Step DFT
Obtenir la DFT d'un tableau de 64 éléments
(limitée aux 6 valeurs de fréquences des pistolets)

Documents utiles :
Cortex-M3 Programming Manual (PM0056.pdf)

Le point théorique …
Rappel DFT
Le format fractionnaire

1. Introduction

Situation dans le projet :

Projet :

- Step Son : la cible doit émettre un bruit de bris de verre si le joueur a touché une cible
- **Step DFT : mise en place de l'analyse DFT sur 6 canaux en format fractionnaire (virgule fixe)**
- Step DFT Réel : ajout de la DMA pour une utilisation en contexte réel
- Step Intégration: fusion de tout les éléments

Plus particulièrement nous allons gérer l'aspect DFT du projet.

1.1. Petit rappel

On part d'un signal temporel échantillonné **x(n)**, **n** allant de **0** à **63**.
On souhaite savoir si ce signal contient une sinusoïde de **fréquence normalisé k** (de 0 à 63) et en particulier, déterminer son amplitude. On utilise alors la **DFT** appliquée à la dite fréquence. Une fréquence normalisée **k = 1** signifie **1 période** sur les 64 points de x(n).

On rappelle le paramétrage de la DFT déterminé dans la première partie du BE :

- Durée de fenêtrage **T** = 200µs
- Fréquence d'échantillonnage **Fe** = 320 kHz (période d'échantillonnage Te=1/Fe)
- → nombre de points **M** = 64

1. Déterminer les 6 valeurs de k (k_1 à k_6) correspondant aux 6 fréquences des pistolets (voir votre rapport intermédiaire ou le sujet signal de la partie I, sur Moodle)

La fréquence normalisée **k** étant connue, la DFT en ce point fréquentiel se calcule comme ceci :

$$X(k)=\sum_{n=0}^{n=M-1} x(n).e^{-j\frac{2.\pi.k.n}{M}}$$ (version non normalisée) , **X(k)** est une grandeur complexe.

Enfin, l'amplitude de la DFT que nous recherchons au point k est $|(X(k))|$

1.2. Mise en œuvre sur micro-contrôleur

Le signal à traiter sera contenu dans un tableau de 64 éléments de 16 bits qui seront ceux capturés par l'*ADC* à la fréquence de 320kHz (l'étendue des 16 bits de sera pas exploitée puisque l'*ADC* est un 12bits).

Dans cette partie, on utilisera pas l'*ADC* directement. On mettra à disposition un signal de référence préfabriqué (un cosinus échantillonné de fréquence normalisée k=1 retranscrite dans un fichier assembleur *signal.asm*) et vous aurez la possibilité de créer vos propres signaux (votre propre fichier *signal.asm)* via un script *Matlab* (disponible sur Moodle) qui produit directement le fichier assembleur contenant le signal. Il sera possible de spécifier, amplitude, phase, fréquence normalisée.

La fonction de calcul de la DFT sera donc une boucle. Le calcul complexe sera séparé en deux parties, la partie réelle et la partie imaginaire.

Deux difficultés dans cet exercice :

- comment peut-on coder des nombres inférieurs à l'unité sans recourir au format float ?
- comment optimiser le calcul des sinus et cosinus ?

1.2.1. Calcul des sinus et cosinus

→ on ne fait pas les calculs ! On utilise deux tables qui contiennent déjà les valeurs calculées. L'angle des cosinus est $\frac{2.\pi.k.n}{M}$, k et n étant entiers, le produit $p = k.n$ l'est aussi. Nous aurons donc besoin de deux tables, *TabCos(p)* et *TabSin(p)* de 64 éléments chacune :

TabCos(p) :

p	***TabCos(p)***
0	$\cos(0)$
1	$\cos(\frac{2.\pi}{64})$
2	$\cos(\frac{2\cdot\pi\cdot 2}{64})$
...	
63	$\cos(\frac{2\cdot\pi\cdot 63}{64})$

Le calcul se résume donc à un accès dans un tableau (*look-up Table* en anglais)

1.2.2. Codage des nombres fractionnaires

La travail se fait en assembleur pour optimiser la vitesse, l'UAL est entière (non flottante). Le codage en float est donc **totalement interdit** dans cette application (volonté d'optimisation).

