Transformée de Fourier rapide

La rapidité d'exécution du calcul est essentielle lorsqu'il s'agit de traiter un signal en temps réel. Elle déterminera la fréquence maximale qui puisse être analysée. Repartons de la formulation complexe :

Posons $A(k) = \mathcal{F}\left(k\frac{f_e}{N}\right)$

$$A(k) = \sum_{n=0}^{n=N-1} x(n)\, e^{-j2\pi\frac{kn}{N}} \tag{1}$$

Comme nous l'avons vu et expérimenté précédemment, l'application directe de cette formulation consiste à faire évoluer les indices n et k de 0 à N ce qui conduite à effectuer N^2 opérations ce qui représente vite, même avec des processeurs rapides, un temps de calcul important (plusieurs ms sur un microcontrôleur tournant à 20MHz, limitant la fréquence d'analyse à quelques dizaines de Hz). Mais on peut réduire ce nombre d'opérations en appliquant quelques astuces algorithmiques :

Posons : $\boxed{W_N^m = e^{-j2\pi\frac{m}{N}}}$

1 Valeurs remarquables de W_N^{kn} :

1.1

$$W_{N/2}^{k(n/2)} = W_{N/2}^{(k/2)n} = W_N^{kn} \tag{2}$$

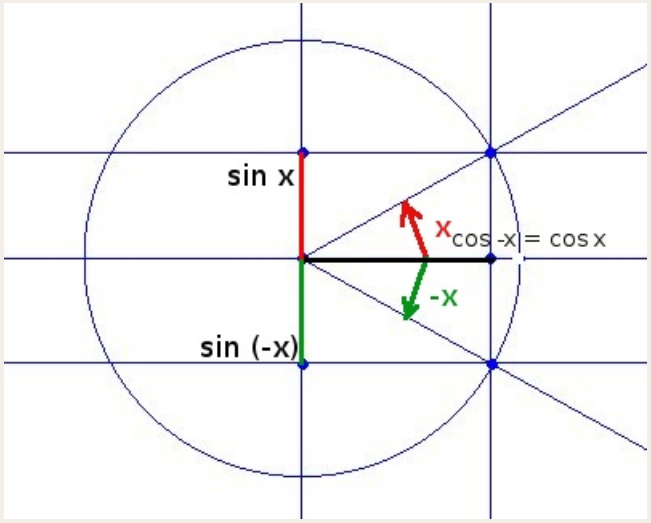
Démonstration. de (2) : □

$$e^{-j2\pi\frac{k(n/2)}{N/2}} = e^{-j2\pi\frac{(k/2)n}{N/2}} = e^{-j2\pi\frac{kn}{N}}$$

1.2 pour $n < N/2$

$$W_N^{kn+N/2} = -W_N^{kn} \tag{3}$$

Rappels :



$\cos(-x) = \cos x$

$\sin(-x) = -\sin x$

$e^{jx} = \cos x + j \sin x$

$e^{-jx} = e^{j(-x)} = \cos(-x) + j\sin(-x) = \cos x - j \sin x$

$e^{-j\pi} = \cos\pi - j\sin\pi = -1 - j\sin 0 = -1 - 0 = -1$

$e^{-j2\pi} = \cos(2\pi) - j\sin(2\pi) = 1 - 0 = 1$

$e^{-jm2\pi} = \cos(m2\pi) - j\sin(m2\pi) = 1 - 0 = 1$

Démonstration. de (3): □

$$\begin{aligned} e^{-j2\pi\frac{kn+N/2}{N}} &= e^{-j2\pi\left(\frac{kn}{N}+\frac{N/2}{N}\right)} \\ &= e^{-j2\pi\frac{kn}{N}-j2\pi\frac{N}{2N}} \\ &= e^{-j2\pi\frac{kn}{N}-j\pi} \\ &= \left(e^{-j2\pi\frac{kn}{N}}\right)(e^{-j\pi}) \\ &= \left(e^{-j2\pi\frac{kn}{N}}\right)\times(-1) \\ &= -e^{-j2\pi\frac{kn}{N}} \end{aligned}$$

