



**ÉCOLE UNIVERSITAIRE  
DE PHYSIQUE ET D'INGÉNIERIE**  
Université Clermont Auvergne

---

# Développement d'un système de réverbération audio-numérique sur SoC-FPGA

---

M2 SYSTÈMES EMBARQUÉS POUR LE TRAITEMENT  
DU SIGNAL, DE L'IMAGE ET DU SON

PROJET DE FIN D'ÉTUDES

Rédigé par  
Vincent FAUGÈRE

01/12/2023 - 26/02/2024

# Table des matières

<b>1 Présentation du projet</b>	<b>3</b>
1.1 Phénomène de réverbération sonore . . . . .	3
1.2 Intérêt du déploiement sur cible FPGA . . . . .	3
1.3 Choix du type de réverbération numérique . . . . .	3
<b>2 Matériels/Logiciels utilisés</b>	<b>4</b>
<b>3 Acquisition des données audio</b>	<b>6</b>
3.1 Configuration du CODEC audio . . . . .	6
3.2 Interface audio avec le FPGA . . . . .	7
<b>4 Conception/Simulation</b>	<b>9</b>
4.1 Modélisation . . . . .	9
4.2 Early reverb block . . . . .	10
4.3 Late reverb block . . . . .	11
4.3.1 LFCF : Low-Pass Feedback Comb Filter . . . . .	12
4.3.2 LPF : Low-Pass Filter . . . . .	13
4.3.3 APF : All-Pass Filter . . . . .	14
<b>5 Implémentation d'une ligne à retard</b>	<b>16</b>
5.1 Contraintes . . . . .	16
5.2 Fonctionnement interne . . . . .	16
<b>6 Partie logicielle</b>	<b>18</b>
6.1 Interface utilisateur . . . . .	18
6.2 Chargement du programme sur carte SD . . . . .	19
<b>7 Gestion du projet</b>	<b>20</b>

# 1 Présentation du projet

Ce projet de fin d'études en Master 2 Systèmes Embarqués consiste à développer un système de réverbération audio-numérique de synthèse sur FPGA. Il s'inscrit dans le cadre du traitement numérique du son.

## 1.1 Phénomène de réverbération sonore

La réverbération sonore est un phénomène physique provenant des multiples réflexions du son sur les parois d'un espace confiné. On retrouve des types de réverbération complètement différents suivant les espaces où l'on se trouve (chambre, salle de concert, stade, etc...). L'oreille humaine est donc particulièrement sensible à ce phénomène, c'est donc pour cela que l'on cherche à en rajouter (notamment lors d'un mixage en studio) pour spatialiser le son et le rendre beaucoup plus agréable à écouter. Avec l'essor des technologies numériques, il est beaucoup plus facile de l'utiliser après enregistrement d'un son, soit pour reproduire de façon réaliste un lieu comme une salle de concert par exemple, ou bien pour ajouter des effets moins réalistes mais tout autant appréciables à l'oreille (réverbération artificielle beaucoup utilisée dans les musiques des années 80 par exemple).

## 1.2 Intérêt du déploiement sur cible FPGA

L'intérêt de développer un tel système sur cible FPGA est son utilisation en temps réel.

Les processeurs actuels sont largement capables de faire tourner des algorithmes de réverbération (des blocs DSP sont présents dans beaucoup de CPU) mais les données doivent être stockées en mémoire avant d'être traitées, ce qui induit une latence plus ou moins importante. C'est donc pour cela que l'ajout de réverbération après enregistrement ne pose aucun problème sur CPU mais qu'il est plus difficile de les utiliser pour un traitement temps réel.

