Rapport de mi-projet de 5ème année

Informatique, Microélectronique, Automatique

Régulation temps réel sur réseau sans fil

Élèves : OBEISSART Morgan – ROBIC Vincent

Encadrants : Alexandre Boé – Thomas Vantroys

Année scolaire : 2016 – 2017

Sommaire

Introduction

Dans le cadre de notre cinquième et dernière année au sein du département Informatique, Microélectronique et Automatique de Polytech Lille, nous avons choisi de nous pencher sur l'un des sujets de projets proposés par des personnes intérieures à l'école. L'intitulé de ce sujet est « Régulation temps-réel sur réseau sans fil » : le but est d'implémenter un protocole de communication avec des notions temps-réel.

Nous avons réalisé ce projet en binôme, avec l'aide de nos encadrants, du personnel de Polytech Lille et de personnes extérieures. Nous souhaitons donc particulièrement remercier M. Boé, M. Vantroys pour leur travail et leur aide en tant que tuteurs de ce projet. Nous remercions M. Redon qui nous a notamment aidé pour la commande du matériel. Enfin, nous tenons à remercier Mme. Rolland pour son aide sur la réalisation de la carte électronique ainsi que M. Flamen, responsable du service électronique de Polytech Lille, pour ses connaissances et pour la gravure de cette carte électronique.

Dans un premier temps, nous commencerons par présenter plus en détail le projet que nous avons choisi. Puis, nous reviendrons sur le travail qui a été effectué jusqu'ici. Enfin, nous détaillerons le travail qu'il reste à faire avec le calendrier prévisionnel qui permettra de le réaliser.

I. Présentation du projet

1. Objectif du projet

L'objectif du projet est d'implémenter un protocole de communication avec des notions temps-réel. L'utilisation de réseaux sans fil pose un grand problème dû à la difficulté d'être sûr de la délivrance des données dans un temps borné. Notre travail consiste donc à nous intéresser à ce problème et à tenter de le résoudre.

2. Description du projet

Ce sujet est un projet très ouvert pour lequel aucun cahier des charges n'a encore été défini (ce sera d'ailleurs l'une de nos tâches). C'est la principale raison pour laquelle nous avons choisi ce sujet, car nous pensons qu'il nous apportera des connaissances multidisciplinaires et qu'il nous apprendra à gérer un projet. Parce qu'il est très ouvert, nous ne pouvons pas foncer tête baissée sur le codage de la solution. Il faudra au préalable faire un état de l'art des réseaux temps-réel sans fil. Nous choisirons notre propre application qui permettra « d'illustrer » la solution que nous aurons adoptée. Il faudra donc avancer méthodiquement, avec une part importante de gestion de projet.

Pour ce projet, il conviendra donc de :

- Réaliser des recherches bibliographiques sur les réseaux sans fils temps-réel,
- Choisir les moyens, matériel et logiciel, selon le résultat de ces recherches,
- Définir une application et le cahier des charges,
- Implémenter un protocole de communication qui servira cette application.

3. Choix techniques : matériel et logiciel

Les différents algorithmes seront développés sur cartes de développement STM32F4 discovery avec un lien radio de type Zigbee. Pour concevoir un réseau temps-réel, il nous faut d'abord choisir un système d'exploitation qui fournisse un support temps-réel. Nous avons donc choisi le système

d’exploitation Riot qui possède un tel support (ce choix s’est fait suite à nos recherches et aux réunions avec nos tuteurs, comme nous l’expliquerons dans la suite).

Pour le portage radio, nous avons choisi de concevoir une carte électronique autour de la puce Atmel AT86RF231 pour laquelle Riot propose un driver (là encore ce choix s’est fait après discussion avec nos tuteurs). La carte sera développée grâce au logiciel Eagle.

Jusqu’ici, nous n’avons pas encore parlé de l’application que nous avons choisie, c’est pourquoi nous n’exposons pas dans ce paragraphe le matériel qui est utilisé pour celle-ci. Ce point sera abordé dans la prochaine partie

Nous avons donc présenté l’objectif de notre projet et en avons fait une courte description. Nous connaissons désormais les choix techniques que nous avons mis en place. Revenons maintenant sur le travail qui a pu être effectué jusqu’à présent.

II. Travail accompli

Nous allons ici exposer le travail que nous avons effectué sur ce projet. Nous commencerons d'abord par revenir sur les différentes réunions avec nos encadrants qui ont permis de poser les bases, de nous orienter et de nous aider dans les choix techniques et/ou logicielles. Puis, nous détaillerons ce que nous avons réalisé au niveau de l'utilisation de la carte de développement STM32F4discovery avec, notamment, la prise en main du système d'exploitation Riot. Ensuite, nous présenterons la phase de conception de la carte électronique qui nous permettra de faire le lien radio entre les différentes cartes de développement (qui serviront en réalité de nœuds dans notre réseau).

1. Réunions avec nos encadrants

Les réunions avec nos encadrants ont évidemment été indispensables pour nous orienter dans la réalisation de notre projet. Nous avons pu organiser trois réunions lors de cette première phase de projet. Nous allons ici revenir sur les discussions que nous avons eues ainsi que sur les décisions qui ont pu être prises.

1.1. Première réunion : 05/10/2016

Nous n'avons pas choisi d'organiser cette première réunion dès l'affectation des projets. En effet, nous savions que des recherches étaient nécessaires en début de projet, dans le but de faire un état de l'art des réseaux temps-réel. Lors de cette première réunion donc, nous avons présenté le résultat de nos recherches, en sachant que celles-ci n'étaient pas très précises.

Côté technique, il nous faudra envisager deux modes de fonctionnement : un mode normal et un mode dégradé dans le cas, par exemple, de pertes de données ou de perturbations dans les communications. La question est donc : Comment doit réagir le système dans le mode dégradé ? Il faudra également réfléchir à une application qui illustrera notre projet. Ensuite, à partir des recherches réalisées, nous tenterons de définir plusieurs grandes problématiques qu'il serait intéressant de traiter, comme par exemple utiliser un réseau avec routage plutôt qu'un réseau linéaire. Nous avons ensuite discuté du système d'exploitation à utiliser : celui-ci doit fournir un support temps-réel. C'est pourquoi nous avons planifié des recherches sur le protocole de routage RPL ainsi que sur les systèmes d'exploitation Contiki et Riot. Enfin, nous avons la possibilité de

développer sous l'IDE de STMicroelectronics (chose que l'on ne fera finalement pas car cela ne simplifie pas significativement le travail).

