المدرسة الوطنية للعلوم التطبيقية – القنيطرة
ECOLE NATIONALE DES SCIENCES APPLIQUEES - KENITRA

Analyseur de spectre

Mémoire

Yassine Ouamer

Encadrant : Moulay Taïb Belghiti

Du principe de fonctionnement jusqu'à la réalisation Mai 2013

Table des matières

Remerciements :

En préambule à ce mémoire, je souhaiterai adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire.

Je tiens à remercier sincèrement mon père, qui s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Remerciements à Monsieur Moulay Taïb Belghiti, encadrant de ce TIPE, pour ses conseils avisés notamment concernant le choix du thème et sa disponibilité.

J'exprime ma gratitude à tous les consultants et internautes rencontrés lors des recherches effectuées et qui ont accepté de répondre à mes questions.

Je n'oublie pas ma mère pour sa contribution, son soutien et sa patience.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches, mon frère Mohammed, mes sœurs Mariame et Zineb, et amis, qui m'ont toujours soutenue et encouragée au cours de la réalisation de ce mémoire.

Merci à tous et à toutes.

Présentation

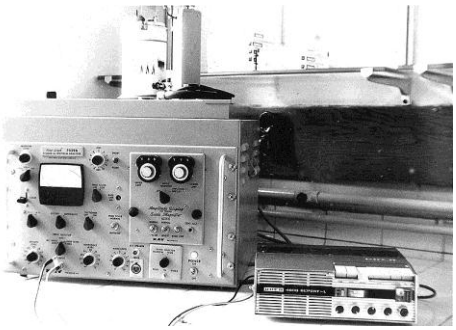
1. L'analyseur de spectre

L'analyseur de spectre (*ou spectromètre*) est un appareil destiné à identifier les différents fréquences constituant un signal suivant leurs amplitudes respectives. Le signal d'entrée d'un analyseur de spectre est électrique, cependant, la composition spectrale des autres signaux tels que l'acoustique, les ondes de pression et les ondes lumineuses peuvent être considérées par le biais d'un transducteur.

En analysant le spectre d'un signal électrique ; la fréquence dominante, l'énergie, la distorsion, les harmoniques, la bande passante, et d'autres composants spectrales du signal peuvent être observé, ce qui est pas le cas dans l'espace-temps du signal. Ces paramètres sont très utiles lors de la caractérisation des appareils électronique, tel que les émetteurs sans-fil.

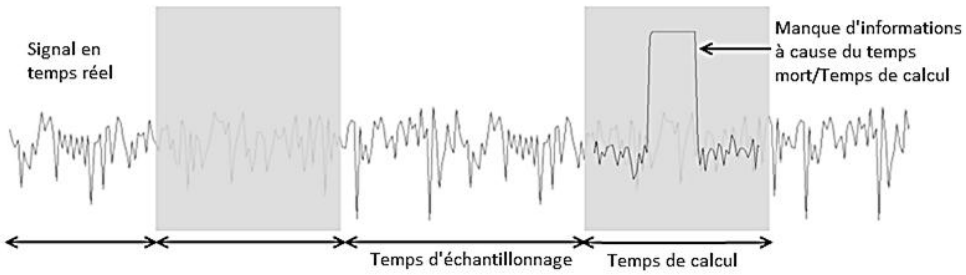
2. Différents principes de fonctionnement

Il existe plusieurs principes de fonctionnements développé au fil du temps et amélioré avec les technologies disponibles, dont :



- **Flèche à l'écoute :** *(swept-tuned)* ce sont les premiers analyseurs de spectre des années 1960. Il convertit une partie du spectre signal d'entrée en une fréquence centrale d'un filtre passe-bande en balayant l'oscillateur de la tension demandée à travers la gamme de fréquences de l'instrument.
- **FFT :** Suite à la découverte de la transformé de Fourier rapide en 1965, les premiers analyseurs FFT ont été introduits en 1967. En faisant des calculs numériques, il est nécessaire d'échantillonner le signal d'entrée avec une fréquence d'échantillonnage f_s qui est au moins le double de la bande passante du signal. Une transformée de Fourier produira, alors, un spectre contenant toutes les fréquences allant de zéro à $f_s/2$. Cela peut imposer des exigences considérables sur le convertisseur analogique-numérique demandé et sur la puissance de traitement pour le calcul. Ce qui rend les analyseurs FFT limités dans la gamme de fréquence.

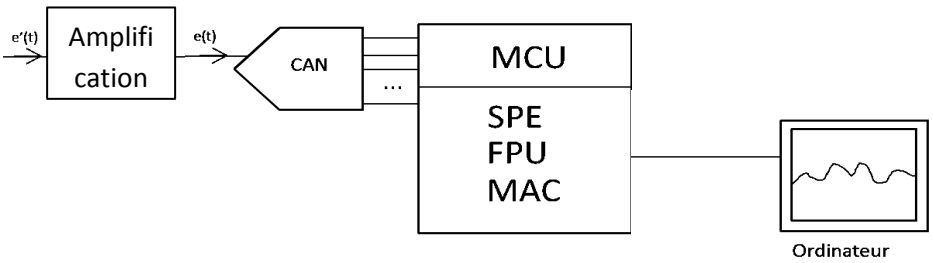
- **FFT en temps réel :** *(Real-Time FFT)* Les analyseurs de spectres modernes sont maintenant presque exclusivement en temps réels, en donnant une amélioration significative du temps de balayage. Cependant, il en reste toujours du temps mort au moment de calcul.



3. Principe opté et synoptique

Ayant cité les différents principes de fonctionnement, le principe opté dans ce sujet est l'analyseur FFT à temps réel.

Grâce à sa fiabilité, cet analyseur de spectre est constitué de plusieurs composants représenté dans le schéma suivant :



Ce principe est basé sur la conversion d'un signal à analyser en valeurs numériques. Ces valeurs qui deviennent des données seront traitées par microcontrôleur. Par la suite, le résultat sera envoyé à l'ordinateur pour être afficher. Le traitement effectué consiste à calculer la FFT à partir des échantillons reçu du convertisseur analogique-numérique.

Ce principe est partagé entre 3 parties principales : Une partie mathématique, une partie électronique et une partie logicielle.

Le coté mathématique s'intéresse au développement de la FFT, tandis que la partie électronique comportera une interface qui permettra l'amplification et l'adaptation du signal. La partie logicielle viennent après pour mettre en œuvre le coté mathématique.

Chapitre 1

Aspect Mathématique

1. Aperçu sur la transformée de Fourier

La transformation de Fourier $\mathcal{F}$ est une opération qui transforme une fonction intégrable sur $\mathbb{R}$ en une autre fonction, décrivant le spectre fréquentiel de cette dernière. Si f est une fonction intégrable sur$\mathbb{R}$, sa transformée de Fourier est :

$$\mathcal{F}(f) : \nu \longrightarrow \hat{f}(\nu) = \int_{-\infty}^{+\infty} f(x) e^{2i\pi\nu x}\, dx$$

Équation 1 : Transformation de Fourier

Néanmoins, son utilisation dans le calcul numérique s'avère impossible. C'est pourquoi, une approximation par des valeurs discrètes a été développée.

Note :
On utilisera par la suite, la notation j pour désigner le nombre complexe.

2. Transformée de Fourier Discrète (DFT)

Appelé en anglais *Discrete Fourier Transform*(DFT), c'est un outil mathématique de traitement de signal numérique qui est l'équivalent discret de la Transformé de Fourier continue.

En approchant l'intégrale par une somme d'aires de rectangles de durée T_e et en limitant la durée d'intégration dans l'intervalle $[0, (N-1)T_e]$. Si x la fonction du signal et X sa transformée de Fourier, on obtient :

$$X(f) \approx T_e \sum_{n=0}^{N-1} x(nT_e) e^{-2j\pi f n T_e}$$

Pour des fréquences $f_k = {kf_e}/{N}$, on aura:

$$X(f_k) \approx T_e \sum_{n=0}^{N-1} x(nT_e) e^{-2j\pi \frac{nk}{N} f_e T_e} \approx T_e \sum_{n=0}^{N-1} x(nT_e) e^{-2j\pi \frac{nk}{N}}$$

On la notera par :

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-2j\pi \frac{nk}{N}}$$

Équation 2 : Transformée de Fourier Discrète

Remarque :

Cette définition n'est pas unique, on peut la normer par $1/N$ ou $1/\sqrt{N}$, le but étant toujours de retrouver le signal original.

3. Lien entre la transformée de Fourier et la DFT

Soit $x(t)$ un signal analogique continue.

D'abord, on échantillonne $x(t)$ à $f_e = 1/T_e$.

$$x(t) \longrightarrow x_e(t) = \sum_{n=-\infty}^{+\infty} x(nT_e)\delta(t - nT_e) = x(t)P(t)$$

Où $P(t)$ est la fonction « peine » :

$$P(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT_e) \rightleftharpoons P(f) = \frac{1}{T_e} \sum_{n=-\infty}^{+\infty} \delta\left(f - \frac{n}{T_e}\right)$$

Équation 3 : Fonction peine

Remarque :

L'échantillonnage rend le spectre périodique.

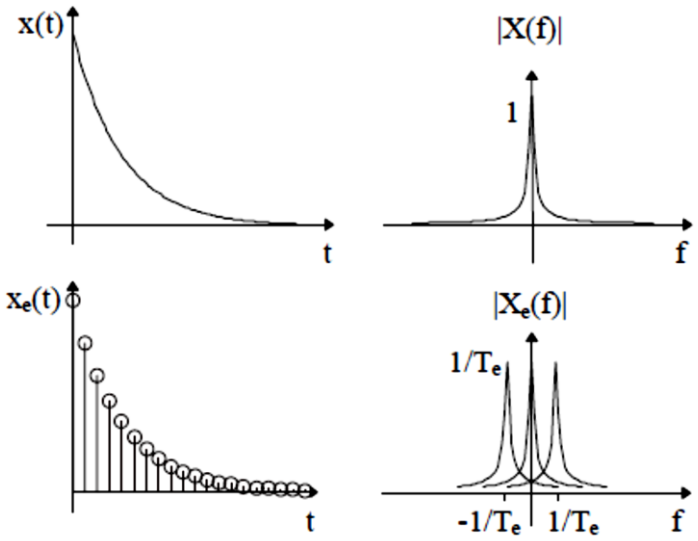


Figure 1 : Signal continue vs signal discret

Ensuite on tronque la suite $x_e(nT_e)$ en ne conservant qu'un nombre fini N de termes pour obtenir le signal $x_{tr}(t)$ formé par les échantillons $x(0) \dots x((N-1)T_e)$:

$$x_{tr}(t) = x_e(t)F(t) = \sum_{n=0}^{N-1} x(nT_e)\delta(t - nT_e) = x(t)P(t)F(t)$$