1.3

$$W_{N/2}^{k+N/2} = W_{N/2}^{k} \tag{4}$$

Démonstration. de (4) : □

$$\begin{aligned} e^{-j2\pi\frac{k+N/2}{N/2}} &= e^{-j2\pi\left(\frac{k}{N/2}+\frac{N/2}{N/2}\right)} \\ &= e^{-j2\pi\frac{k}{N/2}-j2\pi\times 1} \\ &= e^{-j2\pi\frac{k}{N/2}-j2\pi} \\ &= \left(e^{-j2\pi\frac{k}{N/2}}\right)(e^{-j2\pi}) \\ &= e^{-j2\pi\frac{k}{N/2}}\times 1 \\ &= e^{-j2\pi\frac{k}{N/2}} \end{aligned}$$

1.4

$$W_{N/2}^{(k+N/2)m} = W_{N/2}^{km} \tag{5}$$

Démonstration. de (5) : □

$$\begin{aligned} e^{-j2\pi m\frac{k+N/2}{N/2}} &= e^{-j2m\pi\left(\frac{k}{N/2}+\frac{N/2}{N/2}\right)} \\ &= e^{-j2m\pi\frac{k}{N/2}-j2\pi m\times 1} \\ &= e^{-j2m\pi\frac{k}{N/2}-jm.2\pi} \\ &= e^{-j2m\pi\frac{k}{N/2}}\times 1 \\ &= e^{-j2m\pi\frac{k}{N/2}} \end{aligned}$$

1.5

$$W_N^{k(n+p)} = W_N^{kn}.W_N^{kp} \tag{6}$$

Démonstration. de (6) : □

$$\begin{aligned} e^{-j2\pi\frac{k(n+p)}{N}} &= e^{-j2\pi\left(\frac{kn}{N}+\frac{kp}{N}\right)} \\ &= e^{-j2\pi\frac{kn}{N}-j2\pi\frac{kp}{N}} \\ &= e^{-j2\pi\frac{kn}{N}}\times e^{-j2\pi\frac{kp}{N}} \end{aligned}$$

2 Algorithme de Transformée rapide

2.1 Séparation en deux moitiés :

Séparons les échantillons d'ordre pair et impair, en tenant compte de ce que nous avons vu plus haut, il vient :

$$
\begin{aligned}
A(k) &= \sum_{m=0}^{N/2-1} x(2m).W_N^{k2m} + \sum_{m=0}^{N/2-1} x(2m+1).W_N^{k(2m+1)} \\
&= \sum_{m=0}^{N/2-1} x(2m).W_N^{k2m} + \sum_{m=0}^{N/2-1} x(2m+1).W_N^{k2m+k} \\
&= \sum_{m=0}^{N/2-1} x(2m).W_N^{k2m} + \sum_{m=0}^{N/2-1} x(2m+1).W_N^{k}.W_N^{k2m} \\
&= \sum_{m=0}^{N/2-1} x(2m).W_{N/2}^{km} + W_N^k \times \sum_{m=0}^{N/2-1} x(2m+1).W_N^{k2m} \\
&= \left(\sum_{m=0}^{N/2-1} x(2m).W_{N/2}^{km} \right) + \left(W_N^k \times \sum_{m=0}^{N/2-1} x(2m+1).W_{N/2}^{km} \right) \qquad (7)
\end{aligned}
$$

Nous voyons que la transformée de Fourier de taille N est devenue la somme de deux transformées de Fourier de taille N/2. Si le calcul de départ prenait N^2 opérations, maintenant il ne nécessite plus que

$2(N/2)^2 + N = 2N^2/4 + N = N^2 + N$ soit environ la moitié.

Si N est une puissance de 2, on peut répéter le fractionnement jusqu'à l'obtention d'une somme de base 2, l'ensemble des calculs ne nécessitant plus que $N/2\log_2(N)$ opérations ce qui peut être $100\,\text{fois}$ moins lourd.

Nous remarquons également que le facteur $W_{N/2}^{km}$ se retrouve à l'identique dans les termes de rang pair et impair de même ordre et ne sera donc calculé qu'une fois sur deux.

Le facteur W_N^k qui ne dépend pas de n (ni de m) (raison pour laquelle on l'a « sorti » de la somme $\sum$) figurant dans les termes de rang impair, pourra bénéficier de la propriété(4).

L'algorithme le plus célèbre qui en découle est celui inventé en 1805 par Carl Friedrich Gauss pour calculer les trajectoires des astéroïdes Pallas et Juno, puis réinventé en 1965 par James Cooley (IBM) et John Tukey (Princeton) qui l'adaptèrent au calcul sur ordinateur et lui donnèrent leurs noms.