Côté gestion de projet, cela sera un point très important de notre projet étant donné que ce dernier est très ouvert : il faudra être rigoureux dans l'avancement du travail et baliser ce dernier. La gestion sera donc très présente tout au long du travail. Nous décidons alors de mettre un place un modèle de compte-rendu des différentes réunions qui auront lieu. Le but de toute cette gestion est de savoir clairement le travail qui a été fait, de définir les étapes à venir et également de savoir à la fin du projet ce qui a été et ce qui n'a pas été. Il nous faudra également travailler en parallèle, et ne pas faire tous les deux la même tache comme nous avons pu le faire au début lors des recherches. Enfin, il faudra établir des objectifs pour la pré-soutenance afin d'avoir une première échéance.

Il a, pour finir, été décidé qu'une réunion aura lieu toutes les deux semaines, tout en sachant que cela peut varier avec l'avancement de notre travail.

1.2. Deuxième réunion : 19/10/2016

Lors de cette réunion, trois points ont été abordés : le retour sur les recherches réalisées sur Riot et Contiki, une discussion sur l'application choisie, et le passage en revue du calendrier prévisionnel.

Nous avons donc commencé par présenter nos recherches sur Contiki et Riot, ainsi que sur FreeRTOS sur lequel nous nous étions intéressés. Contiki ne fournit pas de support temps-réel, il ne pourra donc pas être utilisé. FreeRTOS, lui, est beaucoup trop minimaliste pour notre projet. Notre choix se porte donc logiquement sur Riot. Nous devons alors réaliser le portage radio, et deux choix s'offrent à nous : le premier est porté sur l'informatique : il consiste à adapter le driver d'une carte disponible à l'école. Alors que le second est porté sur l'électronique : il consiste à concevoir une carte dont Riot fournit déjà le driver. Après discussion, il nous a semblé plus judicieux de choisir la deuxième option, qui semble à priori plus rapide.

En ce qui concerne l'application, nous avions choisi la suivante : détection de présence (par exemple, allumer une lampe lorsqu'une personne entre dans une pièce). Cette application a été jugée trop simpliste pour pouvoir illustrer convenablement notre travail. Une alternative possible est de choisir une communication entre un donneur d'ordre et un robot comme application. Il nous faudra approfondir ce point.

Enfin, notre premier calendrier prévisionnel était trop linéaire, nous avions choisi d'enchaîner les tâches les unes après les autres mais cette méthode n'est pas très efficace. Les objectifs pour la pré-

soutenance sont les suivants : établir le cahier des charges de notre application, le calendrier prévisionnel de janvier/février et avoir tous les outils fonctionnels pour faire les réalisations.

1.3. Troisième réunion : 30/11/2016

Cette réunion est intervenue plus tard que prévu pour être en adéquation avec notre avancement. Le but ici était de faire une mise au point sur l'avancement du projet, de discuter de la réalisation de la carte pour le portage radio, et enfin de présenter le calendrier qui vaut jusqu'à la pré-soutenance.

Le projet avance normalement, même si certaines tâches qui nous paraissaient à priori simple se sont révélées être plus compliquées car il fallait prendre en main les outils. Nous avons également des difficultés sur l'utilisation du protocole de routage RPL (nous y reviendrons dans la suite). Nos encadrants nous ont alors donné un contact qui pourra nous aider sur ce point.

Concernant la carte pour le portage radio, la machine de gravure de l'école est tombée en panne, il nous faut donc attendre que cela soit réparé afin de procéder à la gravure. Le routage lui est prêt, et quelques dernières modifications ont été ajoutées afin de l'optimiser. Notre carte sera gravée en priorité dès que possible.

Enfin, notre calendrier qui va jusqu'à la date de la pré-soutenance est cohérent avec les objectifs qui nous nous étions fixés au début du projet même si nous avons quelques doutes sur la réalisation de la carte.

2. Prise en main de Riot OS et tests sur STM32F4discovery

Lors de la deuxième réunion, nous avons décidé d'utiliser le système d'exploitation Riot qui fournit un support temps-réel. Il nous a donc fallu prendre en main ce système, faire des tests sur notre ordinateur puis sur les cartes STM32F4discovery. Nous allons revenir ici sur les tests que nous avons pu réaliser.

2.1. Prise en main de Riot Os

Nous avons récupéré le code source de Riot depuis Github (code en open source). Pour compiler un projet Riot, il a d'abord fallu installer les chaines de compilation et de débogage ainsi que d'autres paquetages nécessaires. Avant de pouvoir passer aux tests sur la carte de développement, nous avons décidé de tester un des codes exemples proposés par Riot. Ce code, lorsqu'il est compilé et exécuté permet d'afficher dans la console un message de bienvenue : c'est une sorte de « Hello World ». Cet exemple très basique nous a permis de savoir compiler un projet Riot.

Dès lors, nous pouvions passer aux tests sur la carte STM32F4discovery.

2.2. Programmes de test utilisant Riot pour STM32F4discovery

Là encore, le premier test que nous avons réalisé était d'allumer une LED de la carte. Pour réaliser ce test simple, il a fallu chercher dans Riot les fonctions qui permettent de gérer, entre autres, ces LED. L'avantage d'utiliser un système d'exploitation est de nous faciliter la tâche car des fonctions sont déjà proposées et permettent de faire beaucoup de choses. L'inconvénient est qu'il faut connaître ces fonctions, qu'il faut les trouver, savoir où chercher, et cela représente au début beaucoup de temps.

Une fois ce test passé, nous avons décidé d'inclure un timer dans l'exemple. Le but est de faire clignoter une LED toutes les secondes. Nous ne savons pas si nous aurons réellement besoin des timers dans la suite du projet, mais cela nous semblait être un bon test pour maîtriser un peu plus le système d'exploitation et la carte. Nous avons eu quelques difficultés à trouver les bonnes fonctions et à comprendre leur bon fonctionnement, mais nous sommes finalement parvenus à réaliser cette fonctionnalité.

