Rapport de projet IMA4

Projet 57

Mise à jour over the air

Florent LEROY

Sommaire

1. Description du projet

a) Introduction

Les capteurs sont omniprésents dans le domaine de l'entreprise. Une usine peut disposer de plusieurs centaines de capteurs répartis dans toute sa structure. Ces capteurs ont des rôles différents comme la mesure de la température, la présence d'une pièce où encore la mesure métrique d'un élément.

Chacun de ces capteurs peut être amené à être mis à jour afin de modifier un temps de réponse ou changer la couleur de détection d'un capteur optique etc…

Ces mise à jours peuvent être fastidieuses à implémenter car nécessite l'intervention physique d'un opérateur et la mise à jour se fait pour chaque capteur individuellement.

Différentes solutions existent déjà, notamment MYSBootloader qui est un bootloader qui utilise un firmware envoyé à chaque redémarrage alors que le bootloader DualOptiboot permet de démarrer à l'aide d'un firmware sur une mémoire flash. Aucune des deux solutions peut être implémentée dans le cadre industriel afin de mettre à jour des capteurs et qui ne nécessiterait pas de récupérer une mise à jour via les ondes à chaque redémarrage.

b) Les objectifs

L'objectif de ce projet est de proposer un système permettant de reprogrammer un capteur de manière OTA (Over The Air) sans aucune intervention physique et pouvant s'appliquer a un grand nombre de capteur.

Le système devra être :

1. **Sécurisé** : la mise à jour devra être vérifié (provenance, checksum) avant l'implémentation.
2. **Économe en énergie** : l'impact énergétique sera pris en compte dans le choix des technologies.
3. **(Bonus)** Le système sera générique afin de pouvoir mettre à jour un groupe de capteur automatiquement

Avantages :

- Hautement sécurisé : chaque mise à jour sera vérifiée à l'aide d'un checksum.
- Peut théoriquement s'adapter à tout type de capteur.
- Temps de mise hors service très court. (Par rapport à une maintenance physique)

Inconvénients :

- Coûts de fabrication plus important
- Pas de retour vis à vis du nombre de capteur mis à jour

c) Orientation du projet

La concertation avec les encadrants m'a plutôt orienté vers une conception de carte (qui ma pris la majorité du temps consacré au projet) ainsi que la réalisation d'un code plus simple en réduisant la partie sécurité à une vérification de l'entête de la mise à jour et un checksum avant d'appliquer une mise à jour.

Cette carte a pour but de montrer qu'il est possible de réaliser des capteurs à faibles coûts et permettant de se mettre à jour en toute sécurité via des ondes radios et rapidement. Les ondes radio permettent d'éviter d'utiliser un réseau wifi ainsi qu'une connexion internet qui expose à des risques de piratage. Les risques de piratages sont également présent mais peuvent être limités grâce à un système de cryptographie asymétrique (théoriquement implémentable dans le bootloader du microcontrôleur).

La partie sécurité peut être améliorée par la suite car la conception de la carte permet de stocker une mise à jour avant de l'appliquer à l'aide d'une mémoire vive type SRAM ou une mémoire morte type Flash.

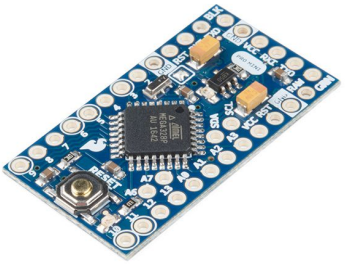
La partie code quant à elle se résumera à une modification du bootloader optiboot (libre de droit) qui permettra lors du boot d'appliquer une mise à jour reçue via onde radio.

2. Conception de la carte

a) Choix des composants

L'idée était de créer une carte qui puisse réagir de différentes manières afin de simuler des capteurs de différent type (capteur de pression, afficheur led …). Un bouton poussoir a été ajouté afin de tester les interruptions. Des leds ont été également introduite dans la conception afin de permettre d'afficher dans quel état on se situe lors du boot et ainsi pouvoir réaliser les différents tests. Ces leds seront également utiles lors de la phase de débogage.

Pour réaliser la carte, je me suis inspiré de l'arduino pro mini qui est une arduino pouvant fonctionner à 5V comme à 3.3V. A 16MHz comme à 8MHz, cette arduino est simpliste et permet donc de se rapprocher de notre objectif qui est de développer un capteur pouvant être mis à jour over the air à faible coût.

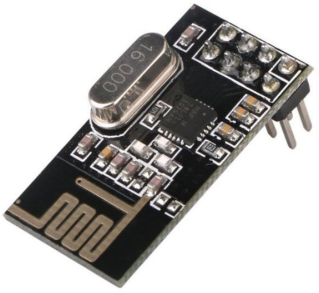


Cette carte sera constituée d'un atmega328p qui nous permet d'obtenir un espace de taille, à priori, suffisante afin de programmer le bootloader. Cependant, un microcontrôleur de taille supérieur aurait également pu être utilisé pour ce projet.

Ayant des difficultés pour choisir le type de mémoire à placer, j'ai décidé de placer les deux types de mémoire. La mémoire vive permet un accès bien plus rapide aux informations mais la non persistances des données ne permet pas de réinstaller la mise à jour précédente en cas de bug. Je pourrais choisir simplement par la suite quelle mémoire utiliser.