2.1.1 transformée de Fourier des échantillons de rang pair :

$$F_P(k) = \sum_{m=0}^{N/2-1} x(2m).W_{N/2}^{km} \tag{8}$$

$F_P(k)$ est la transformée de Fourier d'ordre $N/2$ des échantillons de rang pair.

Remarque, d'après (5):

$$F_P(k+N/2) = F_P(k) \tag{9}$$

Démonstration. de (9) : □

$$\begin{aligned} F_P(k+N/2) &= \sum_{m=0}^{N/2-1} x(2m).W_{N/2}^{(k+N/2)m} \\ &= \sum_{m=0}^{N/2-1} x(2m).W_{N/2}^{km} \\ &= F_P(k) \end{aligned} \tag{10}$$

2.1.2 transformée de Fourier des échantillons de rang impair :

$$F_I(k) = \sum_{m=0}^{N/2-1} x(2m+1).W_{N/2}^{km} \tag{11}$$

$F_I(k)$ est la transformée de Fourier d'ordre $N/2$ des échantillons de rang impair

remarque, toujours d'après (5) :

$$F_I(k+N/2) = F_I(k) \tag{12}$$

La transformée de Fourier complète devient :

$$A(k) = F_P(k) + W_N^k.F_I(k) \tag{13}$$

Nous pouvons maintenant décrire l'algorithme de Cooley-Tukey qui concrétise les équations qui précèdent. Toutefois toutes les descriptions que j'ai pu voir (en français) de cet algorithme (agrémentés de croquis de « papillons ») m'ont paru claires comme du jus de chique. Je vais donc essayer de rendre la chose limpide et j'avoue que je ne suis par sûr d'y parvenir ! Je n'ai moi-même compris la chose qu'en lisant des travaux parus en anglais.

Je pense que le plus simple consiste à se fixer une valeur de k petite, 4 par exemple, d'écrire dans un tableau les quatre lignes de calcul correspondantes, puis de voir les simplifications qu'on peut y apporter.

soit donc $N=8$, et $N/2=4$.

$$\begin{aligned} A(0) &= F_P(0) + W_N^0.F_I(0) \\ A(1) &= F_P(1) + W_N^1.F_I(1) \\ A(2) &= F_P(2) + W_N^2.F_I(2) \\ A(3) &= F_P(3) + W_N^3.F_I(3) \end{aligned}$$

$$
\begin{aligned}
A(4) &= F_P(4) + W_N^4 . F_I(0) \\
A(5) &= F_P(5) + W_N^5 . F_I(0) \\
A(6) &= F_P(6) + W_N^6 . F_I(0) \\
A(7) &= F_P(7) + W_N^7 . F_I(0)
\end{aligned}
$$

Appliquons le résultat connu (9) : $\mathrm{FP}(k+N/2) = \mathrm{FP}(k)$ ainsi que (11) : $\mathrm{FI}(k+N/2) = \mathrm{FI}(k)$

$$
\begin{aligned}
A(0) &= F_P(0) + W_N^0 . F_I(0) \\
A(1) &= F_P(1) + W_N^1 . F_I(1) \\
A(2) &= F_P(2) + W_N^2 . F_I(2) \\
A(3) &= F_P(3) + W_N^3 . F_I(3) \\
A(4) &= F_P(0) + W_N^4 . F_I(0) \\
A(5) &= F_P(1) + W_N^5 . F_I(1) \\
A(6) &= F_P(2) + W_N^6 . F_I(2) \\
A(7) &= F_P(3) + W_N^7 . F_I(3)
\end{aligned}
$$

Les coefficients W_N^k qui multiplient les seconds termes sont appelés en anglais « twiddle factor » ce qui est assez cocasse surtout si vous le traduisez mot à mot... Nous allons nous en occuper maintenant.

reprenons (14) $W_N^{kn+N/2} = -W_N^{kn}$

avec $n=1$, il vient:

$$
W_N^{k+N/2} = -W_N^k \tag{14}
$$

Nous pouvons alors diminuer à nouveau le nombre de termes différents : (on fait sauter W_N^4 à W_N^7)

$$
\begin{aligned}
A(0) &= F_P(0) + W_N^0 . F_I(0) \\
A(1) &= F_P(1) + W_N^1 . F_I(1) \\
A(2) &= F_P(2) + W_N^2 . F_I(2)
\end{aligned}
$$

$$
\begin{aligned}
A(3) &= F_P(3) + W_N^3 . F_I(3) \\
A(4) &= F_P(0) - W_N^0 . F_I(0) \\
A(5) &= F_P(1) - W_N^1 . F_I(1) \\
A(6) &= F_P(2) - W_N^2 . F_I(2) \\
A(7) &= F_P(3) - W_N^3 . F_I(3)
\end{aligned}
$$

N'apparaissent plus que les transformées de Fourier d'ordre N/2 $F_P(0)$ à $F_P(3)$ et $F_I(0)$ à $F_I(3)$ soit quatre TF de rang pair, quatre TF de rang impair et quatre « facteurs de tripatouilages » (W_N^k), ces derniers pouvant être pré-calculés et mémorisés.