Enfin, nous avons tenté de piloter les servomoteurs depuis la carte de développement. Les servomoteurs dont nous disposons sont des servomoteurs à rotation continue. Nous pouvons donc contrôler la vitesse de rotation grâce à un signal PWM. Là encore, Riot propose des fonctions qui permettent de créer ce signal. Une fois ces fonctions trouvées et comprises, nous avons pu contrôler la vitesse de rotation de nos servomoteurs.

Nous avons ainsi réalisé plusieurs tests qui nous ont permis de prendre en main la carte STM32F4discovery sous Riot. Nous savons contrôler nos servomoteurs, et cela sera indispensable pour le bon fonctionnement de notre application (celle-ci sera donnée dans la prochaine partie). Dès lors, nous pouvions nous pencher sur la partie communication entre deux cartes STM32F4discovery.

2.3. Communication entre deux cartes STM32F4discovery

A ce stade, notre carte électronique pour le portage radio n'était pas encore réalisée. Pour ne pas perdre de temps, nous avons décidé de faire un premier test sur notre machine Linux, en « simulant » les nœuds de notre réseau grâce à des interfaces virtuelles connectés entre elles grâce à un bridge. Le but de ce test est de mettre en place une architecture réseau utilisant RPL constituée d'un père et de deux fils. Pour réaliser cette architecture, nous nous sommes appuyés sur un exemple proposé par Riot. Nous parvenons à afficher la table de routage où l'on a bien un parent et deux fils. Lorsqu'on éteint le parent, nous ne pouvons plus communiquer entre les deux fils. Ensuite, nous avons tenté de modifier cette architecture pour avoir un réseau linéaire (soit un père, un fils et un petit-fils). Malheureusement, nous avons rencontré quelques difficultés à faire cela. En effet, la table de routage d'origine semble se remettre à jour périodiquement, ce qui fait que nos changements ne sont pas pris en compte : ce point a d'ailleurs été discuté lors de la troisième réunion comme nous avons déjà pu le dire.

Suite à cela, toujours en attendant que la carte pour le portage radio soit réalisée, nous avons décidé de tester une communication entre deux cartes STM32F4discovery grâce aux UARTs qu'intègrent ces dernières. Le but est le suivant : lorsqu'on appuie sur le User Button de la première carte, celle-ci transmet un caractère à la seconde carte qui génère ainsi un signal PWM afin de faire tourner le servomoteur à une certaine vitesse. Si on relâche ce bouton, un autre caractère est envoyé et cela a pour but de stopper le servomoteur. Nous sommes parvenus à réaliser cette application.

Nous avons donc résumé l'ensemble des tests que nous avons pu effectuer. De manière générale, la plupart d'entre eux nous paraissaient simple à priori, mais ils se sont révélés plus compliqués que prévu car il a fallu faire des recherches sur Riot, sachant qu'il n'y a pas beaucoup de support sur ce système assez récent.

Nous allons maintenant passer à une autre partie très importante pour notre projet : la réalisation de la carte électronique pour le portage radio.

3. Conception de la carte pour le portage radio

Afin de permettre la liaison sans fils de nos nœuds, nous sommes amenés à utiliser des modules de communication radio. Maintenant que nous avons déterminé le système d'exploitation (Riot-OS) que nous souhaitions intégrer sur notre microcontrôleur, il nous a été conseillé de choisir un module compatible (c'est-à-dire que le driver de la puce gérant la radiofréquence doit y être intégré). Lors de la réunion, nous avons finalement choisi l'AT86RF231 car il présentait, dans sa datasheet, un schematic simple et une BOM pour la réalisation du module radio.

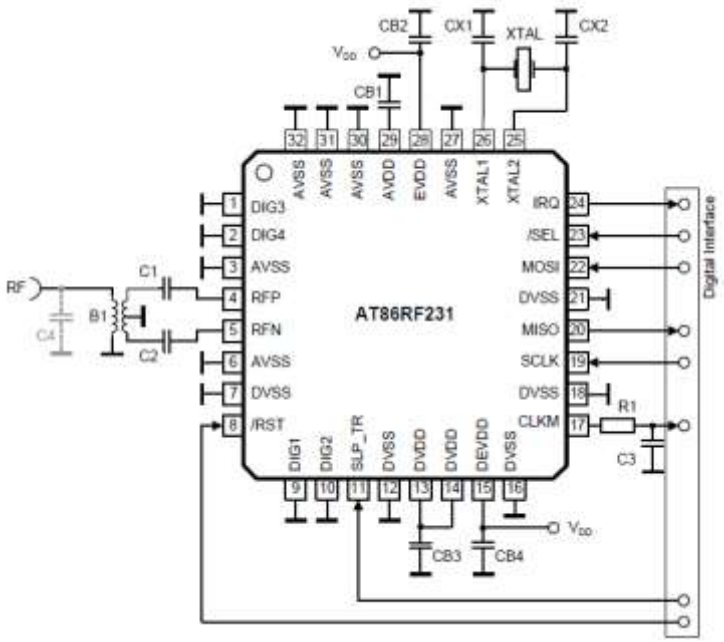


Fig. 1 : Schéma de réalisation du module radio présent sur la datasheet de l'AT86RF231

3.1. Réalisation du schematic sous Eagle

Pour la réalisation de cette carte, nous avons commencé par dessiner, sur Eagle, les différents packages nécessaires. Une fois cela réalisé, nous avons constaté une erreur dans notre commande. Il s'agissait de l'oscillateur à 16 Mhz qui ne correspondait pas et dont l'utilisation était différente. C'est une erreur connue et que l'on a pu corriger. Malgré cela, le schematic a pu être réalisé en intégrant le package de l'oscillateur commandé par la suite sur Conrad. Ensuite, concernant les entrées/sorties numériques de la carte, nous avons respecté le schéma ci-dessus et avons ajouté une entrée d'alimentation 3,3V et la masse.

