Travaux Pratiques

Traitement Numériques du Signal

Sur TMS320C6713

TP1 :

Acquisition, Génération de signaux et Aliasing

Projet1 sur CCS 6.2: (sera nommé **TP1_Acquisition**)

En utilisant le logiciel CoolPro, envoyer via la carte son un signal sinusoïdal de fréquence 10 KHz (la fréquence d'échantillonnage la plus grande possible, par exemple 96 KHz). La carte son va sortir le signal en mode analogique.

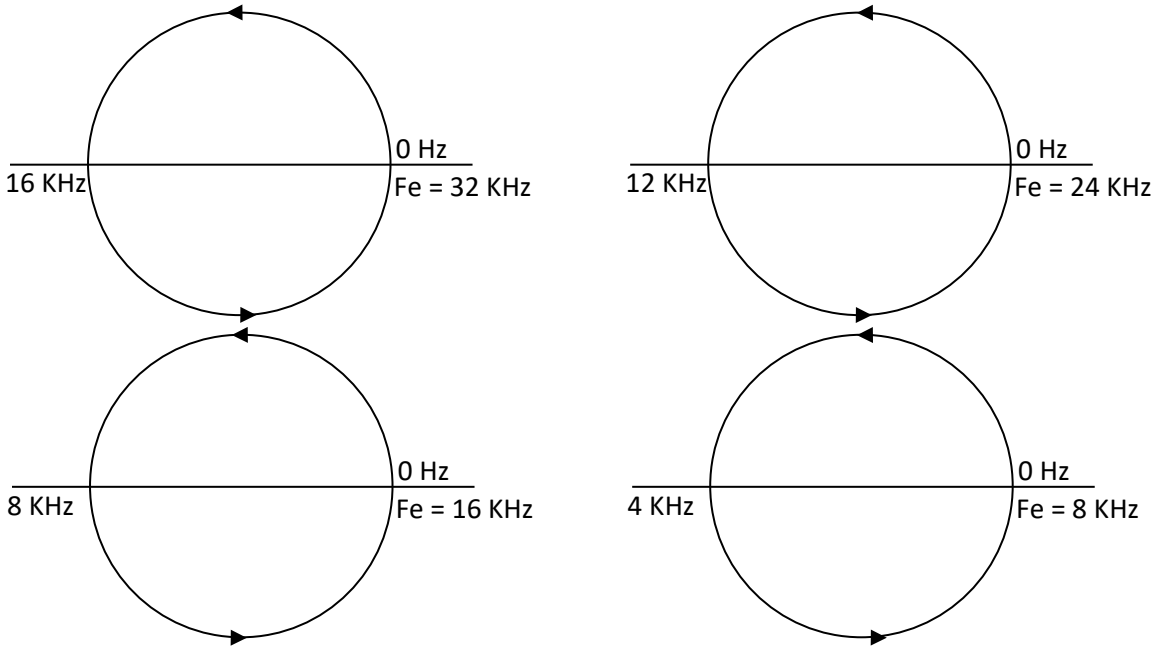
Au niveau du Code Composer Studio, mettez au point un projet qui fait l'acquisition d'un signal analogique appliqué à l'entrée LINE INPUT du kit DSK6713. L'acquisition va être stockée dans un buffer circulaire **XnBuffer** de taille **128**. Le signal acquit est envoyé à la sortie LINE OUTPUT.

On veut vérifier l'effet de l'Aliasing (détorsion du signal) en variant la fréquence d'échantillonnage Fe pendant l'acquisition du signal sur LINE INPUT. Pour effectuer l'analyse, on aura besoin d'afficher le buffer en mode temporel dans un 1^{èr} graphe et en mode fréquentiel dans un 2ème graphe.

Notez la fréquence du signal résultante suivant les fréquences d'échantillonnages suivantes :

Fe	32 KHz	24 KHz	16 KHz	8 KHz
Fréquence F (Hz)				

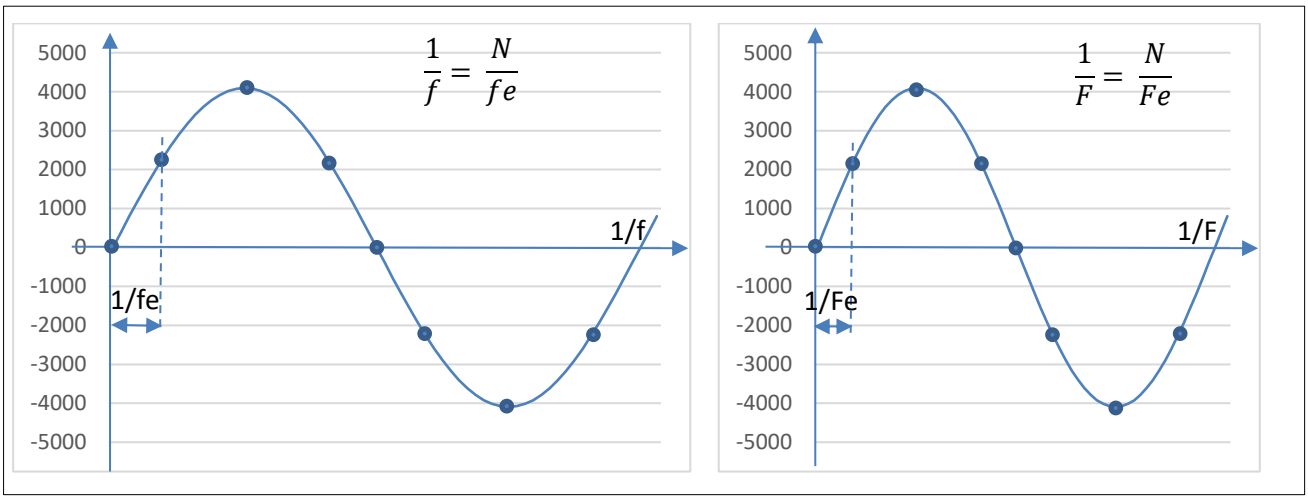
Faite une conclusion en utilisant le cercle fréquentiel suivant:



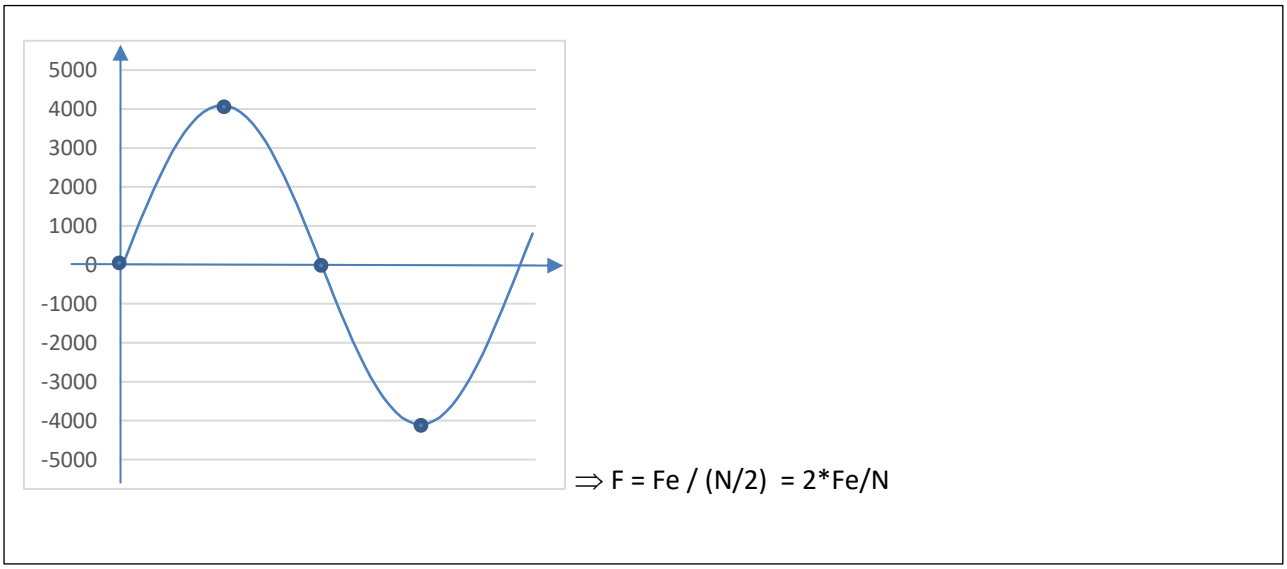
Projet2 sur CCS 6.2: (sera nommé **TP1_GenSignaux**)

