POLYTECH LILLE

Rapport de Projet IMA4
P23 - Table de bar connectée

Delobelle Matthieu

16 mai 2018

Table des matières

Introduction

Ce projet a été réalisé dans le cadre de la quatrième année de formation Informatique Microéléctronique et Automatique à Polytech Lille. Durant le semestre 8 de cette formation, beaucoup de temps a été consacré à la réalisation d'un projet. Mon travail reprend les travaux réalisés par Kevin COLAUTTI et Benjamin LEFFORT lors de leur projet de fin d'étude en 2016.

Leur projet consiste en la conception d'une table de bar connectée. L'objectif de mon travail consiste alors à reprendre cette réalisation et à la développer afin de lui apporter de nouvelles fonctionnalités.

J'expliquerai tout d'abord le contexte du projet en présentant les objectifs et en analysant l'existant. Je développerai ensuite les différentes parties de la réalisation du projet. Finalement je conclurai en envisageant les ouvertures de ce projet, tout en questionnant sa sécurité.

1 Contexte du projet

1.1 Objectifs initiaux et origine du projet

La table de bar présente en E306 était inutilisée depuis sa réalisation en 2016 par les étudiants en IMA5. L'objectif initial du projet était donc de dépoussiérer ce meuble connecté afin de lui offrir de nouvelles caractéristiques et de pouvoir le présenter à différents évènements internes à l'école.

Souhaitant originellement travailler sur un projet axé domotique, différentes discussions avec M. Thomas VANTROYS et M. Alexandre BOE m'ont conduites à choisir ce projet. En effet, je souhaitais initialement réaliser une maquette de maison connectée complète. Cependant mes tuteurs de projet m'ont convaincu de réaliser cette application domotique à échelle humaine, ceci en utilisant la table connectée en guise de centrale domotique.

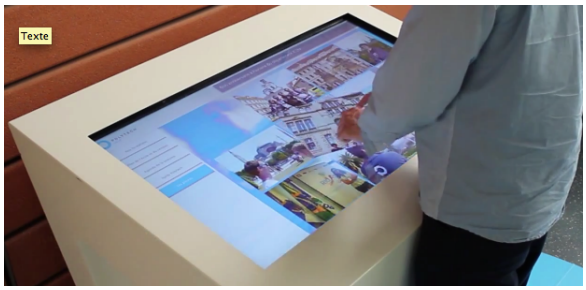
Avant de commencer le développement du projet, il fut nécessaire de réaliser un état de l'art du projet.

1.2 Analyse de l'existant

En observant la table de bar connectée, on se rend compte rapidement qu'il ne s'agit en réalité que d'un ordinateur équipé d'un écran tactile. Un simple changement de connectique entre l'unité centrale et le moniteur à permis de redémarrer la machine et de re-faire fonctionner le projet réalisé par mes prédécesseurs.

En lisant leur rapport et leur wiki, on apprend que leur application utilise une interface WEB pour fonctionner. L'avantage d'une telle interface est qu'elle est assez simple à mettre en place. Toute leur architecture informatique fonctionne encore parfaitement. Cela me permet de partir sur une base stable du coté de la programmation.

1.3 Étude des concurrents



Le marché du meuble connecté n'est pas nouveau. De nombreux concurrents se disputent le développement de nouvelles technologies de plus en plus innovantes. De plus ce genre de meuble devient davantage présent dans les lieux publics ou dans certains établissements à des fins commerciales.

Il est d'ailleurs possible d'en observer un dans le hall de Polytech Lille en face du Fabricarium. Digilor et Dymension sont deux entreprises vendeuses de tables connectées sous différentes formes.

1.3.1 Digilor

Digilor dispose d'un catalogue de tables et de bornes interactives tactiles assez génériques. Ces dernières fonctionnent en général avec un environnement Windows 8/10 et un écran capacitif projeté. Le principal marché de référence de Digilor est le monde de l'entreprise. En effet, leurs produits sont plutôt destinés a des fins commerciales et/ou publicitaires. On pense notamment au magasin de vêtements qui pourrait avoir une borne pour présenter leur catalogue en ligne ou un hôtel, afin que la salle d'accueil soit plus animée. Cela permettrait également de donner un côté moderne et high-tech à l'entreprise hôte.

1.3.2 Dymension

Dymension, possède un panel de meubles un peu plus large. En effet, il est possible de trouver sur leur catalogue des tables de terrasse de restaurant. Mais également d'autres réservées aux personnes à mobilité réduite ainsi qu'aux enfants.

La particularité de Dymension et qu'en plus de fournir les tables aux entreprises. Elle dispose également d'un panel d'applications d(y)imensionnées parfaitement pour leur catalogue. Notamment quelques jeux multi-joueurs, des menus sur lesquels on peut commander directement en restaurant, un espace de travail collaboratif (table de réunion...).

1.4 Redéfinition des objectifs

L'objectif principal est la réalisation d'un réseau domotique dont la centrale sera l'unité centrale dans la table. Pour se faire il est necessaire de connectée le meuble au réseau et de réaliser l'environnement de la table. Avant toutes choses, il est nécessaire de définir quel réseau de capteurs sera utilisé.

Pour une ergonomie de l'ensemble, nous partirons sur un réseau sans fil. Bien qu'un réseau filaire de type ModBus ou CAN aurait pu être envisagé si l'on parlait d'une maison entière à mettre en réseau, il est plus pratique (et sûrement moins cher) dans la réalisation de ce projet de partir sur un réseau sans-fil. Différentes possibilités s'offre à moi. Je choisis de développer un réseau domotique utilisant le réseau ZigBee. La particularité de ce réseau est qu'il dispose d'une portée suffisante à un réseau domestique (entre 10 et 100m en fonction de la puissance) et qu'il utilise de petits émetteurs à faible

consommation d'énergie. Cependant il ne propose qu'un débit de communication assez court (max 250 kb/s). Son développement a bas coût l'a propulsé parmis les réseaux classiquement utilisés dans le cadre de projet domotique.

Il est possible de différencier trois parties majeures dans la réalisation du projet :

- Permettre à la table l'accès au réseau Xbee
- Développer une application de contrôle sur la table
- Mettre en place le réseau Xbee avec les différents capteurs et actionneurs.

Afin de mettre l'unité centrale du meuble connecté sur le réseau Xbee, l'idée d'une clé USB Xbee émerge. Ce périphérique, à l'instar d'une clé WIFI, permettra d'émettre et de recevoir les informations via le réseau Xbee. Il sera alors nécessaire de réaliser électroniquement la clé USB ainsi que de développer un driver permettant l'utilisation de celle-ci. On réalisera également une application permettant l'usage de ce pilote afin de donner des ordres aux différents actionneurs du réseau. Il sera donc nécessaire de concevoir ces actionneurs.

2 Réalisation du projet

2.1 Contrôleur Xbee USB

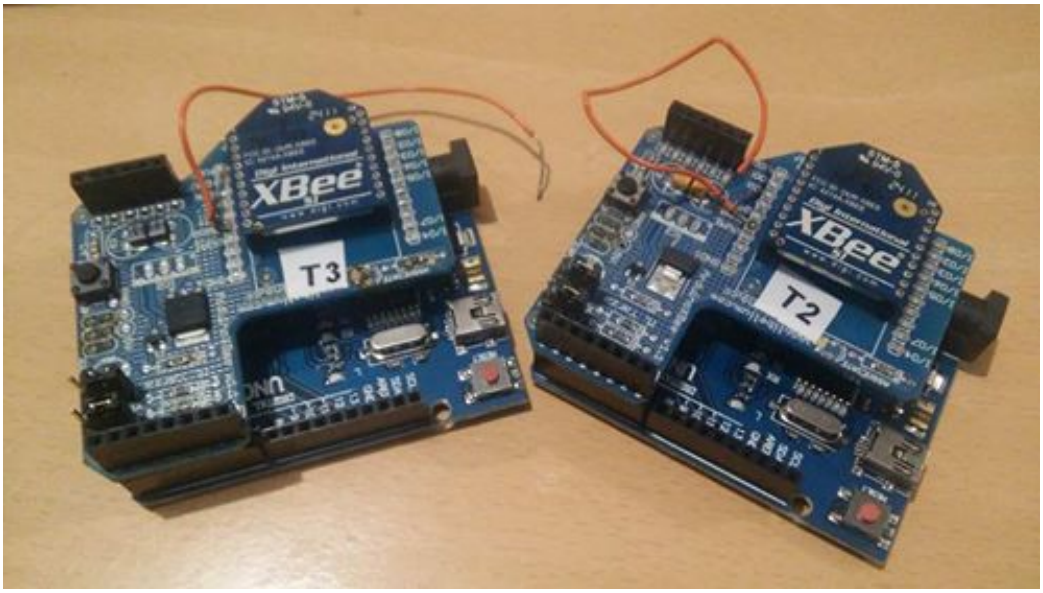
La majeure partie du travail du projet a résidé dans la réalisation de la clé USB. En effet en plus de la programmation de la carte et du driver permettant de communiquer avec, il fut nécessaire de concevoir et fabriquer différents PCB. Et tout ne s'est pas passé comme prévu durant la réalisation des cartes...