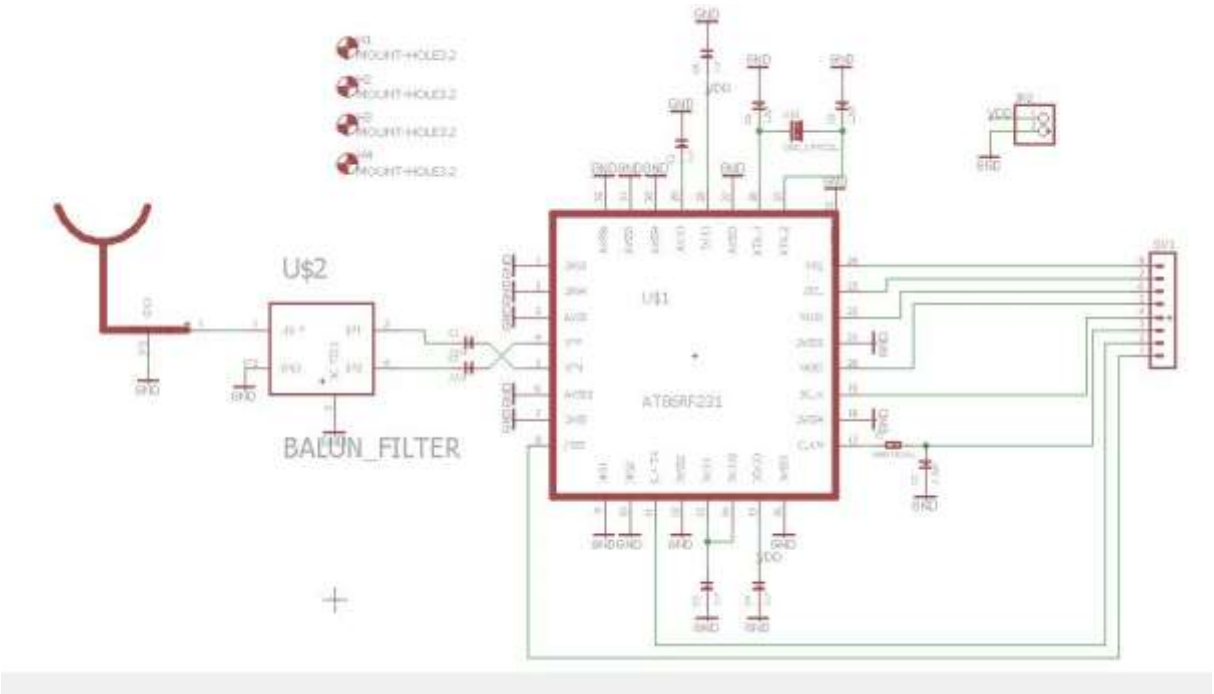


Fig.2 : Schématic du module radio réalisé sur Eagle

3.2. Réalisation du routage

La réalisation du routage est la partie la plus technique pour cette carte et plus particulièrement la transmission haute fréquence. La bande utilisée pour les liaisons Zigbee est à 2,45 GHz ce qui correspond aux hyperfréquences. C'est pourquoi le bloc ci-dessous, constitué d'une antenne, d'un balun et de deux condensateurs, doit être étudié et dimensionné.

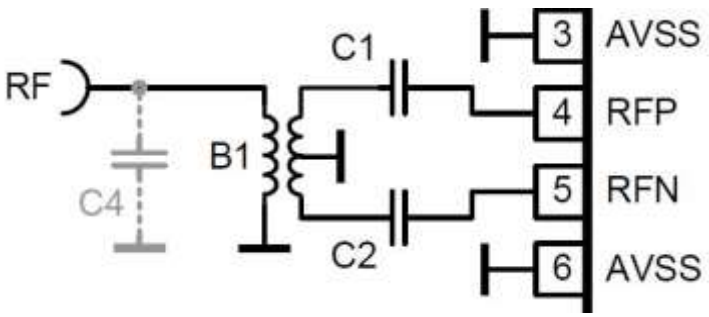


Fig.3 : Bloc à hautes fréquences du module radio

Pour cela rappelons quelques éléments de bases nécessaires à notre étude :

- L'impédance d'entrée Ze d'un quadripôle est l'impédance équivalente qui, mise au borne d'un générateur, donne la même intensité et la même tension.

- L'impédance de sortie Zs d'un quadripôle est l'impédance en série d'un générateur vu par la une résistance Ru en sortie.

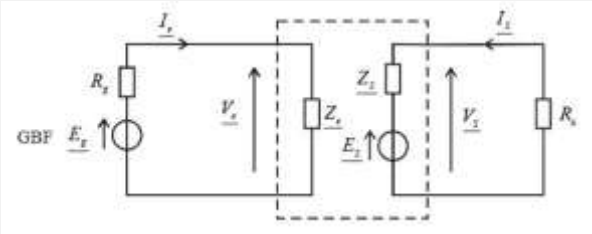


Fig.4 : schéma représentant un impédance d'entrée et de sortie

- « Le principe fondamental de l'adaptation d'impédance est le suivant : en connectant sur une charge de résistance R, une ligne de transmission d'impédance caractéristique R, on retrouvera à l'autre extrémité de la ligne la même résistance R. Autrement dit, la source et la charge de résistance R seront « adaptées » si la ligne qui les relie possède une impédance caractéristique de même valeur. L'adaptation sera conservée quelle que soit la longueur de la ligne. »
(https://fr.wikipedia.org/wiki/Adaptation_d'imp%C3%A9dances)

- L'impédance caractéristique est la résistance vue par le générateur aux premiers instants de la transmission. Elle dépend uniquement des caractéristiques de la ligne.

- Le coefficient de réflexion est un nombre sans dimensions qui indique la quantité d'énergie réfléchie en bout ou en début de ligne. Il est compris entre -1 et +1 et égal à 0 si la ligne est adaptée.

- Les feeders sont les lignes qui alimentent les antennes. Ils doivent véhiculer l'énergie jusqu'à l'antenne avec un minimum d'onde réfléchie. Ils doivent donc avoir comme valeur d'impédance caractéristique la valeur de l'impédance de l'antenne.

L'antenne choisie est une antenne CMS qui sera utilisée pour l'émission et la réception. Selon la datasheet, son impédance est de 50 Ohms et il en est de même pour l'impédance de sortie du balun. Pour adapter ces deux éléments, il nous faut une ligne d'impédance caractéristique 50 Ohms. Pour une ligne microstrip, l'impédance caractéristique dépend de ses dimensions et du matériau isolant.