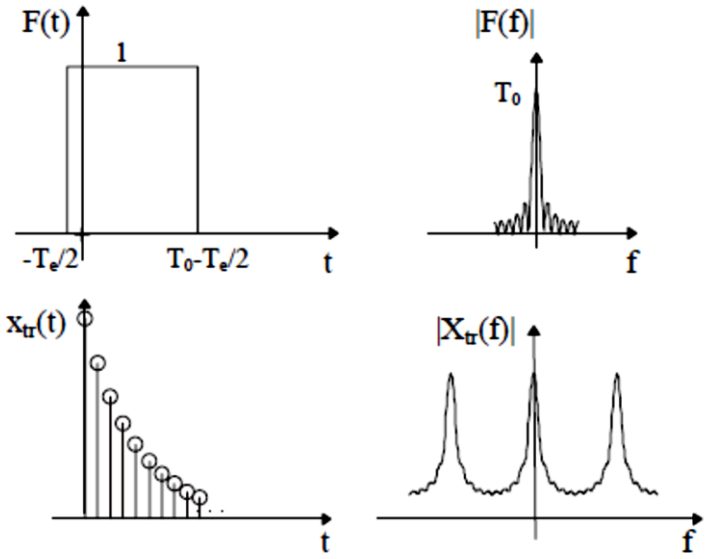


Figure 2 : Fonction fenêtre et son effet sur le signal

$F(t)$ est une fonction fenêtre de durée NT_e :

$$F(t) = \begin{cases} 1 & si \quad t \in \left[-\frac{T_e}{2}, T_0 - \frac{T_e}{2}\right] \\ 0 & sinon \end{cases} \text{ Avec } T_0 = NT_e$$

Équation 4 : Fonction fenêtre

Finalement, on échantillonne $X_{tr}(f)$ à $1/T_0$, on obtient alors N valeurs différentes espacées de $1/T_0$ comprise entre 0 et $1/T_e$ (car $T_0 = NT_e$). Cette dernière opération rend périodique la « fonction » dans le temps. Appelons $x_c(t)$ la fonction résultante.

$$X_c(f) = X_{tr}(f) \sum_{n=-\infty}^{+\infty} \delta\left(f - \frac{n}{T_0}\right) = \sum_{n=-\infty}^{+\infty} X_{tr}\left(\frac{n}{T_0}\right) \delta\left(f - \frac{n}{T_0}\right)$$

$$= \sum_{n=-\infty}^{+\infty} \left(\sum_{k=0}^{N-1} x(kT_e) e^{-j2\pi \frac{nk}{N}} \right) \delta\left(f - \frac{n}{T_0}\right)$$

$x_c(t)$ et $X_c(f)$ sont deux distributions échantillonnées reliées par la transformation de Fourier.

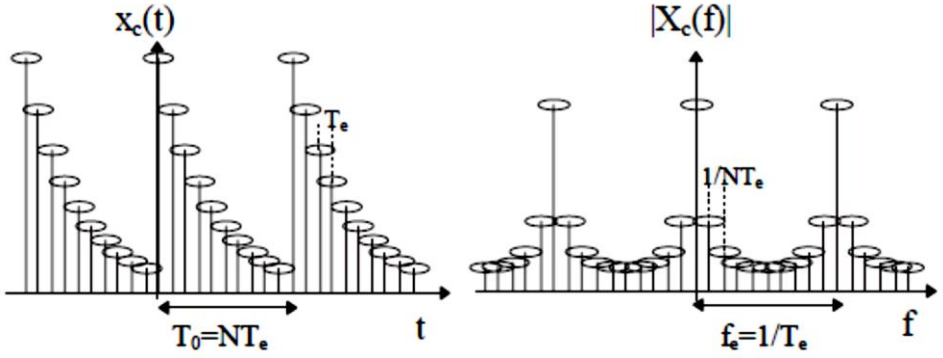


Figure 3 : Signal discret final et sa DFT

On obtient donc une correspondance entre N points dans l'espace temporel $x_c(nT_e)$ et N points dans l'espace fréquentiel $X_c(n/T_0)$, pour n entre 0 et $N-1$. De plus :

$$x_c(nT_e) = T_0 x(nT_e) \quad \text{pour} \quad n \in [0, N-1]$$

$$X_c\left(\frac{k}{T_0}\right) = \sum_{n=0}^{N-1} x(nT_e) e^{-j2\pi \frac{nk}{N}}$$

C'est-à-dire que la suite $X_c(k) = X_c(k/T_0)$ est précisément la DFT de la suite $x(n) = x(nT_e)$ et on peut conclure :

$$F_0 = \frac{k}{NT_0}$$

Équation 5 : Fréquence continue et fréquence discrète

4. Transformée de Fourier rapide *(FFT)*

La Transformée de Fourier Rapide *(Fast Fourier Transform)* est un algorithme permettant de calculer la DFT en réduisant le nombre d'opérations et en particulier le nombre de multiplications à effectuer. Elle est originalement développée par Gauss en 1805. Pourtant, elle n'est reconnue que dans l'année 1965 par Colley et Tukey et ce, à cause du retard technologique à cette époque.

On pose $W_N = e^{-j\frac{2\pi}{N}}$.

Cet algorithme exploite les symétries de W_N, principalement :

- La symétrie du conjugué complexe : $W_N^{k(N-n)} = e^{-j\frac{2\pi}{N}k(N-n)} = e^{-j2\pi k + \frac{j2\pi kn}{N}} = e^{-j2\pi k} e^{\frac{j2\pi kn}{N}} = W_N^{-nk} = (W_N^{nk})^*$
- La périodicité de n et k : $W_N^{kn} = W_N^{(k+N)n} = W_N^{(n+N)k}$:

Démonstration :

$$W_N^{kn} = e^{-\frac{j2\pi kn}{N}} = e^{-j2\pi k} e^{-\frac{j2\pi kn}{N}}$$
$$= e^{-j2\pi k - \frac{j2\pi kn}{N}} = e^{-j\frac{2\pi}{N}k(N+n)} = W_N^{(n+N)k} \text{ et de}$$

même manière qu'on démontre la deuxième.

L'élaboration de l'algorithme FFT avec enlacement temporel consiste à utiliser les valeurs d'entrés d'une façon désordonné en assumant $N = 2^m$ $(\forall m \in \mathbb{N})$ et en séparant $x[n]$ en des suites extraites d'indices pairs $(n = 2r)$ et impairs $(n = 2r + 1)$, d'où :

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} = \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_N^{k(2r+1)}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_N^{2rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_N^{2rk}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r] (W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] (W_N^2)^{rk}$$

Mais : $W_N^2 = e^{-\frac{j2\pi}{N}2} = e^{-\frac{j2\pi}{\frac{N}{2}}} = W_{\frac{N}{2}}$, donc :

$$X[k] = \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r] W_{N/2}^{rk}}_{\frac{N}{2}\text{ DFT des échantillons pairs } X_e[k]} + W_N^k \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_{N/2}^{rk}}_{\frac{N}{2}\text{ DFT des échantillons impairs } X_o[k]}$$

Chose qui donne une somme de deux DFT de $\frac{N}{2}$ points : $X[k] = X_e[k] + W_N^k X_o[k]$

On prend l'exemple de $N = 8$:

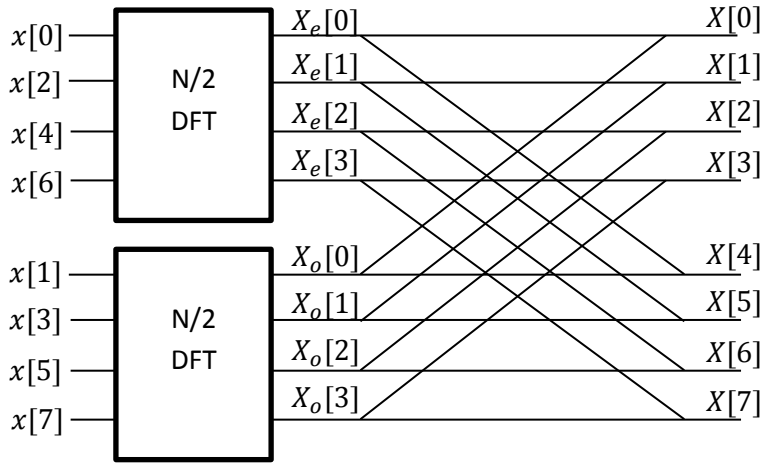


Figure 4 : Exemple de DFT à 8 points séparé en deux DFTs

Les valeurs $x[0], x[1], \ldots, x[7]$ sont rassemblées en 2 groupes : un groupe formé de valeurs d'indices pairs, et un autre d'indice impairs. Pour chaque groupe on effectue une DFT de $N/2$ soit 4 points et on combine les résultats pour obtenir une de N soit 8 points.

On continue la séparation de chacune des deux DFT de $N/2$ points en d'autres DFT de $N/4$ et ainsi de suite. On pourra aller donc de $N/2, N/4, \ldots, N/2^{p-1}, N/2^p = 1$ sachant que $p = \log_2 N$. Il représente le nombre de fois qu'on divise la DFT.

Revenant à l'exemple précèdent :

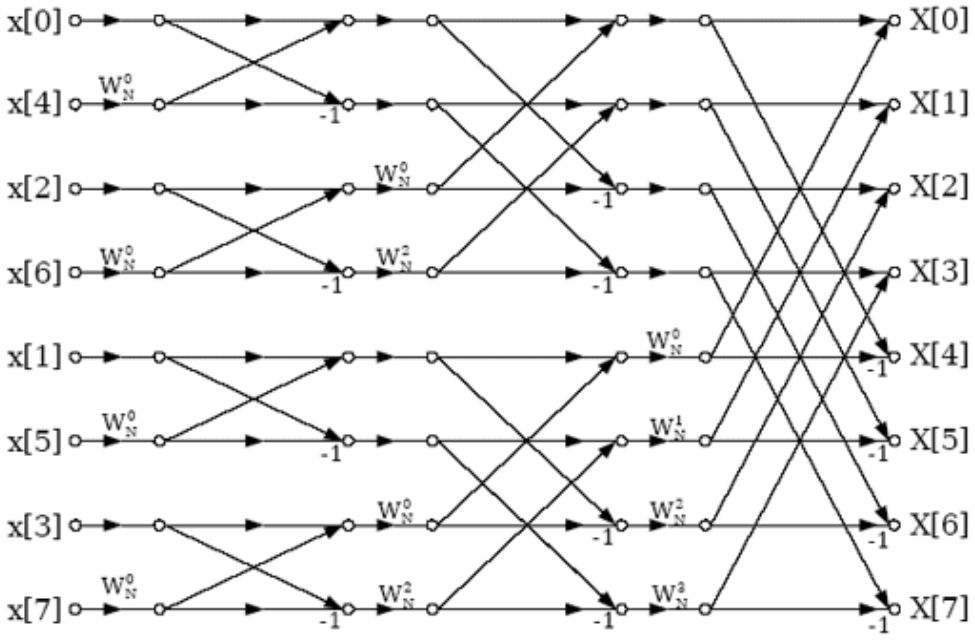


Figure 5 : Représentation de la FFT

Remarque :

Sur ce schéma les valeurs $x[n]$ sont désordonnées, alors que celles de sortie $X[k]$ sont dans leurs ordres naturels. En effet, cet algorithme de FFT s'appelle FFT avec entrelacement temporel. Cependant, il existe un autre algorithme appelé FFT avec entrelacement fréquentiel.

