Modem pour le numérique

Jérôme VAESSEN

14 avril 2014

Table des matières

Chapitre 1

Modulateur et démodulateur pour communications numériques

1.1 Cahier des charges

1.1.1 Présentation générale du projet

Dans le cadre des TPs d'électronique et de transmission RF, nous avons découvert qu'il existe plusieurs modes de modulation, pour transmettre un signal modulant.

Celles vues en TP au S7 sont la modulation en amplitude, en fréquence, en phase, en BPSK et en QPSK.

Or les modulations en phase, en fréquence et en amplitude étant celles les plus connues, la modulation en QAM (Modulation d'Amplitude en Quadrature) est quant à elle moins connue, mais tout aussi utilisées :

-pour la TNT en 64 QAM (en France).

-pour le WiFi en 64-QAM ou 256-QAM (à vérifier).

Or ces deux dernières étant peu représentées au niveau des travaux pratiques, ce projet permettra de fournir un peu plus de pratique sur les modulations de type QAM (en diagramme I/Q) dans le domaine des radiofréquences (800MHz).

(Le nombre devant le QAM indique le nombre de points dans le diagramme I/Q qu'on appelle aussi constellation)

On se réserve le droit de faire "évoluer" le cahier des charges a posteriori, si certaines contraintes étaient amenées à évoluer ou se préciser.

1.1.2 Objectif et intérêt du projet

Pour cela, on nous a fourni des modules commerciaux Mini-circuit pouvant travailler dans les radios-fréquences.

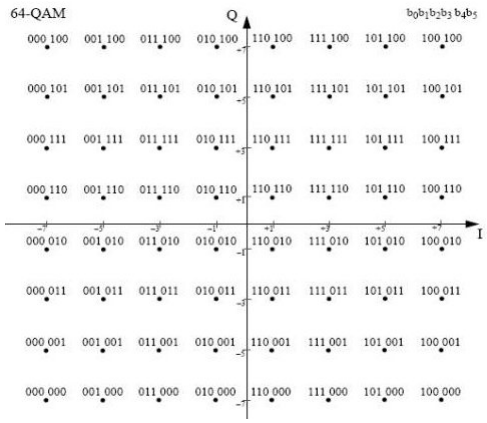

FIGURE 1.1 – Exemple de constellation 64QAM.

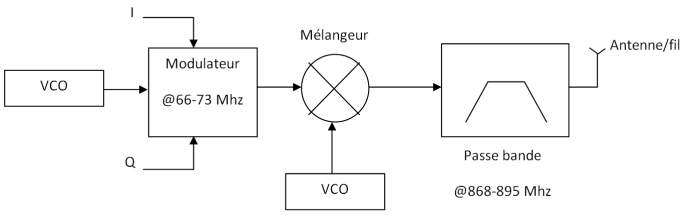

FIGURE 1.2 – Modulation.

Ceux-ci permettront de mettre en place une grande partie de la chaine de transmission (voir schéma de la chaine de transmission).

On a tout d'abord à mettre en place une carte FPGA et/ou un dispositif permettant d'envoyer un flux de données numériques continuellement pour pouvoir visualiser le diagramme I/Q (mise en valeur du diagramme I/Q et voir même un diagramme un peu plus complexe que le BPSK ou le QPSK).

Le canal de transmission qui sera un simple fil électrique, mais pourra devenir une antenne réceptrice et émettrice si le temps imparti le permet (mise en valeur du et mesure du Bit Error Rate qui sera bien sûr différent suivant le canal de transmission).

Du côté réception ,après démodulation du signal, une remise en forme et une interprétation des données seront réalisées, celle-ci dépendra bien entendu de la provenance du flux de données (constater le succès ou l'échec d'un envoi de données).

Enfin si le temps le permets on pourra mettre en une place un système de codage de l'information avec par exemple un codage redondant permettant de corrigé l'erreur (À définir, on pourra faire constater l'efficacité d'une méthode

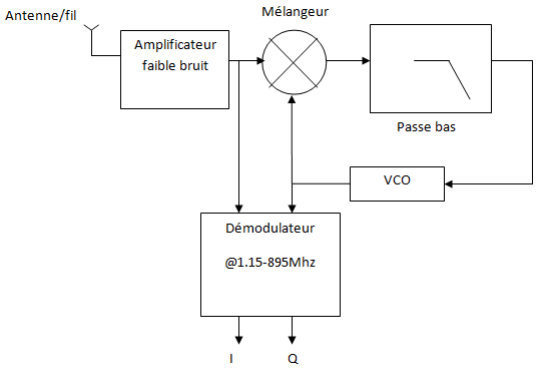

FIGURE 1.3 – Démodulation.

de codage au choix).

1.1.3 Étapes/Déroulement du projet

De même que pour le cahier des charges, il se peut que les étapes diffèrent plus ou moins, en fonction des disponibilités, matérielles et contraintes.

Caractérisation des blocs Mini-circuits et épluchage des documentations techniques

Avant de les utiliser dans la chaine de transmission, nous avons à faire une caractérisation des blocs fournis :

-la caractérisation du VCO.

-l'étude des possibilités offerte par le modulateur (cela déterminera le nombre de points possible dans la constellation, ie savoir si l'on peut aller au-delà de la QPSK).

-la caractérisation des mélangeurs (Mixer), leurs facteurs réels de multiplication.

Utilisation et exploitation des caractérisations

On pourra ensuite trouver les paramètres pour régler correctement dans notre chaine de transmission certains de nos sous-systèmes comme la boucle à verrouillage de phase.

Génération d'un flux des données numériques

Comme dit précédemment, l'idéal et d'avoir un flux de données numériques transmis en permanence. Pour cela plusieurs solutions s'offre à nous, on pourra

utiliser un montage générateur pseudo aléatoire (un circuit que l'on a fait l'année dernière en Conception de Circuit Électronique). On pourra "émuler" ce montage avec un Arduino, ou bien même le faire au moyen de la Nanoboard. On pourra aussi transmettre des chaines de caractère en continu.

Test de la chaine

Une fois les étapes précédentes réalisées, on pourra passer au test de la chaine de transmission. Il faudra vérifier le bon fonctionnement en aval, dedans (boucle à verrouillage de phase), et en amont de la chaine de transmission.

Première mesure du Bit Error Rate

Il sera temps de mesurer/déterminer le B.E.R. de la chaine de transmission avec un fil électrique en tant que canal de transmission. Il est possible que la conception d'un dispositif soit nécessaire que ce soit pour les deux mesures du B.E.R..

Antennes

Si toutes les étapes précédentes sont satisfaites, on pourra alors faire un canal de transmission plus intéressant, à savoir l'air. Nous passerons alors à la conception des antennes, émettrice et réceptrice.

Deuxième mesure du Bit Error Rate

Il sera temps de mesurer/déterminer le B.E.R. de la chaine de transmission avec un fil électrique en tant que canal de transmission.

Codage de l'information et correction d'erreur

En toute fin de ce projet, on pourra même mettre deux dispositifs en aval et en amont de la chaine de transmission. On réalisera alors un codeur et un décodeur permettant de faire de la détection/correction d'erreur.

Mise en place d'outil de débogage (tout au long)

La mise en place de simples outils de débogage et primordiale, ceux-ci permettront de vérifier le bon fonctionnement de la chaine.

Proposition d'un sujet de TP (tout au long)

Il se trouve que lors du déroulement du projet, on pourra être amené à se poser des questions au niveau des réglages de la chaine de transmission. Or il est fort probable que les élèves qui travailleront sur ce TP pourront alors se poser les mêmes questions. C'est pour cela qui sera judicieux de prendre soin de noter les problématiques rencontrées et de donner un certain fil de développement

pour venir à bout de la problématique posée. Ce qui reviendra à proposer des axes/questions sur les manipulations de cette maquette de TP.

1.2 Note valable pour toutes les semaines

Si un point vous semble imprécis, veuillez vous reporter aux détails fournis en annexe.