De nombreuse formules sont établies dans la littérature pour calculer cette impédance, nous avons choisi le modèle présent dans logiciel AppCAD.

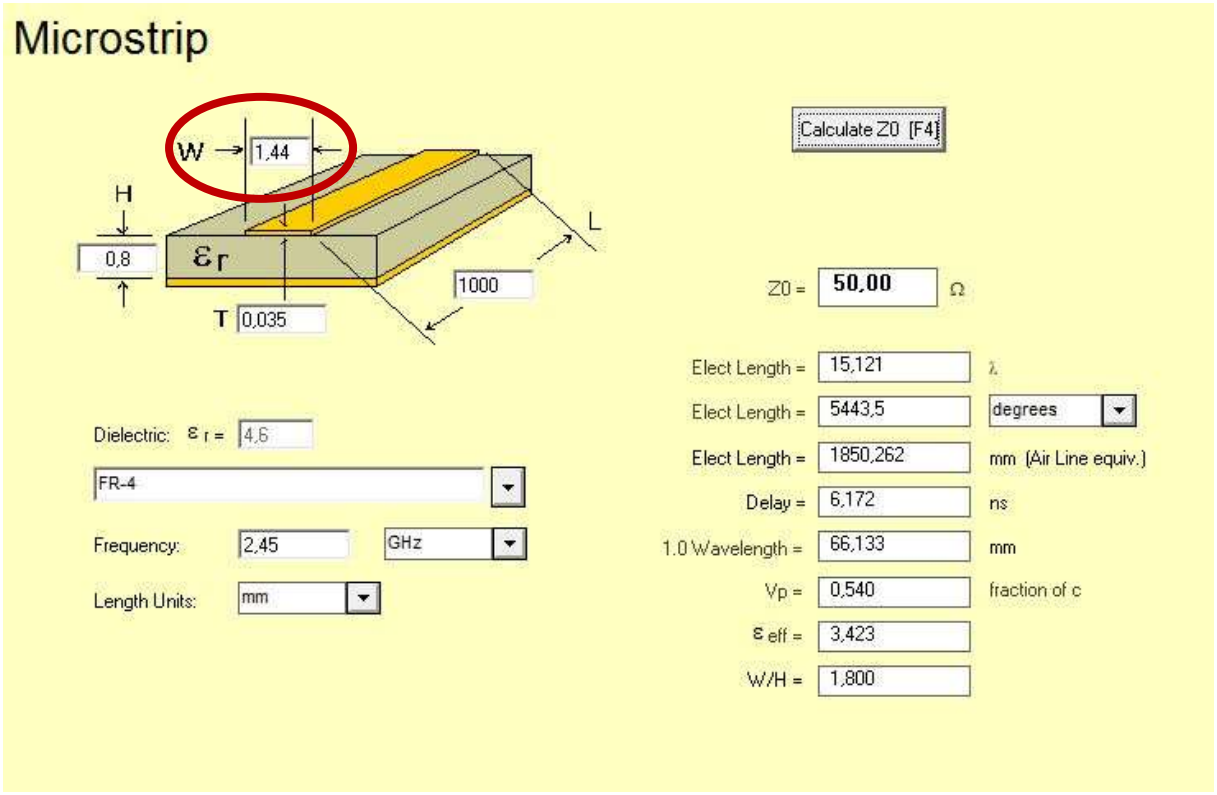


Fig.5 : Résultat du calcul d'impédance caractéristique

Pour réaliser cette opération, il nous faut donc l'épaisseur t de la piste, l'épaisseur h du diélectrique ainsi que sa permittivité. A Polytech, il est possible de réaliser des cartes avec des pistes d'épaisseur 35 um avec un diélectrique verre-epoxy FR4 d'épaisseur 0,8 mm. En utilisant ces informations, nous trouvons une largeur de piste égale à **1,44 mm**.

Le rôle du balun est de transformer une impédance symétrique en une impédance asymétrique et vice versa. Le balun réalise également une fonction d'adaptation d'impédance entre les ports RFP et RFN et l'antenne. Pour ne pas trop modifier l'impédance différentielle, nous avons décidé de réaliser des pistes de longueur faible, assez large mais surtout des lignes symétriques. Enfin les condensateurs à l'entrées des ports RFP/RFN sont utilisés pour supprimer la composante continue de l'entrée RF provenant de l'antenne.

Afin d'améliorer le transport de l'énergie jusqu'à l'antenne, il est conseillé de rajouter de nombreux vias pour que les plans de masse soient au même potentiel. De plus, l'ajout de vias permet d'atténuer

l'effet capacitif des plans de masse situés de part et d'autres du diélectrique. Une première demande de réalisation de la carte a été faite le 17/11/2016. Cependant, à ce moment-là, la machine chargée de la gravure est tombée en panne entrainant un retard.