Les logiciels de traitement audio DAW (*Digital Audio Workstation*) permettent de configurer la taille du buffer audio, c'est à dire la quantité de données audio stockées en mémoire avant traitement par le CPU. Plus la taille de ce buffer sera grande, moins le CPU sera utilisé (car plus de données seront traitées en parallèle et ceci moins souvent), et donc plus la latence sera haute. Inversement, plus le buffer sera petit plus le CPU sera fortement sollicité et plus la latence sera faible. Il s'agit d'un compromis entre faible latence et consommation CPU. D'une manière générale, il est très difficile d'atteindre une latence très faible sans que la charge CPU explose. Ceci est d'autant plus vrai lorsque l'ordinateur est doté d'un système d'exploitation, et d'une multitude de processus allouant des ressources CPU...

En live, il n'est pas du tout souhaitable de jouer d'un instrument et d'entendre la sortie décalée par rapport au jeu (même très légèrement). La latence doit donc être inférieure à la milliseconde pour qu'il n'y ait aucune différence perceptible par le musicien.

Le traitement du flux de données sur le FPGA élimine toute latence. De manière rigoureuse, la latence résultante du système de réverbération ne sera que de quelques cycles d'horloge à 50 MHz (cf 3.2), ce qui est complètement négligeable !

## 1.3 Choix du type de réverbération numérique

On distingue deux grandes familles de réverbération numérique :

- les réverbérations algorithmiques
- les réverbérations par convolution

Les réverbérations par convolution exploitent la mesure de la réponse impulsionnelle d'une salle pour l'utiliser dans le traitement. Elles sont en général très réaliste mais ne sont pas (ou peu) paramétrables. Elles demandent également de stocker un très grand nombre de coefficients selon la longueur de la réponse impulsionnelle dans une mémoire non volatile. Par exemple, pour une réponse impulsionnelle durant 4 secondes, il faudrait stocker 192000 coefficients pour une fréquence d'échantillonnage de 48 kHz seulement (soit 576 kB avec une résolution de 24 bits), ce qui est assez complexe en pratique. D'autre part, le nombre d'opérations MAC (*Multiplication and Accumulation*) serait hallucinant (192000 également). De ce fait, il est a priori difficile d'implémenter une bonne réverbération par convolution sur FPGA.

La réverbération algorithmique utilise différentes structures comprenant des retards pour générer une réverbération de synthèse. Le puissant avantage de celle-ci réside dans le fait qu'elle peut être fortement paramétrable. Bien évidemment, elle est implémentable sur FPGA puisqu'elle utilise très peu de multiplications (par rapport à la réverbération par convolution).

Le projet se basera alors sur une structure de réverbération algorithmique sur FPGA. La structure conçue est inspirée de l'unité de réverbération stéréo Freeverb du domaine public, déjà implémentée dans différents langages (C++, Matlab, etc...) [1].

## 2 Matériels/Logiciels utilisés

En premier lieu, pour effectuer différentes simulations et s'assurer de la bonne structure du système de réverbération conçu, le logiciel **Matlab** et l'extension **Simulink** ont été utilisés.

Pour l'implémentation du système, la carte de développement suivante a été utilisée :

