TSI – Traitement du Son par DSP

PARTIE A - Prise en Main de la carte DSP

PARTIE B - Synthèse Sonore

PARTIE C - Filtrage

PARTIE A	Prise en Main de la carte DSP	Durée : 2UC
	OBJECTIFS	• Prise en main de la carte et des différents branchements • Génération du code automatique avec l'outil Real-Time Workshop de Simulink • Protocole de vérification des programmes avec les outils de mesure

TUTORIEL 1	Prise en Main de Matlab-Simulink	45 min

\A_PRISE_EN_MAIN\TUTORIEL_1

REMARQUE : Bien que s'agissant de rappels, il est recommandé de tester les fonctions suivantes.

Tout travail convenable avec Matlab se fait dans un fichier .m (et non directement dans la console)
Sélectionner les lignes à tester puis appuyer sur F9 pour les exécuter dans la console.

1. Prise en Main de MATLAB : Fonctions de Base

TUTO_MATLAB.m

```
%  Tout travail avec MATLAB s'effectue dans un fichier .m
% Sélectionner les lignes à exécuter et taper F9 (evaluate selection)

var1=3.14
myString='hello world'
pi
a = 10
c = 10*-2*a
nom = 'Roger';

% Modifier le format d'affichage des nombres
format long       % Notation 15 Chiffres
format short
%-------------------------------------------------------------------------------
%                     REPERTOIRE DE TRAVAIL
%-------------------------------------------------------------------------------
% Pour connaitre le répertoire de travail :
pwd
%Pour changer de répertoire, utiliser la commande cd:

%-------------------------------------------------------------------------------
%                       OPERATIONS SCALAIRES DE BASE
%-------------------------------------------------------------------------------
%Opérations de base (+,?,*,/)
7/45
(1+i)*(2+i)
1 / 0
0 / 0

%Exposants (^)
```

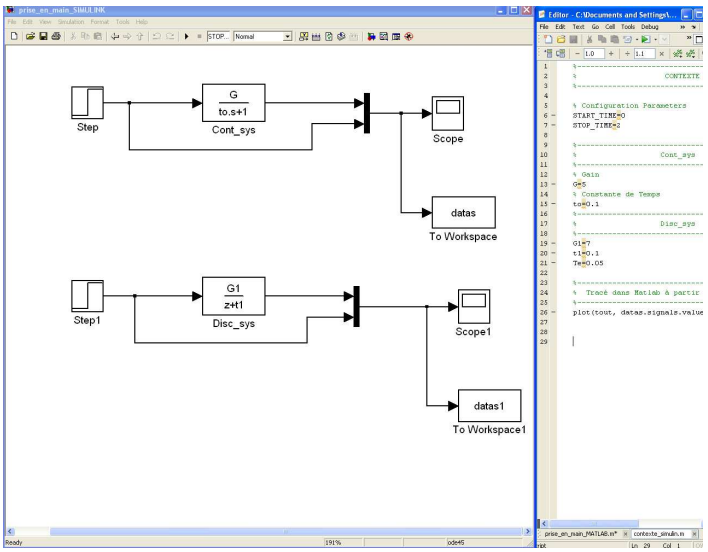
```
4^2
(3+4*j)^2

%Priorité des opérations + parenthèses
((2+3)*3)^0.1

%Pas de multiplication implicite!!
%3(1+0.7) % il y aura un message d'erreur!!
%Commande pour réinitialiser la fenêtre de commande
clc

%Fonctions Préprogrammées
sqrt(2)
log(2)          % Logarithme Népérien ln
log10(0.23)     % Logarithme Décimal
cos(1.2)
atan(-0.8)
exp(2+4*i)
round(1.6)      % Arrondi
floor(3.7)      % Partie Entière
ceil(4.23)      % Arrondi Supérieur

angle(i)
abs(1+i)        % Valeur Absolue ou Module d'un Complexe

%Informations sur l'utilisation d'une commande:
help sin

%Pour une documentation plus complète:
doc sin
%-------------------------------------------------------------------------------
%                               MATRICES
%-------------------------------------------------------------------------------

% Vecteurs
row=[1 2 5.4 -6.6]
row = [1, 2, 5.4, -6.6]
column = [4;2;7;4]

% Matrice
A= [1 2;3 4]

% Transposition
B=transpose(a)
C=A.'

%Sans le point: transposition Hermitienne,
%i.e. transposition + conjugué
%complexe
H = [1+j 2+3*j]
H'
H.'

% opérations élément par élément
A=[1 2 3];B=[4;2;1];
%A.*B        %  erreur!!
%A./B        %  erreur!!
%A.^B        %  erreur!!
A.*B'       %  Ok!!
A./B'       %  Ok!!
A.^(B')     %  Ok!!
```

```
%-------------------------------------------------------------------------------
%                              POLYNOMES
%-------------------------------------------------------------------------------

% P(x) = 3x^2-5x+2, il sera represente par le vecteur P:
P = [3 -5 2]

% Pour Evaluer le polynome, on utilise la fonction polyval.
% Pour calculer P(5) :
polyval(P,5)

% Calcul des Racines d'un Polynome
racines = roots(P)

% La fonction poly permet de créer un polynome
% à partir de ses racines:
P3 = poly([1 -1])

% Tracé Polynome
P=[1, 0, -1]
x=-3:0.1:3
TRACE=polyval(P,x)
plot(x, TRACE)

%-------------------------------------------------------------------------------
%                     FONCTIONS D'INITIALISATION
%-------------------------------------------------------------------------------
o=ones(1,10)      % vecteur ligne avec 10 éléments tous égaux à 1
o=ones(2,10)      % Matrice 2 colonnes *10 lignes avec éléments %tous égaux à 1
z=zeros(23,1)     % vecteur colonne avec 23 éléments
% tous égaux à 0
r=rand(1,45)      % vecteur ligne g avec 10
% éléments aléatoires compris entre 0 et 1
%Pour initialiser une séquence linéaire: linspace
a=linspace(0,10,5) % début à 0, fin à 10 (inclus), 5 valeurs

%On peut aussi utiliser la technique suivante :
b=0:2:10; %?début à 0, par incrément de 2, fin à 10
c=1:5 %?dans ce cas, L'incrément par défaut est 1

%-------------------------------------------------------------------------------
%                        GRAPHIQUES 2D
%-------------------------------------------------------------------------------

x=linspace(0,4*pi,1000);
y=sin(x);

%Valeur de y en fonction des indexes
plot(y);

%Valeur de y en fonction de x
plot(x,y);

%On peut changer la couleur, le type de point,le type de trait
plot(x, y,'b o - '); % x, y, couleur, point, type de trait
plot(x, y, 'o')

%Titres
title('sin(Theta)'); xlabel('Theta'); ylabel('sin(Theta)');

%Pour tracer deux courbes sur le même graphique
y=sin(x);
```

```
plot(x,y);
hold on;
y=cos(x);
plot(x,y);

%Pour tracer un nouveau graphique
%(dans l'exemple, 10 est le no. de la figure)
figure(10);
plot(x, y, 'b o - ')
hold on;
plot(x,-y, 'r o - ')
%ou
figure(11);
plot(x, y, 'b o - ',x, -y, 'r o - ');

%La commande subplot permet
%d'afficher plusieurs graphes dans une même fenêtre.
x=0:0.001:10;
subplot(1,2,1);
plot(x,sin(x));
subplot(1,2,2);
plot(x,cos(x));


%Plusieurs types de graphiques 2D
x=linspace(0,4*pi,1000);
y=sin(x);
plot(x, y);                    % graphique 2D en coord. (x, y)
stem(x, y);
loglog(x, y);            % graphique log(y) vs log(x)
semilogx(x, y);          % graphique y vs log(x)
semilogy(x, y);          % graphique log(y) vs x
bar(x, y);                     % graphique à barre

% Tracer la fonction
% f(x) = e^(-|x|/(2pi))*cos(x) sur l'intervale x = [-10pi 10pi].
% Utiliser une ligne rouge pleine et un nombre de points adéquat.
x=-10*pi:.01:10*pi;
plot(x, exp(-abs(x)/(2*pi)).*cos(x), 'r');

%-------------------------------------------------------------------------------
%                     ECRITURE DANS UN FICHIER
%-------------------------------------------------------------------------------
B=0:0.001:5
fid=fopen('coeff.txt','w')    %emplacement dans  cf pwd
fprintf(fid,'%i,\n',B')
fclose(fid)
%-------------------------------------------------------------------------------
%                     TEST, BOUCLES, ETC..
%-------------------------------------------------------------------------------
doc lang
%-------------------------------------------------------------------------------
%                     FREQUENCY DOMAIN ANALYSIS
%-------------------------------------------------------------------------------
%Définition d'une function de Transfert
g = tf([1 0.1 7.5],[1 0.12 9 0 0])

%Tracé du bode
bode(g)
bode(g,{0.1 , 100})
```

2. Prise en Main de Simulink

Simulink permet de simuler en temporel uniquement un système décrit à partir de blocs.
Il est impératif de définir dans un fichier .m le contexte, à savoir les valeurs numériques des différentes variables du schéma Simulink.
Il est alors nécessaire d'exécuter une fois le contexte pour définir ces variables dans le workspace.