Récapitulons afin d'être bien clair :

- $A(k)$ c'est une des composantes de la transformée de Fourier que l'on cherche à obtenir. C'est une raie dans le domaine fréquenciel.
- $F_P(k)$ c'est une transformée partielle d'indice k=0 calculée sur les échantillons pairs du signal temporel. Elle nécessite de calculer une somme basée sur tous les échantillons pairs.

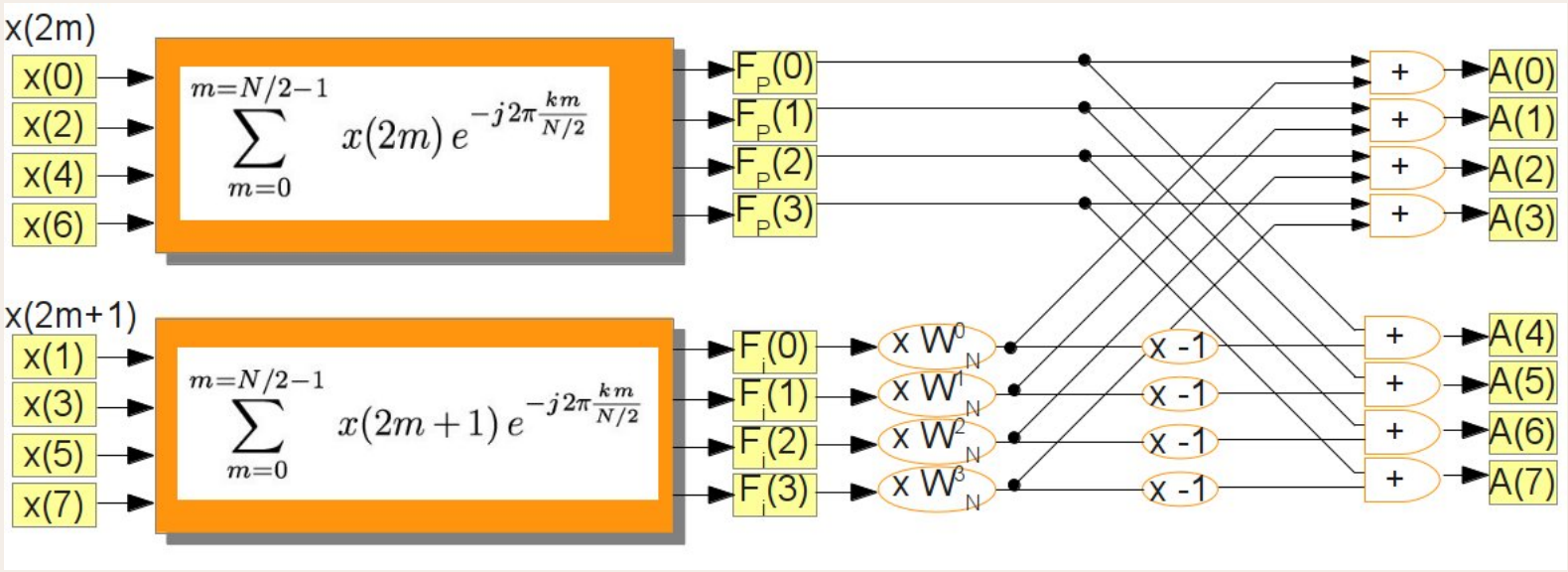
  $F_{P2}(k) = \sum_{m=0}^{m=N/2-1} x(2m)\, e^{-j2\pi\frac{km}{N/2}}$
- $F_I(k)$ c'est une transformée partielle d'indice k=0 calculée sur les échantillons impairs du signal temporel. Elle nécessite de calculer une somme basée sur tous les échantillons impairs.

  $F_{I2}(k) = \sum_{m=0}^{m=N/2-1} x(2m+1)\, e^{-j2\pi\frac{km}{N/2}}$
- Les transformées partielles $F_P(0...3)$ et $F_I(0...3)$ sont obtenues en prenant successivevent $k = 0, 1, 2, 3$ et en balayant toutes les valeurs de m (donc tous les échantillons) pour chacune de ces valeurs de k.