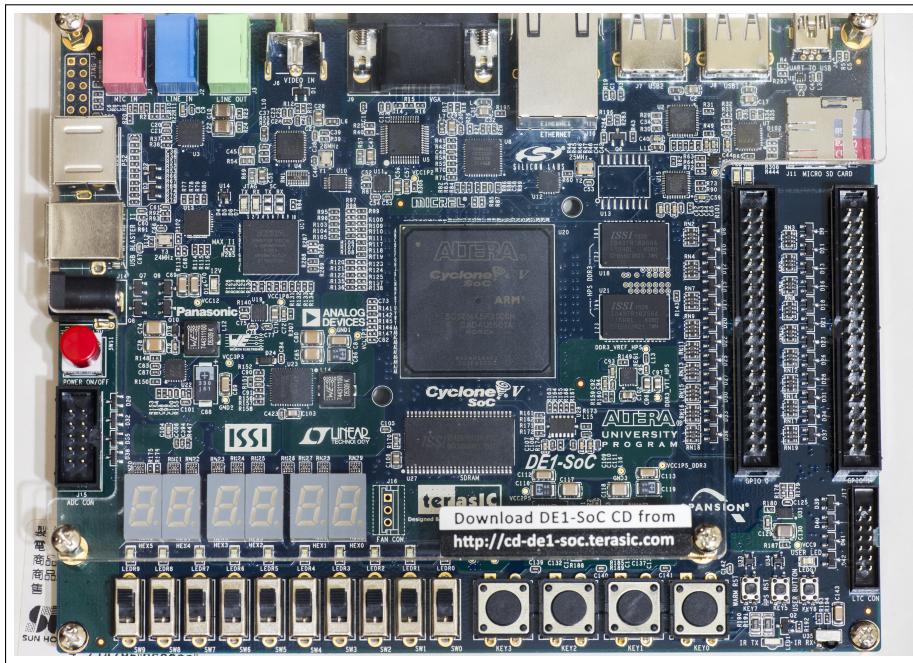


FIGURE 1 – Carte de développement Altera/Terasic DE1-SoC

Elle dispose d'un FPGA Cyclone V, contenant une partie de logique programmable et un HPS (*Hard Processor System*) ARM Cortex-A9. Différents périphériques externes présents sur la carte sont connectés à la partie FPGA ou à la partie HPS.

L'environnement utilisé pour la synthèse FPGA est **Quartus Prime 23.1**, et le design a été

élaboré sous **Plateform Designer Qsys**. L'IDE utilisé pour programmer et déboguer le HPS est **Arm DS IDE 2023.1**. Des outils de **SoC-EDS** ont également été utilisés.

Le FPGA comporte tous les modules pour le traitement de la réverbération tandis que le HPS se charge de la configuration du codec audio et gère l'interface utilisateur.

### 3 Acquisition des données audio

#### 3.1 Configuration du CODEC audio

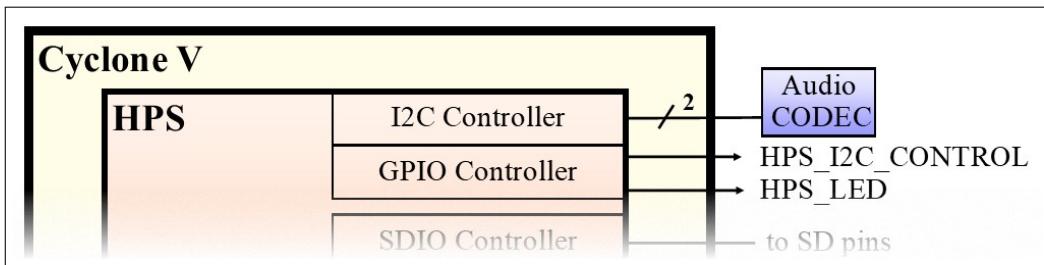


FIGURE 2 – Interface HPS - CODEC audio

Le CODEC audio doit être au préalable configuré avant de pouvoir correctement utiliser les données. Pour cela, on connecte la sortie du contrôleur I2C intégré au HPS au bus de contrôle du CODEC audio.

Le signal HPS\_I2C\_CONTROL (pin du HPS) spécifique à la carte DE1-SoC permet de spécifier si le bus I2C connecté au CODEC audio/décodeur TV de la carte est relié au FPGA, ou au HPS. Le programme l'affecte à 1 au tout début pour pouvoir configurer le CODEC depuis le HPS.

Le signal HPS\_LED est quant à lui connecté à la USER LED sur la carte. Celle-ci s'allumera lorsqu'une erreur de communication avec le CODEC aura lieu pendant sa configuration.

La communication I2C via le programme lancé sur le HPS est facilité par l'utilisation de la bibliothèque HWLib fournie par Intel SoC-EDS.

Les paramètres choisis pour la configuration du CODEC audio (WM8731) sont regroupés dans le tableau suivant. Pour plus d'informations sur les registres internes du CODEC et leur utilité, se référer à la documentation correspondante.