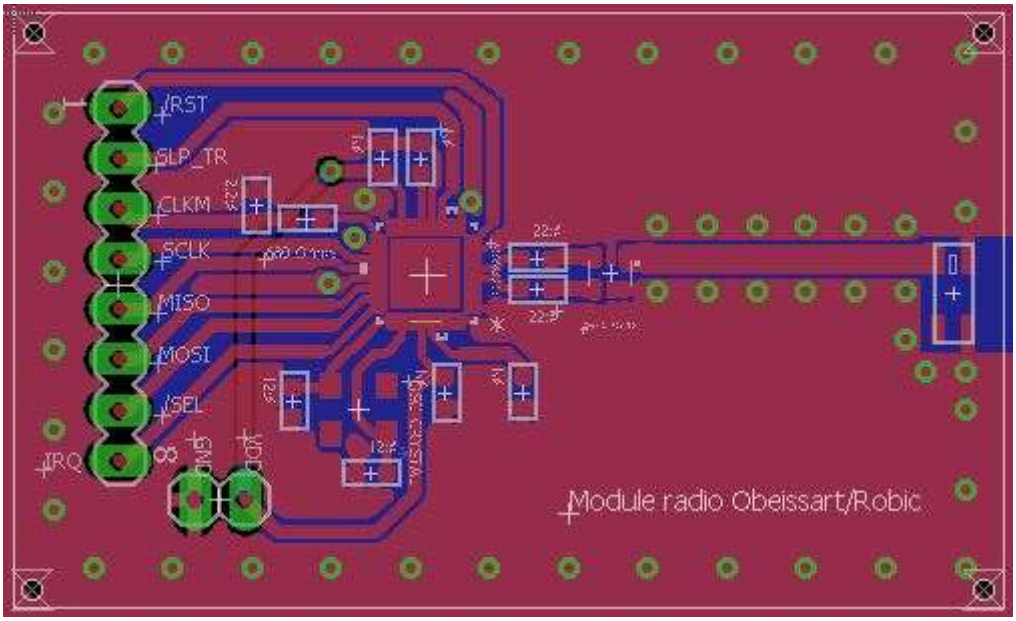


Fig.6 : premier routage du PCB du module Radio

Ce qui nous a permis lors d'une nouvelle réunion, avec notre tuteur Mr Boe, d'améliorer le routage de cette carte. Comme le microcontrôleur est cadencé à 16 MHz pour fonctionner, il faut lui associer un oscillateur à quartz. Pour assurer son bon fonctionnement il est alors nécessaire de l'isoler du plan de masse en TOP afin d'éviter que l'on fonctionne en mode coplanaire. Nous avons également ajouté des trous afin de permettre le vissage des cartes.

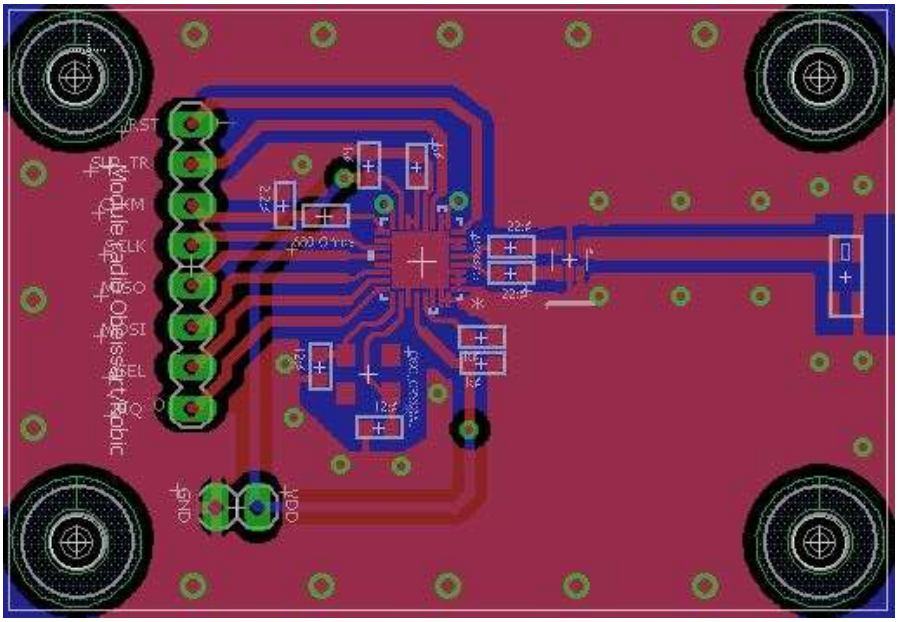


Fig.7 : dernier routage du PCB du module radio

4. Le cahier des charges

La définition d'un cahier des charges était l'un des objectifs à remplir pour la pré-soutenance. Nous allons ici le détailler.

Notre objectif est de contrôler les servomoteurs d'un robot via un réseau sans fil temps-réel. Le robot sera réalisé par nos soins. La liaison sans fil sera dans notre cas une liaison radio de type ZigBee. Chaque carte STM32F4discovery, ainsi que le module radio qui y sera associé, constitueront les nœuds de notre réseau. Le robot sera le dernier de ces nœuds.

Trois modes de fonctionnement sont à prévoir :

- Mode normal : aucune perturbation n'est observée dans le réseau, tout se passe normalement. Le robot avance selon les ordres du donneur d'ordres.
- Mode dégradé : dans ce mode, le robot avancera avec à vitesse réduite dans la dernière direction que lui a communiqué le donneur d'ordres. Le robot peut rester dans ce mode pendant une durée maximale de 30 secondes.
- Mode d'arrêt d'urgence : dans ce mode, le robot s'arrête et attend de recevoir les prochains ordres.

Ainsi, lorsqu'une perturbation a lieu (ligne coupée, parasites sur la ligne, etc) et que la transmission de l'ordre dépasse le délai imparti (délai que nous donnerons dans la suite) ou que le robot ne reçoit plus d'ordres, le robot passe en mode dégradé. Si, pendant la durée du mode dégradé, la transmission est revenue à la normale, le robot repasse en mode normal. On considère que la transmission est revenue à la normale lorsque cinq ordres consécutifs ont pu être transmis dans le délai imparti. En revanche, si la transmission rencontre toujours des problèmes après les 30 secondes du mode dégradé, alors le robot passe en mode d'arrêt d'urgence et attend ensuite de recevoir 5 ordres qui respectent la deadline avant de repartir.

Notre application devra respecter quelques contraintes supplémentaires.

- Un OS fournissant un support temps-réel devra être utilisé
- Il faudra adapter un protocole de routage adapté aux réseaux de capteurs
- Le robot possèdera deux servomoteurs
- Un ordre devra être transmis dans une durée maximale de 500ms, sous peine de voir le système se dégrader comme expliqué ci-dessus

Maintenant que notre cahier des charges est en place, voyons le travail que nous devrons accomplir.

III. Ce qu'il reste à faire

De nombreuses tâches restent à réaliser dans la suite de notre projet. Pour le moment, nous avons mis en place les « briques » de base qui nous permettront de faire nos tests. En effet, nous avons réalisé les cartes pour la communication sans fil (même s'il faut encore souder les composants), pris en main l'environnement de travail, avec Riot notamment. De plus, notre cahier des charges est défini et nous savons donc quelles tâches restent à accomplir.

Il nous faudra tout d'abord souder et tester les derniers modules radios pour finaliser notre réseau. Dans le même temps, nous pourrons commencer à nous pencher sur la communication sans fil entre deux nœuds. Ensuite, nous nous intéresserons au côté temps-réel de notre réseau : nous pourrons par exemple commencer par mettre en place un programme de test qui permettra de connaître la durée que met un ordre à arriver au robot. Pour cela, il sera nécessaire de regarder ce qu'offre le système d'exploitation Riot dans ce domaine. Dans la suite, à chaque nouveau changement dans notre réseau, nous tâcherons d'ajouter des contraintes temps-réel : le but est d'introduire ces problématiques petits à petit afin d'éviter de trop grosses difficultés à la fin de notre projet.

