VILLE DE LIÈGE

Institut de Technologie
Enseignement de Promotion sociale

Année académique 2021 – 2022

Développement d'un codec audio AAC : optimisation de l'algorithme MDCT pour l'architecture ARM

Étudiante :
Laura Binacchi

Lieu de stage :
EVS Broadcast Equipment
Rue du Bois Saint-Jean 13, 4102 Ougrée

Maître de stage :
Bernard Thilmant
Software Engineer

Épreuve intégrée présentée pour l'obtention du diplôme de
BACHELIER.E EN INFORMATIQUE ET SYSTÈMES
FINALITÉ : INFORMATIQUE INDUSTRIELLE

Table des matières

Remerciements

Introduction

Développement d'une solution de software embarqué sur processeur ARM pour encodage audio AAC optimisé aux applications d'EVS :

- Prise de connaissance de l'encodage AAC et de l'environnement EVS qui utilise ce type de format ;
- Prise de connaissance des résultats des optimisations possibles du modèle psycho-acoustique développé par EVS ;
- Développement du code en C ou Assembler pour l'encodage AAC sur plateforme ARM ;
- Test du système et documentation de son implémentation.s possibles du modèle psycho-acoustique développé par EVS ;
- Développement du code en C ou Assembler pour l'encodage AAC sur plateforme ARM ;
- Test du système et documentation de son implémentation.

Ce travail commencera par une présentation d'EVS et du département dans lequel s'est déroulé mon stage. Parmi les nombreux produits d'EVS, seul le serveur XT sera brièvement présenté puisque c'est spécifiquement pour ce dernier que le codec AAC est développé et optimisé.

Quelques notions théoriques indispensables à la compréhension du travail pratique seront ensuite développées avec une section consacrée au son et sa numérisation et une autre consacrée aux codecs MPEG, à leur fonctionnement et en particulier au fonctionnement du bloc MDCT de l'encodeur AAC.

1 EVS Broadcast Equipment

1.1 Présentation d'EVS et du département R&D

Mon stage s'est déroulé au sein de la société EVS Broadcast Equipment dont la figure 1 représente le logo. EVS est une entreprise d'origine liégeoise devenue internationale. Fondée en 1994 par Pierre L'Hoest, Laurent Minguet et Michel Counson, EVS compte aujourd'hui plus de 600 employés dans plus de 20 bureaux à travers le monde mais son siège principal se situe toujours à Liège.

Figure 1 – Logo de la société EVS Broadcast Equipment[1]

EVS est devenu leader dans le monde du broadcast avec ses serveurs permettant l'accès et la diffusion instantanée des données audiovisuelles enregistrées sur ses serveurs. L'entreprise est également célèbre pour ses ralentis instantanés. Ces technologies sont utilisées pour la production live des plus importants événements sportifs dans le monde : le matériel EVS est notamment utilisé pour la retransmission des Jeux Olympiques depuis 1998.

Plus de 50% des employés d'EVS travaillent en recherche et développement afin de répondre au marché du broadcast en constante évolution. Outre ses solutions techniques innovantes, EVS se différencie de ses concurrents par la proximité entretenue avec les clients en leur proposant des solutions à l'écoute de leurs besoins et en leur offrant un service de support de qualité.

C'est en R&D, dans l'équipe Hardware-Firmware, que s'est déroulé mon stage. Sous la direction de Justin Mannesberg, cette équipe se compose d'une vingtaine d'employés spécialisés en développement embarqué et en développement FPGA. La situation particulière dans laquelle s'est déroulé mon stage, en pleine pandémie de Covid et alors que tous les employés étaient confinés, ne m'a pas permis d'interagir avec beaucoup de membres de l'équipe et ni de pouvoir observer leur travail. Bernard Thilmant (Software Engineer dans l'équipe Hardware-Firmare) a cependant réussi à m'apporter le soutien nécessaire à la bonne réalisation de mon stage : il m'a permis de m'initier au C++, m'a aidée à ne pas me perdre dans les concepts parfois complexes de l'encodage audio et m'a aidée à apporter la rigueur scientifique nécessaire à la réalisation de mon travail. J'ai également pu bénéficier de l'expertise technique de Frédéric Lefranc (Principal Embedded System Architect dans l'équipe Hardware-Firmare) ainsi que du suivi de Justin Mannesberg (Manager de l'équipe Hardware-Firmware).

