Electronique pour les systèmes embarqués

Implémentation d'un filtre numérique de décimation

1 Introduction

Dans ce TP, nous allons continuer à nous intéresser à la conception de la chaîne d'acquisition pour le bracelet connecté dont on a étudié les spécifications de son convertisseur analogique numérique (CAN) durant le dernier TP. Pour rappel, l'étude nous a permis de fixer la fréquence d'échantillonnage du CAN à 50 kHz, sa pleine échelle (PE) à 4 V et le nombre de bits à 14.
Dans ce TP, nous allons nous concentrer sur le filtrage numérique. Nous allons étudier le filtrage nécessaire pour le signal issu d'un des trois capteurs considérés, le microphone pour la reconaissance vocale. Pour rappel, la fréquence de sortie maximale de ce capteur est de 4.16 kHz avec une dynamique d'entrée de ± 5 mV à ± 1.5 V sur laquelle il faut garantir un SNDR de 40 dB. Ce signal sera perturbé par des signaux que nous modélisons par deux sinusoïdes d'amplitude 10 mV à 16 kHz et 22 kHz. [1] L'objectif du TP est d'étudier et dimensionner le filtrage numérique nécessaire pour passer du signal cadencé à 50 kHz à un signal décimé cadencé à 8.33 kHz avec un SNDR supérieur à 40 dB. Nous souhaitons également que l'ondulation à l'intérieur de la bande passante [0 4.16 kHz] soit inférieure à 0.5 dB. Le SNDR dans la bande utile à la sortie du CAN est de 41.8 dB pour une sinusoïde à 3.1 kHz d'amplitude 5 mV.

Question 1.1 *A quelles fréquences se replient les deux perturbateurs après décimation ? Charger et exécuter le fichier* `Filter_decimation.m` *et comparer les résultats de simulations à vos calculs.*

1. Sachez qu'en pratique les perturbateurs peuvent être aussi bien des raies parasites que des signaux plus riches en fréquence. La forme et l'amplitude de ces perturbateurs dépendent d'une multitude de paramètres tels que la nature du capteur, le type d'alimentation utilisée, l'environnement,...

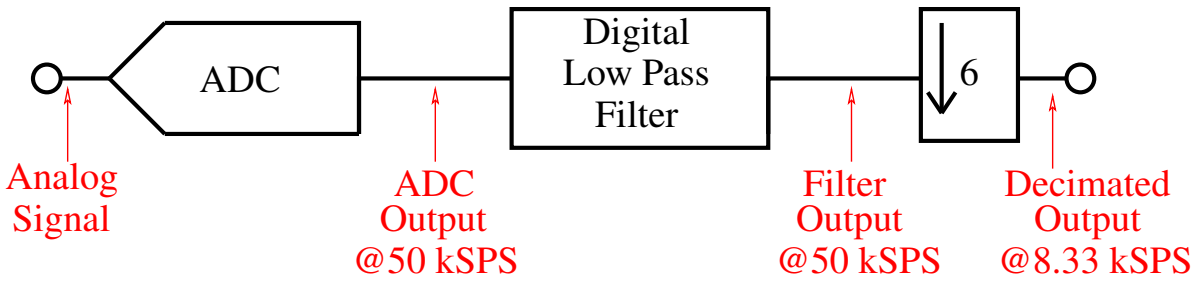

FIGURE 1 – Diagramme bloc du décimateur

Pour concevoir le filtre qui nous permettra de réaliser cette décimation tout en minimisant la complexité, nous allons procéder en trois étapes :

1. Calculer l'ordre et les coefficients du filtre. Nous allons concevoir un filtre FIR symétrique en utilisant l'approche de la transformée de Fourier inverse
2. Coder les coefficients en virgule fixe et déterminer le nombre de bits adéquat
3. Coder les coefficients en *canonical signed digit* (CSD)

Les deux premières étapes vont chacune causer une dégradation au SNDR. Nous allons ainsi attribuer une dégradation de 0.8 dB pour la première étape [2] et une dégradation de 1 dB pour la 2ème [3].

2 Ordre du filtre

Question 2.1 *Charger et exécuter le fichier* `Filter_order.m`. *Analyser les spectres avant filtrage, après filtrage et après décimation ainsi que la réponse fréquentielle du filtre.*

Question 2.2 *Déterminer en simulation l'ordre du filtre minimal* L_{min} *qui permet d'obtenir un SNDR supérieur à la valeur visée pour la première étape et une ondulation inférieure à* 0.5 dB.

Question 2.3 *Relever l'ordre du filtre et tracer sa réponse impulsionelle.*

Afin d'éviter de les recalculer dans la suite, sauvegarder les coefficients du filtre dans un fichier à l'aide de la commande `save -mat hn.mat hn`.

3 Codage en virgule fixe (quantification)

Nous allons à présent coder les coefficients du filtre en virgule fixe ou en d'autres termes nous allons quantifier les coefficients du filtre sur un nombre fini de bits.

Question 3.1 *Charger et exécuter le fichier* `Filter_NumOfBits.m`. *Comparer les réponses fréquentielles du filtre avant et après quantification pour différentes valeurs de* `nbitsfilter`.

Question 3.2 *Déterminer en simulation le nombre de bits minimal* $nbitsfilter_{min}$ *qui permet d'obtenir un SNDR supérieur à la valeur visée pour la deuxième étape.*

Afin d'éviter de les recalculer dans la suite, sauvegarder les coefficients quantifiés du filtre dans un fichier à l'aide de la commande `save -mat hn_quant.mat hn_quant`.

2. ce qui résultera en un SNDR de 41 dB
3. ce qui résultera en un SNDR de 40 dB

4 Calcul de la complexité

Afin de minimiser la complexité du filtre, nous allons l'implémenter à l'aide d'une architecture sans multiplieur. Nous allons en fait utiliser une approche avec des décalages et des sommes. Par exemple, pour implémenter une multiplication par 7, on pourrait faire un premier décalage de l'entrée du multiplieur de 2 bits vers la gauche pour implémenter une multiplication par 4, un deuxième décalage d'un bit pour implémenter une multiplication par 2 et ensuite sommer les deux versions décalées avec l'entrée originale. Avec l'approche CSD, la multiplication par 7 se fera par un décalage de 3 bits vers la gauche pour implémenter une multiplication par 8 suivi d'une soustraction de l'entrée originale de cette version décalée. Ceci permet de réduire la complexité de la multiplication.

Question 4.1 *Pour mieux comprendre le fonctionnement du codage en CSD, charger le fichier* `CSD_basic.m`*. Changer la valeur de in par exemple 7, 54, 127... Comparer les sorties binaires aux sorties CSD.*

Question 4.2 *Pour calculer la complexité du filtre conçu, nous allons calculer le nombre de bits non nuls dans les coefficients. Charger le fichier* `Filter_complexity.m`*, modifier la valeur de* `nbitsfilter` *à la valeur que vous avez retenue dans la section précédente. Relever le nombre de sommateur/soustracteur en binaire et CSD, comparer avec les résultats de vos collègues.*

Question 4.3 *Proposer une architecture multi-étage pour réaliser cette même opération de décimation. Quelles seraient les avantages et inconvénients d'une telle architecture comparée à l'architecture actuelle ?*