REGISTER	VALUE
Left Line In (0b0000000)	0b000010011
Right Line In (0b00000001)	0b000010011
Analog Audio Path Control (0b0000100)	0b00010010
Digital Audio Path Control (0b0000101)	0b00001
Digital Audio Interface Format (0b0000111)	0b01001001
Sampling Control (0b0001000)	0b000000010
Power Down Control (0b0000110)	0b01100010

L'adresse des registres de contrôle est sur 7 bits tandis que la taille de ces registres est de 9 bits. Il a donc fallu écrire une petite fonction pour faire la correspondance vers un buffer de 2 fois 8 bits :

```
1 uint8_t* bufferToSend(uint8_t controlAddrBits, uint16_t controlDataBits)      {  
2     static uint8_t dataBytes[2];  
3     dataBytes[0] = (controlAddrBits << 1) | ((controlDataBits & (1<<8)) >> 8);  
4     dataBytes[1] = 0xFF & controlDataBits;  
5 }
```

```

6     return dataBytes;
7 }

```

Les paramètres importants à retenir sont :

- La fréquence d'échantillonnage (spécifiée par Digital Audio Interface Format ainsi que la fréquence du signal d'horloge XCLK/AUDXCLK (cf 3)) est de 48 kHz
- La résolution est de 24 bits
- L'entrée ligne est activée. L'entrée micro est désactivée.
- Les signaux d'entrée au niveau ligne sont divisés par deux (-6 dB) pour éviter de saturer trop vite... (cf partie 4)

### 3.2 Interface audio avec le FPGA

L'interface entre le CODEC audio et la partie de traitement de la réverbération se présente comme suit :

