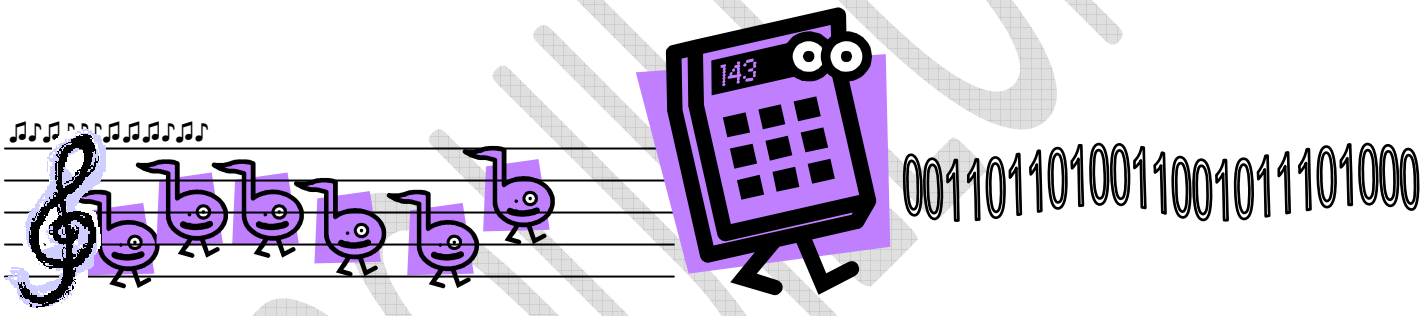
Le traitement de signal numérique sur µ-contrôleur

(Ébauche)



BROUILLON

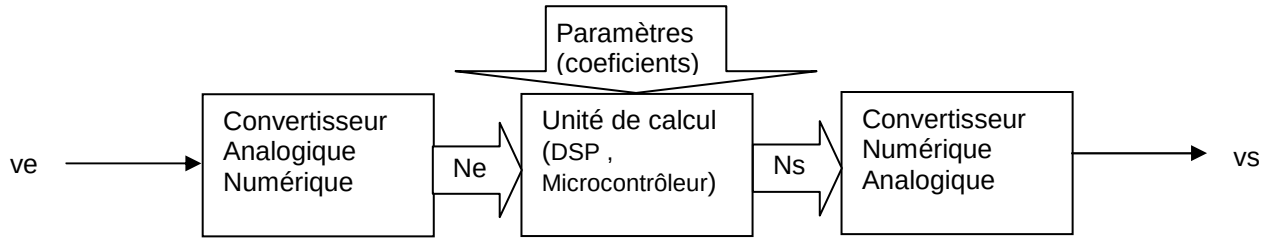
V1.0
Christian Dupaty
Lycée Fourcade
13120 Gardanne
christian.dupaty@ac-aix-marseille.fr

SOMMAIRE

Documentation :
FFT :
http://www.eisi.ch/dem/Mediatheque/Telecom%20et%20traitement%20du%20signal/Cours/FFT.pdf
Synthèse : http://systolix.co.uk
Mise en œuvre : http://www.microchip.com
Applications sur les filtres FIR et IIR : Microchip AN852

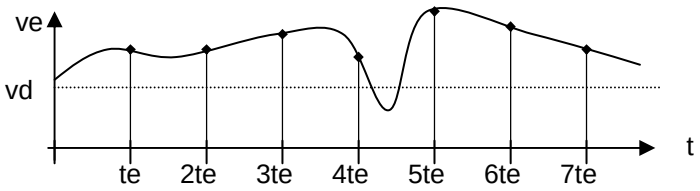
1. Le traitement de signal numérique

Structure d'un système de traitement du signal numérique

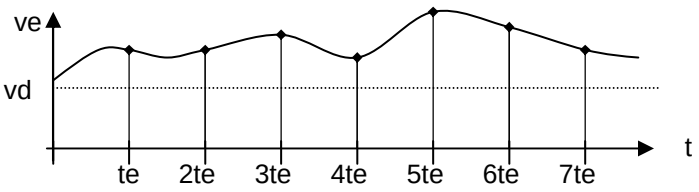


Ne représente une valeur numérique de ve à un instant t.
Ns représente une valeur numérique de ve à un instant t+tc (temps de calcul)
La fonction de transfert Ns/Ne est une fonction mathématique (généralement une convolution)

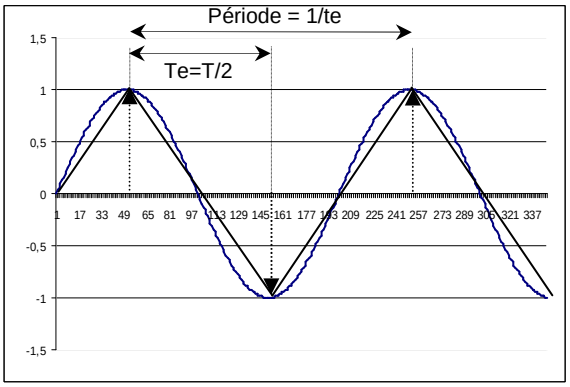
Le problème de l'échantillonnage
Le signal ve est mesuré (échantillonné) avec une période te



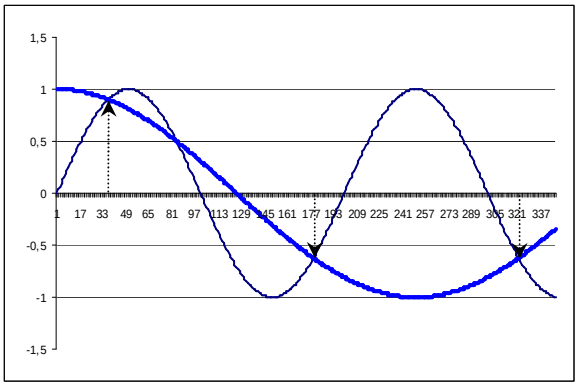
Il n'est donc connu qu'aux instants nte.
On voit ci dessus que le passage de ve sous vd n'est jamais détecté, tout se passe comme si l'on avait un signal ve comme ceci :



Il y a eu perte d'information le signal ci-dessus comporte moins d'harmoniques que le signal réel.



Ici le cas est limite Te=T/2, on peut trouver la fréquence de la sinusoïde par interpolation linéaire



Ici Te>T/2, il y confusion entre plusieurs fréquences possibles

Le critère de Nyquist établit que l'échantillonnage ne doit pas être inférieur au double de la plus grande fréquence mesurée

1.1. Les séries de Fourier

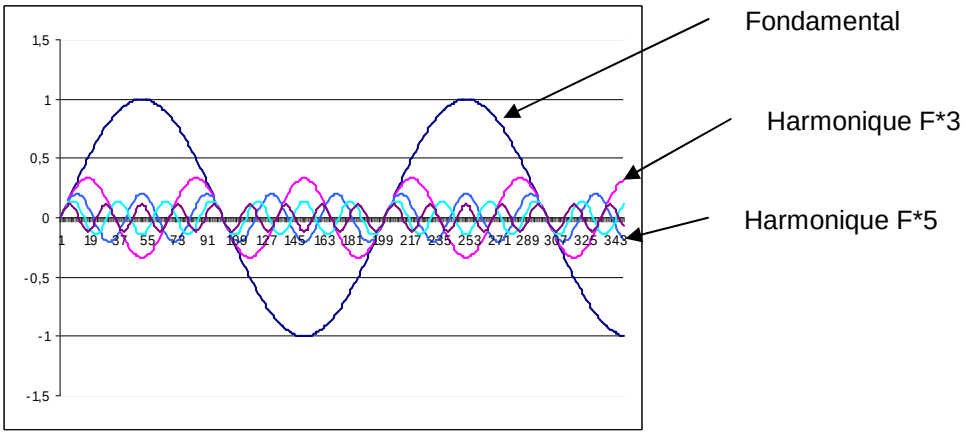
Joseph Fourier a démontré que tout signal périodique pouvait se décomposer en une de signaux sinusoïdaux. La première fréquence représente le fondamental (la fréquence propre du signal) et les autres représentent les harmoniques. Ce sont eux par exemple qui donnent le timbre à une note de musique.

Exemple reconstitution d'un signal carrée :
La décomposition d'un signal carrée en série de Fourier donne une somme de sinusoïdes multiples du fondamental et de rangs impaires
Sa décomposition en série de Fourier s'écrit

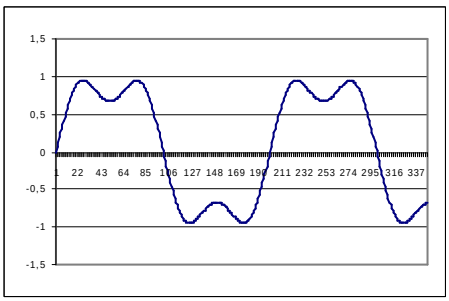
$$u(t) = \frac{4E}{\pi}\left[\sin\omega t + \frac{\sin 3\omega t}{3} + \frac{\sin 5\omega t}{5} + \frac{\sin(2p+1)\omega t}{2p+1} + ...\right]$$

(E représente l'amplitude du fondamental et ω sa pulsation)

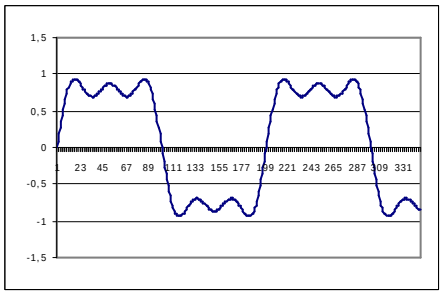
Reconstitution d'un signal carrée
Fondamental et 7 harmoniques de rangs impaires d'un signal carrée



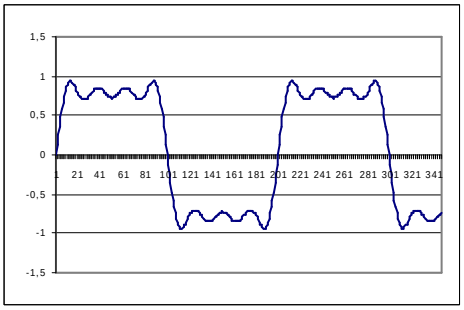
Fondamental + Harmonique 1



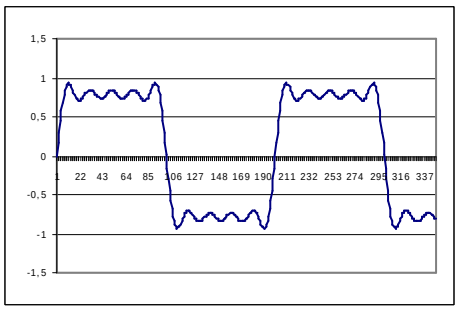
Fondamental + Harmoniques 1 et 3



Fondamental + Harmoniques 1,3 et 5



Fondamental + Harmoniques 1,3 ,5 et 7

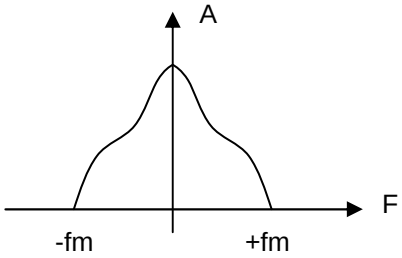


On voit que plus le nombre d'harmoniques augmentent et plus l'on se rapproche d'un signa carrée.
Remarque : Un signal carré possède une infinité d'harmoniques dont l'amplitude tend vers 0.