2.1.1 Partie Électronique

La première étape de la réalisation du circuit électronique est la preuve de concept. En effet, avant de commencer la réalisation de la carte, il faut s'assurer que cette dernière fonctionnera.

2.1.1.1 Une première communication

Pour se faire, j'utilise alors deux Arduinos sur lesquels je branche sur chacun un bouclier de communication. Ce bouclier permet de brancher un module Xbee sur l'Arduino.



Il est cependant nécessaire au préalable de paramétrer chaque module Xbee. Après une étude de la documentation des modules. Trois paramètres majeurs doivent être définis afin que la communication puisse avoir lieu :

- Le canal de communication : Il s'agit du choix de la fréquence de communication du Xbee. Il est possible de choisir parmi 16 canaux de fréquence dans la bande 2,4 GHz. On choisira le canal 0x0B
- L'ID de la communication : Il s'agit du second paramètre à définir afin de permettre aux modules de communiquer entre eux. Il est nécessaire que tous ces modules soient sur le même ID (ainsi que sur le même canal évidemment)
- La Baud Rate : Il s'agit de la vitesse de la communication série. On peut imaginer, dans le cas de deux Arduinos communiquant en série, que les deux modules Xbee serviront de "câble" en reliant les broches TX et RX des deux Arduinos. Il est donc nécessaire de définir la même vitesse pour les deux modules, vitesse qui doit être supportée par les Arduinos. On prendra alors par défaut une vitesse de 9600 bauds.

Un programme simple de test de communication consiste en un système maître/esclave. Une des deux Arduinos ordonne à l'autre d'allumer sa LED connectée au pin 13 et lorsque cette dernière l'a fait, elle répond via un accusé de réception.

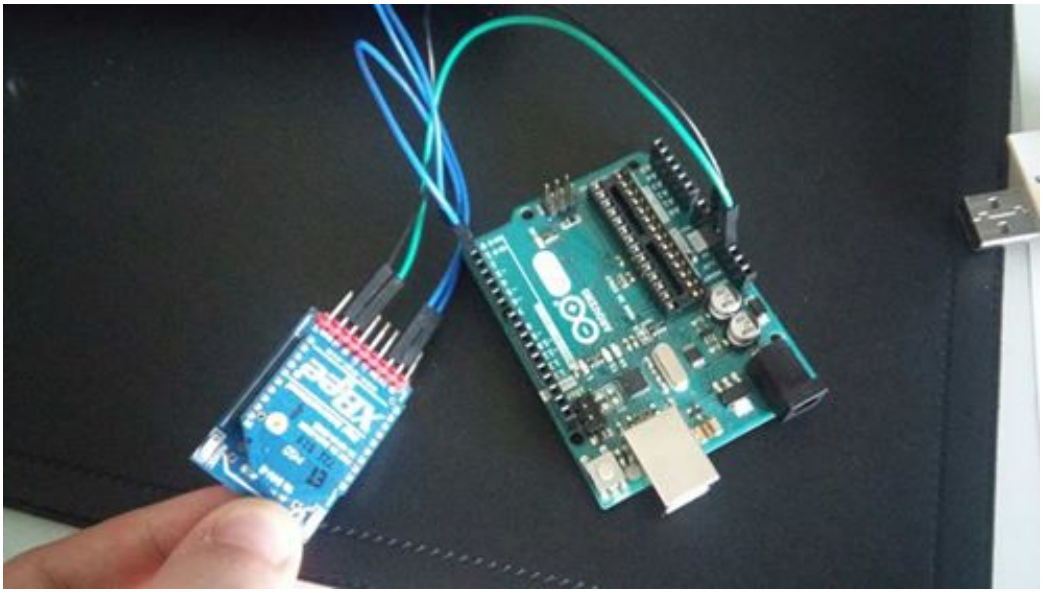
Cette première communication se met rapidement en place, cependant il ne s'agit que d'une petite partie du travail. Il est désormais nécessaire de pouvoir contrôler l'Arduino esclave depuis un ordinateur fonctionnant sous Linux. Pour se faire, je réutilise le travail réalisé lors du Tutorat Système.

2.1.1.2 Premier prototype de clé USB

Lors de ce mini-projet, nous avons reprogrammé un Arduino pour qu'il soit considéré comme un périphérique USB à part entière (au même titre qu'un clavier ou une souris) et non plus comme un périphérique d'échange via le port série.

L'objectif ici est de reprendre l'idée de ce projet en reprogrammant l'ATMEGA16U2 présent sur les Arduinos UNO afin de pouvoir communiquer depuis une application utilisant la librairie libusb avec les modules Xbee. En analysant le schéma électronique d'une carte Arduino, on se rend compte que la communication entre l'ATMEGA16U2 et l'ATMEGA328P se fait via une simple liaison série. Ainsi il est possible de shunter cette communication afin

de connecter directement l'ATMEGA16U2 aux connecteurs du port série des modules Xbee.



Pour se faire, on ôte l'ATMEGA328P de l'Arduino et l'on connecte un module Xbee directement via les pins série de la carte. Attention, il est nécessaire d'inverser le branchement indiqué sur cette dernière. En effet, le pin TX de l'ATMEGA16U2 est connecté au pin RX de l'ATMEGA328P afin que la communication ait lieu. Cependant les pins accessibles depuis les barrettes femelles de l'Arduino sont reliés aux pattes de l'ATMEGA328P. Du coup la broche TX de l'ATMEGA16U2 est celle indiquée par le connecteur RX de l'Arduino, et inversement.

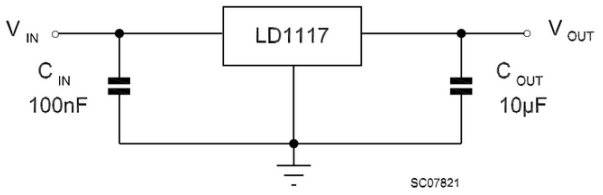
En programmant le microcontrôleur en y intégrant le framework LUFA, il est alors possible de communiquer via le réseau Xbee en utilisant une application console implémentant la libraire libusb. Pour se faire il fut nécessaire de préciser lors de la mise en place de l'architecture USB l'existence de deux interfaces utilisant chacune un Endpoint. Cela fait donc deux Endpoints, un pour chaque direction de communication.

Le circuit fonctionnant parfaitement comme je le voulais, il est alors possible de passer à l'étape suivante, la réalisation du même circuit sur un PCB

2.1.1.3 Conception du premier PCB

Afin de concevoir ce premier circuit, on reprend le schematic d'un Arduino au complet afin de concevoir notre carte. Cependant, nous ne prendrons que la partie concernant l'ATMEGA16U2. En effet, comme expliqué précédemment, seul ce microcontrôleur sera nécessaire.

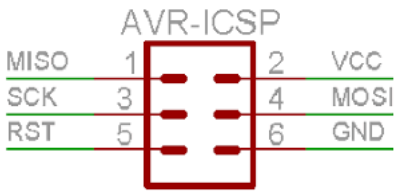
En lisant la datasheet du module Xbee, on se rend compte que ce dernier doit être alimenté par une tension comprise entre 2.8 et 3.4V (3.3 en nominal). Cependant la communication en USB se fait via une alimentation en 5V. Il sera donc nécessaire d'utiliser un convertisseur 5V -> 3V3 afin d'alimenter le Xbee. Après quelques recherches, je décide d'opter pour un LD1117V33. Ce convertisseur nécessite l'utilisation de deux condensateurs, un de 100nF et un autre de 10 µF, comme précisé sur la figure 4 de la datasheet du composant.