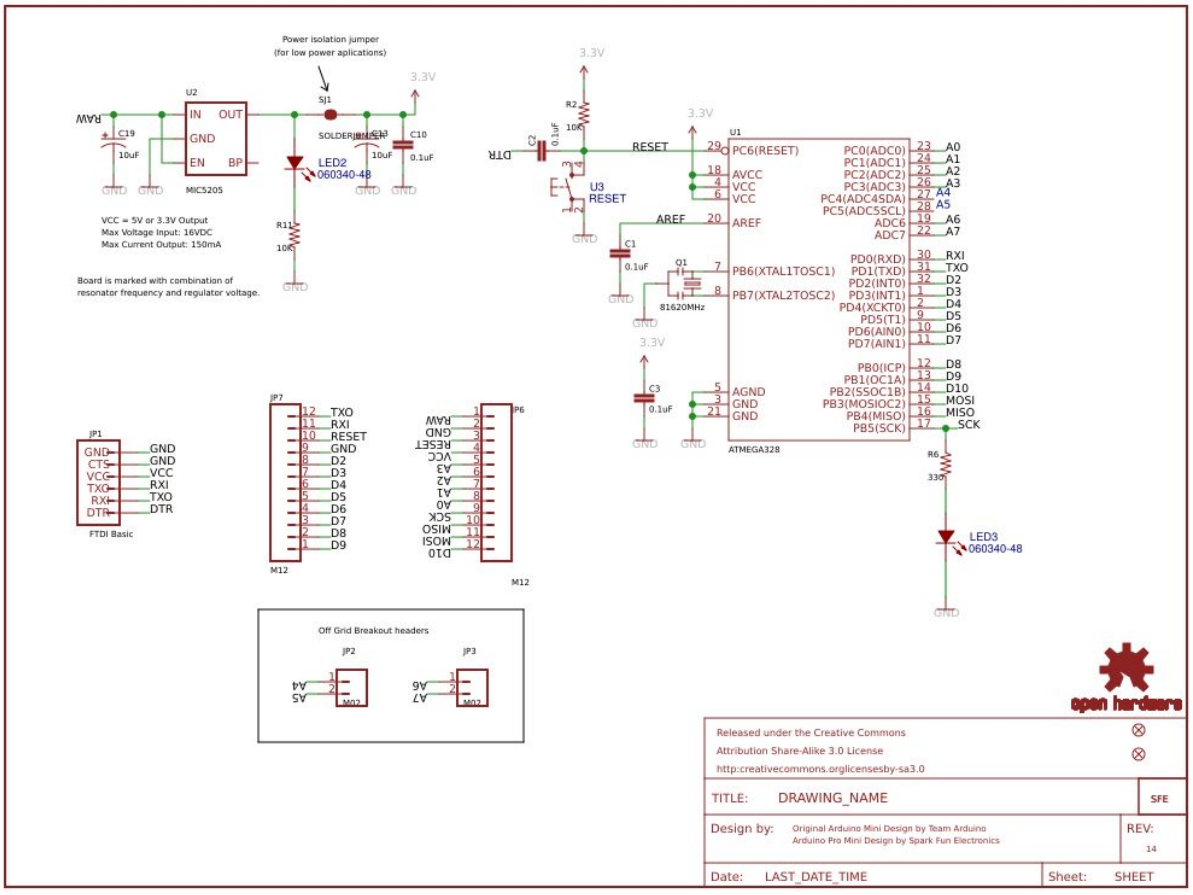
Pour la partie radio, j'ai choisi de me diriger vers un module NRF24L01 pour de nombreuses raisons. Ce module est très répandu et facile d'utilisation. Sa fréquence d'émission de 2.4GHz est utilisable librement en France et permet d'avoir une portée d'une

centaine de mètre en extérieur (50m en intérieur). Le fait que ce soit un module me permet de me consacrer sur d'autres parties et permettrait une maintenance plus rapide en cas de panne.



b) Réalisation du schématique

Le schématique de la carte est inspirée du schématique simple de l'arduino pro mini pour gérer la partie gestion de l'atmega328p. Cependant le composant MIC5205 n'a pas été ajouté. Il s'agit d'un contrôleur d'alimentation qui limite le courant à 150mA et permet une chute progressive du courant et de la tension ainsi qu'une protection anti retour (qui sert principalement pour les alimentations par batterie).



Pour permettre aux mémoires vives et au module NRF24L01 de fonctionner, il faut que leur tension d'alimentation soit de 3.3V maximum. Les signaux logiques les pilotant doivent également être de 3.3V. Ce qui signifie que l'Atmega328p doit impérativement fonctionner en 3.3V et donc nécessairement en 8MHz.

Une solution alternative aurait pu être d'utiliser un Shifter Leveler (convertisseur logique 3.3V) et ainsi pouvoir fonctionner en 5V 16MHz. Cependant cette solution est plus complexe, peut causer plus de bugs. Dans notre cas, nous n'avons pas besoin d'un fonctionnement en 5V 16MHz.

Un convertisseur linéaire LM1117MP-3.3/NOPB permettra d'alimenter ce capteur par une prise micro-usb car celui-ci convertit une tension de 5V en 3.3V.

Niveau connecteurs, des connecteurs ISP et FTDI permettront de téléverser le bootloader ainsi que des programmes.

Voir schématique (version 1.0) en annexe page 24

c) Réalisation du routage

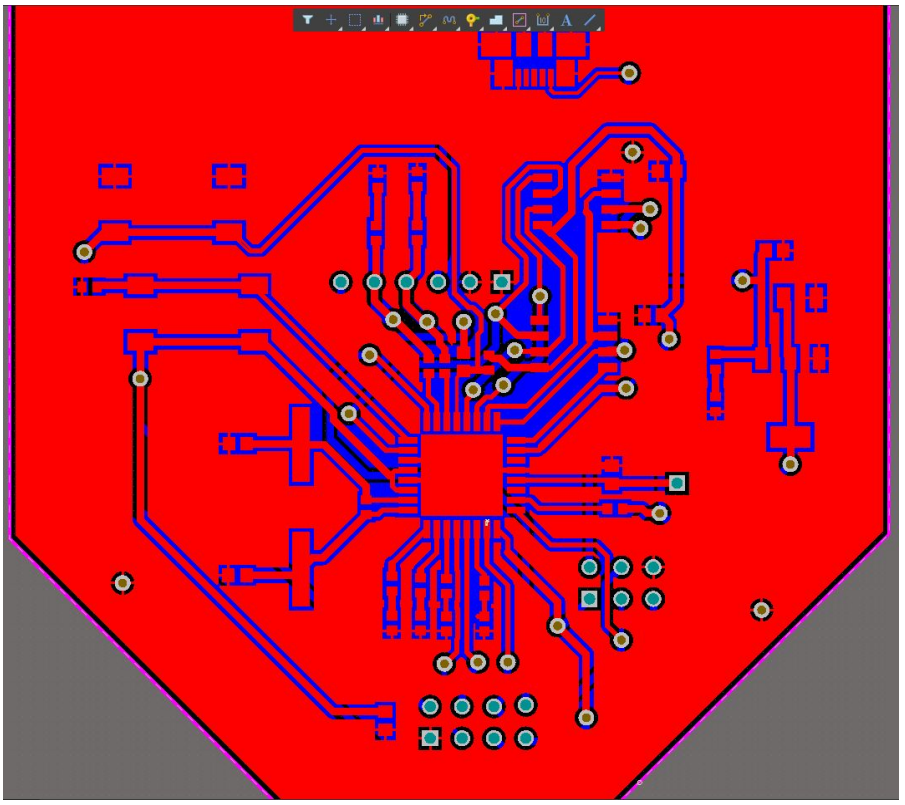
Cette partie a été plutôt fastidieuse et chronophage le circuit étant complexe non pas par le nombre de composant mais parce que très peu de pins peuvent être modifiés afin de permettre de faciliter le routage.

Premièrement les composants sont placés afin d'effectuer le moins de croisement possible. Les croisement sont corrigés si possible avec la modification des pins auxquels ceux ci sont connectés. Les nombreux headers imposent des emplacement précis et rapprochés ce qui ne facilite pas le routage.

Une fois les connexions établies entre les différents composants, j'ai demandé à mon tuteur M. Boé de vérifier mon routage. Parmis les remarques qu'il m'avait faites, une impliquait de refaire une bonne partie du montage. En effet, la réalisation de la carte à Polytech ne permet pas d'obtenir des trous métallisés, les connexions au niveau des headers situés sur la couche du dessus doivent obligatoirement se faire sur les couches du dessous. De plus les headers ne peuvent donc pas êtres utilisés en tant que via afin de passer de la couche du dessus à la couche du dessous.