1.2 Le serveur XT

EVS développe et commercialise de nombreux produits allant des serveurs de production aux interfaces permettant d'exploiter des données audio-visuelles ou de monitorer des systèmes de production[2]. Le serveur de production live XT est un des produits emblématiques d'EVS. Il permet de stocker de grandes quantités de données audio-visuelles et d'y accéder en temps réel afin de répondre aux besoins de la production en live. Par exemple, la remote LSM (*Live Slow Motion* permet d'accéder aux contenus des serveurs XT afin de créer les ralentis pour lesquels EVS est célèbre dans le monde.

FIGURE 2 – Vues avant et arrière (en configuration IP) de l'XT-VIA[2]

Le serveur XT a connu plusieurs versions : XT, XT2, XT2+, XT3 et enfin l'XT-VIA. L'XT-VIA (cf figure 2), la plus récente version du serveur XT, en quelques informations clés[2] :

- offre un espace de stockage de 18 à 54 TB, soit plus de 130h d'enregistrement en UHD-4K ;
- dispose de 2 à plus de 16 canaux selon le format choisi : 2 canaux en UHD-8K (4320p), 6 canaux en UHD-4K (2160p) et plus de 16 canaux en FHD and HD (720p, 1080i, 1080p) ;
- permet une configuration hybride de ses entrées et sorties en IP (10G Ethernet SFP+, 100G en option, ST2022-6, ST2022-7, ST2022-8, ST2110, NMOS IS-04, IS-05, EMBER+, PTP) ou SDI (1.5G-SDI, 3G-SDI et 12G-SDI) ;
- supporte de nombreux formats d'encodage vidéo : UHD-4K (XAVC-Intra et DNxHR), HD/FHD (XAVC-I, AVC-I, DNxHD et ProRes), PROXY (MJPEG et H264) ;
- peut enregistrer 192 audio tracks non compressés et supporte les standards AES et MADI ;
- offre de nombreuses possibilités de connection avec du matériel EVS ou non.

C'est pour la dernière génération du serveur XT, l'XT-VIA, que le codec AAC est développé. La compression avec perte de données de ce codec permet d'optimiser l'espace occupé par les données audio sans en altérer la qualité perçue. Outre la qualité audio, les performances de l'encodage sont importantes à prendre en compte pour permettre l'enregistrement de plusieurs canaux en parallèle tout en conservant un traitement de l'information qui tienne le temps réel. L'optimisation des performances doit tenir compte de l'architecture de l'XT-VIA : l'architecture ARM Neon remplace l'architecture Intel x86 de ses prédécesseurs avec des différences importantes dans les fonctions intrinsèques.

2 L'encodage audionumérique : généralités

2.1 Le son

2.2 La numérisation d'un signal

3 Les codecs audio

3.1 Définition d'un codec

Un codec est un procédé logiciel composé d'un encodeur (***co**der*) et d'un décodeur (***dec**oder*)[3]. Un codec audio permet donc, d'une part, de coder un signal audio dans un flux de donnée numériques et, d'autre part, de décoder ces données afin de restituer le signal audio.

Les codecs sont dits avec perte (*lossy*) ou sans perte (*lossless*). Le PCM est par exemple un codec sans perte puisqu'il encode la totalité des informations sonores dans la bande de fréquences humainement audible. Ce type de codec permet de conserver la qualité de l'audio mais nécessite en contrepartie un espace de stockage conséquent, même avec une compression des données.

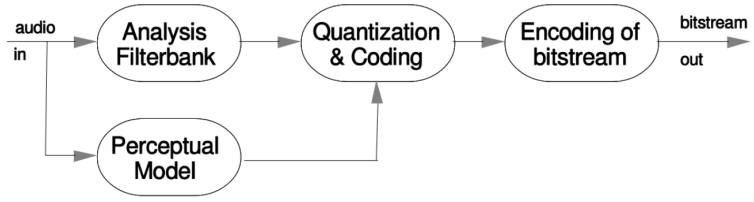
Afin de réduire l'espace de stockage nécessaire, les codecs avec perte permettent de supprimer une partie des données audio. C'est le cas des codecs définis par les normes MPEG dont fait partie le codec AAC.

3.2 Les codecs MPEG

MPEG (*Moving Picture Experts Group*) désigne une alliance de différents groupes de travail définissant des normes d'encodage, de compression et décompression et de transmission de média audio, vidéo et graphiques[4]. Le groupe est actif depuis 1988 et a produit depuis de nombreuses normes.