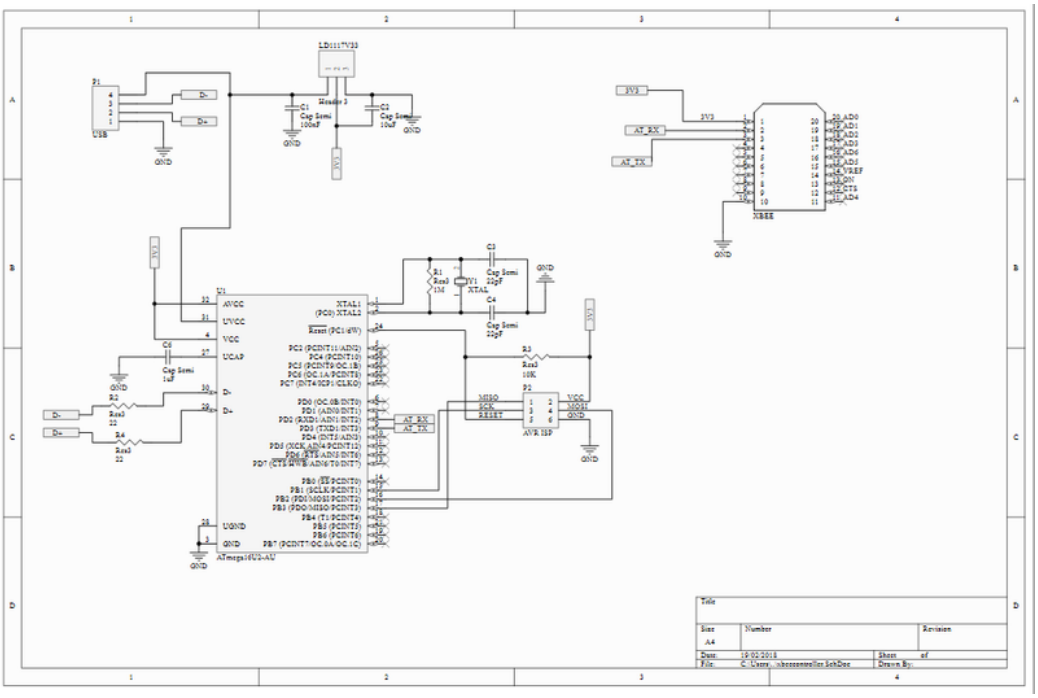
Afin de communiquer avec le module Xbee, seul 4 "cables" sont nécessaires. Les pins d'alimentation, (3V3 et GND), ainsi que les pins de communication série TX et RX. Les autres pins du modules ne sont que des pins de re-paramétrage ou d'entrée/sortie. On supposera dans un premier temps que les modules sont déjà paramétrés en amont, et que l'on ne pourra pas les reparamétrer "on the go"

On remarque également qu'en lisant la datasheet de l'ATMEGA16U2, que le microcontrôleur peut fonctionner dans une plage de 2.7 - 5.5V. Étant donné que les pins digitaux des microcontrôleurs à l'état haut sont à la tension d'alimentation, on alimentera l'ATMEGA16U2 en 3V3 grâce au convertisseur. Cela permettra d'éviter des problèmes de communication entre le microcontrôleur et le module Xbee, et permettra de n'utiliser qu'un seul convertisseur 5V-3V3. On fera fonctionner le microcontrôleur a 16MHz, il est conseillé d'utiliser des condensateurs 22pF directement connectés a l'horloge... C'est ce que l'on fait donc.

Il est également nécessaire de placer un connecteur ICSP afin de pouvoir installer un bootloader sur l'ATMEGA16U2 pour être capable d'installer notre programme dessus. On en profite pour relier la broche de RESET du contrôleur à un pin d'alimentation via une résistance de rappel de tension.



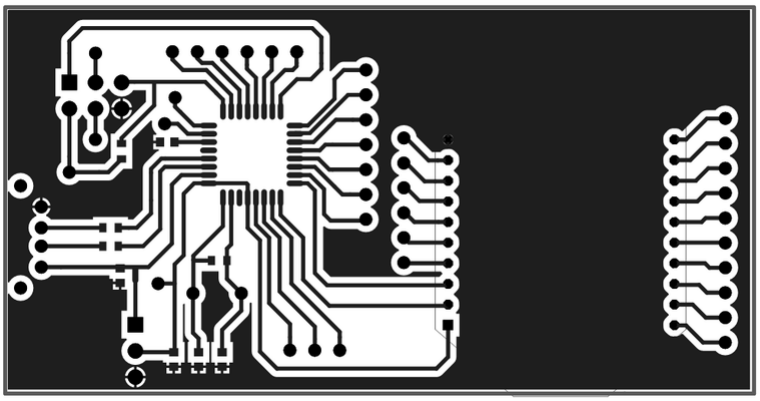
Tout ce raisonnement permet d'obtenir ce schematic :



Il est alors possible, une fois ce schéma vérifié, de procéder à la conception du circuit imprimé. Du fait qu'il s'agit d'un circuit prototype afin de s'assurer que ce circuit simplifié de l'Arduino fonctionne, on ne se préoccupera pas trop

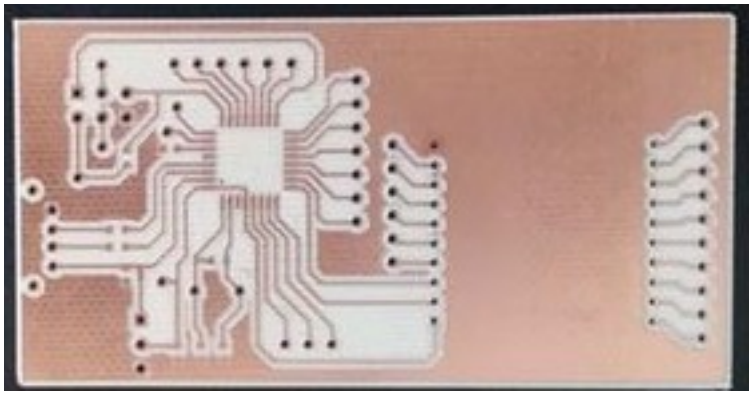
des dimensions de la carte. De plus, en suivant les conseils de M. BOE, je raccorde les broches de l'ATMEGA16U2 et du module Xbee a des barrettes. Sait-on jamais, cela pourrait servir.

Après quelques heures de travail, une première ébauche de circuit est obtenue.



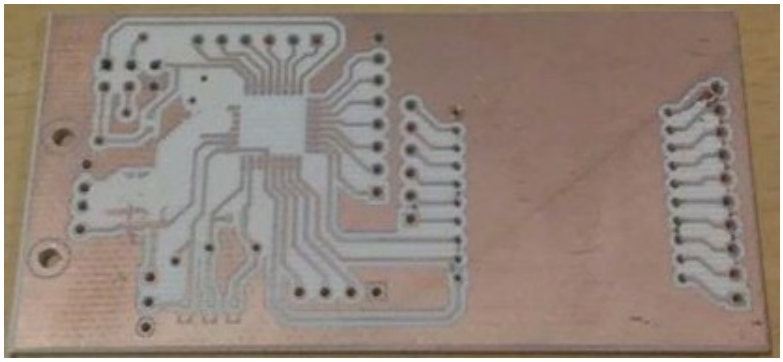
La carte étant validée, elle part en production au service électronique de l'école. Avec l'aide de M. Thierry FLAMAN, cette première carte voit le jour.

On peut alors constater que l'empreinte du connecteur USB choisit ne correspond pas à celui que je possède. De plus, l'écartement des pins de chaque barrettes ne correspond pas à celles standards. Cependant l'empreinte des autres composants, qu'il s'agisse de l'ATMEGA16U2, du module Xbee, du LD1117V33 ou des composants passifs est correct.

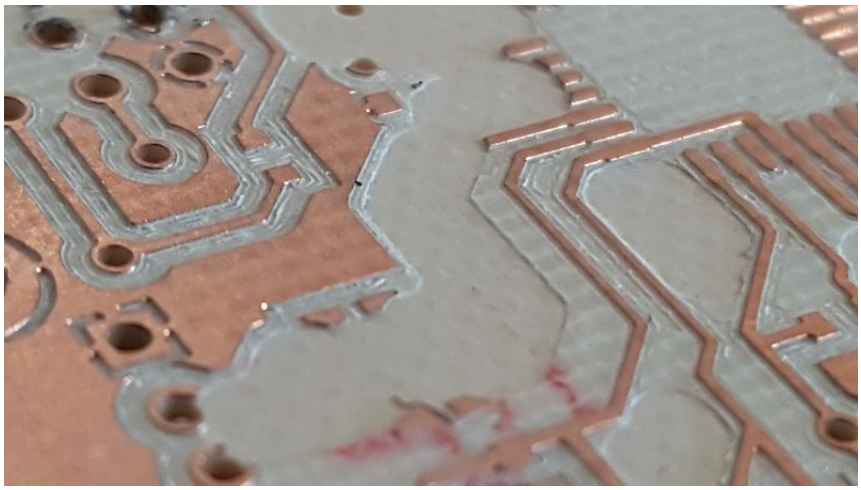


On effectue alors les modifications afin de rendre ce circuit compatible avec tous les composants que l'on dispose. Les fichiers Gerber permettant de fabriquer le circuit sont validés, ils peuvent dont être envoyés à M. FLAMAN pour tirage.