La deuxième version du routage a été corrigée afin de raccourcir certaines pistes afin d'éviter les effets inductifs, en élargir d'autres afin de ne pas trop utiliser l'outil de précision, éviter les angles droits afin d'éviter les rayonnements… Limiter l'utilisation de vias car ceux ci doivent être soudés à la main.

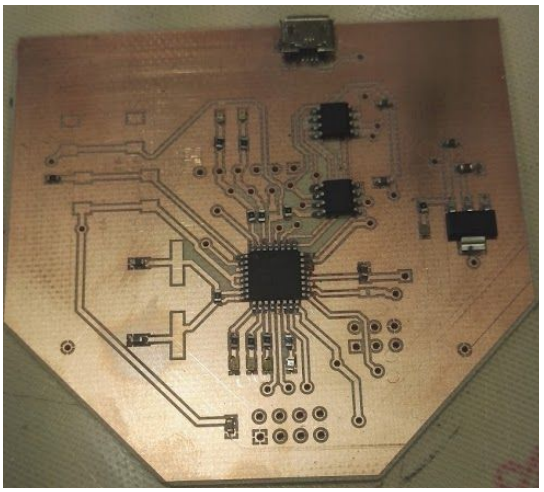
Ces différentes contraintes m'ont demandés beaucoup de temps, certainement plus que nécessaire mais ce temps n'étant pas perdu car permettra d'économiser du temps lors de la réalisation.



3. Réalisation et tests de la carte

a. Réalisation et tests de la carte V1

Pour cette première version, la carte a été réalisée à Polytech'Lille grâce à la graveuse et une plaque de cuivre. Celle ci travaille avec une incroyable précision et peut tracer des pistes très fines. Après une heure de gravure, les différents tests ont montrés qu'il y avait quelques court-circuits dus à des éclats de cuivre lors de la gravure. Après avoir effectuer des tests sur chacune piste afin d'éviter les court-circuits, les premiers composants a être soudés ont été l'atmega328 ainsi que les mémoires et le port micro-usb. Ces composants ont étés soudés grace au four.

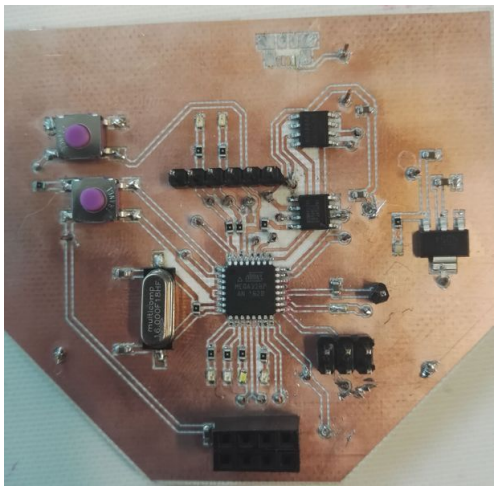


Une fois ces composants soudés, des tests grâce au multimètre en mode contacteur ont démontré que le port micro-usb était mal soudés. Un court-circuit était présent à l'arrière des pattes du composant. Dans l'impossibilité de supprimer ce court-circuit, j'ai décidé de supprimer cette prise qui n'était pas essentiel pour le début du projet.

Après avoir supprimé la prise usb et vérifié les soudures, j'ai décidé de souder une première partie des composants avant de faire des tests. Pour réaliser mes tests j'ai procédés de la manière suivante :

- Vérification au multimètre pour éviter les courts circuits
- Alimentation grâce à une alimentation de laboratoire avec une limitation de courant afin d'éviter de griller les composants
- Vérifications des tensions

J'ai choisi de souder tous les vias avant d'effectuer ces tests car, ceux-ci ne sont pas sources de problèmes tant qu'on vérifie qu'il n'y a aucun court-circuit entre les pistes. Ces vias me permettent d'utiliser le schématique en étant sûr que chacune des pistes est bien reliée sur les deux couches.

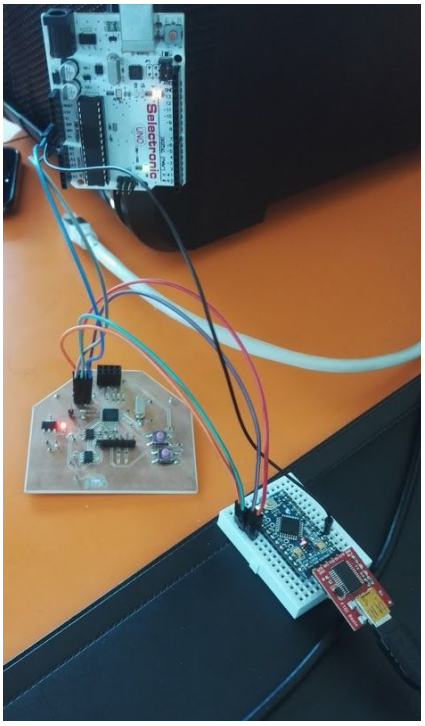


Une fois ces tests de soudure effectués, il a fallu tester logiciellement cette carte. Les premiers tests ne furent pas concluants, une première erreur fut de placer un quartz à 16MHz ainsi que des capacités qui sont en adéquation avec ce quartz. J'ai donc changer ce quartz afin d'en placer un de 8MHz avec des capacités de 22pF. L'erreur rencontrée était que le programmateur n'arrivait pas à dialoguer avec le microcontrôleur, celui-ci ne répondant pas, impossible d'obtenir sa signature ainsi que de graver un bootloader.

Après avoir fabriqué un programmateur ISP à l'aide d'une Arduino UNO, le téléversement du bootloader est, à nouveau, un échec. Le programmateur fournissait des signaux logiques de 5V alors que l'atmega fonctionnait sous 3.3V. Ce dysfonctionnement a été normalement corrigé avec un circuit de pont diviseur de tension.

Voulant vérifier cette théorie avec une autre méthode afin d'obtenir des signaux logiques en 3.3V, j'ai utilisé une arduino pro mini ainsi qu'un FTDI fonctionnant en 3.3V afin de programmer cette arduino en tant que programmeur ISP et ainsi pouvoir flasher le

bootloader grâce à un véritable programmateur ISP en 3.3V. Ce qui fut une nouvelle fois un échec.



Devant cet échec, j'ai demandé de l'aide à M. Redon qui m'a assister pour vérifier toutes les pistes ainsi que toutes les connexions. Après avoir effectués différentes modifications sans que le problème ne soit résolu, j'ai décidé de réaliser un shield qui permettrait de ne pas s'occuper de la partie microcontrôleur car cette dernière serait gérée par l'Arduino Pro Mini.