1.3 semaine 1 : du 27/01/2014 au 01/02/2014

Séance du 05/02/2014 :

Pour bien comprendre les différentes choses à réaliser, on a refait les manipulations vues en TP. On a, pour cela utiliser la maquette didactique TIMS pour générer différent type de modulation :

-la modulation en 4,8,16PSK (on obtient une constellation en forme de cercle).

-La modulation en 4,8,16QAM (on obtient une constellation avec un carré, ou deux : de différentes tailles).

On a très vite dégagé la nécessité de réaliser un montage générateur de tous les mots numériques possibles. On a donc fait un programme sur arduino qui équivaut au générateur pseudo aléatoire. Celui-ci est directement inspiré du montage que l'on a réalisé l'année dernière en conception de circuit électronique.

Séance du 06/02/2014 :

Lors de cette séance on a exploré toutes les possibilités concernant la communication entre la maquette/carte, car cela pour avoir une influence non négligeable sur le choix de la carte F.P.G.A. et du design de la maquette.

On a donc aussi fait des recherches concernant les possibilités de communications et les blocs I.P. (Intellectual Properties) permettant de programmer un F.P.G.A.

Et enfin, on a eu une discussion avec un responsable technique pour ce qui de la C.E.M. (Compatibilité Electro-Magnétique)

1.4 semaine 2 : du 03/02/2014 au 08/02/2014

Réflexion sur les possibilités possibles et offertes par le sujet pour ce qui est des parties de communications entre la maquette en interne et de son environnement.

N'ayant jamais utilisé d'autre carte que la Nanoboard, on s'est attelé à faire des recherches sur la manière de procéder pour programmer une carte F.P.G.A., il se trouve que chaque carte de développement à ses propres manières pour être programmé.

On a pris le soin de noter toutes les questions concernant le F.P.G.A. pour un intervenant industriel qui assura un cours la semaine prochaine.

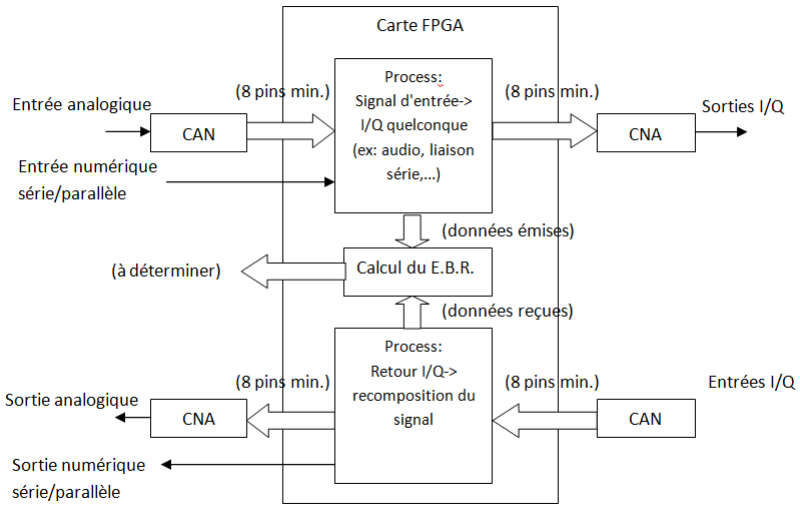

FIGURE 1.4 – Maquette en interne/externe.

1.5 semaine 3 : du 10/02/2014 au 15/02/2014

Ayant les oscillateurs commandables en tension, on a pu commencer à faire une manipulation avec les composants Mini-Circuit.

Mais après un problème suite à une manipulation l'un des VCOs ne fonctionne plus.

Ce qui nous amène à penser qu'il sera judicieux de prévoir une alimentation au niveau de la carte.

Celle-ci devra être inférieure ou égale à la tension maximale d'alimentation du composant (ceux-ci étant très sensible).

Questions/Réponses :

Comment fait-on pour programmer une carte F.P.G.A. est-ce que la programmation se fait "in situe" ? (in situe : c'est à dire sur la carte elle-même) Oui, c'est ce qui est utilisé, via un câble JTAG.

Est-ce possible faire une interface USB pour permettre à un ordinateur de communiquer avec la maquette/carte via un F.P.G.A. ? Oui.

Est-ce que la gestion de l'interface Ethernet est envisageable ? Oui.

Pour une communication en protocole UDP avec un ordinateur via une liaison Ethernet, est-ce qu'il y a d'autres façons que d'utiliser un "SoftCore" ?

Oui, il existe aussi bien en Software, qu'en Hardware.

1.6 semaine 4 : du 17/02/2014 au 22/02/2014

On a donc repris l'idée de la 2éme semaine.

Après avoir mis en place un cahier des charges au niveau des qualités, bande passante minimale.

Nous avons fait des recherches sur les composants à utiliser (i.e. CAN, CAN, ...)

Ce qui nous a permis après une édition rapide de schéma électrique (schematic) d'avoir une idée du nombre de pins dont nous aurons besoin sur une carte FPGA afin de réaliser le projet.

Après prospection, il se trouve que nous nous dirigerions vers la carte Digilent Spartan 3, dont l'école était déjà en possession.

Donc nous nous sommes attelés à faire des recherches et des essais sur cette carte et pour notamment trouver un moyen de la faire fonctionner.

1.7 semaine 5 : du 03/03/2014 au 08/03/2014

Tout d'abord, après plusieurs heures de recherches et d'essai infructueux, on a fini par trouver les drivers pour le câble USB diligent.

Et donc ainsi avoir un fonctionnement de la communication via le câble JTAG plus propice au fonctionnement de la carte.

On a pu enfin se conforter sur une partie des composants et des dimensionnements pour certains d'entre eux (valeur de résistance,capacité...)

Schématics : Édition de multiple bibliothèque pour Altium, intégrant à la fois schématics et leur empreinte PCB associée.

(On a pu constater qu'une interface 3v3 5v n'est pas forcement nécessaire pour assurer le bon fonctionnement entre deux cartes avec des tensions d'alimentation différentes.)

1.8 semaine 6 : du 10/03/2014 au 15/03/2014

On a pu enfin se conforter sur l'ensemble des composants et des choix de régulateur et utilisation d'autre diode Zener.

On trouvera le schéma intégral avec les bibliothèques nécessaires dans la semaine 5.

Cette semaine a été dédiée à l'envoi de la commande des composants au sein du magasin de l'école et du fournisseur.

Après envoi de la commande, il se trouve que le projet a changé d'objectif principal.

Il est maintenant prioritaire de réaliser un code VHDL capable d'évaluer la vitesse de transmission numérique du côté récepteur.

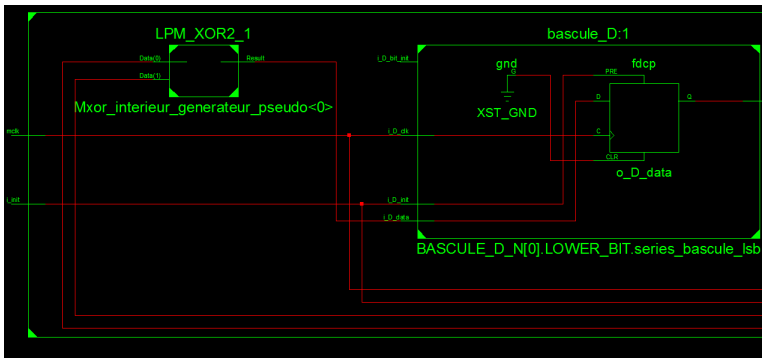

FIGURE 1.5 – extrait du schéma RTL, on peut y voir clairement les inférences de Xilinx.

1.9 semaine 7 : du 17/03/2014 au 22/03/2014

Pour cette semaine, on s'est concentré sur la réalisation d'un générateur pseudo aléatoire en interne au FPGA.