Une DFT de 2 points est nommée « Papillon élémentaire » (en anglais butterfly). Elle est composée de deux entrées et deux sorties. Leurs associations constituent la DFT totale présenté dans la Figure 6 :

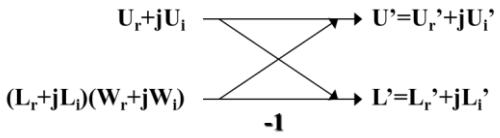


Figure 6 : Papillon élémentaire

Explicitons les valeurs de $(L_r + jL_i)(W_r + jW_i)$, U' et L' :

$$\begin{aligned}(L_r + jL_i)(W_r + jW_i) &= L_rW_r + jL_rW_r + jL_iW_r - L_iW_i \\ &= (L_rW_r - L_iW_i) + j(L_rW_r + L_iW_r)\end{aligned}$$

$$U' = (L_r W_r - L_i W_i) + j(L_r W_r + L_i W_r) + U_r + jU_i$$
$$= (L_r W_r - L_i W_i + U_r) + j(L_r W_r + L_i W_r + U_i)$$

$$L' = (U_r + jU_i) - [(L_r W_r - L_i W_i) + j(L_r W_r + L_i W_r)]$$
$$= (U_r - L_r W_r + L_i W_i) + j(U_i - L_r W_r - L_i W_r)$$

Équation 6 : Formule de calcul d'un seul papillon élémentaire

Remarquons aussi que l'ordre entrelacé des valeurs d'entrées est obtenues à partir de l'ordre naturel dont on retourne l'arrangement des bits pour obtenir l'indice correspondant dans l'ordre entrelacé.

Par exemple pour $N = 8$:

Tableau 1 : Inversement des bits

Indices ordre naturel	Représentation binaire	Représentation retournée	Indices ordre entrelacé
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

5. Comparaison entre la DFT et la FFT

Pour calculer la DFT directement, on fera appelle à N fois la multiplication pour chaque valeur résultante, et on a N valeurs résultantes. Ce qui fera $\boldsymbol{N^2}$ **opérations**.

En ce qui concerne la FFT, on a :

$$1 : \frac{\mathrm{N}}{2} \longrightarrow 2\left(\frac{\mathrm{N}}{2}\right)^2 + \mathrm{N} = \frac{\mathrm{N}^2}{2} + \mathrm{N}$$
$$2 : \frac{\mathrm{N}}{4} \longrightarrow 2\left(2\left(\frac{\mathrm{N}}{4}\right)^2 + \frac{\mathrm{N}}{2}\right) + \mathrm{N} = \frac{\mathrm{N}^2}{4} + 2\mathrm{N}$$
$$3 : \frac{\mathrm{N}}{8} \longrightarrow 2\left(2\left(\left(\frac{\mathrm{N}}{8}\right)^2 + \frac{\mathrm{N}}{4}\right) + \frac{\mathrm{N}}{2}\right) + \mathrm{N} = \frac{\mathrm{N}^2}{8} + 3\mathrm{N}$$
$$\mathrm{p} : \frac{\mathrm{N}}{2^{\mathrm{p}}} \longrightarrow \frac{\mathrm{N}^2}{2^{\mathrm{p}}} + \mathrm{pN} = \frac{\mathbf{N^2}}{\mathbf{N}} + \mathbf{N}\log_2 \mathbf{N}$$
$$\approx \mathbf{O}(\mathbf{N}\log_2 \mathbf{N})$$

Figure 7 : Nombre d'opération de la FFT

DFT se calcule dans l'ordre de N^2 opérations par contre la FFT réduit l'ordre à $N \log_2 N$.
On peut distinguer la différence quand N est suffisamment grand, par exemple :

Tableau 2 : Nombre d'opérations pour chaque méthode

$\boldsymbol{N}$	$\mathbf{1000}$	$\mathbf{10^6}$	$\mathbf{10^9}$
N^2	10^6	10^{12}	10^{18}
$N \log_2 N$	10^4	$20 \cdot 10^6$	$30 \cdot 10^9$

Si une opération se fait dans une durée $1ns$ pour $N = 10^9$ par exemple :

- DFT : $10^8 ns \sim 31.2\ annés$
- FFT : $30 \cdot 10^9 ns \sim 30s$

Chapitre 2

Aspect électronique

Après avoir étudié le développement et la fiabilité de la FFT par rapport à la formule classique de la DFT, on s'intéressera dans ce chapitre, à la partie électronique du sujet. Il inclura l'unité de traitement, l'amplification du signal et l'alimentation.

1. Unité de traitement

L'unité de traitement est un composant très essentiel pour effectuer des calculs. Son temps de calcul dépend de ses performances. L'unité de traitement utilisée ici est un microcontrôleur PXS2010 d'architecture Power™. Il dispose de plusieurs modules :

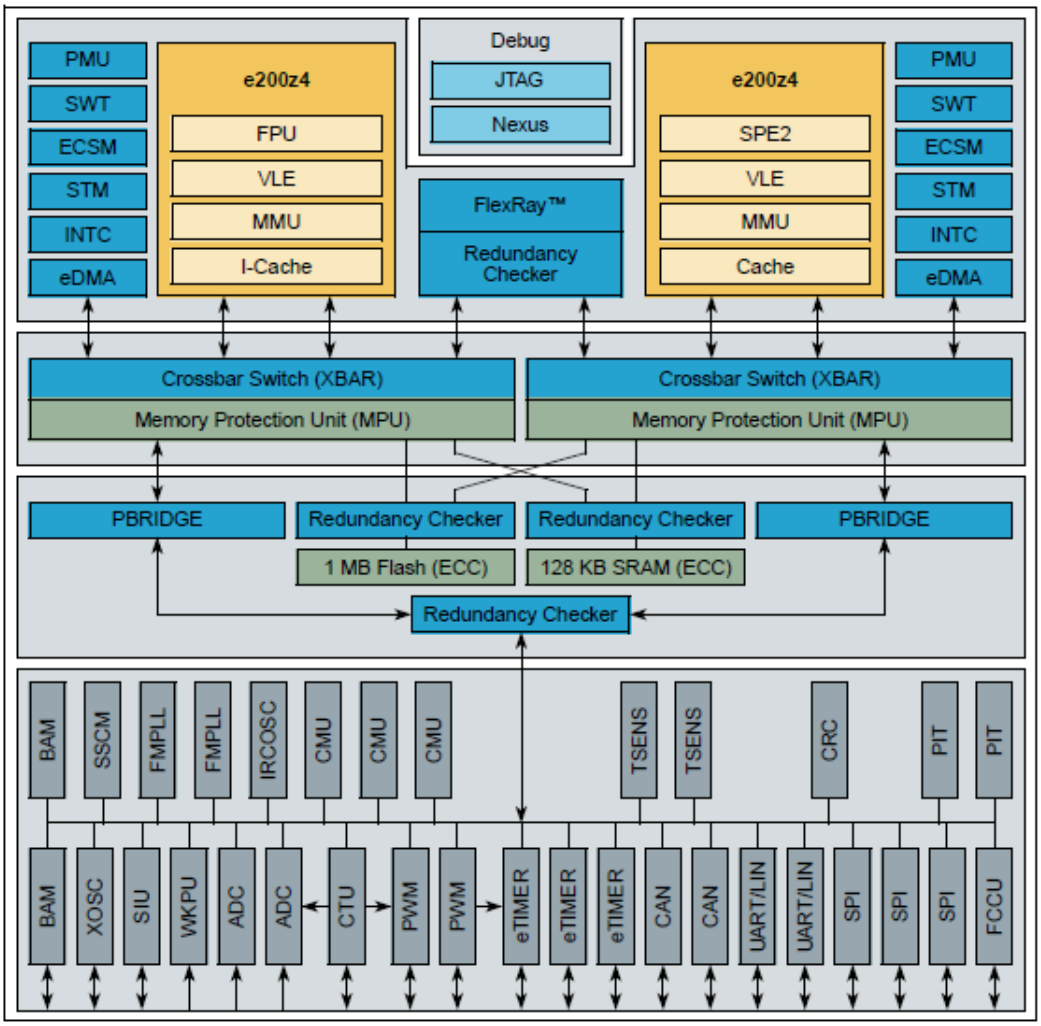


Figure 8 : Modules de PXS20

On peut remarquer dans la Figure 8 que PXS20 dispose de deux noyaux type e200z4 qui peuvent fonctionner indépendamment *(Decoupled operation)* ou exécuter les mêmes instructions en même temps *(Lock-step operation),* avec une vitesse d'exécution qui varie entre 0 et 120MHz et qui peut aller vers 240 MIPS (Million d'instructions par seconde). Il dispose aussi d'une mémoire flash de 1Mo et d'une RAM de 128Ko. Les modules utilisés sont :

- **Periodic Interrupt Timer** *(PIT) :* c'est un module qui génère une impulsion à chaque fois que le temps, précisé au début, est épuisé.
- **UART :** Est le module responsable de la communication du port série.
- **Convertisseur analogique numérique** *(ADC)* **:** Il permet la conversion du signal analogique en valeurs numériques. Composé de 12 canaux, il assure ainsi une conversion 12 signaux en même temps.

2. Convertisseur analogique numérique

Un convertisseur analogique-numérique *(ADC ou A/D)* est un composant qui convertit une grandeur analogique (notamment une tension) en une valeur numérique (codée sur plusieurs bits). L'opération inverse est effectué par un convertisseur numérique-analogique *(DAC)*.

Il existe plusieurs techniques pour convertir un signal analogique en un signal numérique (Convertisseur à simple rampe, à double rampe, sigma delta, approximation successive …), mais on s'intéressera particulièrement à l'utilisation de ce composant plutôt que son principe de fonctionnement.

L'ADC est caractérisé principalement par une résolution qui indique le nombre de valeurs discrètes qu'il peut produire au cours de la plage des valeurs analogiques. Celles-ci sont généralement stockées dans des registres sous une forme binaire. Par conséquent, le nombre de valeurs discrètes disponibles est une puissance de 2. Par exemple, un ADC avec une résolution de 8 bits permet de coder une entrée analogique en 256 niveaux différents ($2^8 = 256$). Les valeurs sont comprise entre 0 et 255 si c'est un entier non signé, ou entre -128 à 127 s'il s'agit d'un entier signé.

