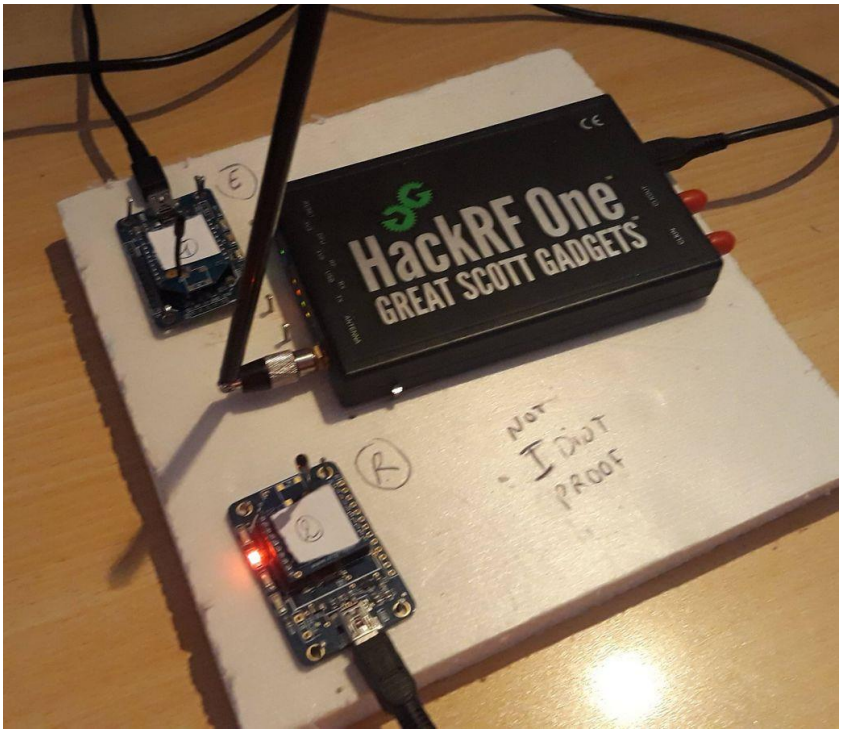
RAPPORT DE PROJET IMA4

Projet 70 : Impact du matériel et du logiciel sur le champ électromagnétique



Réalisé par : KHINACHE Souheib
MOREAU Antoine

Encadré par : BOE Alexandre
REDON Xavier
VANTROYS Thomas

Remerciements :

Nous tenons à remercier nos trois encadrants : Mr BOE , Mr REDON et Mr VANTROYS pour nous avoir suivi et donné des conseils tout au long de la réalisation du projet. Nous les remercions aussi pour tout le matériel qu'ils nous ont fourni , à savoir des XBEE qui nous ont servi de sujets d'expériences et un HackRF one sans lequel nous n'aurions pu réaliser toutes nos expériences.

Nous remercions aussi Gilles GRIMAUD de l'institut de recherche IRCICA pour son aide et Etienne HELLUY-LAFONT qui nous a fourni un outil puissant nous permettant d'avoir de meilleurs résultats finaux.

Enfin , merci à Mr DEFRANCE, Mme BARANOWSKI et Mme ROLLAND pour avoir répondu à nos questions.

Sommaire

Introduction

L'internet des objets est un marché en plein essor. La communication entre les différents objets connectés crée un nuage de signaux autour d'eux. Ces derniers sont vus par les utilisateurs comme des boîtes noires : le fonctionnement se fait sans en connaître les composants matériels et logiciels.

On peut trouver sur le net des datasheets qui précisent les caractéristiques électromagnétiques de certain objets connectés, nous nous posons alors la question : Fonctionnent-ils exactement comme les fournisseurs peuvent les décrire ? Ne pouvons-nous pas les différencier en cherchant très précisément dans le champ électromagnétique émis par ces objets ? Le logiciel et le hardware sont-ils la cause de ces possibles différences dans leurs signatures électromagnétiques ? Quel est donc l'impact du logiciel et du matériel sur le rayonnement électromagnétique d'un objet connecté ?

Nous allons tenter de répondre à ces questions, grâce à nos recherches et expériences sur un exemple d'objets connectés : Les XBEE de *Digi International*.

1. Réflexions et première approche

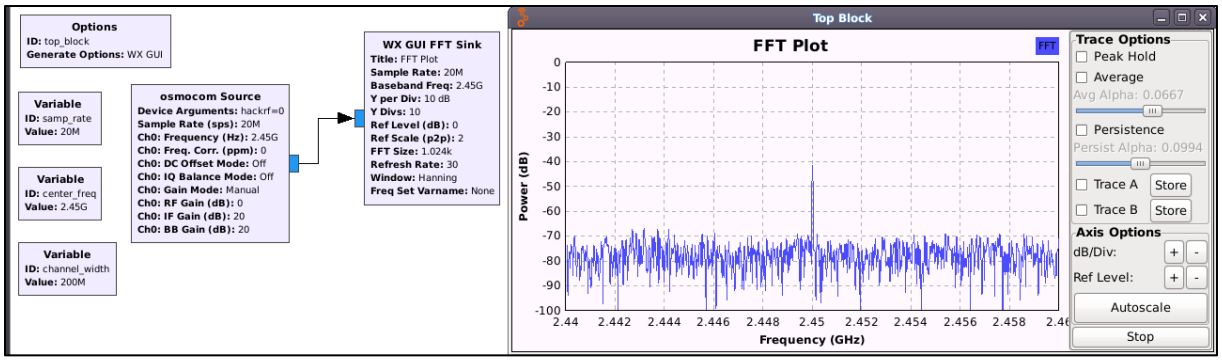
Afin de répondre à la problématique de l'impact du hardware et du software sur l'environnement électromagnétique, il est nécessaire de s'interroger sur le chemin de réflexion à prendre. En effet, le projet se basera sur des études expérimentales et sur l'interprétation des résultats qui en découleront. Une question nous est restée en suspens : Quelle antenne utiliser ? Après tout, nous devons pouvoir étudier le champs électromagnétique environnant, nous devons donc choisir un modèle assez simple d'utilisation qui puisse nous permettre de faire notre étude sur des objets connectés choisis eux aussi stratégiquement. De nombreuses idées nous sont venues : Bluetooth , module Wi-Fi… Après réflexion, le modèle le plus pertinent à étudier serait le **XBee** de *Digi International* puisque celui-ci est simple d'utilisation, facilement disponible et son protocole de communication est peu complexe.

Pour ce qui est de l'antenne réceptrice, nous voulions utiliser une antenne SDR (Software Defined Radio) qui est un récepteur flexible qui peut être utilisé avec de nombreux logiciels (dont des open sources) . Mr BOE nous a conseillé d'utiliser un HACKRF one, et nous a précisé à la même occasion qu'il y en avait des disponibles à prêter. Il faudra coupler notre HackRF à un utilitaire appelé GNU RADIO qui est un logiciel très complet (Pour information : la NASA a utilisé GNU RADIO afin de rétablir le contact avec une sonde spatiale abandonnée)

Durant ce projet, nous avons passé une grande partie du temps à tout préparer pour pouvoir ensuite réaliser des expériences. Cette préparation consistait à :

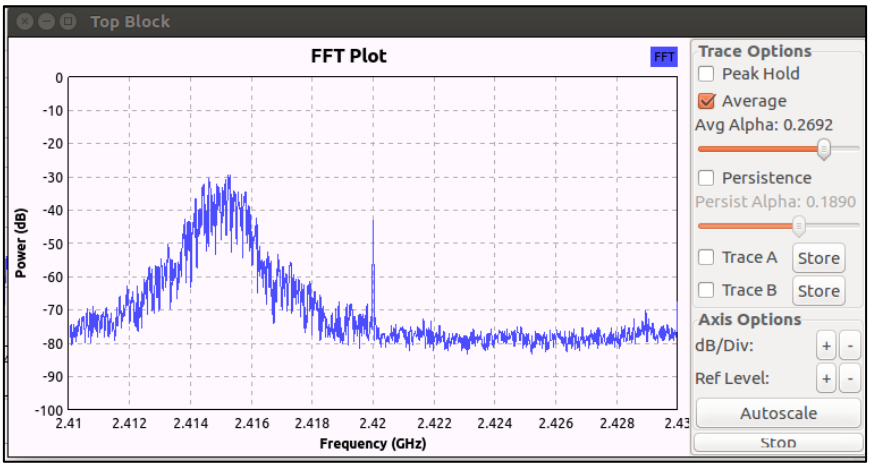
- Découvrir les XBee, et les configurer afin qu'ils communiquent entre eux.

- Prendre en main le logiciel GNU RADIO (Nous n'avions jamais touché à la radio-logicielle). Cette tâche a été très difficile puisqu'il y avait peu de documentation en dehors de la démodulation et du hacking..



