Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

TP : Traitement audio en temps réel sur STM32F746

Baptiste Rossigneux - Simon de Moreau

SIA TP1

16 Janvier 2022

Objectifs

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

- Comprendre l'architecture d'un système audio temps-réel
- Estimer les performances d'un système audio
- Comprendre l'influence de paramètres audio
- Implémenter un filtre de délai avec feedback (échos)
- Afficher un spectrogramme en temps réel
- Comprendre le fonctionnement de RTOS

Performances d'un système

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

D'après le théorème de Shannon, notre système doit avoir une fréquence d'échantillonnage qui corresponds (au minimum) au double de la fréquence maximale du signal audio à traiter.
Ce qui signifie, pour un signal de :

- Paroles : ~3000Hz il faut une fréquence d'échantillonnage d'au moins 6Khz, en général 8Khz.
- Musique : L'oreille humaine est capable d'entendre jusqu'à environ 20Khz, l'échantillonnage doit donc être supérieur à 40Khz, la norme CD est de 44.1Khz, dans des systèmes haute fidélité il est possible de monter jusqu'à 48Khz.

Dans le cas d'un système échantillonnant à 48Khz la période entre deux échantillons est de 20,8µs à titre de comparaison une instruction à 200Mhz prend environ 10ns.
Ce qui signifie que le microprocesseur est capable de réaliser environ 200 instructions, cela s'avère évidemment insuffisant en général.

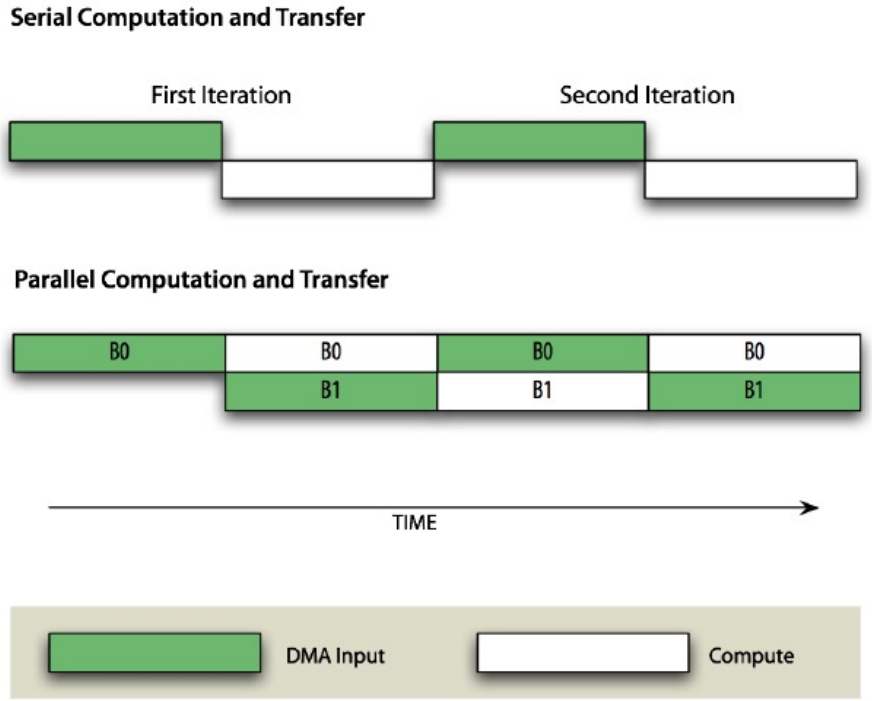
C'est pourquoi nous choisissons d'utiliser une fréquence plus basse de 16Khz ce qui signifie une période de 62,5µs. Le système sera donc incapable de traiter des signaux haute fréquence mais restera très bon si l'on souhaite l'utiliser pour des signaux de paroles.
Mais même avec une fréquence d'échantillonnage plus faible, cela n'est pas suffisant pour réaliser un traitement audio couteux en calcul, c'est pourquoi nous mettons en place un double buffering.

Double buffering

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Le double buffering est une organisation de la mémoire qui utilise deux buffers pour stocker et traiter les données en temps réel. Cette architecture utilise le DMA (Direct Memory Access) qui est une puce spécialisée dans le transfert de mémoire et autonome.