La résolution peut également être définie électriquement et exprimée en volts. Le minimum changement opéré sur de la tension affectant le changement de la valeur du code de la sortie est appelé le bit le moins significatif *(least significant bit LSB)* de la tension. La résolution Q est égale à la tension LSB :

$$Q = \frac{V_{refHi} - V_{refLow}}{2^M - 1}$$

Équation 7 : Résolution de l'ADC

Où V_{refHi} et V_{refLow} sont les extrêmes supérieurs et inférieurs, respectivement, des tensions qui peuvent être codées. Et M est la résolution de l'ADC en bits.

Certains premiers ADC avaient une réponse logarithmique. Hormis ; la plupart sont des types linéaires.

Le type linéaire est défini par la relation linéaire reliant la plage des valeurs d'entrée et la valeur de sortie. Ainsi :

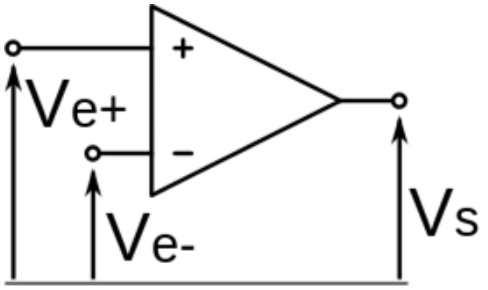
$$V_e = N \cdot Q$$

Équation 8 : Tension calculé

Exemple et pratique :

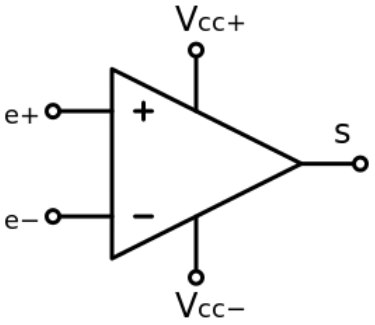
L'ADC utilisé dans ce sujet est de type linéaire. Sa plage de mesure varie de 0 à 5V. sa résolution est de 12bits (d'où 4095 niveaux) Donc $Q = 1.22mV$ *d'erreur* $\Delta E = 0.61mV$.

3. Amplificateur opérationnel



Aussi dénommé AOP, c'est un amplificateur électronique qui amplifie une différence de potentiel électrique présente à ses entrées. Il a été initialement conçu pour effectuer des opérations mathématiques dans les calculateurs analogiques. Ces derniers permettent la modélisation des opérations mathématiques de base entre autre l'addition, la soustraction, l'intégration, la dérivation et d'autres. En outre, l'amplificateur opérationnel est utilisé dans bien d'autres applications comme la commande des moteurs, la régulation d'une tension, les sources de courants ou encore les oscillateurs.

3.1. Brochage



Un AOP dispose typiquement de deux entrées, deux broches d'alimentation et une sortie. L'entrée notée e+ est dite non-inverseuse tandis que l'entrée e- est dite inverseuse. Par correspondance avec leurs rôles respectifs dans les relations entrée/sortie de l'amplificateur. La différence de potentiel entre ces deux entrées est appelée tension différentielle d'entrée ε idéalement égale à 0.

La broche d'alimentation positive repérée V_{cc+} est parfois aussi appelée V_{DD} ou V_{cc}. La broche d'alimentation négative repérée V_{cc-} est parfois aussi appelée V_{ss} ou V_{ee}.

Selon les applications, l'AOP peut aussi être doté de deux broches pour la compensation d'offset ainsi d'une broche pour le réglage de la compensation fréquentielle.

3.2. Amplificateur réel

Bien que l'AOP permette d'effectuer des calculs analogiques, il présente des défauts majeurs : présence d'un offset à l'entrée, l'influence de la tension du mode commun sur la tension de sortie…

- **Gain différentiel et de mode commun :** Le gain différentiel G_{diff} d'un AOP est fini et varie en fonction de la fréquence. Pour un AOP compensé, la variation de la fréquence du gain différentiel peut être assimilée à celle d'un système passe-bas de premier ordre.

- **Tension de décalage et courants d'entrées :** Lorsqu'un amplificateur opérationnel ne reçoit aucun signal sur ses entrées, il subsiste généralement une tension continue de décalage de la tension de sortie vis-à-vis le zéro. Ce décalage (ou offset) provient de deux phénomènes : la tension de décalage propre aux circuits internes de l'AOP d'une part, et l'influence des courants de polarisation de la paire différentielle des transistors d'entrée sur le circuit extérieur d'autre part.

- **Vitesse de balayage :** (ou *Slew rate*) Elle représente la vitesse de variation maximale de tension que peut produire un amplificateur. Lorsque la vitesse de variation du signal de sortie d'un amplificateur est supérieure à sa vitesse de balayage, sa tension de sortie est une droite.

3.3. Choix d'amplificateur

Les AOP en disposition sont : LT1097, OP42 et LF351. Tableau 3 résume les caractéristiques de chacun d'eux afin de faciliter le choix.

Tableau 3 : Caractéristiques des AOP disponibles

AOP	**LT1097**	**OP42**	**LF351**
Fournisseur	Linear Technology	Analog Device	Texas Instrument
Tensions d'alimentation (V)	$\pm 1.2 \sim 20$	$\pm 8 \sim 20$	$\pm 3 \sim 16$
Tension de décalage	$10\mu V$	$1.5mV$	$3mV$
Vitesse de balayage (V/µs)	0.2	50	16
Fréquence du gain différentiel	$700kHz$	$10MHz$	$4MHz$

L'alimentation qu'on peut fournir à un amplificateur est $\pm 2.5V$ ou $\pm 7.5V$, chose qui désavantage l'OP42. L'utilisation de l'amplificateur nous permettra d'amplifier le signal provenant du microphone. Donc un signal présentant beaucoup de variations impose une

vitesse de balayage importante et une tension de décalage très minime. C'est pourquoi, le choix LT1097 est préférable.

Pour mieux visualiser l'effet de la fréquence du gain différentiel, on fera appelle à une simulation pour chaque AOP. L'application utilisée ici est *Altium Designer.*

Partant du schéma ci-dessous :

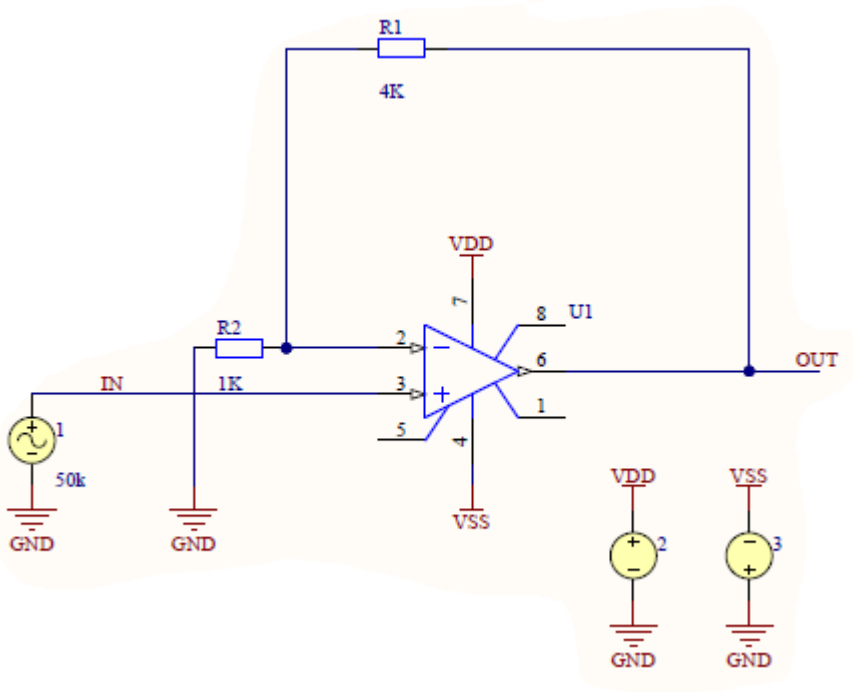


Figure 9 : Montage amplificateur non inverseur

C'est un montage amplificateur non-inverseur d'un gain égale à $\frac{R1}{R2} + 1 = 5$. L'AOP est alimenté par $\pm 15V$, c'est-à-dire $VDD = 15V$ et $VSS = 15V$. Le signal d'entrée est un signal sinusoïdal d'amplitude $1mV$. Pour LT1097 et LF351, on varie la fréquence du signal entre $1kHz$ à $100kHz$, D'où le résultat suivant :

	LT1097					**LF351**				
Fréquence (kHz)	**1**	**10**	**20**	**50**	**100**	**1**	**10**	**20**	**50**	**100**
Signal d'entrée (mV)	1					1				
Signal de sortie (mV)	5	5	4.9	4.6	3.9	5	5	5	5	4.9
Gain expérimental	5	5	4.9	4.6	3.9	5	5	5	5	4.9
Gain idéal	5					5				

D'après les résultats de la simulation, on voit que LF351 ne commence à perdre du gain que jusqu'à la fréquence 100kHz, alors que LT1097 a atteint sa limite en fréquence 20kHz. Ce qui pose pas problème car la fréquence du signal audio ne dépasse pas les 20kHz.

Conclusion :

L'amplificateur opérationnel adéquat à l'utilisation est LT1097, due à ces performances acceptable voire vitesse de balayage (0.2V/µs), une tension de décalage très faible (10µV), une adaptation avec l'alimentation disponible ±7.5V et sa fréquence du gain différentiel contenant les fréquences audibles.

4. Interface électronique

Le signal reçu du microphone s'étends généralement dans l'intervalle de $[-3mV, 3mV]$, or l'ADC dont nous disposons est référencé à une tension de 5V ce qui implique le signal doit être dans l'intervalle. Pour cela, il faut amplifier le signal à la plage de 0 à 5V(Soit 2.5V crête à crête par rapport +2.5V).