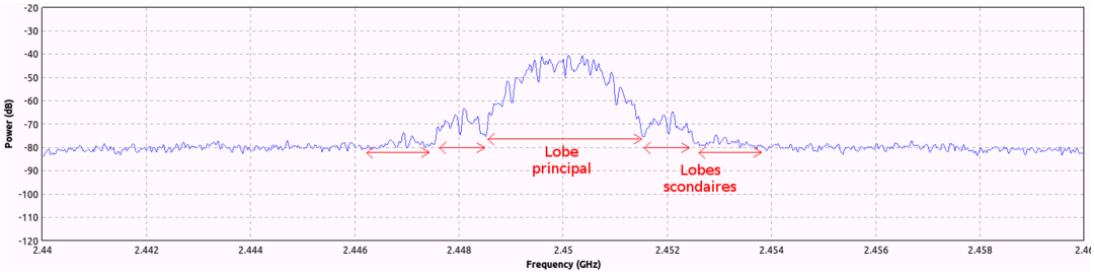
GNU RADIO et son interface

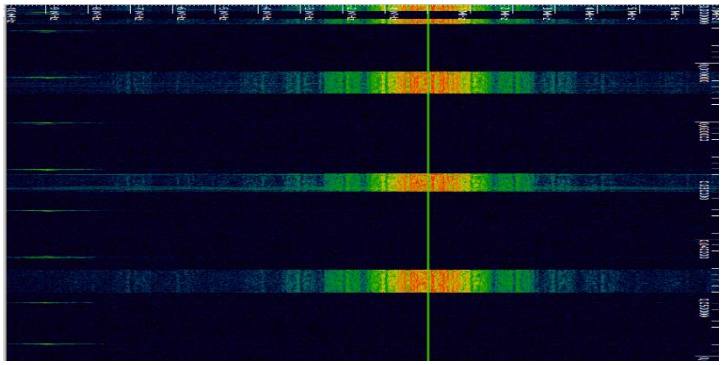
- Comprendre le protocole ZigBee, voir très précisément ce qu'il se passe au niveau électromagnétique.



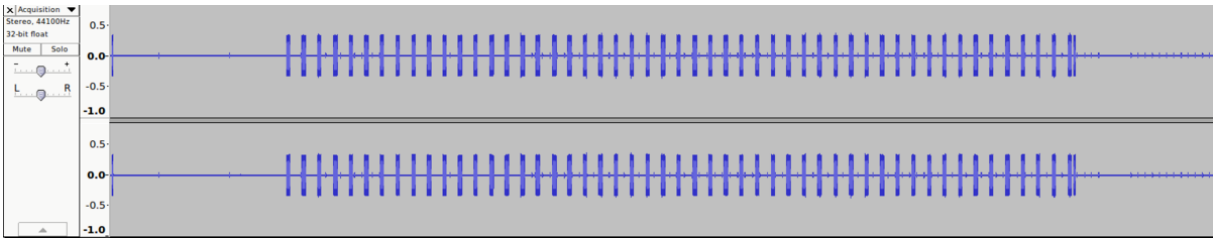
Emission d'un XBee

Nous voyons clairement que le XBee nous transmet quelque chose, mais cela n'est pas suffisant, on va aller plus loin, on va donc essayer de le voir plus clairement:

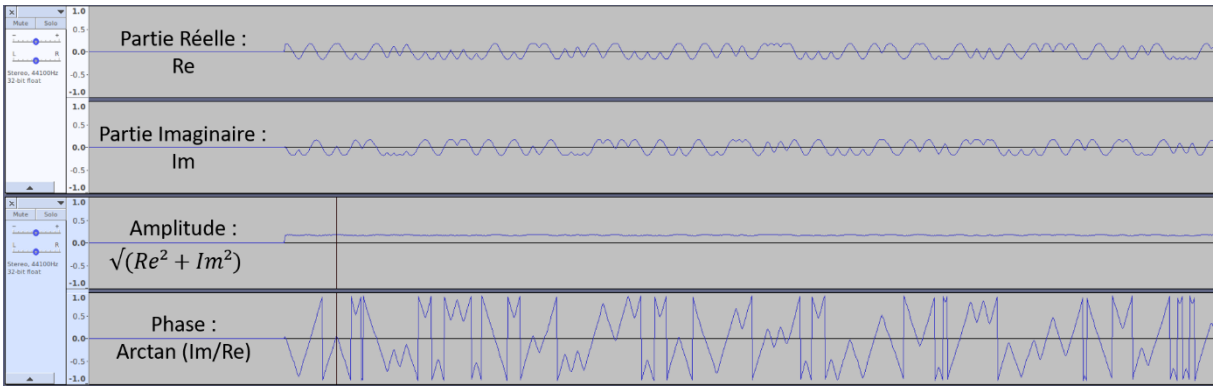




Ensuite , on a regardé des paquets émis dans le temporel :



Toujours en zoomant dans le temporel, on pouvait bien décerner la partie réelle et imaginaire du signal démodulé, et, en traitant l'acquisition, on arrivait à voir son amplitude et sa phase en fonction du temps comme montré ci-dessous :



Avec ces constatations, on a pu réfléchir à « où le logiciel et le matériel pourraient impacter le signal et où nous pourrions regarder et expérimenter »

- Essayer de trouver des expériences qui seraient susceptibles de nous prouver si le logiciel ou le matériel auraient un impact sur le rayonnent des XBee.

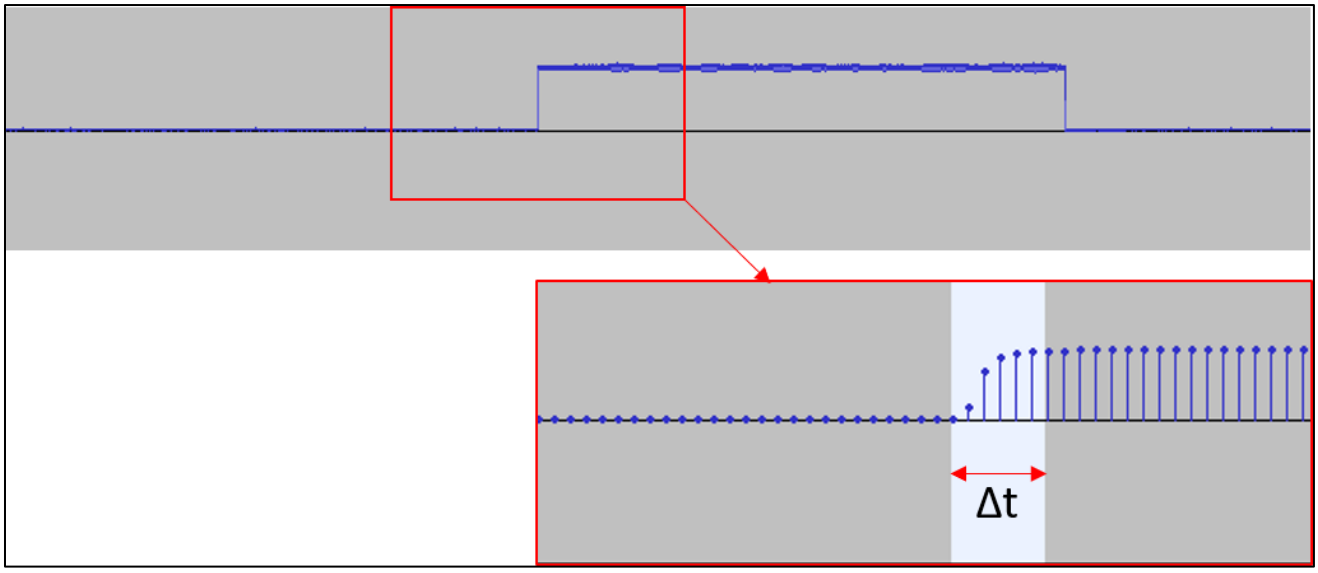
- Réaliser des scripts codés en python principalement qui utilisent des librairies GNU RADIO SDR afin de capturer nous même ce que l'on désire capturer et pouvoir ensuite post-traiter les résultats.

Après toute cette préparation et apprentissage de la théorie, nous étions enfin prêts à créer des protocoles de recherche et s'attaquer au concret .

2. Protocole de recherche

2.1. Expérience : Variation d'amplitude

Cette expérience consiste à calculer le temps Δt mis par le signal pour monter à sa puissance d'émission maximum :



Peut-être que selon le firmware ou le matériel ce Δt varie. Si en testant sur des milliers de paquets reçu, on voit statistiquement qu'il y a une différence entre les différentes configurations testées alors on pourra conclure que le matériel ou le logiciel a un impact sur le champ électromagnétique émis.

Cette étude sera faite avec notre propre script codé en python (voir Annexe)

2.2. Expérience : Mesure du SNR

L'expérience de la mesure du SNR consiste à calculer le rapport signal sur bruit selon plusieurs facteurs :

- La distance variant de 10 cm à 70 cm (Au-delà de 70 cm le SNR devient trop faible pour en faire une interprétation).

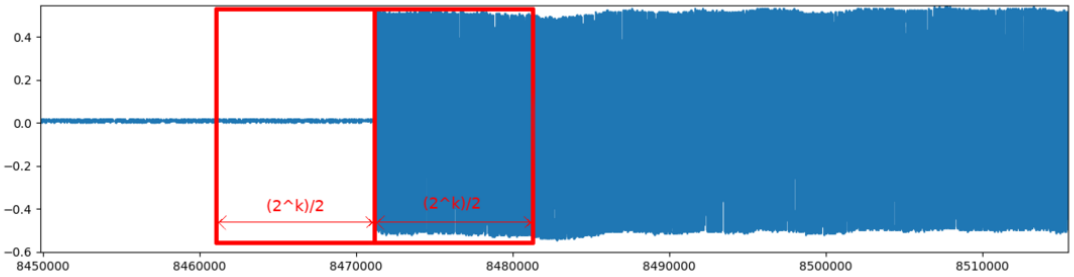
- La fréquence de transmission de l'information variante entre plusieurs canaux d'émission (Canaux 11/13/15 respectivement 2.435 GHz, 2.445 GHz et 2.455 GHz).

On comparera le SNR en fonction de la distance entre deux XBEE série s1 et deux XBEE PRO. On changera aussi le firmware de l'un des XBEE série s1 afin de pouvoir comparer le résultat avec lui-même émettant avec le firmware de base , mais aussi on modifiera le hardware puisqu'au lieu d'émettre seulement via la liaison série, on utilisera des microcontrôleurs ATMEGA328P et ATMEGA2560 pour gérer la transmission des données.