- Entrer dans l'invite de commande Matlab
  >> simulink
- Ouvrir et exécuter le fichier .m de contexte '**CONTEXTE_SIMULINK.m**'.

```
%-------------------------------------------------------
%                     CONTEXTE
%-------------------------------------------------------
% Configuration Parameters
START_TIME=0
STOP_TIME=2
%-------------------------------------------------------
%                    Cont_sys
%-------------------------------------------------------
% Gain
G=5
% Constante de Temps
to=0.1
%-------------------------------------------------------
%                    Disc_sys
%-------------------------------------------------------
G1=7
t1=0.1
Te=0.05
%-------------------------------------------------------
%  Tracé dans Matlab à partir de 'To Workspace'
%-------------------------------------------------------
plot(tout, datas.signals.values)
```

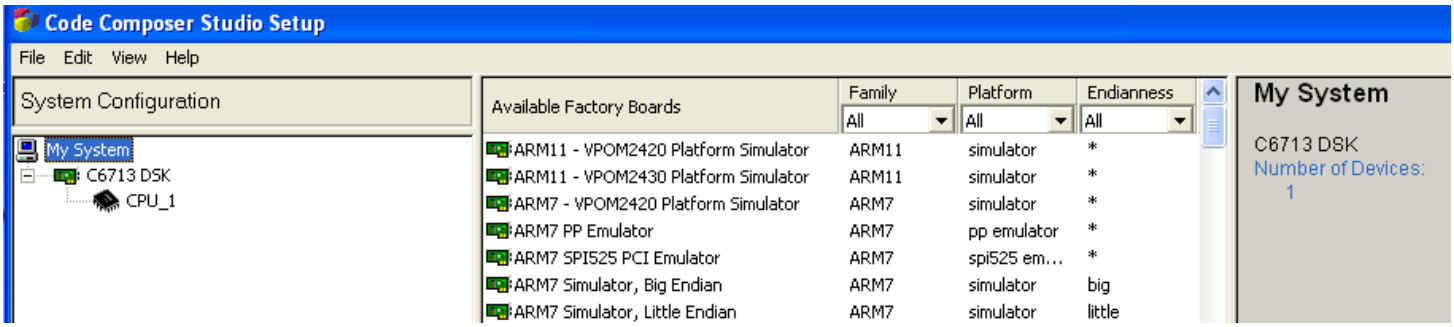
- Ouvrir le fichier '**TUTO_SIMULINK.mdl**'
- Raccourcis Clavier :
  - Zoom in : r
  - Zoom out : v
  - Start Simulation : ctrl + t

TUTORIEL 2	Génération du Code DSP à partir de Matlab-Simulink	1 UC

Configuration de Code Composer Studio (à vérifier à chaque séance)

1- Exécuter 'Setup CCStudio'

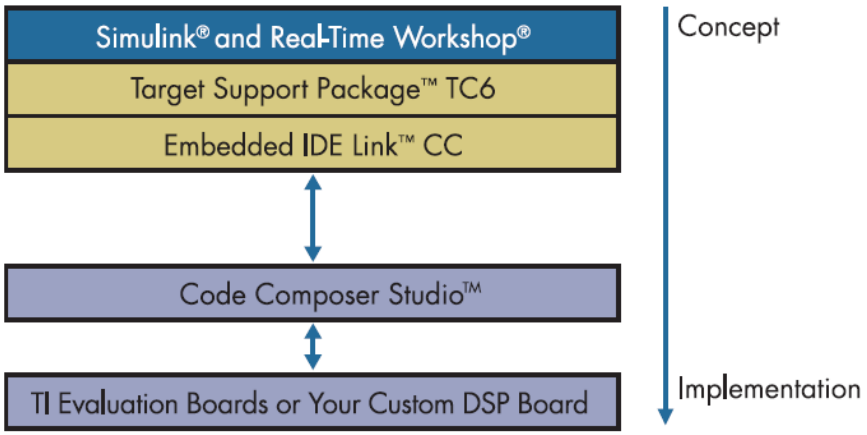
2- La cible doit être C6713 DSK-USB :



3- Faire 'save and Quit'

\A_PRISE_EN_MAIN\TUTORIEL_2

Il est possible de générer directement le code du DSP à partir du schéma-bloc conçu sous SIMULINK, en utilisant la bibliothèque Target Support Package.



Cette démarche permettant le prototypage rapide est également utilisée sur les cartes DSPACE (http://www.dspace.de/fr/fra/home.cfm)

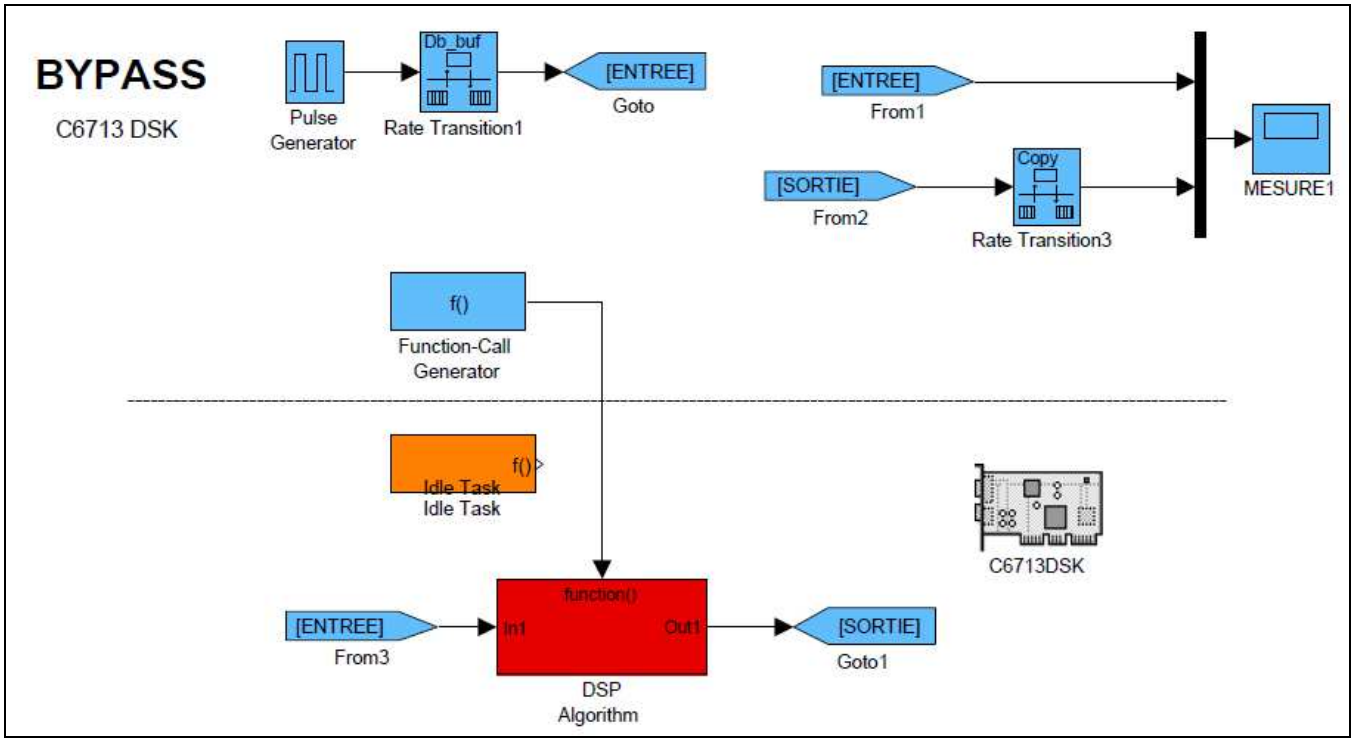
1. Programme BYPASS

Dans toute manipulation de traitement du signal ou d'automatique, il faut s'assurer dans un premier temps du bon fonctionnement des convertisseurs d'entrée et de sortie.

Ce premier programme consiste simplement à recopier l'entrée sur la sortie.

1- Simulation

BYPASS_sim.mdl



Contenu du Bloc DSP Algorithm :

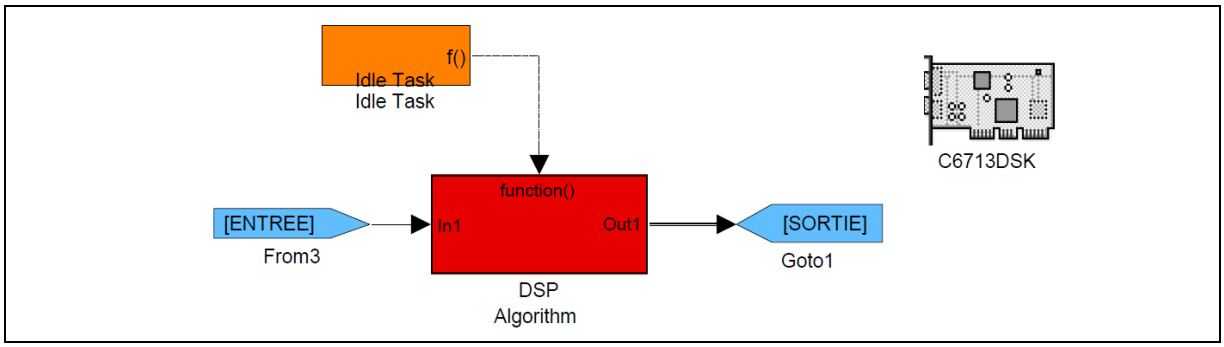
Contexte.m :

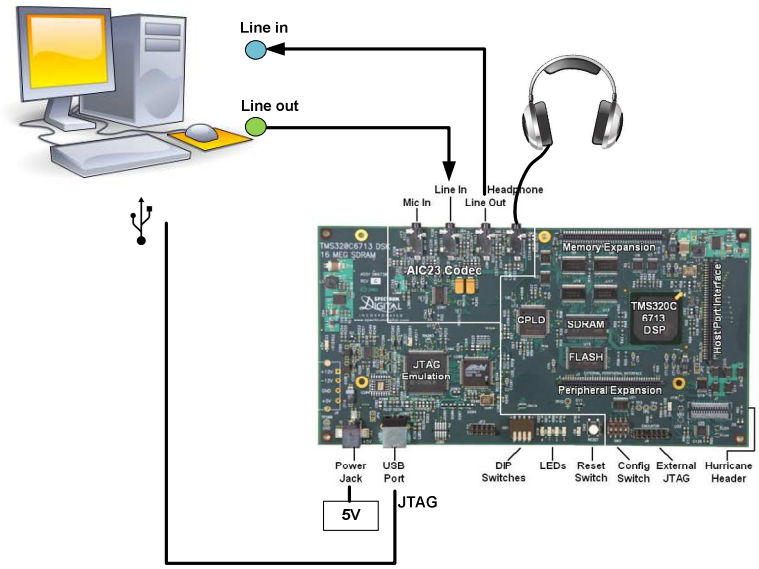
```
Fe=44100        % Fréquence d'échantillonage du CODEC
Te=1/Fe
Ts=Te/10        % Pas de Calcul du Solver
STOP_TIME=5     % Durée de la simulation
```

Exécuter le contexte, puis lancer le programme simulink (CTRL+T)

2- Test sur Cible

BYPASS_cible.mdl





Il suffit d'ouvrir le fichier BYPASS_cible.mdl et de faire CTRL+B pour générer le code du DSP. Le logiciel Code Composer Studio (CCS) se lance alors automatiquement.

Le contexte est le même pour l'essai en simulation ou sur cible.

