# Abréviations

CCS : Code Composer Studio

**Court récapitulatif**

**Codage en C**

# Implémentation

La solution d’implémentation présentée contient deux projets CSS (.cproj) contenu dans le répertoire *CSS-Projets*.

## Projet sur simulateur

Le projet dans le répertoire *Projet\_SYS835\_test* a été utilisé pour développer l’algorithme de suppression du bruit. Le répertoire *include* contient tous les fichiers d’en-têtes (.h) utilisé pour le projet. On y trouve une description détaillée de chacune des fonctions et variables développées. Nous invitons le lecteur à en prendre connaissance, puisque beaucoup d’informations pertinentes à la compréhension s’y retrouve. Le répertoire *source* contient l’implémentation des diverses fonctions décrites dans les fichiers d’en-têtes.

Afin de faciliter le développement, une cible du simulateur de la carte DSK6713 a été générée pouvoir lancer le code et ainsi vérifier le fonctionnement des fonctions implémentées. Le répertoire *data* contient le fichier de simulation *car.raw[[1]](#footnote-1)* (le même fichier qui a été utilisé pour le développement Matlab) qui est utilisé pour générer les données d’entrée pour la simulation. Ce fichier est l’enregistrement d’une conversation entre un homme(voix plus grave) et une femme(voix plus aigüe) avec la présence de bruits de voitures.

Dans le fichier *main.c*, on génère des échantillons de tailles fixes en lisant le fichier *car.raw*  et les traiter en appelant la fonction d’algorithme de suppression du bruit.(NoiseSuppressionAlgorithm).

## Projet sur la carte DSK6713

Le projet dans le répertoire *Projet\_XXXX* a été utilisé pour vérifier l’algorithme de suppression du bruit en temps réel. Le projet contient pratiquement la même arborescence et les mêmes fichiers que le projet *Projet\_SYS835\_test*. Nous y avons ajouté les fichiers *C6713dskinit.h*, *Vectors\_intr.asm* et *C6713dskinit.c* tirés du fichiers *Ex22b.zip*.[[2]](#footnote-2) Nous nous sommes inspirés de l’exemple *Ex22b.zip* comme point de départ pour l’implémentation en temps réel de notre solution.

Dans le fichier *main.c*, nous configurons le DSP afin d’utiliser les interruptions pour produire des trames de taille fixe et de déclencher le traitement par la fonction NoiseSuppressionAlgorithm. De plus nous avons créé une cible utilisée pour lancer notre implémentation sur la carte DSK6713 et permettre de vérifier notre solution en temps réel.

**Codage en ASM (filtre) (P)**

\_FilterAsm

**Compilation DSP (organisation code/projet?)**

Modifications à Ex22b

Configuration « Reléease »

**Tests en temps réel + résultats**

# Résultats

Clipping/bruit à plus de 8 ch (P)

## Simulation

À l’aide du simulateur en mode *debug*, nous avons mesuré les différents gains appliqués pour chacun des canaux en différentes circonstances, soit aucune voix n’est présente et lorsque la voix est présente. À noter que sur les graphiques l’axe des X correspond aux canaux et l’axe des Y aux valeurs de gains appliquées.