(Un signal carrée n'existera jamais, on aura toujours affaire à une approximation représentée par les temps de monté et de descente du signal)

1.2. Echantillonnage

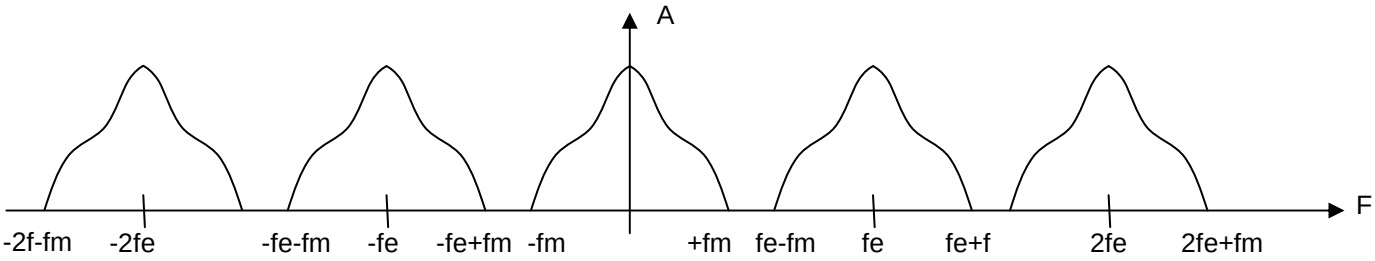
Si l'on respecte le critère de Nyquist Fe > Fmax/2 on voit qu'il ne sera pas possible de traiter numériquement un signal carrée. Dans la réalité un signal carrée n'existe pas, les fronts ont toujours un temps de monté non nul.
Le spectre d'un signal analogique est donc borné, il a généralement une forme de ce type



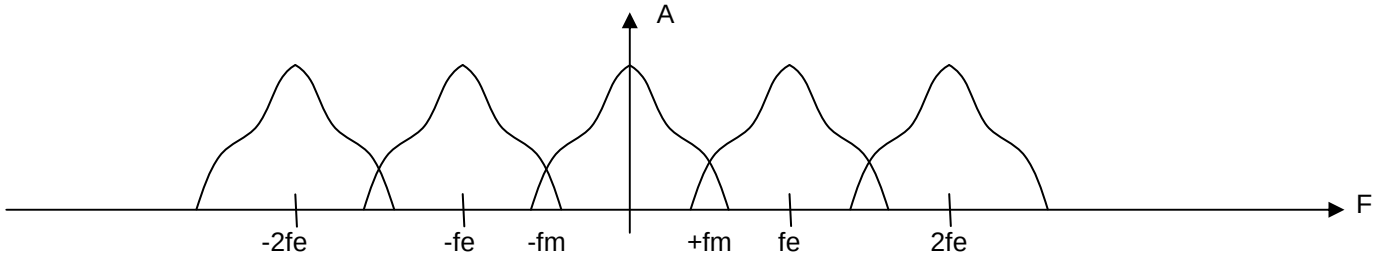
Remarque : les fréquences négatives ne représentent qu'un concept mathématique de des séries de fourrier

Théorème de Shannon

L'échantillonnage a pour effet de multiplier le signal avec une infinité de sinusoïdes multiples de Fe



Si fe <2fmax on a recouvrement de spectre (aliasing)

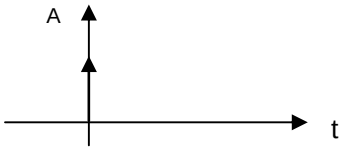


Le signal échantilloné n'est plus le signal d'origine

Le signal présent à l'entrée du CAN ne doit pas présenter de fréquences supérieurs à fe/2. On place toujours un filtre passe bas (appellé filtre anti-aliasing) avant le CAN .

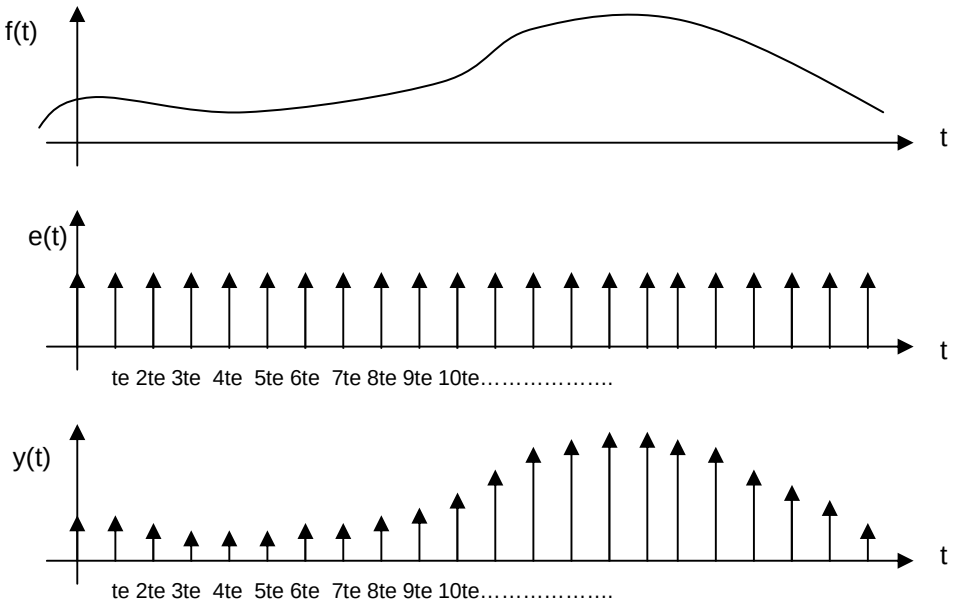
Plus la fréquence d'échantillonnage est grande :
Plus la détermination des différentes composantes d'un signal est facile.
Moins il a de contrainte sur les caractéristiques du filtre anti-repliement
Moins il a de bruit du à l'échantillonnage.

1.3. Représentation mathématique d'un signal échantillonné



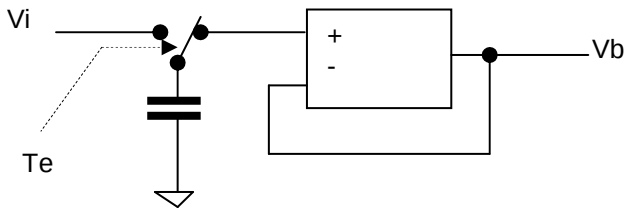
Une impulsion de Dirac est un concept mathématique, sa surface vaut $\int_{-\infty}^{\infty} \delta(t)dt = 1$

L'échantillonnage revient à multiplier le signal analogique par un peigne d'impulsions de Dirac

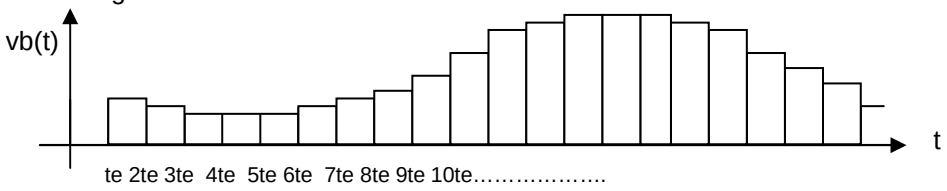


$$y(t) = \sum_{n=-\infty}^{n=\infty} f(t) \times \delta(t - nte)$$

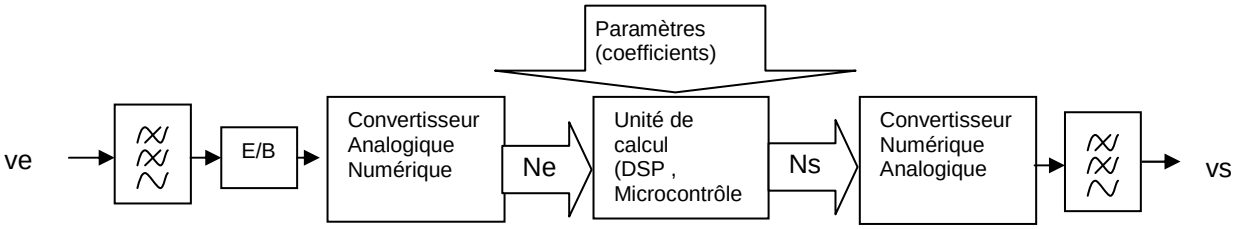
En pratique les convertisseurs analogiques numériques ont toujours un temps de conversion non nul. Il est indispensable que le signal mesuré n'évolue pas pendant la conversion. On place avant le CAN un échantillonneur bloqueur



Le signal Vb a cette allure.



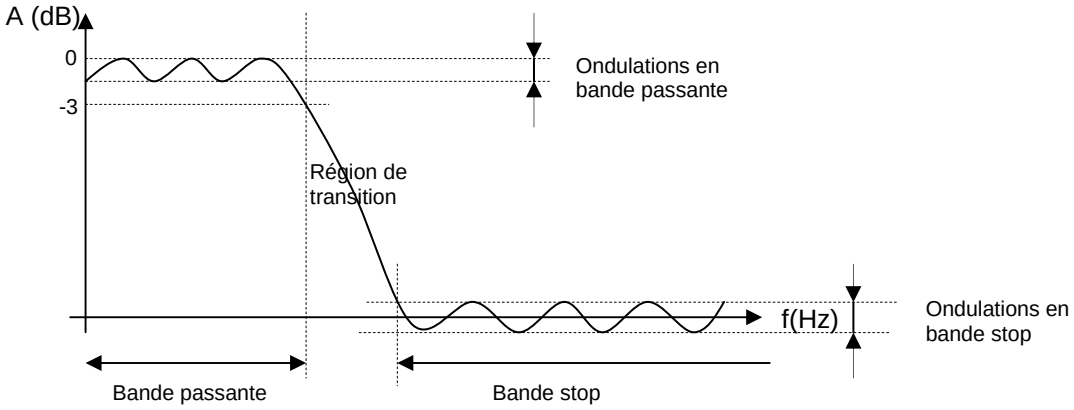
C'est un signal différent du signal à mesurer, il possède toutes les fréquences de celui ci plus des harmoniques héritées du peigne de Dirac.
La chaîne complète de traitement de signal est donc la suivante :



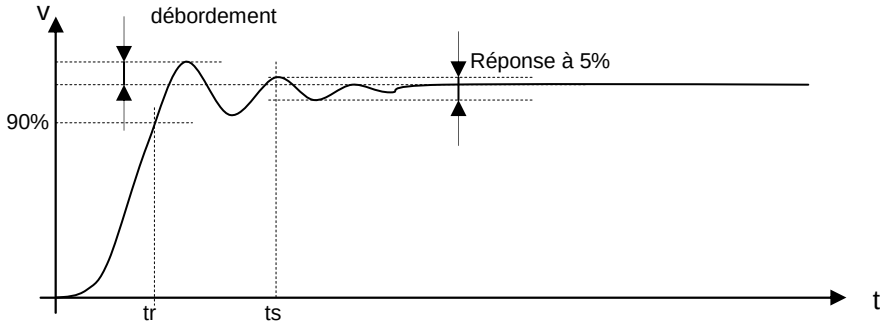
Après avoir saisit un échantillon, le calculateur définit la sortie Ns, le temps de calcul doit donc être inférieur à te. Les filtres numériques ne font appel qu'à des additions et à des multiplications, un microcontrôleur adapté au traitement de signal possédera une instruction MAC (multiplie and accumule) qui effectue une multiplication et additionne le résultat au résultat partiel.
Les DSP (Digital Signal Processor) possèdent toujours une instruction de ce type câblée, son temps d'exécution ne dépasse pas quelques 10nS à 100nS.