Et c'est là, que les complications commencent.



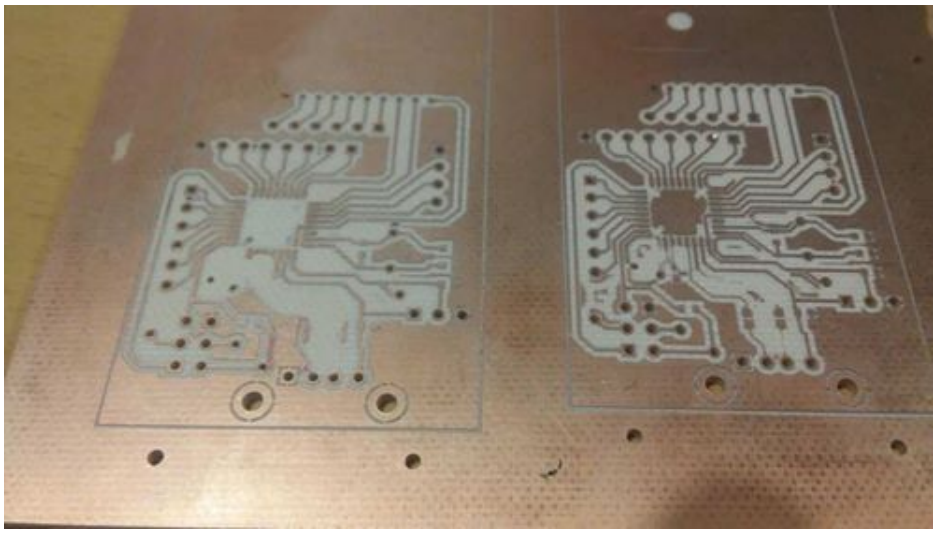
Comme on peut le voir sur cette photo, une partie de la carte a été "mangée" par la graveuse numérique, rendant inutilisable la carte. Cependant, une bonne nouvelle pour rattraper la mise, cette fois-ci, toutes les empreintes sont correctes.



M. FLAMAN retente alors de lancer l'impression de la carte. Le même problème a lieu, exactement au même endroit. Je l'assiste donc dans la maintenance de la graveuse numérique afin d'identifier le problème. Cependant aucun problème n'est relevé durant ce processus.

On relance donc cette fois-ci la même impression, mais nous suivrons tout le procédé sans quitter la machine du regard pour déterminer à quel moment exact, et avec quel outil la machine fait l'erreur. Ainsi il a été possible d'établir la cause du problème. La machine procède à un retrait de matière orpheline entre deux pistes. Les pistes incriminées sont les deux reliant les port de communication de connecteur USB à l'ATMEGA16U2. Cependant suite à une erreur de calcul de la part du logiciel contrôlant la machine, cette dernière utilise un outil de 2mm pour gratter de la matière sur une surface de 0.8mm de hauteur...

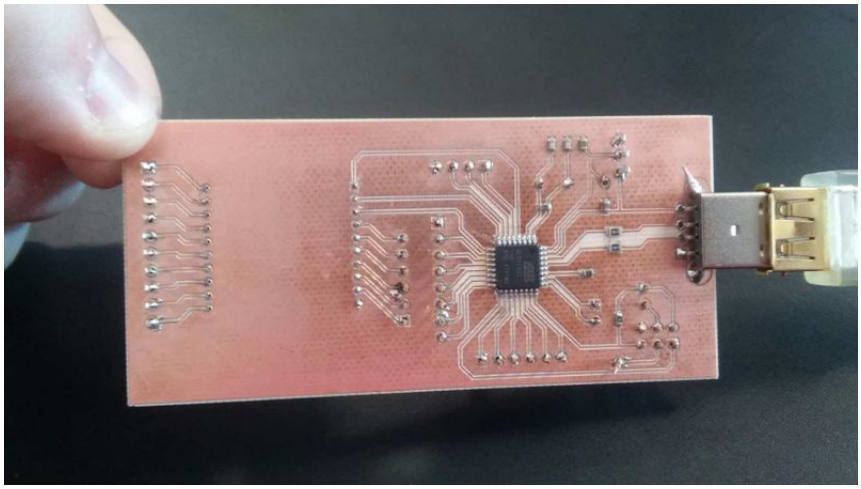
Nous avons essayé, avec M. FLAMAN, de corriger cette erreur manuellement, cependant bien que le résultat est apparant, cela n'est toujours pas exploitable. Vous pouvez voir l'évolution des gravures avant et après re-réglage sur la photo suivante.



Même une régénération des fichiers de gravure n'ont permis d'obtenir le résultat escompté. Ne trouvant pas d'autres idées pour résoudre le problème, et ayant déjà perdu trop de temps, je décide de refaire le projet sur Altium afin de m'assurer qu'aucun fichier ne soit corrompu. En effet en comptant environ 30 minutes d'impressions par essai, 6 essais réalisés, et le fait que je ne pouvais pas tous les faire d'affiler pour ne pas pénaliser mes camarades, beaucoup de temps a été perdu à cette étape.

Après une régénération complète du projet, le redessinnage du circuit imprimé sous Altium, et une ultime vérification avec M. FLAMAN, il est enfin possible d'obtenir une carte exploitable. dessin

Il est alors possible de placer les composants CMS afin de les souder au four disponible au service électronique. Après une vérification des connections des éléments de surfaces, il est possible de souder les composants traversants.