Le but est de générer des fonctions sinusoïdales à partir d'une seule table de N échantillons de la fonction **sin(2π.i.f/fe) avec i : 0, 1, 2, …, N-1**. Cette fonction a comme fréquence **f** et fréquence d'échantillonnage **fe**. Si la fréquence d'échantillonnage de cette fonction change et devient **Fe**, et que ces mêmes échantillons sont obtenus pendant les temps 1/Fe, 2/Fe, …, N-1/Fe, on aura :

i	0	1	2	…	N-2	N-1
Sin(2π.i.f/fe)	0	Sin(2π.f/fe)	Sin(2π.2.f/fe)	…	Sin(2π.(N-2).f/fe)	Sin(2π.(N-1).f/fe)
Temps	0	1/Fe	2/Fe	…	N-2/Fe	N-1/Fe
Sin(2π.i.F/Fe)	0	Sin(2π.F/Fe)	Sin(2π.2.F/Fe)	…	Sin(2π.(N-2).F/Fe)	Sin(2π.(N-1).F/Fe)



$$\Rightarrow N = \frac{fe}{f} = \frac{Fe}{F}$$



Exemple 1 :

f = 1 Hz, N = 64, Fe = 8000 Hz ⇒ fe = N = 64 Hz & F = Fe/N = 8000/64 = 125 Hz

Exemple 2 :

f = 1 Hz, N/2 échantillons, Fe = 8000 Hz ⇒ fe = N/2 = 32 Hz & F = Fe/(N/2) = 8000/32 = 250 Hz

La fréquence du signal sinusoïdale échantillonné à Fe est obtenue par la formule suivante :

$$\mathbf{Fe/F = fe/f \Rightarrow F = f * Fe/fe}$$

Cela signifie que si on dispose d'une fonction sinusoïdale de fréquence f et de fréquence d'échantillonnage fe, et qu'on la ré-échantillonne à la fréquence Fe, on obtient alors une autre fonction sinusoïdale de fréquence F = f*Fe/fe.

Si fe/f est égale à un entier N, on aura Xn(i) = sin(2π.i.f/fe) = sin(2π.i/N). Cela signifie que pendant une période du signal Xn, on aura besoin que de N échantillons de la fonction sin(2π.i/N) de fréquence 1 Hz. Dans ce cas, on a :

$$\mathbf{F/Fe = f/fe = 1/N \rightarrow F = Fe/N}$$

Travail à faire :

- Générer sur EXCELL une table sin(2π.i.f/fe) avec f = 1 Hz et fe = N = 64 (i : 0, 1, …, N-1) en format virgule fixe Q12. Cette table sera utilisée dans votre projet CCS sous le nom **SinTable**.
- Développer un projet CCS qui envoie les échantillons de cette table sur la sortie LINE OUT en utilisant la fonction BSL **output_sample()** suivant la fréquence d'échantillonnage **Fe = 8000 Hz** et un gain contrôlé par la variable nommé **GAIN** initialisé à 1 :

  $$\mathbf{XnBuffer(i) = GAIN * SinTable(i)}$$

- Avec un oscilloscope ou en utilisant un buffer de sortie (graph temporel et fréquentiel), vérifier que la fréquence du signal est 8000/64 = 125 Hz.
- Modifier la routine d'interruption **c_int11()** afin de sous échantillonner la table **SinTable** en prenant dedans des échantillons en sautant par pas contrôlé par la variable nommé **STEP** initialisé à 1. Ces échantillons vont être stockés dans l'ordre dans le buffer **XnBuffer** :

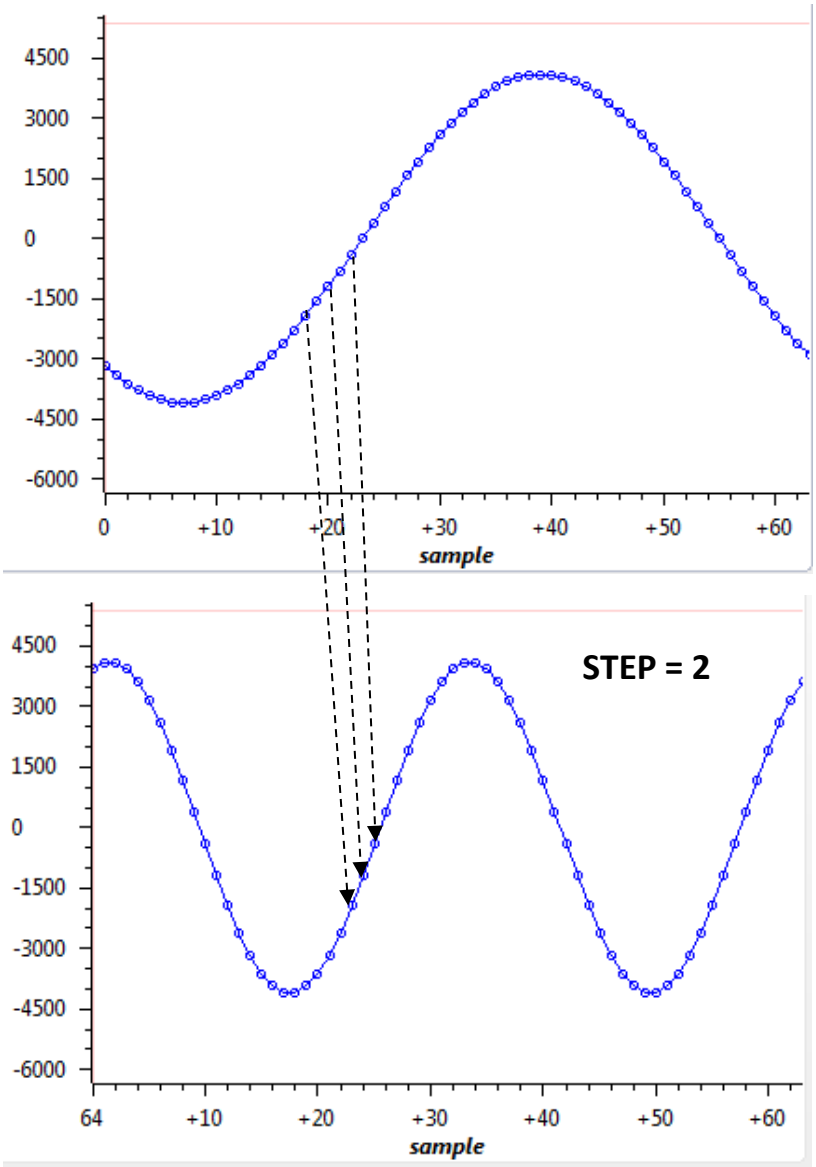
  $$\mathbf{XnBuffer(i) = GAIN * SinTable(j)}$$

  $$\mathbf{i = (i + 1) \text{ modulo } N \text{ et } j = (i * STEP) \text{ modulo } N}$$

  Dans ce cas, la formule précédente devient :

  $$\mathbf{F/Fe = 1/(N/STEP) = STEP/N \rightarrow F = Fe*STEP/N \quad ou \quad N = STEP*Fe/F}$$

SinTable



XnBuffer

- Le programme GEL suivant permet de faire varier la variable **GAIN** de 1 à 20 par pas de 1 :