Puis nous ferons de même mais en calculant le SNR selon la fréquence.

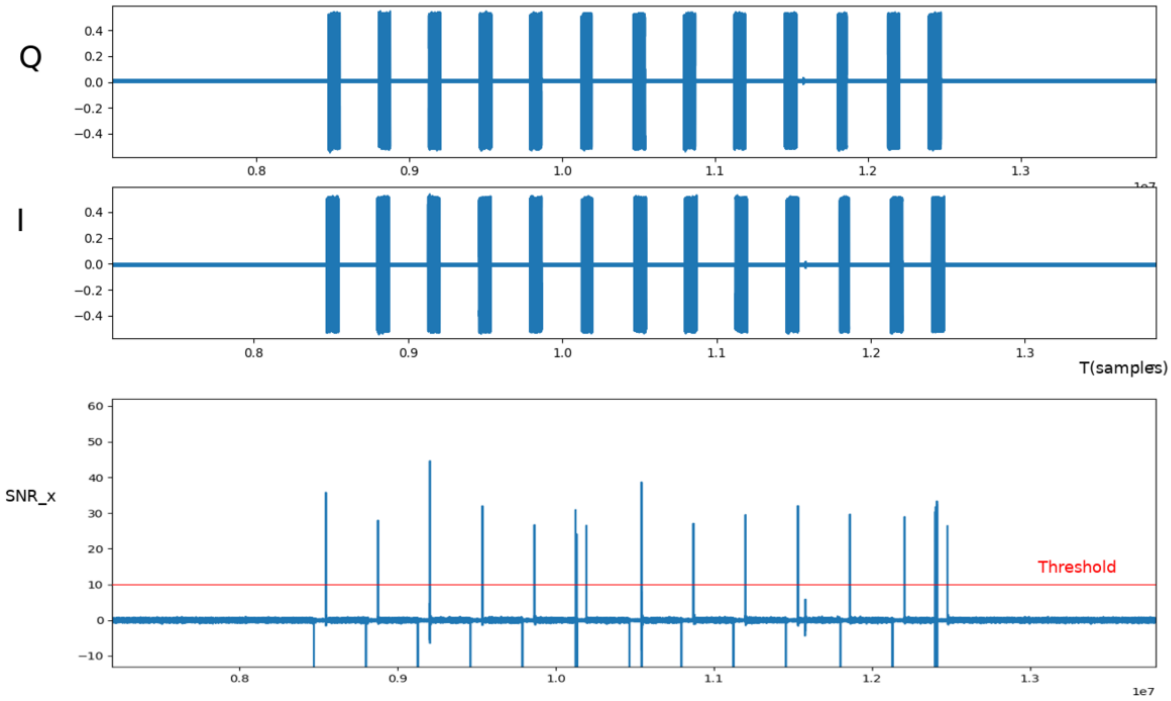
Pour chaque acquisition de données (à une certaine distance OU à une certaine fréquence d'émission), nous prendrons la valeur du SNR de 1000 paquets reçu environ, puis nous calculerons la moyenne.

Nous avons réalisé un algorithme permettant de calculer le SNR. Cet algorithme utilise une fenêtre glissante qui calcule le SNR le long du signal en temporel.



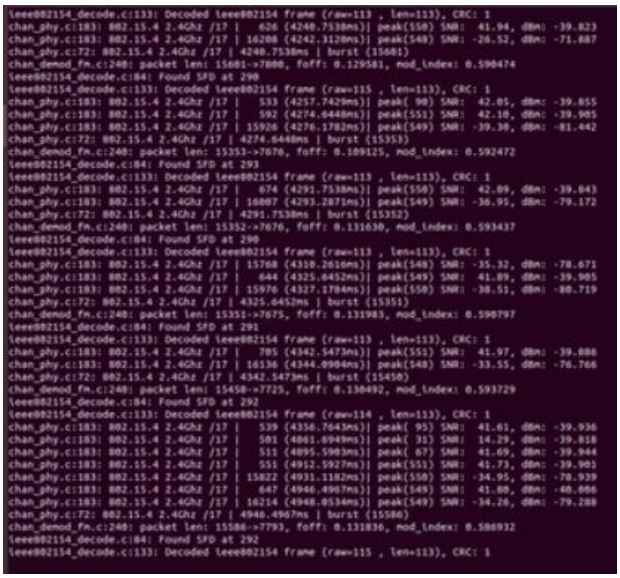
Fenêtre glissante calculant le SNR

On calcule le logarithme de la valeur moyenne de la partie droite sur la partie gauche, et ainsi on obtient des PICS de SNR à chaque paquet reçu, comme ceci :



Néanmoins notre algorithme n'était pas assez performant et nous avions quelques résultats incohérents. Nous avions besoin de conseils et d'un outil plus puissant, développé méticuleusement. C'est là que le doctorant Etienne Helluy-Lafont a pu nous aider. Nous avons choisi d'utiliser son outil développé en C qui utilise la librairie SoapySDR afin d'avoir les résultats les plus précis possibles pour cette expérience. L'avantage de l'utilisation de l'application nous permet directement de nous focaliser sur les paquets correctement reçus et non les données parasites qui peuvent perturber l'antenne.

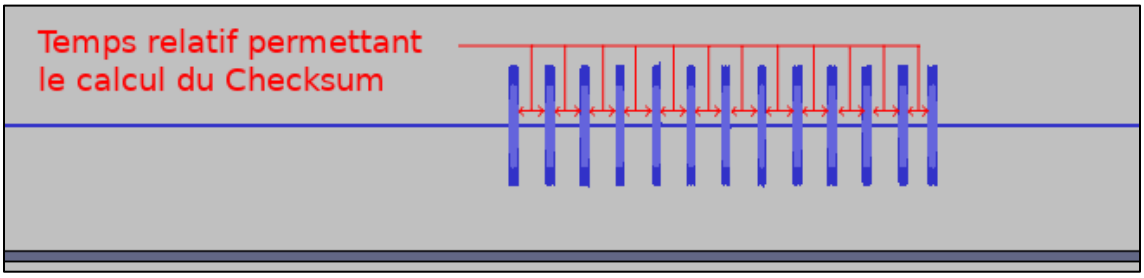
En effet, le programme reste à l'écoute de l'environnement électromagnétique sur une bande de fréquence autour de la fréquence d'écoute du HackRF. Lors de la réception d'un signal, le programme démodule le signal et vérifie si le CRC (ou le FCS pour le Frame Check Sequence) et les données concordent. Si c'est le cas, le CRC nous renvoie 1. Il s'agit là alors d'un signal. L'utilisation ainsi de ce programme nous permet de ne pas obtenir de valeurs parasites pour le SNR et filtrer directement les informations qui nous sont intéressantes.



Aperçu du programme

2.3. Expérience : Temps de calcul du FCS

Pour cette expérience on transmet la chaine de caractère HELLO 256 fois toutes les 1 seconde. Les données étant assez conséquentes, le XBEE transmet plusieurs fragments du paquet. L'objectif de cette expérience est d'étudier le temps entre chaque fragment d'un paquet. L'espace de temps entre chacun des fragments permet de calculer le FCS (Frame Check Séquence qui sert à minimiser les erreurs de transmission).



*Aperçu du temps de calcul du FCS d'un Paquet de 256 * « HELLO » fragmenté*

Cette étude sera réalisé à l'aide d'un de nos script joint en annexe. Le problème avec les acquisitions faites avec GNU RADIO est qu'au bout de 5 secondes de capture, on observe des erreurs dans le signal capturé dues à la consommation du CPU.
Notre script fait les tâches suivantes dans l'ordre :

- Effectuer une capture du signal pendant 1 seconde et l'enregistrer
- Traiter les paquets par un algorithme en python qui étudie le temps entre chacun des fragments. Ce temps sera obtenu en nombre d'échantillons, défini avec un seuil d'acceptation absolu entre 150.000 et 300.000 échantillons. Et pour avoir de bons résultats on impose un seuil d'erreur de +-10% autour de la moyenne. La moyenne est calculée en continue.
- Toutes les moyennes sont enregistrées (ajoutées à la fin) dans un fichier
- On revient à la tâche 1 (re-capture)

On répète le processus un certain nombre de fois afin d'obtenir un nombre de valeurs nous permettant de nous conforter sur l'issue de l'expérience (environ 1000 valeurs)

La capture et l'étude de chacun des modules dure une à deux heures.

