UNIVERSITE DE JIJEL
Département d'électronique
ESE16/TP N5
Email: adelmellit2013@gmail.com

Transformée de Fourier discrète et rapide (TFD et FFT)

Objectif : L'objectif principal de ce TP est de familiariser les étudiants avec l'implémentation de base des algorithmes FFT/DFT sur la carte d'évaluation DSK6713.

Matériels : Carte DSK6713, Matlab et CCS 5.5

Prérequis : FFT, DFT, Langages de programmation C, ou C^{++}

La transformation de Fourier discrète (TFD), outil mathématique, sert à traiter un signal numérique. ... La TFD permet seulement d'évaluer une représentation spectrale discrète (spectre échantillonné) d'un signal discret (signal échantillonné) sur une fenêtre de temps finie (échantillonnage borné dans le temps). Le programme utilisé pour calculer la DFT pour le signal d'entrée est illustré ci-dessous.
La transformation de Fourier rapide (sigle anglais : FFT ou fast Fourier transform) est un algorithme de calcul de la transformation de Fourier discrète (TFD). ... Son efficacité permet de réaliser des filtrages en modifiant le spectre et en utilisant la transformation inverse (filtre à réponse impulsionnelle finie).

Partie 1 soit le code suivant DFT :

```
#include <stdio.h>
#include <math.h>
void dft(short *x, short k, int *out); //function prototype
#define N 8 //number of data values
float pi = 3.1416;
int out[2] = {0,0}; //init Re and Im results
short x[N] = {1000,707,0,-707,-1000,-707,0,707}; //1-cycle cosine
//short x[N]={0,602,974,974,602,0,-602,-974,-974,-602,
// 0,602,974,974,602,0,-602,-974,-974,-602};//2-cycles sine
void dft(short *x, short k, int *out) //DFT function
{
int sumRe = 0, sumIm = 0; //init real/imag components
float cs = 0, sn = 0; //init cosine/sine components
int i = 0;
for (i = 0; i < N; i++) //for N-point DFT
{
cs = cos(2*pi*(k)*i/N); //real component
sn = sin(2*pi*(k)*i/N); //imaginary component
sumRe = sumRe + x[i]*cs; //sum of real components
sumIm = sumIm - x[i]*sn; //sum of imaginary components
}
out[0] = sumRe; //sum of real components
out[1] = sumIm; //sum of imaginary components
}
void main()
{
int j;
```

```
for (j = 0; j < N; j++)
{
dft(x,j,out); //call DFT function
```

Etapes:

1. Pour vérifier les résultats, Exécuter le projet, charger le programme et suivre les étapes : Sélectionner View → voir la Window et insérez les deux expressions j et out (clic droit sur la fenêtre Watch). Cliquez sur + out pour développer et afficher [0] et out [1], qui représentent respectivement les composants réels et imaginaires.

2. Placez un point d'arrêt sur le crochet «}» qui suit l'appel de fonction DFT.

3. Sélectionnez Débogage → Animer (la vitesse d'animation peut être contrôlée via les options). Vérifiez que la valeur réelle du composant out [0] est grande (3996) à j = 1 et à j = 7, tandis qu'elle est petite sinon. Puisque x [n] est une séquence à un cycle, m = 1. Puisque le nombre de points est N = 8, un «pic» se produit à j = m = 1 et à j = N - m = 7.

4. Utilisez le tableau de données sinusoïdales à deux cycles (dans le programme) avec 20 points comme entrée x [n]. Dans le programme, modifiez N = 20, commentez la table qui correspond au cosinus (première entrée) et utilisez à la place les valeurs de la table des sinus. Reconstruisez et animez à nouveau. Vérifiez une grande valeur négative à j = m = 2 (-10,232) et une grande valeur positive à j = N - m = 18 (10,232). Pour une implémentation en temps réel, la magnitude de X (k) k = 0,1, ... peut être trouvée. Avec F_s = 8 kHz, la fréquence générée correspondrait à f = 800 Hz.