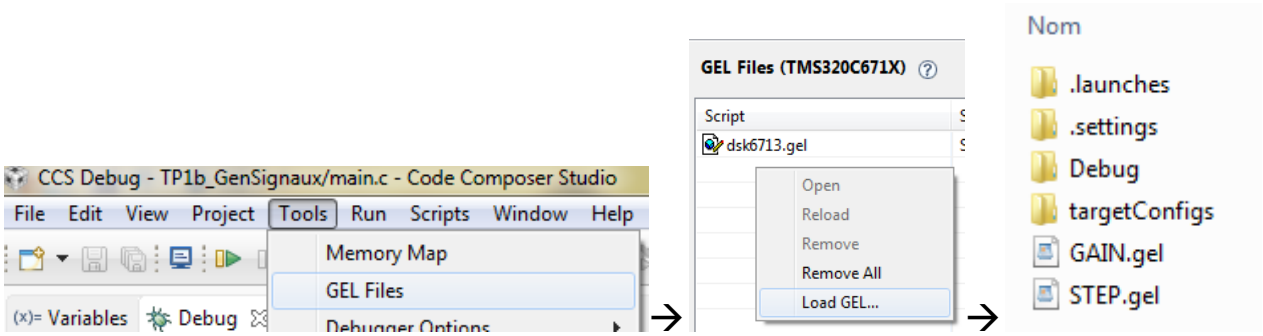
```
menuitem "Gain"
slider GAIN(1, 20, 1, 1, nom_param)
{
   GAIN = nom_param;
}
```

- Le programme GEL suivant permet de faire varier la variable **STEP** de 1 à 40 par pas de 1 :

```
menuitem "Step"
slider STEP(1, 40, 1, 1, nom_param)
{
   STEP = nom_param;
}
```

L'activation des GEL se fera sous le menu Debug comme suit :

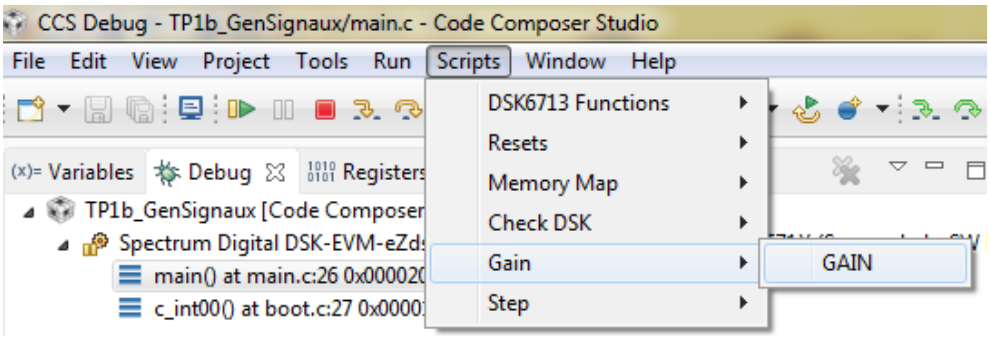
Tools → GEL Files → Dans la fenêtre GEL FILES, avec le bouton droit de la souris → Load GEL → Chercher dans le répertoire de votre projet et sélectionner les fichiers GEL : GAIN.gel et STEP.gel



Pour faire apparaitre les fenêtres des GEL, procédez comme suit :

Scripts → Gain → GAIN

Scripts → Step → STEP



En fixant STEP à 1 et en variant le GEL GAIN, vérifiez avec l'oscilloscope ou avec les graphes temporel et fréquentiel, l'amplitude du signal XnBuffer en temps réel. Faites une conclusion.

En fixant GAIN à 1 et en variant le GEL STEP, vérifiez avec l'oscilloscope ou avec les graphes temporel et fréquentiel, la fréquence F du signal XnBuffer en temps réel. Vérifiez qu'on a :

$$F = f * Fe/(N/STEP) = STEP*(f*Fe/N)$$

Quelle est la fréquence du signal en sortie LINE OUTPUT pour la valeur maximale de STEP ? Cette fréquence est-elle un Aliasing et pourquoi ?

A quelle valeur de STEP commence l'effet d'Aliasing ?

Exercice 1 : *On veut générer avec le kit DSK6713 des signaux sinusoïdaux de fréquence variable de 100 à 10 KHz par pas de 100 Hz. Déterminer les valeurs de N et Fe ?*

Exercice 2: *Les fréquences audibles par un humain sont dans l'intervalle 20 Hz à 20 KHz. On veut diagnostiquer les fréquences non audibles pour un patient malade. Pour cela, nous allons lui faire entendre toutes les fréquences de 20 Hz à 20 KHz par pas de 20 Hz.*

Quelle est la fréquence d'échantillonnage Fe ?

Combien le nombre d'échantillons N dans une période de la sinusoïde de fréquence f à 1 Hz ?

Sachant que la fréquence d'échantillonnage Fe est la plus grande possible et que le pas sera de 50 Hz. Combien le nombre d'échantillons N dans une période de la sinusoïde de fréquence f à 1 Hz ?

Sachant que la fréquence Fe est 48 KHz et le pas fréquentiel est 100 Hz. Combien le nombre d'échantillons N dans une période de la sinusoïde de fréquence f à 1 Hz ?

Mettre au point un projet complet qui balaye l'intervalle fréquentiel 100 Hz – 20 KHz sachant que le signal reste pendant 1 seconde à chaque fréquence.

TP2 : Filtrage à Réponse Impulsionnelle Finie (RIF)

L'objectif de ce TP est l'implémentation d'un filtre RIF (Passe Bas, Passe Haut, Passe Band, Band Passe) sur le DSP après sa conception sur MATLAB. L'implémentation sera en mettant au point un seul programme C ou en combinant deux programmes en C et en assembleur.

1- Conception d'un filtre sur MATLAB :

Dans la fenêtre « Command Window » de MATLAB, entrer la commande en ligne : « >> filterDesigner ». Faite la conception d'un filtre RIF Passe Bas sur MALAB suivant les caractéristiques suivantes :

- N = 128 (sous le Designer de MATLAB, on entrera N-1 = 127)
- Fréquence de coupure Fc = 1000 Hz.
- Fréquence d'échantillonnage Fs = 8000 Hz
- Window Kaiser avec Beta = 10

2- Filtrage virgule fixe avec un programme C suivant l'algorithme 1:

Ecrire le programme C qui implémente ce filtre RIF en virgule fixe suivant l'algorithme 1 qui utilise les 2 boucles suivantes:

- boucle1 : N multiplications/accumulations
- boucle2 : N décalages

		Cycle		
Coefficients	**XnBuffer**	**n**	**n+1**	**n+2**
h(0)	XnBuffer[0]	X(n) → XnBuffer[0]	X(n+1) → XnBuffer[0]	X(n+2) → XnBuffer[0]
h(1)	XnBuffer[1]	X(n-1) → XnBuffer[1]	X(n) → XnBuffer[1]	X(n+1) → XnBuffer[1]
h(2)	XnBuffer[2]	X(n-2) → XnBuffer[2]	X(n-1) → XnBuffer[2]	X(n) → XnBuffer[2]
…		…	…	…
…		…	…	…
h(N-2)	XnBuffer[N-2]	X(n-(N-2)) → XnBuffer[N-2]	X(n-(N-3)) → XnBuffer[N-2]	X(n-(N-4)) → XnBuffer[N-2]
h(N-1)	XnBuffer[N-1]	X(n-(N-1)) → XnBuffer[N-1]	X(n-(N-2)) → XnBuffer[N-1]	X(n-(N-3)) → XnBuffer[N-1]
Filtrage FIR		Y(n) = h(0)*X(n) + h(1)*X(n-1) + h(2)*X(n-2) +…+h(N-1)*X(n-(N-1))	Y(n+1) = h(0)*X(n+1) + h(1)*X(n) + h(2)*X(n-1) +…+h(N-1)*X(n-(N-2))	Y(n+2) = h(0)*X(n+2) + h(1)*X(n+1) + h(2)*X(n) +…+h(N-1)*X(n-(N-1))

Déterminer le nombre de cycle entre les fonctions input_sample() et output_sample() dans le sous-programme d'interruption c_int11().