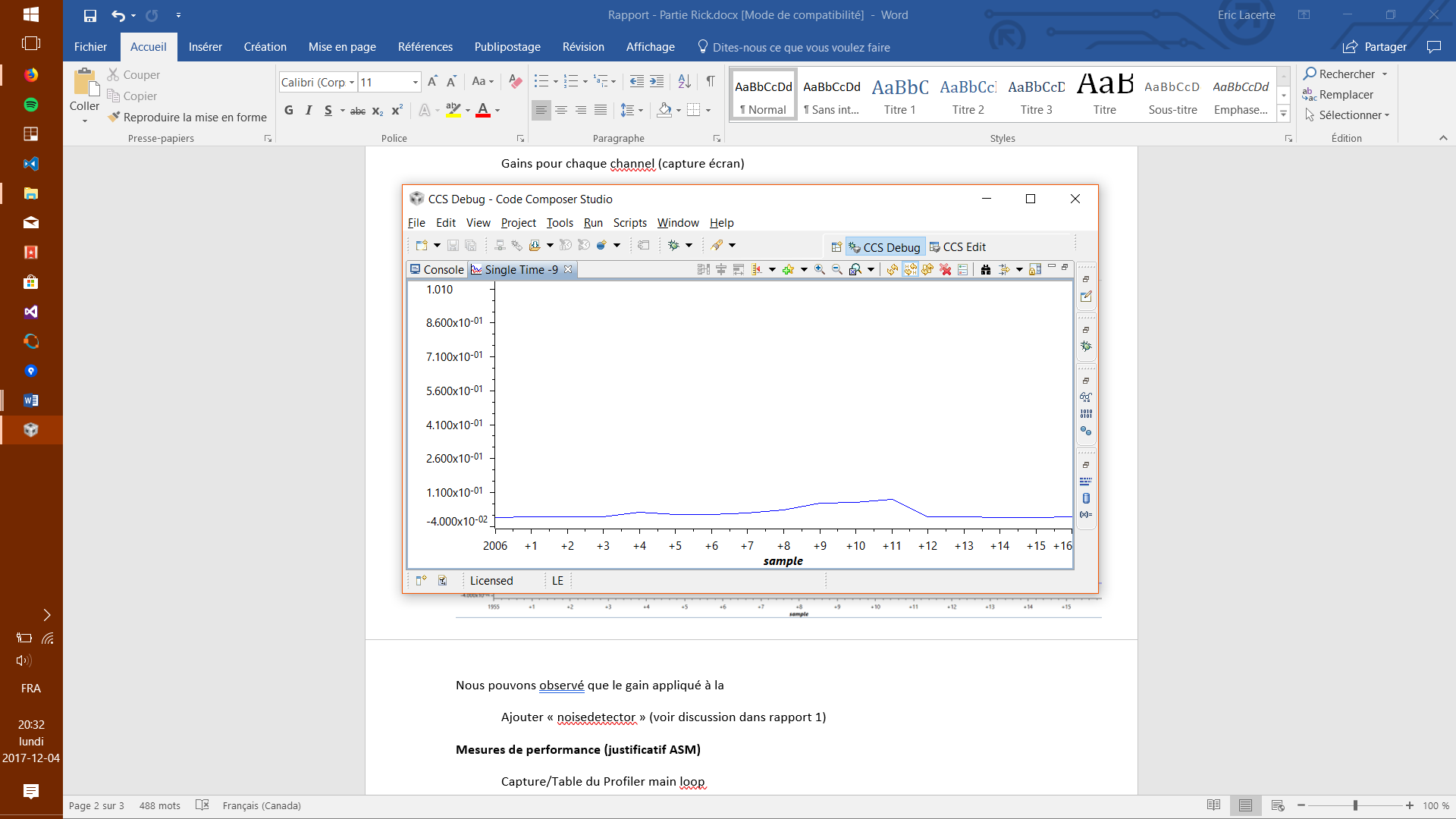


Figure 1 Valeurs de gains calculées par canaux lorsqu'aucune voix n'est présente

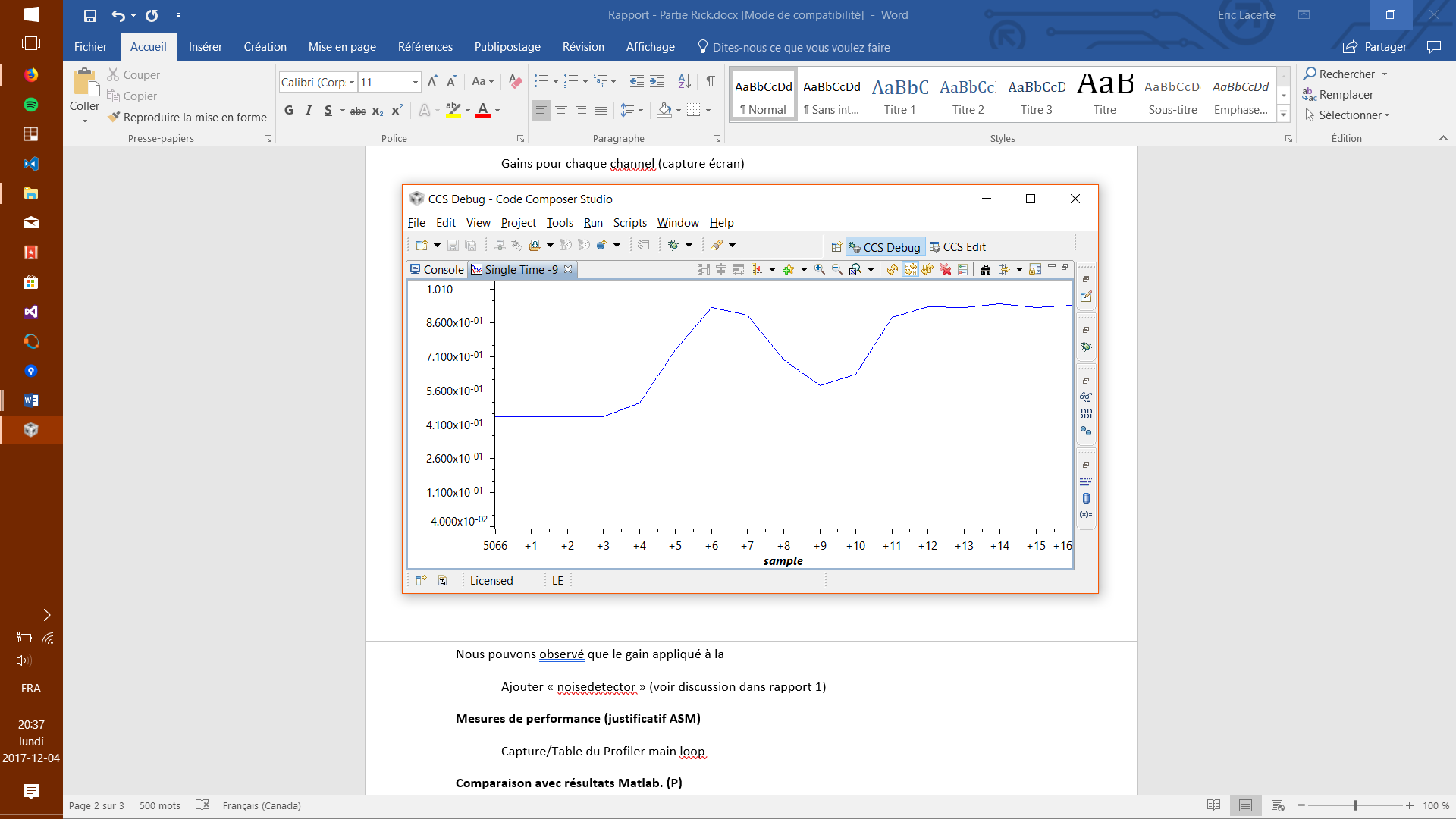


Figure 2 Valeurs de gains calculées par canaux lors de la présence de voix

Nous pouvons constater que le gain appliqué est trop faible lorsque la voix n’est pas présente et que la valeur du gain augmente lorsqu’il détecte la voix. L’algorithme adapte le gain afin d’affaiblir les trames de bruits.

En comparant les gains appliqués entre deux trames consécutives, nous pouvons également observer que la variation du gain appliqué se fait tout en douceur.