Les codecs audio qui implémentent les normes MPEG ont pour point commun d'être des codecs avec perte de données basés sur un modèle psychoacoustique. Le premier est le MP3, défini par la norme MPEG-1 Layer-3 ISO/IEC 11172-3 :1993. Le codec AAC est conçu en 1997 pour remplacer le MP3. Il est défini par les normes MPEG-2 partie 7 ISO/IEC 13818-7 :2006[5] et MPEG-4 partie 3 ISO/IEC 14496-3 :2019[6].

Les normes MPEG définissent les grandes lignes de l'encodage et du décodage et le format du conteneur mais pas l'implémentation du codec qui peut de ce fait être plus ou moins performant. Les codecs MPEG sont typiquement composés des blocs suivants :



(a) Schéma fonctionnel de l'encodeur

(b) Schéma fonctionnel du décodeur

Figure 3 – Vue simplifiée d'un codec MPEG basé sur un modèle psychoacoustique[7]

L'encodeur est composé des blocs suivants :

filter bank la banque de filtres décompose le signal temporel d'entrée en différentes composantes fréquentielles

perceptual model le modèle psychoacoustique utilise le signal temporel et/ou sa décomposition fréquentielle pour éliminer les données audio dont l'absence ne nuira pas à la qualité perçue à l'écoute ()

quantization and coding la quantification attribue une valeur numérique aux données du spectre de fréquences : elle sont typiquement codées avec une méthode entropique qui peut être optimisée avec le modèle psychoacoustique

encoding of bitstream les données sont formatées en un flux contenant typiquement le spectre de fréquences codé et des informations supplémentaires permettant l'encodage

Le décodeur a un fonctionnement inverse : le flux de données est décodé (**decoding of bitstream**), les composantes fréquentielles du signal sont retrouvées par l'opération inverse à la quantification (**inverse quantization**) et ces sous-bandes fréquentielles sont finalement rassemblées pour reconstituer le signal temporel (**synthesis filter bank**).

Le fonctionnement du décodeur ne sera pas plus développé dans ce travail car le bloc MDCT fait partie de la banque de filtres de l'encodeur. Le fonctionnement spécifique de l'encodeur AAC sera par contre détaillé dans la section 4.

3.3 Les modèles psychoacoustiques

Le section précédente a défini les codecs MPEG comme étant basés sur un modèle psychoacoustique. La psychoacoustique est une branche de la psychophysique qui étudie la manière dont l'oreille humaine perçoit le son[8]. Cette discipline permet d'améliorer la compression d'un signal audio en éliminant les sons qui sont captés par un microphone mais qui ne peuvent pas être perçus par l'oreille humaine et les avancées dans cette discipline permettent de développer des encodeurs audio de plus en plus performants. Les codecs basés sur un modèle psychoacoustique sont toujours des codecs avec perte puisqu'une partie des informations auditives sera définitivement perdue, ce qui ne nuit pour autant pas à la qualité perçue du son.

L'encodage audionumérique tient déjà compte des seuils de fréquences humainement audibles pour limiter les données audio enregistrées : nous l'avons vu dans la section 2.2, aucun son n'est perçu en-deça de 20Hz ou au-delà de 20kHz. La psychoacoustique permet de mieux dessiner limite entre ce qui est humainement audible ou non afin d'éliminer un maximum des informations non pertinentes et ainsi augmenter le facteur de compression des données : le facteur de compression des codecs MPEG est environ 15 fois supérieur à celui du CD[9].

Les effets de masque sont au centre des différents modèles psychoacoustiques utilisés pour la compression audio. L'enjeu afin d'obtenir le meilleur taux de compression est de calculer le plus finement possible les seuils de masquage, i.e. la limite entre les informations pertinentes et celles qui peuvent être éliminées. Les effets de masques dans le domaine fréquentiel (*spectral masking effects*) sont parmi les plus utilisés mais il en existe d'autres, e.g. dans le domaine temporel. La figure suivante représente différents effets de masque du domaine fréquentiel :

