrer comment nous avons su néanmoins faire face aux très nombreux problèmes tant techniques qu'organisationnels auxquels nous avons été confrontés tout au long du projet.

3.1 Manque de pratique en conception de carte

Dans un premier temps, le problème principal que nous avons rencontré est notre manque de pratique et d'expérience dans la conception de circuits électroniques, et dans le routage. En effet, nous n'avions pas certains réflexes de base tels qu'aller chercher les reference design des composants que nous utilisions (nous allions chercher toutes les informations dans la datasheet au lieu d'utiliser les reference design déjà faits). De même, nous manquions de pratique dans tout ce qui est détermination des valeurs des capacités de découplage pour l'alimentation des divers éléments. Ce manque d'expérience a fait que des tâches pas fondamentalement compliquées nous ont pris bien plus de temps que si nous y avions été habitués. Cela a notamment induit un retard de plus de deux semaines pour le routage de la première version de la carte.

Cette partie conception de circuit fut difficile, mais néanmoins fut la plus formatrice pour nous. Nous avons de ce fait gagné en compétence sur le logiciel de CAO Eagle, aussi bien d'un point de vue schematic/conception d'empreintes que du point de vue du routage. Les différents échanges que nous avons eu l'occasion d'avoir avec Pierrick Buret, Alexandre Boé et Thierry Flamen nous ont permis d'acquérir les bonnes pratiques en matière de routage, de CEM et d'organisation de nos circuits.

3.1.1 Tirage de la première carte

Le second gros problème de notre PFE a été le tirage de la première version de la carte. En effet, outre les quelques erreurs de conception non critiques que nous avions commises (tailles de vias par exemple), nous nous sommes aperçus que la machine de l'IRCICA n'avait pas correctement détouré certaines pistes. Malheureusement, la piste en question était une bus d'alimentation qui s'est de ce fait retrouvé complètement fondu dans le plan de masse sur toute la longueur de la carte. Afin de corriger cela, nous avons dû retoucher la carte au cutter et à la Dremel de manière à essayer de la rendre exploitable, sans succès.

La carte n'étant pas exploitable, nous avons décidé d'en refaire tirer une seconde, en en profitant pour corriger les erreurs de conception présentes sur la première version. Nous avons notamment corrigé la taille de certains vias, rapproché un quartz de son microcontrôleur et rapproché des capacités de découplage de leurs composants respectifs. Nous avions donc une version améliorée de la première carte.

3.1.2 Tirage de la seconde carte

Cette carte étant conçue, nous avons donc été la faire tirer à l'IRCICA à nouveau. Malheureusement, même après une vérification minutieuse des gerbers pour s'assurer que la piste

qui n'avait pas été détourée au premier tirage était présente, le second tirage possédait le même défaut. Cela était relativement incompréhensible puisque la seule hypothèse qui a été avancée était une limitation du logiciel Eagle dans sa version d'essai, qui n'est pas présente sur le site de l'éditeur.

Comme pour la première version, il nous a fallu corriger les défauts à la Dremel. Ce faisant, il est possible que la meule ait projeté sur les pistes des particules de cuivre qui ont engendré beaucoup de micro courts circuits qui ont rendu le débug plus que laborieux. Il a donc fallu sectionner et isoler des pans entiers du circuit de manière à avoir une bonne isolation entre la masse et VCC.

Malgré cela, nous n'avons toujours pas été capables de flasher le moindre code sur la carte.

Plusieurs hypothèses ont été avancées et essayées. D'abord, nous suspections que notre version du microcontrôleur, le MSP430F5418A n'était pas compatible avec le flasheur que nous utilisions pour le programmer (un Launchpad de chez TI), mais internet nous a montré que c'était en fait possible au travers d'exemples de projets utilisant ce microcontrôleur. Nous avons ensuite pensé qu'un court circuit entre deux pattes du composant pouvait avoir été à l'origine de l'incapacité de flasher un programme, il a donc été nécessaire de tester deux à deux les pattes du composant pour vérifier à la fois leur isolation entre elles, et le fait qu'elles étaient bien reliées au reste du circuit, sans succès. Une fois que les courts circuits entre l'alimentation et la masse ont été réglés, nous avons essayé d'alimenter la carte de manière à observer comment elle se comportait, notamment en regardant son comportement via un oscilloscope, de manière inexplicable le quartz du MSP n'oscillait absolument pas, malgré les bonnes valeurs de capacités de charge extraites de la datasheet.

3.1.3 Solution : conception modulaire

Suite à ces deux échecs et face au manque de temps et au manque de disponibilités du technicien de l'IRCICA sur la fin du projet, nous sommes partis sur une reconception modulaire du projet, afin d'identifier les problèmes sur des portions de circuits bien spécifiques avec le moins de composants possible. Nous avons donc décidé de faire fabriquer à Polytech deux cartes distinctes :

— Une carte mère comprenant le MSP et les pins déportés
— Une carte fille contenant le composant radio CC2520

De cette sorte nous pouvons analyser quelle partie du circuit pose problème. Cette conception modulaire nous permet de valider le fait que certaines parties du projet fonctionnent ou non, et de déterminer avec précision les endroits où se situent les problèmes. Ces cartes ont été conçues lors de la dernière semaine du projet.

Les schematics de ces deux cartes sont disponibles en annexe de ce rapport.