Les différents montages dont on aura besoin sont :

Montage amplificateur non inverseur :

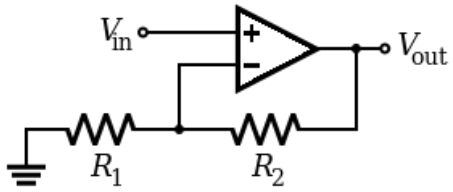


Figure 10 : Montage amplificateur non inverseur

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}$$

Équation 9 : Tension de sortie vs d'entrée

$$G = 1 + \frac{R_2}{R_1}$$

Équation 10 : Gain en tension

Montage soustracteur :

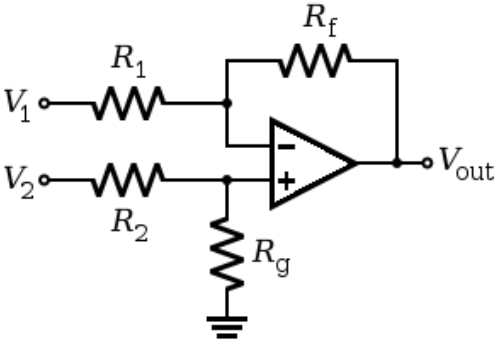


Figure 11 : Montage soustracteur

$$V_s = V_2 \frac{(R_1 + R_f)R_g}{(R_g + R_2)R_1} - V_1 \frac{R_f}{R_1}$$

Équation 11 : Tension sortie vs tensions d'entrées

$$G_1 = \frac{R_f}{R_1}$$

Équation 12 : Gain en tension V_1

$$G_2 = \frac{(R_1 + R_f)R_g}{(R_g + R_2)R_1}$$

Équation 13 : Le gain en tension V_2

La Figure 12 représente l'association de deux montages amplificateurs non inverseur et un gain multiplieur $G = G_1 \cdot G_2$:

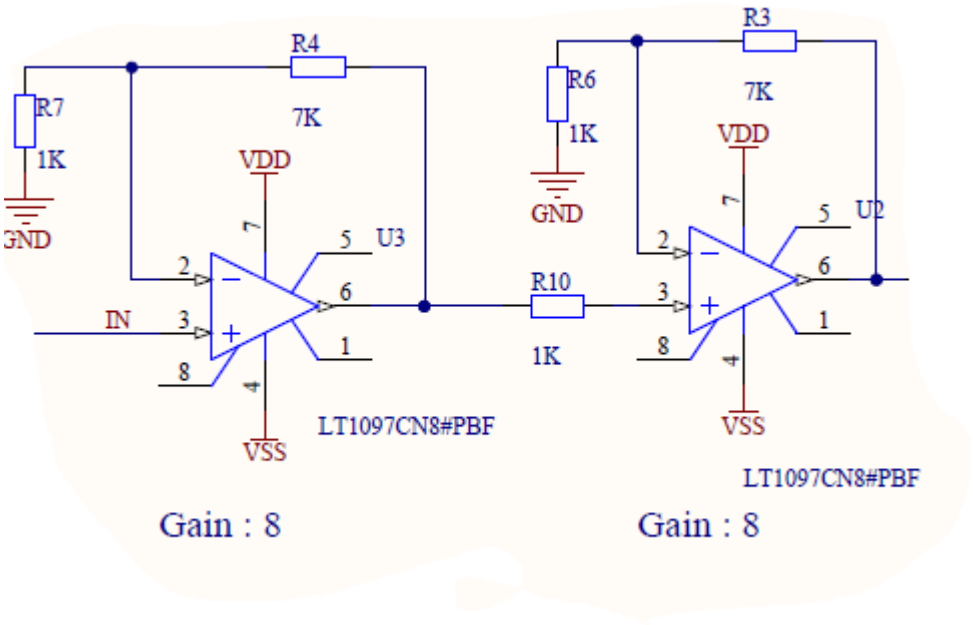


Figure 12 : Schéma d'amplification

Pour effectuer un décalage de signal vers le niveau +2.5V, on fait appel à un montage soustracteur. Sachant que VCC est une source de tension continue 5V :

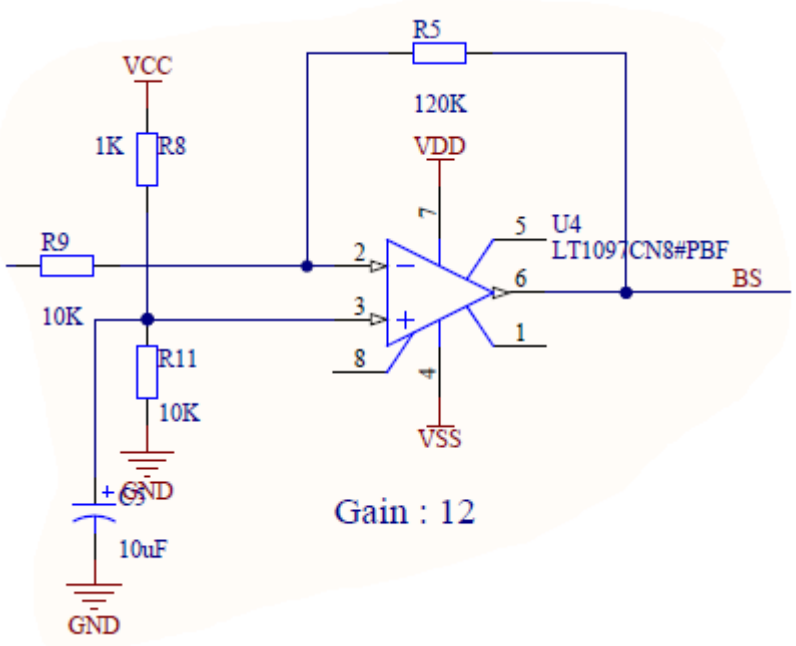
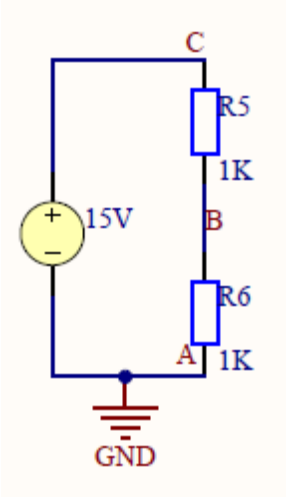


Figure 13 : Translation du signal vers une tension de 2.5V

Les sources de tensions disponibles sont +5V et +15V. L'AOP ne sera pas dans son état linéaire si sa borne VSS=0V. Donc l'alimentation la plus conseillé est ±2.5V ou ±7.5V, mais s'il est alimenté par ±2.5V l'amplitude maximale n'atteindra pas 5V, donc le choix s'adresse vers ±7.5V.

Pour avoir des potentiels +7.5V et -7.5V à partir d'une seule tension 15V, on utilise un diviseur de tension avec deux résistances égaux :



$$V_A = 0V$$
$$V_B = 7.5V$$
$$V_C = 15V$$

Maintenant on place la référence au point B, les potentiels seront :

$$V_A = -7.5V, \qquad V_B = 0, \qquad V_C = +7.5V$$

Comme ça, on aura un générateur de tension fictif, mais le problème qui reste, est que sa résistance interne est grande (soit $5K\Omega$). Pour la minimiser, on ajoute un montage suiveur (L'impédance à l'entrée du l'AOP est très grande et son impédance de sortie est faible) :

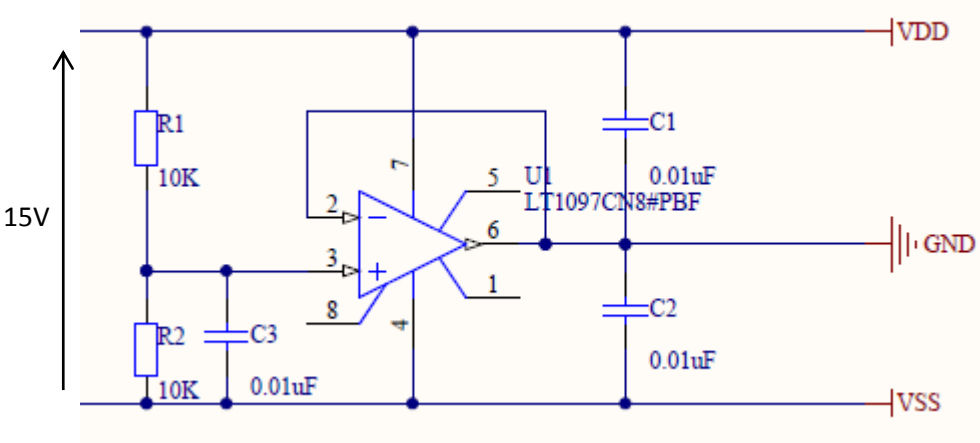


Figure 14 : Conversion de tension

Les condensateurs ne sont ici que pour le couplage.

Le Schéma finale est la Figure 17 dans l'annexe, et son circuit imprimé est la Figure 18.

La Figure 19 et la Figure 20 représentent les autres cartes à utiliser.

Chapitre 3

Aspect informatique

1. Architecture du programme

La partie Software se compose en deux parties ; un programme principal qui collectera les valeurs du signal, calculera la FFT et enverra le résultat. Il sera implanté dans le microcontrôleur. Et un programme d'interface qui s'occupera de l'affichage des valeurs dans un graph sur pc.

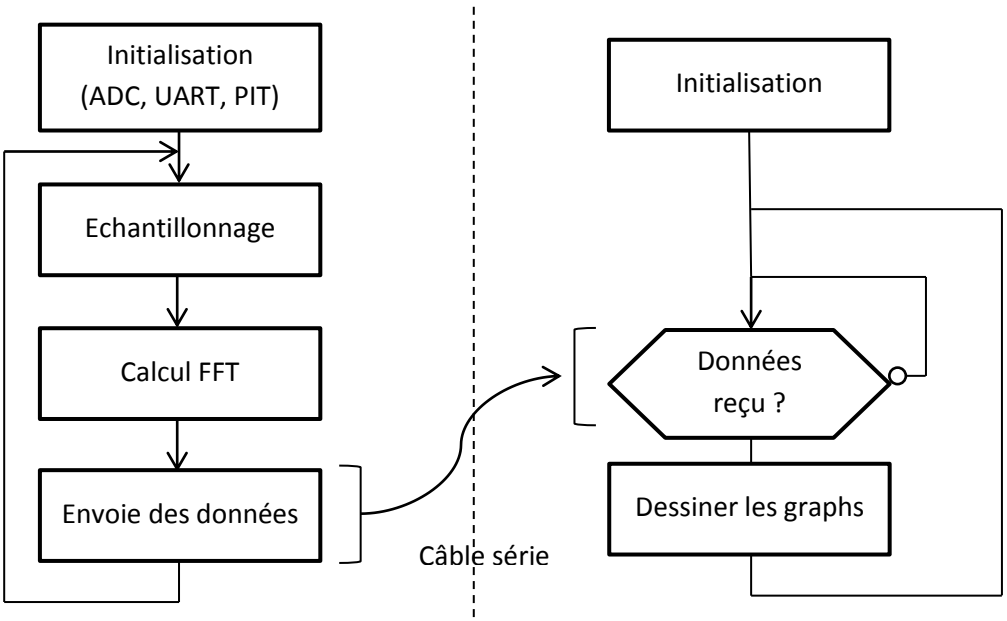


Figure 15 : Architecture du programme

Le diagramme dans la Figure 15 illustre le principe de fonctionnement du software ; Au démarrage du microcontrôleur, il faudra obligatoirement initialiser les modules utilisés.

A l'aide de l'ADC, le programme commencera à échantillonner le signal avec des intervalles, le PIT assurera la précision de ces intervalles. Quand on collectera le nombre des échantillons désirés, le programme commencera à calculer la FFT et envoyer le résultat vers le pc par une connexion port série et ainsi de suite formant de cette façon un cycle fermé.

Note :

Tous les registres utilisés pour l'initialisation et l'utilisation des modules sont expliqués dans le manuel de référence du microcontrôleur

2. Initialisation

Pour bien se familiariser avec la programmation dans l'architecture Power™, il faut savoir que chaque broche peut être associée, au plus, à 4 modules et qu'elle ne peut être opérationnelle qu'avec un seul module à la fois. Pour cela, il faut toujours configurer l'association de la broche au module souhaité.