On réalisera une échelle de résistance afin de réaliser une conversion numérique analogique afin de l'envoyer sur la chaine de transmission.

Ce code de "base" nous permettra de réaliser un flux de données fictif pour obtenir toutes les combinaisons récupérant le résultat du bus.

Ce schéma a deux entrées et une sortie, une entrée d'initialisation, une horloge et une sortie pour la trame en pseudo aléatoire.

On fera l'essai sur la carte lors de la semaine suivante.

1.10 semaine 8 : du 24/03/2014 au 26/03/2014

Après le premier test, on pourrait penser que l'on récupère du bruit en sortie de la carte :

Pour pouvoir remédier à ce problème, on a fait une prédivision de l'horloge

Archive contenant toutes les modifications :

On a trouvé un moyen de faire un CNA facilement, via une "échelle dite R-2R".

`http://www.uchobby.com/index.php/2008/01/08/arduino-audio-dac-options/`

Il faut encore modifier le VHDL pour pouvoir l'exploiter et ainsi générer une constellation.

1.11 semaine 9 : du 28/03/2014 au 03/04/2014

On a pris donc le soin de modifier le VHDL voir en annexe : VHDL final.

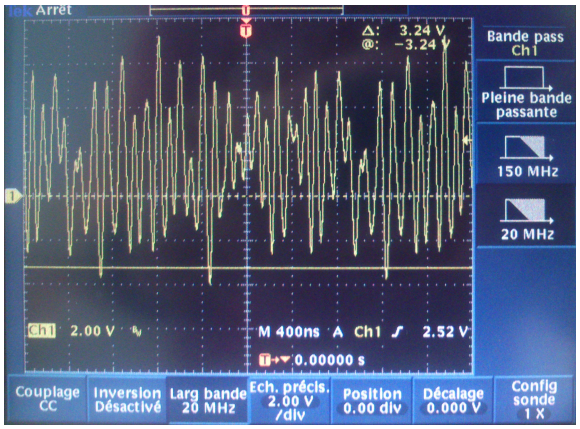

FIGURE 1.6 – Overshoot.

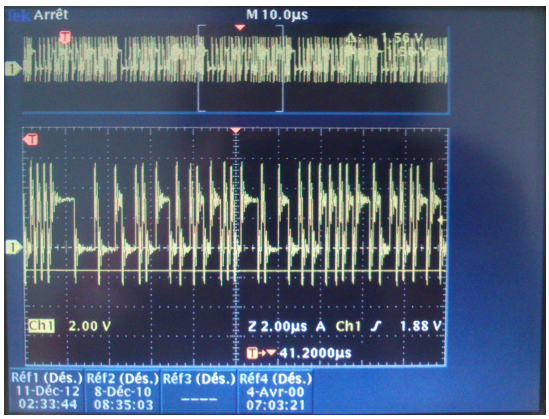

FIGURE 1.7 – Overshoot après division.

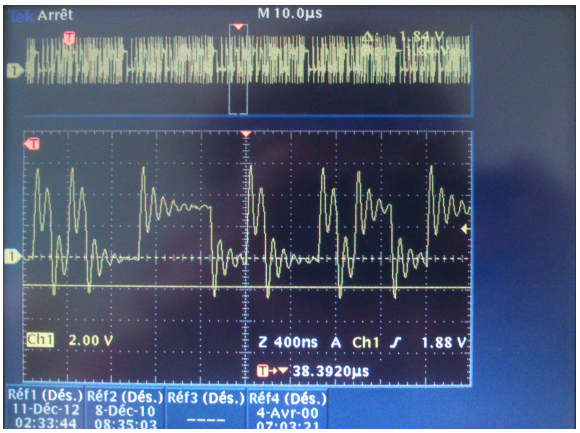

FIGURE 1.8 – Overshoot après division (Zoom).

Ce VHDL a été mis à jour, avec deux entrées (initialisation et une horloge) et un bus de sortie (8 bits), avec les trames pseudoaléatoires.

On envoie ces signaux sur deux C.N.A.s (4 bits chacun) pour générer deux signaux I et Q, on peut voir une constellation se dessiner en mode X-Y.

D'autres constellations réalisées sont disponibles en annexe. On peut monter jusqu'à une constellation de 256 points. ($2^4 * 2^4$).

On a réalisé le montage complet de toute la chaine de transmission. Il se trouve que la Boucle à Verrouillage de Phase (ou appellée PLL, Phase Loop Locker) ne se verrouille pas sur la porteuse du signal que l'on souhaite récupéré, de ce fait on ne peut pas récupérer les signaux I et Q.

1.12 semaine 10 : du 07/04/2014 au 10/04/2014

On a réalisé le montage que l'on évoquait en semaine 9 :

Il se trouve qu'après un réglage précis de la tension d'offset permettant d'être un proche de la fréquence à capturer on obtient :

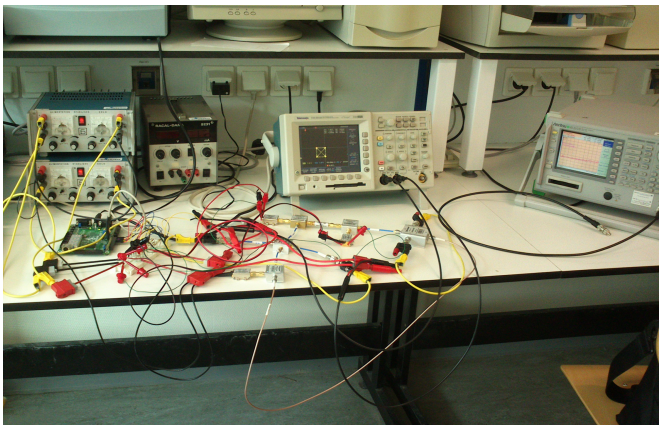

FIGURE 1.9 – Montage avec constellations 4QAM.

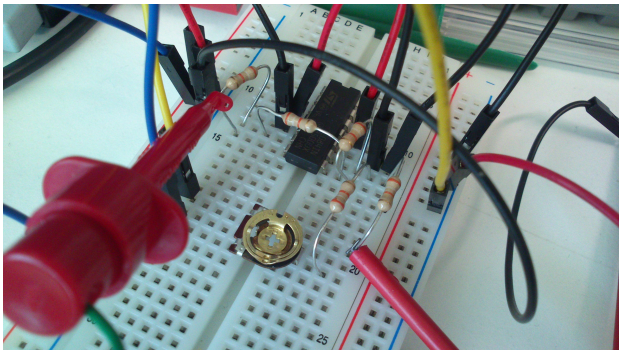

FIGURE 1.10 – Montage pour améliorer la PLL.

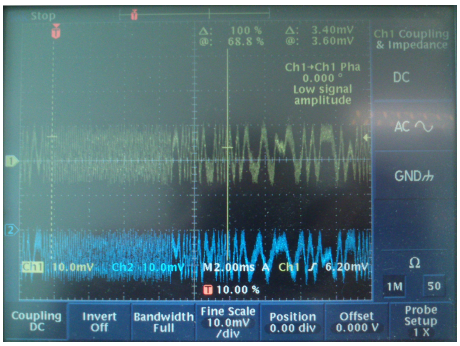

FIGURE 1.11 – Signaux I/Q après ajout du montage.

Annexe A

Annexes.

A.1 Détails techniques des semaines

A.1.1 semaine 1 : du 27/01/2014 au 01/02/2014

Après on a réfléchi sur la potentielle conception/utilisation d'une carte FPGA. Il sera judicieux de ne pas se précipiter sur le schéma et la conception/utilisation d'une carte FPGA. Sans avoir, au préalable fait le VHDL et le testez avec un banc de test virtuel. L'intérêt et de mettre en valeur la nécessité ou non de certains signaux pour le schéma de la carte. On va mettre en oeuvre les connaissances acquises au S7 en VHDL (en TP de Circuit Numérique Programmble).