3.1.4 Tirage des trois dernières cartes

Ces cartes modulaires, bien que conçues par nos soins n'ont pas pu être tirées à l'école. Pour le premier essai, c'est parce que le technicien a utilisé des plaques étamées qui n'étaient pas forcément de très bonne qualité, et que la machine n'a pas été recalibrée pour prendre en compte les quelques microns d'épaisseur en plus induits par la couche d'étain.

Pour le second essai, le même problème plus ou moins est survenu. Cette fois-ci une plaque de cuivre vernie a été utilisée, mais l'épaisseur devait être encore trop grande pour que le fraisage puisse se faire correctement. Cette version présentait donc encore des courts-circuits, et des pistes mal détourées.

Pour le dernier essai, la gravure ne s'est pas passée correctement non plus puisque les fraises fines que nous utilisions sont arrivées en fin de vie. La carte n'a donc pas pu être réalisée correctement à cause de l'état d'usure avancé de la fraise.

La conclusion de cela est qu'il aurait fallu utiliser des fraises RF, non disponibles à l'école, et des plaques de 18 microns de cuivre au lieu de 35, que nous avions utilisées jusqu'à présent.

3.2 Portage de Contiki

En parallèle de la conception de cartes, il nous a fallu nous pencher sur le portage du système d'exploitation embarqué Contiki afin de l'intégrer dans le projet. Cette partie a posé problème du fait du manque cruel de documentation pour réaliser un portage (tous les tutoriaux disponibles sont en effet orientés sur l'utilisation de l'OS, mais jamais sur l'adaptation sur une nouvelle plateforme). Même en se basant sur des portages existants, il est extrêmement difficile de comprendre de quoi il retourne, dans la mesure où les codes sources disponibles ne sont absolument pas commentés.

La seconde difficulté vient du fait que lors du portage, nous avons trouvé du code non *hardware indépendant* dans des parties du code qui théoriquement devait l'être (présence de registres non présents sur notre MSP). De ce fait il a fallu modifier le code de manière à définir artificiellement ces registres de manière à ne plus avoir d'erreurs de compilation.

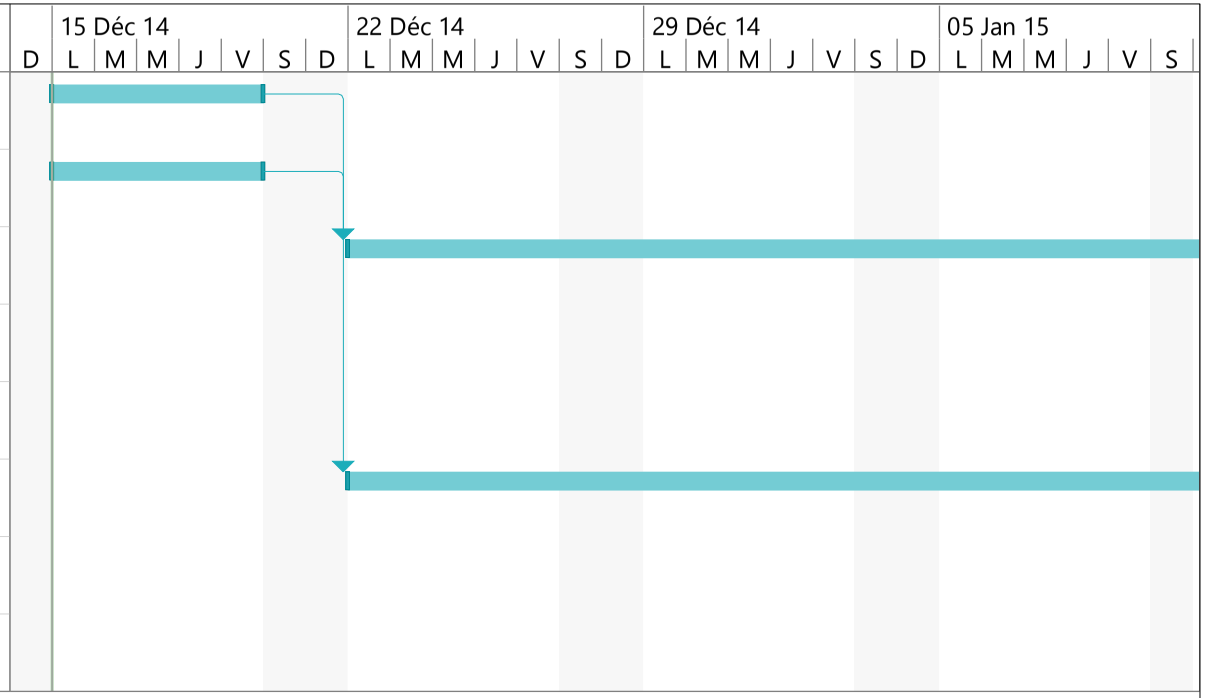
Pour contourner la difficulté, nous avons copié et épuré les fichiers le portage utilisé pour la plateforme TMote en enlevant les éléments dépendants du matériel.

3.3 Comparatif avancée prévisionelle/réelle

3.3.1 Avancée prévisionelle

Lors de la soutenance de mi-projet, nous étions sur le point de tirer la première version de la carte. Le Gantt ci-dessous résume les étapes que nous avions prévues.