2.1. Convertisseur analogique numérique

La Figure 15 liste les modules qui doivent être initialisé. Commençant par l'ADC, en se référant au schéma de la carte, on perçoit que le canal 7 de l'ADC 0 est associé à la broche 68. Et il faut savoir que l'ADC fonctionne en deux mode ; Scan mode, c'est-à-dire se boucler à la conversion du signal. One Shot, effectuer une seule conversion et attendre le signal pour recommencer une autre. La fonction **`init_adc()`** s'occupe de son initialisation complète :

```
void init_adc()
{
	SIUL.PCR68.B.APC = 1;		//Broche 68 en mode analogique afin
					//de l'associer à l'ADC
	ADC_0.NCMR0.B.CH7 = 1;		//Activer uniquement le canal 7
	ADC_0.MCR.B.MODE = 0;		//activer le mode One Shot
	ADC_0.MCR.B.PWDN = 0;		//Désactiver le mode
					//PowerDown(Réduction de le
					//consommation d'énergie)
	ADC_0.MCR.B.WLSIDE = 0;		//Lecture de gauche à droite
	ADC_0.CDR8.B.OVERW = 0;		//Pour ne pas écraser la valeur
					//précédente dans une nouvelle
					//lecture
}
```

Source 1 : Initialisation de l'ADC

2.2. Module du décompteur

Le module PIT joue le rôle d'une horloge, son principe de fonctionnement est de compter à partir d'une valeur initiale vers 0 par un pas précisé par le quartz. Il dispose de 4 canaux, chaque canal effectue sa tache indépendamment de l'autre. Son initialisation est simple :

```
void init_timer()
{
       PIT.PITMCR.B.MDIS=0x0;    // Activer le module PIT
       PIT.TCTRL0.B.TIE=0x0;     // Désactiver les interruptions[1] du
                                 //canal 0
       PIT.TCTRL1.B.TIE=0x0;     // Désactiver les interruptions du
                                 //canal 1
}
```

Source 2 : Initialisation du PIT

2.3. Module UART

Le module UART est le module responsable de la communication grâce à un port série, son initialisation est :

[1] Une interruption est un arrêt temporaire de l'exécution normale d'un programme, afin d'exécuter un autre programme (appelé service d'interruption).

```
volatile LINFLEX_tag *linflex;

void uart_init(void)
{
	linflex =(LINFLEX_tag *)&LINFLEX1;//Définition du pointeur LINFLEX[2]
	SIU.PCR[94].B.PA=0x1;             //Associer les broches 94 et 95
	SIU.PCR[95].B.PA=0x1;             //au module UART

	linflex->LINCR1.B.SLEEP = 0x0;    // Activer le module LINFLEX
	linflex->LINCR1.B.INIT  = 0x1;    // Entrer dans le mode init
	linflex->UARTCR.B.UART  = 0x1;    // Activer le mode UART

	linflex->UARTCR.B.TDFL_TFC = 0x1; // Spécifier la taille du buffer
	linflex->UARTCR.B.RDFL_RFC0 = 0x0;//à une taille de 1o
	linflex->UARTCR.B.RXEN = 0x1;     // Activer la réception
	linflex->UARTCR.B.TXEN = 0x1;     // Activer l'envoie
	linflex->UARTCR.B.PCE  = 0x0;     // Désactiver control de la
	                                  // parité
	linflex->UARTCR.B.WL0   = 0x1;    //Longueur de données et 8bits

	linflex->LINIBRR.B.IBR = 11;      // Vitesse d'envoi est 460800
	linflex->LINFBRR.B.FBR = 1;

	linflex->LINIER.B.DRIE  = 0x0;   // Désactiver tous les
	                                 //interruptions
	linflex->LINIER.B.DTIE  = 0x0;
	linflex->LINIER.B.DBFIE = 0x0;
	linflex->LINIER.B.DBEIE_TOIE = 0x0;

	linflex->LINCR1.B.INIT = 0x0;  // Quitter le mode d'initialisation
}
```

Source 3 : Initialisation de l'UART

3. Echantillonnage

3.1. Inversement des bits

Avant d'insérer le code d'échantillonnage, il faut d'abord se procurer une fonction d'inversion de bits pour assurer l'ordre des valeurs en sortie.

La fonction qui s'occupera de cette tâche est :

```
uint32_t bitrev(uint32_t n, uint32_t bits)
{
	uint32_t nrev, N;
	uint32_t count;
	N = 1<<bits;
	count = bits-1;
	nrev = n;
```

[2] LINFLEX est le module qui contient le sous module UART

```
        for(n>>=1; n; n>>=1)
        {
                nrev <<= 1;
                nrev |= n & 1;
                count--;
        }

        nrev <<= count;
        nrev &= N - 1;

        return nrev;
}
```

Source 4 : La fonction qui inverse l'ordre des bits

La fonction ci-dessus procède comme suit ; d'abord elle stocke la valeur du nombre à inverser dans la variable **nrev** et commence à décaler la variable n vers la droite tout en gardant son dernier bit pour l'ajouter à la droite de **nrev** jusqu'à l'obtention de n=0. A la fin, on conserve le nombre de bits indiqué dans le paramètre **bits**, en comptant de droite.

La variable **count** sert à compenser les zéros non aperçu à la gauche de **n**. Par exemple si l'on prend **n**=10010 et **bits**=8, à chaque itération on ajoute le bit du poids faible de la variable **n** au poids faible du **nrev.** Donc, après la boucle, on aura **n**=10010 et **nrev**=100**10101001.** Si on considère les 8 derniers bits -ceux en gras-, on remarque clairement que **nrev** n'est pas l'inverse de **n** car les zéros à gauche du nombre **n** ne seront pas visible, donc il faut décaler **nrev** à gauche count fois avant de sélectionner les bits souhaités pour combler ces 0. D'où **nrev** vaudra 01001000.

3.2. Echantillonnage

Cet analyseur de spectre est destiné à des fréquences audibles c'est-à-dire des fréquences qui ne dépassant pas les 20KHz.

Pour que la fréquence maximale calculée soit dans les environs de 20KHz, on calcule F_0 à partir de l'Équation 5 pour $k = N/2 - 1$:

$$20KHz = \frac{(N/2 - 1)}{NT_0}$$

Donc $T_0 = 24\mu s$.

```
//Initier décompteur 0 à envoyer un signal chaque 24µs
PIT.LDVAL0.R=1200;
PIT.TCTRL0.B.TEN=0x1;
for (;;) {
      //Attendre 24µs pour collecter une valeur du signal
```

```
        while(!PIT.TFLG0.B.TIF);
        PIT.TFLG0.B.TIF=1;

        //Collection de l'échantillon
        a=bitrev(i, LOG_N);             //Elle est défini dans Source 4
        signal[i].real=(get_adc0ch7_data()*ADC_Resolution);
        signal[i].imag=0;
        spectrum[a].real=signal[i].real;
        spectrum[a].imag=0;
        i++;

        //Après la collection de toutes valeurs, on calcul
        if(i==N_SAMPLES)
        {
              //Calcul et envoie des résultats
        }
}
```

Source 5 : Echantillonnage

La fonction `get_adc0ch7_data()` nous permet de lire la valeur actuelle de la tension :

```
uint32_t get_adc0ch7_data()
{
        uint32_t data = 0;

        ADC_0.MCR.B.NSTART = 1;         //Démarrer l'ADC 0

        if (ADC_0.CDR[7].B.VALID)       //Une fois la donnée prête on
        {                               //collecte
              data = ADC_0.CDR[7].B.CDATA;
        }
        return data;
}
```

Source 6 : Prélèvement d'échantillon

4. Implantation de la FFT

4.1. Les différents boucles de la FFT

En regroupant les papillons élémentaires de la Figure 5 en des étages et blocs on aura :

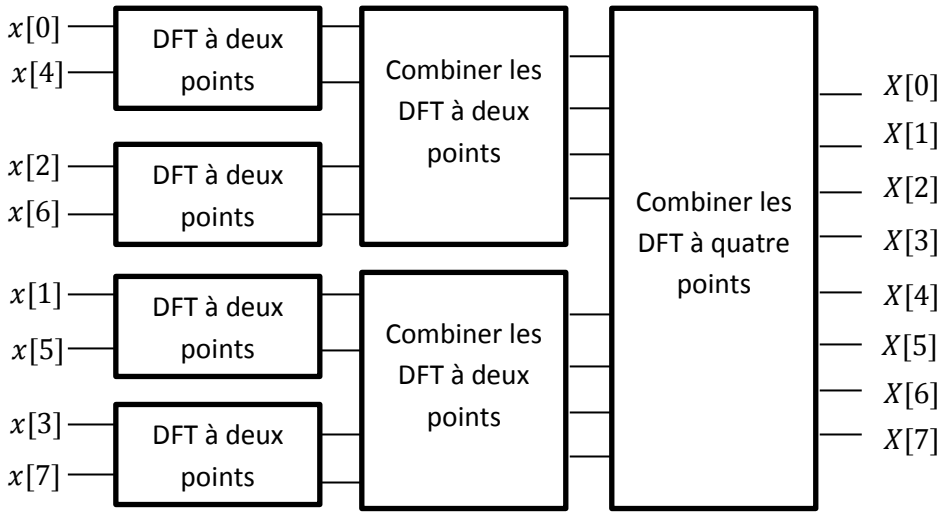


Figure 16 : Regroupement de la FFT

Et donc, on voit lisiblement qu'il faut disposer de trois boucles :

- La première qui se répète pour chaque étage ($p = \log_2 N$ fois)
- La deuxième pour chaque bloc se trouvant à l'intérieur de l'étage courant
- La troisième pour chaque papillon élémentaire :

```
for (k=N_SAMPLES; k>1; k=k>>1)          /* Boucle pour chaque étage */
{
      for (j=0; j<N_SAMPLES; j+=GS)     /* Boucle pour chaque bloc */
      {
            for (n=j; n<(j+step); n++)/* Boucle pour chaque papillon */
            {
                  /* Calcul d'un seul papillon ici */
            }
      }
}
```

Source 7 : Les différentes boucles

4.2. Calcul d'un seul papillon

Après avoir défini les boucles de la FFT, on traduira par la suite l'Équation 6 en C :

```
temp1 = (spectrum[n+step].real * Wk[wIndex].real);
temp2 = (spectrum[n+step].imag * Wk[wIndex].imag);
temp3 = (spectrum[n+step].real * Wk[wIndex].imag);
temp4 = (spectrum[n+step].imag * Wk[wIndex].real);

temp1_2 = temp1 - temp2;
temp3_4 = temp3 + temp4;

spectrum[n+step].imag = spectrum[n].imag - temp3_4;
```

```
spectrum[n+step].real = spectrum[n].real - temp1_2;
spectrum[n].real = temp1_2 + spectrum[n].real;
spectrum[n].imag = temp3_4 + spectrum[n].imag;
```

Source 8 : Calcul d'un seul papillon

5. L'envoie du résultat

Pour envoyer le résultat du calcul effectué, il faut tout d'abord établir les fonctions nécessaires :

- Une 1ère fonction envoie un caractère ;
- Une 2ème envoie une chaine de caractère en faisant appel à la 1ère fonction.