Afin de relancer le code dans la cible (sans modification), le plus simple est de faire dans CCS :

- Disconnect (ALT+C)
- Connect(ALT+C)
- Run

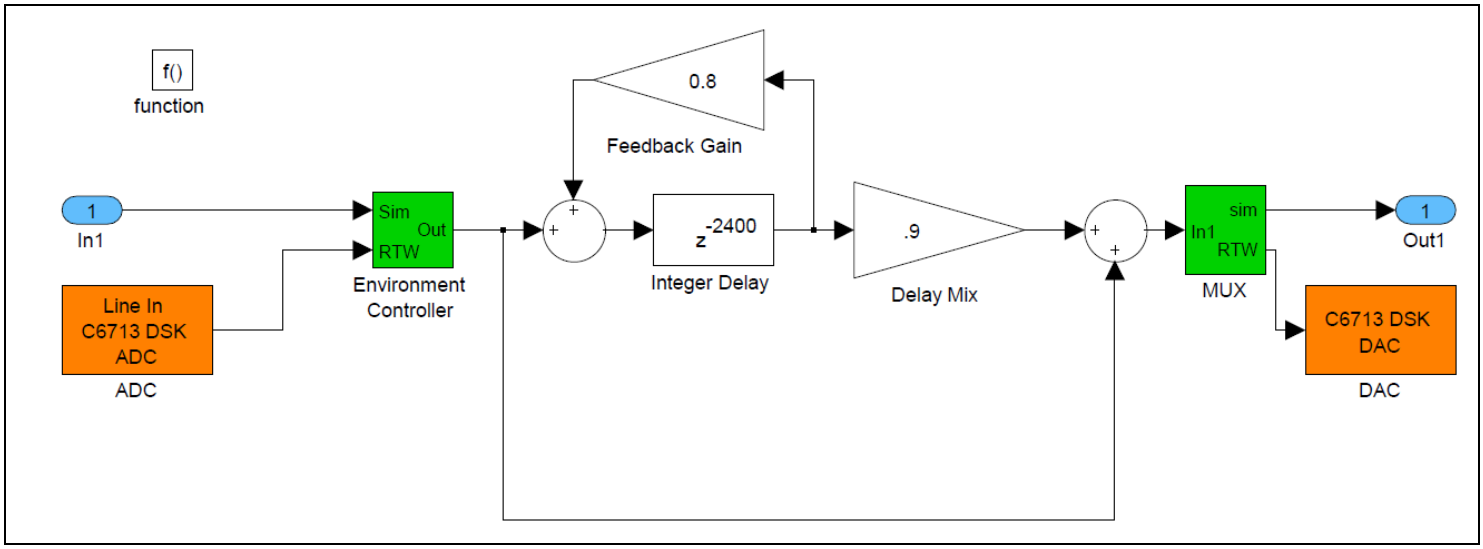
Pour relancer le code dans la cible après modification, il est nécessaire dans un premier temps de fermer CCS, et de supprimer la variable CCS_Obj dans l'environnement de MATLAB .
Il ne reste plus qu'à faire CTRL+B pour générer à nouveau le code à destination de la cible.

Faire lire un fichier son par le PC et vérifier au casque que le son est présent en sortie de la carte DSP.

2. Programme REVERB

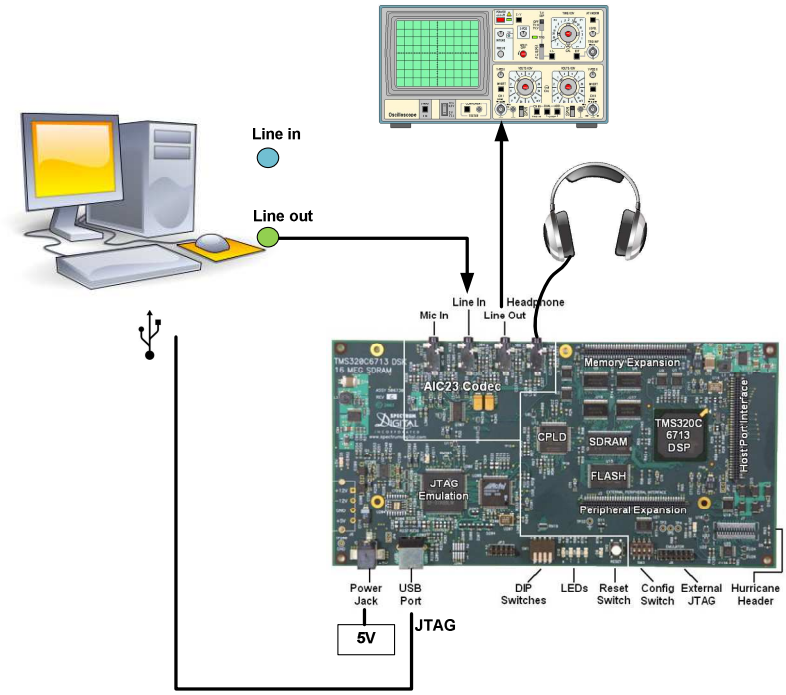
1- Simulation

REVERB_sim.mdl



2- Test sur Cible

REVERB_cible.mdl

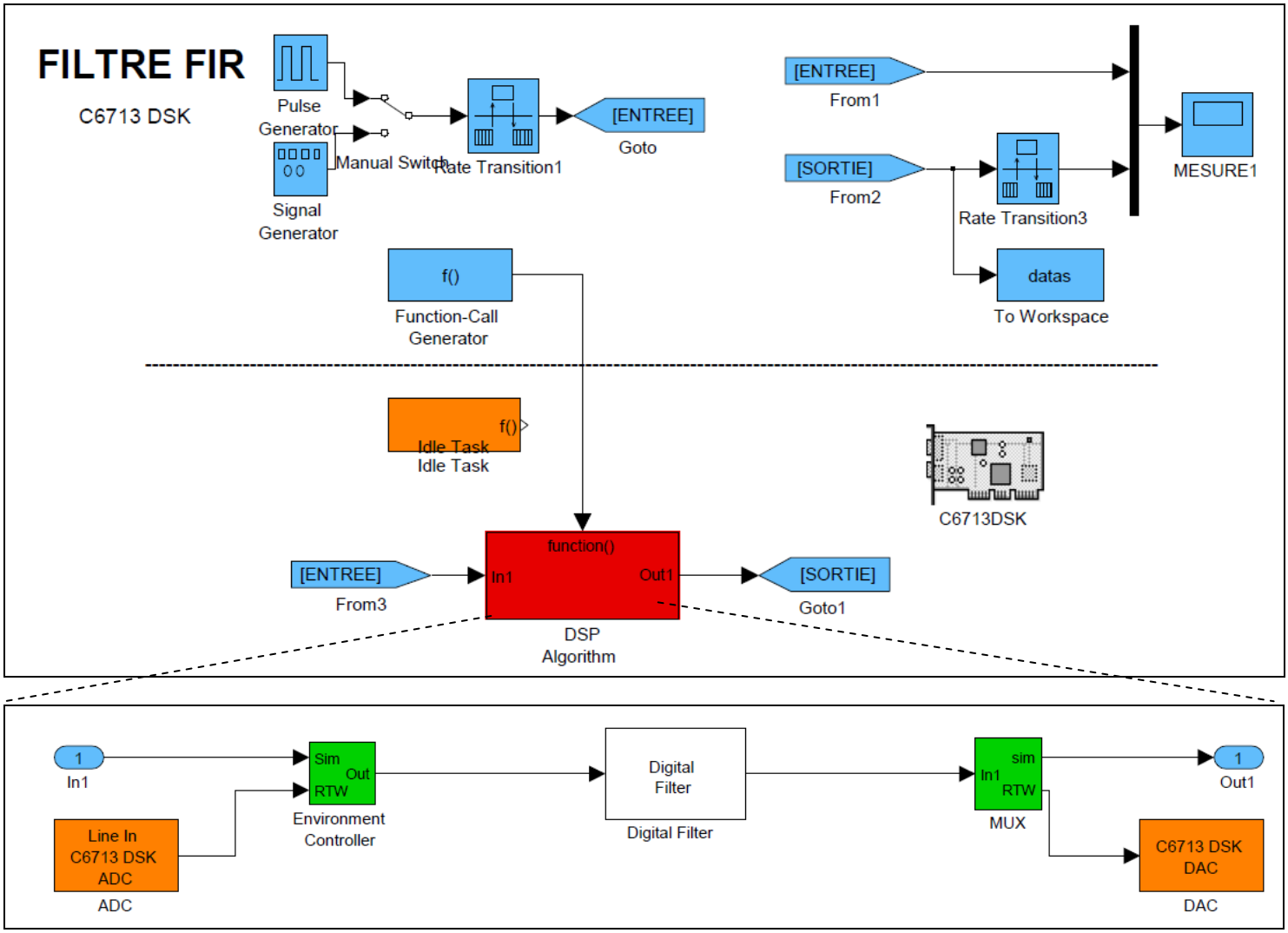


Tester ce programme dans un premier temps avec un fichier son quelconque (l'effet sera plus reconnaissable avec une voix), puis faire lire le fichier metronome_T_1s. Vérifier l'effet de reverb sur l'oscilloscope et justifier par le calcul les signaux observés (durée entre deux impulsions)

3. Filtre FIR

1- Simulation

FILTRE_FIR_sim.mdl



Simulink ne permettant pas de tracer directement des diagrammes fréquentiels, il est nécessaire de sauvegarder les points de simulation en temporel (avec un bruit blanc en entrée) dans une variable et de tracer la FFT de ces points avec le script suivant :

FILTRE_FIR_FFT.m

```
%-------------------------------------------------
%  Tracé dans Matlab à partir de 'To Workspace'
%-------------------------------------------------

B=datas.signals.values(1,1,:)
C=B(:)

[M,N]=size(C)

X=abs(fft(C,M))
X=fftshift(X)
F=Fe*[-M/2:M/2-1]/M
plot(F,X)
```