N°	Nom de la tâche	Durée	Début	Fin
1	Production de la carte	5 jours	Lun 15/12/14	Ven 19/12/14
2	Application Web fonctionnelle	5 jours	Lun 15/12/14	Ven 19/12/14
3	Soudure des composants	17 jours	Lun 22/12/14	Mar 13/01/15
4	Avancée sur Contiki	15 jours	Mer 14/01/15	Mar 03/02/15
5	Interfaçage MSP/Raspberry	7 jours	Mer 04/02/15	Jeu 12/02/15
6	Amélioration application Web	49 jours	Lun 22/12/14	Jeu 26/02/15
7	Réglage Contiki	10 jours	Ven 13/02/15	Jeu 26/02/15
8	Soutenance projet	0 jour	Ven 27/02/15	Ven 27/02/15



Projet : PFE
Date : Lun 15/12/14

Tâche
Fractionnement
Jalon
Récapitulative
Récapitulatif du projet
Tâche inactive
Jalon inactif
Récapitulatif inactif
Tâche manuelle
Durée uniquement
Report récapitulatif manuel
Récapitulatif manuel
Début uniquement
Fin uniquement
Tâches externes
Jalons externes
Échéance
Avancement
Progression manuelle

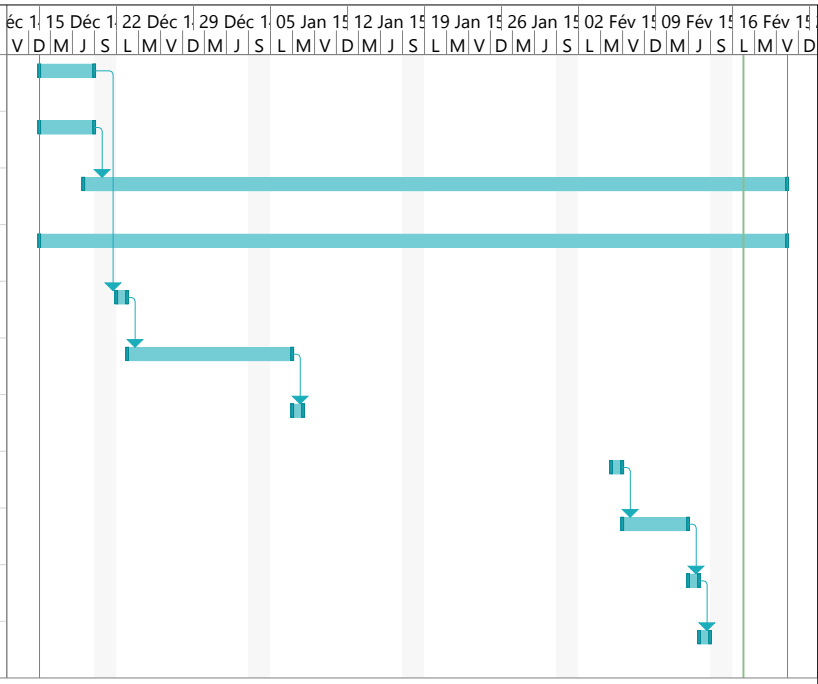
La carte aurait dû être prête pour la mi-janvier afin de pouvoir tester Contiki sur notre plateforme et d'y travailler jusqu'à la fin.

3.3.2 Avancée réelle

L'avancée réelle du projet ne correspond pas à ce que nous avions prévu. La raison de cette différence concerne le tirage de la carte qui a posé plus d'un problème.

Le Gantt ci-dessous résume l'avancée réelle du projet

N°	Nom de la tâche	Durée	Début	Fin
1	Production carte V1	5 jours	Lun 15/12/14	Ven 19/12/14
2	Application Web Fonctionnelle	5 jours	Lun 15/12/14	Ven 19/12/14
3	Amélioration application Web	46 jours	Ven 19/12/14	Ven 20/02/15
4	Contiki	50 jours	Lun 15/12/14	Ven 20/02/15
5	Soudure composants principaux	1 jour	Lun 22/12/14	Lun 22/12/14
6	Debug et amélioration carte	11 jours	Mar 23/12/14	Mar 06/01/15
7	Envoi V2	1 jour	Mer 07/01/15	Mer 07/01/15
8	Tirage et soudure carte V2	1 jour	Jeu 05/02/15	Jeu 05/02/15
9	Debug V2	4 jours	Ven 06/02/15	Mer 11/02/15
10	Création plateforme de test	1 jour	Jeu 12/02/15	Jeu 12/02/15
11	Tirage plateforme de test	1 jour	Ven 13/02/15	Ven 13/02/15



Projet : PFE_final
Date : Mar 17/02/15

Tâche	Récapitulatif inactif	Tâches externes
Fractionnement	Tâche manuelle	Jalons externes
Jalon	Durée uniquement	Échéance
Récapitulative	Report récapitulatif manuel	Avancement
Récapitulatif du projet	Récapitulatif manuel	Progression manuelle
Tâche inactive	Début uniquement	
Jalon inactif	Fin uniquement	

Le premier tirage de la carte n'a pas été concluant puisque nous n'avons rien pu en tirer. Une deuxième carte a dû être tirée et a donc ralenti fortement notre avancée sur Contiki. Malheureusement, la deuxième carte n'a pas pu être utile à Contiki, car elle avait les mêmes soucis que la première.

3.3.3 Résumé

L'avancée sur Contiki ne correspond pas à ce que nous attendions, nous avons développé la plateforme Contiki pour notre carte, mais sans être sûr qu'elle fonctionne (faute d'avoir la carte).