Pour ce qui est de l'Arduino, on a répondu aux interrogations suivantes (certaines de ces questions paraissent triviales, mais on préfère éviter les surprises) :

Quelle est la vitesse maximale de la communication série de l'Arduino ? - 115200bauds

Est-ce que, un programme de type "Analog Write" est-il une vraie sortie analogique ?

-Non, c'est juste une PWM. Elle sera exploitable pour faire les signales I/Q mais très contraignantes, car il faudra filtré le signal, et ce sera beaucoup trop lent (du au fait que la modulation est faite au alentour de 850Mhz), le FPGA sera un passage obligé cela ouvrira les portes d'un plus grand débit.

Manipulation réalisée sur la maquette TIMS :

On a fait un des montages présentés dans l'ouvrage "Volume D1 Fundamental Digital Experiments" dans le but de visualiser l'objectif à atteindre.

On obtient la constellation suivante (16QAM) :

Ce qui correspond à deux signaux en visualisation temporelle de cette forme :

On pourra facilement obtenir le même résultat en générant un bus de trame pseudo aléatoire et en l'injectant dans un C.N.A. (Conversion Numérique Analogique)

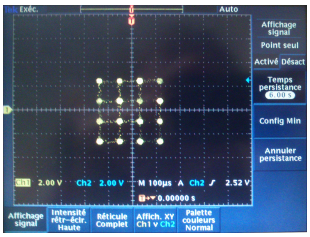

FIGURE A.1 – Constellation visualisée en mode X-Y.

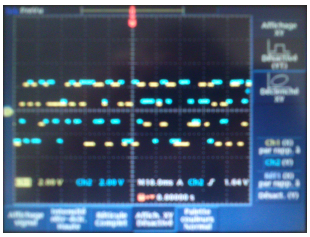

FIGURE A.2 – Signaux I et Q. (Image très floutée, à remplacer)

Code Arduino réalisé :

Comparaison carte réelle/résultat Arduino :

C'était exactement les signaux attendus (même s'il y un décalage a par rapport à l'original).

On aura bien pour une QPSK (4PSK) tous les couples (ie 00,01,11 et 10).

À propos de l'arduino, on pourra envisager une amélioration pour générer une séquence plus longue :

-il faudra simplement utiliser un tableau d'entier.

-paramétrer sa taille avec un N par exemple.

-N correspondra à la taille de séquence de longueur 2^N.

Voir la section "Code Arduino réalisé" en annexe B dans la semaine correspondante.

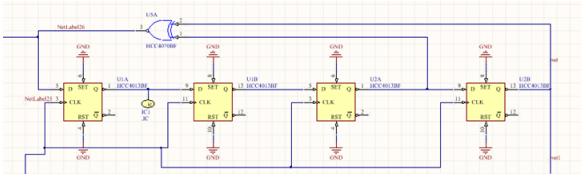

FIGURE A.3 – Schéma électronique du générateur pseudo aléatoire.

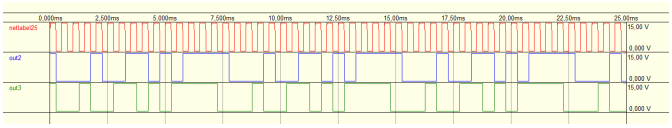

FIGURE A.4 – Simulation du générateur pseudo aléatoire, on a eu la même chose en pratique.

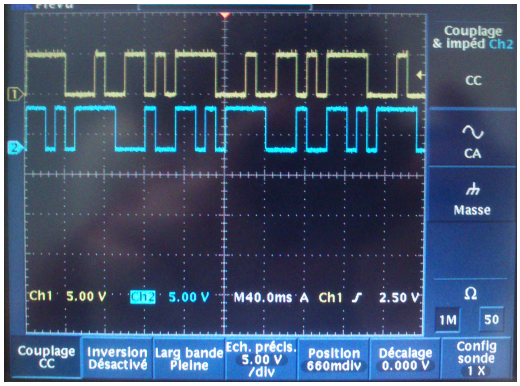

FIGURE A.5 – Resultat avec le générateur pseudo aléatoire "émuler" avec l'Arduino.

Possibilités de communications :

Exploration des possibilités de communication entre le PC et la carte en amont :

Ethernet : + Vitesse de transmission (10M-100M-G)bauds
- La gestion du flux de données doit être faite par un microcontrôleur (celui-ci pourra être implémenté un bloc I.P. à l'intérieur du FPGA)
-> - Haute consommation de CLB du FPGA.

SPI (communication série) : - Vitesse de transmission
-> Arduino capable de communiquer jusqu'a 115200 bauds
-> D'après certaines discussions sur des forums, on pourrait « pousser » au-delà.

I2C : Communication abandonnée, ce n'est pas une priorité.

À propos des blocs I.P. :

En consultant les ordinateurs et en s'appuyant des T.P.s de CNP, on constate que les blocs suivants existent :
- Xilinx Microblaze/Xilinx Power PC ("µP FPGA")
-> Est-ce possible d'utiliser ces blocs I.P. pour programmer un FPGA sur une carte autre que la Nanoboard ? Comment ?
- EMACx (Ethernet Media Acess Control)(:lien physique entre processeurs et "standart Physical Layer device IEEE802.3")

Conception de la carte :

L'idéal est de se documenter sur la Compatibilité électromagnétique.

A.1.2 semaine 3 : du 10/02/2014 au 15/02/2014

Lors de cette semaine, après avoir eu le problème de manipulation, nous avons pu prendre connaissance d'un certain montage qui s'avéra utile pour la conception de la carte/maquette.

On remarquera que ce montage se retrouve dans la partie "typical application" de la documentation technique du 7805.

A.1.3 semaine 4 : du 17/02/2014 au 22/02/2014

On souhaite pouvoir réaliser une communication unidirectionnelle ou même transmettre un signal audio ce qui nous contraint à une bande passante minimale de 48kHz, on prendra 200kHz (pour être bien au-delà de la condition du théorème de Shannon) : le but serait donc de brancher par exemple une source audio quelconque puis de la transmettre à travers la maquette et de la recomposer en fin de chaine.

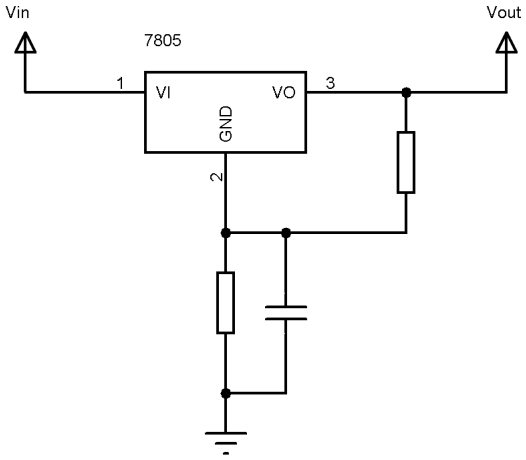

FIGURE A.6 – Schéma limiteur de courant.

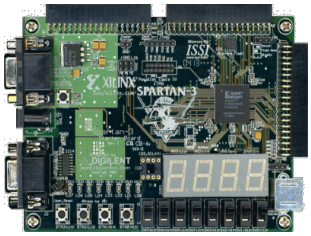

FIGURE A.7 – Digilent Spartan 3.

Après édition rapide du schéma, on en dégage qu'il nous faudra un minimum d'environ 50 entrées/sorties. Mais ceci ce nombre sera très probablement revu à la hausse lors de prochaine séance.

Voici la carte que l'on utilisera dans la suite du projet :

A.1.4 semaine 5 : du 03/03/2014 au 08/03/2014

Voir le Design complet en annexe : Schéma réalisée les semaines 5 et 6.

A.1.5 semaine 9 : du 28/03/2014 au 03/04/2014

Pas de photos pour la constellation à 256 points, mais elle a été réalisée.