2- Test sur Cible

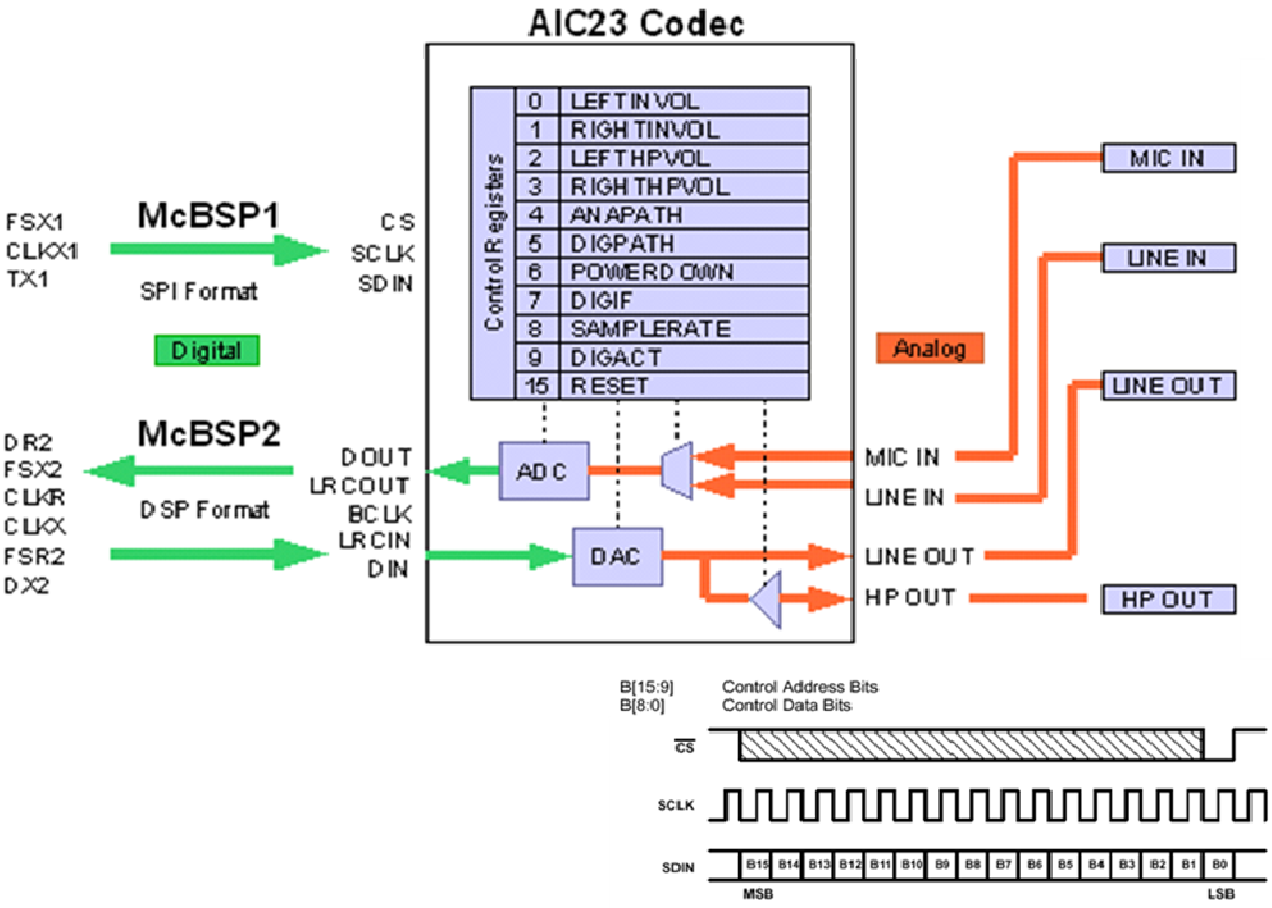
Caractériser le filtre (type + fréquence(s) de coupure) à l'aide du programme 'analyseur de spectre' (WAVETOOL → SPECTRUM ANALYZER).

Pour cela faire lire un son contenant toutes les fréquences (idéalement un bruit blanc ou de la musique)

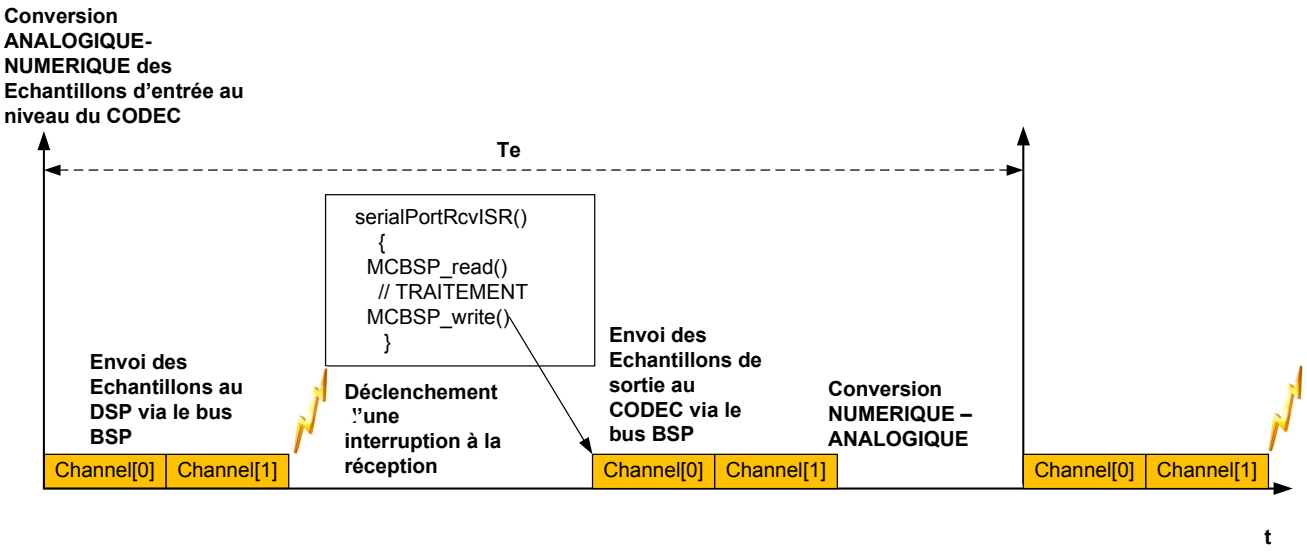
TUTORIEL 3	Programmation du DSP en C - BYPASS	45 min

\A_PRISE_EN_MAIN\TUTORIEL_3

Les signaux analogiques audios sont convertis au niveau des CODECs de la carte de développement.
Le DSP communique avec les CODEC via des bus BSP (équivalents à du SPI).
La configuration du CODEC permet de préciser entre autres la fréquence d'échantillonnage Fe.
Les Echantillons stéréo correspondent à deux entiers de 16 bits envoyés successivement.



A chaque fin de conversion, le CODEC envoie via le bus BSP les échantillons Gauche et Droite.
Cela a pour effet de déclencher une interruption serialPortRcvISR().
Cette routine peut contenir l'équation de récurrence permettant de traiter les échantillons afin de fournir les données de sortie.



Extrait de **base.c** :

```
/*-----------------------------------------------------------------
                        PROGRAMME PRINCIPAL
-----------------------------------------------------------------*/
void main()
{
      DSK6713_init();
      TIMER0_init();
      CODEC_init();
      IRQ_init();
      IRQ_globalEnable();

      while(1)
      {
      }
}

/*-----------------------------------------------------------------
          ROUTINE INTERRUPTION CODEC (AU RYTHME DE Fe)
-----------------------------------------------------------------*/
interrupt void serialPortRcvISR()
{
      union {Uint32 combo; short channel[2];} temp;
      float y;

      temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);

/*********************************************************
  A COMPLETER POUR TRAITEMENT DES ECHANTILLONS D'ENTREE
  y Résultat du Calcul
*********************************************************/

//    temp.channel[0]= (short)(y*16384.0) ;
//    temp.channel[1]= (short)(y*16384.0) ;

      MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);

}
```

TEST DU PROGRAMME

1- Ouvrir Code Composer Studio V3.3

2- Se connecter à la cible :
Debug > Connect

3- Ouvrir un projet
Project > Open

Sélectionner base.pjt

4- Créer l'Exécutable :
Project > Build (F7)

5- Charger l'Exécutable dans la cible :
CTRL+L
Puis sélectionner le fichier .out dans le répertoire DEBUG du projet.

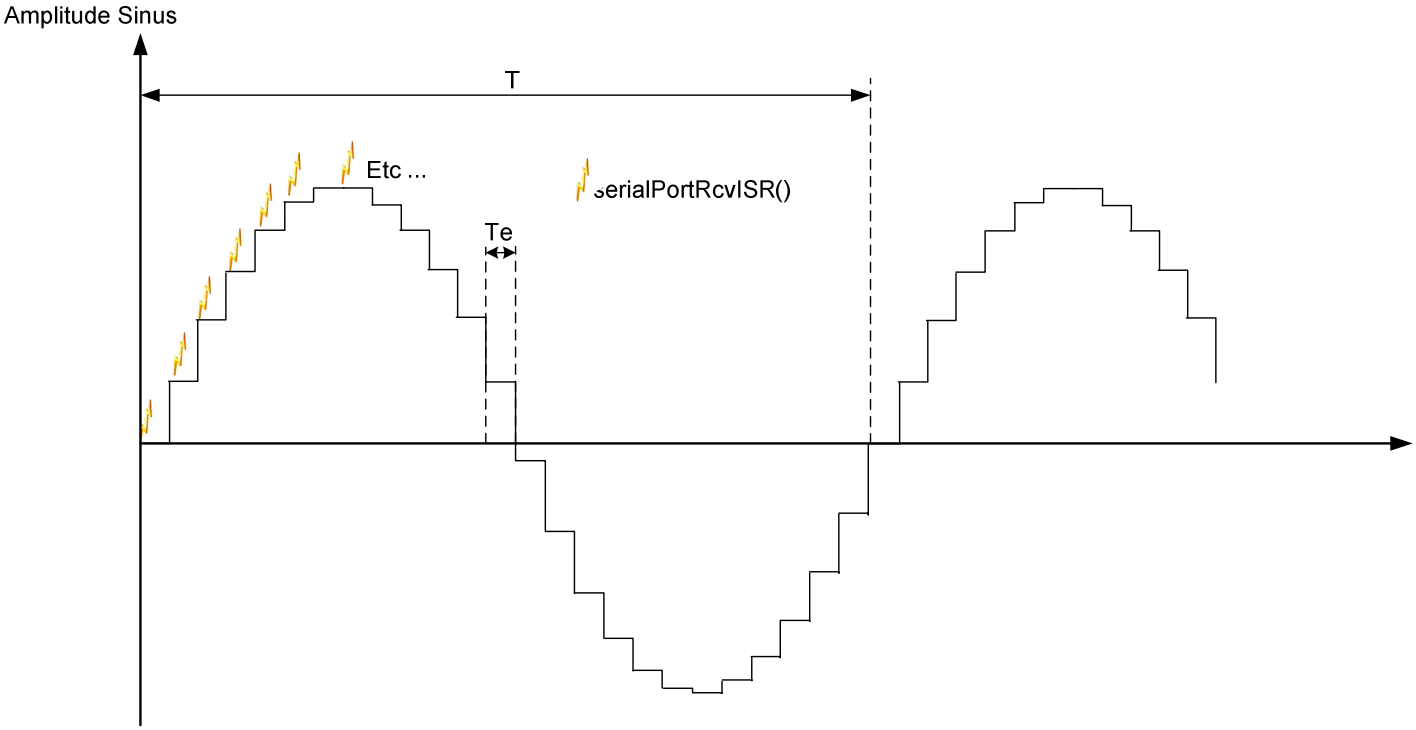
Vérifier alors le bon fonctionnement du programme BYPASS

REMARQUE :
Penser à placer des Points d'arrêt et faire du pas à pas
Watch Window → Clic Droit 'add to watch window' : Permet de visualiser l'évolution de toute variable du programme

PARTIE B	Synthèse Sonore	Durée : 4UCs
	OBJECTIFS	• Comprendre les techniques de synthèse des signaux et de modulation de fréquence

TUTORIEL	Génération d'une Sinusoïde	1.5 UC

Le problème consiste à générer à chaque période d'échantillonnage un point d'un signal.