3- Filtrage virgule fixe avec un programme C suivant l'algorithme 2:

Ecrire le programme C qui implémente ce filtre RIF en virgule fixe suivant l'algorithme 2 qui utilise une seule boucle suivante:

- **N multiplications/accumulations et décalages**

		Cycle		
Coefficients	**XnBuffer**	**n**	**n+1**	**n+2**
h(0)	XnBuffer[0]	X(n) → XnBuffer[0]	X(n+1) → XnBuffer[0]	X(n+2) → XnBuffer[0]
h(1)	XnBuffer[1]	X(n-1) → XnBuffer[1]	X(n) → XnBuffer[1]	X(n+1) → XnBuffer[1]
h(2)	XnBuffer[2]	X(n-2) → XnBuffer[2]	X(n-1) → XnBuffer[2]	X(n) → XnBuffer[2]
…		…	…	…
…		…	…	…
h(N-2)	XnBuffer[N-2]	X(n-(N-2)) → XnBuffer[N-2]	X(n-(N-3)) → XnBuffer[N-2]	X(n-(N-4)) → XnBuffer[N-2]
h(N-1)	XnBuffer[N-1]	X(n-(N-1)) → XnBuffer[N-1]	X(n-(N-2)) → XnBuffer[N-1]	X(n-(N-3)) → XnBuffer[N-1]
	Poubelle : XnBuffer[N]	***X(n-(N)) → XnBuffer[N]***	***X(n-(N-1)) → XnBuffer[N]***	***X(n-(N-2)) → XnBuffer[N]***
Filtrage FIR		**Y(n) = h(0)*X(n) + h(1)*X(n-1) + h(2)*X(n-2) +…+h(N-1)*X(n-(N-1))**	**Y(n+1) = h(0)*X(n+1) + h(1)*X(n) + h(2)*X(n-1) +…+h(N-1)*X(n-(N-2))**	**Y(n+2) = h(0)*X(n+2) + h(1)*X(n+1) + h(2)*X(n) +…+h(N-1)*X(n-(N-1))**

Déterminer le nombre de cycle entre les fonctions input_sample() et output_sample() dans le sous-programme d'interruption c_int11().

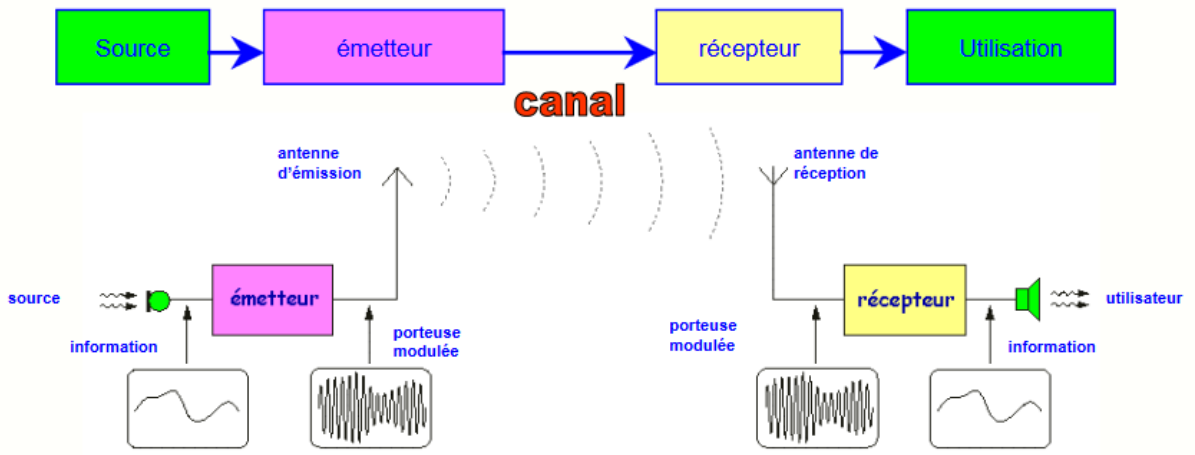
4- Filtrage virgule flottante avec un programme C:

Ecrire le programme C qui implémente ce filtre RIF en virgule flottante suivant l'algorithme 2 précédent.

Déterminer le nombre de cycle entre les fonctions input_sample() et output_sample() dans le sous-programme d'interruption c_int11().

TP3 : Modulation et Démodulation d'Amplitude

La modulation d'amplitude consiste à faire varier l'amplitude d'un signal de fréquence élevée (signal *porteur)* en fonction d'un signal de plus basse fréquence (*signal qui sera modulé)*. Ce dernier est celui qui contient l'information à transmettre (signal audio, vidéo, analogique ou numérique). Cette technique de transmission consiste à moduler une information pour la transmettre via une antenne émettrice et, de l'autre côté, en utilisant une antenne réceptrice, on démodule le signal reçu pour déduire l'information envoyée :



En résumé pour faire la modulation d'amplitude, il nous faut une porteuse de fréquence élevée p(t) et un signal de fréquence basse s(t). L'expression de la modulation en amplitude est comme suit :

$$S(t) = p(t)\left(1 + \frac{s(t)}{K}\right)$$

Pour ce TP, nous allons prendre pour les deux signaux p(t) et s(t) des signaux sinusoïdaux et cette expression en mode discret devient en supposant que $K = 2^k.D$:

$$S(n) = A_p.sin(2\pi.n\frac{F}{Fe})\left(1 + \frac{A_s}{D}sin\left(2\pi.n\frac{f}{Fe}\right) \gg k\right)$$

Avec $p(n) = A_p.sin\left(2\pi.n\frac{F}{Fe}\right)$, $s(n) = A_s.sin\left(2\pi.n\frac{f}{Fe}\right)$ et $K = 2^k.D$

Dans notre cas, on prendra les paramètres suivants :

Paramètres fixes : $Fe = 96\ KHz \quad et \quad A_p = A_s = 2^{14} = 16384$

Paramètres variables : $F \in [18.5, 19, 19.5, 20, 20.5, 21, 21.5, 22\ KHz] \quad 1\ KHz \leq f \leq 3\ KHz$

$1 \leq D \leq 10 \qquad 8 \leq k \leq 16$

On remarque que les 8 fréquences de F sont espacées d'un pas fréquentiel de 500 Hz. Si on prend en compte la formule vue en TP1 : N = STEP * Fe/F, pour STEP = 1, F = 500 Hz. Donc, N = 1*96000/500 = 192. Pour F = 18500 Hz, **STEP = N*F/Fe** = 192*18500/96000 = 37. Le tableau suivant donne la valeur STEP pour chaque fréquence :

F (Hz)	18500	19000	19500	20000	20500	21000	21500	22000
STEP	37	38	39	40	41	42	43	44

On a vu aussi en TP1 que $N = \frac{fe}{f} = \frac{Fe}{F}$, si f = 1 Hz alors fe = N = 192 Hz. On va donc générer une table contenant N échantillons d'une seule période de la sinusoïde de fréquence 1 Hz comme suit :

$$\boldsymbol{TabSinus1Hz}(n) = A_p.\sin\left(2\pi.n\frac{f}{fe}\right) = A_p.\sin\left(2\pi\frac{n}{192}\right) \quad avec \quad 0 \le n \le 191 \quad \text{et} \quad A_p = 16384$$

Afin de générer, à partir de cette table, n'importe quelle porteuse de fréquence F $\in [18.5, 19, 19.5, 20, 20.5, 21, 21.5, 22\ KHz]$, il suffit d'échantillonner cette table par Fe/STEP, avec Fe = 96 KHz.