La carte a été développée, mais elle demande quelques tests supplémentaires (via les petites cartes de test) pour être sûr de situer le problème.

L'interface Web est prête et améliorée selon le planning.

Conclusion

En conclusion, ce projet a été pour nous très intéressant et particulièrement formateur dans la mesure ou nous avons pu approfondir de manière poussée des techniques que nous ne maîtrisions pas, notamment au niveau de la conception de circuits électroniques.

En revanche, il a été particulièrement frustrant puisque notre progression n'a eu de cesse d'être entravée par des soucis matériels totalement indépendants de notre volonté, qui nous ont empêchés de mener le projet à terme.

Nous sommes cependant confiants quant au fait que ce que nous fournissons à la fin du projet en terme de schémas, de PCB et d'application web, ainsi que le retour que nous avons fourni sur les problèmes inhérents à la réalisation de cartes par l'école, sera utile et exploitable, dans une poursuite éventuelle du projet dans les années qui suivent.

Annexes

Annexe A - Manuel de l'application web

PFE - Conception d'un système domotique sans fil

Ceci est la documentation de l'application web du projet.

Developpé par :

- Thomas MAURICE thomas@maurice.fr
- Benoit MALIAR benoit@maliar.fr

Prérequis

Pour faire fonctionner cette application web, vous aurez besoin des éléments suivants :

- nodejs
- grunt
- bower
- npm
- postgresql

Les composants postgres, nodejs et npm s'installent via un simple `apt-get`, grunt et bower en revanche s'installent via `npm` comme il suit :

```
npm install -g grunt-cli bower
```

et voilà !

Configuration de Postgres

Vous devez créer un utilisateur de Postgres, loguez vous et entrez les commandes suivantes :

```
CREATE USER pifou;
ALTER USER pifou WITH ENCRYPTED PASSWORD 'glopglop';
```

Et enfin on crée la base de donnée !

```
CREATE DATABASE pifoudb OWNER pifou;
```

Ensuite on autorise l'utilisateur à se connecter à la base dans le fichier de configuration de Postgres :

```
host    all             all             127.0.0.1/32            md5
```

Et voilà !

Construire et déployer l'application

Construire l'application

Placez vous dans le dossier de l'application web, et lancez les commandes : npm install bower install

Afin d'installer les dépendances de nodejs via `npm` et les dépendances web (les css, libs js et autre) via `bower`. Puis lancez la construction de l'application via :

```
grunt
```

L'application est maintenant construite dans le dossier `build/app`. Vous pouvez la lancer avec un

```
grunt deploy
```

ou avec un deamonizer type `forever`.

Construire la base de donnée

Logguez vous en tant que l'utilisateur que vous avez précédemment créé sur postgres, et chargez le fichier create_database_prod.sql via la commande

```
\i create_database_prod.sql
```

Cela créera les tables nécessaire et deux utilisateurs :

- root avec le mot de passe `glopglop`, c'est l'administrateur
- pifou avec le mot de passe `pasglop` c'est un utilisateur normal

Vous pouvez maintenant utiliser l'appli :)

Configuration de l'appli

Si vous ouvrez le fichier de configuration conf/app.json, vous allez voir :

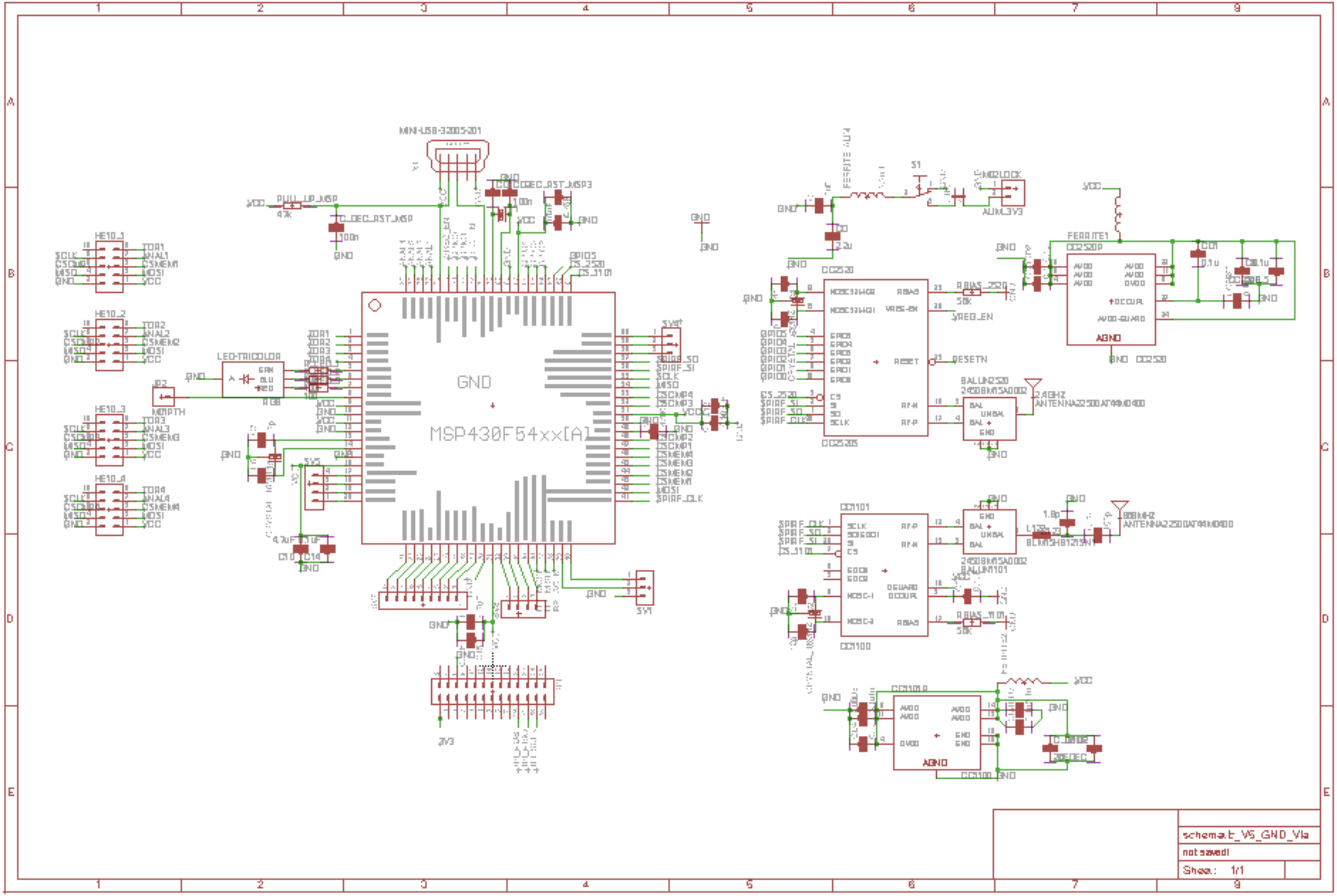
```
{
    "port": 8080,
    "dbcostring": "postgres://pifou:pasglop@localhost/pifou",
    "datarender": {
        "TMP": {
            "display_name": "Temperature",
            "presentation": "Temperature : %v C",
            "medium_presentation": "Medium temperature : ~%v C",
            "unit": "C"
        },
        "HUM": {
                "display_name": "Humidity",
                "presentation": "Humidity : %v%",
                "medium_presentation": "Medium humidity : ~%v%",
                "unit": "%"
        }
    },
    "needAuth": true
}
```