Le DMA va transférer la mémoire dans un buffer depuis le SAI audio pendant que le processeur est entrain de traiter l'autre buffer, puis les deux buffers sont inversés. Le DMA est autonome et capable d'envoyer des interruptions au processeur pour l'informer de l'état de remplissage des buffers.
Ainsi cela permet au processeur de pouvoir traiter un nombre N d'échantillons à la fois plutôt qu'un à la fois.
De plus il dispose d'un temps plus grand pour traiter les données car le DMA prend en charge les données arrivantes.

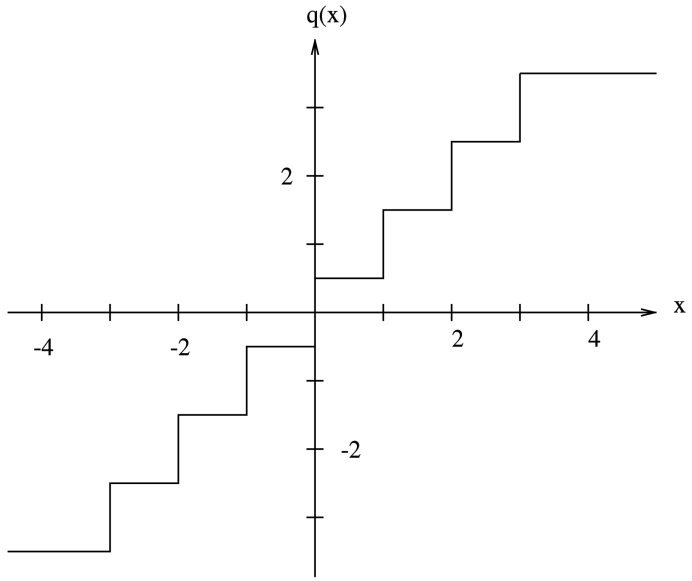


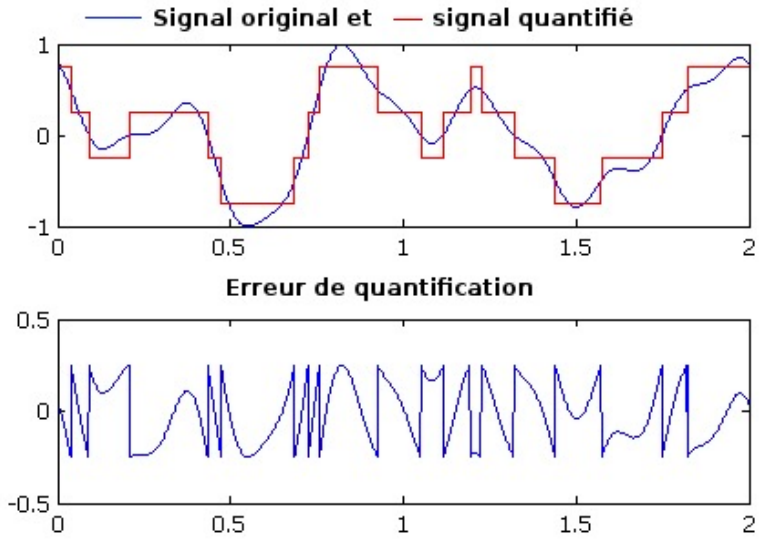
Ainsi le processeur dispose de $N/fréquence$ secondes pour traiter les données sans perte d'information, il est important que ce temps ne dépasse pas 20ms pour que l'oreille humaine ne ressente pas le retard. Dans notre cas : 1024/16000 = 6.4ms.
En augmentant la taille du buffer on augmente donc la latence audio, mais cela permet de gagner du temps de calcul pour les traitements du processeur.

Quantification

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

La quantification est le procédé qui permet de transformer un signal continu en un ensemble de valeurs discrètes que l'on est capable de représenter numériquement à l'aide de bits.





Il s'agit d'un processus non linéaire qui entraine des erreurs de quantifications, plus ou moins importantes en fonction notamment du nombre de bits utilisés. L'ordre de grandeur en termes de dynamique est de 6dB par bits utilisé pour la quantification.

