

Rapport mini-projet Audio

Real-time digital audio signal processing with a STM32F746 Discovery board

Pierre CHOUTEAU & Elisa DELHOMME, 3SIA

23 janvier 2022

1 Introduction

L'objectif de ce mini-projet est de comprendre comment fonctionne la chaîne d'acquisition, de traitement et de restitution du signal audio en temps réel sur la carte STM32F746-Discovery. Ce travail permet à la fois de mettre en œuvre une analyse du son ambiant (via la mise en place d'un spectrogramme en temps réel) mais également la réalisation d'effets sur ce dernier.

Présentation de la carte utilisée

Pour ce projet, nous utilisons la carte STM32F746NG-Discovery. Celle-ci dispose de plusieurs éléments indispensables :

- Un Codec Audio WM8994ECS/R de CIRRUS avec 4 DACS (Digital to Analog Converter) et 2 ADC (Analog to Digital Converter) connectés à l'interface SAI de la carte. Celui-ci permet de compresser ou de décompresser un signal numérique, en l'occurrence ici : un signal audio.
Ce Codec Audio comprend également :
 - *Un Analog Line Input* qui est connecté à l'ADC de la carte via le jack bleu.
 - *Un Analog Line Output* qui est connecté à un des DAC de la carte via le jack vert
 - Et enfin, deux microphones (ST-MEMS microphone) sont présents sur la STM et sont connectés sur l'entrée micro du Codec
- Un écran LCD tactile permettant de pouvoir visualiser tout type d'informations
- Mémoire SDRAM connecté à l'interface FMC de la carte STM. La taille de SDRAM est de 128-Mbit, mais seuls les 16 bits de poids faibles sont utilisables. Il n'y a donc que 64-Mbit d'accès.



FIGURE 1 – Carte STM32F746NG-Discovery

2 Travaux

2.1 Introduction et réflexions

Initialement, le code fourni permet la récupération du son ambiant et sa restitution via un casque connecté au port jack présent sur la carte. L'écran LCD est également déjà initialisé et permet d'afficher les niveaux d'intensité sonore perçue par les deux micros (en dB).

Après avoir essayé différentes valeurs pour la taille du buffer `AUDIO_BUF_SIZE`, il est notable qu'à partir d'**une taille de 1024** (2048 pour le buffer DMA), de la latence apparaît à l'écoute. Plus la taille d'`AUDIO_BUF_SIZE` est grande et plus la latence sera perceptible par l'oreille humaine. Ce paramètre est donc important afin de traiter le signal en temps réel efficacement. Dans la suite du projet, on initialise la **taille de ce buffer à 512**.

2.2 Réalisation d'un écho

Le premier traitement que l'on souhaite appliquer sur notre signal audio est un effet d'écho. Celui-ci se compose de plusieurs éléments paramétrables : un retard, de période " d ", le feedback noté " fb " correspondant au taux de ré-injection, ainsi que le taux de mélange entre le signal reçu en temps réel (" DRY ") et le signal retardé (" WET ").

Son implémentation est illustrée ci-dessous :

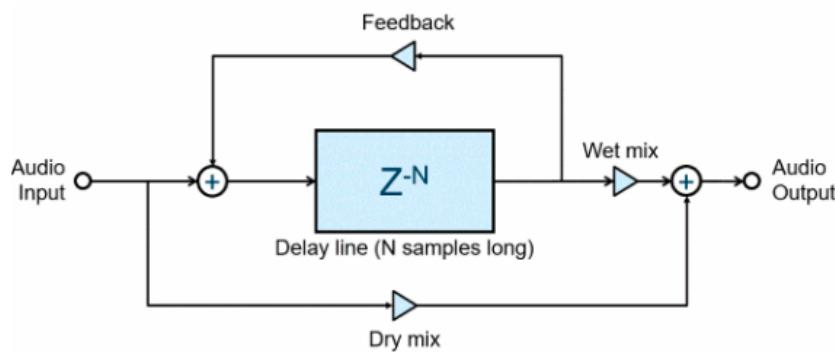


FIGURE 2 – Schéma bloc de la fonction écho

En pratique, la réalisation de cet effet se fait en deux étapes :

- Récupération du signal retardé : `memory = in[n] + fb * readFromAudioScratch(pos)` ; qui permet la récupération de l'entrée actuelle sommée à un échantillon passé (correspondant au retard) atténué par fb . La variable `pos` ci-dessus est bien entendu incrémentée sur la durée du `delay`.
- Calcul de la sortie à renvoyer : `out[n] = DRY * in[n] + WET * memory` ; déterminant l'importance de la trame actuelle et celle du retard grâce aux coefficients DRY / WET à définir selon le rendu souhaité.