Donc le calcul de chaque valeur de $A(k)$ nécessite d'utiliser tous les échantillons $x(n)$ (pairs et impairs) du signal au même titre que les transformées partielles.

2.2 Représentations graphiques :

Voici une représentation graphique de ces calculs. Les moulinettes qui brassent tous les échantillons se situent dans les gros pavés oranges.

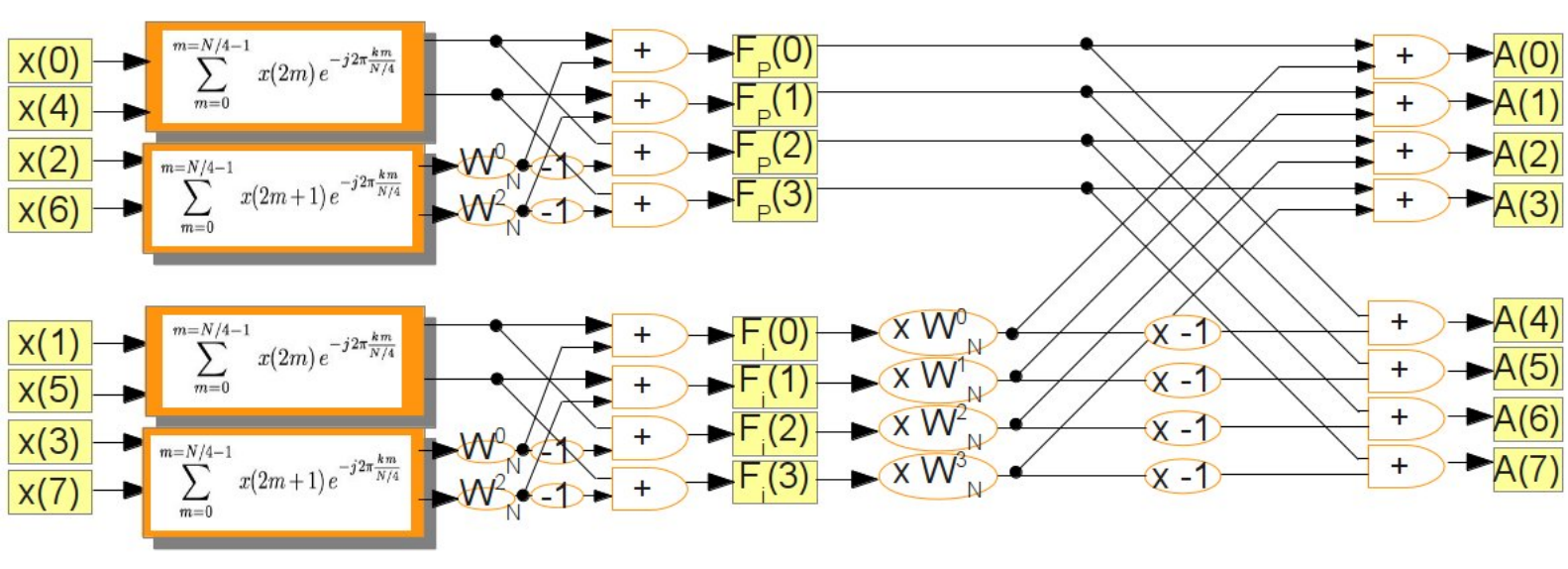


Remarquons que les mécanismes de ces deux moulinettes sont identiques, la seule différence se trouve dans les échantillons qu'on leur donne à traiter, pairs pour celle du haut, impairs pour celle du bas.

Remarquons également que celle du haut traite un nombre pair d'échantillons (quatre), on peut donc la remplacer par deux moulinettes traitant chacune la moitié de ces échantillons séparés en échantillons pairs et impairs, leurs sorties une fois multiplexés suivant le même schéma que celui utilisé sur la figure actuelle.

Et la même remarque vaut pour la moulinette du bas.

Allez, hop au boulot, traçons ce nouveau schéma qui devrait nous permettre de diviser par deux le nombre d'opérations mathématiques à effectuer.



Oui mais pourquoi les échantillons qui entrent dans les boites à gauche ne sont plus disposés dans le même ordre ? $0, 4, 2, 6$ par exemple pour les quatre du haut au lieu de l'ordre naturel des entiers pairs, à savoir $0, 2, 4, 6$???