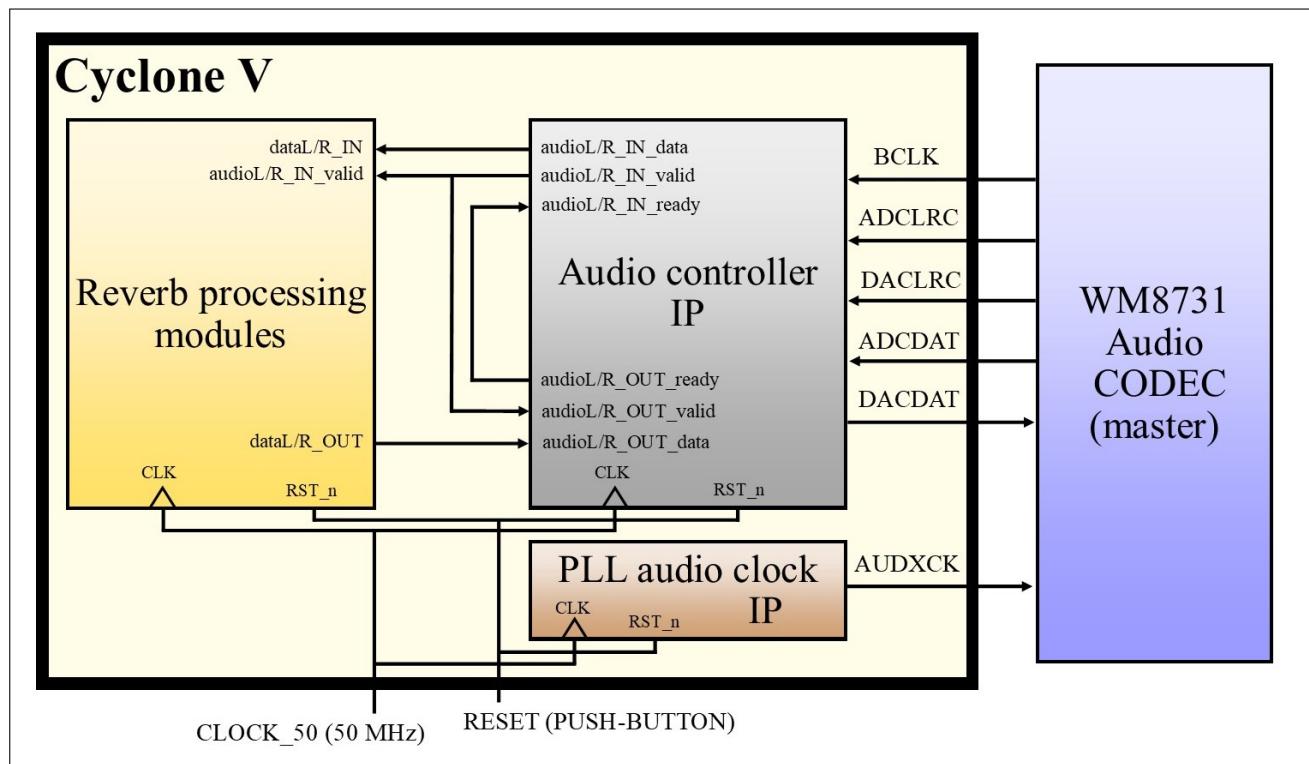


FIGURE 3 – Interfaçage des signaux du CODEC audio avec le bus Avalon-STreaming côté FPGA

Le bloc Audio Controller IP fourni par Altera permet de faire l'interface entre les signaux du CODEC audio (5 signaux de 1 bit) et le bus Avalon-ST de l'autre côté. Cet IP implémente des FIFO pour les données d'entrées et de sortie. Dans tous les cas, les données d'entrée sont valides ( $\text{audioL/R\_IN\_valid}=1$ ) lorsque les deux canaux (gauche + droite) ont reçu tout deux la donnée. De même les données sont envoyées au CODEC uniquement lorsque chaque donnée (gauche + droite) est reçue. Ceci permet d'avoir une sortie cohérente et synchrone entre la piste gauche et la piste droite.

Les signaux  $\text{audioL\_IN\_valid}$  et  $\text{audioR\_IN\_valid}$  permettent de cadencer les opérations sur les éléments synchrones (à savoir les décalages d'échantillons).

Etant donné que audioL/R\_IN\_valid est relié à audioL/R\_OUT\_valid, les FIFO internes au contrôleur audio ne sont jamais remplies de plus d'un échantillon. Ainsi le signal audio audioL/R\_OUT\_ready sera toujours à 1 (le contrôleur audio est tout le temps prêt à recevoir). Ce schéma de cablage permet ainsi d'obtenir une latence extrêmement faible. La seule latence restante se situe au niveau du transfert entre le contrôleur audio et le CODEC, mais reste bien inférieure à la période d'échantillonnage.

## 4 Conception/Simulation

### 4.1 Modélisation

La réverbération peut être modélisée par deux parties distinctes (plus ou moins en fait dans la réalité) que l'on nomme la réverbération proche (*Early Reverb*), et la réverbération lointaine ou queue de réverbération (*Late Reverb*) :

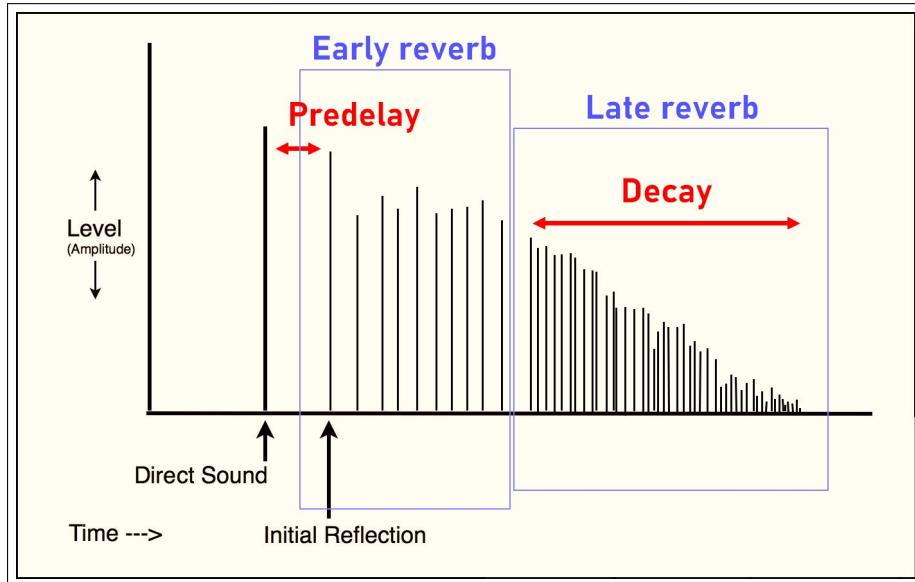


FIGURE 4 – Principe de la réverbération (cas d'une réponse à une impulsion)