Quantification

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Ainsi avec 16bits (uint_16) nous obtenons 96dB de dynamique. Suivant la bande dans laquelle se trouve notre signal, cette dynamique est acceptable compte tenu de la dynamique de l'oreille.

Si l'on souhaite une plus grande dynamique (plus de fidélité) il est possible de se tourner vers les floats (32bits) avec des virgules flottantes, qui nous offrent une dynamique théorique de 1638dB. Cependant en pratique lorsque notre traitement requiert des additions la dynamique va être réduite car leur valeur ne peut pas différer de plus de 144dB.

Les floats permettent donc une bien meilleure représentation des valeurs cependant, leur traitement par le processeur est plus long à cause de la virgule flottante, il faut donc prendre cela en compte lors du calcul des temps de traitement.

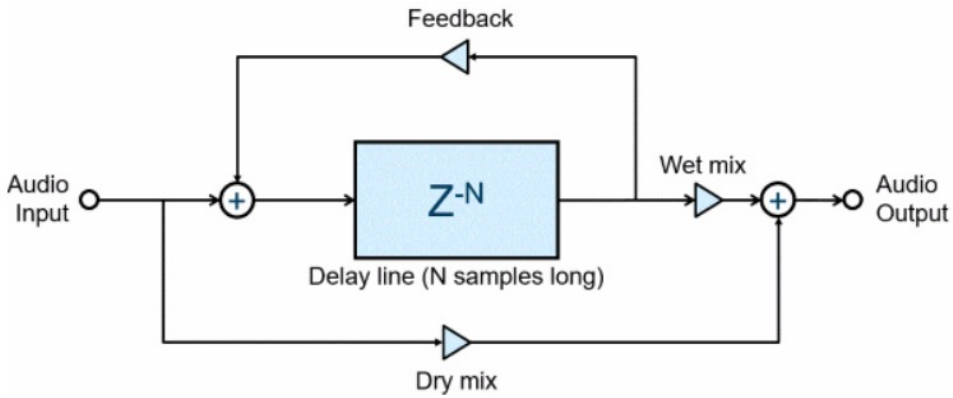
Dans le cas de la STM32, nous disposons de FPU (floating point unit) qui sont spécialisés dans le traitement de ce type de données et qui vont fortement accélérer les calculs.

Délais et échos

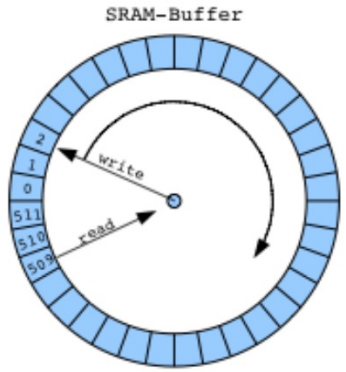
Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Dans un premier temps pour bien comprendre le fonctionnement du traitement audio nous souhaitons réaliser un effet très simple, à savoir un délai puis un écho.

Un effet d'écho correspond simplement à un délai auquel on ajoute le feedback suivant le schéma bloc ci-contre.
Pour implémenter le délai nous allons devoir stocker en mémoire le nombre d'échantillon souhaité.
La méthode la plus adapté à ce type de tâche est d'utiliser une mémoire circulaire avec le nombre d'échantillon souhaité.



Ainsi nous n'utilisons que l'espace mémoire nécessaire et les nouveaux échantillons écrasent les anciens au fur et à mesure grâce au rebouclage.
Comme la RAM de la STM32 est déjà très remplie par l'affichage, nous devons utiliser la SDRAM pour réaliser cette mémoire, bien qu'elle soit plus lente cela ne sera pas un problème dans ce cas précis.



Délais et échos

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Cela nous donne les codes suivants :

Gestion de la mémoire circulaire

```
static int circlePos(int pos, int size){
    if (pos<0)
        return (size-pos)%size;
    if (pos>=size)
        return pos%size;
    return pos;
}
```