3. Mise en application du protocole de recherche

3.1. Résultat de l'expérience : Variation d'amplitude

Pour cette expérience nous nous sommes rendus compte que le taux d'échantillonnage du HACKRF one n'était pas assez élevé. Bien qu'il soit de 20MHz ce qui est une bonne performance, ce n'est pas assez pour avoir suffisamment de points entre le début et la fin de la montée en amplitude : On obtient en moyenne 3 à 4 points seulement (voir ci-dessus). Sachant que le temps entre 2 échantillons est de 1/(20*10^6).
En faisant varier le firmware ou le matériel lors de l'expérience, il était à chaque fois impossible de trouver une différence de Δt suffisamment grande pour pouvoir conclure quoi que ce soit…

```
speedclocker@speedclocker-X756UQ:~/Documents/Polytech/IMA4/Projet$ sudo python statsStudy.py
[sudo] password for speedclocker:
Moyenne : 3.10515021459
Ecart-type : 0.842599237651
Variance : 0.70997347529
```

Valeur moyenne des échantillons Δt durant (liaison série)

```
speedclocker@speedclocker-X756UQ:~/Documents/Polytech/IMA4/Projet$ sudo python statsStudy.py
Moyenne : 2.87692307692
Ecart-type : 0.644686979685
Variance : 0.415621301775
```

Valeur moyenne des échantillons durant Δt (XBEE + ATMEGA328)

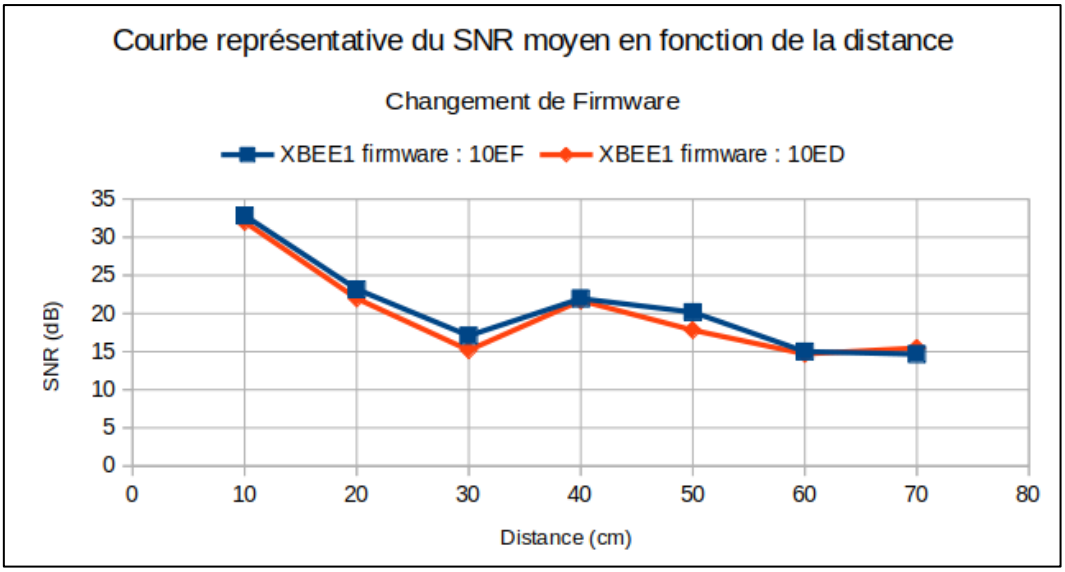
La montée en amplitude dure en moyenne 0.155 µs pour le XBEE en liaison série contre 0.143 µs pour le XBEE avec un ATMEGA328. Mais le manque de précision et donc d'échantillons ne nous permet pas de conclure sur cette expérience. Néanmoins notre script python est prêt à être réutilisé avec une antenne plus puissante, peut-être aurions-nous eu un meilleur résultat ?

3.2 Résultat de l'expérience : Mesure du SNR

3.2.1. Variation de la distance

3.2.1.1. XBEE 1 firmware 10EF / XBEE 1 firmware 10ED

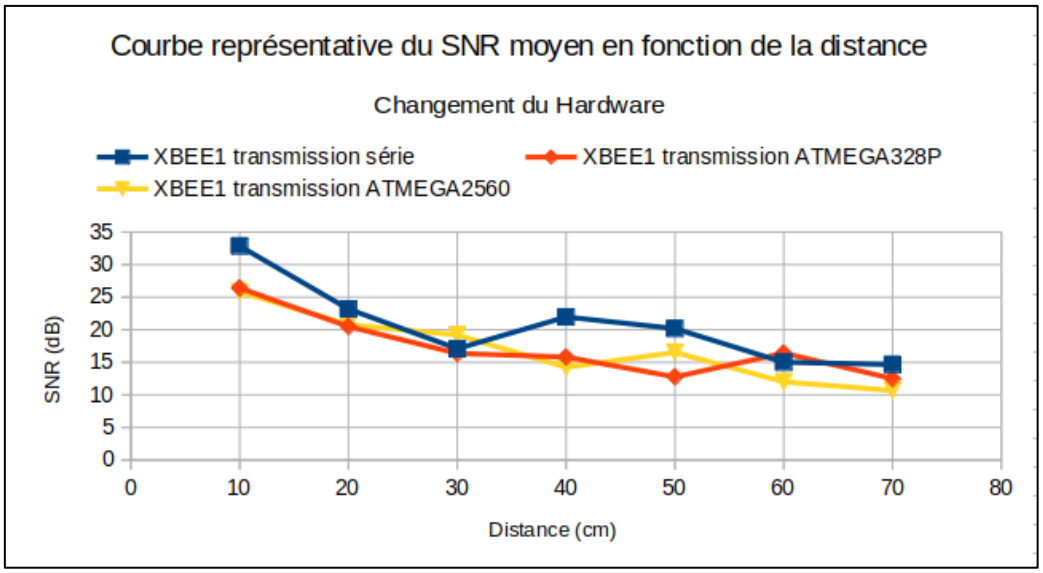
Pour cette capture, on change le firmware du XBEE en 10ED (initialement 10EF).



Les acquisitions étant faites dans les mêmes conditions, il semblerait que le changement de firmware ait un petit impact sur la puissance d'émission : Cette petite différence de décibel ne semble pas aléatoire puisqu'on peut clairement voir que la courbe orange reste toujours inférieure à la courbe bleue, or le niveau de bruit est resté inchangé.

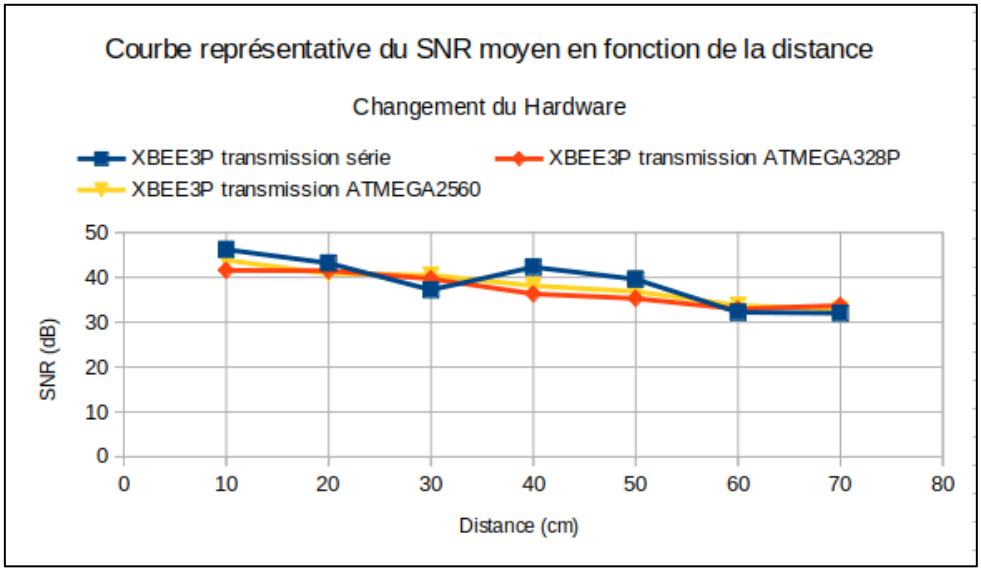
3.2.1.2. XBEE1 / XBEE1 ATMEGA328P / XBEE1 ATMEGA2560

Pour cette capture, on modifiera le hardware. Après plus de 22 000 captures de paquets et de post-traitement on obtient les courbes suivantes :



Les paquets émis via liaison série ont plus de puissance que ceux émis grâce aux Arduino UNO et MEGA notamment à 10 cm, 40 cm et 50 cm. Néanmoins on ne peut pas trop conclure au vu de ces résultats. Seul le résultat à 10 cm de l'antenne semble pertinent puisqu'on a une différence de 10 dB par rapport aux microcontrôleurs.
Il faut ressayer avec un autre XBEE afin de voir si cette hypothèse est vraie.

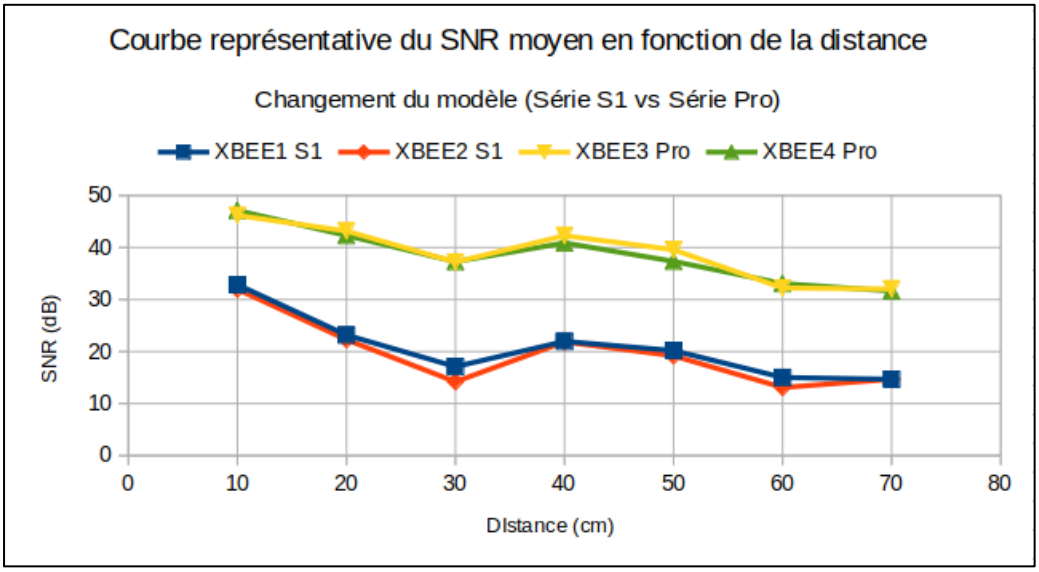
3.2.1.3. XBEE3 / XBEE3 ATMEGA328P / XBEE3 ATMEGA2560



Remarque : On peut voir que les XBEE PRO émettent plus puissamment leurs signaux (45 dB contre 35 dB chez les séries s1). Aussi , ils sont plus robustes face à la distance : là où notre SDR ne captait plus les séries s1 au-delà de 70 cm, les PRO pouvaient être détectés à plus d'1 m.