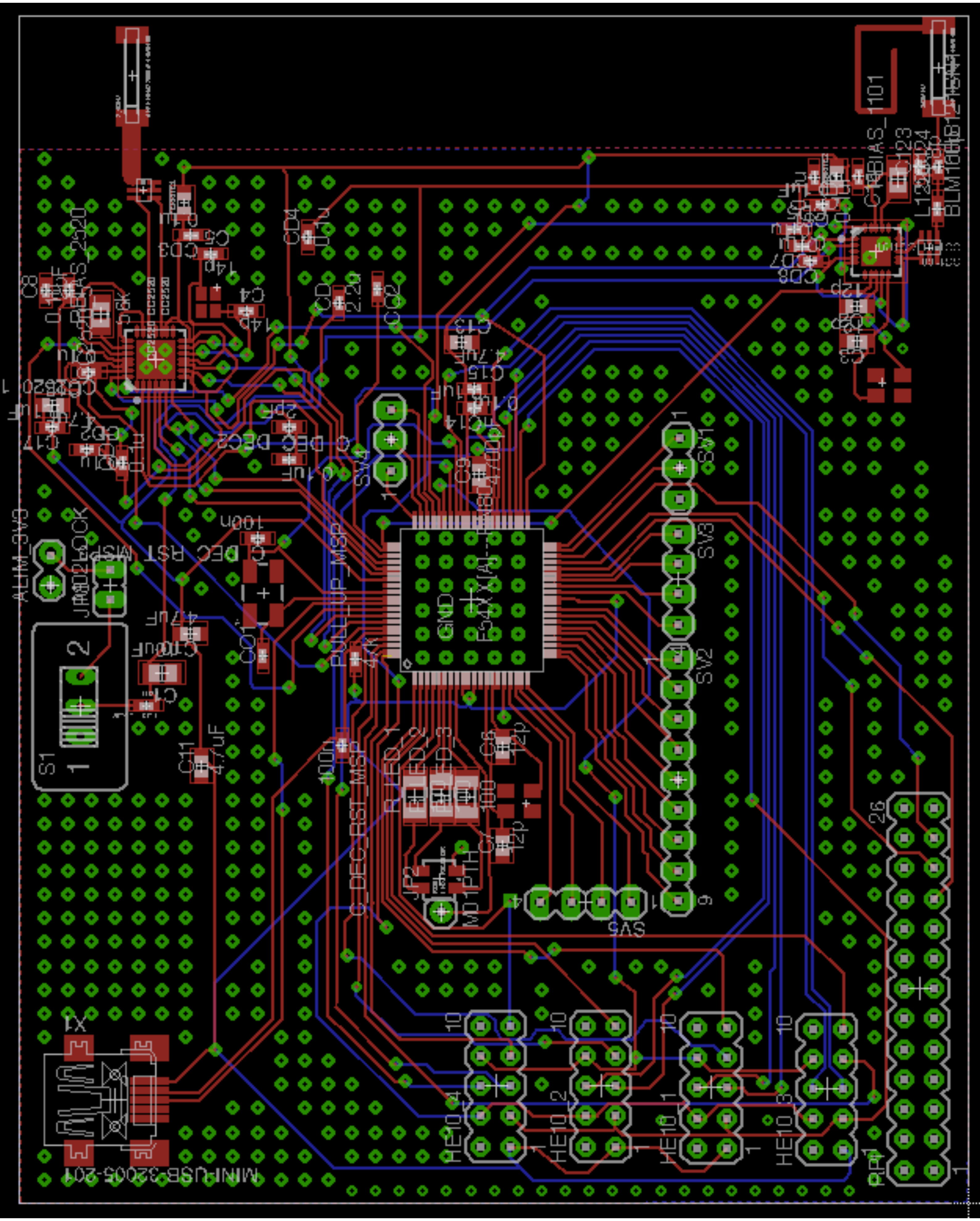
Les paramètres sont les suivants :