Donc, l'expression discrète d'une porteuse à la fréquence F est la suivante :

$$p(n,F) = \boldsymbol{TabSinus1Hz}\big((n.STEP)\ modulo\ 192\big) \quad avec \quad 0 \le n \le 191 \ \ et \ \ STEP = \frac{192.F}{Fe}$$

```
short TabSinus1Hz[192] =
{16384,16375,16349,16305,16244,16165,16069,15956,15826,15679,15515,15334,15137,14924,14694,14449,
14189,13913,13623,13318,12998,12665,12318,11958,11585,11200,10803,10394,9974,9543,9102,8652,
8192,7723,7246,6762,6270,5771,5266,4756,4240,3720,3196,2669,2139,1606,1072,536,
0,-536,-1072,-1606,-2139,-2669,-3196,-3720,-4240,-4756,-5266,-5771,-6270,-6762,-7246,-7723,
-8192,-8652,-9102,-9543,-9974,-10394,-10803,-11200,-11585,-11958,-12318,-12665,-12998,-13318,-13623,-13913,
-14189,-14449,-14694,-14924,-15137,-15334,-15515,-15679,-15826,-15956,-16069,-16165,-16244,-16305,-16349,-16375,
-16384,-16375,-16349,-16305,-16244,-16165,-16069,-15956,-15826,-15679,-15515,-15334,-15137,-14924,-14694,-14449,
-14189,-13913,-13623,-13318,-12998,-12665,-12318,-11958,-11585,-11200,-10803,-10394,-9974,-9543,-9102,-8652,
-8192,-7723,-7246,-6762,-6270,-5771,-5266,-4756,-4240,-3720,-3196,-2669,-2139,-1606,-1072,-536,
0,536,1072,1606,2139,2669,3196,3720,4240,4756,5266,5771,6270,6762,7246,7723,
8192,8652,9102,9543,9974,10394,10803,11200,11585,11958,12318,12665,12998,13318,13623,13913,
14189,14449,14694,14924,15137,15334,15515,15679,15826,15956,16069,16165,16244,16305,16349,16375};
```

Travail à faire :

1- En partant d'un projet d'acquisition (Copier, coller et renommer votre projet), modifier le programme main pour générer une porteuse sous forme d'une table nommé ***SignalPorteuse*** :

$$SignalPorteuse(n) = TabSinus1Hz(\ (n.STEP)\ modulo\ 192\ )\ avec\ 0 \le n \le 191\ \ et\ \ STEP = \frac{192.F}{Fe}$$

Dans votre programme, la variable STEP est de type Uint16. Mettez au point un programme GEL pour faire varier la variable STEP de 37 à 44 par pas de 1. En utilisant la fonction output_sample(), sortez le signal de la porteuse vers l'oscilloscope Analog Discovery. Activer au niveau de l'oscilloscope le mode temporel et fréquentiel (FFT). Varier STEP avec son GEL et vérifier que la fréquence de la porteuse avec l'oscilloscope.

2- Modifier ce projet pour mettre au point la modulation. Le signal qui sera modulé, généré par le SYNTRILLIUM COOL EDIT, est envoyé vers l'entrée LINE IN du CODEC du kit DSK6713. Sa fréquence variera entre 1 et 3 KHz. Le signal de la modulation sera stocké dans le tableau ***SignalModulation*** :

$$SignalModulation(n) = SignalPorteuse(n) + \frac{SignalPorteuse(n).Xn}{D} \gg k$$

$$avec\ \ Xn\ l'échantillon\ du\ signal\ modulant, échantillonné\ à\ la\ fréquence\ Fe = 96\ KHz$$

Sortez le signal de la modulation ***SignalModulation*** vers l'oscilloscope par la fonction output_sample(). Mettez au point deux programmes GEL :

- un pour D variant de 1 à 10 par pas de 1
- un pour k variant de 8 à 16 par pas de 1.

En fixant la fréquence F de la porteuse à 22000 Hz, observer l'effet de D et k sur la modulation en variant leurs GEL. Fixez les deux paramètres D et k pour avoir un bon rapport de modulation et varier la fréquence de la porteuse F avec son GEL. Observer la FFT au niveau de l'oscilloscope et faites une conclusion.

3- La démodulation permet de retrouver le signal qui a été envoyé par la modulation. Elle consiste tout simplement à multiplier le signal de la modulation par la porteuse :

$$SignalDemodulation(n) = SignalModulation(n).SignalPorteuse(n)$$

Compléter votre programme en ajoutant la démodulation. Sortez le signal de la démodulation vers l'oscilloscope et observer la FFT. Faites une conclusion.

4- En s'inspirant de filtrage FIR passe bas du TP précédent, mettez au point un filtre passe bas de fréquence de coupure Fc = 3.2 KHz et au nombre de coefficient 192.

5- Compléter votre programme pour y rajouter le filtrage passe bas du signal démoduler. Faites sortir le signal filtré vers l'oscilloscope et faites une conclusion.

Filtre Passe Bas de fréquence de coupure Fc = 3.2 KHz :

```
#define  N   192   // Nombre de coefficients du filtre

#define  Qk   18   // representation en virgule fixe en Qk

// Frequence d'echantillonnage Fs = 96000

short h[192]= {2,2,2,2,2,1,-1,-3,-6,-10,-14,-19,-23,-27,-31,-33,
-33,-31,-26,-18,-7,7,25,44,65,87,108,127,143,153,156,150,
135,110,74,27,-29,-93,-163,-235,-305,-371,-426,-468,-490,-490,-464,-411,
-328,-216,-78,83,262,451,642,825,990,1126,1222,1269,1258,1182,1037,822,
539,193,-205,-644,-1106,-1571,-2018,-2423,-2760,-3005,-3134,-3124,-2958,-2620,-2101,-1397,
-510,552,1772,3130,4598,6145,7732,9322,10872,12342,13690,14878,15874,16647,17175,17444,
17444,17175,16647,15874,14878,13690,12342,10872,9322,7732,6145,4598,3130,1772,552,-510,
-1397,-2101,-2620,-2958,-3124,-3134,-3005,-2760,-2423,-2018,-1571,-1106,-644,-205,193,539,
822,1037,1182,1258,1269,1222,1126,990,825,642,451,262,83,-78,-216,-328,
-411,-464,-490,-490,-468,-426,-371,-305,-235,-163,-93,-29,27,74,110,135,
150,156,153,143,127,108,87,65,44,25,7,-7,-18,-26,-31,-33,
-33,-31,-27,-23,-19,-14,-10,-6,-3,-1,1,2,2,2,2,2 };
```

TP4 : Conception d'un émetteur acoustique en utilisant l'algorithme de Goertzel

Objectif : Mis au point d'un émetteur acoustique qui détermine d'abord les canaux fréquentiels occupés par les autres émetteurs et, ensuite, émet un signal acoustique à un des canaux libres trouvés. La détection d'un canal se fera en utilisant l'algorithme de Goertzel. Les huit canaux utilisés par cet émetteur sont les suivants :

Fréquence (Hz)	**18500**	**19000**	**19500**	**20000**	**20500**	**21000**	**21500**	**22000**
Nom Canal : Fx	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Indice Canal : f	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Cet émetteur doit d'abord balayer le milieu marin pour déterminer les canaux déjà occupés par les autres émetteurs dans une portée de 500 m (Emission Cool Edit Pro entre 0 et -40 DB) et, ensuite, émet chaque seconde un pulse acoustique de largeur 10 ms et de fréquence un canal libre. Cet émetteur dont le schéma électronique est dans l'annexe, a été conçu dans les années 2000 en utilisant 8 filtres analogiques et un microcontrôleur. Les inconvénients de cet émetteur sont le coût, la consommation, l'encombrement, … Nous souhaitons le moderniser en implémentant des solutions basées sur le traitement numérique du signal en utilisant le DSP. Plusieurs techniques peuvent être utilisées pour détecter les canaux occupés : Transformé de Fourier Rapide, Filtrage Passe Bande, Corrélation, Algorithme de Goertzel, …. Dans ce TP, nous allons utiliser l'algorithme de Goertzel détaillé dans l'annexe de ce TP.

Lorsque la détection ne porte que sur quelques fréquences, l'algorithme de Goertzel est plus efficace que toutes les autres techniques. Au contraire de la FFT où on a besoin de deux tables cosinus et sinus, l'algorithme de Goertzel ne nécessite qu'un seul coefficient par fréquence. Cet algorithme permet d'effectuer le calcul de façon récursive et n'a besoin que d'une constante et de trois mémoires par fréquence : Qn, Qn_1 et Qn_2 selon l'expression suivante :

Qn = Xn + Coeff(F) * Qn_1 – Qn_2.

La fréquence d'échantillonnage **Fe** qu'on va utiliser est **96 KHz**. Puisque les huit canaux utilisés sont par **pas fréquentiel** de **500 Hz** (**bin_width = 500**), le nombre d'échantillons **N** nécessaire pour l'algorithme de Goertzel est **N = Fe/bin_width = 96000/500 = 192**.