Si on analyse le graphique, on se rend compte que notre hypothèse est confirmée puisqu'on retrouve la même configuration, à savoir une transmission série plus puissante à 10,40 et 50 cm et une faiblesse à 30 cm. Coté ATMEGA , le SNR reste plutôt stable et décroit très doucement. C'est plutôt réconfortant de trouver une **clé d'identification**. En effet lors d'un calcul de SNR on pourra estimer si le XBEE est connecté au PC en liaison série ou à une ATMEGA quelconque selon si son SNR suit l'allure ou non de la courbe bleue.
En cherchant des raisons à cela, on se demande si ce phénomène n'a pas un rapport avec la longueur d'onde du signal modulé : la longueur d'onde est de 12 cm, et la distance 30 cm coïnciderait avec sa **demi-longueur d'onde**. Pour aller encore plus loin dans la vérification on effectue 15 000 captures de paquets supplémentaire pour avoir les 4 XBEE à comparer entre eux.

3.2.1.4. XBEE 1 / XBEE 2 / XBEE 3 / XBEE 4



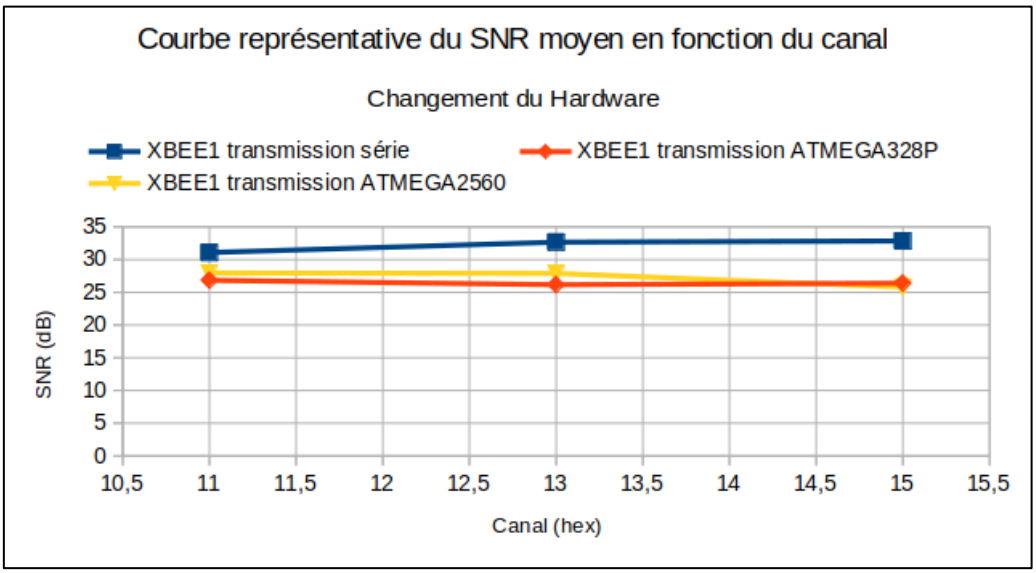
Cette variation du SNR caractéristique (transmission série) est moins flagrante avec les microcontrôleurs dont les courbes de SNR/distance sont plus lissent et subissent moins cet effet de cassure à 30 cm et de remontée à 40 cm et 50 cm.

L'expérience du SNR en fonction de la distance et du hardware était une expérience pertinente finalement, nous n'avions pas anticipé un résultat positif. Reste à savoir si le SNR en fonction de la transmission des paquets sur différents canaux nous aidera à faire la distinction entre les deux ATMEGA ?

3.2.2. Variation du canal de transmission

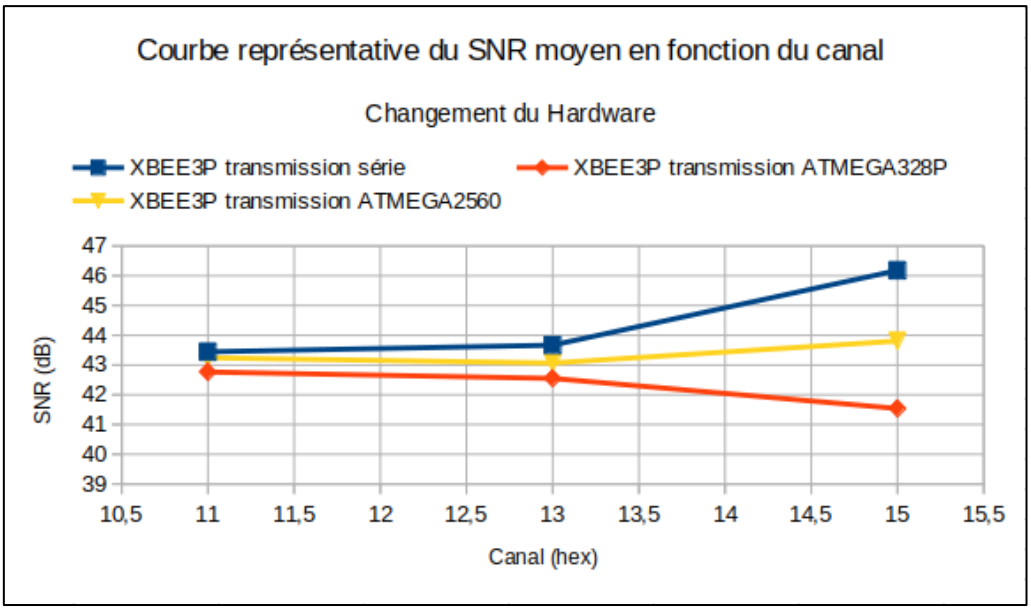
Maintenant le canal d'émission varie de 0x11 à 0x15

3.2.2.1. XBEE1 / XBEE1 ATMEGA328P / XBEE1 ATMEGA2560



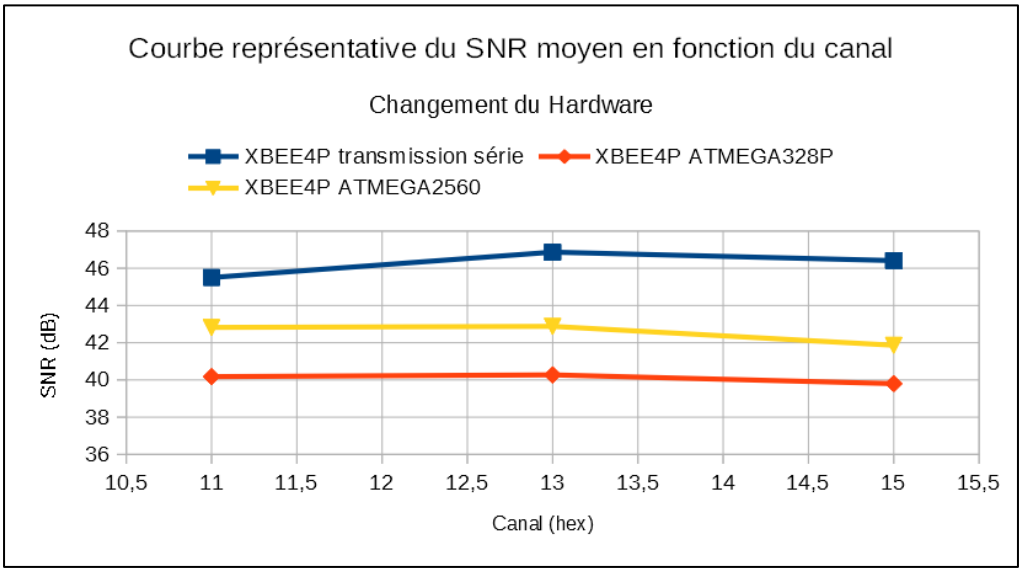
Mis à part un SNR toujours plus élevé via transmission série, on peut rien conclure pour le moment. Essayons avec un autre XBEE.

3.2.2.2. XBEE3 / XBEE3 ATMEGA328P / XBEE3 ATMEGA2560



On peut voir sur ce graph des caractéristiques intéressantes puisque chaque courbe possède une pente différente lorsqu'on passe au canal 15. Aurons-nous la même chose avec un autre XBEE PRO?