1. Méthode 1 : Equation de Récurrence

Un oscillateur numérique, capable de générer un signal sinusoïdal de fréquence $f_0 = 1/T$ est relativement simple à mettre en œuvre.

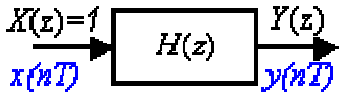
Si la phase n'a pas d'importance, ce qui est souvent le cas, on aura à générer des signaux du type $y(t) = \sin(2.\pi.f_0.t)$

La transformée en Z correspondante est :

La transformée en z est du type $Y(z) = \dfrac{b.z^{-1}}{1 + a.z^{-1} + z^{-2}}$, avec $\begin{cases} b = \sin(2.\pi.f_0.Te) \\ a = -2.\cos(2.\pi.f_0.Te) \end{cases}$

pour réaliser $yn = y(n.Te) = \sin(2.\pi.f_0.n.Te)$

1- Réalisation du signal y(nT) par excitation impulsionnelle de H(z)



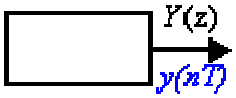
La réalisation des y(nT) peut se faire en considérant qu'ils sont la sortie du système numérique $H(z) = \dfrac{b.z^{-1}}{1 + a.z^{-1} + z^{-2}}$ (2) soumis, à partir de conditions initiales nulles, à une entrée impulsionnelle de fonction de transfert X(z) = 1.

Dans ce cas, l'équation de récurrence suivante générera le signal :

$y_n = -a.y_{n-1} - y_{n-2} + b.x_{n-1}$ (3)

et dans dans laquelle $x_n = 1$, pour n = 0 et $x_n = 0$, pour $n \neq 0$,

2- Réalisation "simple et directe" de l'oscillateur de sortie y(nT)



On peut faire plus simple :

Puisqu'il y a lieu, dans (3), de considérer comme nuls tous les termes d'indice négatif , il est plus judicieux de calculer à part les 2 premiers échantillons y0 et y1 avant d'appliquer ensuite une récurrence réduite.

Ainsi, au lieu de programmer (3) avec un signal d'entrée xn qui devient nul, sitôt passé l'instant 0, on préfèrera utiliser un algorithme en deux parties :

1- Le « lancement », l'initialisation de l'oscillateur :

$y_0 = b_0 = 0$

$y_1 = b - a$

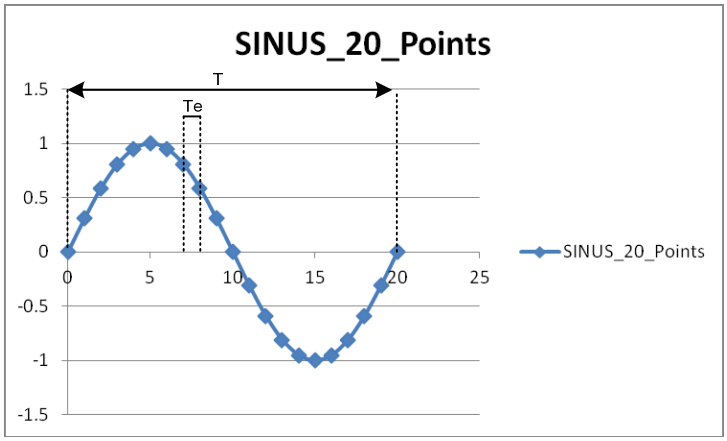
2- Et, pour n ≥ 2, une équation récurrente à exécuter en boucle : $y_n = -a.y_{n-1} - y_{n-2}$

Q1. Modifier le projet CCS BYPASS de la partie A pour générer une sinusoïde de fréquence f_0 =440 Hz avec une fréquence d'échantillonnage Fe=44.1kHz
Ce projet sera nommé GENE_SINUS_M1

2. Méthode 2 : Utilisation de Tables

Considérons 20 points de sinusoïdes :

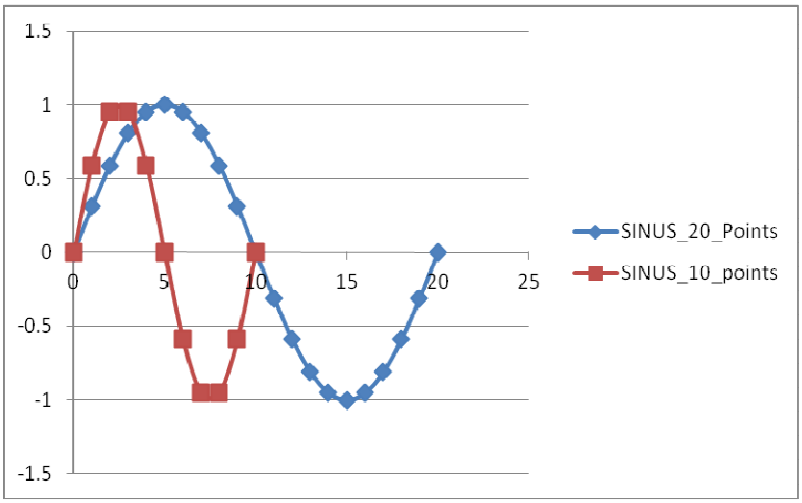
	THETA	SIN(THETA)
0	0	0
1	0.314159265	0.309016994
2	0.628318531	0.587785252
3	0.942477796	0.809016994
4	1.256637061	0.951056516
5	1.570796327	1
6	1.884955592	0.951056516
7	2.199114858	0.809016994
8	2.513274123	0.587785252
9	2.827433388	0.309016994
10	3.141592654	1.22515E-16
11	3.455751919	-0.30901699
12	3.769911184	-0.58778525
13	4.08407045	-0.80901699
14	4.398229715	-0.95105652
15	4.71238898	-1
16	5.026548246	-0.95105652
17	5.340707511	-0.80901699
18	5.654866776	-0.58778525
19	5.969026042	-0.30901699
20	6.283185307	-2.4503E-16



Si je lis tous les points à la période Te, j'aurai une sinusoïde période T=20*Te.

Si je décide de lire un point sur 2 à même période Te, j'obtiendrai une sinusoïde de période T=(20/2)*Te.

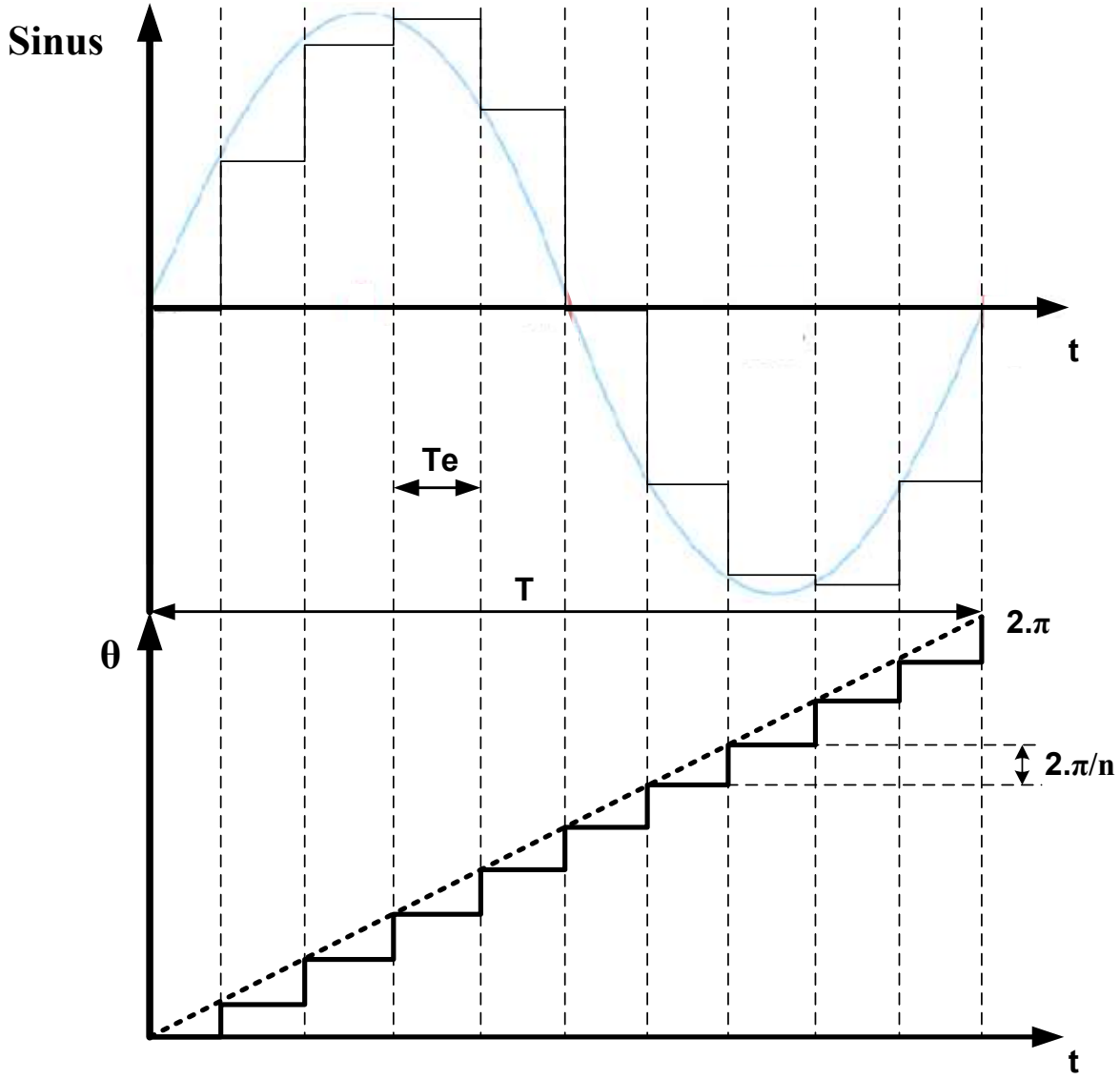
A partir d'une table contenant un ensemble de points d'un signal, je peux faire varier la fréquence de ce signal en lisant plus ou moins de points dans la table.