1.4. Caractéristiques d'un filtre

Caractéristiques d'un filtre passe bas



Réponse à un échelon d'un filtre passe bas



Le filtre est caractérisé par :
Le temps de montée à 90% (tr)
Le temps d'établissement à 5% (ts)
L'amplitude du débordement
Les oscillations finales (fréquence et nombre)

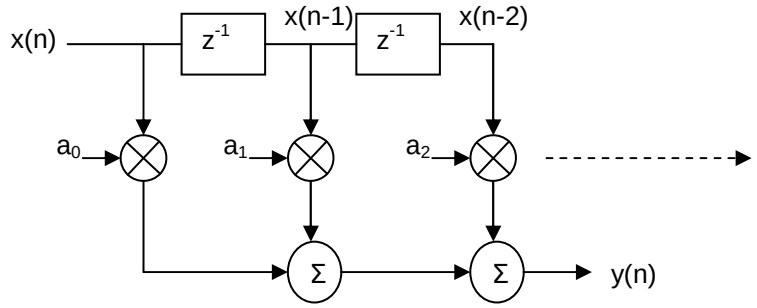
La réponse à un échelon est utilisée pour déterminer les coefficients d'un filtre numérique à réponse impulsionnelle finie (FIR)

1.5. Les filtres numériques à réponse impulsionnelle finie (FIR)

Les FIR n'utilisent que les échantillons en entrée (les valeurs précedentent des sorties n'interviennent pas. Leur sortie est toujours de la forme :

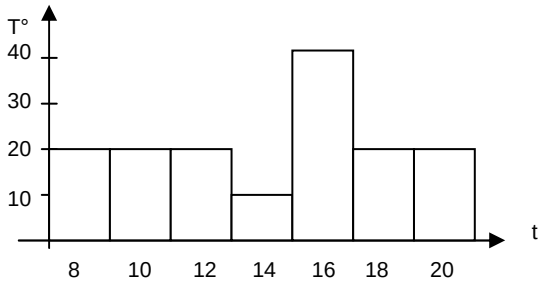
$y(_n)=a_0x(n) + a_1x(n-1) + a_2x(n-2)+ \ldots. a_mx(n-m)$

Ils sont souvent représentés comme ceci (z^{-1} représente un retard te)

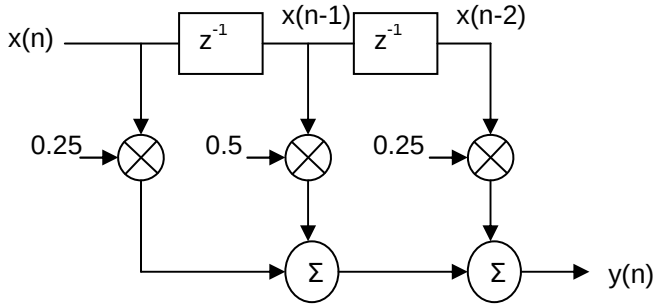


Un exemple de filtre FIR
Le tableau ci dessous donne l'évolution de la température d'un local

HEURE	PERIODE	X(N)	T°
8h	0	x(0)	20
10h	1	x(1)	20
12h	2	x(2)	20
14h	3	x(3)	10
16h	4	x(4)	40
18h	5	x(5)	20
20h	6	x(6)	20



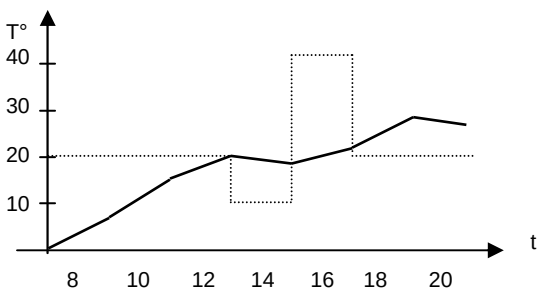
Appliquons ce « signal » au fitre précédent avec les coefficients
$a_0=0.25$, $a_1=0.50$, $a_2=0.25$



La sortie $y(n)=0.25x(n) + 0.5\ x(n-1) + 0.25\ x(n-2)$

On obtient facilement

N	Y(N)
0	5
1	15
2	20
3	18
4	21
5	28
6	25

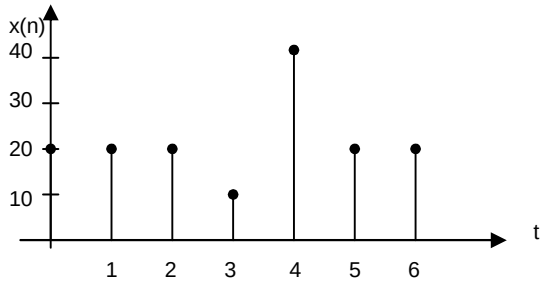


Le graphe ci dessus représente y(n) après passage dans un convertisseur analogique numérique et un filtre passe bas.
On voit que le filtre numérique est un passe bas

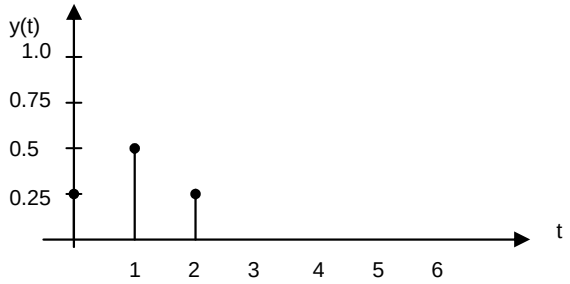
Définir les coefficients d'un filtre en fonction de sa réponse impulsionnelle

Le signal en entrée a été échantillonné, donc multiplié par un peigne de Dirac.
Chaque valeur mesurée peut être représenté son amplitude fois une impulsion de Dirac,
ex : à 8h on a

$$x(0) = \int_{-\infty}^{\infty} 20 \times \delta(t) dt = 20$$



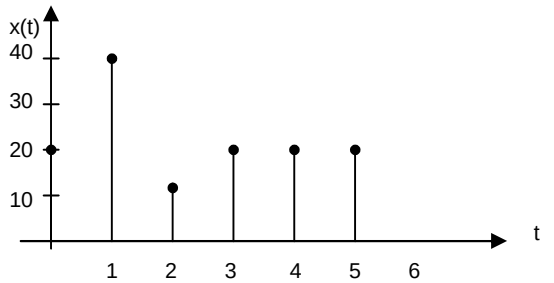
Il est facile de tracer de la même manière la réponse impulsionnelle du filtre. On la stimule avec une impulsion de Dirac à l'instant t=0.



On remarque que la réponse impulsionnelle donne exactement les valeurs des trois coeficients. Il sagit là d'une méthode utilisée par les logiciels de calcul pour produire les coeficients

Il est facile de calculer la sortie du fitre à n'importe quel moment en multipliant la réponse impulsionnelle par le train d'impulsion en entrée au moment souhaité.

Par exemple regardons x(t) à 18h (n=5) et représentons les échantillons précédents



La valeur présente (18h) est représentée à t=0 et les valeurs précédentes suivent. 8h se trouve à t=5
Par exemple, pour connaître y(5), on multiplie la réponse impulsionnelle par le train d'impulsion en entrée

y(5)=0.25x20 + 0.5x40 + 0.25x12 + 0x20 + 0x20 + 0x20 = 28

On remarque que nous avons multiplié la l'inverse de la séquence d'entrée par la réponse impulsionnelle.
Pour calculer la nouvelle valeur de y(n), il suffit de décaler la séquence d'entrée et d'introduire la nouvelle valeur x(n). Ceci s'apparente à un produit de convolution

Ecriture mathématique :
Le filtre est décrit pas l'équation : $y(_n)=a_0x(n) + a_1x(n-1) + a_2x(n-2)$
Nous avons vu qu'un délai te est représenté mathématiquement par z^{-1}
$x(n-1)=x(n) . z^{-1}$
$x(n-2)=x(n) . z^{-1} . z^{-1}$
$=x(n) . z^{-2}$

$$H(n) = \frac{y(n)}{x(n)} = a_0 + a_1 z^{-1} + a_2 z^{-2}$$

Ces filtres possèdent une phase linéaire, (la sortie ne dépend que des échantillons en entrée) mais le calcul de la sortie s'effectue sur un plus grand nombre d'échantillons que les FIR donc le temps de calcul est plus long.
Pratiquement un microcontrôleur 8 bits RISC avec un quartz 16MHz effectue une opération du type *y0=ax0+bx1+cx2+dx3+ex4+fx5+gx6+hx7+ix8+jx9* en environ 50 uS, La fréquence d'échantillonnage ne peut donc excéder 20KHz, et les fréquences filtrables 10KHz

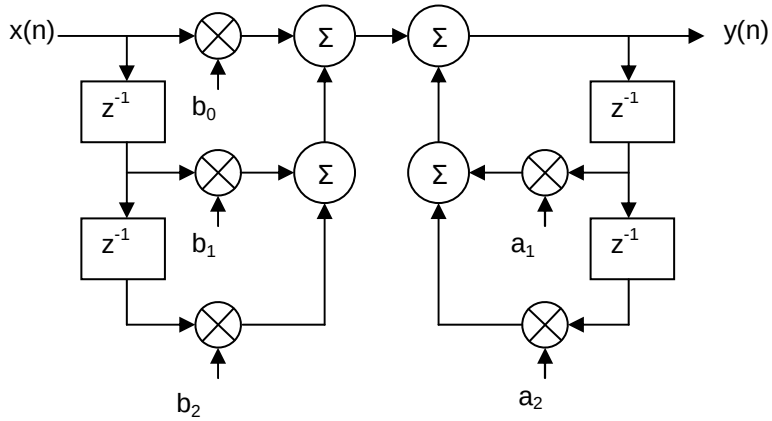
1.6. Les filtres numériques à réponse impulsionnelle infinie (IIR)

Ces filtres sont récursifs, le calcul de y(n) dépend des x(n-m) et des y(n-k), ces filtres sont donc instables et présentent une phase non linéaire et parfois très « bizarre ».
Ils sont caractérisés par une équation de ce type (appelée BIQUAD représentant un filtre du deuxième ordre) :

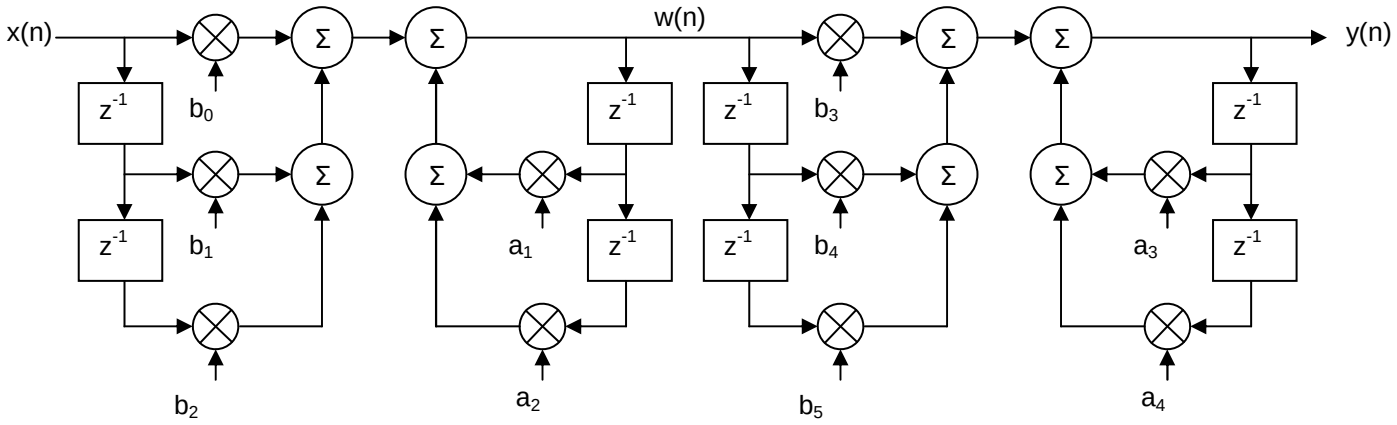
$$\frac{y(n)}{x(n)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

on en déduit que:
$y[n] = a_0*x[n] + a_1*x[n–1] + a_2*x[n–2] –b_1*y[n–1] – b_2*y[n–2]$

Représentation graphique d'un filtre d'ordre 2.



Pour obtenir un ordre 4 il suffit de cascader les cellules BIQUAD.



Rq : Il existe de nombreuses autres méthodes pour décrire les filtres IIR.