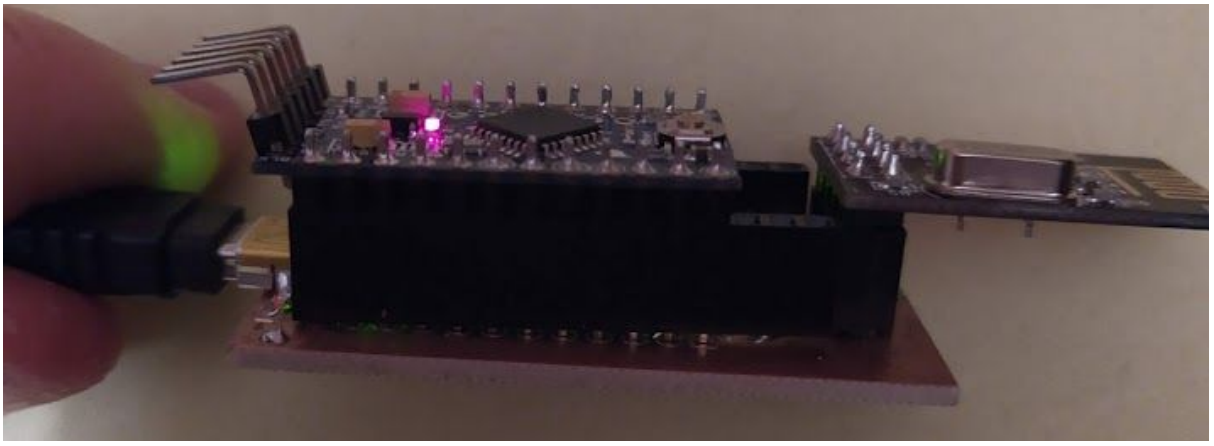
En parallèle de cela, M. Redon a fait la demande pour la fabrication de cartes chez un professionnel afin d'éviter les erreurs provenant de la réalisation.

b. Réalisation et tests du shield

Étant à la 11ème semaine de projet et approchant de l'échéance, j'ai décidé de concevoir et réaliser une nouvelle carte en attendant la nouvelle version commandée par M. Redon afin de pouvoir effectuer les premiers tests pour la programmation du bootloader.

Cette conception est très basique et est largement inspirée de la précédente carte. La conception et le routage a été très rapide (2 heures). J'ai pu procéder rapidement à la réalisation une fois la carte validée par M. Redon.

Pour la réalisation, tous les composants ont été soudés au fer avec de la pâte à braser et parfois l'utilisation de flux (pour les pattes du connecteur mini-usb).



Les tests pour cette carte étaient surtout effectués grâce à la programmation. La librairie RF24 a permis de démontrer qu'avec la raspberry pi en émetteur, il a été possible de communiquer. Pour les mémoires SPI, il était impossible de dialoguer avec elles. Un court circuit entre la piste HOLD et le GND empêchait les mémoires de communiquer. Une fois celui ci retiré les deux types de mémoires ont pu être utilisés.

c. Réalisation et tests de la carte V2

Cette carte a été réalisée avec le même schématique mais elle a été gravés par des professionnels et les pistes sont dissimulées derrière une couche isolante et seuls les plots sont visibles et sont en étain.

Cette carte a pour but premier de trouver d'où provient l'erreur commise par la première version. Il faut donc être très minutieux. Pour cela, j'ai d'abord souder les

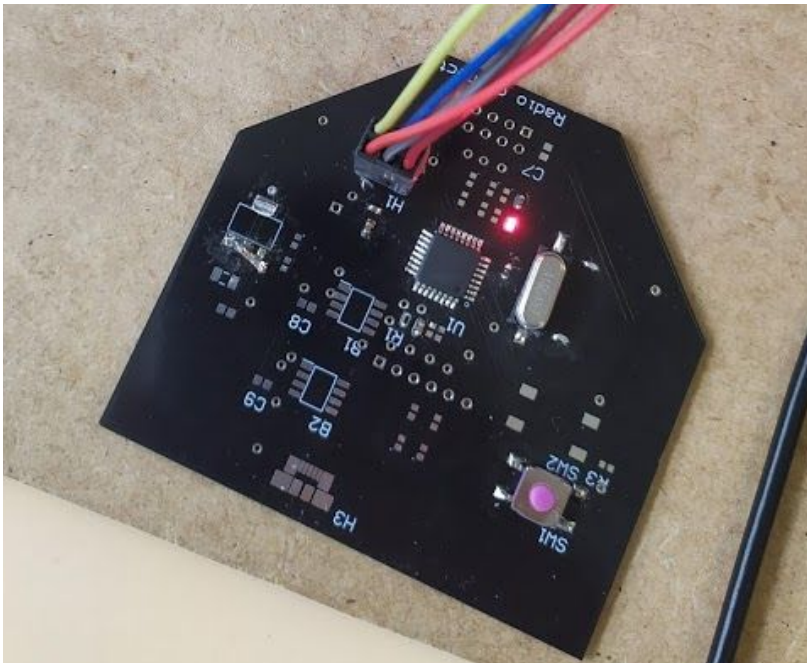
composants essentiels à la programmation du bootloader de l'Atmega328. Je décide donc de placer, en plus du microcontrôleur, les capacités de découplage et une led de test ainsi que l'oscillateur de 16 MHz afin d'essayer de le programmer sous 5V.

Le bootloader a bien été écrit et le composant fonctionne parfaitement. On observe bien la signature du composant qui n'est pas nulle cette fois-ci. Le programme de blink fonctionne également.

L'étape suivante était de passer sous 3.3V et pour cela il fallait ajouter le régulateur 3.3V ainsi que remplacer le quartz par un autre de 8GHz. Le blink précédemment installé ne fonctionne plus. J'essaie de graver le bootloader à l'aide d'un AVR Dragon (programmateur ISP qui fonctionne avec toutes les tensions de programmation). Il est toujours impossible d'installer le bootloader.

Le problème a donc été identifié, il s'agit de l'utilisation du quartz en 3.3V. L'arduino pro mini, elle utilise un système de résonateur qui lui permet de fonctionner en 20MHz, 16 MHz ou 8 MHz. Fort heureusement, l'utilisation d'un quartz pour un Atmega n'est pas obligatoire. Celui ci est composé d'une horloge interne qui fonctionnera à 8MHz mais n'est pas aussi précise qu'un quartz. Cependant, le but étant de faire valider notre hypothèse, je décide de passer sous horloge interne.

Pour passer en fonctionnement avec l'horloge interne, il faut modifier les fuse (fusibles électriquement programmables) afin de lui indiquer qu'il doit fonctionner avec une horloge interne. Pour pouvoir modifier ces fusibles il faut pouvoir communiquer avec lui, je dois donc repasser en 5V (retirer le régulateur de tension et réaliser un court-circuit entre les broches de 3.V et 5V) avec un quartz de 8 MHz.



A l'aide d'un calculateur de fusible trouvé sur internet je choisis les différentes valeurs afin de permettre un fonctionnement à l'aide de la clock interne de 8 MHz. Il faut, pour cela, modifier les low fuses. Je choisis d'avoir un CK de +65ms, il s'agit du boottime maximum proposé par ce composant. Ce temps est prévu afin de laisser l'oscillateur osciller parfaitement. avant de démarrer.