Pour chaque canal F qu'on souhaite détecter, on utilise une constante qu'on va nommer **Coeff(F)** calculé comme suivant :

$$\boldsymbol{Coeff(F) = 2.Cos(k\frac{2\pi}{N})}$$

$$\textbf{avec } \boldsymbol{k = ENTIER\left(0.5 + N\frac{F}{Fe}\right)} \quad \boldsymbol{N = 192} \quad \boldsymbol{Fe = 96000\ Hz}$$

La constante **Coeff(F)** en virgule fixe sera en format $\mathbf{Q_{15}(16)}$:

Coeff_Q15(F) = arrondi(2^15 * Coeff(F))

Le tableau suivant donne les valeurs des **Coeff_Q15(F)** pour chaque canal :

F (Hz)	18500	19000	19500	20000	20500	21000	21500	22000
k	37	38	39	40	41	42	43	44
Coeff_Q15	23085	21066	19024	16962	14882	12785	10676	8554

Sachant que Xn est un échantillon du signal à traiter, le pseudo code de l'algorithme de Goertzel pour les huit canaux est le suivant :

Boucle : pour f de 0 à 7

Qn_1(f) = Qn_2(f) = 0

Fin Boucle

Boucle : pour i de 0 à N-1

Boucle : pour f de 0 à 7

Qn(f) = Xn + Coeff(f) * Qn_1(f) – Qn_2(f)

Qn_2(f) = Qn_1(f)

Qn_1(f) = Qn(f)

Fin Boucle

Fin Boucle

Boucle : pour f de 0 à 7

Module(f) = Qn_1(f) * Qn_1(f) + Qn_2(f) * Qn_2(f) – Qn_1(f) * Qn_2(f) * Coeff(f)

Fin Boucle

Avant l'utilisation du code composer studio, faite l'implémentation de l'algorithme de Goertzel sur **EXCELL**. Remarquez qu'on a un très bon **contraste** à la fin des 192 échantillons du signal acoustique. Le module maximum trouvé va probablement correspondre au canal détecté. Pour être sûr, il faut que le module maximum soit suffisamment éloigné des autres modules. Pour cela nous allons déterminer le module maximum nommé « **ModuleMax** » et le module qui vient juste après le module maximum nommé « **ModuleMaxSuivant** ». Afin d'avoir un très bon **contraste de détection** du canal, on va appliquer cette condition :

ModuleMax > ModuleMaxSuivant * CONTRASTE avec CONTRASTE ≥ 8

Pendant la phase de recherche des canaux occupés de 5 secondes, l'algorithme de Goertzel sera exécuté $\frac{5}{192/96000} = \frac{5.96000}{192}$ = 2500 fois.

Après avoir trouvé les canaux occupés, on émettra un pulse acoustique à un des canaux libres trouvés. Ce pulse sinusoïdal de largeur 10 ms sera émis toutes les secondes.

Pour le traitement en virgule fixe, on va adopter les formats Qk et les cadrages suivants :

Signal	**Xn**	**Coeff(f)**	**Qn(f)**	**Qn_1(f)**	**Qn_2(f)**	**Module(f)**
Format Qk	Q15(16)	Q15(16)	Q8(16)	Q8(16)	Q8(16)	Q16(32)
Type	short	short	short	short	short	Uint32

Implémenter dans le sous-programme d'interruption le pseudocode suivant :

```
Si Période de recherche 5 secondes
    Acquisition d'un nouvel échantillon Xn
    Xn → XnBuffer
    Boucle : Pour f de 0 à 7
                Qn(f) = Xn + Coeff(f) * Qn_1(f) – Qn_2(f)  // Calcul en virgule fixe
                Qn_2(f) = Qn_1(f)
                Qn_1(f) = Qn(f)
    End Boucle
    Incrémentation circulaire de l'indice n
    Décrémentation du compteur 5 secondes
    Si acquisition des N échantillons
        Boucle : Pour f de 0 à 7
            Module(f) = Qn_1(f) * Qn_1(f) + Qn_2(f) * Qn_2(f) – Qn_1(f) * Qn_2(f) * Coeff(f)
            Si Module(f) > ModuleMax
                ModuleMax = Module(f)
                fmax = f
            End Si
        End Boucle
        Boucle : Pour f de 0 à 7
            Si Module(f) != ModuleMax ET Module(f) > ModuleMaxSuivant alors ModuleMaxSuivant = Module(f)
            End Si
            Qn_1(f) = 0
            Qn_2(f) = 0
        End Boucle
        Si ModuleMax > ModuleMaxSuivant*CONTRASTE  alors  Tab_Detection(fmax) = 1
        End Si
    End Si acquisition des N échantillons
    Si Fin Période 5 secondes
        Boucle : Pour f de 0 à 7
            Si Tab_Detection(f) = 0
                CanalLibre = f
                STEP = f + 37
                f = 8       // Pour sortir de la boucle dès le 1èr canal libre détecté
                n = 0       // n est l'indice d'un échantillon à la fréquence Fe = 96 KHz
                j = 0       // j est l'indice dans TabSinus1Hz utilisant STEP
            End Si
        End Boucle
    End Si Fin Période 5 secondes
    ModuleMax = 0
    ModuleMaxSuivant = 0
Si Non Période de recherche 5 secondes
    Si le compteur de temps est inférieur à 10 ms
        Acquisition de Xn à partir TabSinus1Hz indexé par j
        Incrémentation circulaire de l'indice n
        Incrémentation circulaire de l'indice j en tenant compte de STEP du canal libre
        Sortie de Xn via le CODEC
    Si Non
            On sort un signal nul via le CODEC
    End SI
    Incrémentation circulaire du compteur de temps
End Si
```

Dans un premier temps, nous allons injecter un signal généré par EXCEL nommé XnBufferExcel afin de vérifier les calculs entre le DSP et EXCEL.

```
#include "dsk6713_aic23.h"


Uint32 input_left_sample();
void output_left_sample(int out_data);
void comm_intr();


#define      N            192
#define      CONTRASTE    8
#define      TIME_5sec    2500      // 5*Fe/N = 2500
#define      TIME_10ms    960       // 0.010 * Fe = 960
#define      TIME_1sec    96000     // 1 * Fe = 96000


short XnBufferExcel[N] =                       // Table Sinus de fréquence 22 KHz
{0,32488,8481,-30274,-16384,25997,23170,-19948,-28378,12540,31651,-4277,-32768,-4277,31651,12540,
-28378,-19948,23170,25997,-16384,-30274,8481,32488,0,-32488,-8481,30274,16384,-25997,-23170,19948,
28378,-12540,-31651,4277,32767,4277,-31651,-12540,28378,19948,-23170,-25997,16384,30274,-8481,-32488,
0,32488,8481,-30274,-16384,25997,23170,-19948,-28378,12540,31651,-4277,-32768,-4277,31651,12540,
-28378,-19948,23170,25997,-16384,-30274,8481,32488,0,-32488,-8481,30274,16384,-25997,-23170,19948,
28378,-12540,-31651,4277,32767,4277,-31651,-12540,28378,19948,-23170,-25997,16384,30274,-8481,-32488,
0,32488,8481,-30274,-16384,25997,23170,-19948,-28378,12540,31651,-4277,-32768,-4277,31651,12540,
-28378,-19948,23170,25997,-16384,-30274,8481,32488,0,-32488,-8481,30274,16384,-25997,-23170,19948,
28378,-12540,-31651,4277,32767,4277,-31651,-12540,28378,19948,-23170,-25997,16384,30274,-8481,-32488,
0,32488,8481,-30274,-16384,25997,23170,-19948,-28378,12540,31651,-4277,-32768,-4277,31651,12540,
-28378,-19948,23170,25997,-16384,-30274,8481,32488,0,-32488,-8481,30274,16384,-25997,-23170,19948,
28378,-12540,-31651,4277,32767,4277,-31651,-12540,28378,19948,-23170,-25997,16384,30274,-8481,-32488 };

short Tab_ALPHA[8] = {23085, 21066, 19024, 16962, 14882, 12785, 10676, 8554};


short TabSinus1Hz[N] =
{0,1072,2143,3212,4277,5338,6393,7441,8481,9512,10533,11543,12540,13524,14493,15447,
16384,17304,18205,19087,19948,20788,21605,22400,23170,23916,24636,25330,25997,26635,27246,27827,
28378,28899,29389,29847,30274,30668,31029,31357,31651,31912,32138,32330,32488,32610,32698,32750,
32767,32750,32698,32610,32488,32330,32138,31912,31651,31357,31029,30668,30274,29847,29389,28899,
28378,27827,27246,26635,25997,25330,24636,23916,23170,22400,21605,20788,19948,19087,18205,17304,
16384,15447,14493,13524,12540,11543,10533,9512,8481,7441,6393,5338,4277,3212,2143,1072,
0,-1072,-2143,-3212,-4277,-5338,-6393,-7441,-8481,-9512,-10533,-11543,-12540,-13524,-14493,-15447,
-16384,-17304,-18205,-19087,-19948,-20788,-21605,-22400,-23170,-23916,-24636,-25330,-25997,-26635,-27246,-27827,
-28378,-28899,-29389,-29847,-30274,-30668,-31029,-31357,-31651,-31912,-32138,-32330,-32488,-32610,-32698,-32750,
-32768,-32750,-32698,-32610,-32488,-32330,-32138,-31912,-31651,-31357,-31029,-30668,-30274,-29847,-29389,-28899,
-28378,-27827,-27246,-26635,-25997,-25330,-24636,-23916,-23170,-22400,-21605,-20788,-19948,-19087,-18205,-17304,
-16384,-15447,-14493,-13524,-12540,-11543,-10533,-9512,-8481,-7441,-6393,-5338,-4277,-3212,-2143,-1072 };


Uint32 fs = DSK6713_AIC23_FREQ_96KHZ;
```