1.7. FIR vs IIR

Le choix entre un filtre FIR et IIR dépend

- ☑ Des performances recherchées
- ☑ De l'application
- ☑ De la vitesse du processeur
- ☑ De la mémoire RAM disponible

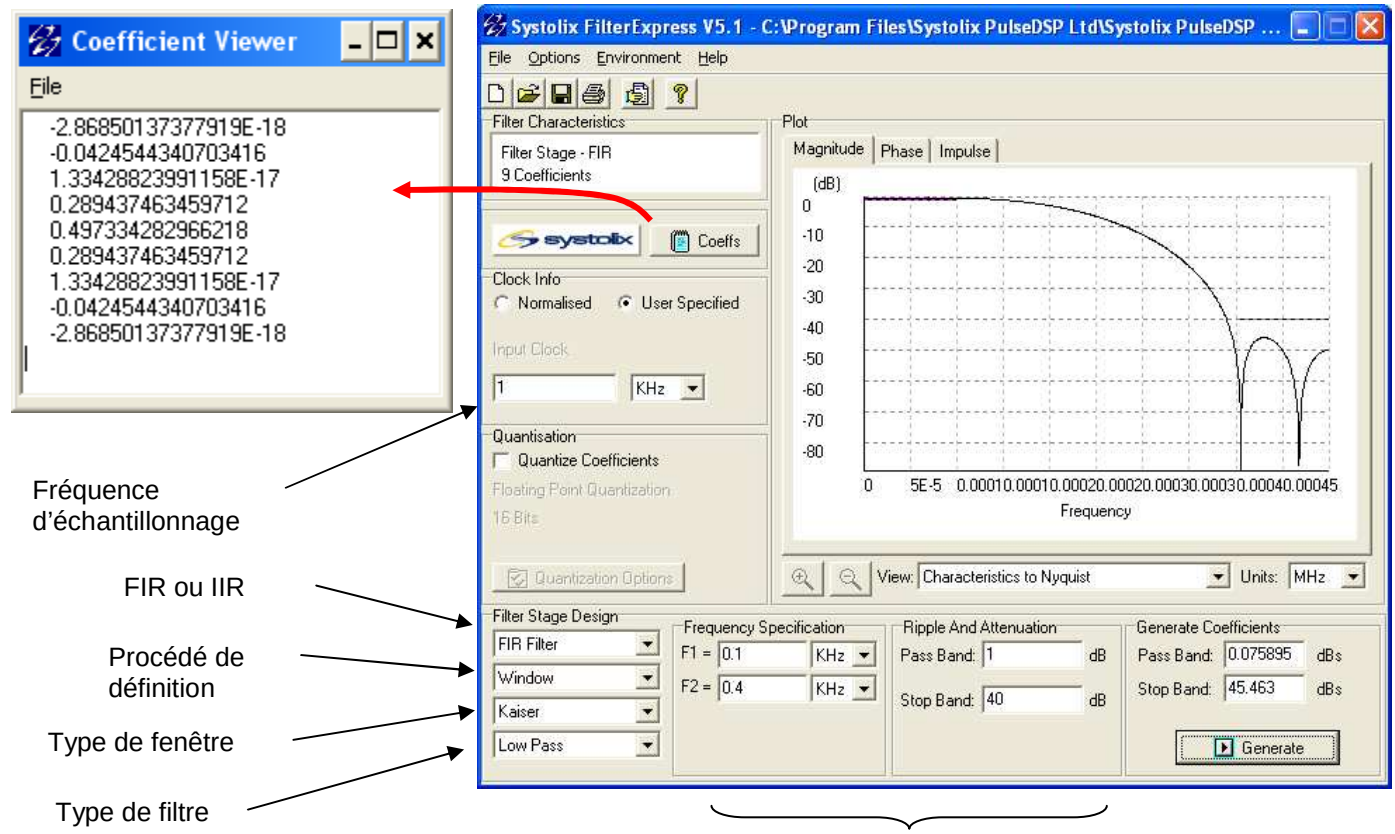
	FIR	**IIR**
Phase	Généralement linéaire	Non linéaire
Stables	oui	Pas toujours
Mémoire	plus	Moins
Temps de calcul	plus	moins
Pente (même nombre de coefficients)	moins	plus

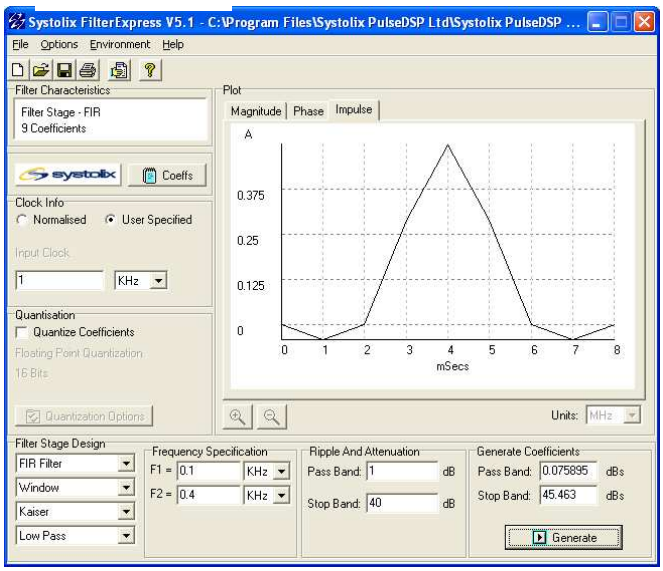
1.8. Synthèse des filtres numériques

Il existe heureusement de nombreux logiciels permettant la synthèse des filtres FIR et IIR qui éviterons aux concepteurs de long et fastidieux calculs. La société SYSTOLIX (http://www.systolix.co.uk/) propose des outils logiciels commerciaux pour le traitement de signal, en particulier pour ANALOG DEVICE. Elle propose également le logiciel FILTER EXPRESS qui est un FREEWARE permettant la synthèse générale des filtres FIR et IIR. Après téléchargement, une clé logicielle est nécessaire pour l'activer, cette clé est obtenue gratuitement par email.

1.8.1. Filtre FIR

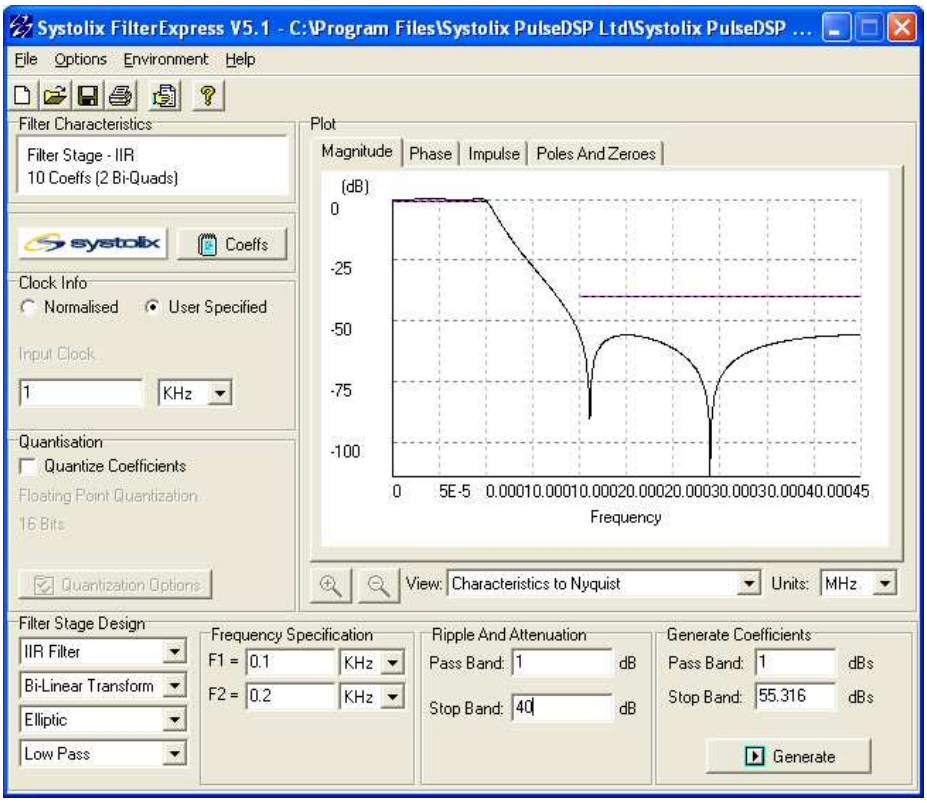
Ici le filtre possède 9 coefficients a_n, présentés dans l'ordre a_0, a_1, …. Avec Fe=1KHz
La sortie du filtre est de la forme $y=a_0xe_n+a_1xe_{n-1}+a_2xe_{n-2}+\ldots$

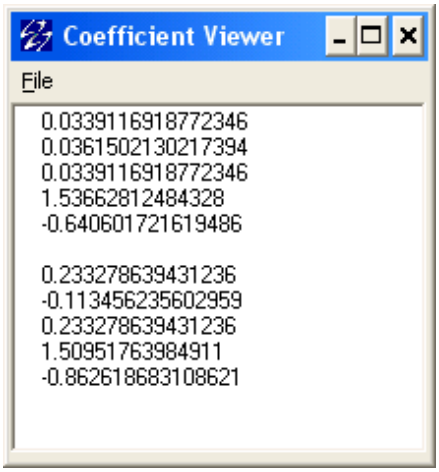




Réponse impulsionnelle du filtre FIR

1.8.2. Filtre IIR



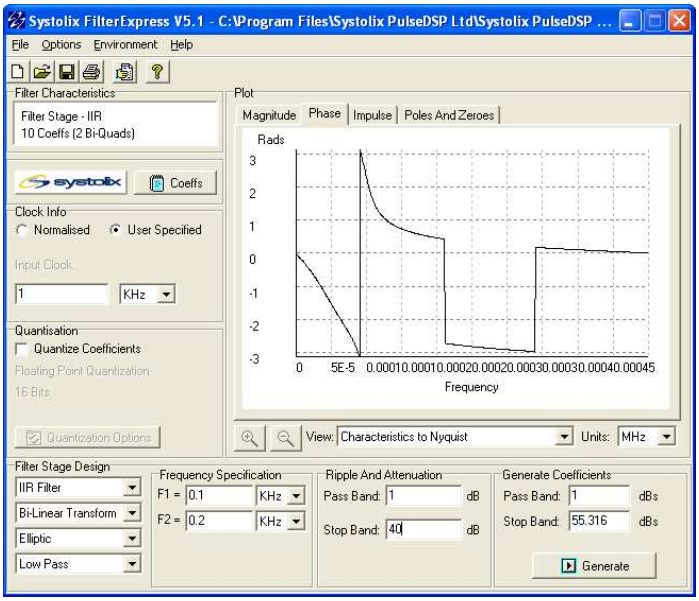


Ici le filtre possède 2 bi-quad avec 5 coefficients chacun
Les 3 premiers représentent les b_n et les 2 derniers les a_n

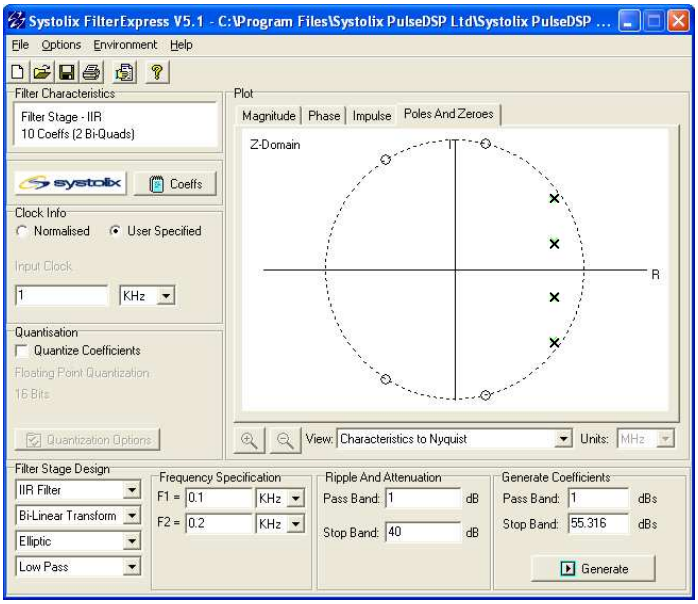
$$\frac{y(n)}{x(n)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

on en déduit que:
$y[n] = a_0*x[n] + a_1*x[n–1] + a_2*x[n–2] –b_1*y[n–1] – b_2*y[n–2]$

Phase :



Stabilité : pôles et zéros



1.8.3. Correcteur PID numérique

Après détermination des coefficients Kp, Ki, Kd par une méthode analogique classique (un excellent cours ici http://hydro.marseille.free.fr/cours/lti.pdf)

Approximation du correcteur PID par transformation bilinéaire

C'est la méthode la plus simple elle demande cependant pour être fiable une fréquence d'échantillonnage 4 à 5 fois supérieur à la fréquence de travail la plus élevée.

A partir de la loi de commande analogique

$$u(t)=K_p\varepsilon(t)+K_i\int\varepsilon(x)dx+K_d\frac{d\varepsilon(t)}{dt}$$

dont la fonction de transfert en p est

$U(p)=K_p.E(p)+K_i.\frac{E(p)}{p}+K_d.p.E(p)$ qui donne H(p) = $\frac{U(p)}{E(p)}=K_p+\frac{K_i}{p}+K_d.p$

La transformation bilinéaire consiste à remplacer p par $p=\frac{2}{te}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$

d'où $H(z)=K_p+\frac{K_i\cdot te(1+z^{-1})}{2\cdot(1-z^{-1})}+K_d\cdot\frac{2}{te}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$

$x.z^{-1}$ correspond à l'échantillon x(n-1)

après réduction on obtient $u(n)=a_0e(n) + a_1e(n-1) + a_2 e(n-2) + b_2u(n-2)$
avec

$a_0=K_p+K_i\frac{te}{2}+K_d.\frac{2}{te}$

$a_1=K_i.te-K_d.\frac{4}{te}$

$a_2=-K_p+K_i\frac{te}{2}+K_d.\frac{2}{te}$

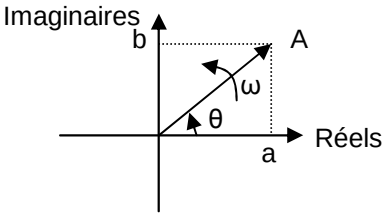
b2=1

On voit qu'un correcteur PID numérique aura la même structure logicielle qu'un filtre récursif.