On considère toujours une table contenant des points de sinusoide.
On cherche à générer un signal sinusoïdal de fréquence F, à la fréquence d'échantillonage Fe.

Il s'agit faire évoluer l'angle θ tel que $\theta(t+Te)= \theta(t)+2.\pi/n$, avec $0 \leq \theta \leq 2.\pi$

n étant le nombre de points de la sinusoïde définit par n=Fe/F



La table comporte TAILLE_TABLE points ; soit k le $k^{ème}$ élément du tableau (k image de θ)
On a donc :

$0 \leq \theta \leq 2.\pi$
$0 \leq k \leq$ `TAILLE_TABLE-1`

La règle de 3 donne : k= `(TAILLE_TABLE-1)*` θ / (2.π)
Il suffira donc d'envoyer à chaque période d'échantillonage sin(k) en faisant évoluer θ.

Cette deuxième méthode a pour avantage de générer tout type de signal sans avoir à faire de calcul (mais nécessite un espace mémoire conséquent). A noter que cette méthode est utilisée dans les GBF Numériques.

Q1. Modifier le projet CCS BYPASS de la partie A pour générer une sinusoïde de fréquence 440 Hz avec une fréquence d'échantillonnage Fe=44.1kHz ; Même question pour un signal carré.

Ce projet sera nommé GENE_SINUS_M2

GENE_SON.m

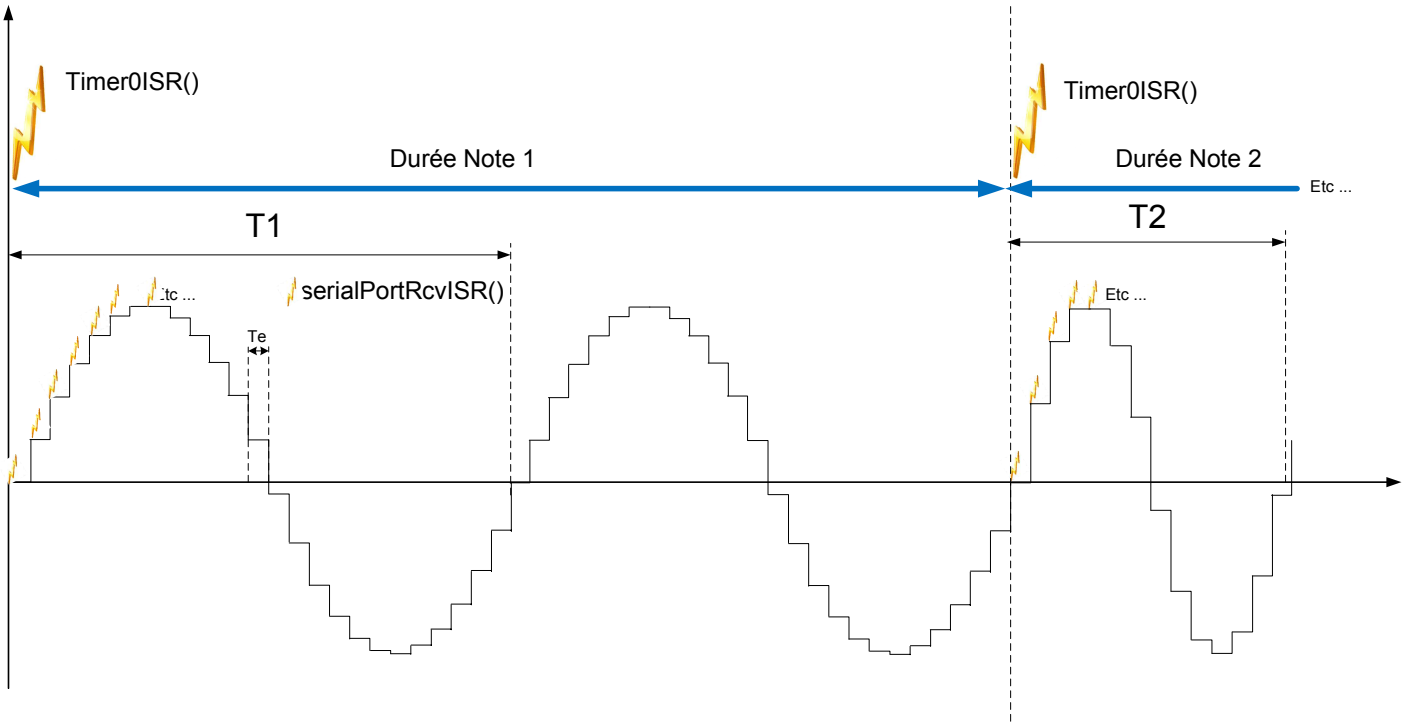
```
TAILLE_TABLE=1000

%*****************************************
% GENERATION D'UNE SINUSOÏDE
%*****************************************

figure(1)
x=0:(2*pi)/TAILLE_TABLE:2*pi
stem(sin(x));
figure(2)
plot(sin(x));
fid = fopen('sinus_float.txt', 'wt');
fprintf(fid,'%12.8f,', sin(x));
fclose(fid)

x=0:(2*pi)/TAILLE_TABLE:2*pi
stem(sin(x));

SINUS=round(2^14*sin(x))
stem(SINUS)

fid = fopen('sinus_int.txt','wt');
fprintf(fid,'%i,',SINUS');
fclose(fid)

%*********************************************************
%      GENERATION D'UN SIGNAL CARRE (à Valeur moyenne nulle)
%*********************************************************

carre=[2.0*ones(1,TAILLE_TABLE/2) zeros(1,TAILLE_TABLE/2)]
carre=carre-1
stem(carre)

fid = fopen('carre_float.txt', 'wt');
fprintf(fid,'%12.8f,', carre);
fclose(fid)


%*********************************************************
%     GENERATION D'UN SIGNAL TRIANGULAIRE
%*********************************************************

%                   A COMPLETER
```

APPLICATION 1	Génération d'un signal Triangulaire	45 min

Q1.Modifier le fichier GENE_SON.m afin de générer 1000 points d'un signal triangulaire.
Q2.Modifier GENE_SINUS_M2 pour générer ce signal triangulaire.

APPLICATION 2	Lecteur de Partition	2 UC

Le fichier partition.h permet de déclarer des tableau associant Note (Fréquence de sinusoïde) et durée.
En parcourant le champs duree_note de la structure partition, on peut mettre à jour le registre du compteur Timer0 pour déterminer le temps avant la prochaine interruption.
Lors de l'interruption, on passe à la note suivante (changement de fréquence de la sinusoïde de sortie) et on met à jour Timer0.



Q1. Modifier le projet GENE_SINUS_M2 pour pouvoir lire l'une des partitions proposées.

On appellera ce projet LECTEUR_PARTITION

APPLICATION +	Enveloppe + Polyphonie	1-2 UCs

Q1. Proposer une modification de LECTEUR_PARTITION pour associer une enveloppe trapézoïdale à chaque note.
Q2. Modifier le programme précédent pour pouvoir lire 2 structures simultanément (il est possible d'utiliser Timer1, penser alors à modifier la table des vecteurs dans le fichier vectors.asm)

PARTIE C	Filtrage	Durée : 2.5UC
	OBJECTIFS	• Apprendre à Mesurer les temps d'exécution dans un processeur • Comparer plusieurs solutions de codage • Mettre en œuvre le Filtrage pour éliminer une fréquence parasite

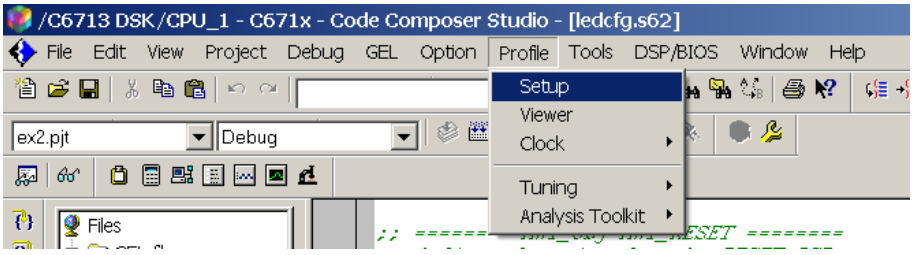
TUTORIEL	Mesure des Temps d'Exécution de la Carte	

La fonction profiling ou benchmarking permet de connaître le temps d'exécution de portions de codes d'un programme.

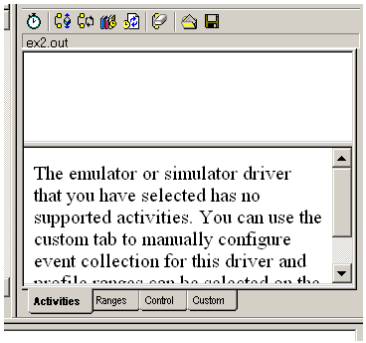
Ce diagnostique s'établit en temps réel **et affecte donc le temps d'exécution du programme en cours** (soit on écoute/mesure le traitement du DSP, soit on mesure le temps d'exécution)

1- Compiler et charger le fichier .out dans le DSP

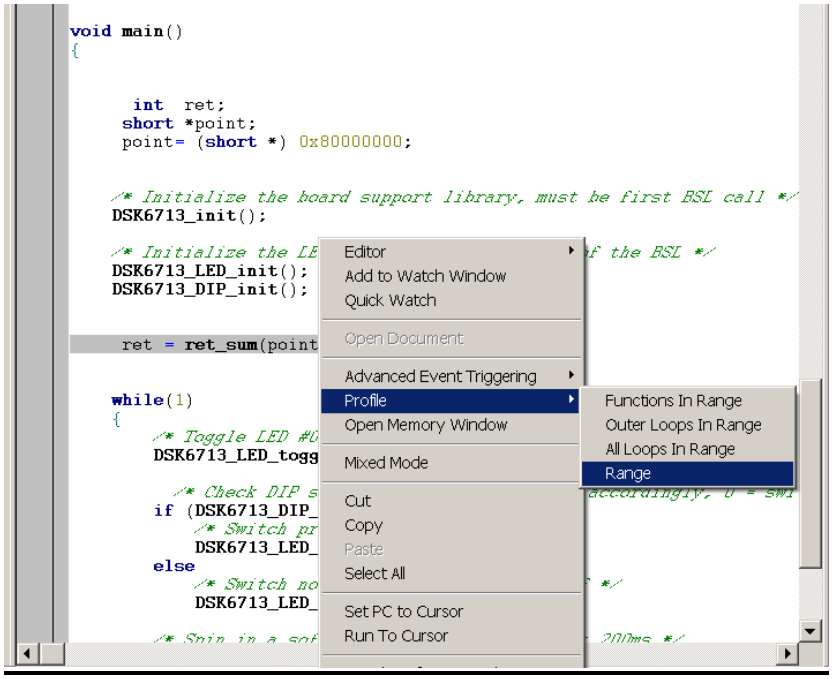
2- Faire PROFILE → SETUP



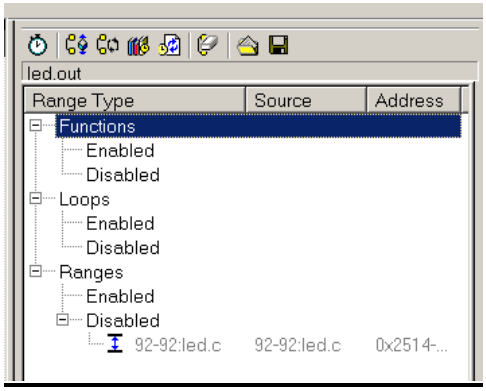
La fenêtre suivante apparaît :