TP5 : Conception d'un émetteur acoustique par traitement non récursif (Filtres FIR)

Objectif : Mise au point d'un projet industriel complet contenant le traitement non récursif (8 Filtres FIR) en virgule fixe.

On veut concevoir un émetteur acoustique sous marin qui émet une onde acoustique à une des huit fréquences suivantes :

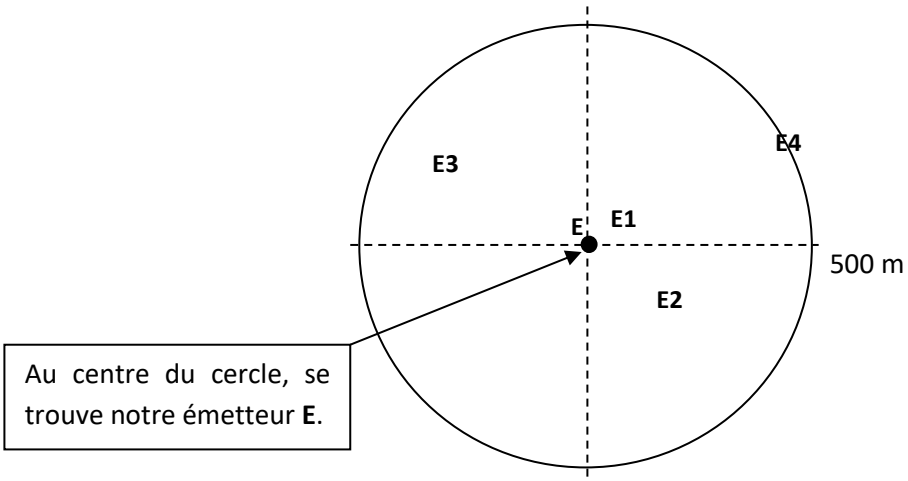
Fréquence (KHz)	18.5	19	19.5	20	20.5	21	21.5	22
Nom Canal : Fx	F0	F1	F2	F3	F4	F5	F6	F7
Indice Canal : f	0	1	2	3	4	5	6	7

Cet émetteur doit d'abord balayer le milieu marin pour déterminer les canaux déjà occupés par d'autres émetteurs éventuels dans une portée de 500 m et ensuite émet chaque seconde un pulse acoustique à la fréquence d'un canal libre.

Nous allons utiliser ici huit filtres en cascade. Chaque filtre est un FIR passe bas autour du canal à détecter. La conception des huit filtres ainsi que leur mise à niveau en virgule fixe se feront sur MATLAB. Dans le programme C que vous allez développer sur le Code Composer Studio, vous créez un tableau à deux dimensions contenant les coefficients des huit canaux : ***short TabFiltre[8][N]***, N spécifie le nombre de coefficients de chaque filtre FIR. Nous aurons besoin d'un tableau ***h[N]*** qui va contenir, à chaque fois qu'on recherche la présence d'un canal, les coefficients du filtre de ce canal.

Après avoir déterminé les canaux libres et occupés, vous allez sortir un pulse acoustique d'un canal libre. Ce pulse de 2 ms de largeur sortira toutes les secondes.

Le seuil de détection a une importance capitale. Si on suppose que le niveau du signal Xn à un mètre de l'émetteur est A, le niveau sera A/10 à 10 mètres, A/100 à 100 mètres et A/1000 à 1000 mètres. On a donc les atténuations suivantes : -20 DB à 10 m, -40 DB à 100 m et -60 DB à 1000 m. Par exemple, si notre zone de travail est un cercle de rayon inférieur à 500 m (atténuation max = -54 DB), il faut qu'on règle le seul de détection suffisamment petit pour pouvoir détecter les émetteurs distants de moins de 500 m. On peut avoir par exemple le cas suivant :



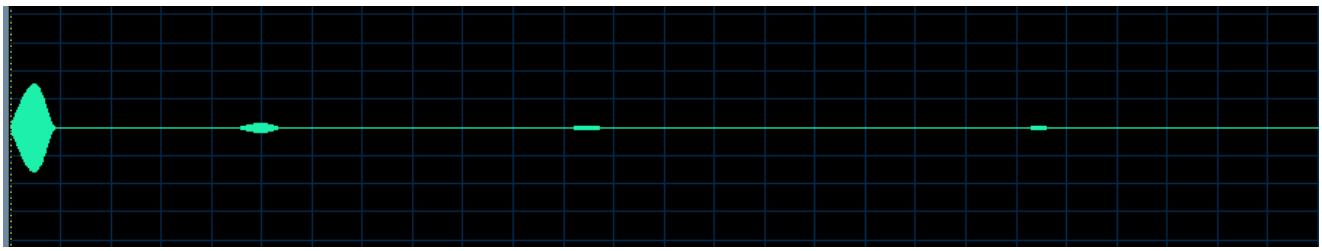
Emetteur	E1	E2	E3	E4
Canal (Hz)	18500	20000	21000	22000
Distance (m)	10	100	316	500
Atténuation	-20 DB	-40 DB	-50 DB	-54 DB

(Atténuation = -20 log(Amplitude/Amplitude à 1 m) = -20 log(Distance)

Si on fixe le seuil de détection a un niveau très faible, en présence du bruit, on risque de détecter tous les canaux même ceux qui sont réellement libre. Donc, à chaque fois qu'on fait le tour des huit canaux et qu'on ne trouve aucun canal libre, on double le seuil de détection.

Pour le test en mode débogage, mettez au point avec « Cool Edit Pro », un ou plusieurs pulses sinusoïdales de fréquences différentes. Par exemple :