3.2.2.3. XBEE3 / XBEE3 ATMEGA328P / XBEE3 ATMEGA2560



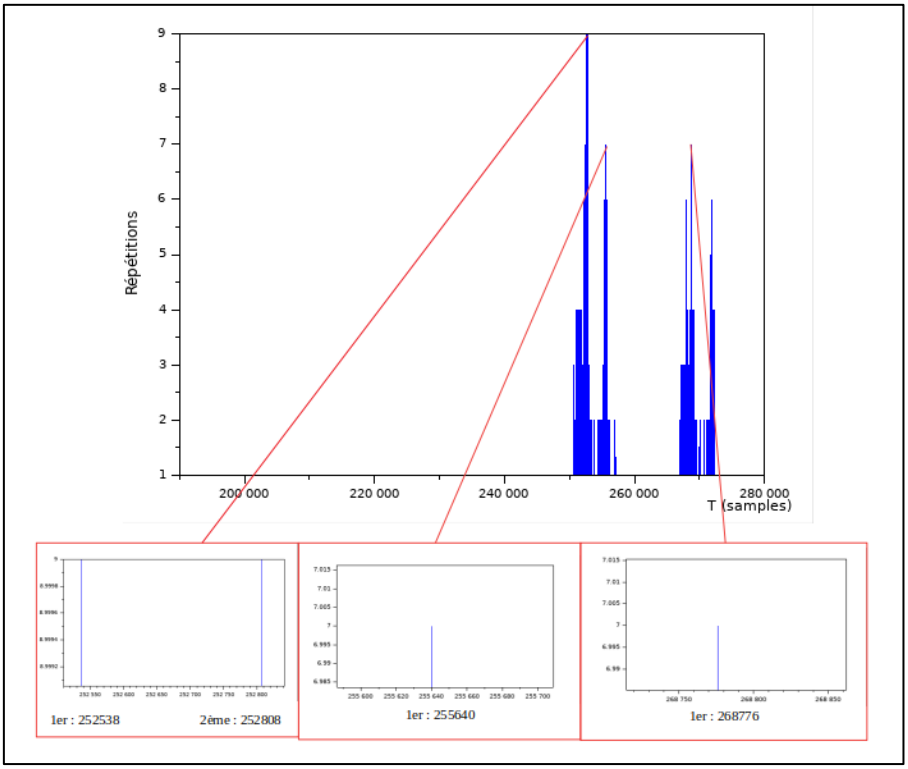
Après avoir comparé tous les graphs, on peut dire que systématiquement la puissance émise selon les canaux est la même pour chaque hardware :

SNR_série > SNR_ATMEGA2560 > SNR_ATMEGA328P

Cependant, les courbes ne suivent pas un pattern caractéristique qui permettrait à l'aide à l'identification du matériel. Si par exemple deux XBEE émettent avec un hardware différent, on pourra seulement conclure « *Les XBEE n'utilisent pas le même matériel pour émettre* » mais on ne pourra pas dire lequel utilise quoi, du moins on utilisera plus l'expérience du SNR en fonction de la distance qui en dira un peu plus.

3.3. Résultat de l'expérience : Temps de calcul du FCS

3.3.1 XBEE1 / XBEE2



XBEE1 : Temps de calcul du FCS

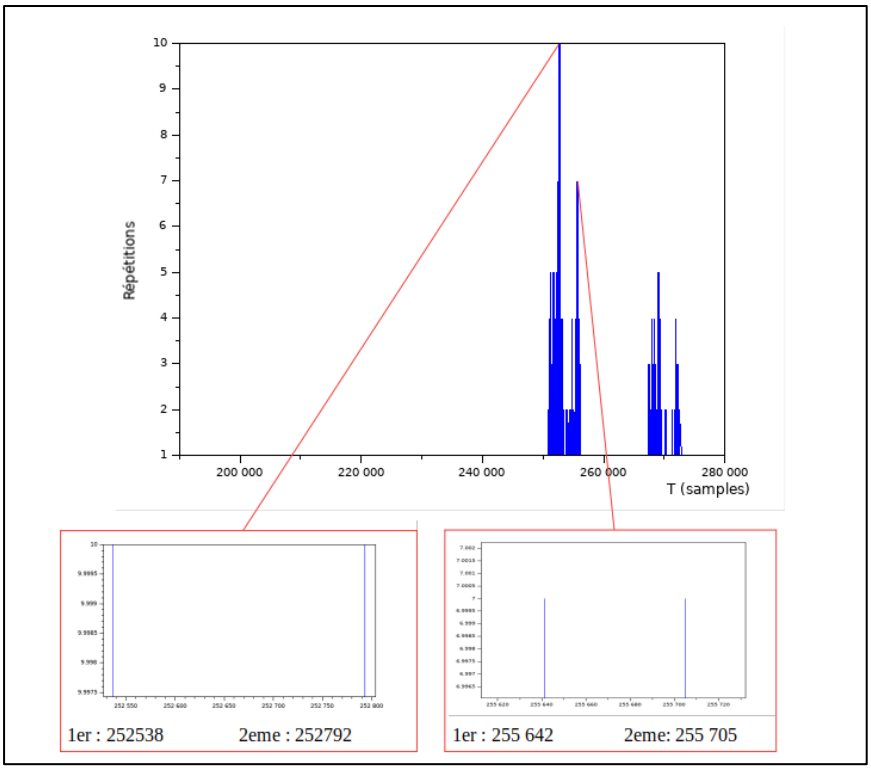
On rappelle qu'on s'intéresse au nombres d'échantillons entre chaque fragment (temps de calcul du FCS) :

Moyenne : 259675,81 (0.0129s)

Ecart-type : 8369.7417 (0.418s)

Variance : 70052576 (3.5s²)

Comparons maintenant avec un second XBEE :



XBEE2 : Temps de calcul du FCS

Ici nous avons :

Moyenne : 258762.58 (0.0129s)

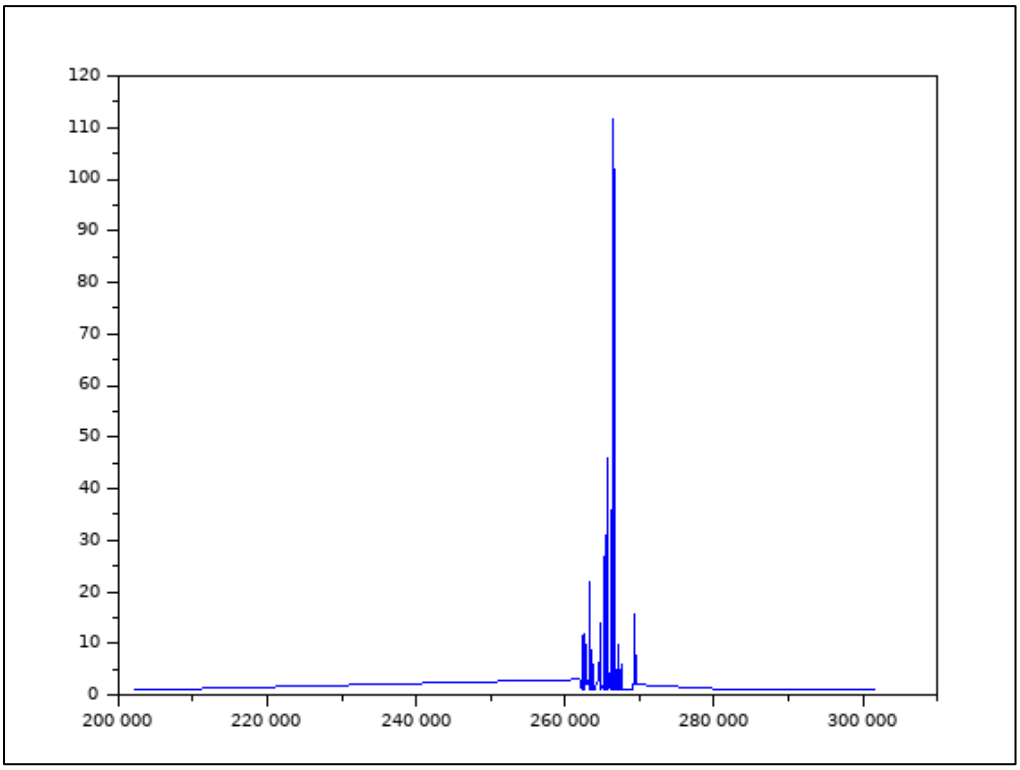
Ecart-type : 9165.4097 (0.458s)

Variance : 84004736 (4.2s²)

C'est intéressant car la moyenne pour le calcul du FCS est la même pour les deux XBEE. On peut voir que seul la variance permet de vraiment les différencier (3.5s² contre 4.5s²).

3.3.2. ATMEGA328P / ATMEGA2560

On change le matériel désormais, on va pouvoir comparer nos deux graphs précédents avec ceux des ATMEGA328P et 2560.

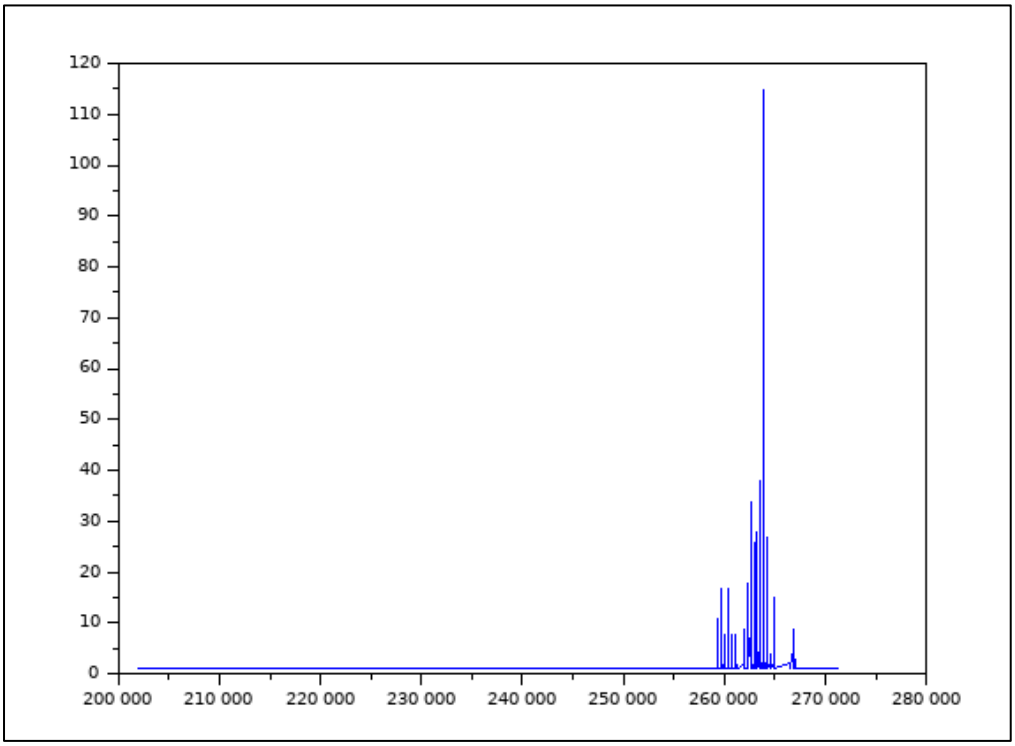


Moyenne : 265818.58

Ecart-type : 4748.5965

Variance : 22549168 (1.127s²)