5.1. Fonctions d'envoie

Pour envoyer une donnée par le port série, il faut préciser la taille de la donnée. Une taille qui ne peut dépasser les 4octets, puis mettre les données dans des registres DATA3, DATA2, DATA1, DATA0 suivant cet ordre, et finalement, il faut attendre le succès de l'envoi de la donnée :

```
void outChar(char c)
{
	LINFLEX1.UARTCR.B.TDFL_TFC=1;		//Un seul octet à
envoyer
	LINFLEX1.BDRL.B.DATA0=c;		//La donnée
	while(!LINFLEX1.UARTSR.B.DTF_TFF){}	//Attente d'envoi
	LINFLEX1.UARTSR.B.DTF_TFF=0x1;
}
void OutString(char* s)
{
	while(*s)outChar(*(s++));		//Envoi d'octet par
octet	outChar('\n');				//Fin de ligne
}
```

Source 9 : Fonctions d'envoi des données

5.2. Format de données

Sachant que le port série est doté d'un débit pas assez important, il faut minimser le nombre d'octets à envoyer. Donc on envisage le format suivant :

<Temps_de_calcul> :<valeur_du_signal_0> :… : <valeur_du_signal_N-1> : <valeur_du_spectre_0> : … :<valeur_du_spectre_N/2-1>

Les « : » jouerons le rôle d'un séparateur des valeurs.

```
sprintf(buffer, "%d", (int)(CalculTime*0.020));

for(k=0;k<N_SAMPLES;k++)
```

```
{
        sprintf(tmpBuffer, ":%d",(int)(signal[k].real));
        strcat(buffer, tmpBuffer);
}
for(k=0;k<N_SAMPLES/2;k++)
{
        sprintf(tmpBuffer, ":%d", (int)(spectrum[k].real*spectrum[k].real+
                        spectrum[k].imag*spectrum[k].imag));
        strcat(buffer, tmpBuffer);
}

OutString(buffer);
```

Source 10 : Envoie des données

Biographie

Jean Baptiste Joseph Fourier

C’est un mathématicien et physicien français né le 21 mars 1768 à Auxerre et mort le 16 mai 1830 à Paris. Il est diplômé de l’école normale supérieur. Il est connu pour ses travaux sur la décomposition de fonctions périodiques en séries trigonométriques convergentes appelées séries de Fourier et leur application au problème de la propagation de la chaleur. Parmi ces travaux :

C'est à Grenoble qu'il conduit ses expériences sur la propagation de la chaleur qui lui permettront de modéliser l'évolution de la température au travers de séries trigonométriques. Ces travaux, qui apportent une grande amélioration à la modélisation mathématique de phénomènes, ont contribué aux fondements de la thermodynamique. Ils ont ouvert la voie à la théorie des séries de Fourier et des transformées de Fourier. Toutefois, la simplification excessive que ces outils proposent sera très contestée, notamment par Pierre-Simon de Laplace et Joseph-Louis Lagrange.

Fourier est probablement l'un des premiers à avoir proposé, en 1824, une théorie selon laquelle les gaz de l'atmosphère terrestre augmentent la température à sa surface – c'est une première ébauche de l'effet de serre. Ses travaux sur la chaleur le poussèrent à étudier les équilibres énergétiques sur les planètes : elles reçoivent l'énergie sous forme de rayonnement à partir d'un certain nombre de sources – ce qui augmente leur température – mais en perdent également par radiation infrarouge (ce qu'il appelait « chaleur obscure ») d'autant plus que la température est élevée – ce qui tend à diminuer cette dernière. On atteint donc un équilibre, et l'atmosphère favorise les températures plus élevées en limitant les pertes de chaleur. Il ne put cependant déterminer avec précision cet équilibre, et la loi de Stefan-Boltzmann, qui donne la puissance du rayonnement du corps noir, ne sera établie que cinquante ans plus tard.

Alors que l'effet de serre est aujourd'hui à la base de la climatologie, Fourier est fréquemment cité comme le premier à avoir présenté cette notion (voir par exemple John Houghton). Ces citations prennent souvent la date de 1827 comme première évocation de l'effet de serre par Fourier. Pourtant l'article cité en 1827 n'est qu'une nouvelle version de l'article original publié dans les Annales de chimie et de physique en 1824.

Johann Carl Friedrich Gauss

Né le 30 avril 1777 à Brunswick et mort le 23 février 1855 à Göttingen, est un mathématicien, astronome et physicien allemand. Il a apporté de très importantes contributions à ces trois domaines. Surnommé « le prince des mathématiciens », il est considéré comme l'un des plus grands mathématiciens de tous les temps.

La qualité extraordinaire de ses travaux scientifiques était déjà reconnue par ses contemporains. Dès 1856, le roi de Hanovre fit graver des pièces commémoratives avec l'image de Gauss et l'inscription Mathematicorum Principi (« au prince des mathématiciens » en latin). Gauss n'ayant publié qu'une partie de ses découvertes, la postérité découvrit surtout l'étendue de ses travaux lors de la publication de ses Œuvres, de son journal et d'une partie de ses archives, à la fin du XIXe siècle.

Gauss dirigea l'Observatoire de Göttingen et ne travailla pas comme professeur de mathématiques – d'ailleurs il n'aimait guère enseigner – mais il encouragea plusieurs de ses étudiants, qui devinrent d'importants mathématiciens, notamment Gotthold Eisenstein et Bernhard Riemann.

James William Cooley

James William Cooley (né en 1926) est un mathématicien américain renommé pour l'algorithme de Cooley-Tukey (1965), un algorithme de transformée de Fourier rapide (FFT) fondamental dans le traitement du signal moderne.

John Wilder Tukey

(16 juin 1915 - 26 juillet 2000) est l'un des plus importants statisticiens américains du XXe siècle. Il a créé et développé de nombreuses méthodes statistiques. Il est notamment connu pour son développement en 1965, avec James Cooley, de l'algorithme de la transformée de Fourier rapide. Les concepts et méthodes statistiques qu'il a inventés sont aujourd'hui au programme des lycées et des universités.

Annexes

RS-232

RS-232 (parfois appelée EIA RS-232, EIA 232) est une norme standardisant un bus de communication de type série sur trois fils minimum (électrique, mécanique et protocole). Disponible sur presque tous les PC jusqu'au milieu des années 2000, il est communément appelé le « port série ». Sur les systèmes d'exploitation MS-DOS et Windows, les ports RS-232 sont désignés par les noms COM1, COM2, etc. Cela leur a valu le surnom de « ports COM », encore utilisé de nos jours. Cependant, il est de plus en plus remplacé par le port USB.

Le standard RS-232 recouvre plusieurs autres standards : les recommandations UIT-T V.24 (définition des circuits) et V.28 (caractéristiques électriques), ainsi que la norme ISO 2110 pour la connectique.

Les liaisons RS-232 sont fréquemment utilisées dans l'industrie pour connecter différents appareils électroniques (automate, appareil de mesure, etc.).

Source code

ComplexWk.h :

```
/*
 * ComplexWk.h
 *
 *  Created on: Apr 10, 2013
 *      Author: Yassine Ouamer
 */

#ifndef COMPLEXWK_H_
#define COMPLEXWK_H_

#endif /* COMPLEXWK_H_ */

#define N_Wk              32

typedef struct{
      float real;
      float imag;
}Complex;

Complex Wk[N_Wk]={
            1.000000,0.000000,
            0.980785,0.195090,
            0.923880,0.382683,
```

```
0.831470,0.555570,
0.707107,0.707107,
0.555570,0.831470,
0.382683,0.923880,
0.195090,0.980785,
0.000000,1.000000,
-0.195090,0.980785,
-0.382683,0.923880,
-0.555570,0.831470,
-0.707107,0.707107,
-0.831470,0.555570,
-0.923880,0.382683,
-0.980785,0.195090,
-1.000000,0.000000,
-0.980785,-0.195090,
-0.923880,-0.382683,
-0.831470,-0.555570,
-0.707107,-0.707107,
-0.555570,-0.831470,
-0.382683,-0.923880,
-0.195090,-0.980785,
0.000000,-1.000000,
0.195090,-0.980785,
0.382683,-0.923880,
0.555570,-0.831470,
0.707107,-0.707107,
0.831470,-0.555570,
0.923880,-0.382683,
0.980785,-0.195090};
```

Source 11 : ComplexWk.h

Main.c :

```
#include "PXS2010.h"
#include "init.h"
#include "ComplexWk.h"
#include "stdio.h"
#include "string.h"
#include "math.h"

#define N_SAMPLES                           64
#define LOG_N                               6
#define ADC_Resolution                      1.22
#define DefaultTimerStartingValue           0xFFF
#define PI                                  3.1415926535

extern Complex Wk[N_Wk];

void outChar(char c);
void OutString(char* s);
uint32_t get_adc0ch8_data();
uint32_t bitrev(uint32_t n, uint32_t bits);
```

```
//float get_sinusoids(uint32_t i);


int main(void) {
	Complex signal[N_SAMPLES];
	Complex spectrum[N_SAMPLES];

	uint32_t i=0;                   /* Number of sample counter */
	uint32_t GS = 2;                /* Block step initial value */
	uint32_t step = 1;              /* Initial value */
	uint32_t wIndex = 0;            /* w Index */
	uint32_t CalculTime = 0;        /* Count calculation time */

	char buffer[4096], tmpBuffer[16];

	int x=0;
	uint32_t k,j,n,a;
	float temp1, temp2, temp3, temp4, temp1_2, temp3_4;

	//Initialisation of hardware
	SIUL.PCR54.B.OBE=1;
	SIUL.PCR107.B.OBE=1;
	init();
	SIUL.GPDO54.B.PDO=1;
	SIUL.GPDO107.B.PDO=1;

	//Set Timer 0 to trigger every 24µs
	PIT.LDVAL0.R=1200;

	/* Loop forever alone */
	for (;;) {
		//Wait 24µs to collect a sample
		while(!PIT.TFLG0.B.TIF);
		PIT.TFLG0.B.TIF=1;

		//Now collecting a sample of signal
		a=bitrev(i, LOG_N);
		signal[i].real=(get_adc0ch7_data()*ADC_Resolution);
		signal[i].imag=0;
		spectrum[a].real=signal[i].real;
		spectrum[a].imag=0;
		i++;

		//When finishing Calculate result
		if(i==N_SAMPLES)
		{
			//Start timer 1
			PIT.LDVAL1.R=DefaultTimerStartingValue;
			PIT.TCTRL1.B.TEN=0x1;
			SIUL.GPDO107.B.PDO=0;

			//FFT Calculation
```

```
/* Repeat this loop for each stage */
for (k=N_SAMPLES; k>1; k=k>>1)
{
	/* Repeat this loop for each block */
	for (j=0; j<N_SAMPLES; j+=GS)
	{
		wIndex=0;
		/* Repeat this loop for each butterfly */
		for (n=j; n<(j+step); n++)
		{
	 temp1=(spectrum[n+step].real*Wk[wIndex].real);
	temp2=(spectrum[n+step].imag*Wk[wIndex].imag);
	 temp3=(spectrum[n+step].real*Wk[wIndex].imag);
	 temp4=(spectrum[n+step].imag*Wk[wIndex].real);

			temp1_2 = temp1 - temp2;
			temp3_4 = temp3 + temp4;

	spectrum[n+step].imag=spectrum[n].imag-temp3_4;
	spectrum[n+step].real=spectrum[n].real-temp1_2;
	spectrum[n].real=temp1_2+spectrum[n].real;
	spectrum[n].imag=temp3_4+spectrum[n].imag;

			wIndex+=k/2;
		}
	}
	/* Change the GS and step for the next stage */
	GS <<= 1;		//<<1
	step <<= 1;		//<<1
}

//Now preparing to the next range of values
i=0;
GS = 2;
step = 1;
wIndex = 0;

//Stop Timer 1 and read value of blind time
SIUL.GPDO107.B.PDO=1;
CalculTime=DefaultTimerStartingValue-PIT.CVAL1.R;
PIT.TCTRL1.B.TEN=0x0;

//Sending values to the computer
/* horloge timer is 40Hz which is 25ns */
sprintf(buffer, "%d", (int)(CalculTime*0.025));

for(k=0;k<N_SAMPLES;k++)
{
	sprintf(tmpBuffer,":%d",(int)(signal[k].real));
	strcat(buffer, tmpBuffer);
}
for(k=0;k<N_SAMPLES/2;k++)
```