1.9. Transformée de Fourier discrète (DFT)

Un peu de maths (pas beaucoup)

On peut représenter une sinusoïde à l'aide d'un modèle complexe



$$x(t) = a + j.b$$
$$A = \sqrt{a^2 + b^2}$$
$$\theta = \omega.t = \tan^{-1}\frac{b}{a}$$

On peut également représenter ce nombre complexe sous la forme polaire : $x(t) = A.e^{j(\omega.t)}$

Avec $e^{j(\omega.t)} = \cos(\omega.t) + j.\sin(\omega.t)$

On peut écrire de même

$e^{j\theta} = \cos\theta + j.\sin\theta$ et $e^{-j\theta} = \cos\theta - j.\sin\theta$

avec $\theta = \omega.t + \alpha$ ou $\theta = n.\omega.Te + \alpha$ dans le cas d'un signal échantillonné

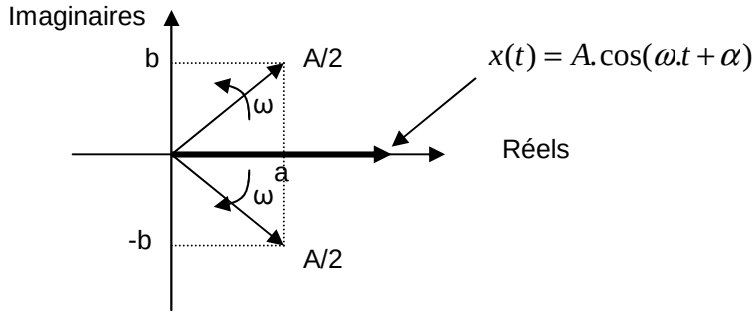
On en déduit

$$\cos\theta = \frac{e^{j\theta} + e^{-j\theta}}{2} \text{ et } \sin\theta = \frac{e^{j\theta} - e^{-j\theta}}{2j}$$

Cela montre qu'un signal sinusoïdal peut être décrit par la somme de deux vecteurs

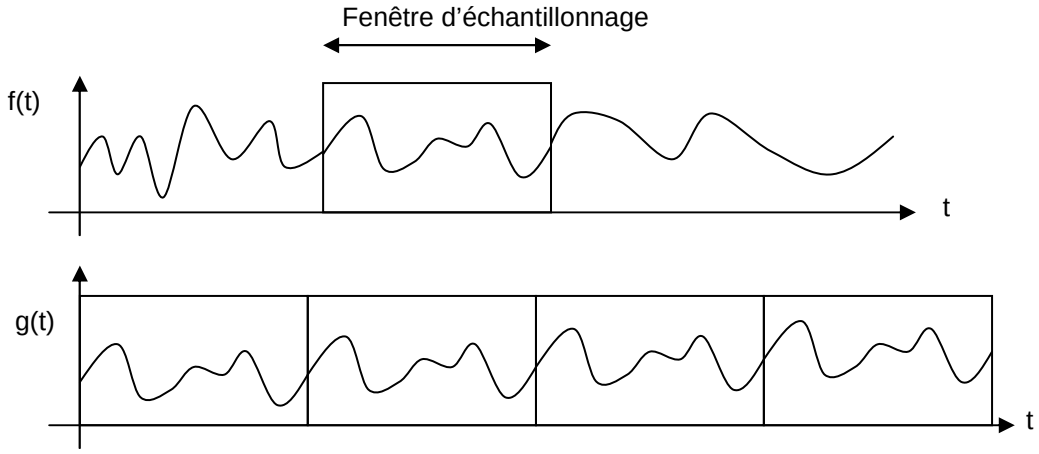
$$x(t) = A.\cos(\omega.t + \alpha) = \frac{A}{2}\left(e^{j(\omega.t+\alpha)} + e^{-j(\omega.t+\alpha)}\right)$$

dont la représentation graphique est



Ces propriétés montrent que tout signal réel peut être représenté par la somme de deux nombres complexes conjugués. Ceci permet de mettre simplement en place l'approche mathématique de la transformée de Fourrier discrète.

Après échantillonnage d'un signal, la FFT permet d'isoler certaines composantes spectrales de ce signal (une fenêtre). On connaîtra donc les harmoniques d'un signal fenêtré supposé périodique.



Après calcul de la FFT le spectre obtenu sera celui du signa g(t) et non f(t), il est donc évident comme dans le cas des filtres que la précision dépendra du nombre d'échantillons et du rapport fe/fm. (Fréquence d'échantillonnage / fréquence maximum du signal)

La décomposition en séries de Fourier d'un signal f(t) donne (on ne cherche pas ici à la démontrer)

$x(t) = \sum_{k=-N}^{N} C_k e^{j(\omega_k t)}$ C représente les amplitudes et ω les pulsations

Dans la réalité la plupart des signaux ne sont pas périodiques, la somme précédente devient une intégrale

$x(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\omega) \cdot e^{j(\omega t)} \cdot d\omega$ X représente les amplitudes et ω les pulsations

On peut déduire X(ω) de cette équation, on obtient la transformée de Fourier

$X(\omega) = \int_{-\infty}^{+\infty} x(t) \cdot e^{j(-\omega t)} \cdot dt$ qui représente l'amplitude des raies du signal x(t)

La décomposition en séries de Fourier discrète du signal échantillonné x(n) avec la période Te donne

$x(n) = \sum_{k=-N}^{N} C_k e^{j(k\omega_0 T_s n)}$ Représentant les N composantes de x(n)

On pratique de même avec le signal échantillonné et l'on obtient la transformée de Fourier discrète (Discrete Fourier Transform)

$x(n) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} X(\omega) \cdot e^{j(\omega T_e n)} \cdot d(\omega T_s)$ et $X(\omega) = \sum_{n=-\infty}^{\infty} x(n) \cdot e^{j(-\omega T_s n)}$

Ces deux équations forment les équations de référence de la transformée de Fourier discrète (DFT)

Considérations pratiques :
Dans la réalité l'échantillonnage n'est pas infini, et la somme des x(n) est donc fini, il y a un effet de fenêtrage, une erreur est donc introduite ici.

Le spectre résultant du fenêtrage est donné par
$X_N(\omega) = X(\omega) * F(\omega)$ ou représente le produit de convolution du spectre de X par celui de la fenêtre F. Un produit de convolution est simple une somme de produit. On peut l'écrire comme cela

$$X_N(\omega) = \sum_{r=0}^{N-1} X(r) \cdot F(N-r)$$

Une fenêtre rectangulaire serait idéale, malheureusement elle est pratiquement impossible à construire, on a donc recours à des compromis. (Fenêtre de Haming par exemple)

Le spectre d'un signal échantillonné se répète avec une période ω_e et les raies sont espacée de δ. SI N représente le nombre d'échantillons de la fenêtre on a $\omega_e = N.\ \delta$

Chaque raie peut donc s'écrire $X(k\delta) = X(\delta) = \sum_{n=0}^{N-1} x(n) \cdot e^{j(-k\delta T_s n)}$

On sait que $\omega_e = \frac{2\pi}{T_e}$ et $\delta = \frac{\omega_e}{N}$

On peut donc écrire $X_N(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j(\frac{2\pi kn}{N})}$

Comme $e^{jx} = \cos(x) + j\sin(x)$ on a $X_N(k) = \sum_{n=0}^{N-1} x(n) \cdot \left[\cos(\frac{2\pi kn}{N}) + j\sin(\frac{2\pi kn}{N}) \right]$

Il est donc maintenant possible de calculer l'amplitude et la phase de chacune des raies composant le signal

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

Et l'on obtient finalement l'équation utile de la DFT

$$X_N(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn}$$

Les W_N^{kn} sont une suite de nombres constants. La DFT n'est donc qu'une somme de produits.
La précision de l'analyse spectrale dépend de N, plus N sera grand et meilleur sera l'analyse (plus δ sera petit)
Plus N est grand et plus le calcul sera long, le nombre de multiplications complexes vaut N^2.
L'algorithme de transformée de Fourier rapide (Fast Fourier Transform ou FFT) réduit considérablement le temps de calcul.

L'algorithme de FFT le plus connu est celui de Cooley-Tukey, également appelé algorithme de réduction à base 2 dans le domaine temporel. Cet algorithme requiert un nombre d'échantillons qui soit une puissance de 2. Par exemple : N=128, 256, 4096, etc. D'autres algorithmes FFT existent qui requièrent d'autres exigences. Dans le cas d'une FFT selon l'algorithme de Cooley-Tukey, le nombre d'opérations est considérablement réduit : $\frac{N}{2}\log_2 N$ multiplications complexes

Cet algorithme est particulièrement bien décrit ici :

http://www.eisi.ch/dem/Mediatheque/Telecom%20et%20traitement%20du%20signal/Cours/FFT.pdf

1.10. Détection de fréquence, algorithme de Goetzel

On rappelle que l'expression de la DTF (Discret Fourier Transform) pour un signa x(n) échantillonné à la fréquence Fe lorsque l'on considère N points de ce signal est donnée par

$$X_N(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j(\frac{2\pi kn}{N})}$$

Lorsque l'on ne désire connaître l'amplitude que d'une fréquence l'algorithme de Goetzel permet d'effectuer ce calcul de façon récursive.

On pose $W_N = e^{-j(\frac{2\pi}{N})}$ et $W_N^{kN} = W_N^{-kN} = 1$ d'où $X_N(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{-k(N-n)}$

On pose $Y_k(m) = \sum_{n=0}^{m} x(n) \cdot W_N^{-k(m-n)}$ et l'on obtient $X_N(k) = Y_k(N-1) \cdot W_N^{-k}$

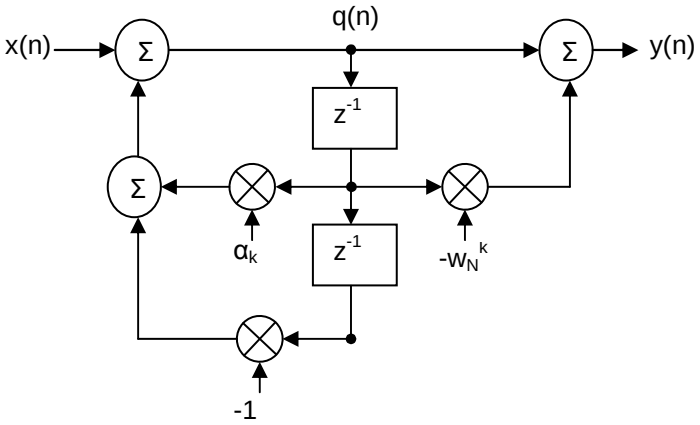
On obtient la relation $Y_k(m) = x(m) + Y_k(m-1) \cdot W_N^{-k}$ et $Y_k(0) = x(0)$

Cette relation est équivalente au filtre : $H(z) = \frac{1}{1 - W_N^{-k} \cdot z^{-1}} = \frac{1 - W_N^k \cdot z^{-1}}{1 - 2\cos\left(\frac{2\pi k}{N}\right) \cdot z^{-1} + z^{-2}}$

Le module du pôle est égal à 1 mais comme le nombre de points considéré est fini, il n'y aura pas d'instabilité.