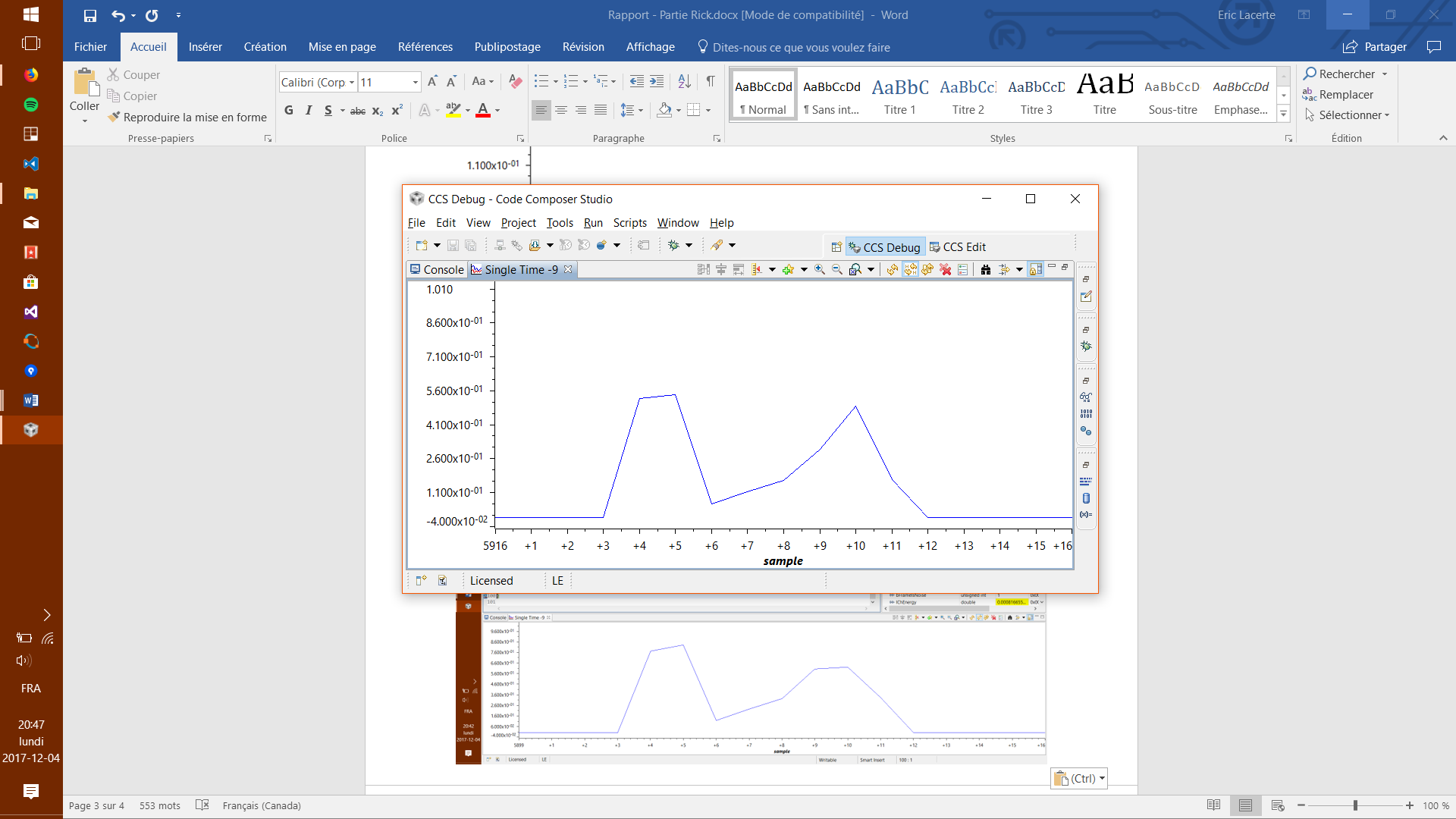


Figure 3 Valeurs de gains appliquées par canaux à la trame n.