Pour ce qui est du montage, malgré que la P.L.L., on a un signal d'erreur après filtrage :

Ce qui met en valeur immédiatement deux problèmes majeurs :

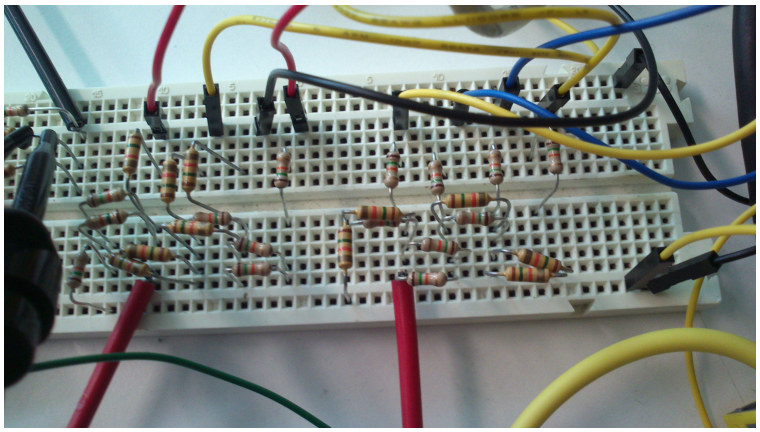

FIGURE A.8 – Le réseau R-2R.

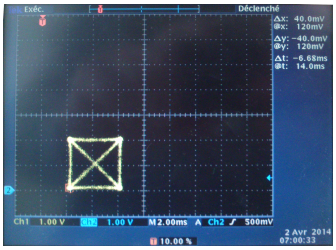

FIGURE A.9 – 4QAM.

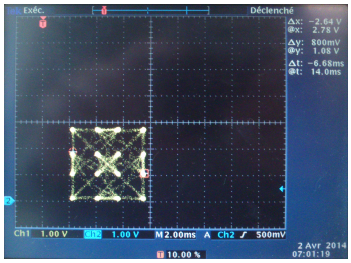

FIGURE A.10 – 16QAM.

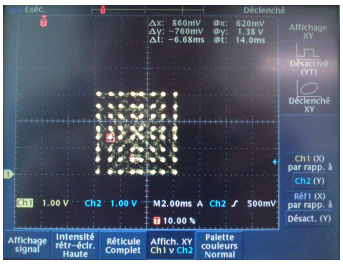

FIGURE A.11 – 64QAM.

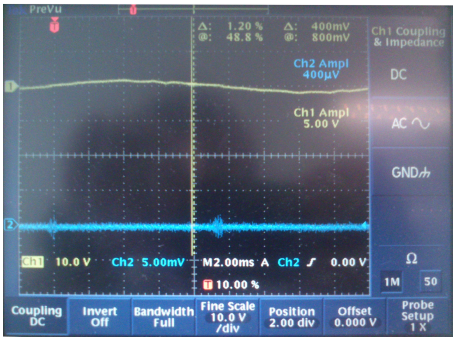

FIGURE A.12 – Chaine 1 : Vtune (commande du V.C.O.)/Chaine 2 : "erreur" de commande

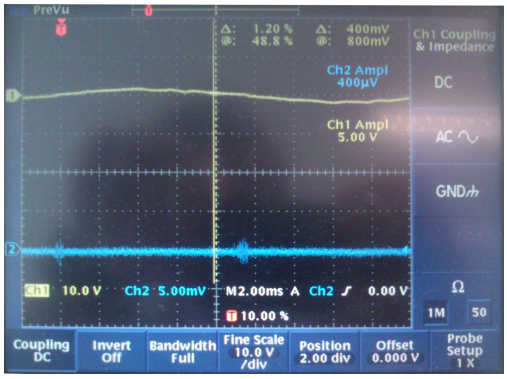

FIGURE A.13 – Proposition de voie d'amélioration.

La chaine fonctionne sur le papier, mais lors de sa mise en pratique, il faut penser à mettre une tension d'offset et aussi réaliser une amplification de l'erreur pour avoir des grandeurs de commande adaptée au Vtune du V.C.O..

On propose l'amélioration suivante (qui sera réalisé à la prochaine séance) :

On remarquera que les valeurs des résistances et des capacités on était choisi pour permettre un fonctionnement correct des A.O. (Amplificateur Opérationnel), mais pas pour ce qui de l'asservissement de la boucle par manque de temps.

A la fin de cette, on a eu à démonter le montage à cause de contraintes concernant les Mini-circuits (car utilisé en travaux pratiques).

A.1.6 semaine 10 : (du 07/04/2014 au 10/04/2014)

Spectres des signaux de la chaine de transmission :

Porteuse utilisée pour la modulation I/Q à environ 60MHz.

Porteuse utilisée pour le transport en fréquence à environ 810MHz :

Question en suspens : faudrait-il utiliser un filtre passe-bande incluant 868Mhz pour pouvoir se verrouiller sur cette fréquence ?

On remarquera le fait que l'oscillateur du côté de la chaine d'émission à d'importantes non-linéarités (on peut observer à l'analyseur de spectre 3 à 4 raies aux fréquences multiples).

Ce qui donne un diagramme de constellation plutôt mauvais si l'on fait le montage suivant :

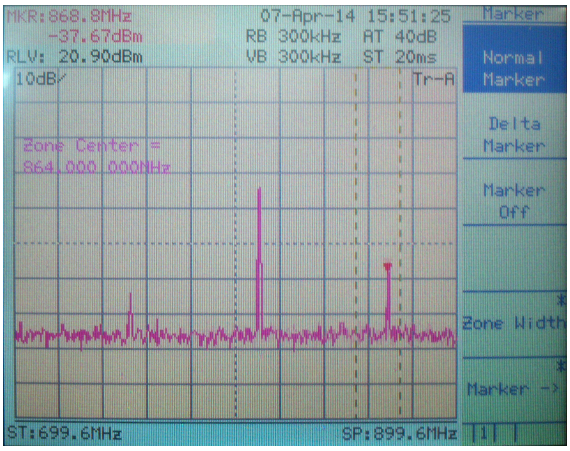

FIGURE A.14 – Spectre en réception.

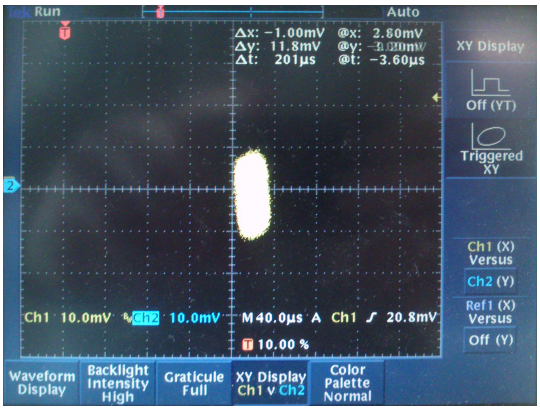

FIGURE A.15 – "Constellation" récupérée avec la PLL défectueuse (on souhaite se verrouiller sur la raie à 868MHz).

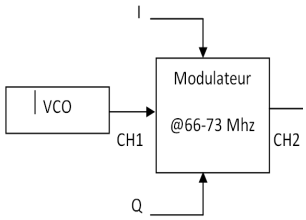

FIGURE A.16 – Montage réalisée.

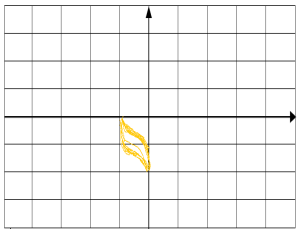

FIGURE A.17 – La constellation 4QAM est attendue, on est loin du résultat attendu (aucune photo n'a était réalisé d'ou le dessin).

A.2 Digilent Spartan 3

A.2.1 Procédure pour faire fonctionner la carte Digilent Spartan 3 :

Ne connectez pas de connecteur JTAG pendant les installations. Redémarrez après chaque installation.