Fuses		
	Avant	Après
Low	FF	E2
High	DA	D8
Extend	05	07

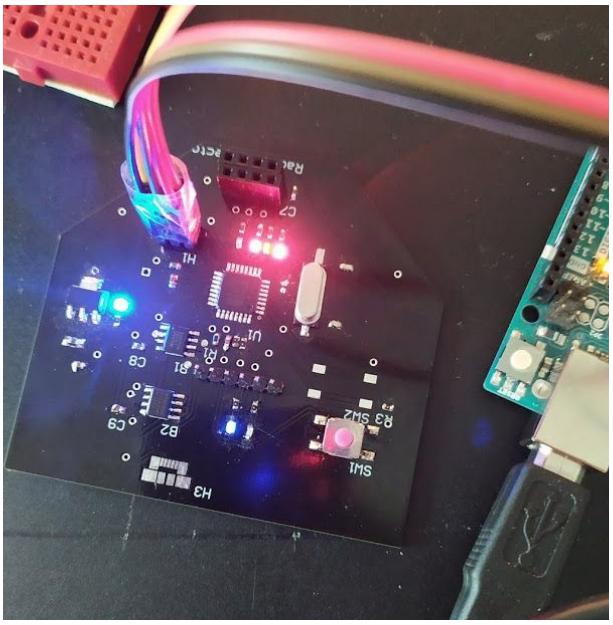
Les High fuses me permettent d'utiliser un bootloader de 2K (valeur maximum admissible par ce composant) et les Extend fuses me permettent de désactiver la détection de brown-out. Cette détection permet d'éviter que le composant soit sous alimenté et réagisse de manière anormale. Pour cela, quand la valeur de la tension chute en dessous d'un seuil, le composant se met en mode reset. Cette détection étant source d'erreur je l'ai désactivée.

Une fois ces fusibles modifiés, je retire l'oscillateur afin de constater que le composant fonctionne toujours et le blink n'a pas changé de fréquence. Je décide donc de changer la tension afin d'observer que le composant fonctionne mais que la fréquence de clignotement de la led est 2 fois plus faible due au passage à 8 MHz.

J'ai essayé de changer certains fusibles afin de vérifier si je pouvais activer la protection Brown-out mais les extend-fuses ne peuvent être programmés en 3.3V et ceci cause le dysfonctionnement total du microcontrôleur. Je prends donc la précaution de ne plus utiliser Arduino IDE et je décide d'utiliser avrdude afin d'être sûr de ne pas changer la valeur des fuses lors d'un téléversement.

Une fois cette précaution prise, je décide de graver tous les autres composant et j'effectue les tests précédents sur la nouvelle carte.

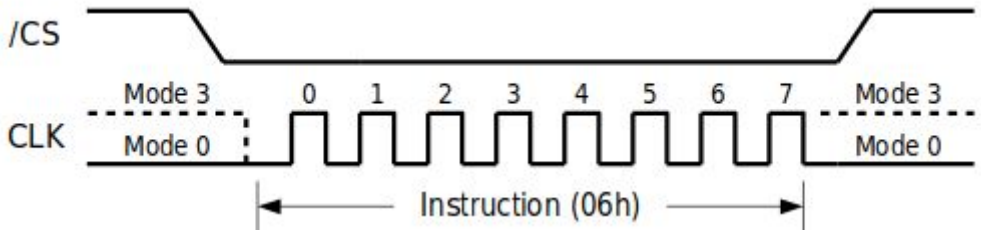
Le module radio fonctionne parfaitement. Les mémoires ne répondent pas. Un défaut de conception en est la cause. Ces mémoires SPI n'étaient pas connectés au bus SPI du microcontrôleur mais étaient reliées à des broches classiques. Les pistes sont donc coupées et des fils sont soudés afin de relier ces mémoires au bus SPI. Après les modifications effectuée, les mémoires sont fonctionnelles.



4. Le bootloader

a. Communication avec les mémoires

La communication avec les mémoires s'effectue via une communication SPI, le fonctionnement de cette communication est très simple. La mémoire est sélectionnée via sa broche CS qui passe à l'état bas puis, la mémoire reçoit un code de commande qui va dire ce que celle ci doit faire, des bytes peuvent suivre ou non cette commande et enfin la broche CS passe à l'état haut pour dire que la commande est terminée.



Un tableau sur les datasheets des composants permettent de connaître ces commandes. Heureusement, ces composants utilisent les mêmes codes d'instructions pour les commandes que je désire utiliser. Parmis ces commandes, on peut retrouver :

- **STATUSREAD** qui permet de savoir si la mémoire est prête à recevoir d'autres informations.
- **WRITEENABLE** qui permet d'activer l'écriture.

- **ARRAYREADLOWFREQ** qui permet de demander de lire les bytes présents à l'adresse indiqués.
- **BYTEPAGEPROGRAM** qui permet d'écrire dans la page pointée par l'adresse données. Cette page peut accueillir 256 octets. Il faut que l'écriture soit activée afin d'utiliser cette fonction.
- **JEDECID** qui permet d'afficher le JEDEC du composant. Ce JEDEC permet de savoir qui est le fabricant et permet surtout de savoir si la mémoire fonctionne. (0x00 n'étant pas un JEDEC valide)

Grâce à ces commandes j'ai développé (en m'inspirant de la librairie SPIFlash codée en C++) les fonctions suivantes :

- **SPI_transfer** qui transfère ou lie des données sur le bus SPI (récupérée sur la librairie).
- **FLASH_busy** qui indique si la flash est occupée.
- **FLASH_command** qui exécute la commande donnée en paramètre en activant l'écriture si cela est spécifié en paramètre.
- **FLASH_writeBytes** qui permet d'écrire à l'adresse donnée un buffer de taille variable.
- **FLASH_readByte(s)** qui permettent de lire le(s) octet(s) présent à l'adresse indiquée.

Ces fonctions ont étés testées avec la lecture et l'écriture de valeurs dans les mémoires.

Pour initialiser les mémoires une fonction InitFlash permet de définir les entrées sorties, configurer la communication SPI, et mettre en état haut les pins HOLD et WP.

b. Communication avec le module radio

La communication avec le module radio était beaucoup plus complexe et chronophage. Le module NRF24L01 fonctionne à partir de registres. Ces registres permettent d'indiquer tous les paramètres pour effectuer la communication. Il y a un nombre assez conséquent de registre à configurer. C'est pour cela que je ne détaillerais pas chacune des valeurs dans le rapport mais ces valeurs peuvent être trouvée dans la fonction InitRF() dans le code source du bootloader.

Le principe est assez similaire que pour les mémoires SPI, le composant reçoit une instruction et les informations après que le CS soit passé à bas et comprend la fin de la commande lorsque le CS est passé à haut.