Eh bien c'est parce qu'il faut renuméroter les échantillons du premier diagramme :

- le premier échantillon qui entrait dans la boite était le numéro 0
- le second était le numéro 2
- le troisième était le numéro 4
- le quatrième était le numéro 6.

Ces quatre échantillons étant la totalité de ce qui entre dans la TF, on les renumérote par ordre d'arrivée, puis on prend les deux pairs pour la petite moulinette du haut puis les deux impairs pour la petite moulinette suivante. Les deux pairs par ordre d'arrivée ce sont les num 0 et 4, les deux impairs ce sont les num 2 et 6.

num	arriv	
0	0	pair
2	1	impair
4	2	pair
6	3	impair

Voilà, je pense que la seule difficulté pour comprendre la recette réside dans ce point. Le reste c'est de la cuisine ordinaire.

Nous pouvons aller encore plus loin en fractionnant les TFR d'ordre $N/4$ en deux TFR d'ordre $N/8$ et ne traitant plus chacune qu'un seul échantillon.

Qu'avons-nous gagné à fractionner ainsi les transformées de Fourier ? On gagne le fait que les sommes qui brassent les échantillons s'effectuent avec des boucles bien plus petites, ce qui conduit à $8(N/8)^2$ au total au lieu de N^2 opérations (plus les quelques multiplications par les facteurs de tripatouillages...)

$8(N/8)^2 = 8N^2/8^2) = N^2/8$

Notons qu'avec un N bien plus grand, (mais toujours égal à une puissance de 2) on peut également multiplier le fractionnement jusqu'à l'obtention de cellules de base, ce qui divise d'autant le nombre d'opérations. Le diagramme devient difficile à tracer, toutefois l'ordinateur n'aura aucun mal à le suivre avec des procédures qui peuvent être récursives, mais pas obligatoirement. Ainsi pour $N = 1024$ on divisera le nombre d'opérations nécessaires (par la partie gauche du diagramme) par... 1024. Mais au total, en comptant les opérations de multiplications par les Wn et les additions, le gain sera moins important.

Remarquons que pour $N=8$, la TF paire d'ordre $N/8$ devient:

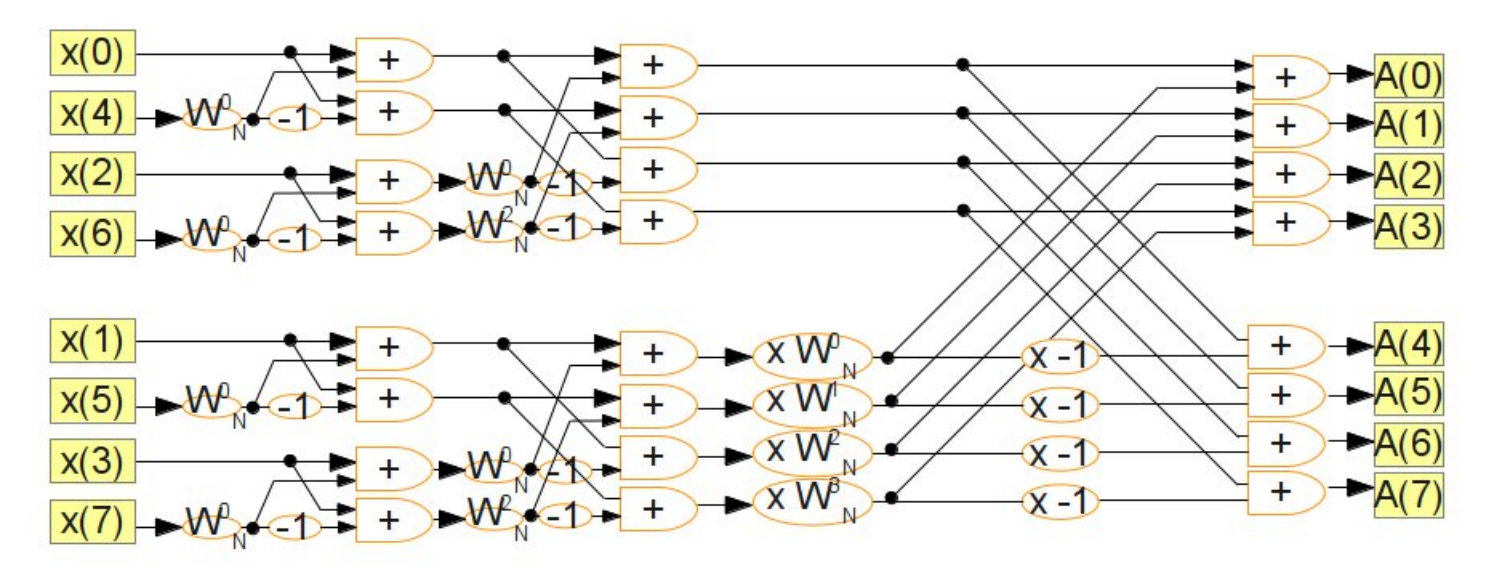
$$\begin{aligned} F_{P8}(k) &= \sum_{m=0}^{m=N/8-1} x(2m)\, e^{-j2\pi\frac{km}{N/2}} \\ &= \sum_{m=0}^{m=0} x(2m)\, e^{-j2\pi\frac{km}{N/2}} \\ &= x(0)\, e^{0} \\ &= x(0) \end{aligned}$$