- `port` : le port d'écoute de l'appli, ne pas le changer
- `dbcostring` : chaine de connexion à la base de données, ne pas la changer non plus. Le format est `postgres://user:motdepass@host/database`
- `needAuth` : Protège ou non l'appli par mot de passe (ne pas changer)
- `datarender` : Définit comment représenter les données en fonction du trigramme, les exemples sont assez clairs il me semble. Cependant :
- `display_name` est le nom affiché tel qu'en haut d'un graphique
- `presentation` est la manière dont est affichée la valeur sous forme longue, le `%v` sera remplacé par la valeur.
- `medium_presentation` sera le rendu de la moyenne
- `unit` est l'unité employée

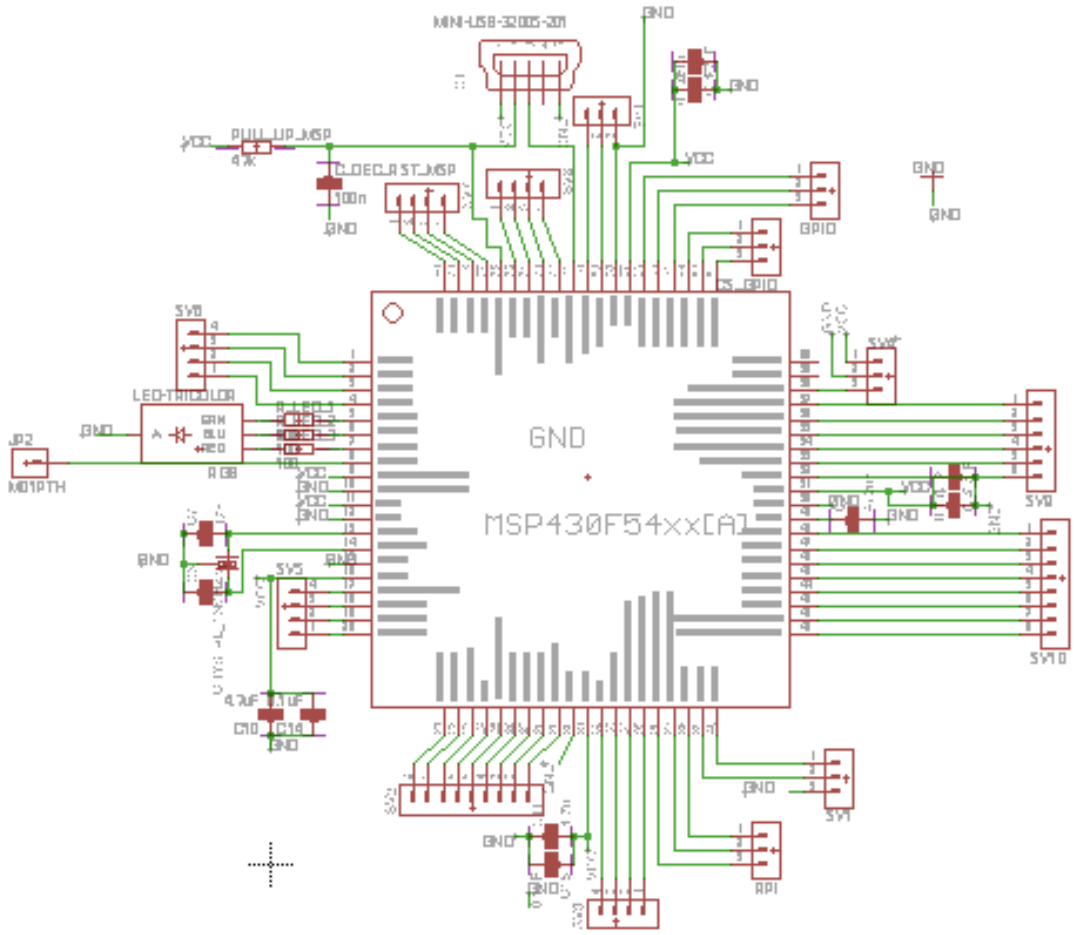
Le reste de la configurations se fait simplement depuis l'interface web.