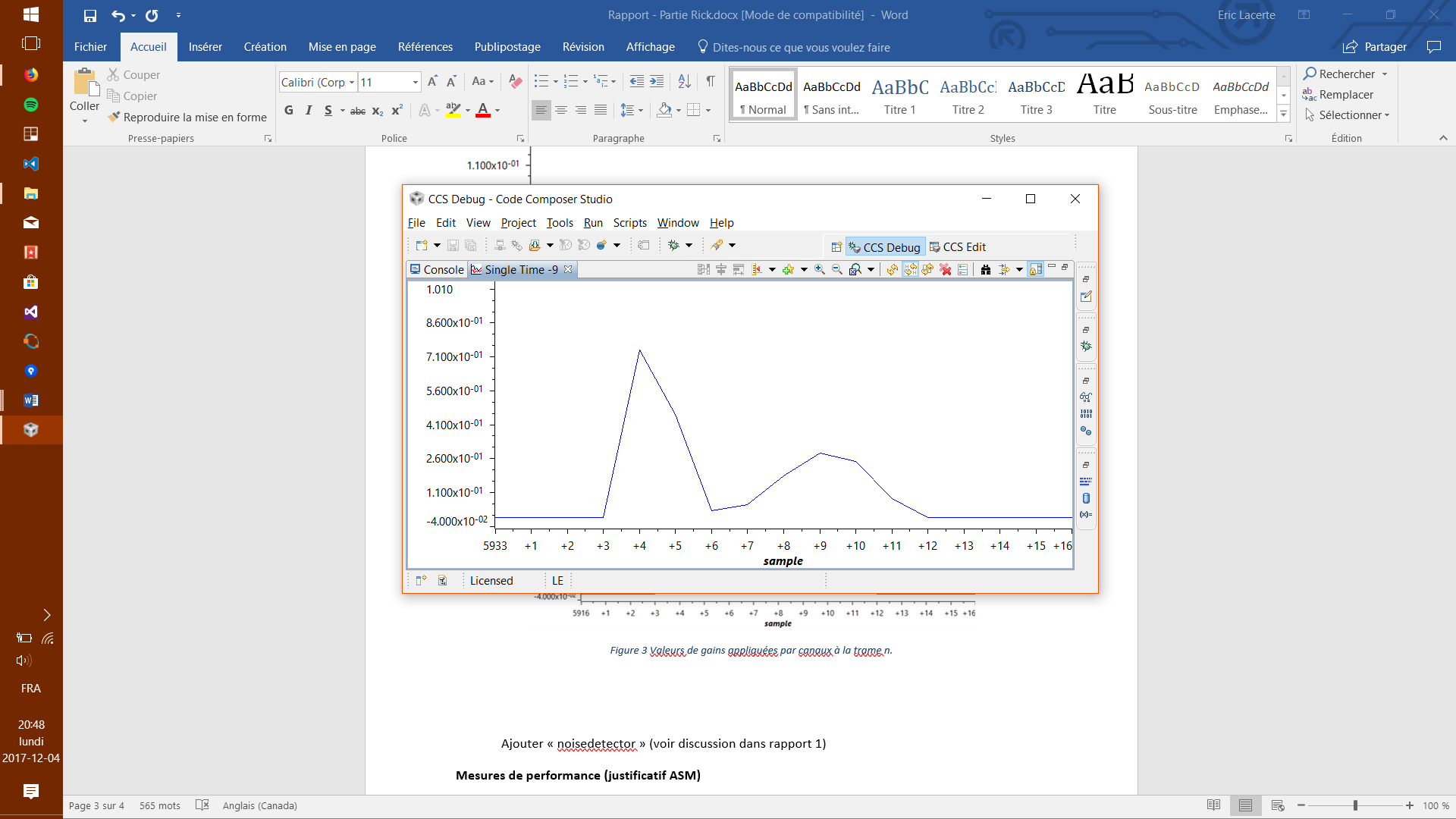


Figure 4 Valeurs de gains appliquées par canaux à la trame n + 1.

Nous avons cependant éprouvé quelques difficultés au niveau du seuil de détection du bruit. Tel que mentionné dans le précédent rapport[[3]](#footnote-3), la mise à l’échelle de l’énergie ne semble pas être adéquate afin d’utiliser l’algorithme du détecteur de bruit (NoiseDetector). Nous avons donc opté pour un seuil de détection du bruit fixe (UINT16\_MAX). En utilisant cette méthode nous pouvons discerner adéquatement les trames contenant seulement du bruit.

## Performances

À l’aide de l’outil de profilage de CCS, nous avons mesurer les nombres de cycles requis pour chacune des fonctions de l’algorithme.