et la TF impaire d'ordre $N/8$ devient:

$$\begin{aligned} F_{I8}(k) &= \sum_{m=0}^{m=N/8-1} x(2m+1)\, e^{-j2\pi\frac{km}{N/2}} \\ &= \sum_{m=0}^{m=0} x(2m+1)\, e^{-j2\pi\frac{km}{N/2}} \\ &= x(1)\, e^{0} \\ &= x(1) \end{aligned}$$

La somme se réduit ainsi à sa plus simple expression ! (On pouvait s'y attendre, les « moulinettes à échantillons » n'ayant plus alors qu'un seul échantillon à mouliner !!)

Ce qui nous donne au final le diagramme suivant :



... dans lequel les grandes sommes ont totalement disparu.

Rappel : $\boxed{W_N^n = e^{-j2\pi\frac{n}{N}} = \cos\left(2\pi\frac{n}{N}\right) - j\sin\left(2\pi\frac{n}{N}\right)}$

Les multiplications par les facteurs de tripatouillages (« twiddle factor ») sont donc des multiplications de nombres complexes avec partie réelle et partie imaginaire. On voit qu'il définissent une rotation discrète dans le plan complexe.

Ces facteurs ne seront calculés qu'une fois et gardés en mémoire dans un tableau.

Dans le programme en langage C que je vais vous proposer, j'ai créé une classe (un objet) « Complexe » pourvue d'un opérateur de multiplication interne. Pour le portage sur microcontrôleur, il sera sans doute plus pertinent de décomposer les multiplications complexes en multiplications réelles effectuées par des fonctions.

Remarque 1. (pouvant être utilisée lors de l'implantation sur microcontrôleur)

Les échantillons sont présentés en entrée suivant l'ordre $0, 4, 2, 6, 1, 5, 3, 7$. Représentons ces nombres dans un tableau avec leur écriture en binaire puis ces valeurs binaires lues de droite à gauche (inversion des bits):

dec	bin	bits inv
0	000	000
4	100	001
2	010	010
6	110	011
1	001	100
5	101	101
3	011	110
7	111	111

Nous voyons qu'il faut inverser les bits des nombres entiers naturels écrits en binaire pour obtenir les numéros des échantillons dans l'ordre qui convient au traitement.

Remarque 2. Concernant les « twiddle factor » :

Voici un petit tableau qui repésente les valeurs de twiddle factors pour une TFR avec 16 échantillons (c'est l'équivalent des diagrammes vus plus haut, pour 16 lignes au lieu de 8) :

n°ligne	x()	W	W	W	W
0	0				
1	8	0			
2	4		0		
3	12	0	4		
4	2			0	
5	10	0		2	
6	6		0	4	
7	14	0	4	6	
8	1				0
9	9	0			1
10	5		0		2
11	13	0	4		3
12	3			0	4
13	11	0		2	5
14	7		0	4	6
15	15	0	4	6	7

La seconde colonne indique les numéros d'échantillons à traiter après mise dans l'ordre « bits reverse »

Les colonnes qui suivent donnent les valeurs « k » pour les W_N^{kn} à appliquer.