Une fois tous les composants placés, et toutes les soudures réalisées, un essai de branchement est réalisé. Je décide de vérifier la tension en sortie du connecteur USB, le 5V est correct. Il est également possible de mesurer le 3V3 en sortie du convertisseur. Normalement, l'ATMEGA16U2 et le module Xbee sont correctement alimentés.

```
andreas@andreas-X200CA:~$ lsusb
Bus 004 Device 004: ID 0bda:5605 Realtek Semiconductor Corp.
Bus 004 Device 010: ID 1781:0c9f Multiple Vendors USBtiny
Bus 004 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hu
b
Bus 004 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 003: ID 03eb:883e Atmel Corp.
Bus 003 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hu
b
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

Ma curiosité me pousse à faire un essai de la commande lsusb. Cette dernière liste tous les périphériques connectés via les ports USB. Et là, grande surprise. Un périphérique nommé "Atmel Corp" est présent, cela veut dire deux choses :

— La carte fonctionne, et le microcontrôleur est bien allumé.
— Un bootloader est déjà installé sur le contrôleur. Cela réduira un peu la quantité de travail avant d'utiliser la carte. Cela signifie également que l'ICSP que j'avais installé sur la carte perd un peu de son intérêt.

Je fais un essai de l'outil dfu-programmer afin de voir s'il m'est possible de reprogrammer le microcontrôleur directement. L'upload du nouveau firmware fonctionne parfaitement ! On essaie alors de faire fonctionner une communication Xbee via le prototype de clé USB. Pour se faire, je réutilise le programme réalisé en prototypage sur l'Arduino, il ne devrait rien avoir à changer car l'environnement utilisé sur la clé USB est le même que celui de l'ATMEGA16U2 d'un Arduino classique. A l'essai de l'application avec la clé, on se rend compte que des caractères sont bien envoyés, mais ces derniers n'ont aucun sens : au lieu d'une trame de 5 octets de la forme "[X-Y]", j'obtiens un message de 13 octets valant tous '0x80', un caractère non défini dans la table ASCII.

	V_{CC}=2.7-5.5V		V_{CC}=4.5-5.5V		
Parameter	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
Oscillator Frequency	0	8	0	16	MHz
Clock Period	125		62.5		ns

La communication ne se fait donc pas de manière effective, sûrement un problème d'horloge. En relisant la datasheet, je me rends compte qu'il est impossible d'utiliser une horloge de 16MHz si le CPU est alimenté en 3,3V. Je change donc le cristal externe pour une fréquence de 8MHz, qui est compatible. Il s'agissait d'un composant traversant, donc il n'y a pas de soucis pour le changer. Bien qu'un nouveau quartz fut installé, cela ne marchait toujours pas. Pour voir si l'horloge externe est bien utilisée je test un clignotement d'une LED avec un changement d'état toutes les secondes. Je remercie M. Boe pour m'avoir conseillé de raccorder toutes les sorties de l'ATMEGA à des barrettes, en effet sans cela il m'aurait été impossible de connecter une résistance et une LED.

En chronométrant le clignotement de la LED, on remarque qu'elle change d'état toutes les 8 secondes (précisément) au lieu de toutes les secondes. Plusieurs essais avec des valeurs de temps différentes donnent le même résultat. La période d'un état est 8 fois plus longues qu'attendu. Cela signifie que la fréquence du microcontrôleur est 8 fois plus faible.

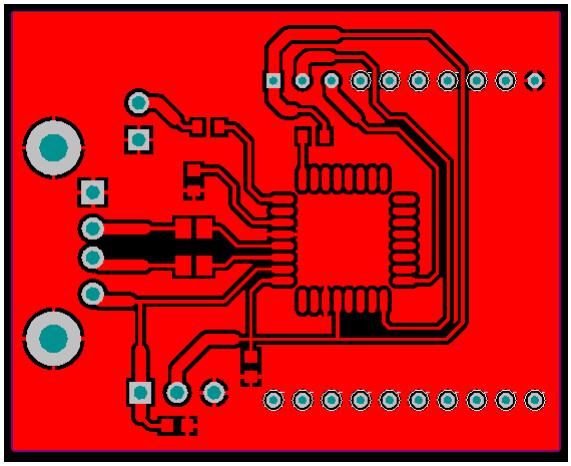
En faisant des recherches sur le net et dans la datasheet de la raison de l'utilisation d'une fréquence de 1MHz, je découvre que les bits de fuses paramétrés avec le bootloader pré-installé semblent utiliser le générateur de clock interne de l'ATMEGA. Cette horloge fournit une fréquence d'utilisation de 8MHz au contrôleur. Cependant, il existe un paramètre dans les bits de fuses permettant de diviser la fréquence par 8, le bit : CKDIV8. Ce dernier doit donc être activé.

Au lieu de changer la vitesse de travail du microcontrôleur, je décide de m'adapter à une fréquence d'1 MHz. Je change alors la cadence du CPU précisée dans le Makefile permettant de générer le .hex, ainsi la bonne fréquence sera utilisée dans les calculs de délai, de baud rate etc... Cependant, une fréquence aussi basse ne permet pas d'avoir une vitesse de communication très élevée, il est donc nécessaire de passer à une baud rate plus faible.

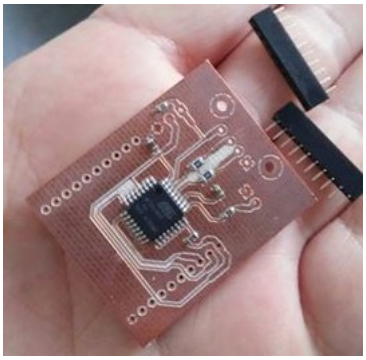
f_{osc} = 1.0000 MHz				
Bit Rate (bps)	**U2Xn = 0**			
	UBRR (dec)	**UBRR (hex)**	**Actual Bit Rate**	**Error**
300.00	207	0x0CF	300.48	0.2%
600.00	103	0x067	600.96	0.2%
1200.00	51	0x033	1201.92	0.2%
2400.00	25	0x019	2403.85	0.2%
4800.00	12	0x00C	4807.69	0.2%
9600.00	6	0x006	8928.57	-7.0%

J'utiliserai donc une baud date de 4800 au lieu de 9600, je recompile le programme avec tous les changements, re-téléverse le programme dans l'ATMEGA. Il est également nécessaire de re-paramétrer tous les modules Xbee pour qu'ils utilisent aussi la même vitesse. Une fois cela fait, on réutilise le sniffer Xbee et l'application XCTU afin de voir si les messages sont corrects. Et enfin, la communication s'effectue sans problèmes. Il est alors parfaitement possible d'utiliser ce prototype de clé USB pour contrôler les LEDs des différents Arduinos. La carte étant fonctionnelle, il est alors possible de travailler sur la version finale de la carte.

2.1.1.4 Circuit final

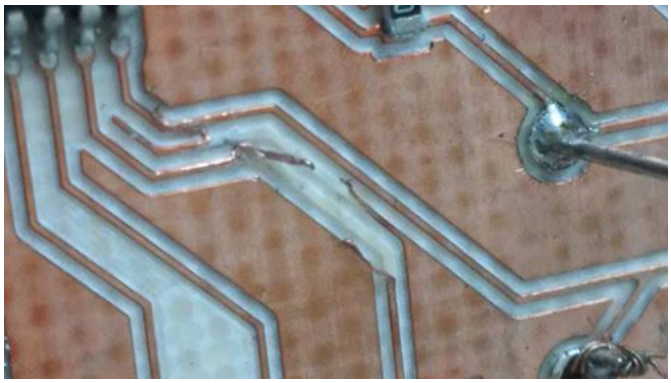


En prenant en considération la pré installation d'un bootloader sur l'ATMEGA16U2 ainsi que l'utilisation par défaut de l'oscillateur interne, je décide d'enlever l'ICSP et le cristal externe afin de gagner de la place sur le circuit. Ainsi, il est possible de réduire grandement la taille de carte et d'obtenir un PCB de 2,8 x 3,5 cm. Une taille beaucoup plus acceptable pour une clé USB.

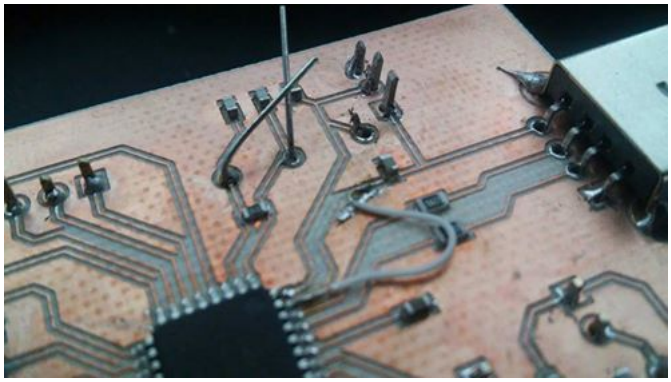


Une fois la soudure de tous les composants réalisée, je procède aux différents essais afin de m'assurer de l'intégrité du circuit. Cependant bien que les tensions appliquées sur le l'ATMEGA et le module Xbee semblent correctes, la carte n'est pas reconnue par l'ordinateur.

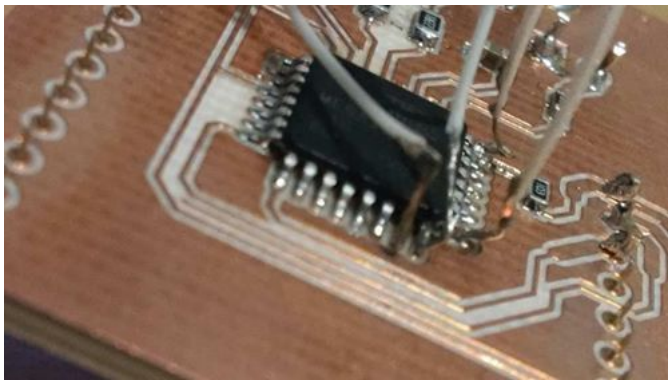
L'ATMEGA16U2 étant normalement livré avec un bootloader pré-installé comme on a pu le voir sur la précédente clé, j'explore différentes possibilités en comparant les niveaux de tensions entre les deux cartes en chaque point du circuit. Cependant, lors de la mesure de tension sur la piste reliant le connecteur USB au CPU du prototype de la clé USB, j'effectue un court-circuit. De cette erreur résulte la surchauffe de la piste, la détruisant au passage comme on peut le voir.



Bien qu'il ne s'agisse que d'une tension de 5V, un court-circuit délivre suffisamment de courant pour cramer une piste de 0.2mm de large. Il a donc été nécessaire de réparer le circuit en redessinant la piste avec un fil. L'aide de M. REDON a grandement était utile afin de sauver la carte. Il a fallut un certain à l'ATMEGA pour se remettre de ses émotions avant qu'il n'accepte de fonctionner à nouveau.



Après des essais de détection de différence non concluant, l'option de l'inexistence du bootloader est envisagée, bien qu'elle était initialement écartée. L'absence d'ICSP sur la carte complique grandement les choses, il fut alors nécessaire de le souder directement sur les pattes du CPU.



Ne possédant pas de programmateur ISP, un Arduino a été reprogrammé en tant que tel. Il est alors possible de programmer un microcontrolleur AVR depuis cet Arduino et l'IDE. Il est également possible de le faire directement en console via l'outil avr-dude (outil utilisé par l'IDE Arduino).

Bien qu'il fut possible de reprogrammer l'ATMEGA16U2 d'un autre Arduino via cette technique, celui de ma clé USB semble ne pas du tout être reprogrammable. Il fut impossible de lire ne serait-ce que la signature AVR du composant, impossible de dire informatiquement qu'il s'agit d'un ATMEGA16U2 que l'on souhaite programmer. Le gros de la partie informatique du projet ne pourra donc être utilisé que sur les cartes de prototypages.

2.1.2 Partie Informatique

2.1.2.1 Programmation de l'ATMEGA

La programmation du comportement de la clé s'est fait sans trop d'encombre comparée à la partie électronique. Les différents essais ont été réalisés grâce à l'Arduino de preuve de concept. En effet, bien que l'architecte matérielle des 3 cartes change légèrement d'une version à l'autre, l'architecture logicielle est la même. Il est donc tout à fait possible de transférer le même programme réalisé sur l'Arduino de test vèrs les différentes clés USB.

Afin de reprogrammer un ATMEGA16U2 sur lequel un bootloader est installé, il est nécessaire d'effectuer un reset matériel. Pour se faire, il suffit d'effectuer un court-circuit entre la broche de reset et la masse de la carte. Cette étape permettra de faire entrer le CPU dans le mode Firmware Upgrade. Afin de vérifier que cette étape est correctement réalisée, la commande lsusb devrait lister parmi les périphériques connectés un produit du nom "Atmel Corp".

Si tout est bon, l'utilitaire dfu-programmer (Device Firmware Upgrade) permettra de reprogrammer le composant via les commandes suivantes :