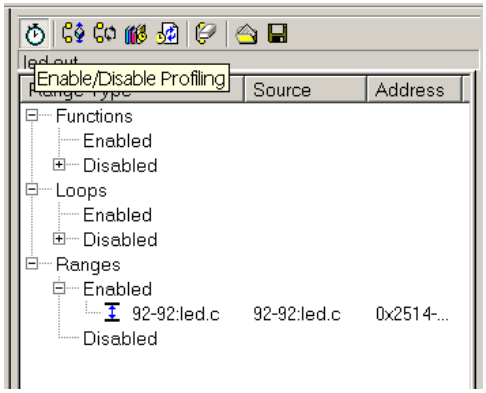
3- Surligner la ligne ou fonction dont on veut connaitre le temps d'exécution et faire :
CLIC DROIT → Profile > Range



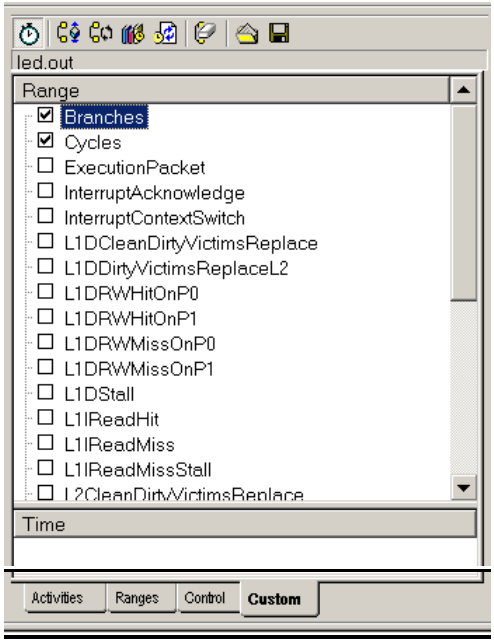
Dans l'onglet RANGE de la fenêtre PROFILE apparaît alors :



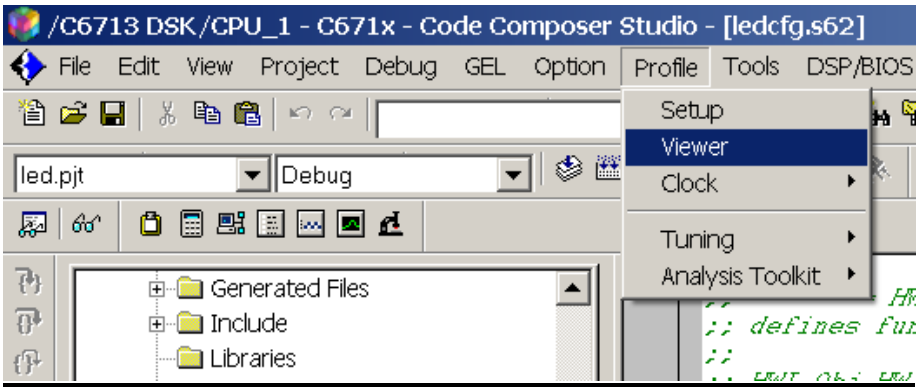
4- Autoriser le Profiling en cliquant sur la montre :



5- Sélectionner les informations à afficher dans l'onglet CUSTOM :

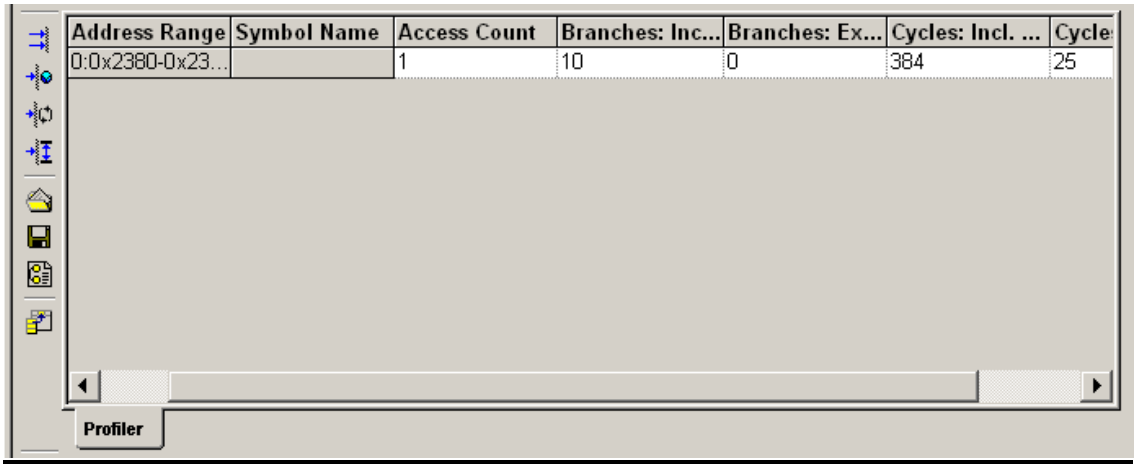


6- Faire PROFILE → VIEWER



7- Lancer alors le programme pour permettre la mesure (puis stopper)

Il apparaît alors la fenêtre suivante :

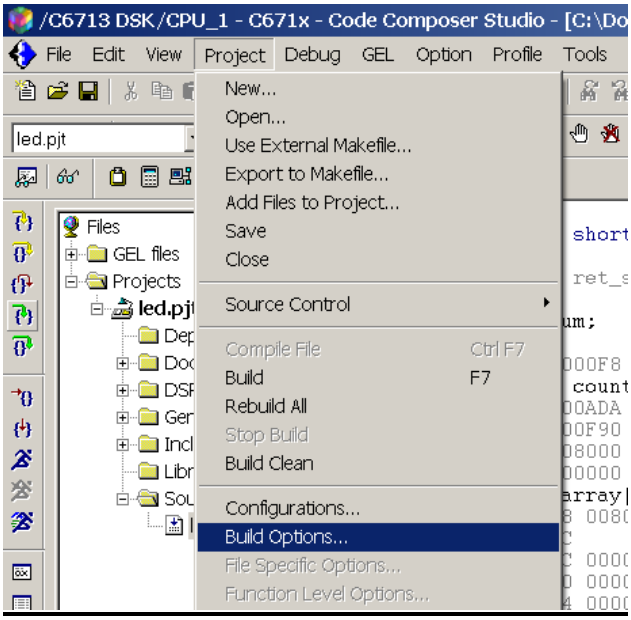


Access Count → Nombre d'exécution de la ligne à tester
Cycles Incl → nombre de coups d'horloge total
Cycles Excl → nombre de coups d'horloge en ne considérant qu'une seule itération des boucles

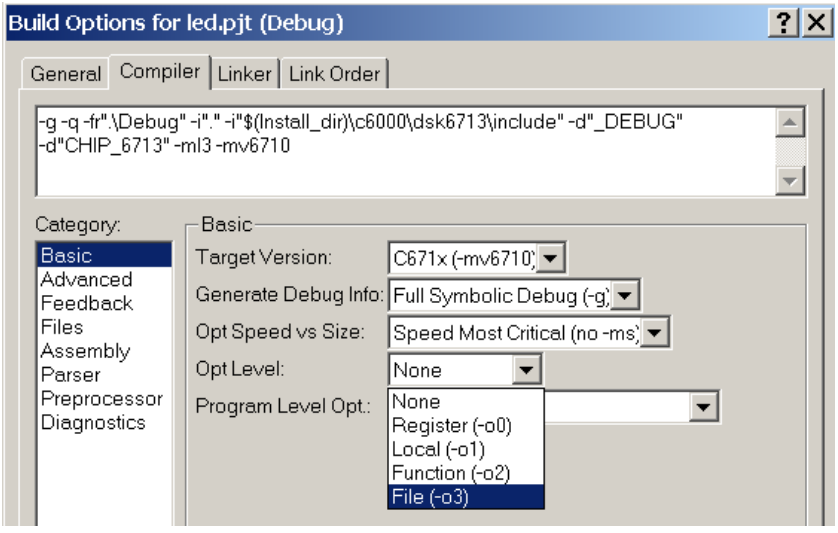
On a donc : Temps d'Exécution de la Ligne = (Cycles Incl / Access Count) * T_CPU

REMARQUE : Plusieurs niveaux de compilations sont possibles.

Faire PROJECT → BUILD OPTIONS



Le champs Opt Level est alors modifiable :



Recompiler alors le projet.