Le son initial subit une première réflexion principale, que l'on appelle **Pre-delay** et qui est directement liée à la taille de la pièce. Ce retard temporel induit par la première réflexion dépend également de l'endroit où se trouve l'auditeur par rapport à la source sonore ; plus il se trouve près de la source plus le son mettra du temps à revenir vers lui par rapport au son initial.

A l'issue de la première réflexion (en général la plus forte mais pas toujours), d'autres réflexions moins fortes ont lieu juste après, à cause des différences de distances entre l'auditeur et les parois de la salle tout autour de lui.

Ensuite, les réflexions sont tellement nombreuses que l'oreille humaine ne distingue plus chaque réflexion mais seulement une sorte de résonance confuse qui forme alors la queue de réverbération (*Late Reverb*). La durée de celle-ci correspond à la durée de la réverbération que l'on appelle le **Decay**.

Bien évidemment, il est possible de réaliser de nombreuses réverbérations différentes (avec une multitude de paramètres à régler) pour modéliser au mieux la réverbération naturelle d'une salle. Ces structures peuvent être plus ou moins complexes mais beaucoup reposent sur ce principe général de réverbération en deux parties.

La structure retenue est alors la suivante :

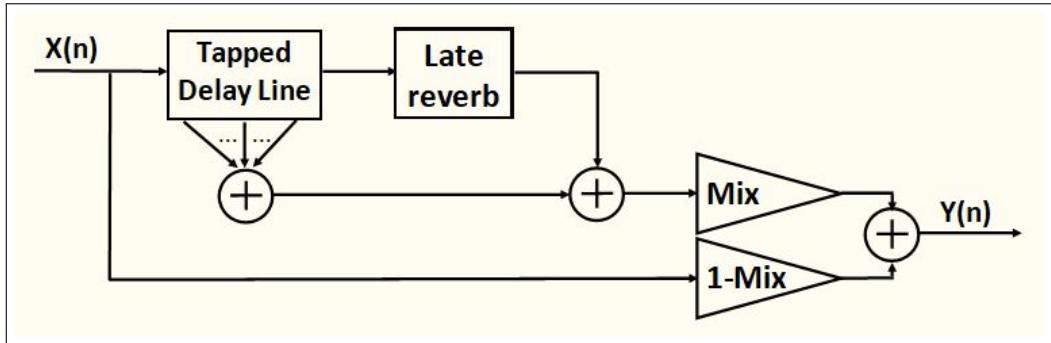


FIGURE 5 – Structure du système de réverbération implémenté

L'échantillon d'entrée est tout d'abord conduit au bloc *Tapped Delay Line* qui forme la partie *Early Reverb*. Il s'agit simplement d'une ligne à retards décalant l'échantillon d'entrée.

La première sortie du bloc (après le sommateur) correspond à l'ensemble des échantillons décalés de différents retards, ce qui correspond à la zone *Early Reverb* de la figure 4. La seconde sortie correspond uniquement à l'échantillon initial ayant subit le retard maximal du bloc, c'est à dire le dernier pic de la partie *Early Reverb* de la figure 4.

Vient ensuite le bloc *Late Reverb* dont l'entrée est alimentée par la seconde sortie du bloc précédent.

La partie réverbérée du son est ensuite obtenue en sommant la sortie générale du bloc *Early Reverb* et du bloc *Late Reverb*.

Pour couronner le tout, il est important de pouvoir régler la quantité de réverbération ajoutée par rapport au son pur, ce qui est possible avec l'ajout du paramètre **Mix** comme le montre la figure 5.