```
dfu-programmer atmega16u2 erase
dfu-programmer atmega16u2 flash XbeeController.hex
dfu-programmer atmega16u2 reset
```

Il est donc au préalable nécessaire de compiler le programme utilisant le firmware LUFA afin d'obtenir un fichier .hex qui sera televerser au CPU. Ce programme n'est pas très compliqué. En effet, l'ATMEGA16U2 tiendra uniquement le rôle de douane. Il laissera transiter les messages provenant de la liaison USB vers la liaison série (ou inversement) s'ils sont conformes.

Les codes du programme qui sera téléversée vers le CPU sont trouvable dans le dossier XbeeControlerUSB/PolytechLille du repository git.

2.1.2.2 Librairie libusb

L'objectif de cette partie de la programmation était la remise au propre du code réalisée lors du Tutorat Système. Beaucoup de code fantôme fut enlevé, et une cible de déverminage activant les affichages dans la console a été ajoutée dans le Makefile.

Cette réécriture du code avait principalement pour but de permettre la création d'une librairie .a permettant d'utiliser la clé USB plus facilement dans les différentes applications, mais également pour permettre un portage de cette partie du projet à d'autres travaux futurs.

Ainsi l'ensemble de la librairie USB, un peu complexe a été simplifiée en 4 fonctions élémentaires. Une fonction de connexion, d'écriture de message, de lecture ou de déconnexion de la clé. Pour se faire, beaucoup d'éléments sont passés en variables globales dans le code afin de permettre le bon fonctionnement de celui-ci.

Ainsi pour utiliser le code, il suffit d'effectuer les commandes suivantes :