On pose $Y(z) = H(z)X(z) = D(z)N(z)X(z)$ avec $N(z) = 1 - W_N^k \cdot z^{-1}$

Et $D(z) = \frac{1}{1 - \alpha_k \cdot z^{-1} + z^{-2}}$ avec $\alpha_k = 2\cos\left(\frac{2\pi k}{N}\right)$



L'algorithme de calcul de q(n) est la suivant :

$q(0) = x(0)$ puis $q(n) = x(n) + \alpha_k q(n-1) - q(n-2)$ pour n=1 à N-1

X(k) est obtenu par

$$y(n) = q(n) - W_N^k q(n-1)$$
$$X(k) = y(N-1)$$
$$X(k) = (q(N-1) - W_N^k q(N-2))W_N^{-k}$$

Pour connaître X(k) il faudra donc attendre de connaître q(N-1) et q(N-2)
X(k) est un nombre complexe dont on désire connaître le module

$$|X(k)|^2 = |q(N-1) - W_N^k q(N-2)|^2$$
$$|X(k)|^2 = q(N-1)^2 + q(N-2)^2 - 2R(W_N^k q(N-2)q(N-1)$$
$$|X(k)|^2 = q(N-1)^2 + q(N-2)^2 - \alpha_k q(N-2)q(N-1)$$

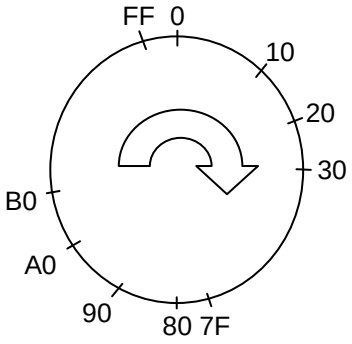
2. La mise en œuvre des filtres numériques

2.1. Numérisation et représentations binaires

Les nombres utilisés par une ALU sont toujours entiers et finis.
Le traitement de signal impose des calculs (additions et multiplications) sur des nombres réels. D'ou la nécessité de représenter les nombres signés ainsi que leur partie décimale.

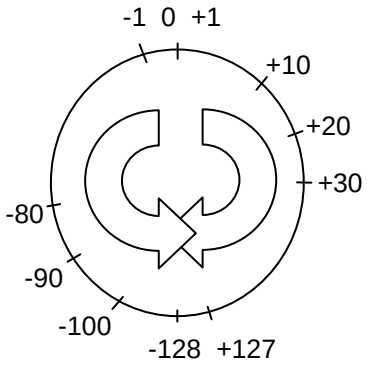
Représentation signée d'un nombre entier - codage binaire en complément à 2

Exemple de codage d'entiers positifs pour un octet
On voit ici que FF+1 donnera 00 avec une indication de retenue (généralement dans un bit appellé carry). Il y a débordement



Pour les entiers relatifs on voit que 0 - 1 donne FF, en binaire complément à 2, FF représente la valeur -1. FE représente -2 etc…

On peut alors représenter les nombres comme cela



Le débordement s'effectue maintenant par 127+1 qui donnera -128 (7F+1 = 80)
On remarque que le bit de poids fort représente maintenant le signe. 0 pour + et 1 pour –

Conversion binaire signé en binaire complément à 2

Pour inverser un nombre binaire en binaire signé, on complémente tous les bits pluis l'on ajoute 1 au résultat
Ex : -5 (codage sur 8 bits)

+5	0000 0101
Complément	1111 1010
+1	1111 1011
-5	F B

Addition et soustraction en binaire complément à 2.

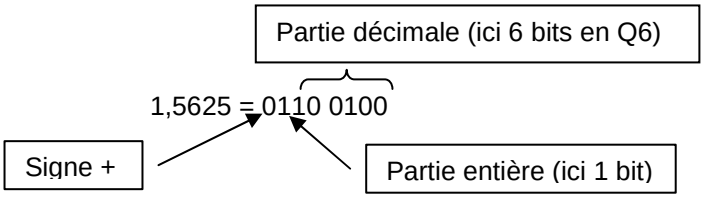
-5 – 2 = - 7 FB – 2 = F9 FB 1111 1011 -2 0000 0010 F9 1111 1001	-5 + 11 = + 6 FB + B = + 6 FB 1111 1011 +B 0000 1011 +6 1 0000 0110 (résultat sur 9 bits) (la retenue indique un passage par 0 et sera ignorée)

Il s'agit maintenant de représenter la partie décimale d'un nombre réel.
Calcul en virgule fixe

Le calcul s'effectue en codage Q
On choisit (selon les besoins) un nombre de bits pour la partie entière et un nombre de bits pour la partie décimale. Les bits de la partie entière représentent les puissances de 2^n et ceux de la partie décimale les 2^{-n}. L'erreur absolue sur un réel représenté en virgule fixe est toujours inférieure à 2^{-m}

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
256	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625

Exemple : codage de la valeur 1,5625 sur un octet positif en Q6
6 chiffres binaires décimaux
1 chiffre binaire pour la partie entière (0 ou 1 donc)
le poids fort représente le signe (0 pour +, 1 pour -)



$= 1*2^0+1*2^{-1}+0*2^{-2}+0*2^{-3}+1*2^{-4}+0*2^{-5}+0*2^{-6} = 1+0.5+0.0625= 1.5625$

-1,5625 = 11011100 en complément à 2
Le complément à 2 de la partie entière 01 donne 11
Le complément à 2 de la partie décimale 100100 donne 011100

Addition de nombres codés en Qn

Comme en décimal, il faut prendre soin d'aligner les 2^n correspondant avant addition, ce qui revient à affecter des décalages à gauche (ajouter des 0) sur le nombre codé avec le plus petit Qn

```
  117,0000   (codé en Q0)              0111 0101,000000  (codé en Q6)
+    1,5625   (codé en Q4)            +          01,10 0100 (codé en Q6)
-----------                           ---------------------------
 118,5625   (codé en Q4)              01110110,100100  (résultat en Q6)
                                      118             , 5625

 14,7500 (codé en Q4)                 01110,11 0000   (codé en Q6)
+ 1,5625 (codé en Q4)               +      01,10 0100   (codé en Q6)
-----------                          ---------------------
16,3125 (codé en Q4)                0 0010 00,01 0100  (résultat en Q6)
                                      16         , 3125
```

Les nombres négatifs doivent être codés en binaire complément à 2
Ex : -14,75
+14 = 1110 donc -14 = 10010
75 codé décimal donne 1100 don -75 codé décimal donne 0100

```
- 14,7500 (codé en Q4)               10010,01 0000    (codé en Q6 complément à 2)
 + 1,5625 (codé en Q4)              +      01,10 0100   (codé en Q6)
 -----------                        ---------------------
-13,1875 (codé en Q4)                100 11,11 0100          (résultat en Q6 complément à 2)
                                     -13       , 1875
```

(Conversion de la partie décimale /110100 = 001011 plus 1 soit 001100 soit $2^{-3}+2^{-4} = 0.1875$)

Dans une somme, les deux nombres doivent être codés suivant le même Qn

Multiplication de nombres codés en Qn

```
  117,0    (codé en Q0)          0111 0101,0 (codé en Q0)
x1,5625   (codé en Q4)          x01,10 0100   (codé en Q6)
-----------                     ------------------
182.8125 (codé en Q4)           0010 1101 10,11 0100  (résultat sur 16 bits en Q6)
                                182              , 8125

  14,75   (codé en Q2)            01110,110   (codé en Q3)
x1,5625 (codé en Q4)            x 01,10 0100   (codé en Q6)
-----------                     ------------------
23.046875 (codé en Q6)    0010 111,0 0001 1000
                                23          , 046875 (résultat sur 16 bits en Q9)
```

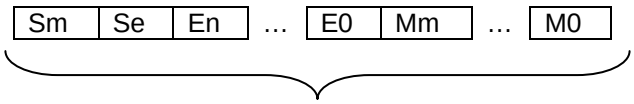
Dans un produit de deux nombres codés en Qa et Qb, le résultat est codé en Q(a+b)
Un calcul en codage Q donne un résultat en codage Qn. Pour récupérer la partie entière d'un résultat on peut simplement effectuer n décalage à droite du résultat.

Calcul en virgule flottante
Ce type de codage est très performant car il permet de coder les nombres réels sur une très grande plage numérique avec un format binaire constant. Il demande en revanche une manipulation binaire plus complexe pour les additions et les multiplications et généralement utilisé par les langages évolués comme le C.

$$r = Sm.M.10^{Se.E}$$

r : réel à coder
Sm : signe de la matrice (0 = positif, 1 = négatif)
M : matrice
Se : signe de l'exposant
E : exposant

Un nombre réel codé en virgule flottante a cet aspect :



16, 32, 64 ou128 bits suivant les processeurs et les compilateurs

Nb bits	Exposant N	Mantisse M
16	4	10
32	8	22
64	14	48

La mantisse représente les chiffres significatifs d'un réel inférieur à 0 codé en 2^{-n}
Par exemple 245,36 a une mantisse égale à +24536 et un exposant égale à +3 : $245,36=0.24536.10^{3}$

Exemple de codage en virgule flottante :
$-5,635 = -0,5635.10^{1}$ et $2^{-1}+2^{-4}+2^{-10}=0.5634765$

Sm	Se	E3	E2	E1	E0	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1

1 0.5634765

2.2. Mise en œuvre des filtres FIR et IIR

Il s'agit de programmer une équation de la forme
$Y[n] = a_0.x_n+a_1.x_{n-1}+a_2.x_{n-2}+\ldots$ Pour les filtres FIR ou
$y[n] = a_0.x_n + a_1.x_{n-1} + a_2x_{n-2} - b_1y_{n-1} - b_2y_{n-2}$ pour chaque BI-QUAD des filtres IIR

Les coefficients a_n et b_n sont des nombres réels positifs ou négatifs, les échantillons x_{n-x} sont des nombres entiers non signés
En C les opérations sur les nombres réels ne posent pas de problème. Cependant les temps de calcul sont beaucoup plus longs que sur les nombres entiers. Sur un uC PIC18 de Microchip une multiplication 8x8 non signé dure un cycle d'horloge alors qu'il faut compter une centaine de cycle pour une multiplication entre deux nombres réels.
Exemple en langage C de filtre FIR : application sur 68HC11 sur Control Boy F1
(http://www.controlord.fr)

```
/*Filtre Numérique FIR passe bas
/* OC3 le cadence  */
/* CD 12/2000 */
//y0=ax0+bx1+cx2+dx3+ex4+fx5+gx6+hx7+ix8+jx9
#include <hc11.h>       /*déclaration des bibliotheques*/
#include <lycee.h>     /* pour analogin et analogout
#define tech 22222u   /* échantillonage pour 180Hz (5.55mS) sur CBOYF1 16MHz*/
#define a -0.0385324
#define b 0.0762931
#define c 0.0347435
#define d 0.3035086
#define e 0.4541574
#define f 0.3035086
#define g 0.0347435
#define h -0.0762931
#define i -0.0385324
#define j 0.0

float x0,x1,x2,x2,x3,x4,x5,x6,x7,x8,x9,y0;

#pragma interrupt_handler oc2Int   /*déclaration de l'IT d'oc5*/
void oc2Int(void)                  /*routine d'interruption*/
{
//      PORTA=1;
        x9=x8;
        x8=x7;
        x7=x6;
        x6=x5;
        x5=x4;
        x4=x3;
        x3=x2;
        x2=x1;

        x0=0.0196*((float)(analogin(7)));
        // durée de calcul : environ 4.7 mS avec Q=16MHz
        y0=a*x0+b*x1+c*x2+d*x3+e*x4+f*x5+g*x6+h*x7+i*x8+j*x9;
        x1=x0;
        analogout(1,(int)(51.0*y0));
        TOC2 +=tech;
        TFLG1 |= 0x40;
//      PORTA=0;
}

#pragma abs_address:0xFFE6      /*routine d'IT , vecteur RESET*/
void (*oc2int_vector)(void) = oc2Int ; /*adresse de oc5Int en FFEO*/
#pragma end_abs_address

void main(void)                         /*programme principal*/
{
        OPTION|=0x80;  //active CAN
        TMSK1 |=0x40;   /*IT de oc2 activée */
        TCTL1 &=0x3f;   /*aucune action définie sur le port A*/
        DDRA=1;
        asm(" cli ");   /*autoriser les interruptions*/
        while(1);
}
```

Réalisation d'un filtre FIR avec des coefficients signés sur 8 bits.
Les nombres réels ne seront généralement pas utilisés dans les algorithmes de filtres numériques, on préfèrera utiliser des nombres entiers qui permettront une vitesse de calcul bien plus élevée.

Le filtre obtenu n'aura bien évidement pas exactement les mêmes caractéristiques que celui recherché.