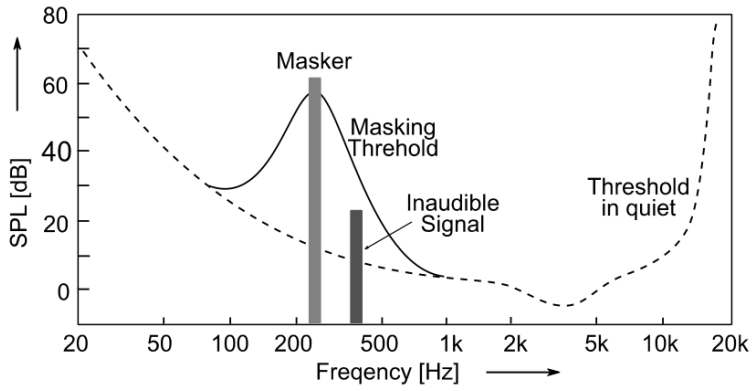


FIGURE 4 – Effets de masque dans le domaine fréquentiel[9]

Les lignes représentent le seuil

Threshold in quiet la ligne en pointillé représente le seuil d'audibilité dans le calme, indépendamment de tout autre élément qui pourrait interférer

Masking threshold

Le calcul du seuil de masquage tient compte[9] :

- des effets de masquage monophonique de la perception non-linéaire des fréquences, plus fine pour les basses fréquences ;
- des effets de masque dans le temps ;
- de l'effet de masque dans une bande de fréquence ou entre bandes de fréquences ;
- l'impact de la tonalité sur le masquage
- masking over time

4 Le codec AAC

4.1 Fonctionnement de l'encodeur AAC

Le codec AAC (Advanced Audio coding) est définit , défini par la norme . Les recherches en psychoacoustique ont permis de développer un algorithme d'encodage plus performant pour l'AAC que pour le MP3 : il permet d'encoder moins données audio tout en gardant la même qualité perçue au décodage[7].

4.2 Le bloc MDCT

5 Algorithmes MDCT de référence

5.1 Description mathématique

La transformation effectuée par la MDCT est donnée par l'équation suivante[10] :

$$X_k = \frac{2}{\sqrt{2N}} \sum_{n=0}^{2N-1} x_n \cos\left[\frac{\pi}{N}\left(N + \frac{1}{2} + \frac{N}{2}\right)\left(k + \frac{1}{2}\right)\right]$$

X_k avec $k \in [0, N[$ pour une fenêtre d'entrée de 2N échantillons

x_n avec $n \in [0, 2N[$: la fenêtre d'entrée

$F : R^{2N} \rightarrow R^N$ la MDCT est une fonction linaire qui pour 2N nombres réels d'entrée produit N nombre réels en sortie

La MDCT a été implémentée avec une fenêtre d'entrée de $2N = 1024$ échantillons. Le bloc de sortie, i.e. le spectre de fréquences de la fenêtre d'entrée, aura donc une taille de 512. Ces différentes valeurs, utilisées à de très nombreux endroits du code sont rassemblées dans le header `mdct_constants.h` présenté dans l'annexe A.

La section suivante présente deux implémentations simples de cette formule. Ces implémentations ne pourraient pas être utilisées sans avoir été optimisées car elles seraient beaucoup trop lentes pour un codec qui doit tenir le temps réel. La complexité d'une implémentation de cette formule est de $O(N^2)$ opérations (où N est la taille de la fenêtre d'entrée). Cette complexité peut être ramenée à $O(N \log N)$ opérations par une factorisation récursive. La complexité peut également être diminuée en se basant sur une autre transformation, e.g. une DFT (*Discrete Fourier Transorm*) ou une autre DCT (*Discrete Cosine Transform*) : la complexité sera alors de $O(N)$ opérations de pré et post processing en plus de la complexité de la DFT ou de la DCT choisie[10]. C'est cette dernière option qui a été retenue pour ce travail.

5.2 Algorithmes développé et utilisation

La formule mathématique de la MDCT donnée à la section 5.1 a été implémentée très simplement en algorithmique flottante avec la possibilité d'obtenir le spectre de fréquences codés en *float*, *double* ou *int32* depuis une fenêtre d'entrée en *int16*. L'objectif de cet implémentation est de pouvoir vérifier les données des implémentations optimisées de la MDCT et de mesurer leur précision.

La première implémentation de l'équation de la MDCT est présentée dans l'annexe B. La fonction développée permet de faire ses calculs et d'obtenir un résultat aussi bien en *float*) (32 bits) qu'en *double* (64 bits) grâce à l'utilisation d'un *template*.