Pour réaliser cet écho comme nous le souhaitions, nous avons eu besoin d'utiliser un buffer audio de grande taille, afin de pouvoir de stocker des échantillons sur une longue durée (dans notre cas la taille doit au moins être égale à " d ", le delay). C'est pour cela, que nous avons utilisé les fonctions `readFromAudioScratch()` et `writeToAudioScratch()`, qui permettent d'écrire directement dans la SDRAM externe. C'est celle-ci que l'on va utiliser comme buffer audio de grande taille.

En plus de cela, l'emploi d'un "ring buffer" pour éviter de dépasser les bornes du buffer et reboucler dessus le cas échéant était nécessaire.

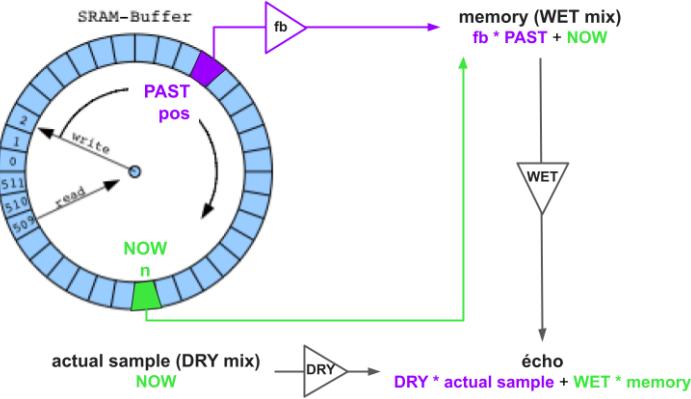


FIGURE 3 – Processus de réalisation d'un écho avec l'aide d'un ring buffer

2.3 Mise en place de RTOS

Afin de mener à bien la suite du projet et de pouvoir approfondir le traitement de l'audio (affichage de propriétés, effets...), la mise en place de RTOS est nécessaire.

Mais qu'est-ce qu'un système RTOS ?

Un système RTOS (Real-Time Operating System) est un système d'exploitation multitâche dédié aux microcontrôleurs et aux applications temps réel. Ce type d'OS permet le *scheduling* des différentes tâches que l'on attend du microcontrôleur. Celles-ci seront gérées automatiquement en fonction de leurs niveaux de priorité. Entre autres, il permettra dans la suite de pouvoir afficher le spectrogramme en temps réel sans altérer la restitution du son.

En effet, le problème qui peut se poser si on n'utilise pas de RTOS, est qu'une des tâches à exécuter prenne trop de temps par rapport au temps total alloué à cela. Ainsi, les autres tâches ne pourront pas être effectuées entièrement, ce qui peut poser problème (apparition d'artefacts audio, des clics). Par exemple, si l'affichage du spectrogramme prend trop de temps, il peut y avoir de problème lors du traitement de l'audio, comme l'apparition d'artefacts sonores.

En pratique, on cherche à séparer l'affichage graphique, du traitement audio afin qu'il n'y ait aucun problème. On définit donc deux tâches (deux threads) :

- **defaultTask** : Tâche dédiée au traitement audio temps réel (Priorité Haute).
Cette tâche doit attendre l'arrivée d'un signal provenant de l'IRQ Handler du DMA, l'informant qu'une nouvelle trame DMA est prête, pour pouvoir appeler la fonction `processAudio()` ;
- **uiTask** : Tâche dédiée à l'affichage graphique du spectrogramme (Priorité Basse).
Cette tâche doit attendre l'arrivée d'un signal l'informant que l'affichage nécessite un rafraîchissement pour être exécuté

Les deux fonctions utiles pour l'attente et l'envoi des signaux sont :

- `osSignalWait (signal, millisec)`, permettant d'attendre un signal provenant d'une autre tâche.
- `osSignalSet(task, signal)`, permettant d'envoyer un signal à une tâche donnée, et donc de la 'débloquer'.

2.4 Calcul en temps réel de la TF d'une trame audio et affichage

Dans le but de tracer le spectrogramme du son ambiant capté par les microphones embarqués sur la carte, il est important d'utiliser le principe du "double buffering". Celui-ci consiste simplement en la récupération de la moitié de la trame audio par DMA simultanément au traitement de l'autre moitié.