On voit vite la logique des opérations : en partant de la droite vers la gauche, il faut prendre les entiers naturels (n), puis une valeur sur deux (soit $2n$), puis de nouveau une valeur sur deux ($2 \times 2n$)... etc... En numérotant « c » les colonnes de 0 à 3 en partant de la gauche, la valeur des « k » pour la colonne « c » est donc :

$k_c = 2^c n$ avec n = entiers naturels $1, 2, 3...$

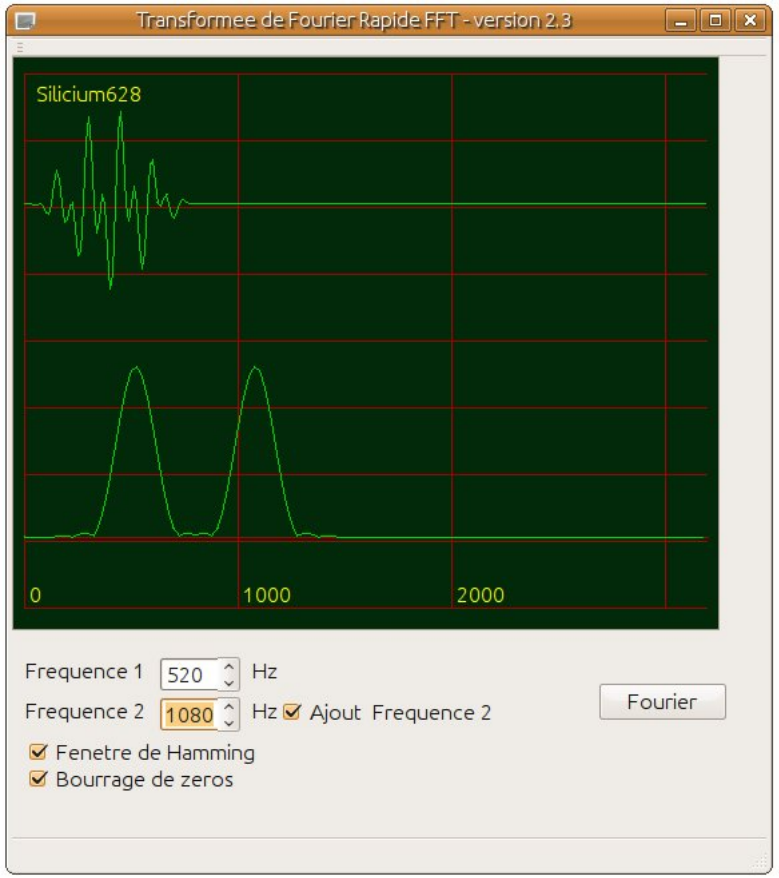
Et il faut prendre des groupes de $N/2$ puis $N/4$ puis $N/8...$ k c'est à dire un nombre $N/2^{c+1}$ de k.

Il fallait préciser tout cela car notre programme traitera 1024 échantillons en dix étapes.

Voici le coeur de ce programme en C++ et Qt4, la fonction « calcul_fourier() » qui parcourt les dix étapes, traite pour chacune de ces étapes les 1024 lignes d'échantillons, c'est à dire réalise les multiplications complexes par les twiddles factors ainsi que les additions en suivant les croisements en « papillons » vus précédemment. Comme vous pouvez le constater cette fonction comme tous le reste de mon programme est commentée en français, c'est pas du luxe !

```
void MainWindow::calcul_fourier()
{
    Complexe produit; // voir la classe "Complexe" : complexe.h et complexe.ccp
    uint decal =2;

    uint etape, e1, e2, li, w, ci;
    uint li_max=nb_ech;

    e2=1;

    for (etape=1; etape<=nb_etapes; etape++)
    {
        e1=e2; //(e1 evite d'effectuer des divisions e2/2 plus bas)
        e2=e2+e2;

        for (li=0; li<li_max; li+=1)
        {
            ci=li & (e2-1);  // ET bit à bit
            if (ci>(e1-1))
            {
                w=li_max/e2*(li & (e1 -1));  // ET bit à bit  calcul du numéro du facteur de tripatouil

                produit = tab_W[w] * tab_X[li]; // le twiddle factor est lu en memoire; le produit est

                tab_X[li]=tab_X[li-e1]-produit;          // concerne la ligne basse du croisillon; soust
                tab_X[li-e1]=tab_X[li-e1] + produit;     // concerne la ligne haute du croisillon; addit
            }
        }
    }
}
```

Et voici une saisie d'écran de son exécution sous Linux Mint :



Le signal à analyser se compose de la somme de deux trains sinus dont on fait varier les fréquences. Donc le tracé du haut se fait dans le domaine temporel, celui du bas dans le domaine fréquentiel. Il est intéressant de voir ce qui se passe lorque les fréquences se rejoingnent.