```
1     char msg[6] = "HELLO";
      init_xbee_controller();
3     write_endpoint(msg, sizeof(msg)-1); //Permet de ne pas ecrire
      le caract re de fin de chaine
      int x = read_endpoint();
5     close_xbee_controller();
```

Cela simplifie grandement l'utilisation de la libusb qui dispose d'une architecture utilisant une arborescence complexe.

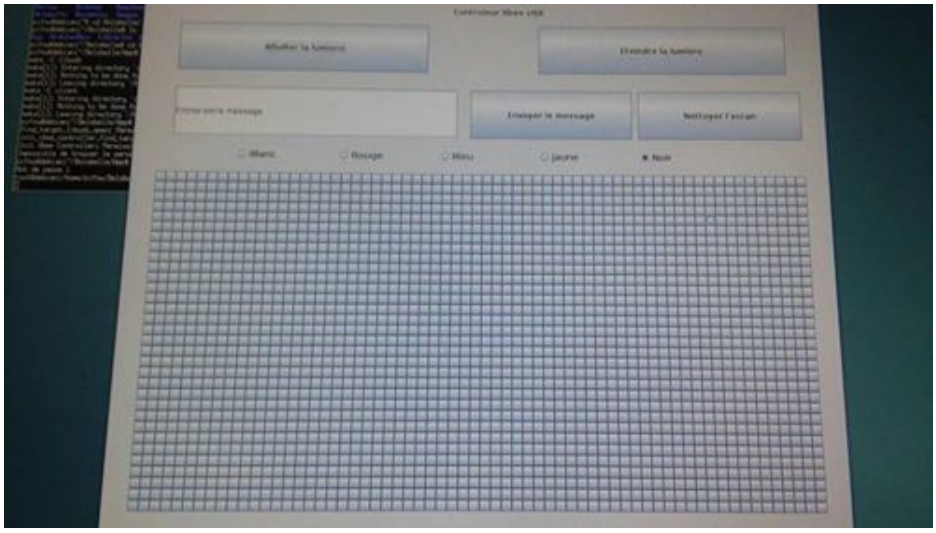
2.2 Application graphique

L'objectif était d'utiliser la librairie précédemment décrite au sein d'une application graphique afin de contrôler le réseau Xbee. Le développement d'interface graphique grâce au langage C étant un peu complexe, je décide d'utiliser un second processus dans un autre langage afin de développer cette IHM.

Cependant afin de communiquer entre deux processus distinct, une relation doit être établie. Comme vu en cours de réseau, l'utilisation d'une communication via des sockets semble le plus simple à mettre en place. Une mise en place rapide d'une librairie permettant l'usage de socket rend capable le contrôle du réseau Xbee depuis un autre processus client réalisé en C.

Ce second processus a donc vocation d'être remplacé par une interface graphique. S'inspirant du projet réalisé par mes prédécesseurs sur la table, l'idée d'une interface graphique WEB est explorée. Cependant, on se heurte à un obstacle de taille. Il est impossible d'utiliser les sockets classique depuis un navigateur web. Seuls les websockets sont exploitables. Ne souhaitant pas changer le fonctionnement de mon serveur socket qui marchait parfaitement comme je le voulais, je décide d'explorer d'autres possibilités d'interfaces.

Après quelques recherches, le langage Java permet à la fois la mise en place de client socket INET et d'interface graphique. Nous optons donc pour cette solution. Un premier essai d'une application console Java permet de s'assurer de la communication socket entre le serveur implémentant la librairie XbeeController et le client Java. Il est alors possible de s'attaquer au développement de l'interface graphique à part entière.



Cette interface est réalisée via le package Swing. Ce package permet la mise en place facile d'une interface graphique basique. La logique d'une telle interface ressemble à celle utilisée par la librairie Tkinter en Python. Chaque élément graphique représenté par un widget (boutons, fond, champ de saisie, image etc..) vient se superposer aux autres widgets sur une fenêtre définie initialement. Il suffit alors, une fois que l'interface est définie, de relier les évènements de chaque widget à des envois de messages via la liaison socket. Messages qui seront interprétés par le serveur C et transmis sur le réseau Xbee.

Il ne reste donc plus qu'à mettre en place les différents actionneurs et capteurs du réseau que l'on pourra contrôler depuis la table.

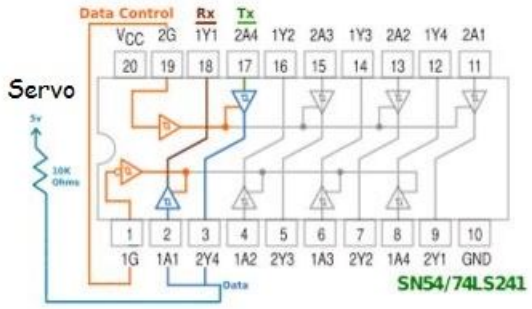
2.3 Périphériques annexes

2.3.1 Activateurs d'interrupteur

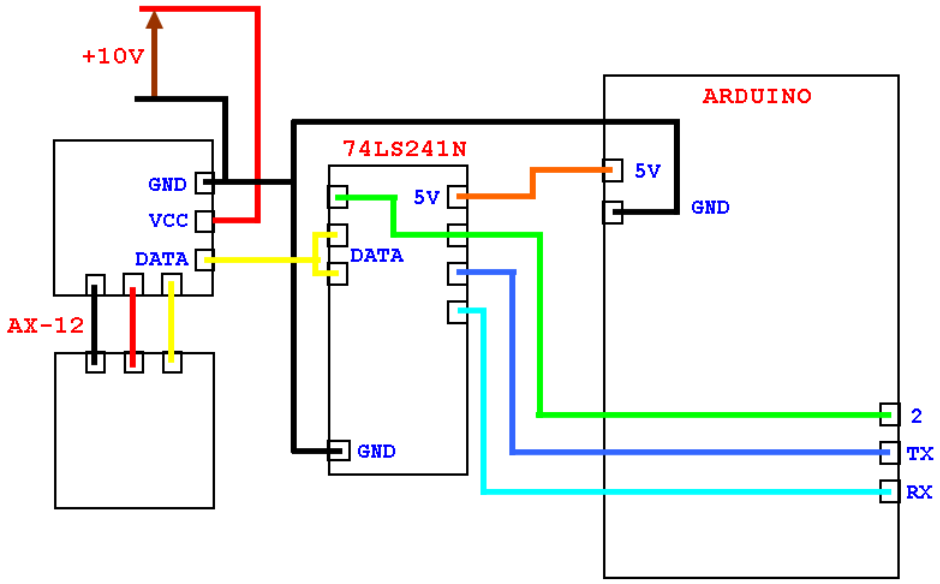
Une idée simple d'actionneur à but domotique est un système de deux moteurs que l'on poserait sur l'interrupteur de la lumière d'une salle afin de contrôler l'éclairage de la salle à distance.

Pour se faire, on utilise deux moteurs AX-12 de la marque Dynamixel. Il s'agit de servo-moteur utilisés dans les robots Bioloids. Ces moteurs ont la particularité d'être alimentés entre 7 et 11V, d'être contrôlable en vitesse et en position, et de proposer des retours d'informations sur le nombre de tours réalisés ou la position du moteur. Mais surtout de pouvoir contrôler plusieurs moteurs via un seul et même réseau.

Le contrôle de ces moteurs se fait via une communication série bi-directionelle (sur un même câble) à 1 mégabaud. Cette communication est également accompagnée d'un protocole de communication particulier. Ainsi, une seule communication série permet de contrôler plusieurs moteur.



Afin de pouvoir communiquer avec ces moteurs, l'utilisation d'un buffer 3 états est nécessaire. En effet, nous allons fusionner les connecteurs TX et RX de l'Arduino au sein d'un 74LS241N (Octal Tri-State Buffer) afin de pouvoir les réunir en un seul cable. Ce buffer trois états permettra "d'ordonnancer" le sens de la communication entre le réseau de moteur et l'Arduino. Le principe du montage est expliqué par le schéma suivant.



Les broches de données 1 et 2 côté AX-12 étant reliées entre elles, l'état activé ou non du pin Data-Control de l'octal buffer permettra le transfert dans un sens ou dans l'autre de la donnée vers l'Arduino.

Il est alors possible de contrôler ces moteurs depuis la table en connectant un module Xbee à un second port série (Il est donc nécessaire d'utiliser un Arduino Mega)

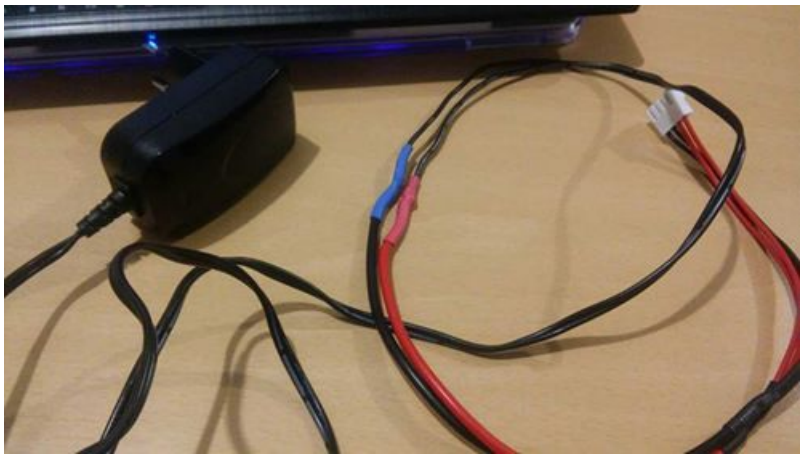
2.3.2 Matrice de LED

L'objectif de l'utilisation de cette matrice était de montrer qu'il est possible de communiquer avec des périphériques nécessitant plusieurs informations pour fonctionner. Contrairement aux actionneurs d'interrupteur, ici la matrice possède plusieurs fonctionnalités.

La première : la possibilité d'écrire directement un message (maximum 6 caractères) sur la matrice. Mais également de pouvoir directement dessiner sur la matrice via un tableau de boutons présents sur l'interface graphique. Il est également possible d'effacer le contenu de la matrice.

L'utilisation de cette matrice nécessite une alimentation de 5V - Max 4A. Je décide alors de recycler une alimentation 5V/3A qui était destinée a recharger un autre appareil. Certes, la consommation maximale n'est pas

atteinte, cependant 3A devrait être suffisant pour faire fonctionner la matrice, même si elle ne brille pas à son maximum. Avec l'aval de M. REDON, je sectionne le câble reliant la sortie du transformateur à la connectique, dénude le câble et le soude avec le connecteur de l'alimentation fournie avec la matrice. On fait attention à souder les deux câbles d'alimentation dans le bon sens. Cette alimentation improvisée peut être finalisée en utilisant de la gaine thermoformable afin de ne pas laisser le cuivre apparent.



Afin d'utiliser cette matrice plus facilement, il est possible de réaliser un bouclier venant se connecter à un Arduino Mega. Il est ici nécessaire d'utiliser un Arduino MEGA car un UNO ne dispose pas suffisamment de RAM pour faire fonctionner le programme de la matrice.

Il fut possible de reprendre le bouclier qui avait déjà été réalisé par des étudiants lors du module Internet des Objets afin de gagner du temps dans la conception. Il n'a fallut que voir avec M. FLAMAN afin de réimprimer la carte.