XBEE1+ATMEGA328P : Temps de calcul du FCS



Moyenne : 262877.83

Ecart-type : 3675.037

Variance : 13505897 (0.6s²)

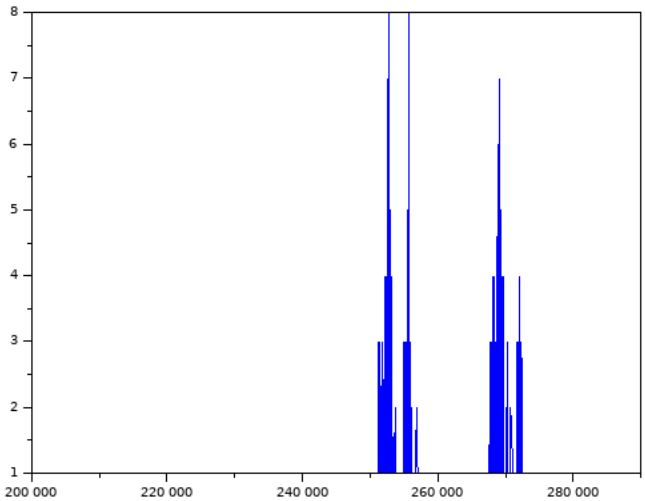
XBEE1+ATMEGA2650 : Temps de calcul du FCS

On aurait alors :

Variance_XBEE1_série >> Variance_XBEE1_ATMEGA328P > Variance_XBEE1_ATMEGA2560

On peut remarquer que via les deux Arduino, la plage de nombre d'échantillon entre les fragments semble être similaires entre les deux (259 000 -> 267 000).
Si on prête attention aux plages d'échantillons des 4 graphs on voit que la plage d'échantillons des Arduino se trouve exactement entre les deux plages du graph du XBEE1 sans microcontrôleur

3.3.3. XBEE1 firmware 10ED



Cette expérience nous montre uniquement que le firmware n'a aucun impact sur le temps de calcul du FCS :

Moyenne : 259735.52 (0.0129s)

Ecart-type : 8501.3996 (0.4s)

Variance : 72273796 (3.6s²)

On a seulement une variance qui augmente de +0.1s² à celle avec le firmware d'origine. Devrions-nous prendre en compte cette variation sachant que les écarts entre les autres configurations sont de minimum +0.6s² ?

Tableau récapitulatif des données :

	XBEE1	XBEE1(10ED)	XBEE2	XBEE1(AT328)	XBEE1(AT2560)
Moyenne	0.0129s	0.0129s	0.0129s	0.0132s	0.0131s
Ecart-type	0.41s	0.40s	0.45s	0.237s	0.183s
Variance	3.5s²	3.6s²	4.2s²	1.127s²	0.6s²

Conclusion de l’expérience :

Pour cette expérience nous pouvons conclure que le firmware n’a pas réellement d’impact sur le rayonnement émis par le XBee. Cependant cette étude nous montre que les ATMEGA ont des pics caractéristiques très précis et fiables. Ainsi cette expérience pourrait nous permettre d’identifier quel est le microcontrôleur qui gère le XBee (Un pic à 266 441 échantillons indiquerait que c’est un ATMEGA328P qui s’occupe de la transmission des messages , tandis qu’un pic à 263 881 échantillons indiquerait que le microcontrôleur est un ATMEGA2560).

Conclusion

Durant ce projet, nous avons cherché à déterminer l'impact potentiel du matériel et du logiciel sur le champ électromagnétique émis par un objet connecté. On s'attend à ce que le signal émis par une antenne de même type soit le même, afin de le démoduler correctement au niveau de la réception. Néanmoins, il est très rare d'un point de physique d'avoir des signaux parfaitement identiques et cela dans les mêmes conditions expérimentales.

On a pu voir que de nombreux facteurs témoignaient de potentielles - et il est important de souligner le terme potentielles - caractéristiques uniques provoquées non pas par l'antenne émettrice, mais par le matériel et/ou le logiciel. En effet, les conjectures émises en conclusion des expériences ne peuvent être validées de manière absolue, mais sont néanmoins la preuve d'une réelle influence du matériel et sont selon nous des pistes à développer (le modèle de boîte noire se rapporte à notre projet).

Dans le cadre de notre projet, c'est l'antenne XBee qui a été étudiée.

A l'issue de ces expériences, nous avons pu mettre en évidence de nombreux points :

- Les signaux émis par les XBee respectent les caractéristiques temporelles et fréquentielles données par leur datasheet (première expérience)

- Le firmware ne semble pas avoir pas de réel impact sur les signaux, ni sur la variation d'amplitude

- Le SNR témoigne de la puissance d'émission de l'antenne et permet de distinguer un module XBee Pro d'un module Xbee Série S1, permet aussi de différencier les signaux (sans nécessairement les identifier) et présente un potentiel gap entre un signal émis directement via liaison série et un signal émis par un Arduino.

- Le temps de fragmentation d'un paquet transmis ainsi que le temps de calcul du FCS montre un schéma pouvant potentiellement être unique à tout un chacun des composants.

Nous aurions pu faire d'autres expériences pour nous conforter sur nos positions, néanmoins, nous n'avions pas trouvé d'autres d'idées qui pouvaient être réalisées avec nos outils (la variation de l'amplitude du signal nous a montré que le HackRF n'est pas exempt de défaut).

Ce projet a été pour nous une expérience nouvelle. Là où la plupart des projets étaient plus orientés à la conception et la réalisation, avec un objectif bien défini, le nôtre lui était tout autre. En effet, l'approche était plus expérimentale et orientée vers la recherche.

Les premières pistes de ce projet, bien qu'elles ne soient pas à négliger, se basaient sur les fuites électromagnétiques dispersées par les différents composants électroniques et informatiques d'un système. Néanmoins, avec le matériel et les compétences actuelles, il nous aurait été difficile de mener à bien ce projet de recherche et de poursuivre nos idées. Ainsi, le fait d'étudier les signaux directement par canal direct et non annexe (pas de CEM), nous permettant de chercher des potentielles caractéristiques uniques (le terme utilisé étant les clés) était donc la piste privilégiée.

La difficulté n'en était pas amoindrie. De nombreux obstacles devaient être franchis.

Tout d'abord, nous devions nous initier à la radio logicielle. Cependant, hormis dans le cadre de la démodulation de signaux et dans celui du Hacking (attaque radio, par replay etc...), il n’y a que très peu de documentation au niveau de la radio logicielle et cela reste un domaine peu accessible au grand public (matériel et conditions d'acquisition obligent). Le procédé de recherches et les expériences nous ont donc été propre.

Ce qui nous a mené à la seconde difficulté : la mise en place du protocole de recherche. La caractérisation d'un signal ou d'une empreinte propre nécessite de faire des tests qui peuvent mener ou alors à des pistes intéressantes voire concluantes, ou alors à des échecs.

Dans notre cas, tout ne nous permettait pas de développer le sujet. La rencontre de Etienne Helluy-Lafont a été pour nous d'une grande aide et nous a permis de savoir si les pistes qui avaient été développées étaient pertinentes ou non.