Tout d'abord, il faut savoir que le l'ISE Webpack 14.6 fonctionne avec la carte (n'utilisez pas la 14.7, elle ne supporte pas la génération spartan 3!) :

Installez donc l'ISE WebPack 14.6 (ou antérieur à condition que celle-ci soit supérieur à la 7.1i (d'après leur documentation))

(voir lien utile sur Xilinx)

Une fois l'installation de l'ISE terminé, faites en sorte d'avoir celui que soit opérationnel (la licence est gratuite!).

Installez la "library" ADEPT 2.3 voir le lien utile.

Après vous n'avez plus qu'a utilisez ISE Xilinx normalement et vous serez capable de reconnaitre la carte via la liaison JTAG, et de le programmer.

A.2.2 Liens utiles :

Lien constructeur/vendeur (avec ressources utiles en bas de la page) :

`http://www.digilentinc.com/Products/Detail.cfm?Prod=S3BOARD`

Lien "Library" pour le câble USB digilent :

`http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,69&Prod=ADEPT`

Lien Xilinx (Pour l'ISE 14.6) :

`http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm`

Lien Xilinx (Pour les versions antérieures veuillez faire attention au système d'exploitation, votre architecture matérielle et le FPGA cible) :

(Les informations de compatibilité son sur la page suivante :) (mots clés : xilinx classic(s))

http://www.xilinx.com/tools/classics.htm

A.2.3 Procédure pour faire une programmation en volatile :

Générer le bitstream et le charger.

A.2.4 Procédure pour faire une programmation en "non"-volatile :

Générer le bitstream et l'utiliser pour générer un .mcs qui vous permettra d'inscrire le programme dans la mémoire (E ?)PROM.

On remarquera que la programmation en non volatile est utile pour faire un remise à zéro et recharger le contenu de l'(E ?)PROM, mais semble être sensible au débranchement de l'adaptateur d'alimentation de la carte.

A.2.5 Exemple de programmation :

Voir le lien digilent pour ceci dans cette annexe.

A.3 Schéma réalisée les semaines 5 et 6.

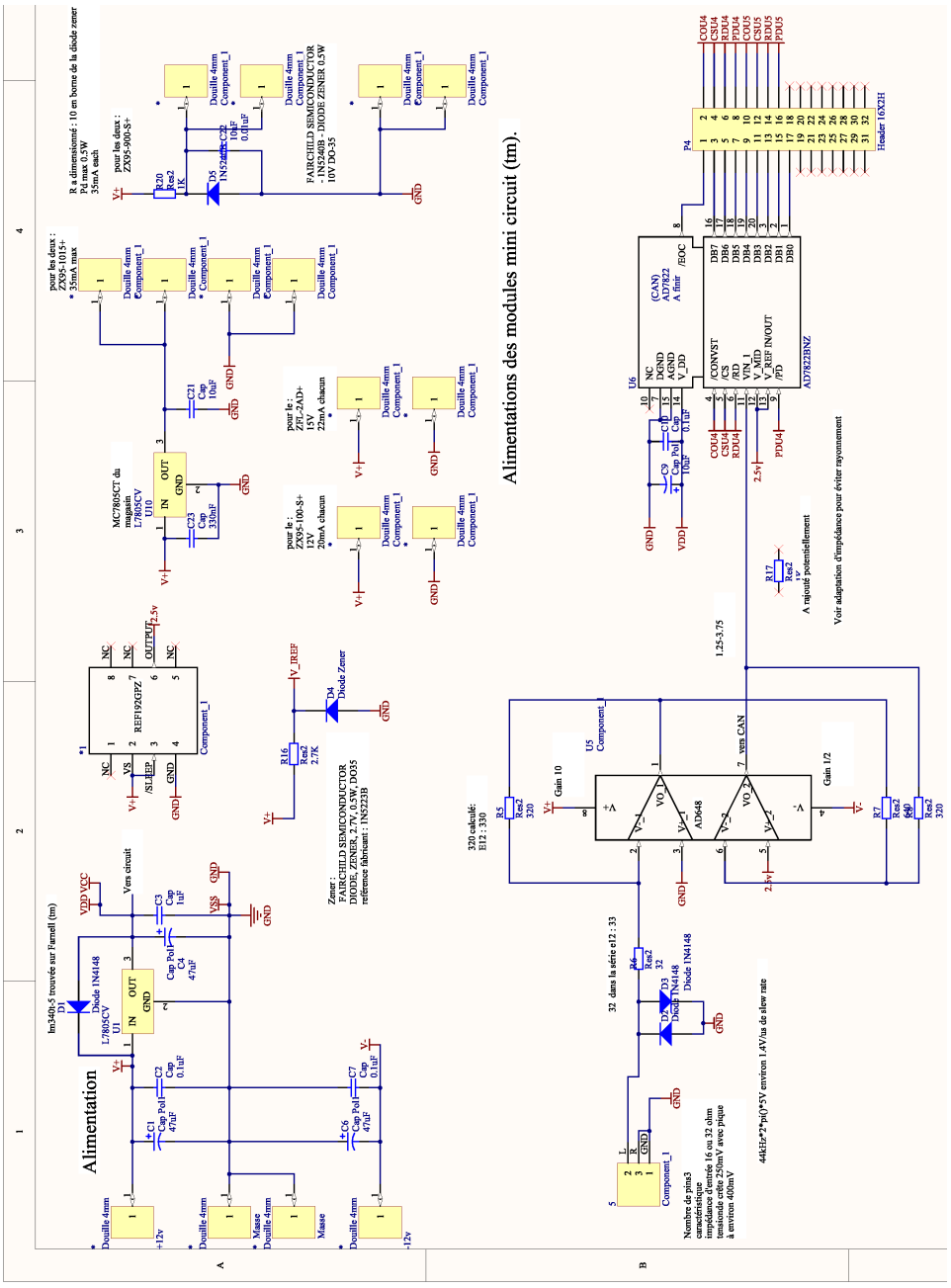

FIGURE A.18 – Schéma au format A2 (1/4).