1) les coefficients doivent être convertis en nombres entiers signés.
Les coefficients réels sont toujours inférieurs à 1. Ils représentent un pourcentage de l'échantillon considéré.
Voici un exemple de coefficients obtenus avec FILTER EXPRESS

a_0= -2.86850137377919E-18
a_1= -0.0424544340703416
a_2= 1.33428823991158E-17
a_3= 0.289437463459712
a_4= 0.497334282966218
a_5= 0.289437463459712
a_6= 1.33428823991158E-17
a_7= -0.0424544340703416
a_8= -2.86850137377919E-18

Les coefficients a_0, a_2, a_6 et a_8 peuvent être considérés comme nuls
$a_1 = a_7$ et $a_3 = a_5$
On fait correspondre le plus grand coefficient à 0x7F (127) les autres sont calculés par rapport à celui-ci (en complément à 2).
$a8_4$= 127
a_3/a_4= 0.289/0.497 = 0.58
$a8_3$= 127x0.58 = 73.68 => 74 = 0x4A

a_1/a_4= -0.042/0.497 = -0.084
$a8_1$= 127x-0.084 = -10,7 => -11 = 0xF5

2) On calcule y1[n] en ajoutant 128 à chaque coefficient ce qui revient à les rendre non signé. On peut alors utiliser les instructions de multiplication matérielle intégrées dans les uC (PIC, AVR ou HC08)

$y_1[n] = x[n]*(a0 + 128) + x[n–1]*(a1 + 128) + x[n–2]*(a2 + 128)+...+x[n–N+1]*(aN-1 + 128)$

Ici
$y_1[n] = x[n–1]*(0xF5 + 128) + x[n–3]*(0x4A + 128) + x[n–4]*(127 + 128) + x[n–5]*(0x4A + 128) + x[n–7]*(0xF5 + 128)$
$y_1[n] = x[n–1]*(0x75) + x[n–3]*(0xCA)+ x[n–4]*(0xFF) + x[n–5]*(0xCA) + x[n–7]*(0x75)$

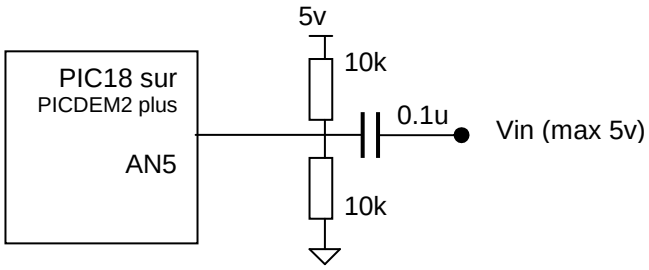
Ainsi que X[n] = x[n]*128 + x[n–1]*128 +...+ x[n–N+1]*128,
X[n] représente les n échantillons multipliés par 128.

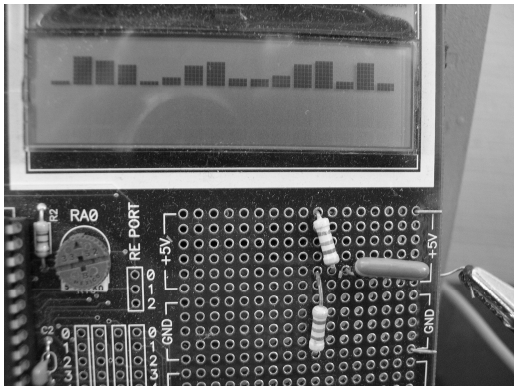
Il n'y a plus qu'a calculer $y[n]=y_1[n]-X[n]$

Microchip propose dans son document AN852
(http://www.microchip.com/1010/suppdoc/appnote/all/an852/) des exemples de bibliothèques écrirent en assembleur en calcul sur 8bits pour les filtres FIR et IIR

2.3. Mise en oeuvre de la FFT

Exemple sur Microchip PICDEM2+ avec PIC18F452





Le PIC18 est un processeur RISC standard, possédant une multiplication non signé câblée, il est cependant adapté au traitement signal.
Le spectre d'un signal échantillonné est donnée par :

$$X(F) = \sum_{n=1}^{n=N} \sqrt{X(n) \otimes \sin(2\pi nF / N)^2 + X(n) \otimes \cos(2\pi nF / N)^2}$$

N : nombre d'échantillons
F : numéro de la raie
Fréquences des raies : Fr=Fx Fe/N
Exemple : pour une fréquence d'échantillonnage de 5Kz avec 32 échantillons :

Fr = F * (5,000 / 32) = F * 156.25 Hz

F1 = 156.25 Hz	F5 = 781.25Hz	F9 =1406.25 Hz	F13 = 2031.25 Hz
F2 = 312.5 Hz	F6 = 937.5 Hz	F10 = 1562.5 Hz	F14 = 2187.5 Hz
F3 = 468.75 Hz	F7 = 1093.75 Hz	F11 = 1718.75 Hz	F15 = 2343.75 Hz
F4 = 625 Hz	F8 = 1250 Hz	F12 = 1875 Hz	F16 = 2500 Hz

Mise en œuvre d'une DFT avec les fonctions de la bibliothèque dft.asm
Après saisie des 32 échantillons, on calcule la partie réelle de chaque raie (indice bincount) (rappel : seules les 16 premières raies ont un sens physique) on calcul donc pour bincount de 1 à 16.

$$Qbin[bincount] = \sum_{n=1}^{n=32} INBUFFER[n] \otimes table[\mathrm{mod}_{32}(8 + N \cdot bincount)]$$

On peut faire de même pour la partie imaginaire

$$Ibin[bincount] = \sum_{n=1}^{n=32} INBUFFER[n] \otimes table[\mathrm{mod}_{32}(N \cdot bincount)]$$

Pour les deux calculs on a (16) raies de fréquences * (32) échantillons = (512) 24-bit opérations MAC (multiply and addition)

INBUFFER[32] : buffer d'échantillons sur 8 bits
FTABLE[32] : Table de sinus signée, le cosinus provident d'une lecture décalée de cette table soit 90 degrés ou 8 échantillons.
IBIN[16] : résultats de la partie imaginaire sur 24-bit
QBIN[16] : résultats de la partie réelle sur 24-bit
magnitude[16] : résultat sur 32 bits de $IBIN^2 + QBIN^2$

Algorithme du programme FFTpd2.C (utilise la bibliothèque dft.asm)

- ☑ Saisie de 32 échantillons
- ☑ Appelle de la fonction dft() qui effectue un produit de convolution entre INBUFFER[32] et FTABLE[32] pour calculer IBIN[16] et QBIN[16]
- ☑ Calcul de magnitude[F] = (unsigned long)(IBIN[F]>>8)2 + (unsigned long)(QBIN[F]>>8)2 (mise à l'échelle sur 16 bits)
- ☑ Affichage

DFT.asm

```
;* Discrete Fourier Transform (dft.asm) MICROCHIP
;* - init_sine_table places 8-bit signed 32-sample sine wave into FTABLE[32]
;* - Time domain input samples stored in INBUFFER[32]
;* - dft correlates FTABLE against INBUFFER to create IBIN and QBIN results
;* - (16) 24-bit Imaginary and Real frequency bin results stored in IBIN and QBIN

        #include p18f452.inc

;FTABLE   size=32 bytes (32) 8-bit sine table
;IBIN             size=48 bytes (16) 24-bit Real frequency sum bins
;QBIN             size=48 bytes (16) 24-bit Imaginary frequency sum bins
;INBUFFER         size=32 bytes (32) 8-bit samples
;BINCOUNT         size=1  byte  Bin counter varible
;MSB_TEMP         size=1  byte  MSB of bin counter for 2's complement calculations

        extern FTABLE,IBIN,QBIN,INBUFFER,BINCOUNT,MSB_TEMP
        global dft,init_sine_table

        ERRORLEVEL -315         ; Turn off LFSR assembler warnings, LFSR works on PIC18FXXX devices
        ERRORLEVEL -314

        code
dft
        banksel BINCOUNT
        lfsr    0,FTABLE        ; FSR0 points to the start of sine frequency table
        lfsr    1,INBUFFER              ; FSR1 points to the start of the input sample buffer
        lfsr    2,IBIN                  ; FSR2 points to the Real frequency sum bin
        clrf    BINCOUNT                ; Point to first I-Bin table entry
        call    loop_bin        ; Perform I-bin Multiply and Accumulate operation
        lfsr    0,FTABLE+8              ; FSR0 points to the start of cosine frequency table
        lfsr    1,INBUFFER              ; FSR1 points to the start of the input sample buffer
        lfsr    2,QBIN                  ; FSR2 points to the imaginary frequency sum bin
        clrf    BINCOUNT                ; Point to first Q-bin table entry
        call    loop_bin        ; Perform Q-bin Multiply and Accumulate operations
        call    abs_bin                 ; Now convert magnatude from 2's complement to unsigned
        return

correlation macro
        local add_negative
        local add_positive
        clrf    MSB_TEMP                ; Clear MSB of MAC result
        movf    INDF0,W                 ; Move next input sample waveform value to W register
        mulwf   POSTINC1        ; Multiply input sample by correlated frequency wave value
        btfss   FSR0L,4                 ; Skip if correlated waveform is negative
        bra             add_positive    ; Correlated waveform is positive so don't take 2's complement
add_negative
        comf    PRODL,F                 ; Convert result to negative 2's complement number
        comf    PRODH,F                 ; Invert product result and temporary 24-bit MSB register
        comf    MSB_TEMP                ;
;       movlw   1                               ; Officially, we should add one after complementing
;       addwf   PRODL,F                 ; but this simply result in a offset of 16 that will show up in all bins
;       clrw
;       addwfc  PRODH,F
;       addwfc  MSB_TEMP,F
add_positive
        movf    PRODL,W                 ; Add product low to I or Q bin LSB, point to next byte in bin

        addwf   POSTINC2,F
        movf    PRODH,W                 ; Add product high to I or Q bin MSB, point to next byte in bin
        addwfc  POSTINC2,F
        movf    MSB_TEMP,W              ; Move MSB into W register to add carry from product high I or Q bin result to MSB byte
of bin
        addwfc  POSTDEC2,F
        movf    POSTDEC2,F              ; Decrement I or Q pointers to reset them to beginning of this bin
        incf    BINCOUNT,W              ; Skip to next sine / cosine input value
        addwf   FSR0L,W
        andlw   0x1F                    ; Module sine / cosine correlation table by 32
        movwf   FSR0L
        endm

loop_bin
        clrf    MSB_TEMP                ; Clear MSB of MAC result
        bcf             FSR1L,5                 ; Modulo 32 the input sample waveform pointer
        movf    INDF0,W                 ; Place sine table value into W register
        mulwf   POSTINC1        ; Multiply sample by sine table, place result into PROD register
        btfss   FSR0L,4                 ; Skip if correlated waveform is negative
        bra             add_positive    ; Correlated waveform is positive so don't take 2's complement
add_negative
        comf    PRODL,F                 ; Convert result to negative 2's complement number
        comf    PRODH,F                 ; First invert the product result
        comf    MSB_TEMP                ; Now add 1 to the inverted result...
;       movlw   1                               ; Officially, we should add one after complimenting to it...
;       addwf   PRODL,F                 ; but this simply result in a offset of 16 that will show up in all bins
;       clrw
;       addwfc  PRODH,F
```

```
;           addwfc    MSB_TEMP,F
add_positive
            movff     PRODL,POSTINC2          ; Initialize bin with the first corelation results
            movff     PRODH,POSTINC2
            movff     MSB_TEMP,POSTDEC2
            movf      POSTDEC2                ; Reset pointer to LSB of current bin
            incf      BINCOUNT,W              ; Point to next entry in frequency sine wave frequency
            addwf     FSR0L,W
            andlw     0x1F                    ; Modulo-32 in sine frequency table
            movwf     FSR0L
correlate
            local i
i = .31
            while (i > 0)
            correlation                       ; Perform correlation macro (24-bit) sum FTABLE * INBUFFER (31) more
times
i--
            endw
            incf      BINCOUNT,F              ; Point to next bin and sine wave frequency
            btfsc     BINCOUNT,4              ; Skip when if we have not finshed the last bin
            return                            ; done with all bins so return
            movlw     0x3                     ; Advance I or Q pointer to next bin by adding 3 locations
            addwf     FSR2L,F
            bra             loop_bin          ; Not done with this bin so loop back

abs_bin                                       ; Convert IBIN and QBIN results from 2's complement to absolute
magnatude
            lfsr      0,IBIN+0x5F             ; Point FSR0 to the MSB of the last value in IBIN
loop_abs
            movlw     IBIN-1                  ; Place first IBIN entry (minus 1) into WREG for comparison
            cpfseq    FSR0L                   ; Skip if we are finished with IBIN and QBIN
            bra             check_negative    ; Not done yet so check next value
            return                            ; Return once we are finished
check_negative
            btfss     INDF0,7                 ; Skip if this is a negative number
            bra       abs_positive            ; Not negative, so we don't need to complement
abs_negative
            comf      POSTDEC0,F              ; Negative number, so compliment it to make it positive
            comf      POSTDEC0,F
            comf      POSTDEC0,F
            bra       loop_abs
abs_positive
            movf      POSTDEC0,F              ; Positive number, so move FSR0 (3) positions to point it to the next 24-bit
entry
            movf      POSTDEC0,F
            movf      POSTDEC0,F
            bra             loop_abs

init_sine_table                               ; Initializes FTABLE[32] with 2's complements sine table
            lfsr      0,FTABLE                ; Point to the start of the sin table in RAM
            movlw     upper     sine_table    ; Init MSB of table pointer to beginning of sine table
            movwf     TBLPTRU
            movlw     HIGH      sine_table    ; Init middle byte of table pointer to beginning of sine table
            movwf     TBLPTRH
            movlw     low       sine_table    ; Init LSB of table pointer to beginning of sine table
            movwf     TBLPTRL
            movlw     0x20
            movwf     BINCOUNT                ; Initialize BINCOUNT to 32 entries
loop_sine_lookup
            tblrd*+                           ; Read byte from sine lookup table, place in TABLAT
            movff     TABLAT,POSTINC0         ; Place sine look-up byte into next entry of RAM based sine table
            decfsz    BINCOUNT,F              ; Decrement counter, skip when zero
            bra             loop_sine_lookup  ; Not done initializing RAM table, loop back
            return                            ; Done with last entry so return

sine_table                    ; 2's complement sine table
            data      0x03200 ; Sample Numbers  0    and  1
            data      0x08E62 ; Sample Numbers  2    and  3
            data      0x0D4B4 ; Sample Numbers  4    and  5
            data      0x0FAEC ; Sample Numbers  6    and  7
            data      0x0FAFF ; Sample Numbers  8    and  9
            data      0x0D4EC ; Sample Numbers  10   and  11
            data      0x08EB4 ; Sample Numbers  12   and  13
            data      0x03262 ; Sample Numbers  14   and  15
            data      0x03200 ; Sample Numbers  16   and  17
            data      0x08E62 ; Sample Numbers  18   and  19
            data      0x0D4B4 ; Sample Numbers  20   and  21
            data      0x0FAEC ; Sample Numbers  22   and  23
            data      0x0FAFF ; Sample Numbers  24   and  25
            data      0x0D4EC ; Sample Numbers  26   and  27
            data      0x08EB4 ; Sample Numbers  28   and  29
            data      0x03262 ; Sample Numbers  30   and  31

            end
```