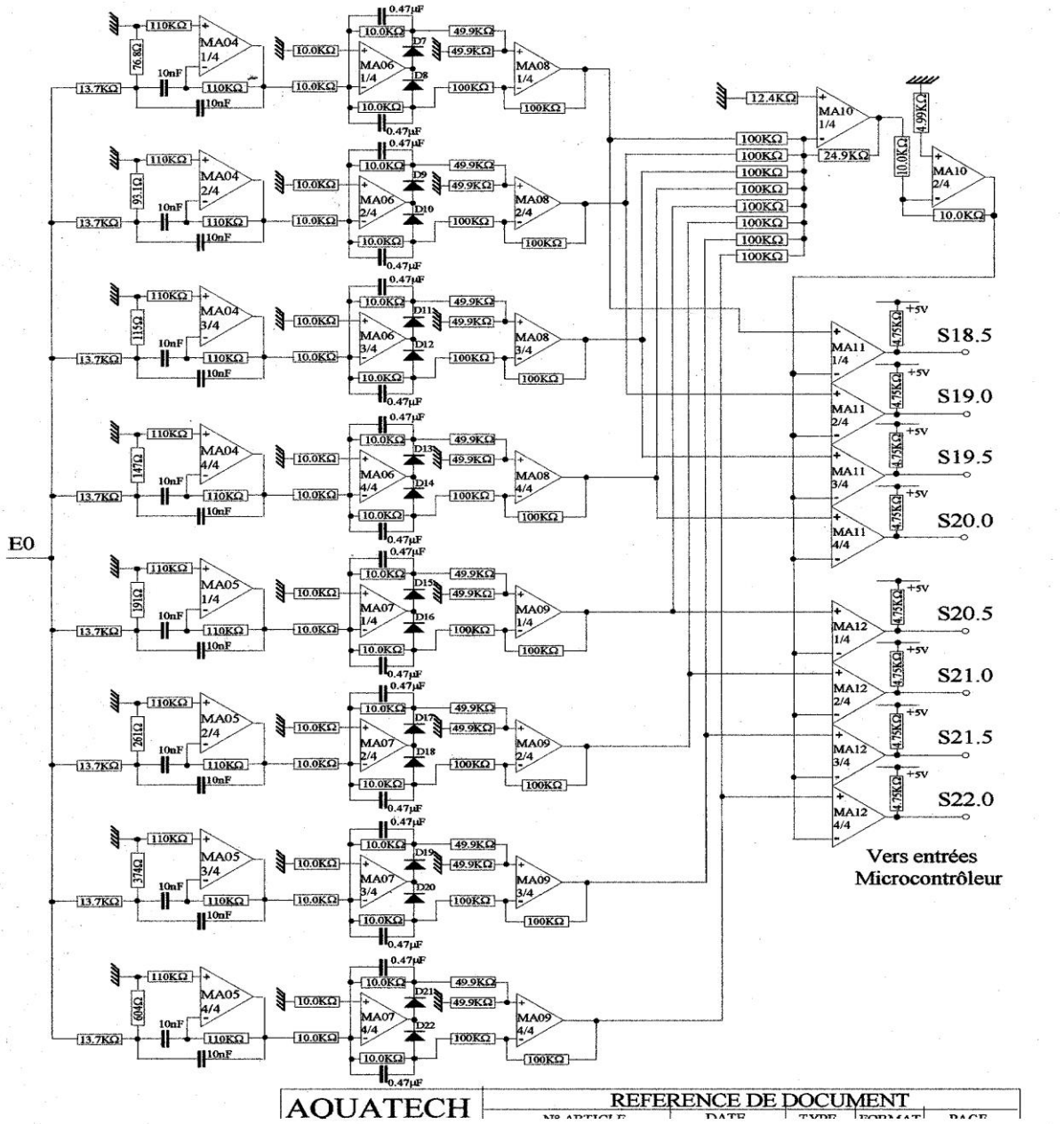
- Fréquence = 18500 Hz, Atténuation = -20 DB et Périodicité = 0.2 seconde
- Fréquence = 20000 Hz, Atténuation = -40 DB et Périodicité = 0.2 seconde
- Fréquence = 21000 Hz, Atténuation = -50 DB et Périodicité = 0.2 seconde
- Fréquence = 22000 Hz, Atténuation = -54 DB et Périodicité = 0.2 seconde



Mettez au point, sur le Code Composer Studio (CCS), un projet qui implémente les 8 filtres FIR en cascade selon les spécifications données par le pseudo code dans la page suivante.

Temps = 48000 (dans une seconde, on a 48000 acquisitions séries)
Canal_Libre = 0 (témoin indiquant qu'un canal libre est trouvé)
Boucle : Pour f de 0 à 7
TabTrace(f) = 0
End Boucle
Seuil = SEUIL/2
Boucle : Attente détection d'un canal libre
*Seuil = 2 * Seuil*
Boucle : pour f de 0 à 7
Boucle : pour i de 0 à N-1
Initialisation des buffers XnBuffer et YnBuffer
Table des coefficients du filtre d'un canal → table h
Fin Boucle
Temps = 0
IntgC = 0
Boucle : Attente Traitement pendant 1 seconde ou détection du canal (Temps < 48000)
Acquisition du signal à Fe = 48000 Hz
Incrémentation du compteur d'acquisition (Temps++)
Filtrage FIR par rapport aux coefficients de la table h
Intégrateur court de la sortie Yn du filtre (RC = 128) :
IntgC = IntgC + (|Yn| - IntgC)/RC;
Si IntgC > Seuil (Si oui, Canal Détecté)
TabTrace(f) = 1
Temps = 48000 (finir le traitement du canal f et passer au suivant)
End Si
End Boucle
End Boucle
Boucle : Pour f de 0 à 7
Si TabTrace(f) = 0 alors Canal libre détecté
Mémorisation du canal : Fx = f
End Boucle
End Boucle
Boucle infinie : Sortir un pulse acoustique de 2 ms toutes les secondes
Boucle : Pour i de 0 à 95
Sortir Echantillon du Pulse TablePulse(Fx)(i) vers le CODEC
End Boucle
Boucle : Pour i de 0 à (48000 – 95)
Sortir Echantillon = 0 vers le CODEC
End Boucle
End Boucle infinie
End

Annexe :
Emetteur Acoustique à base de 8 filtres analogiques et un microcontrôleur

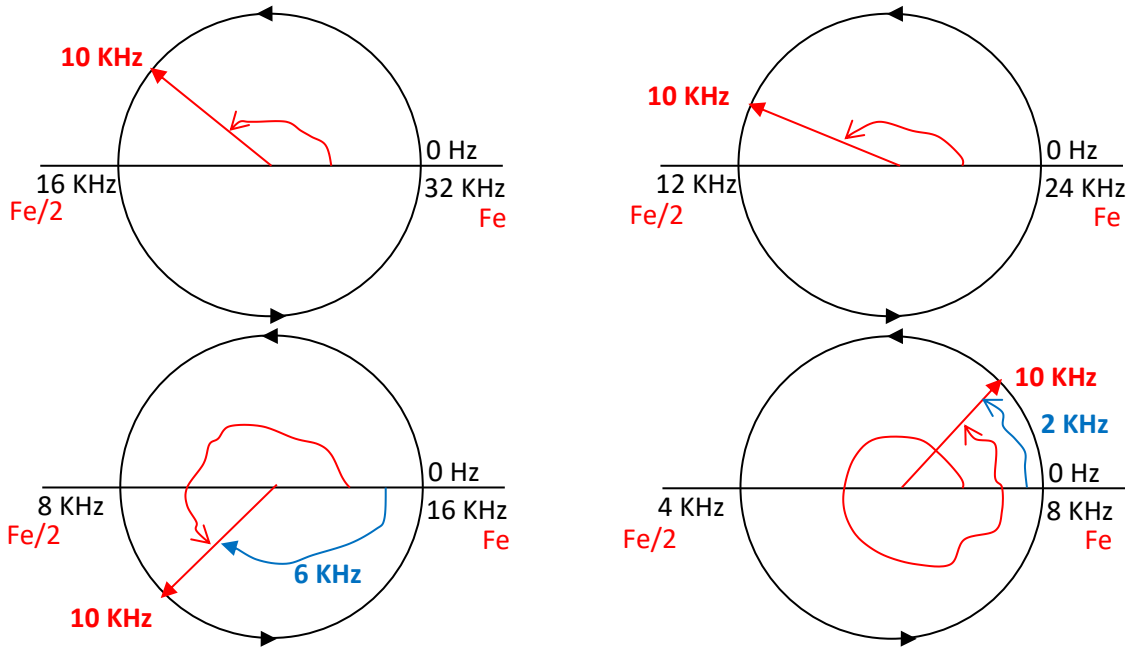


(inclure Algorithme de Goertzel (Goertzel-EETimes.pdf)

Notez la fréquence du signal résultante suivant les fréquences d'échantillonnages suivantes :

Fe	32 KHz	24 KHz	16 KHz	8 KHz
Fréquence F (Hz)	**10 KHz**	**10 KHz**	**6 KHz**	**2 KHz**

Faite une conclusion en utilisant le cercle fréquentiel suivant:



31 * 125 = 3875 Hz

32 * 125 = 4000 Hz

33 * 125 = 4125 Hz