Nous réaliserons le robot entre deux tâches. Cela devrait être très rapide, le but n'est pas d'avoir un robot très esthétique mais simplement d'avoir une petite maquette qui puisse se déplacer sans problème.

Lorsque la communication entre deux nœuds sera maitrisée, nous ajouterons un troisième nœud et nous utiliserons alors un routage statique. C'est ici que nous commencerons à utiliser le protocole de routage RPL.

Dès lors, nous pourrons programmer les différents modes du robot : mode normal, mode dégradé et mode d'arrêt d'urgence. Il faudra ici simuler des perturbations afin de constater si notre robot adopte le comportement souhaité.

Ensuite, nous pourrons ajouter un quatrième nœud et nous passerons dans le même temps à un routage dynamique : le but est de tenter de contourner les perturbations afin de respecter les délais et de ne pas entrer dans le mode dégradé.

Nous avons donc mis en place un calendrier prévisionnel jusqu'à la fin de la durée de ce projet afin de baliser notre parcours et de savoir si nous sommes dans les temps. Evidemment, ce calendrier

n’est pas figé et sera sûrement amené à bouger selon les obstacles rencontrés. Pour des raisons de lisibilité, nous ne mettrons pas ici le calendrier prévisionnel, mais ce dernier sera disponible sur la page de notre wiki.

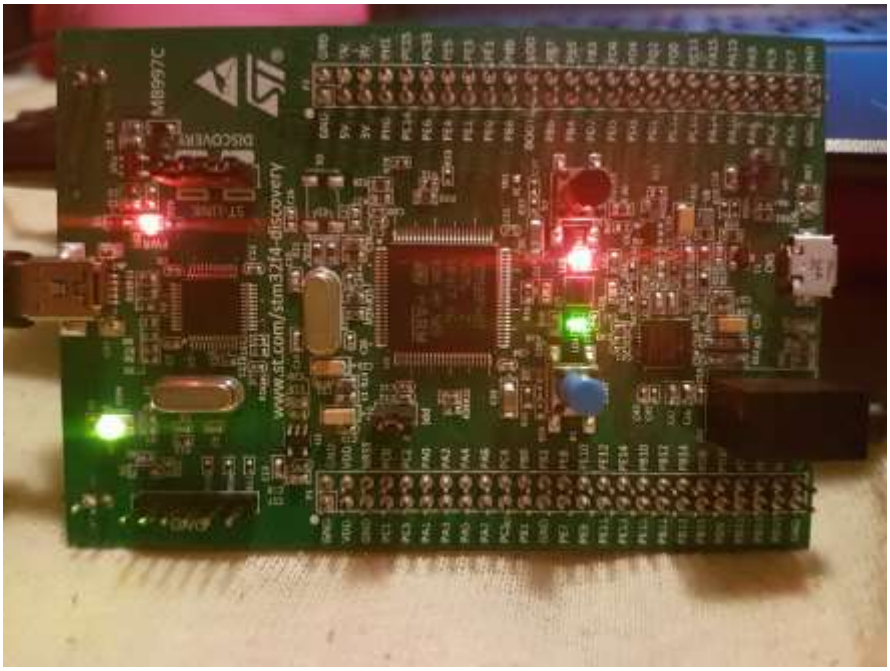
Concernant la gestion du projet, nous adopterons la même méthode que jusqu’à présent, à savoir une réunion régulière (puisque nous aurons beaucoup de temps à consacrer au projet, nous ferons une réunion hebdomadaire) et un compte-rendu détaillé de chaque réunion qui sera mis en ligne et consultable par tous les intéressés.

Conclusion

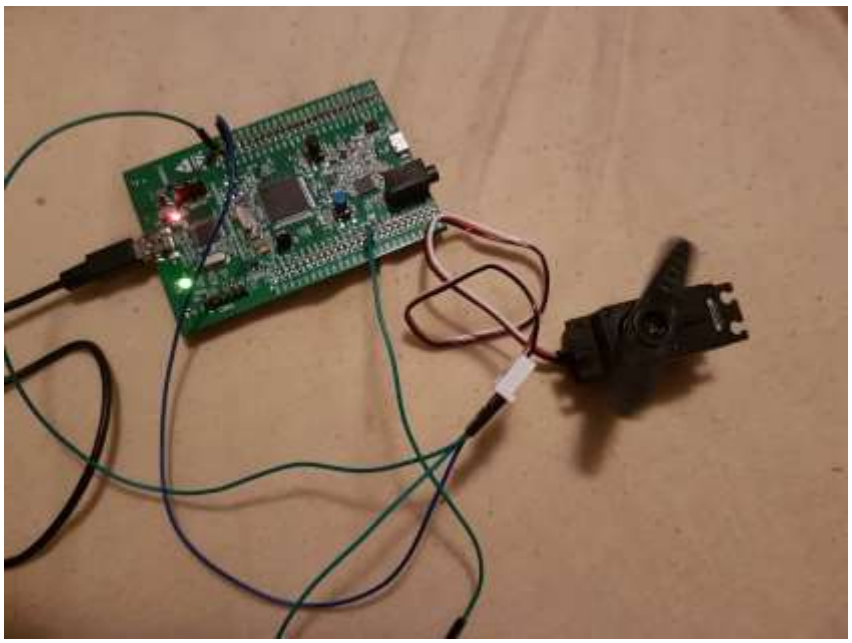
Au terme de cette première partie du projet, nous avons respecté les objectifs fixés en début de projet, à savoir la définition d'un cahier des charges, d'un calendrier prévisionnel et la mise en place de tous les éléments de base. La deuxième moitié du projet sera plus intensive et nous permettra de nous consacrer plus pleinement à celui-ci.