APPLICATION 1	Caractérisation de Filtres et Mesure des Temps d'Exécution	1UC

ATTENTION, LES FILTRES FONCTIONNENT A UNE FREQUENCE D'ECHANTILLONNAGE DE 24kHz (cf CODEC_init() → DSK6713_AIC23_setFreq)

Q1. Compléter le tableau suivant :

	FIR_FLOAT		FIR_INT_C		FIR_INT_ASM	FIR_INT_ASM_OPT
Niveau de Compilation	O0	O3	O0	O3	O0	O0
Type de Filtre (PB/PH/PBde/SBde)						
Fréquence(s) de Coupure						
Nombre de Cycles d'exécution (profile)						
Temps d'exécution						

TUTORIEL	Génération des Coefficients d'un Filtre FIR avec Matlab	

A l'aide de MATLAB on récupère l'ensemble des coefficients pour un filtre FIR que nous allons étudier :

H=	-0.0023910978,	0.0212210438,	-0.0070036552,	0.0019809643,
0.0000000000,	-0.0033129903,	0.0077212135,	-0.0145502755,	0.0012622031,
0.0000026569,	-0.0028815259,	-0.0128436380,	-0.0153286573,	0.0002071354,
0.0000136101,	-0.0010281241,	-0.0330632466,	-0.0100796514,	-0.0007015681,
0.0000246801,	0.0016589890,	-0.0436825693,	-0.0017180422,	-0.0011392103,
0.0000145387,	0.0040892797,	-0.0366546170,	0.0060838248,	-0.0010395179,
-0.0000341119,	0.0050903342,	-0.0081772863,	0.0104004045,	-0.0005673467,
-0.0001128816,	0.0039414942,	0.0393832465,	0.0100656511,	-0.0000000000,
-0.0001785952,	0.0007888470,	0.0974657869,	0.0058739784,	0.0004137162,
-0.0001689047,	-0.0032725636,	0.1533443079,	0.0000000000,	0.0005511119,
-0.0000400583,	-0.0065424732,	0.1936590301,	-0.0050358998,	0.0004363988,
0.0001923004,	-0.0073951421,	0.2083425888,	-0.0073951421,	0.0001923004,
0.0004363988,	-0.0050358998,	0.1936590301,	-0.0065424732,	-0.0000400583,
0.0005511119,	0.0000000000,	0.1533443079,	-0.0032725636,	-0.0001689047,
0.0004137162,	0.0058739784,	0.0974657869,	0.0007888470,	-0.0001785952,
-0.0000000000,	0.0100656511,	0.0393832465,	0.0039414942,	-0.0001128816,
-0.0005673467,	0.0104004045,	-0.0081772863,	0.0050903342,	-0.0000341119,
-0.0010395179,	0.0060838248,	-0.0366546170,	0.0040892797,	0.0000145387,
-0.0011392103,	-0.0017180422,	-0.0436825693,	0.0016589890,	0.0000246801,
-0.0007015681,	-0.0100796514,	-0.0330632466,	-0.0010281241,	0.0000136101,
0.0002071354,	-0.0153286573,	-0.0128436380,	-0.0028815259,	0.0000026569,
0.0012622031,	-0.0145502755,	0.0077212135,	-0.0033129903,	0.00000000000
0.0019809643,	-0.0070036552,	0.0212210438,	-0.0023910978,	
0.0019283786,	0.0051750403,	0.0241496980,	-0.0007075582,	
0.0009458712,	0.0173941788,	0.0173941788,	0.0009458712,	
-0.0007075582,	0.0241496980,	0.0051750403,	0.0019283786,	

On désire exécuter les calculs avec des entiers. On cherche donc à déterminer le format de codage en virgule fixe. On cherche le nombre le plus grand :

Max(H) = 0.2083425888

On peut faire tenir ce nombre sur un format « 1.15 »

0.2083425888

2^0	2^-1	2^-2	2^-3	2^-4	2^-5	2^-6	2^-7	2^-8	2^-9	2^-10	2^-11	2^-12	2^-13	2^-14	2^-15
0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	1

Arrondi(0.2083425888* 2^15)= 6827

6827

2^15	2^14	2^13	2^12	2^11	2^10	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	1

On constate que 6827et 0.2083425888 ont le même codage, il suffit donc de multiplier l'ensemble des coefficients par 2^15.

B_entier=

0,	35,	99,	-354,	5149,	132,	-98,	-17,
0,	26,	190,	102,	2908,	363,	-17,	-15,
0,	0,	195,	598,	727,	395,	59,	-7,
-1,	-34,	96,	876,	-853,	242,	95,	1,
-1,	-58,	-71,	739,	-1545,	0,	81,	6,
0,	-55,	-228,	156,	-1385,	-207,	34,	6,
3,	-21,	-290,	-676,	-676,	-290,	-21,	3,
6,	34,	-207,	-1385,	156,	-228,	-55,	0,
6,	81,	0,	-1545,	739,	-71,	-58,	-1,
1,	95,	242,	-853,	876,	96,	-34,	-1,
-7,	59,	395,	727,	598,	195,	0,	0,
-15,	-17,	363,	2908,	102,	190,	26,	0,
-17,	-98,	132,	5149,	-354,	99,	35,	0
-9,	-142,	-210,	6826,	-575,	-24,	26,	
7,	-117,	-501,	7447,	-501,	-117,	7,	
26,	-24,	-575,	6826,	-210,	-142,	-9,	

GENE_FILTRE_FIR.m

```
%*************************************************************
% GENERATION DE COEFFICIENTS POUR FILTRE FIR
%-------------------------------------------------------------
% TSN - LABOS DSP
% kerhoas@enib.fr
%*************************************************************
N=128;                              %Ordre du Filtre
Fe=24000                            %Fréquence d'échantillonage
We=2*pi*Fe

Fc=5000;                            %Fréquence de Coupure
Wc=2*pi*Fc

Wn=Wc/(We/2);                       %Fréquence normalisée

B=fir1(N,Wn,'low')                  %Calcul des coefficients
                                    % d'un filtre FIR
C=B(1,1:N)

freqz(C)
max(C)
min(C)

C_entier=round(C*2^15)

fid=fopen('coeff_entiers.txt','w')  %emplacement à vérifier avec pwd
fprintf(fid,'%i,\n',C_entier')
fclose(fid)
```

```
fid=fopen('coefficients.txt','w')
fprintf(fid,'%6.12f,\n ',C')
fclose(fid)

%------------------------------------------------------------
% Calcul d'un passe bande / Coupe Bande
%------------------------------------------------------------

N=128;                                  %Ordre du Filtre
Fe=24000                                %Fréquence d'échantillonage
We=2*pi*Fe

Fc1=1000;                               %Fréquence de Coupure
Fc2=3000;
Wc1=2*pi*Fc1
Wc2=2*pi*Fc2

Wn=[Wc1,Wc2]/(We/2);                    %Fréquence normalisée

B=fir1(N,Wn,'low')
C=B(1,1:N)

freqz(C)
max(C)
C_entier=round(C*2^15)

fid=fopen('coeff_entiers.txt','w')
fprintf(fid,'%i,\n',C_entier')
fclose(fid)
```

APPLICATION 2	**Elimination d'une Sinusoïde Parasite**	45 min

Q1. Repérer la fréquence parasite du fichier son JAMES_BOND_SINUS et proposer un filtre permettant de l'éliminer (le type de codage est au choix)

APPLICATION 3	**Filtre Configurable**	1 UC

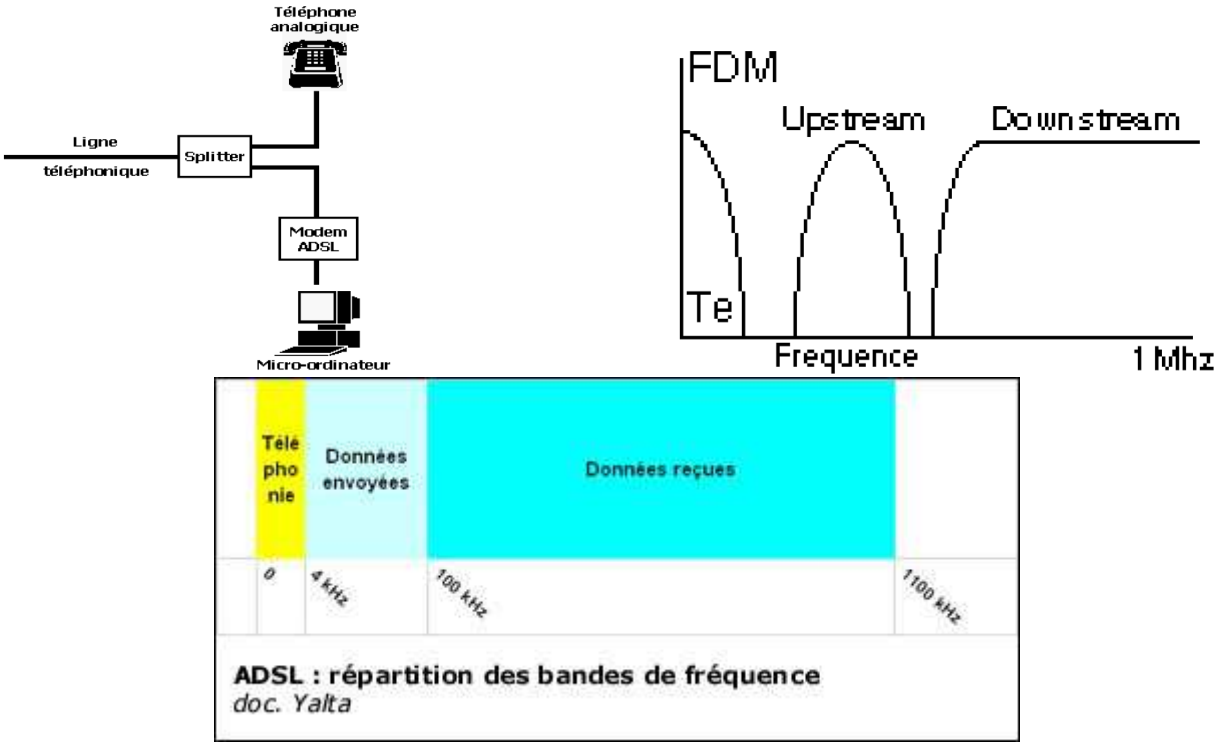
Q1. Proposer un programme d'un filtre PASSE BAS/PASSE HAUT configurable avec les boutons poussoir. En fonction des combinaisons sur les boutons (à difinir), le filtre devra avoir les fréquences de coupure suivantes : 1 kHZ, 5 kHZ, 10 kHz, 15 kHz

Cette synthèse de filtre peut se faire soit à partir de Simulink, soit directement sur CCS.

APPLICATION +	Filtre IIR pour la téléphonie	1-2 UC

L'ADSL est une technique permettant d'utiliser un support déjà en place (les fils de cuivre téléphonique) pour transmettre les informations internet. En effet le téléphone utilise une bande passante d'environ 4 kHz, alors que la bande passante du cuivre va au-delà du mégahertz.

Par un simple multiplexage fréquentiel, on peut alors recevoir internet via le réseau téléphonique, d'où le succès rencontré.



ADSL : répartition des bandes de fréquence
doc. Yalta

Il s'agit de tester le filtre passe bas numérique du commutateur avec la carte DSP.

Caractéristiques du Filtre :

- Fréquence d'échantillonnage : 44.1 kHz
- 100 dB d'atténuation maximum au-delà de 4300 Hz
- Ondulation maximum dans la bande passante : 0,1 dB

Il n'y a pas de contrainte de phase, on choisira un filtre de chebychev de type 1

Q1. Proposer un Programme MATLAB permettant de fournir les coefficients du filtre IIR Correspondant.

Q2.Mettre en œuvre ce filtre, soit sur Simulink, soit directement en C sur CCS.