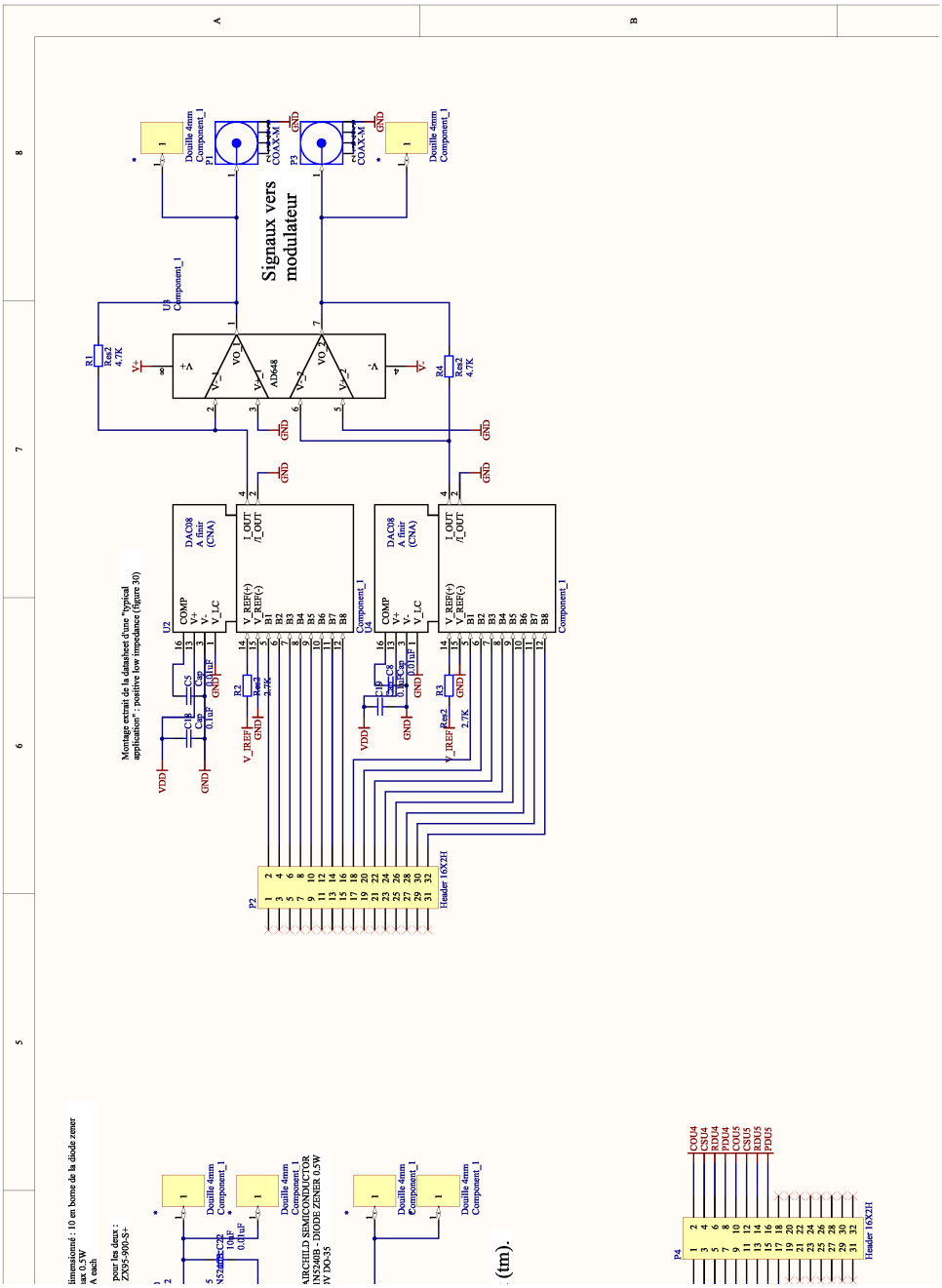

FIGURE A.19 – Schéma au format A2 (2/4).

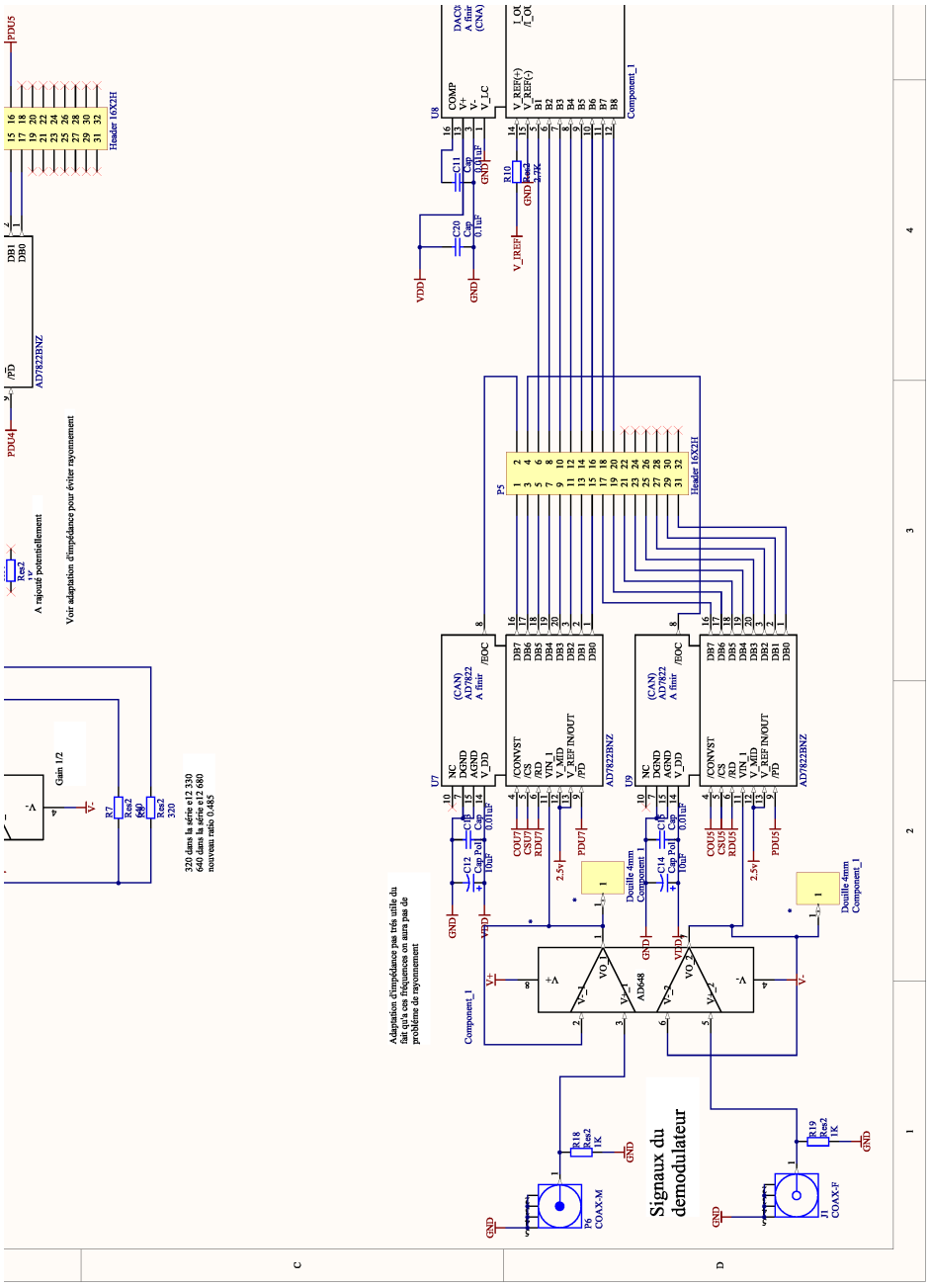

FIGURE A.20 – Schéma au format A2 (3/4).