Implémentation du schéma bloc

```
int delay = AUDIO_BUF_SIZE*100;
float fb = 0.6;
float global_gain = 0.8;
// ---------------------------- AUDIO ALGORITHMS ----------------------------

/**
 * This function is called every time an audio frame
 * has been filled by the DMA, that is,  AUDIO_BUF_SIZE samples
 * have just been transferred from the CODEC
 * (keep in mind that this number represents interleaved L and R samples,
 * hence the true corresponding duration of this audio frame is AUDIO_BUF_SIZE/2 divided by the sampling frequency).
 */
static void processAudio(int16_t *out, int16_t *in) {
    LED_On(); // for oscilloscope measurements...
    int sdpos = 0;
    for (int n = 0; n < AUDIO_BUF_SIZE; n++) {
        sdpos = circlePos(d,delay);
        int y = global_gain*in[n]+fb*readFromAudioScratch(sdpos);
        writeToAudioScratch(y, sdpos);
        out[n] = y;

    }
    LED_Off();
}
```

Le résultat est très satisfaisant, nous obtenons ainsi un écho qui fonctionne parfaitement avec un temps de retard de $\frac{1024\times100}{16000} = 0{,}64s$.
Le système bien que simple permet de bien comprendre le traitement entre les échantillons d'entrée et de sortie et de voir les effets lorsque le traitement est trop long (ex : apparition de clics) ou lorsque l'on change la taille du buffer (changement de la latence)

RTOS

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Pour la suite de ce TP nous souhaitons réaliser des traitements beaucoup plus long.
Il devient donc nécessaire de nous intéresser à RTOS, il s'agit d'un mini système d'exploitation pour les microcontrôleurs qui permet de la gestion temps réel.
Cet OS permet de générer plusieurs threads et de leur associer des priorités, ainsi les traitements prioritaires (ici l'audio) s'exécutent toujours en premier même si une autre tâche moins prioritaire était en cours (dans notre cas l'affichage de l'écran).
Cela permet des traitement « parallèle » avec une plus grande souplesse. Les différents threads peuvent s'échanger des Signaux pour savoir lorsqu'un thread doit s'exécuter ou bien attendre à l'aide de *osSignalSet()* ou *osSignalWait()*.
Ainsi nous séparons le système en deux threads : Audio (prioritaire) et Display (non prioritaire). Tous les traitements de type attente (*while(1)*) sont remplacés par des *osSignalWait()* ce qui permet de libérer le processeur pour traiter l'autre thread.

Lorsqu'un nouveau buffer est complété le DMA réalise un *osSignalSet()* pour prévenir le thread audio qu'il doit traiter le buffer. De même, lorsque suffisamment de données ont été traitées, un signal est généré pour que l'affichage se mette à jour.
Avec cette architecture le système est capable de gérer des traitements audio beaucoup plus longs, car la tâche la d'affichage (la plus longue) n'est pas critique et sera donc réalisée dans les « temps morts » du traitement audio ainsi l'expérience utilisateur au niveau de l'écoute sera toujours bonne.

TFTD en temps réel

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Maintenant que nous utilisons RTOS, il nous est possible de réaliser des traitements plus couteux en calculs, tel qu'une FFT, nous allons donc pouvoir réaliser une TFTD en temps réel.

Pour cela nous utilisons CMSIS-DSP qui implémente le calcul de FFT en utilisant à la fois les MAC (Multiply and Accumulate) et les FPU de la carte, ce qui permet un temps de calcul optimal.

Nous pouvons donc calculer la FFT (réelle car signal audio) puis passer au module pour l'affichage, à l'aide de la structure *arm_rfft_fast_instance_f32* puis des fonctions *arm_rfft_fast_f32()* et *arm_cmplx_mag_f32()*.

Concernant l'affichage, nous passons au log pour avoir un niveau en dB puis nous définissons une valeur minimale, pour filtrer le bruit ambiant et se concentrer sur le signal de parole.

Après normalisation nous transformons chaque valeur en une couleur à l'aide puis nous l'affichons avec l'axe temporel en horizontal et l'axe fréquentiel en vertical.

Les fréquences affichées vont jusqu'à 8Khz étant donné notre fréquence d'échantillonnage.