Pour utiliser ces différents registres, j'ai retranscris les fonction présentes dans la librairie C++ RF24 en les modifiant pour que celles ci fonctionnent en parallèles avec les fonctions des mémoires. Notamment :

- **NRFWriteRegister** qui permet d'écrire dans un registre la valeur passée en paramètre.
- **NRFReadRegister** qui permet de lire le registre indiqué.
- **NRFReadBuffer** qui permet de lire un registre qui retourne plusieurs octets (comme, par exemple, l'adresse)
- **NRFWriteBuffer** qui permet d'écrire dans un registre qui demande plusieurs octets (comme l'adresse). Cette fonction vérifie que le registre a bien été écrit et exécute la commande à nouveau dans le cas contraire.
- **NRFMessage** qui permet de savoir si un message est disponible.
- **NRFGetMessage** qui permet de lire un message de n octets et vide la pile FIFO.

Comme indiqué précédemment, les messages reçus sont stockés (sans intervention de l'Atmega) dans une pile FIFO qui peut contenir jusqu'à 4 messages. Une fois qu'un message est stocké, il faut que celui ci soit lue par l'atmega pour qu'il soit supprimé de la pile.

Parmis les configurations du registres on peut noter qu'il y a le channel de communication (définissant la fréquence de communication) (ici 76), l'adresse de communication, le débit de communication (ici 1Mb), la configuration de la taille du CRC (checksum). Le nombre d'essai d'envoi de paquet lorsque celui n'est pas bien reçu…

c. Fonctions supplémentaires

Les fonctions supplémentaires permettent de se donner une idée du déroulement d'une mise à jour. Premièrement, une entête est reçue, celle-ci indique la taille, l'identifiant entreprise, l'identifiant groupe, la version et un code CRC. Cet entête est vérifiée et les informations importantes sont stockées. Si l'identifiant et le groupe corresponde au capteur,

celui-ci va recevoir la mise-à-jour. Une fois la mise à jour reçue il suffit de la copier dans la mémoire de programme.

A chaque reset ou démarrage, la fonction CheckOTAUpdate est appelée. Cette fonction va initialiser la mémoire et le module radio puis va écouter pendant une temporisation donnée (10 secondes par défaut). Si un message est reçu pendant ce délai, on vérifie que celui ci soit conforme (grâce à la fonction DecodeTrame) si cette dernière retourne la valeur 1, la réception de la mise à jour peut commencer (ReceptionOTA). La fonction Update permet de mettre à jour le firmware. Ici cette fonction ne permettra seulement que de recevoir correctement la mise à jour et l'indique grâce à un signal lumineux pendant quelques secondes avant de continuer le démarrage.

5. Conclusions et performances

a) Partie codage

La vérification par CRC ainsi que le renvoi de paquet peut poser problème en entreprise lorsqu'il y a beaucoup de capteur qui désirent se mettre à jour et qu'il y a une mauvaise communication. Je préconiserais, dans ce cas de réduire le nombre d'essaie et si un paquet est mal reçu par un capteur, celui ci ne se mettra simplement pas à jour. Cette détection peut se faire par la taille de la mise à jour qui sera donc erronée car il manquera un ou plusieurs buffer. Le CRC global sera également erroné.

Pour des raisons de mémoire, la fonction Update réceptionne la mise à jour et la stocke dans la mémoire mais il est impossible d'écrire la suite qui permettrait d'appliquer cette mise à jour dans la mémoire de programme (grâce à la fonction writebuffer). Le nombre de ligne nécessaire reste important.

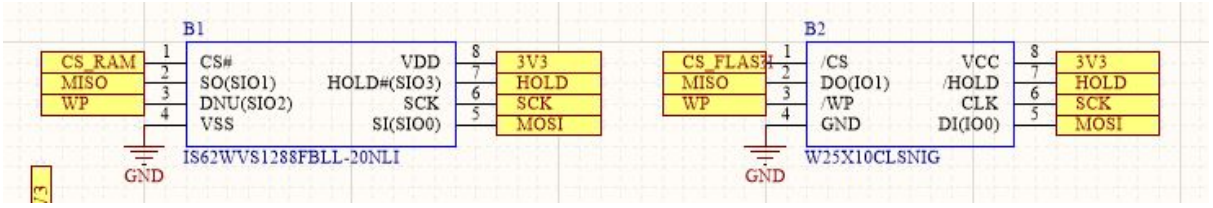
La taille utilisée par le bootloader est la taille maximale admissible par ce composant (Atmega328p). Il faudrait utiliser un composant disposant d'une mémoire réservée au bootloader plus grande comme l'atmega2560 qui dispose d'un bootloader de 8ko contrairement aux 2 ko disponible sur l'Atmega328p.

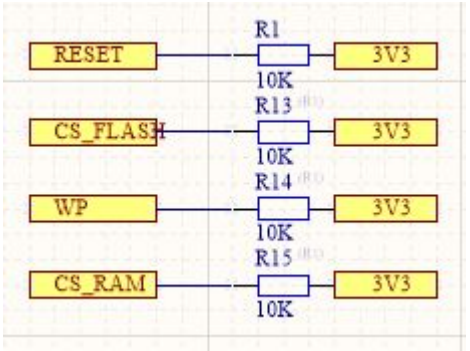
J’ai pris soin de désactiver les fonctions inutiles dans notre cas, (les fonctions présente lorsque la condition BigBoot est définie) afin d’économiser l'espace mémoire.

Pour palier à ce besoin d’espace j’aurais pu également coder dans la mémoire de programme en prenant le risque qu’il y ait des conflits d’espace.

b) Partie conception

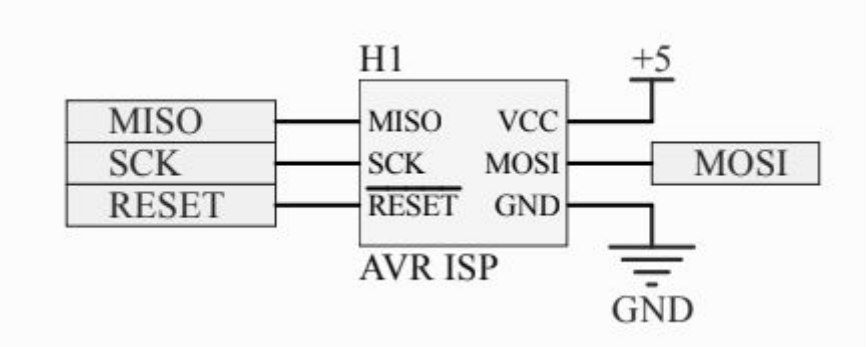
Des erreurs de conceptions auraient pu être évitées lors de la conception notamment avec des résistances de pull-ups présentent sur le shield mais pas sur la carte principale et également des mémoires RAM reliées au bus SPI par défaut. Ce qui aurait évité les soudures et suppression de liaisons au cutter sur le prototype.





Un FTDI aurait pu être ajouté afin de faciliter la programmation du prototype directement par USB. Bien-sûr, celui-ci n'aurait pas du tout été ajouté sur une version de production car la programmation par USB n'est pas fréquente dans le cadre de capteurs qui peuvent être mis-à-jour over the air.