Jusqu'ici, nous avons vu beaucoup d'aspects différents dans ce projet. En effet, nous n'avons pas eu de cahier des charges précis, il a donc fallu faire des recherches, mettre en place une méthode rigoureuse pour gérer le projet. Nous avons eu le côté électronique, avec la conception du module radio notamment, et le côté informatique avec l'aspect réseau. Nous avons donc un aperçu de ce qu'est un véritable projet comme nous pourrons le rencontrer dans notre avenir professionnel. C'est pourquoi nous sommes satisfaits et pensons que la deuxième partie sera vraiment intéressante.

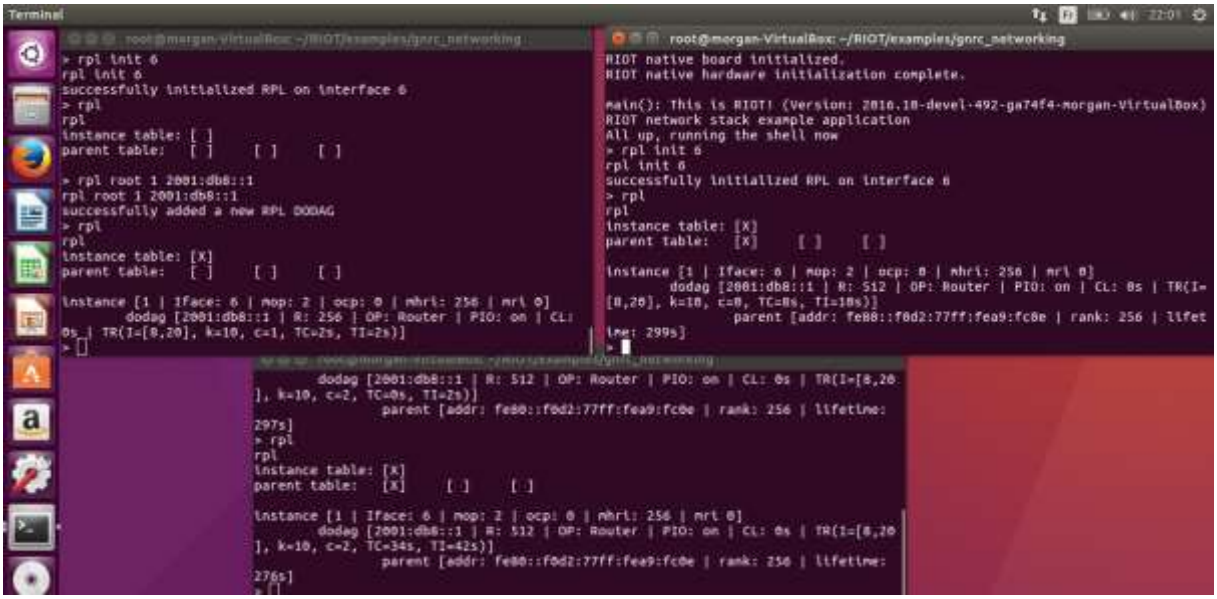
Annexe



Annexe 1 : Clignotement des LEDs en utilisant un timer

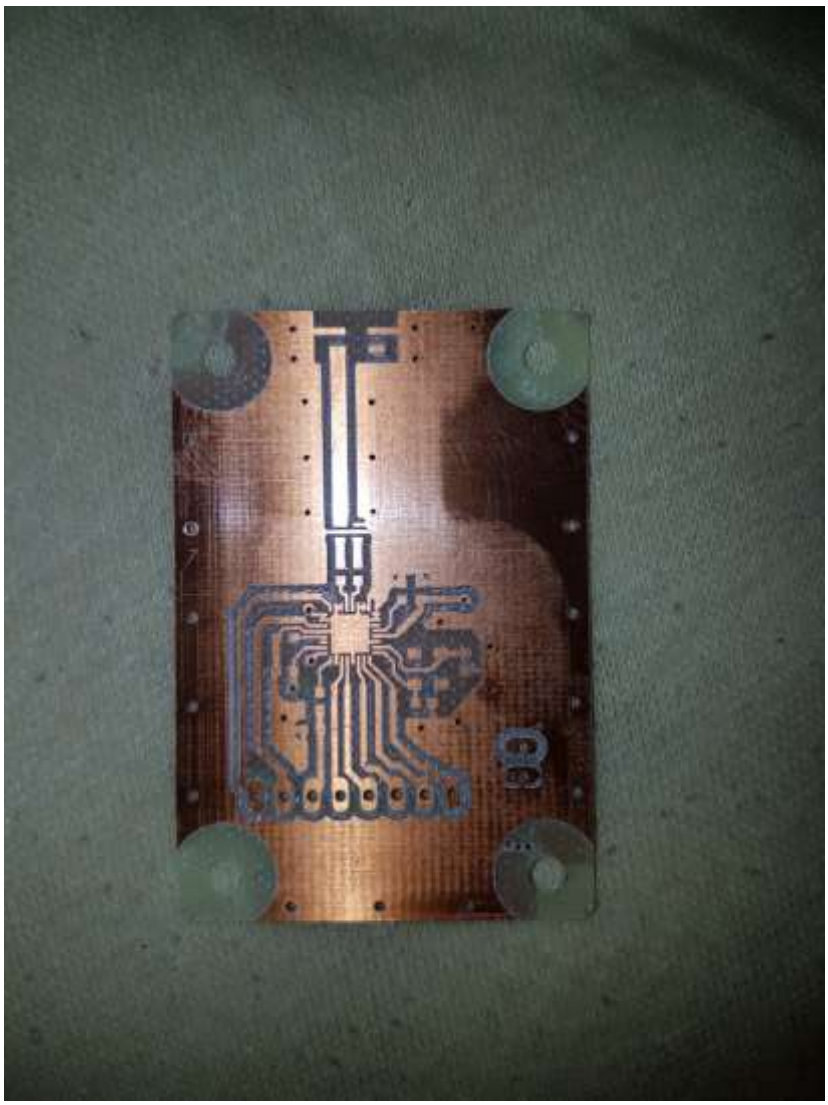


Annexe 2 : Pilotage d'un servomoteur depuis la carte STM32F4discovery



Annexe 3 : Exemple d'architecture de réseau utilisant RPL

Annexe 4 : Communication par UART pour piloter un servomoteur



Annexe 5 : PCB du module radio (composants non-soudés)

Annexe 6 : Calendrier prévisionnel