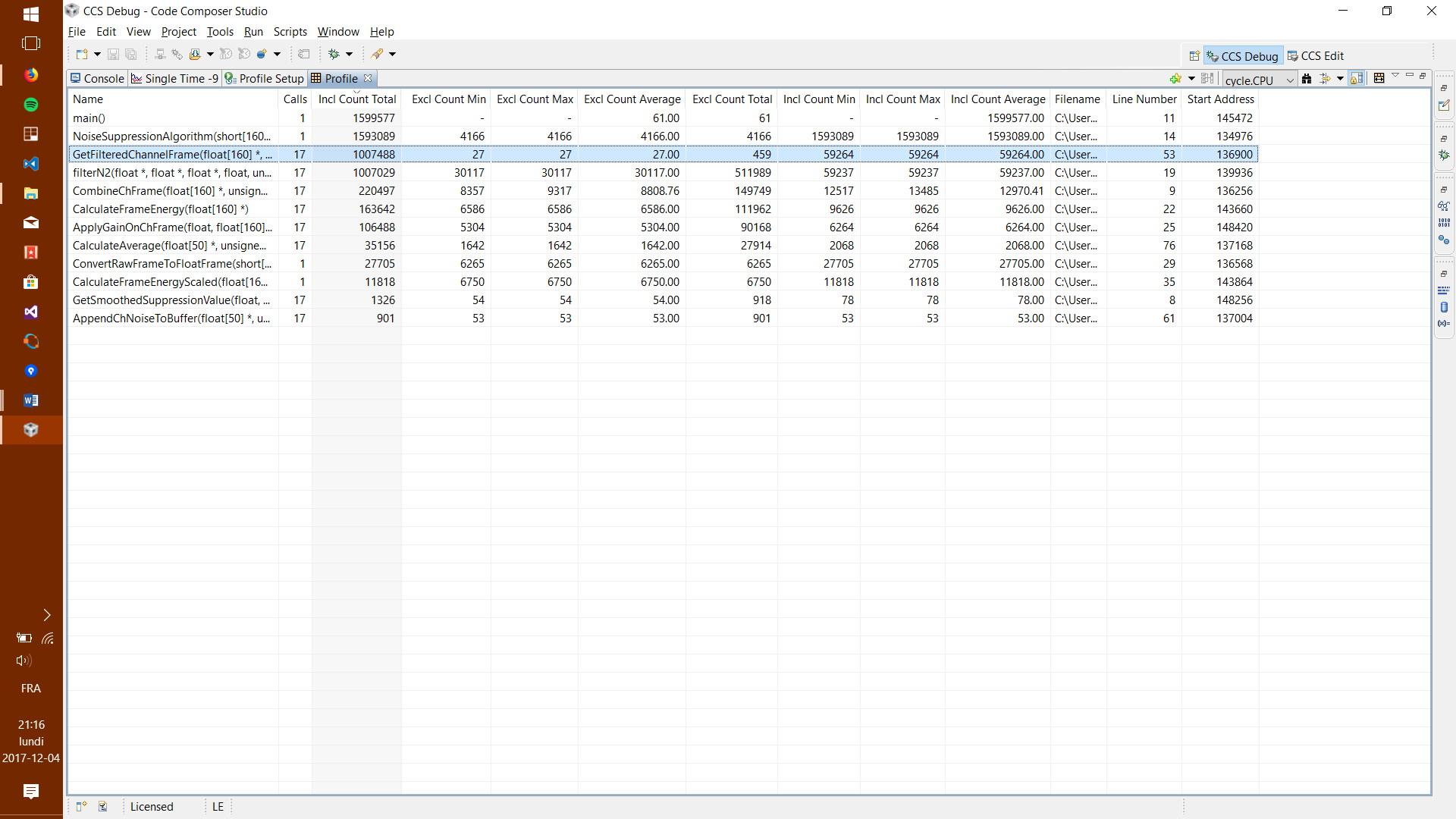


Figure 5 Nombres de cycles requis par fonctions

Nous pouvons constaté que la fonction *GetFilteredChannelFrame*  (qui fait appel à la fonction *filterN2*) est la fonctions la plus coûteuse en terme de ressources. Ceci justifie l’optimisation de cette fonction *filterN2* en assembleur. Par contrainte de temps, nous avons écrit cette fonction mais elle n’est pas pu être intégrée dans le projet final.

**Comparaison avec résultats Matlab. (P)**

Sim vs Matlab : **Accumulation d’erreurs?**

Detection bruit : Matlab 50% frame == bruit vs DSP : 90% : Problème d’échelle?

C/Release

"filterN2(float \*, float \*, float \*, float, unsigned int)",17,**5495**,5584,5502.59,93544,24055,24199,24065.82,409119,C:\ets\sys835\NoiseReduction\Release\..\filter.c,19,7776

ASM

Parallel

1. Site de signalogic <http://www.signalogic.com/index.pl?page=codec_samples> [↑](#footnote-ref-1)
2. Site du cours SYS835, https://ena.etsmtl.ca/mod/folder/view.php?id=316910 [↑](#footnote-ref-2)
3. ERIC LACERTE et PHILIPPE LAVOIE, Rapport première étape de projet. [↑](#footnote-ref-3)