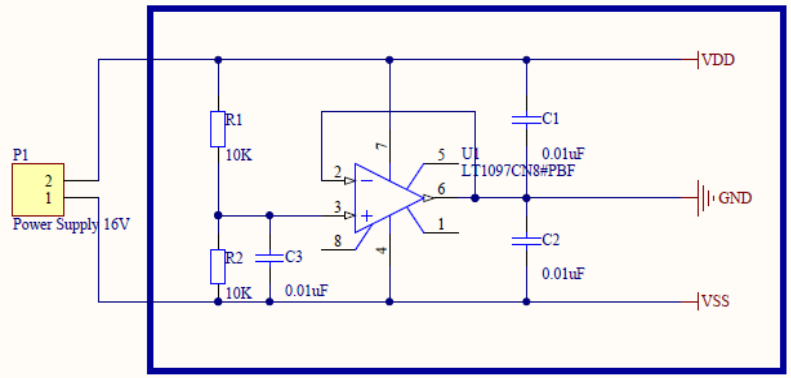
```
                    {
                        sprintf(tmpBuffer, ":%d",(int)(spectrum[k].real
                                *spectrum[k].real+spectrum[k].imag
                                    *spectrum[k].imag));
                        strcat(buffer, tmpBuffer);
                    }

                    OutString(buffer);

                    //Take a rest ?
                    //for(x=0;x<=1000000;x++){}
                }
        }
}

////////////////////////////////////////////////////////////////////////
//          Bit Reverse :
//          Example :  0101001 -> 1001010
////////////////////////////////////////////////////////////////////////
uint32_t bitrev(uint32_t n, uint32_t bits)
{
        uint32_t nrev, N;
        uint32_t count;
        N = 1<<bits;
        count = bits-1;
        nrev = n;
        for(n>>=1; n; n>>=1)
        {
                nrev <<= 1;
                nrev |= n & 1;
                count--;
        }

        nrev <<= count;
        nrev &= N - 1;

        return nrev;
}

////////////////////////////////////////////////////////////////////////
//          Read value from Channel 8 of ADC 0
////////////////////////////////////////////////////////////////////////
uint32_t get_adc0ch8_data()
{
        uint32_t data = 0;

        ADC_0.MCR.B.NSTART = 1;                 //Start ADC0

        if (ADC_0.CDR[8].B.VALID)
        {
                data = ADC_0.CDR[8].B.CDATA;
        }
```

```
	return data;
}

////////////////////////////////////////////////////////////////////
//		Sinusoids generator
////////////////////////////////////////////////////////////////////
/*float get_sinusoids(uint32_t i)
{
	return
1000*sinf((float)((8*PI*i)/N_SAMPLES))+2000*sinf((float)((4*PI*i)/N_SAMPL
ES))+500*sinf((float)((PI*i)/N_SAMPLES));
}*/

////////////////////////////////////////////////////////////////////
//		Send 8bit value using Serial Port
////////////////////////////////////////////////////////////////////
void outChar(char c)
{
	LINFLEX1.UARTCR.B.TDFL_TFC=1;			//One byte to send
	LINFLEX1.BDRL.B.DATA0=c;			//Data
	while(!LINFLEX1.UARTSR.B.DTF_TFF){}		//Waiting for sending
	LINFLEX1.UARTSR.B.DTF_TFF=0x1;			//Reset flag
}

////////////////////////////////////////////////////////////////////
//		Send a string 8bit by 8bit using outChar function
////////////////////////////////////////////////////////////////////
void OutString(char* s)
{
	while(*s)outChar(*(s++));			//Send byte by byte
	outChar('\n');					//End of line
}
```

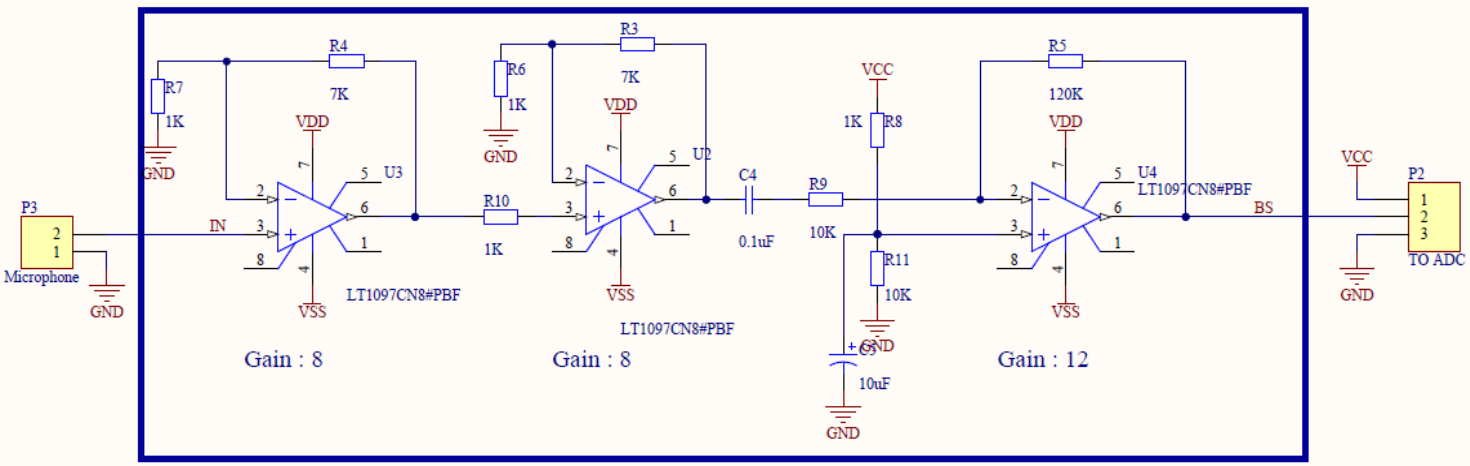
Source 12 : Main.c

Schéma final

Figure 17 : Schéma final



Power Supply +/- 8V



Signal Amplifier

Set offset to 2.5V and amplify

Circuit imprimé

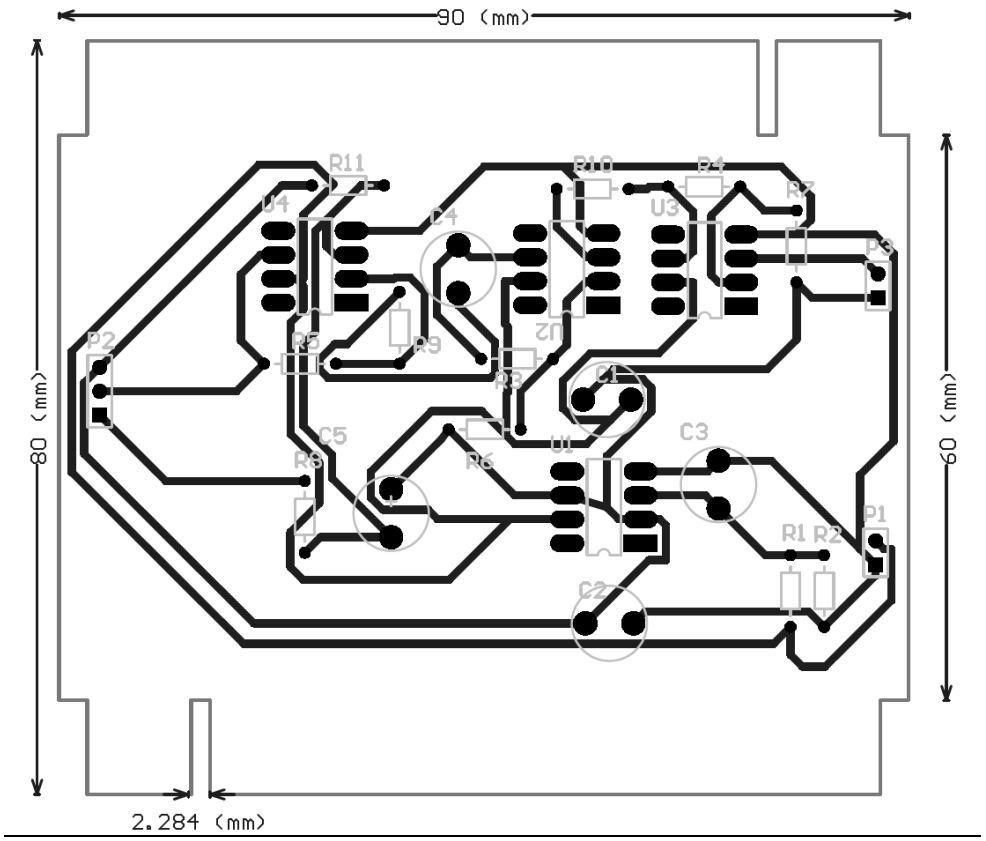


Figure 18 : Circuit imprimé

Carte d'unité de traitement

Figure 19 : TWR-PXS20, unité de traitement

Carte port série

Figure 20 : TWR-Ser, communication port série

Référence :

http://fr.wikipedia.org/
http://en.wikipedia.org/
http://www.ti.com/
http://www.freescale.com/
http://www.youtube.com/user/allsignalprocessing
https://ccrma.stanford.edu/
http://www.esiee.fr/

Yassine Ouamer
Mai 2013