FFTpd2.C

```
include <p18f452.h>                     // Defines all processor specific peripherals
#include <stdlib.h>                     // Defines standard function library
#include <lcdpd2.h>

// les deux fonctions suivantes sont dans dft.asm (à placer dans le projet)
extern void dft (void);                 // routines assembleur pour la DFT (utilise la multiplication HARD sur 8 bits)
extern void init_sine_table (void);     // recopie la table de sinus en ram dans la ram

#pragma udata analizer_vars=0x100       // Les vaiables C sont en Bank 1 (début en 0x100)
unsigned char start_dft;
unsigned long magnitude[16];            // (16) 32-bit I carré + Q carré (amplitudes de fréquences)

#pragma udata dft_vars=0x300            // Les variables assembleur pour la DFT sont en bank 3 en 0x300
unsigned char FTABLE[32];       // Table de sinus sur 8bits pour le calcul de convolution
unsigned short long IBIN[16];           // (16) 24-bit nombres imaginaires pour les fréquences
unsigned short long QBIN[16];           // (16) 24-bit nombres réels pour les fréquences
unsigned char INBUFFER[32];     // (32) tableau d'échantillons sur 8 bits
unsigned char  BINCOUNT;        // un compteur
unsigned char MSB_TEMP;         // MSB 24-bit produit temporaire

#pragma interrupt echantillonnage  save=PROD
void echantillonnage (void)             // IT haute priorité provenant de TIMER tous les 2,000 cycles
{static char buffer_index=0;            // soit 2000*0.25uS = 500mS sur PICDEM2+
        if (PIR1bits.TMR2IF){
                ADCON0bits.GO_DONE = 1;         // SOC
                PIR1bits.TMR2IF = 0;            // efface drapeau IT TMR2
                while(ADCON0bits.GO_DONE);      // Attend EOC
                INBUFFER[buffer_index++] = ADRESH;      // mémorise l'échantillon sur 8 bits
                if(buffer_index == 32){ buffer_index = 0;start_dft = 1;}      //              autorise le calcul, de la DFT
                        }
}

#pragma code it_priorite_haute=0x8
void it_priorite_haute (void)
{ _asm GOTO echantillonnage _endasm }
#pragma code

void main (void)
{unsigned char tampon[17];              // pour putsLCD
unsigned char result_index;
unsigned long max_result,valeur;        // pour la calcul du facteur d'échelle
unsigned int echelle;                   // facteur d'échelle pour LCD
        OpenLcd();
        SetDisplay(DisplayOn, CursorOn, BlinkingOn);
        ADCON0 = 0b00100001;                            // ADC on, canal 4 sur RA4, CLK conversion = Fosc/2
        ADCON1 = 0b01000010;                            // justification à gauche , CLK/2 clock, AN0~AN4 entrées analogiques
                                                        // (pb avec LCD qui utilise RA1 et RA2 (AN1 et AN2) !
        T2CON = 0b00001101;                             // acrive TMR2 prédiviseur = 4:1, postdiviseur = 2:1 (8 en tout)
        PR2 = 249;                                      // periode = 250 cycles
                                                        // Soit sur PICDEM2+ une IT toutes les 8*250*0.25uS soit Fe= 2000Hz
        PIE1bits.TMR2IE = 1;                            // Autorise IT sur TIMER2
        IPR1 = 0b00000010;                              // Haute priorité pour Timer 2
        IPR2 = 0;                                       // Basse priorité pour le reste
        start_dft = 0;                                  // pas encore saisit les 32 échantillons
        RCONbits.IPEN = 1;                              // Active le mode IT hautes et basses (PIC18)
        init_sine_table();                              // Initialise FTABLE[32] (recopie le tableau sinus de ROM en RAM)
        INTCON = 0b11000000;                            // Active les IT hautes et bassesd
while(1){       if(start_dft)                           // attend que le buffer en entrée contienne 32 échantillons
                {       start_dft = 0;
                        INTCONbits.GIEH = 0;            // désactive les IT hautes
                        dft();                                                  // calcul de la DFT
                        result_index = 0;
                        max_result = 0;                 //on ne calcul l'amplitudes que des transmormées réelles (16 premières)
                        for (result_index=0;result_index<16;result_index++)
                        {       magnitude[result_index] = (unsigned long)(IBIN[result_index]>>8) * (unsigned
long)(IBIN[result_index]>>8) +  (unsigned long)(QBIN[result_index]>>8) * (unsigned long)(QBIN[result_index]>>8);
                        if (magnitude[result_index] > max_result) max_result = magnitude[result_index];
                        }
                        echelle=max_result/6; // pour les 7 possibilités de caractères semi-graphiques sur LCD de PICDEM2+
                        for (result_index=0;result_index<16;result_index++)             // affichage
                        {       valeur = magnitude[result_index]/echelle;
                                if (valeur<0) valeur=0;
                                if (valeur>6) valeur=6;
                                tampon[result_index]=128+valeur;// la barre en bas est le caractère 128
                        }
                        tampon[16]=0;           // en C une chaine doit finir par 0
                        // Sur PICDEM2+ l'afficheur LCD utilise RA1,2,3 en mode numérique
                        ADCON1=0b01000110;              // tous les bits du portA repassent en numérique
                        SetPosition(0,0);
                        putsLCD(tampon);
                        ADCON1=0b01000010;              // réactive les entrées analogiques
                        INTCONbits.GIEH = 1;            // réactive les IT hautes priorités
                }
        }
}
```

2.4. Détection de fréquence : exemple sur PIC18

Sur PICDEM2+ le câblage sera le même que pour le TP sur DFT.

```
#include <p18f452.h>
#include <stdlib.h>
#include <lcdpd2.h>

#define nbcoef 32
const float ak=1.4142; //si FE=2000Hz, k=4, N=32 F(k)=250Hz        //idem si k=8 et N=64 et  si k=16 et N=128
float r;        // le résultat sera rangé ici

#pragma udata echanti=0x100
unsigned char echantillon[nbcoef];
#pragma udata divers=0x300
unsigned char start_calcul;

#pragma interrupt echantillonnage  save=PROD  // IT haute priorité provenant de TIMER tous les 2,000 cycles
void echantillonnage (void)                   // la fonction echantillonne nbcoef et autorise la calcul
{static unsigned char cptech=0;               // soit 2000*0.25uS = 500uS sur PICDEM2+
if (PIR1bits.TMR2IF)
        {       ADCON0bits.GO_DONE = 1;                 // SOC
                PIR1bits.TMR2IF = 0;            // efface drapeau IT TMR2
                while(ADCON0bits.GO_DONE);      // Attend EOC
                echantillon[cptech]=ADRESH;
                if(++cptech == nbcoef)          {start_calcul = 1;      //      autorise le calcul
                                                 cptech=0;
                                                 }
                }
}

#pragma code it_priorite_haute=0x8
void it_priorite_haute (void)
{_asm GOTO echantillonnage _endasm}
#pragma code

float goertzel(void)      // calcul de l'amplitude de la raie k par la méthode de Goertzel
{ float qn=0.0,qn1=0.0,qn2=0.0;  // représente les q en z, z-1 zt z-2
float ech;
unsigned char cpt;
// recherche qN-1 et qN-2, la boucle s'effectue nbech fois
for (cpt=0;cpt<nbcoef;cpt++)
{       ech=(float)(echantillon[cpt]);      // le calcul s'effectue sur des réels
        if (cpt==0) {
                                            qn=ech;qn1=0,qn2=0;}
                                            else
                                            {qn2=qn1;
                                            qn1=qn;
                                            qn=ech + ak*qn1 - qn2;
                                            }
                }
return (qn*qn+qn1*qn1-ak*qn*qn1);           //Calcul de F(k)2
}

void main (void)
{unsigned tampon[17];                               // pour putsLCD
        OpenLcd();
        SetDisplay(DisplayOn, CursorOn, BlinkingOn);
        ADCON0 = 0b00100001;                                    // ADC on, canal 4 sur RA4, CLK conversion = Fosc/2
        ADCON1 = 0b01000010;                                    // justification à gauche , CLK/2 clock, AN0~AN4 entrées analogiques
                                                                // (pb avec LCD qui utilise RA1 et RA2 (AN1 et AN2) !
        T2CON = 0b00001101;                                     // acrive TMR2 prédiviseur = 4:1, postdiviseur = 2:1 (8 en tout)
        PR2 = 249;                                              // periode = 250 cycles
                                                                // Soit sur PICDEM2+ une IT toutes les 8*250*0.25uS soit Fe= 2000Hz
        PIE1bits.TMR2IE = 1;                                    // Autorise IT sur TIMER2
        IPR1 = 0b00000010;                                      // Haute priorité pour Timer 2
        IPR2 = 0;                                               // Basse priorité pour le reste
        start_calcul = 0;                                       // pas encore saisit les échantillons
        RCONbits.IPEN = 1;                                      // Active le mode IT hautes et basses (PIC18)
        INTCON = 0b11000000;                                    // Active les IT hautes et basses
while(1){
        if(start_calcul)                                        // attend que le buffer en entrée contienne N échantillons
                {       start_calcul = 0;
                        INTCONbits.GIEH = 0;                    // désactive les IT hautes
                        r=goertzel();                           // Module au carrée de la raie
                        // Sur PICDEM2+ l'afficheur LCD utilise RA1,2,3 en mode numérique
                        ADCON1=0b01000110;                      // tous les bits du portA repassent en numérique
                        SetPosition(0,0);
                        putsLCD(ftoa(r,tampon,4,'S'));
                        putsrLCD("     ");
                        ADCON1=0b01000010;                      // réactive les entrées analogiques
                        INTCONbits.GIEH = 1;                    // réactive les IT hautes priorités
                }
        }
}
```