Annexe B - Schematic et PCB de la carte v3

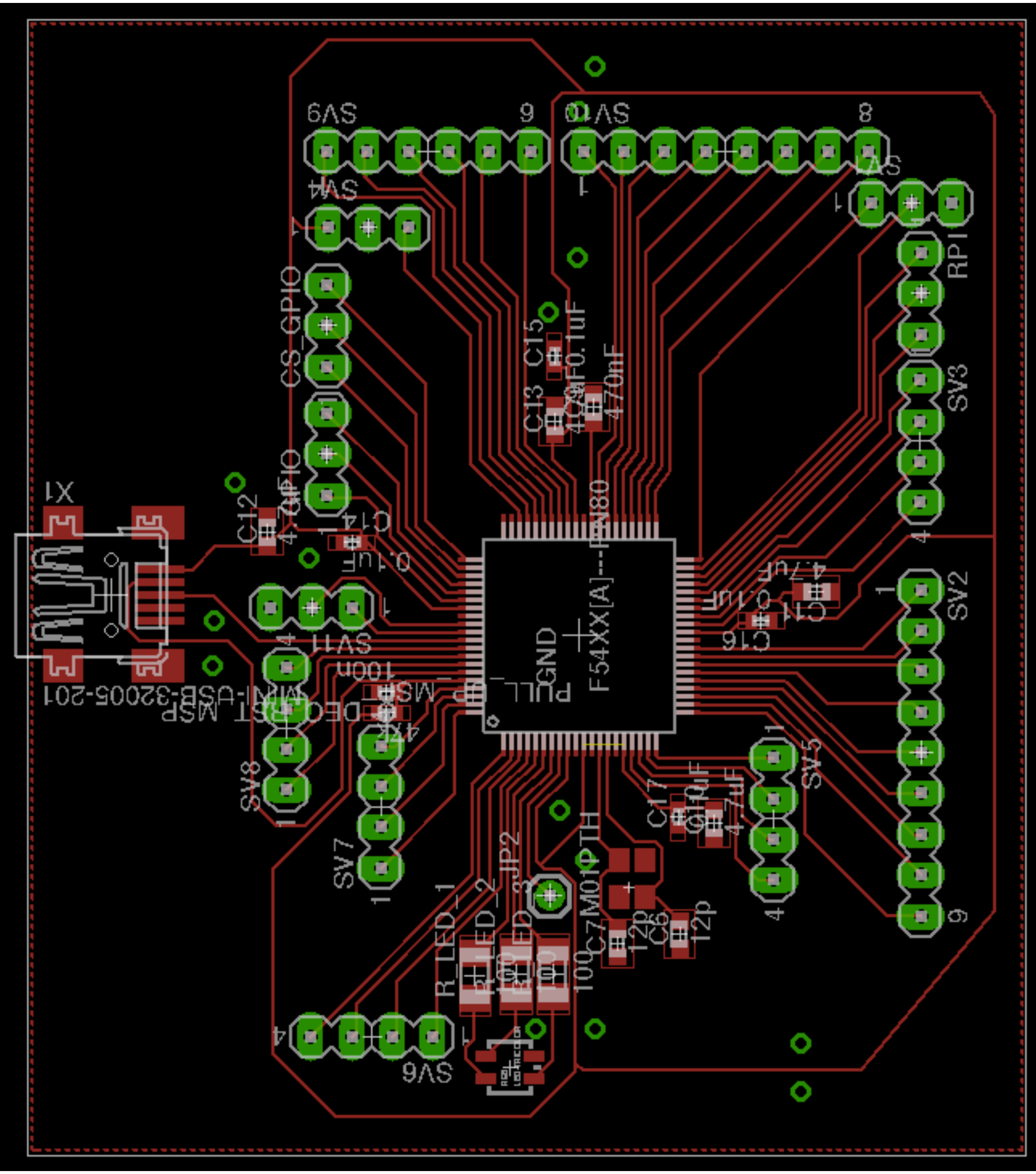

schematic_VS_GND_V1a
not saved!
Sheet: 1/1
MSP430F54xx[A]
CC2520
CC1101
CC2520P
FERRITE1
LED-TRICOLOR
MINI-USB-32005-201