La partie ISP est mal conçue et ne permet pas d'alimenter la carte pendant la programmation, il faut nécessairement que le capteur soit alimenté d'une autre manière. Il suffit de changer la connexion du VCC avec le bus 3.3V du régulateur.



Un problème de perturbation électromagnétique est présent au niveau de la carte, le capteur NRF24L01 se situe au dessus du microcontrôleur ce qui l'empêche de recevoir correctement, il aurait fallu qu'il se situe dans l'autre sens vers l'extérieur de la carte.

Et enfin, un résonateur comme celui présent sur l'arduino pro mini permettrait d'avoir des oscillations plus précises et donc d'avoir un microcontrôleur plus fiable

c) Partie architecture du projet

Une solution intéressante serait d'utiliser un composant seul qui va gérer la mise à jour (comme je le pensais initialement). Les avantages sont divers :

- Utilisation possible de tout l'espace de programme
- Possibilité de recevoir la mise à jour pendant le fonctionnement du capteur sans aucune interruption et mise à jour possible lors du reboot (grâce à dualoptiboot)
- Consommation maîtrisable par une interruption lorsqu'un paquet est reçue

Le seul véritable inconvénient est le prix qu'engendrerait ce composant supplémentaire.

6. Remerciements

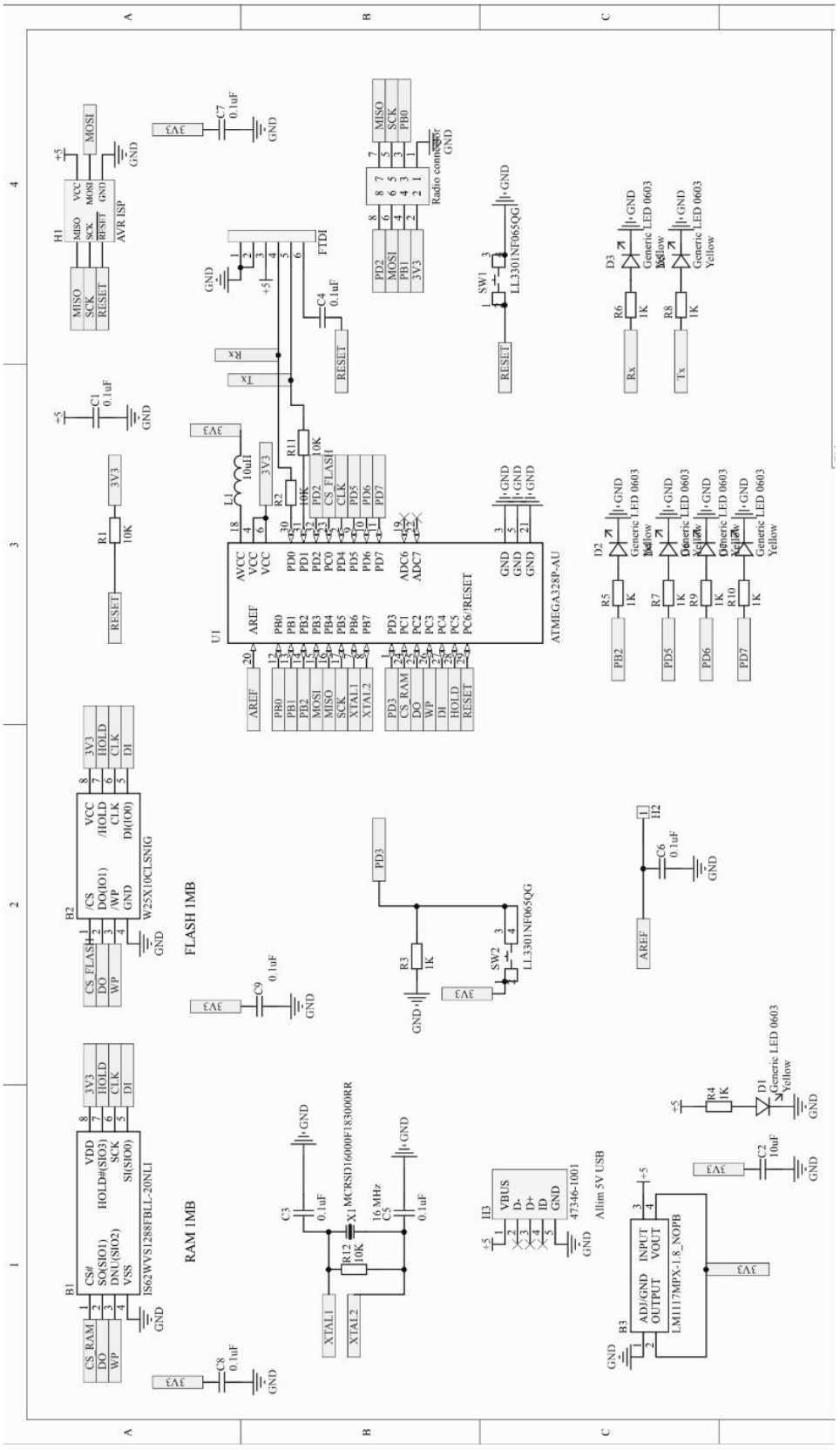
Je tenais tout d'abord à remercier certains camarades de classe comme Théau MOINAT qui m'a conseillé pendant la conception de carte. Je remercie également M. Flamen pour son temps et ses précieux conseils.

Merci également à M. Boé pour avoir vérifié ma carte à de nombreuses reprises et pour m'avoir corrigé et aussi conseillé. Je remercie M. Vantroys pour ses conseils avisés lors de mes recherches sur la programmation du bootloader.