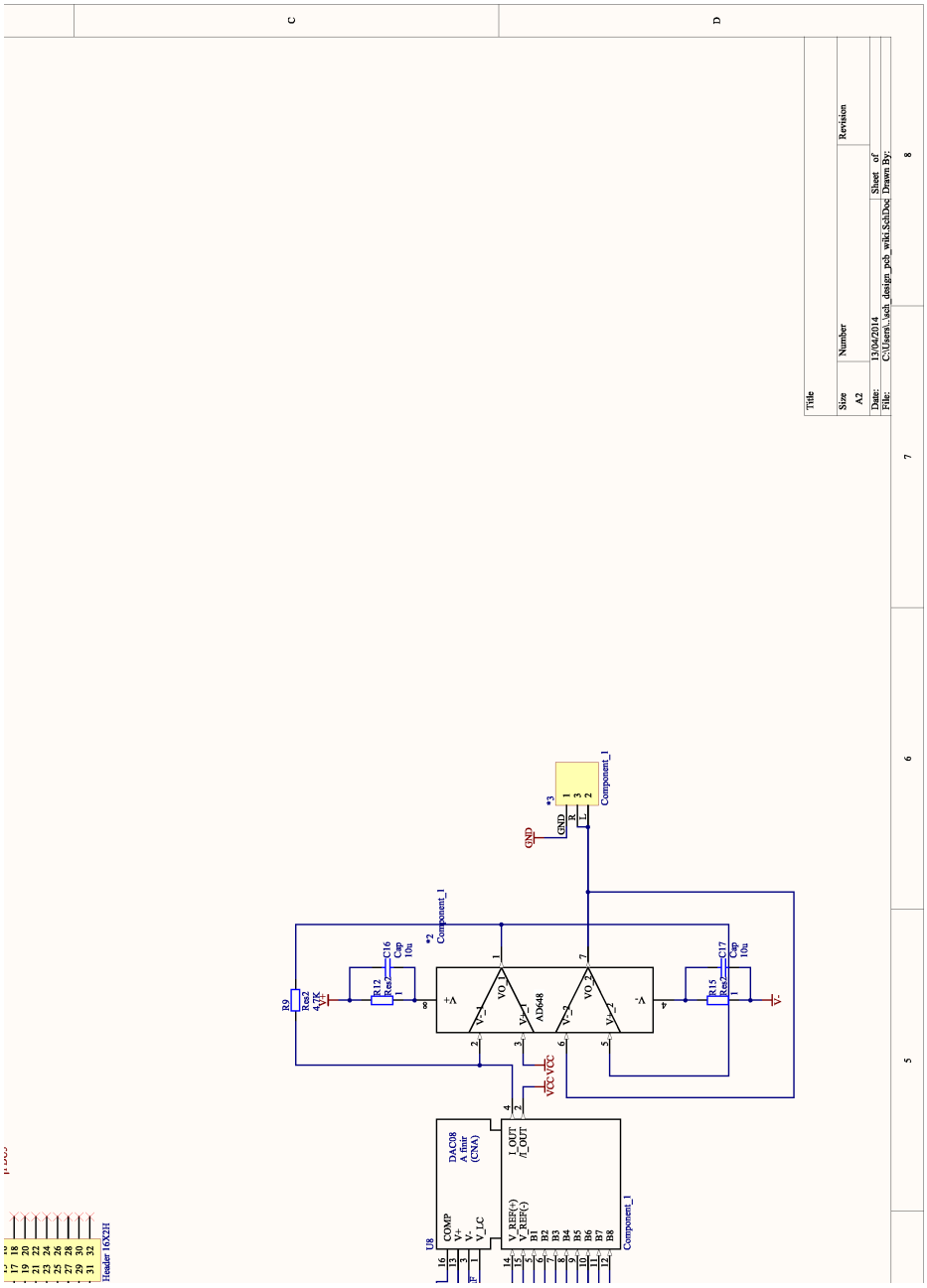

FIGURE A.21 – Schéma au format A2 (4/4).

A.4 VHDL final.

A.4.1 S3Demo.ucf

```
NET "o_Data<0>"  LOC = "E6"  ;
NET "o_Data<1>"  LOC = "D5"  ;
NET "o_Data<2>"  LOC = "C5"  ;
NET "o_Data<3>"  LOC = "D6"  ;
NET "o_Data<4>"  LOC = "C7"  ;
NET "o_Data<5>"  LOC = "D7"  ;
NET "o_Data<6>"  LOC = "C8"  ;
NET "o_Data<7>"  LOC = "D8"  ;
NET "i_init"  LOC = "F12"  ;#interrupteur
NET "mclk"  LOC = "T9"  ;#horloge
```

A.4.2 Freq_Div_mod2.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:18:51 03/26/2014
-- Design Name:
-- Module Name:    Freq_Div_mod_2 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity Freq_Div_mod_2 is
    Port ( i_clk_freq_div_mod : in  STD_LOGIC;
           o_clk_freq_div_mod : out  STD_LOGIC);
end Freq_Div_mod_2;

architecture Behavioral of Freq_Div_mod_2 is

signal niveau : std_logic;

begin
  o_clk_freq_div_mod <= niveau;
process(i_clk_freq_div_mod)
begin
    if(i_clk_freq_div_mod'event AND i_clk_freq_div_mod='1') then
niveau <= NOT niveau;
    end if;
end process;

end Behavioral;
```

A.4.3 bascule_D.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:26:18 03/19/2014
-- Design Name:
-- Module Name:    bascule_D - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity bascule_D is
    Port ( i_D_data : in  STD_LOGIC;
           i_D_clk : in  STD_LOGIC;
  i_D_init : in STD_LOGIC;
  i_D_bit_init : in STD_LOGIC;
           o_D_data : out  STD_LOGIC);
end bascule_D;

architecture Behavioral of bascule_D is

begin
process(i_D_clk,i_D_init,i_D_bit_init,i_D_data)
begin
if(i_D_init='1') then
o_D_data<=i_D_bit_init;
else
if(i_D_clk'event and i_D_clk='1') then
o_D_data<=i_D_data;
end if;
end if;
end process;

end Behavioral;
```

A.4.4 Le_principale.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:19:41 03/19/2014
-- Design Name:
-- Module Name:    Le_principale - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
```

```
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Le_principale is
Port(mclk: in std_logic;
i_init: in std_logic;
o_data: out std_logic_vector (7 downto 0));
end Le_principale;

architecture Behavioral of Le_principale is

--======================
-- Composant--
--======================
component bascule_D is
    Port ( i_D_data : in  STD_LOGIC;
           i_D_clk : in  STD_LOGIC;
  i_D_init : in STD_LOGIC;
  i_D_bit_init : in STD_LOGIC;
           o_D_data : out  STD_LOGIC);
end component;

component Freq_Div_mod_2 is
    Port ( i_clk_freq_div_mod : in  STD_LOGIC;
           o_clk_freq_div_mod : out  STD_LOGIC);
end component;

--======================
-- Signaux
```

```
--======================
signal interieur_generateur_pseudo : std_logic_vector (7 downto 0);

signal mclk_intermediaire : std_logic_vector (10-1 downto 0);

begin

interieur_generateur_pseudo(0)<=
interieur_generateur_pseudo(6) XOR interieur_generateur_pseudo(7);

o_Data<= interieur_generateur_pseudo;

--débité sur une résistance 3.3/0.002*10
   BASCULE_D_N: for I in 0 to 7-1 generate

LOWER_BIT: IF I=0 generate
series_bascule_lsb : Bascule_D
  Port Map( i_D_data =>interieur_generateur_pseudo(I),
  i_D_clk =>mclk_intermediaire(10-1),
  i_D_init =>i_init,
  i_D_bit_init =>'1',
  o_D_data =>interieur_generateur_pseudo(I+1)
  );
end generate LOWER_BIT;

UPPER_BITS: IF I>0 generate
series_bascule_lsb : Bascule_D
  Port Map( i_D_data =>interieur_generateur_pseudo(I),
  i_D_clk =>mclk_intermediaire(10-1),
  i_D_init =>i_init,
  i_D_bit_init =>'0',
  o_D_data =>interieur_generateur_pseudo(I+1)
  );
end generate UPPER_BITS;

   end generate BASCULE_D_N;

   FREQ_DIV_D_N: for I in 0 to 10-1 generate

  PREMIER_DIV : IF I=0 generate
series_freq_div_mod2 : Freq_Div_mod_2
  Port Map( i_clk_freq_div_mod =>mclk,
  o_clk_freq_div_mod =>mclk_intermediaire(I)
  );
  end generate PREMIER_DIV;
```

```
  SUITE_DIV : IF I>0 generate
  series_freq_div_mod2 : Freq_Div_mod_2
  Port Map( i_clk_freq_div_mod =>mclk_intermediaire(I-1),
  o_clk_freq_div_mod =>mclk_intermediaire(I)
  );
  end generate SUITE_DIV;

   end generate FREQ_DIV_D_N;

end Behavioral;
```

A.5 Code arduino.

```
#include <avr/io.h>
#include <util/delay.h>

void output_init(void){
  DDRB |= 0x03; //DDRB |= (1<<PB8)|(1<<PB9);
}

int main(void){
  int r1=1,r2=0,r3=0,r4=0,r5=0,retard=0;
  PORTB=0x00;
  while(1)
    {
      r5=r4;
/*chaque affectation représente une recopie de la bascule
sur le "front montant", ici c'est simplement l'exécution du code.*/
      r4=r3;
      r3=r2;
      r2=r1;
      r1=r4^r5;
      PORTB=0x01*r1+0x02*retard;//écriture des signaux
      _delay_ms(10);//le "front montant", ce qui permet de faire un genre d'horloge.
      retard=r5;//déphasage du signal
    }
  return 0;
}
```