Annexe C - Schematic et PCB de la carte mère

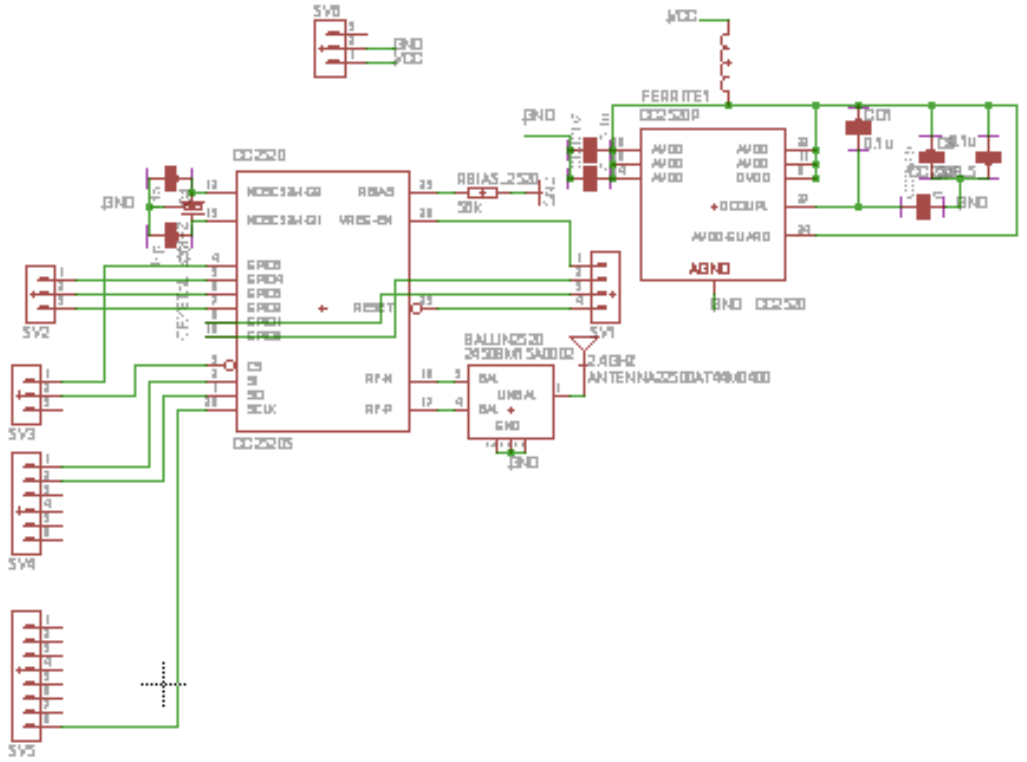


bmallari	
not saved!	
Sheet: 1/1	

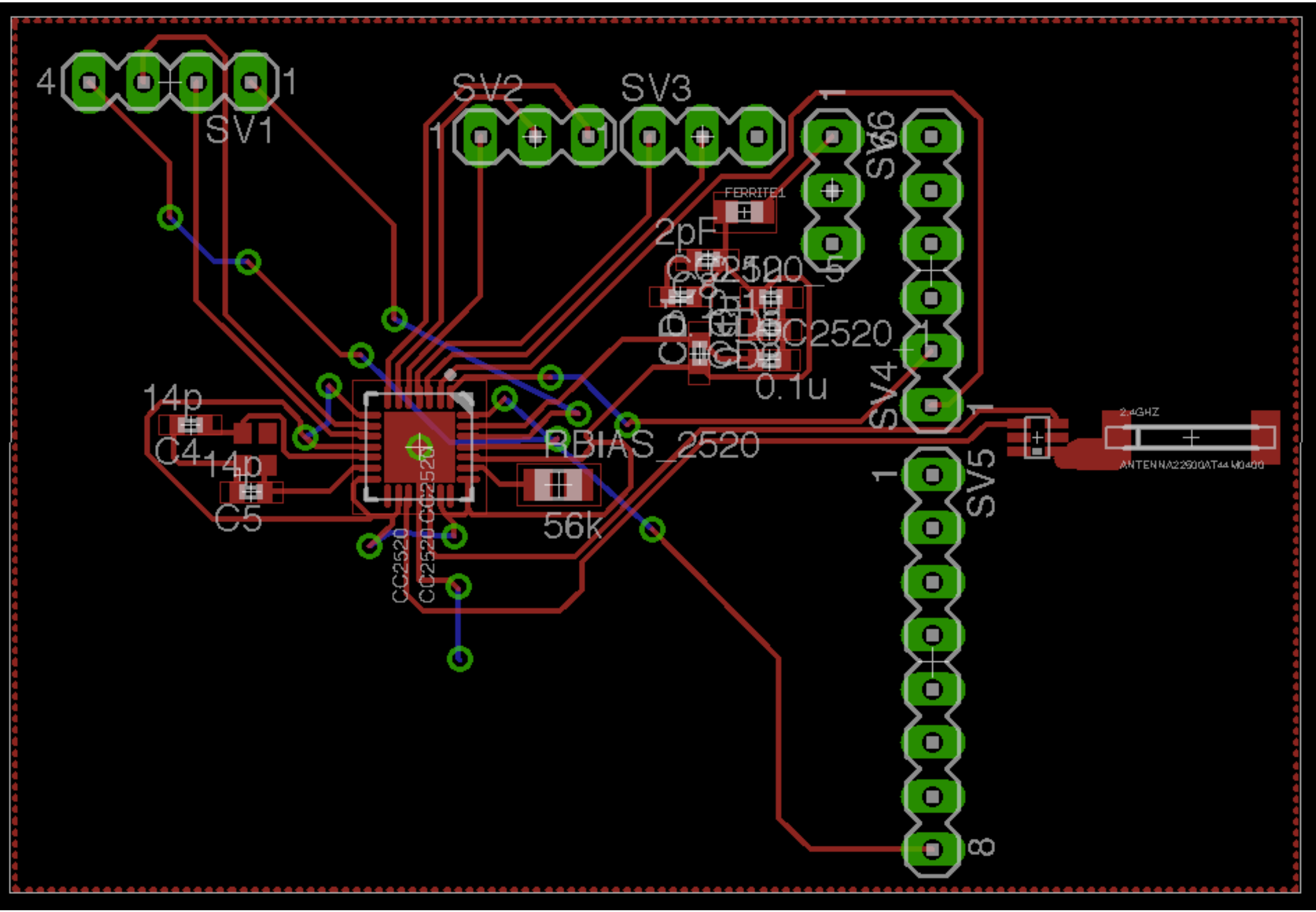


SV2
SV3
SV5
SV7
SV8
SV11
RPI
CS_GPIO
F54XX[A]---PN80
GND
PULL_UP_MSP
C12
C13
C15
C16
C17
C7
C6
JP2
R_LED_1
0.1uF
4.7uF
470nF
100n
12p

Annexe D - Schematic et PCB de la carte fille



bmalla:2	
12/02/2015 13:07	
Sheet: 1/1	


4
1
SV1
SV2
SV3
1
1
SV6
1
5
FERRITE1
2pF
C?
0.1u
SV4
1
14p
C4
14p
C5
RBIAS_2520
56k
1
SV5
8
2.4GHZ