Cependant un autre problème technique eu lieu lors de l'impression de cette carte. Ce problème avait pour problème un erreur réalisée par M. FLAMAN lors de la maintenance de son outil : la planéité du plan de gravure des cartes n'était plus assuré. Cela a un peu ralenti l'obtention de ce PCB, mais pas autant que lors de la dernière fois.

Une fois la soudure réalisée, et les différents ordres définis, il fut une nouvelle fois possible de contrôler la matrice de LED depuis la table

3 Finalisation du projet et ouverture

3.1 Idées utilisant cette technologie

Cette technologie permet le développement de beaucoup de produit, de par sa simplicité d'utilisation.

La communication étant assez rapide à mettre en place une fois l'application installée, il est alors très facile de pouvoir contrôler différents périphériques depuis la table. Toujours dans un esprit domotique, on peut très bien imaginer contrôler des vannes robotisées pour réguler la température au sein d'une salle, mais également des volets électriques, des ventilateurs etc...

Il est de plus possible d'utiliser cette carte comme récepteur. Ainsi il est possible d'obtenir des informations de certains périphériques, comme par exemple des accusés de réception ou des validations d'action. Il est également possible de proposer un panel de capteurs dont on pourrait récupérer les données à distance.

3.2 Limites actuelles

L'implémentation actuelle du système sur les différents périphériques fait que lorsque le réseau devient surcharger en ordres simultanés, certains actionneurs perdent le fil de leurs tâches. Ceci pourrait être expliqué par l'implantation du système à 4800 bauds ou des temps de calculs trop long sur les périphériques. Je pense notamment à la matrice de LED qui rate certains ordres lorsque l'on appuie trop vite sur différents boutons.

De plus les système actuels nécessite encore une alimentation externe provenant d'un ordinateur. En effet les actionneurs fonctionnent encore via un système de scrutation de la liaison série. Cette méthode est très énergievore et ne permet pas l'utilisation de pile pour l'alimentation. La mise en place d'une lecture sur interruption aurait pu résoudre en partie se problème.

3.3 Questionnement sur la sécurité

Comme montré via le logiciel XCTU et un "sniffer" Xbee, il est très simple d'analyser les communications transitant dans l'air, et de pouvoir envoyer des ordres mal-intentionnés. Ceci est d'autant plus facilité par l'utilisation d'un protocole de communication compréhensible par l'humain. De plus la

communication série utilisée est assez standard (4800 baud, 8 bits de données, 1 bits de stop, pas de parité) et sera vite détectée.

Afin de palier à ce problème, il est tout d'abord imaginable d'utiliser un protocole de communication plus poussé et d'utiliser une liaison série un peu moins "évidente". Par exemple, le temps (estimé par le logiciel XCTU) nécessaire au scan de toutes les combinaisons possible de paramétrage de liaison série prendrait environ 6H30. De plus cette combinaison pourrait être redéfinie dynamiquement et aléatoirement pour les différents périphériques lors de l'utilisation. En utilisant par exemple une synchronisation via une horloge de temps commune à tous les périphériques.

Il est également possible de redéfinir dynamiquement le canal de communication et l'ID du réseau dynamiquement afin de changer régulièrement la fréquence à laquelle les modules Xbee émettent.

3.4 Ce qui n'a pas été fait

Beaucoup de temps fut perdu de part les différents échecs de réalisation de la clé USB. Ceci a rendu impossible la réalisation de certaine partie du projet. Il était par exemple question de créer un boîtier en 3D dans lequel se serait placer la clé USB.

Il n'a de plus pas été possible de mettre en place des capteurs en plus d'actionneur qui auraient pu communiquer avec l'application présente sur la table. Pourtant je disposais du matériel nécessaire à la réalisation de capteur de température ou de luminosité.

De plus, une toute autre partie du cahier des charges initial concernait l'utilisation de tablettes Android connectées à l'unité centrale de la table. Ces tablettes devaient être capable de pouvoir également utiliser la clé USB via la librairie créée à cet effet. Cependant le manque de temps a fait que cette partie du travail n'a pas du tout été entrevue lors du projet.

Conclusion

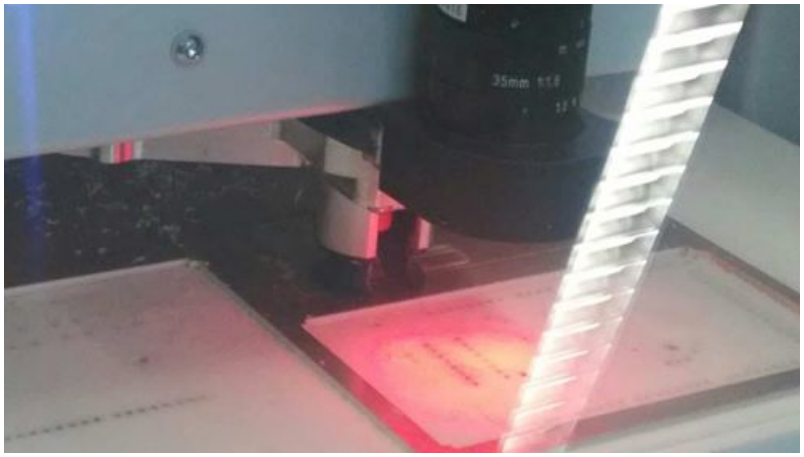
Bien que tout ce qui avait été prévue dans le cahier des charges initial n'a pu être réalisé, ce projet m'a permis de beaucoup progresser tant d'un point de vue informatique qu'électronique. La partie informatique du projet à permis de solidifier et de mettre à profit les connaissances obtenues lors du Tutorat Système du S7 et du cours de réseau de ce semestre. Et la partie électronique a permis de m'initier à une "vraie" conception de circuit imprimé de A à Y (La dernière partie Z n'ayant pas fonctionné...). Bien que j'ai eut affaire a plusieurs problèmes lors de la réalisation de mes différentes cartes, j'ai beaucoup apprécier travailler sur cette partie que l'on voit très peu lors de notre formation.

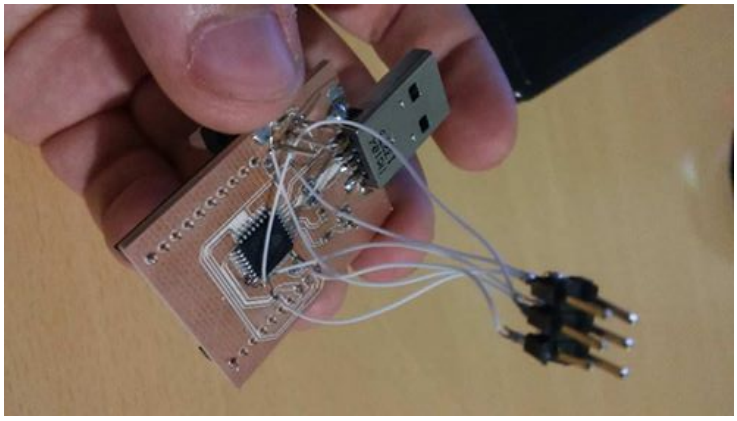
La technologie de meuble connectée est très intéressante, surtout d'un meuble aussi grand avec autant de possibilité. Lors de l'utilisation de la table, beaucoup d'idée de potentiels projets me sont parvenus. Notamment l'utilisation de la table à des fins ludiques (jeux de société interactifs, jeux vidéos multi-joueurs) mais également à des fins plus professionnelles (espace de travail collaboratif et participatif, par exemple pour des dessins industriels)

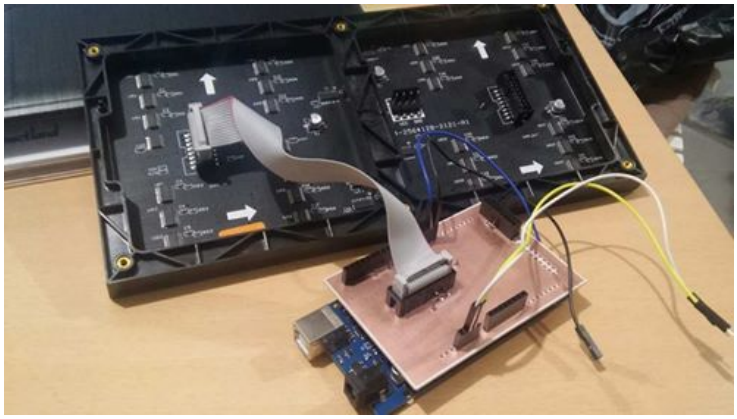
Vous trouverez en annexes différentes photos prises lors de la réalisation du projet, ainsi qu'une bibliographie regroupant les différents sites qui ont pu m'offrir une aide. L'ensemble des codes de mon projet est disponible sur mon github public : http ://github.com/Ruubbis/ProjetIMA4

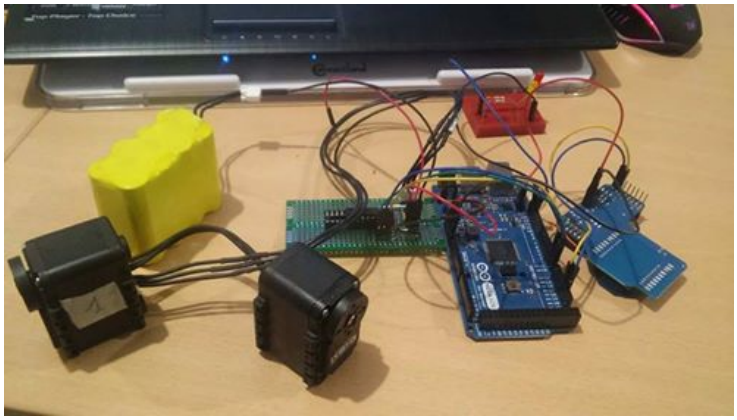
Annexes

Photos









Sources

Les codes du projet sont disponible sur mon github public : http ://github.com/Ruubbis/ProjetIMA4

Liste des datasheets utilisées (liens cliquables) :
— ATMEGA16U2
— XBEE S1
— LD1117V33
— Dynamixel AX-12
— 64*32 LED Matrix

Liste des sites webs utilisés :
— https ://rex.plil.fr
— https ://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader
— http ://www.instructables.com/id/How-to-Restore-the-Arduino-UNO-R3-ATmega16U2-Firmw/
— https ://projets-ima.plil.fr/mediawiki/index.php/BIOLOID_commandé_par_Arduino
— https ://www.javatpoint.com/java-swing
— http ://libusb.sourceforge.net/api-1.0/
— http ://www.fourwalledcubicle.com/LUFA.php