Travail demandé :

Une FFT de 256 points en temps réel sera implémentée en temps réel à l'aide d'un signal d'entrée externe. La mise en œuvre de cette partie de l'expérience repose sur la fonction FFT et sur le code ci-dessous

```
nclude "dsk6713_aic23.h"
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
#include <math.h>
#define PTS 256 //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n); //FFT prototype
float iobuffer[PTS]; //as input and output buffer
float x1[PTS]; //intermediate buffer
short i; //general purpose index variable
short buffercount = 0; //number of new samples in iobuffer
short flag = 0; //set to 1 by ISR when iobuffer full
COMPLEX w[PTS]; //twiddle constants stored in w
COMPLEX samples[PTS]; //primary working buffer
main()
{
for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/512.0); //Re component of twiddle constants
w[i].imag =-sin(2*PI*i/512.0); //Im component of twiddle constants
}
```

```
comm_intr(); //init DSK, codec, McBSP
while(1) //infinite loop
{
while (flag == 0) ; //wait until iobuffer is full
flag = 0; //reset flag
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
iobuffer[i] = x1[i]; //processed frame to iobuffer
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(samples[i].real*samples[i].real
+ samples[i].imag*samples[i].imag)/16;
}
x1[0] = 32000.0; //negative spike for reference
} //end of infinite loop
} //end of main
interrupt void c_int11() //ISR
{
output_sample((short)(iobuffer[buffercount]));//output from iobuffer
iobuffer[buffercount++]=(float)((short)input_sample());//input>iobuffer
if (buffercount >= PTS) //if iobuffer full
{
buffercount = 0; //reinit buffercount
flag = 1; //set flag
}
}
```

Les constantes twiddle sont générées dans le programme. Les composants imaginaires des données d'entrée sont mis à zéro pour illustrer cette implémentation. L'amplitude de la FFT résultante (mise à l'échelle) est prise pour sortie vers le codec. Trois tampons sont utilisés:

1. échantillons: contient les données à transformer

2. iobuffer: utilisé pour produire des données traitées ainsi que pour acquérir de nouvelles données échantillonnées en entrée

3. x1: contient la magnitude (mise à l'échelle) des données transformées (traitées)

Dans chaque période d'échantillonnage, une valeur de sortie provenant d'un tampon (iobuffer) est envoyée au DAC du codec et une valeur d'entrée est acquise et stockée dans le même tampon. Un index (buffercount) de ce tampon est utilisé pour définir un indicateur lorsque ce tampon est plein.

Lorsque ce tampon est plein, il est copié dans un autre tampon (échantillons), qui sera utilisé lors de l'appel de la fonction FFT. L'amplitude (mise à l'échelle) des données FFT traitées, contenues dans un tampon x1, peut maintenant être copiée dans le tampon d'E / S, iobuffer, pour la sortie. Dans un algorithme de filtrage, le traitement peut être effectué à mesure que chaque nouvel échantillon est acquis. D'un autre côté, un algorithme FFT nécessite qu'une trame entière de données soit disponible pour le traitement.

- Pour tester l'entrée du programme, une onde sinusoïdale de 2 kHz avec une amplitude de 1 Vc.
- Connectez la sortie LINE OUT du kit DSK à l'oscilloscope et observez le signal mesuré
- Mesurez la fréquence des premier et deuxième pics positifs et expliquez le résultat obtenu
- Mesurez l'intervalle de temps entre les deux pics négatifs. Que représente cet intervalle?

_____-

[1] **Référence :**

1. https://www.mathworks.com/matlabcentral/fileexchange/19770-simulink-labs-based-on-dsp-first-labs
2. https://eng.najah.edu/media/filer_public/00/c2/00c2c468-b7da-4c0d-9619-eeb1a77a5b07/
3. https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/s/89952_93049v00_Simulink_Real-Time_Whitepaper.pdf
4. Chassaing, R. (2004). Digital Signal Processing and Applications with the C6713 and C6416 DSK (Vol. 16). John Wiley & Sons.
5. Kumar, B. P. (2016). Digital signal processing laboratory. CRC press.
6. https://www.youtube.com/watch?v=9DGjAKEB0eU // TMS320C6713 DSK Tutorial 7A - FIR and IIR filter design using MATLAB & SIMULINK