La seconde fonction de référence est présentée dans l'annexe C. Elle permettra de vérifier les résultats des implémentations optimisées en *fixed point*. Tous les calculs ne sont pas fait en algorithmique entière : la fonction fait les mêmes calculs que la fonction de référence en algorithmique flottante (uniquement en *double* cette fois garder le plus de précision possible) et cast le résultat final dans un integer de 32 bits qui correspond à une notation Qx.15 signée.

5.3 Validation

Validation avec un code d'exemple qui correspond à un signal sinusoidal (single tone) -> annexes : génération d'un signal single tone + code qui sort les données.

Présentation des résultats sous forme de données brutes ou de graphique.

Explication de la lecture des résultats, calcul des bandes de fréquences représentées. Mise en évidence qu'on a bien une seule composante fréquentielle comme attendu pour un signal single tone

+ les résultats auraient été plus précis avec la fonction de fenêtre -> voir améliorations possibles

6 Algorithme MDCT basé sur la FFT

6.1 Optimisations attentues

Choix de l'optimisation de la MDCT basé sur sur une DCT : la FFT -> complexité de O (N log N) (N taille de la fenetre) + O(N) opérations de pré et de post processing.

But : utiliser une FFT déjà optimisée pour n'avoir à optimiser "que" les opérations de pré et de post processing Exemple trouvé sur le site DSP related -> Annexe ? Citation ?

6.2 Implémentation de la MDCT basée sur la FFT de la librairie FFTW3

Code développé à partir de cet exemple en annexe. Il est basé sur la librairie FFTW3 qu'on ne peut pas garder car ne permet pas de travailler en integer -> sera amené à être retravaillé.

Explications des paramètres du code :

- fenêtre de 1024
- pre-twiddle -> 256
- FFT => FFT avec une fenêtre d'entrée réduite et donc une complexité réduite
- post-twiddle -> 512

6.3 Validation

Validé avec un single tone signal + l'algo de référence.

Code d'exemple qui teste l'algo de référence et l'algo FFTW3 avec les mêmes données d'entrée pour comparaison. (on fait la différence pour la précision ? ce n'est peut être pas pertinent de la faire déjà)

Présentation des résultats sous forme de données brutes ou de graphiques.

7 Utilisation de la librairie Ne10

7.1 Choix de la librairie

Changement obligatoire car la FFT de FFTW3 est uniquement dispo en algo flottante et on veut une implémentation en fixed point.

Librairie optimisée pour l'architecture ARMv8. Mais a aussi des implémentation plain C (intéressant car permet de mesurer la diff de perf entre les 2).

7.2 Implémentation de la MDCT basée sur la FFT Ne10 en arithmétique *floating point*

Explications de l'algo en annexe

8 Algorithme MDCT en arithmétique fixed point

8.1 Arithmétique fixed point

trivial en float (reprendre le code de dsp) mais pour l'implémentation en arithmétique entière, attention à mettre dans les bonnes ranges

8.2 Améliorations attendues

Le code de DSP related est en floating point -> passage en fixed point : opérations sur des entiers plus performantes + meilleure intégration (éviter le passage artificiel de l'entier au flottant et inversement)

8.3 Implémentation de la MDCT basée sur la FFT Ne10 en arithmétique *fixed point*

Explication du code en annexe en accord avec les explications de la première subsection sur le fixed point

8.4 Performances

Explication du code en annexe qui permet de mesurer les performance (avec code Timer)

Démonstration des résultats

Explication des résultats pas attendus : à ce niveau, résultats pas intéressants car + d'opérations qu'en flottant et ARM a des fonctions d'algo flottante

Pour soutenir cette hypothèse : Test de la librairie et des performances des différentes FFT : performances équivalentes en 32 bits (floating ou fixed) et deux fois plus rapides en 16 bits

8.5 Arithmétique fixed point

9 Optimisations à l'architecture ARM

9.1 Spécificités de l'architecture ARMv8

Globalement reprend les tutos ARM

9.2 Utilisation des fonctions Neon SIMD (intrinsic)

Les opérations SIMD permettent de faire plusieurs opérations en une fois où l'algo normal n'en fait qu'une à la fois. L'algo SIMD permet de faire plusieurs modifications à la fois -> il faut ranger les données de manière à pouvoir l'appliquer facilement (fonction fenêtre -> tri des données ?)

Plus les données sur lesquelles on travaille sont petites, plus on peut faire d'opérations en parallèle -> il faut voir si la perte de précision est ok. -> mettre une représentation graphique, c'est beaucoup plus simple à comprendre. D'où l'intérêt de passer à du 16 bits, plutôt que de rester en 32 bits et il faut absolument éviter le 64 bits (aucun intérêt).