La solution permettant de contourner facilement cet obstacle consiste à envisager dans un nombre entier (8 bits, 16 bits ou 32bits) une virgule fictive qui sépare le nombre en deux parties, l'une étant la partie **entière**, l'autre la partie **fractionnaire**.

Notation : **E.F** (***E*** = entier, ***F***=fractionnaire), avec E+F = 8 ou 16 ou 32 selon la taille de la variable.

Exemple : *10.22* signifie que le codage est signé 32bits avec 10 bits représentant la partie entière, 22 bits pour la partie fractionnaire.

Le cosinus et le sinus vont être codés en format 1.15.

Voici quelques correspondances :

écriture hexadécimale → valeur décimale représentée en format 1.15 :

0x0001 → 0.000030517578125

0x7FFF → 0.999969482421875

0x4000 → 0.5

0x8000 → -1.0

0xFFFF → -0.000030517578125

2. Expliquer la logique de ce codage, c'est à dire comment on passe de la valeur hexadécimale à la valeur décimale fractionnaire. Pour vérifier si c'est compris, déterminer les correspondances suivantes au format **4.12** :
   0x02C1 → ?
   0xFE01 → ?

1.2.3. Opérations en format fractionnaire

L'addition :
Tout comme en base décimale, il faut un alignement des virgules pour pouvoir additionner. Autrement dit les opérandes sont supposées être au même format fractionnaire. Bien évidemment, l'opération réelle de l'UAL ignore totalement le présupposé des formats fractionnaires. A l'utilisateur d'être cohérent.

Exemple

0,5 (Format 3.5) + 1,5 (Format 3.5) = 2,0 (Format 3.5),

En effet,

0x0001 0000 + 0x0011 0000 = 0x0100 0000 (en réalité 16 + 48 = 64 en format 16.0)

Attention : L'opération d'addition peut déborder. A l'utilisateur d'y être vigilant !

La multiplication :
Là encore, on peut faire l'analogie avec la multiplication en base décimale que nous connaissons depuis toujours. Le mode opératoire consiste à :

- ignorer totalement la notion de virgule et à effectuer le produit entier classique,
- ajouter une virgule au résultat en prenant en compte la position des virgules de chacun des arguments

  → le format fractionnaire résultant sera donc nécessairement différent de ceux des opérandes

→ La multiplication peut hybrider les formats fractionnaires des opérandes (contrairement à l'addition),
→ Le résultat doit tenir dans un contenant égal à la somme de ceux des opérandes. Le programmeur doit y veiller. Dans ces conditions, l'opération de multiplication ne déborde jamais.

Le format fractionnaire de sortie est donc :
(E1.F1) * (E2.F2) → ([E1+E2].[F1+F2])

Exemple :
0,5 (Format 3.5) * 1,5 (Format 3.5) = 0,75 (Format 3+3.5+5, soit Format 6.10),

En effet,

opération brute : 0x0001 0000 * 0x0011 0000 = 0x0000 0011 0000 0000 (16*48 = 768)
ajout des virgules: 0x0001 0000 * 0x0011 0000 = 0x0000 0011 0000 0000 (0,5*1,5=0,75)
2^{-1} 2^{-2}

NB : Dans la suite du projet, les conversions hexadécimal vers décimal fractionnaire E.F (et inversement) pourront avantageusement être obtenus grâce à la petite application *html* de JL Noullet :

https://noullet-gei.gitlab.io/asm/BE/calc_fixed_point.html
(à placer en marque page, elle vous sera très utile ...)

2. Codage de la DFT

Pour des raisons de simplicité, on procédera à un calcul de la DFT en « non normalisé » (comme *Matlab*), c'est à dire que l'opération sera :

$$X(k)=\sum_{n=0}^{n=M-1} x(n).e^{-j\frac{2.\pi.k.n}{M}}$$ au lieu de ~~$X(k)=\frac{1}{M}.\sum_{n=0}^{n=M-1} x(n).e^{-j\frac{2.\pi.k.n}{M}}$~~

On demande de coder la fonction dont le prototype C est :
*int DFT_ModuleAuCarre(short int * Signal64ech, char k) ;*

paramètre de sortie :
- $|X(k)|^2$ sur 32 bits (cela évite de calculer la racine carrée, l'information est tout aussi pertinente pour notre application),

paramètres d'entrée :
- *short int * Signal64ech*, adresse du premier échantillon du signal à traiter 16 bits,
- *char k*, fréquence normalisée.