L'objectif est ensuite de calculer alternativement la TF de la moitié de la trame et d'en afficher le module de façon glissante. Pour cela, on choisit d'utiliser les fonctions de la bibliothèque CMSIS-DSP.

CMSIS-DSP est une bibliothèque dédiée au traitement du signal. Elle contient une centaine de fonctions mathématiques pour du traitement du signal (filtres, FFT, convolutions, etc...). Avec cette bibliothèque, il est très simple de calculer une transformée de Fourier rapide sur un signal réel, il faut simplement utiliser la fonction `arm_rfft_fast_f32()`.

La documentation de CMSIS-DSP indique les étapes à suivre pour le faire correctement :

- Initialisation d'une structure avec : `arm_rfft_fast_instance_f32` et `arm_rfft_fast_init_f32`
- Calcul de la transformée de Fourier rapide du signal
- Puis comme le résultat de la transformée est un tableau de complexes, il faut le convertir en tableau de modules grâce à : `arm_cmplx_mag_f32()`.

NB : On utilisait déjà le double-buffering pour le traitement audio afin de le rendre plus rapide.

Affichage du spectrogramme sur l'écran LCD

Après le calcul du spectrogramme du signal audio, on cherche désormais à paramétrier son affichage en temps réel avec les fonctions existantes du fichier `disco_lcd.c`. On utilise notamment la fonction `LCD_DrawPixel()` que l'on a modifiée afin qu'elle puisse prendre en argument une variable `COULEUR` (correspondant aux modules de la STFT calculés précédemment).

Pour l'affichage, on choisit de conserver les informations affichées précédemment et de ne réservé qu'une partie de l'écran à l'affichage du spectrogramme. On obtient finalement le résultat ci-dessous :

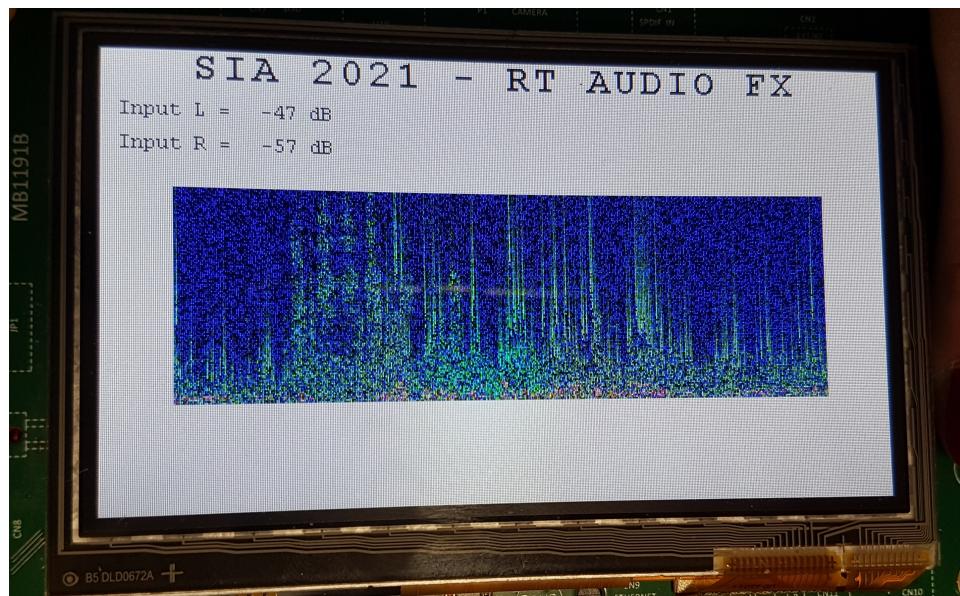


FIGURE 5 – Résultat de l'affichage du spectrogramme en temps réel

3 Conclusion

Ce projet nous a permis de faire face au traitement d'un signal audio en temps réel, et des contraintes impliquées par celui-ci. En particulier, on retiendra les éléments suivants :

- Les tailles des *buffers* méritent une attention toute particulière. La latence, si faible soit elle, ou un échantillon manquant est facilement perceptible par l'oreille humaine ;
- Le processus de traitement nécessite parfois le recours de méthodes telles que le *double buffering* ou l'utilisation d'un *ring buffer* pour ne pas affecter le signal et permettre un traitement plus aisé ;
- Si d'autres tâches sont souhaitées, comme par exemple un affichage gourmand (celui du spectrogramme en temps réel dans notre étude), il peut être utile de développer l'outil via RTOS afin de s'assurer que toutes les tâches sont correctement exécutées.