10 Résultats

10.1 Protocole de validation

10.2 Gain en performance

10.3 Perte de précision

11 Améliorations possibles

- Fonction fenêtre intégrée aux opérations de pre twiddling
- Quantification intégrée au post twiddling

Conclusion

Références

[1] Wikipedia, "Evs broadcast equipment." [https://evs.com], consulté le 21 avril 2022.

[2] Wikipedia, "Page de présentation des produits commercialisés par evs broadcast equipment." [https://evs.com/products/live-production-servers/xt-via], consulté le 21 avril 2022.

[3] Wikipedia, "Codec." [https://en.wikipedia.org/wiki/Codec], consulté le 2 mai 2022.

[4] Wikipedia, "Moving picture experts group." [https://en.wikipedia.org/wiki/Moving_Picture_Experts_Group], consulté le 30 mars 2022.

[5] "Information technology – Generic coding of moving pictures and associated audio information – Part 7 : Advanced Audio Coding (AAC)," standard, International Organization for Standardization, 2006. [https://www.iso.org/standard/43345.html].

[6] "Information technology – Coding of audio-visual objects – Part 3 : Audio," standard, International Organization for Standardization, 2019. [https://www.iso.org/standard/76383.html].

[7] K. Brandenburg, "Mp3 and aac explained," *AES 17th International Conference on High Quality Audio Coding*, 1999.

[8] Wikipedia, "Psychoacoustics." [https://en.wikipedia.org/wiki/Psychoacoustics], consulté le 2 mars 2022.

[9] J. Herre and S. Dick, "Psychoacoustic models for perceptual audio coding—a tutorial review," *Applied Sciences*, vol. 9, p. 2854, 07 2019.

[10] Wikipedia, "Modified discrete cosine transform." [https://en.wikipedia.org/wiki/Modified_discrete_cosine_transform], consulté le 17 septembre 2021.

Annexes

A Valeurs constantes utilisées dans le code

```
// Sampling frequency: 48kHz
#define FS                  48000

// Window length and derived constants
#define MDCT_WINDON_LEN     1024
#define MDCT_M              (MDCT_WINDON_LEN>>1)    // spectrum size
#define MDCT_M2             (MDCT_WINDON_LEN>>2)    // fft size
#define MDCT_M4             (MDCT_WINDON_LEN>>3)
#define MDCT_M32            (3*(MDCT_WINDON_LEN>>2))
#define MDCT_M52            (5*(MDCT_WINDON_LEN>>2))
```

B Implémentation de référence de la MDCT en algorithmique flottante

```
template<typename FLOAT>
void ref_float_mdct(FLOAT *time_signal, FLOAT *spectrum)
{
    FLOAT scale = 2.0 / sqrt(MDCT_WINDON_LEN);
    FLOAT factor1 = 2.0 * M_PI / static_cast<FLOAT>(MDCT_WINDON_LEN);
    FLOAT factor2 = 0.5 + static_cast<FLOAT>(MDCT_M2);
    for (int k = 0; k < MDCT_M; ++k)
    {
        FLOAT result = 0.0;
        FLOAT factor3 = (k + 0.5) * factor1;
        for (int n = 0; n < MDCT_WINDON_LEN; ++n)
        {
            result += time_signal[n] * cos((static_cast<FLOAT>(n) + factor2) * factor3);
        }
        spectrum[k] = scale * result;
    }
}
```

C Implémentation de référence de la MDCT en algorithmique entière

```
void ref_int_mdct(int16_t *time_signal, int32_t *spectrum)
{
    double scale = sqrt(MDCT_WINDON_LEN) / 2;
    double factor1 = 2.0 * M_PI / MDCT_WINDON_LEN;
    double factor2 = 0.5 + MDCT_M2;
    for (int k = 0; k < MDCT_M; ++k)
    {
        double result = 0.0;
        double factor3 = (k + 0.5) * factor1;
        for (int n = 0; n < MDCT_WINDON_LEN; ++n)
        {
```

```
            result += time_signal[n] * cos((n + factor2) * factor3);
        }
        assert(round(result*scale) == static_cast<int32_t>(round(result*scale)));
        spectrum[k] = static_cast<int32_t>(round(result/scale));
    }
}
```