TFTD en temps réel

Objectifs

Performances d'un système

Double buffering

Quantification

Délais et échos

RTOS

TFTD en temps réel

Résultat

Conclusion

Voici le code correspondant :

Calcul de la FFT en temps réel

```
int delay = AUDIO_BUF_SIZE*100;
float fb = 0.6;
int d = 0;
float global_gain = 0.8;
float outfft[FFT_SIZE];
float infft[FFT_SIZE];
int indfft = 0;
// ---------------------------- AUDIO ALGORITHMS -----------------------------

/**
 * This function is called every time an audio frame
 * has been filled by the DMA, that is,  AUDIO_BUF_SIZE samples
 * have just been transferred from the CODEC
 * (keep in mind that this number represents interleaved L and R samples,
 * hence the true corresponding duration of this audio frame is AUDIO_BUF_SIZE/2 divided by the sampling frequency).
 */
static void processAudio(int16_t *out, int16_t *in) {
    LED_On(); // for oscilloscope measurements...
    int sdpos = 0;
    indfft = 0;
    for (int n = 0; n < AUDIO_BUF_SIZE; n++) {
        sdpos = circlePos(d,delay);
        int y = global_gain*in[n]+fb*readFromAudioScratch(sdpos);
        writeToAudioScratch(y, sdpos);
        out[n] = y;
        if(n%2 == 0){
            infft[indfft] = in[n];
            indfft++;
        }
        d++;
    }

    // FFT
    arm_rfft_fast_f32(&fft, infft, outfft, 0);
    arm_cmplx_mag_f32(outfft, displayFFT, FFT_SIZE/2);
    osSignalSet (*puiTaskHandle, 0x02);
    LED_Off();
}
```

Affichage du spectrogramme

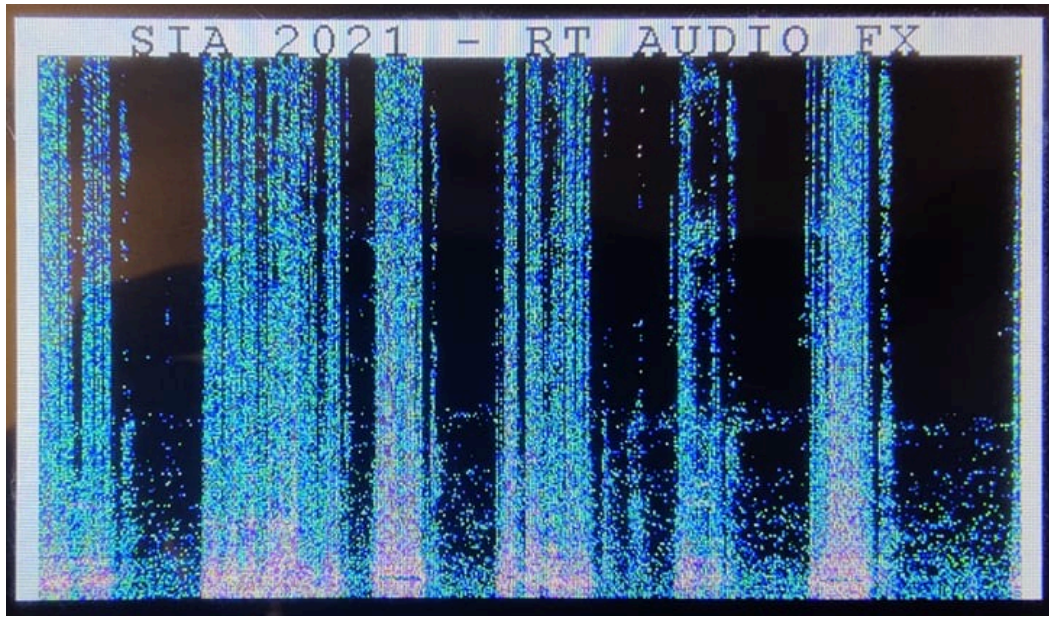
```
#define  LCD_SCREEN_WIDTH    480
#define  LCD_SCREEN_HEIGHT   272
int x=10;
/**
 * Displays FFT on screen
 **/
void uiDisplayFFT(float* fftabs, int fftsize) {
    float maxvalue = 40;
    float minvalue = 12;
    //arm_max_f32(fftabs, fftsize, &maxvalue, NULL);
    //arm_min_f32(fftabs, fftsize, &minvalue, NULL);
    for(int i=0; i<fftsize; i++){
        float val = 20.*log10(fftabs[i]/fftsize);
        if(val<minvalue)
            val=minvalue;
        //if(val>maxvalue)
        //  val=maxvalue;
        double normval = (val-minvalue)/(maxvalue-minvalue);
        uint16_t color = normval * 0x7FFF;
        //LCD_SetStrokeColor(color);
        LCD_DrawPixel_Color(x, LCD_SCREEN_HEIGHT-i,color);
    }
    x++;
    if(x>LCD_SCREEN_WIDTH-10)
        x=10;

}
```