On procédera en trois temps :
- **première étape**, la fonction ne remontera que la **partie réelle** de X(k), dont on précisera le format fractionnaire,
- **seconde étape**, la fonction ne remontera que la **partie imaginaire** de X(k), même format fractionnaire que précédemment,
- **troisième étape**, la fonction remontera le module au carré tel que demandé, dont on précisera le nouveau format fractionnaire.

3. Déployer l'archive *PjtKEIL_StepDFT.zip*, ouvrir le projet et choisir la cible simulation.

4. Ajoutez les deux fichiers manquants qui se trouvent dans */Src* :
   - *DFT.s* (fichier dans lequel vous écrirez la fonction, et qui contient déjà les tables cosinus et sinus sur 64 éléments),
   - *Signal.asm* (qui contient le signal à traiter, à savoir un cosinus échantillonné 12 bits **pleine échelle** de l'ADC, fréquence normalisée k = 1)

5. Procéder au codage selon la progressivité demandée. A chaque étape, on établira un test exhaustif sur toutes les fréquences normalisées (64 points fréquentiels) de manière à bien vérifier la conformité du spectre en entier. La vérification doit être **qualitative et quantitative**. On pourra modifier le signal de test pour affiner les essais. Enfin les tests seront faits en C dans la fonction *main()* en appelant votre fonction sous test, *DFT_ModuleAuCarre*.

Aide pour les validations :

Procéder à un essai de validation nécessite de connaître le résultat attendu...

Concernant la troisième étape ... :

Pour un signal sinusoïdal de fréquence normalisée ***k****, et d'amplitude* ***A*** *, la DFT prise au point k donne un module |X(k)| égal à* $\frac{M \cdot A}{2}$ *(raie de rang k et par symétrie, raie de rang 64-k : |X(k)|=|X(64-k)|). La valeur attendue de la fonction lors de la dernière étape est donc sans surprise* $\boxed{\frac{M^2 \cdot A^2}{4}}$**.**

La difficulté ici, c'est que le résultat de la fonction devra être interprété, puisque le paramètre de sortie est un entier dans un format fractionnaire donné (qu'il vous appartient de définir...).

Concernant les première et seconde étapes … :

Lors de la première étape, le calcul à faire ne concerne que la partie réelle :

$$X_{Reel}(k) = \sum_{n=0}^{n=M-1} x(n).\cos(\frac{2.\pi.k.n}{M})$$

Lors de la seconde étape :

$$X_{Imag}(k) = \sum_{n=0}^{n=M-1} x(n).\sin(\frac{2.\pi.k.n}{M})$$ ***nb*** *: normalement il y a un signe '-' mais la mise au carré ultérieure annulera son influence...*

→ La partie ***réelle*** *correspond à la « teneur de la composante de rang k en terme de* ***cosinus*** *», alors que la partie* ***imaginaire*** *indique la « teneur de la composante de rang k en terme de* ***sinus****».*

Ainsi,
→ si l'on applique un signal échantillonné en ***cosinus pur****, la* ***partie réelle*** *sera* ***maximale*** *alors que la* ***partie imaginaire*** *sera* ***nulle****. Réciproquement si le signal est un sinus pur.*

→ Si l'on applique un signal ***cosinus pur*** *de rang k et d'amplitude A, nous aurons :*

$$|X(k)| = \sqrt{X_{Reel}^2(k) + X_{Imag}^2(k)} = \sqrt{X_{Reel}^2(k)} = X_{Reel}(k) = \boxed{\frac{M \cdot A}{2}}, \quad X_{Imag}(k) = 0 \quad .$$

→ Si l'on applique un signal ***sinus pur*** *de rang k et d'amplitude A, nous aurons :*

$$|X(k)| = \sqrt{X_{Reel}^2(k) + X_{Imag}^2(k)} = \sqrt{X_{Imag}^2(k)} = X_{Imag}(k) = \boxed{\frac{M \cdot A}{2}}, \quad X_{Reel}(k) = 0 \quad .$$

De cette manière, il est tout à fait possible de tester les deux premières étapes de la DFT, puisque les résultats sont connus, reste à appliquer les bons signaux de test (celui par défaut est un cosinus pur). On peut le modifier, notamment la phase avec le script Matlab disponible sur Moodle.