Annexes

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################################
# Le script permet de faire une acquisition à partir de la bibliothèque
# fournie par gnuradio
##################################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import osmosdr
import time
import wx
import os



class top_block(grc_wxgui.top_block_gui):

    def __init__(self, nom, append, central_frequency):
        grc_wxgui.top_block_gui.__init__(self, title="Top Block")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        ##################################################
        # Variables
        ##################################################
        self.slider_frequency = slider_frequency = central_frequency
        print("Fréquence : "+str(slider_frequency))
        self.samp_rate = samp_rate = 20e6
        self.frequency = frequency = 2.410e9

        ##################################################
        # Blocs fonctionnels
        ##################################################
        _slider_frequency_sizer = wx.BoxSizer(wx.VERTICAL)
        self._slider_frequency_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_slider_frequency_sizer,
            value=self.slider_frequency,
            callback=self.set_slider_frequency,
            label='slider_frequency',
            converter=forms.float_converter(),
```

```
            proportion=0,
        )
        self._slider_frequency_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_slider_frequency_sizer,
            value=self.slider_frequency,
            callback=self.set_slider_frequency,
            minimum=2.40e9,
            maximum=2.50e9,
            num_steps=100,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
        )
        self.Add(_slider_frequency_sizer)
        self.osmosdr_source_0 = osmosdr.source( args="numchan=" + str(1) +
" " + 'hack_rf=0' )
        self.osmosdr_source_0.set_sample_rate(samp_rate)
        self.osmosdr_source_0.set_center_freq(slider_frequency, 0)
        self.osmosdr_source_0.set_freq_corr(0, 0)
        self.osmosdr_source_0.set_dc_offset_mode(0, 0)
        self.osmosdr_source_0.set_iq_balance_mode(0, 0)
        self.osmosdr_source_0.set_gain_mode(False, 0)
        self.osmosdr_source_0.set_gain(0, 0)
        self.osmosdr_source_0.set_if_gain(16, 0)
        self.osmosdr_source_0.set_bb_gain(16, 0)
        self.osmosdr_source_0.set_antenna('', 0)
        self.osmosdr_source_0.set_bandwidth(0, 0)

        ## Vérification de l'existence d'un fichier d'acquisition
        path_base =
'/home/speedclocker/Documents/Polytech/IMA4/Projet/Acquisitions/'

        index_file=0

        while(os.path.isfile(path_base+nom+str(index_file))):
            index_file = index_file + 1

        self.blocks_float_to_complex_0 = blocks.float_to_complex(1)

        if (append==0):
            self.blocks_file_sink_1 =
blocks.file_sink(gr.sizeof_gr_complex*1, path_base+nom+'_PhaseMeg', True) #
Append
        elif(append==1):
            self.blocks_file_sink_1 =
blocks.file_sink(gr.sizeof_gr_complex*1,
path_base+nom+str(index_file)+'_PhaseMeg', True) # AddFile
        else:
            self.blocks_file_sink_1 =
blocks.file_sink(gr.sizeof_gr_complex*1, path_base+nom+'_PhaseMeg', False)
# Overwrite

        self.blocks_file_sink_1.set_unbuffered(False)

        if(append==0):
            self.blocks_file_sink_0 =
blocks.file_sink(gr.sizeof_gr_complex*1, path_base+nom, True) # Append
        elif(append==1):
            self.blocks_file_sink_0 =
blocks.file_sink(gr.sizeof_gr_complex*1, path_base+nom+str(index_file),
```

```
True) # AddFile
        else:
            self.blocks_file_sink_0 =
blocks.file_sink(gr.sizeof_gr_complex*1, path_base+nom, False) # Overwrite
        self.blocks_file_sink_0.set_unbuffered(False)
        self.blocks_complex_to_magphase_0 = blocks.complex_to_magphase(1)

        ##################################################
        # Connexions entre les blocs
        ##################################################
        self.connect((self.blocks_complex_to_magphase_0, 0),
(self.blocks_float_to_complex_0, 0))
        self.connect((self.blocks_complex_to_magphase_0, 1),
(self.blocks_float_to_complex_0, 1))
        self.connect((self.blocks_float_to_complex_0, 0),
(self.blocks_file_sink_1, 0))
        self.connect((self.osmosdr_source_0, 0),
(self.blocks_complex_to_magphase_0, 0))
        self.connect((self.osmosdr_source_0, 0), (self.blocks_file_sink_0,
0))

    def get_slider_frequency(self):
        return self.slider_frequency

    def set_slider_frequency(self, slider_frequency):
        self.slider_frequency = slider_frequency
        self._slider_frequency_slider.set_value(self.slider_frequency)
        self._slider_frequency_text_box.set_value(self.slider_frequency)
        self.osmosdr_source_0.set_center_freq(self.slider_frequency, 0)

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.osmosdr_source_0.set_sample_rate(self.samp_rate)

    def get_frequency(self):
        return self.frequency

    def set_frequency(self, frequency):
        self.frequency = frequency


## Afin de faire une acquisition, le script doit prendre en paramètre le
temps d'acquisition et la fréquence centrale autour du quel la bande
passante du hackrf sera établie tel que :
##  python capture.py [tmps en secs] [Nom_du_fichier]
[0:AppendOnFile/1:AddNewFile/2+:Overwrite] [Fréquence]
def main(top_block_cls=top_block, options=None):
    time_pr=int(sys.argv[1])
    print(len(sys.argv))
    if(len(sys.argv)<5):
        tb = top_block_cls("Acquisition", 1, 2.45e9)
    else:
        tb = top_block_cls(sys.argv[2], int(sys.argv[3]),
float(sys.argv[4]))

    tb.Start(True)
    #tb.Wait()
```

```
    time.sleep(time_pr)
    tb.stop()


# -*- coding: utf-8 -*-
##########################################################################
#
# Le script permet d'étudier le SNR par la méthode de glissement de fenêtre
##########################################################################
#


import scipy
import numpy
import sys
import math

import matplotlib.pyplot as plt
from collections import deque


def snr_study(name):
    tmp=scipy.fromfile(open(name+"_PhaseMeg"), dtype=scipy.complex64)

    f = open(name+"_SNR_Values","a")

    p=1
    size = tmp.__len__()
    magnitude = (numpy.real(tmp)).tolist()

    threshold=10

    P=0
    SNR_graph=[]
    SNR=[]

    win_sz=64

    if(size>=win_sz):
        for val in range(0,size-win_sz):

            if(val>=((size-win_sz)/100)*p): # Affiche l'état de l'étude
                print(str(p)+" %")
                p=p+1

            SNR_graph.append( 20*math.log((numpy.mean(magnitude[val +
win_sz/2 : val + win_sz-1]))/(numpy.mean(magnitude[val+0 : val+win_sz/2-
1])), 10) )

    size=len(SNR_graph)

    for i in range(1, size-1):
        if (SNR_graph[i]>SNR_graph[i-1] and SNR_graph[i]>SNR_graph[i+1] and
SNR_graph[i]>threshold):
            SNR.append(SNR_graph[i])


    #plt.plot(SNR);
    #plt.show();

    tmp=numpy.asarray(SNR)
    tmp.tofile(f)
```

```
# -*- coding: utf-8 -*-
##########################################################################
#######################################################################
# Ce script rassemble l'étude de variation d'amplitude et de subdivision de
paquets afin d'effectuer le traitement en post capture plus rapidement
##########################################################################
#######################################################################

import scipy
import numpy
import sys



## Calcul la moyenne
def moyenne(tab):
    moy=0
    for i in range(0,len(tab)):
        moy=moy+tab[i]

    moy=moy/len(tab)
    return moy


## La fonction effectue une étude statistique sur le temps entre l'emission
d'un morceau de paquet et d'un autre. Le résultat obtenu est en nombre
d'échantillons et est stocké dans la liste inter_time_values prises en
paramètre du signal name en paramètre
def etudeSubdivAmpli(name, inter_time_values, liste_temps):

    amppha=scipy.fromfile(open(name+"_PhaseMeg"), dtype=scipy.complex64)

    p=1
    size=amppha.__len__()

    ground=False

    threshold_0=0.05 # Threshold
    threshold_1=0.4
    max_transition_time=50 #O
    tps_variation=0
    transition=0
################################
    threshold=0.30
    debut=1
    inter_time=0
    min_limit=150000
    max_limit=350000
    accept_range=1./10. ## Permet de mettre un seuil d'intervalle
d'acceptation
    signalFnd=False

    for val in range(debut,size):
        if(val>=((size-debut)/100)*p):
            print(str(p)+" %")
            p=p+1

        if(amppha[val].real>threshold):
```

```
            if(ground==True):
                if(signalFnd==True):
                    if(len(inter_time_values)>0):
                        moy=moyenne(inter_time_values)
                        if(inter_time >= moy-accept_range*moy and
inter_time <= moy+accept_range*moy):
                            print("Yes "+str(inter_time))
                            inter_time_values.append(inter_time)
                        inter_time=0
                    elif(inter_time>=min_limit and inter_time<=max_limit):
                        inter_time_values.append(inter_time)
                else:
                    signalFnd=True
                ground=False
        else:
            ground=True
            if(signalFnd==True):
                inter_time=inter_time+1


        if(amppha[val].real>threshold_0 and amppha[val].real<threshold_1):
# Phase de transition du signal
            if(transition==False):
                transition=True
            tps_variation=tps_variation+1

        elif(transition==True and amppha[val].real>threshold_1):
            if(amppha[val].real>=amppha[val-1].real-amppha[val-1].real*0.05
and amppha[val].real<=amppha[val-1].real+amppha[val-1].real*0.05): #
L'amplitude du signal est stable
                if(tps_variation < max_transition_time):
                    liste_temps.append(tps_variation)

                transition=False
                tps_variation=0
            else:
                tps_variation=tps_variation+1
        else:
            transition=False
            tps_variation=0

    if(len(inter_time_values)>0):
        print("Moyenne_inter"+ str(moyenne(inter_time_values)))

    if(len(liste_temps)>0):
        print("Moyenne_liste"+ str(moyenne(liste_temps)))




def main():
    liste_temps=[]
    inter_time_values=[]
    try:
        name=sys.argv[1]
        etudeSubdivAmpli(sys.argv[1], inter_time_values, liste_temps)
    except IndexError:
        print("Donnez le nom du fichier\n")
        return -1
    except IOError:
```

```
        print("Le fichier n'existe pas\n")
        return -1

    if(len(sys.argv)>2 and sys.argv[2]=="a"):
        f = open(name+"_inter_time","a")
      f2 = open(name+"_variation_mag","a")
    else:
        f = open(name+"_inter_time","w+")
      f2 = open(name+"_variation_mag","w+")

    tmp=numpy.asarray(inter_time_values)
    tmp.tofile(f, "\n", "%s")
    f.write("\n")
    tmp2=numpy.asarray(liste_temps)
    tmp2.tofile(f2, "\n", "%s")
    f2.write("\n")



if __name__=='__main__':
    main()
```