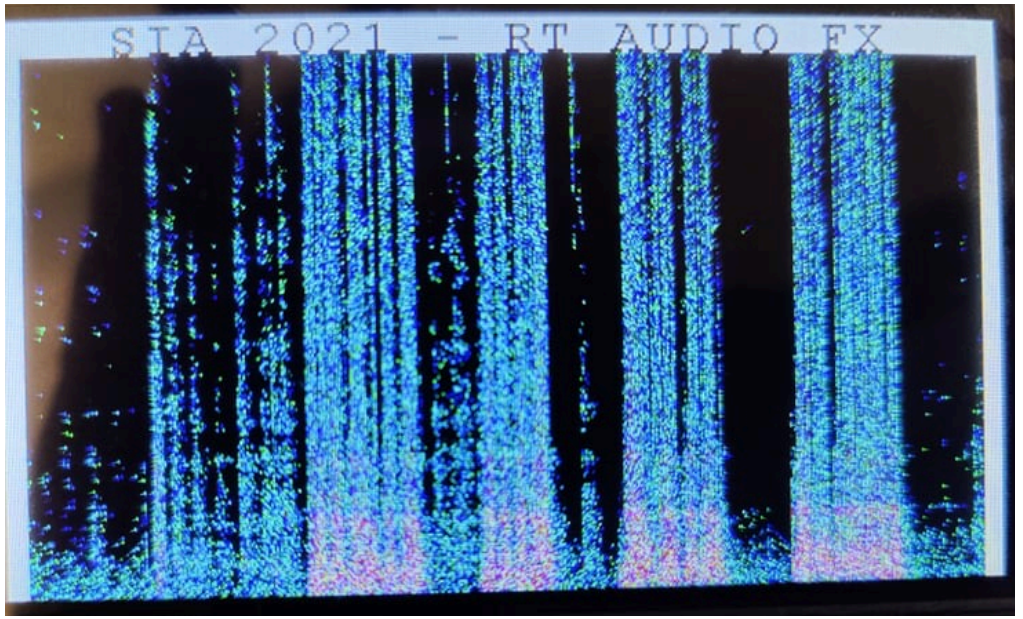
Le résultat est très satisfaisant (cf slide suivante), le temps de rafraichissement est constant et nous pouvons facilement observer les variations de fréquence dans le signal capté.
Il serait possible d'appliquer un meilleur filtrage du bruit et de trouver une meilleure fonction pour déterminer la couleur à afficher (éventuellement un tableau).

Résultat

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

Voici des exemples de spectrogrammes calculés par la carte :





L'axe horizontal corresponds au temps et reboucle lorsqu'il arrive à droite.
L'axe vertical corresponds aux fréquences jusqu'à 8000Hz.
La couleur correspond à la valeur en dB , du moins élevé au plus élevée : Noir, bleu, vert, rouge, blanc.

Conclusion

Objectifs
Performances d'un système
Double buffering
Quantification
Délais et échos
RTOS
TFTD en temps réel
Résultat
Conclusion

- Nous avons compris l'architecture d'un système audio temps-réel
- Nous savons maintenant estimer les performances d'un système audio en fonction de ses caractéristiques
- Nous comprenons maintenant l'influence des différents paramètres audio sur le signal final, sa latence, sa fidélité, etc…
- Nous avons implémenté un filtre d'échos et testé les différents paramètres
- Nous avons réussi à afficher un spectrogramme en temps réel à l'aide de RTOS pour faciliter le temps réel.
- Nous avons compris le fonctionnement de RTOS et de ses signaux
- Pour aller plus loin nous pourrions réaliser un effet audio plus complexe tel qu'un compresseur ou une réverbération en utilisant la FFT ou un vocodeur
- Code disponible sur Github : https://github.com/Urashx/AUDIO_TP

Merci de votre lecture