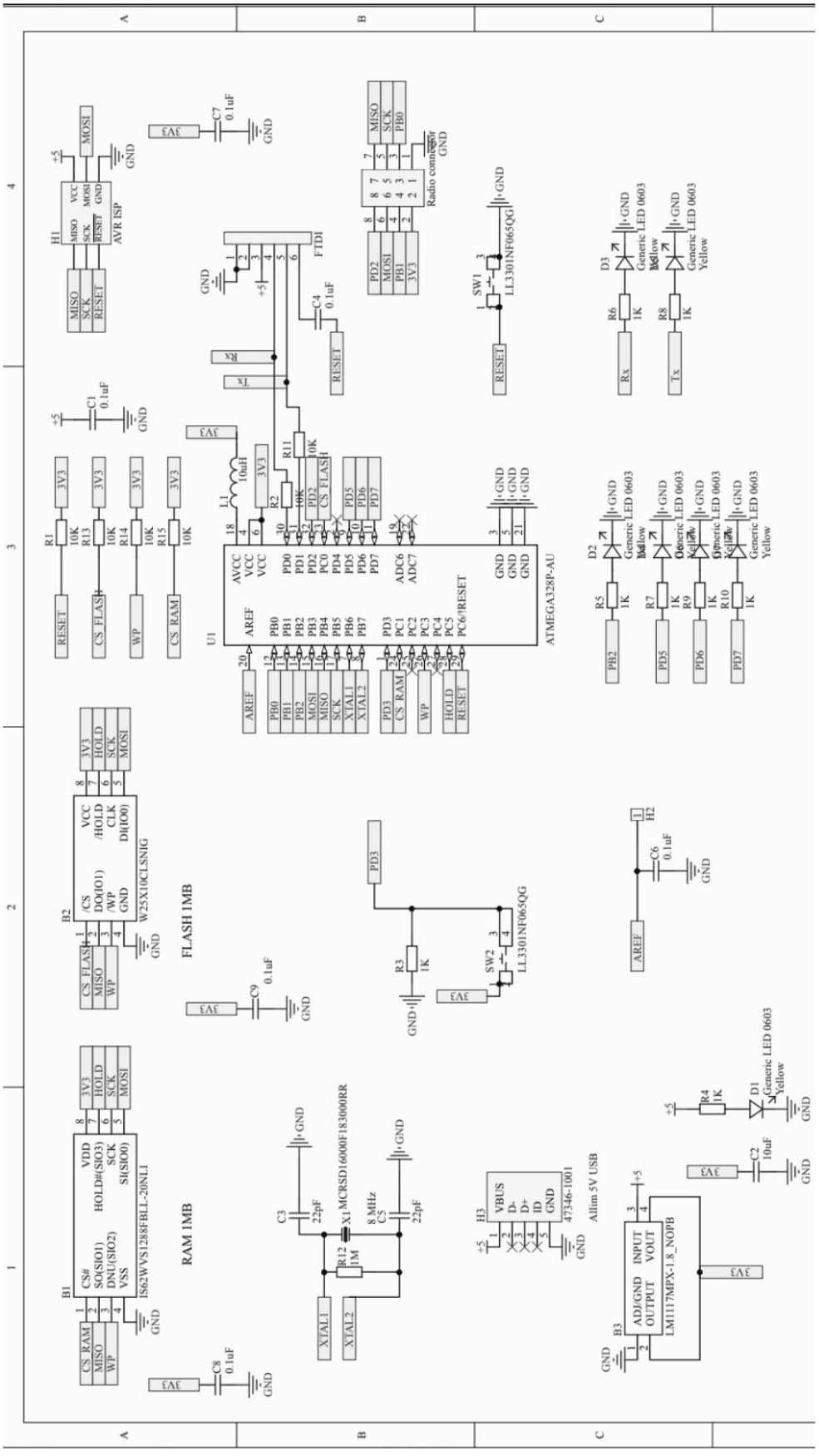
Je souhaite remercier plus particulièrement M. Redon, qui m'a aidé en ayant réalisé une partie d'un double de ma carte afin de me prouver que tout n'était pas mauvais sur ma conception. Je le remercie également pour les nombreuses heures qu'il m'a consacré et pour la confiance qu'il m'a accordé.

Annexes

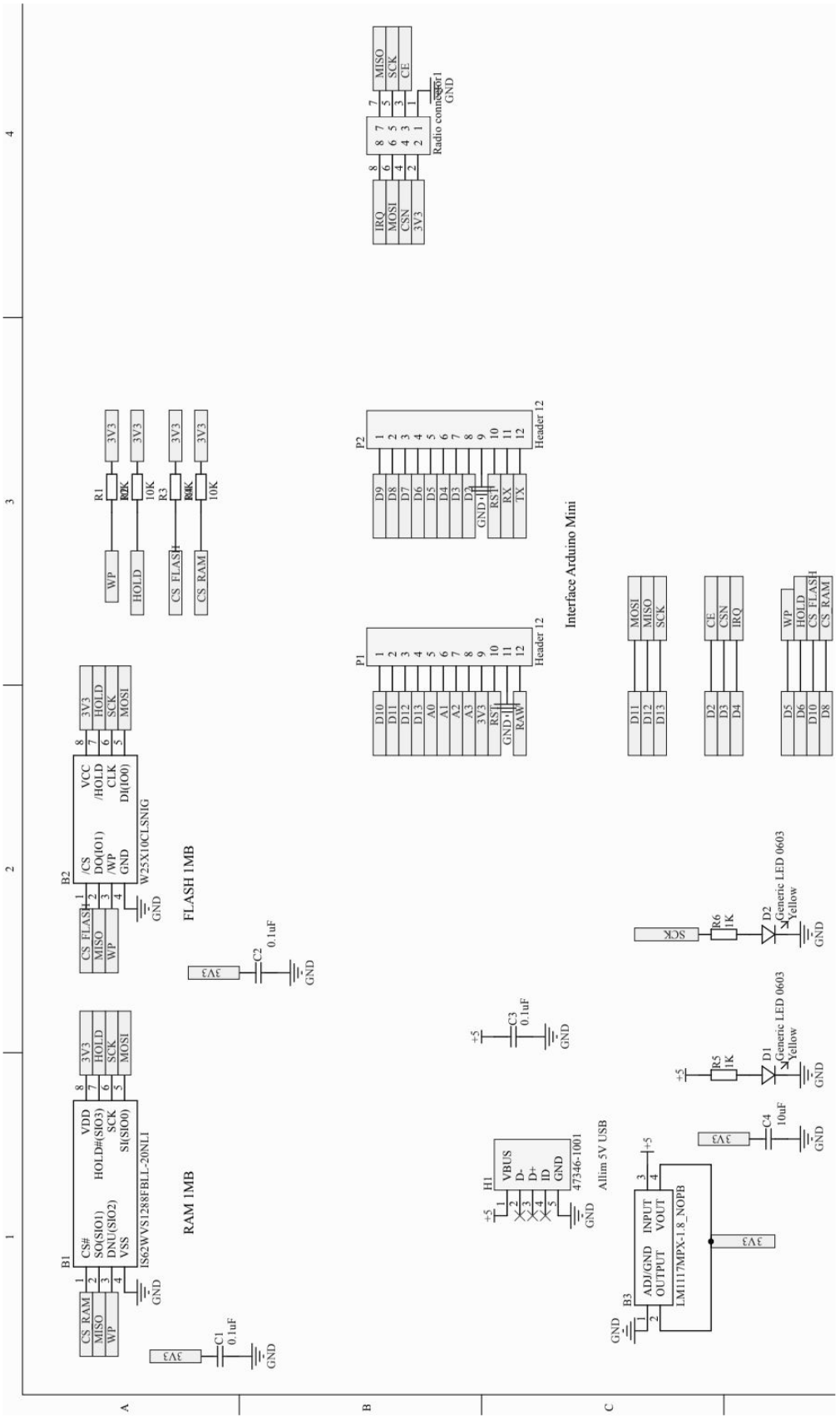
Prototype Carte V1.0



Prototype Carte V2.0



Shield


Interface Arduino Mini
FLASH 1MB
RAM 1MB
Header 12
Radio connector1
Generic LED 0603
Yellow
Alim 5V USB
W25X10CLSNIG
IS62WVS1288FBLL-20NLI
LM1117MPX-1.8_NOPB
47346-1001