Les paramètres de réglage de cette réverbération seront donc le **Mix**, le **Pre-delay**, le **Decay** et le **Damping** qui sera vu dans la suite.

Les sections suivantes regroupent en même temps la partie simulation et certains détails utiles pour l'implémentation des modules en VHDL, en particulier la taille des échantillons lors des calculs.

## 4.2 Early reverb block

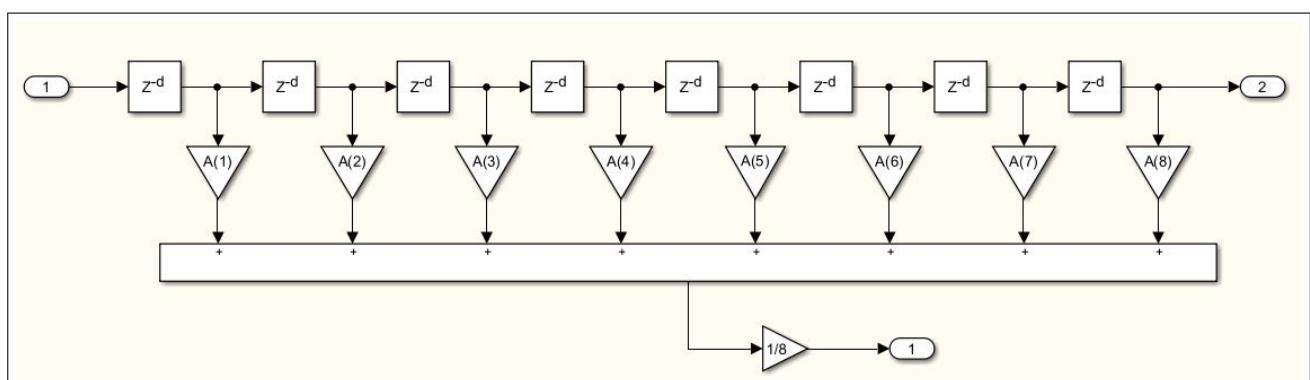


FIGURE 6 – Early Reverb (Tapped Delay Line)

Le premier retard tout à gauche correspond à la première réflexion et donc au **Pre-delay** ; il s'agit du retard le plus grand et il sera paramétrable (entre 0 et 12000 soit entre 0 ms et 250 ms à 48 kHz). Les retards suivants sont beaucoup plus faibles et permettent de générer les multiples réflexions succéderantes à la première réflexion (cf 4). Un ratio R (par exemple 50) entre le premier retard et les autres est utilisé pour calculer les retard successifs en fonction du premier retard. Ainsi, pour un retard principal de N, les retards successifs seront tous de  $N/R$ .

Chaque réflexion a un certain poids entre 0 et 1 (coefficients  $A(i)$ ).

Les retards en série ne constituent pas de changement de dynamique de la valeur d'entrée, de même que les coefficients puisqu'ils sont entre 0 et 1. Le sommateur à 8 entrée modifie quant à lui la dynamique ; le gain en dynamique est de 8. Il faut donc augmenter la taille des opérandes du sommateur de 3 bits pour éviter tout débordement. Il suffit ensuite de prendre les 24 bits de poids fort en sortie du sommateur (revient à une division par 8).

Le gain de ce module est inférieur à 1, ce qui est très important. L'objectif idéal est de concevoir des blocs dont le gain est unitaire afin de garder la même dynamique partout. Si l'on est supérieur, alors la saturation numérique peut avoir lieu, et si l'on est inférieur, alors la sortie du bloc peut être bien plus faible que l'entrée. De plus une dynamique trop différente entre les blocs induit indéniablement un poids plus important d'un bloc par rapport à un autre, ce qui n'est pas souhaitable.

### 4.3 Late reverb block

Cette partie a été inspirée du réverbérateur opensource Freeverb. Il s'agit d'une structure particulière de réverbération de Shroeder [1].

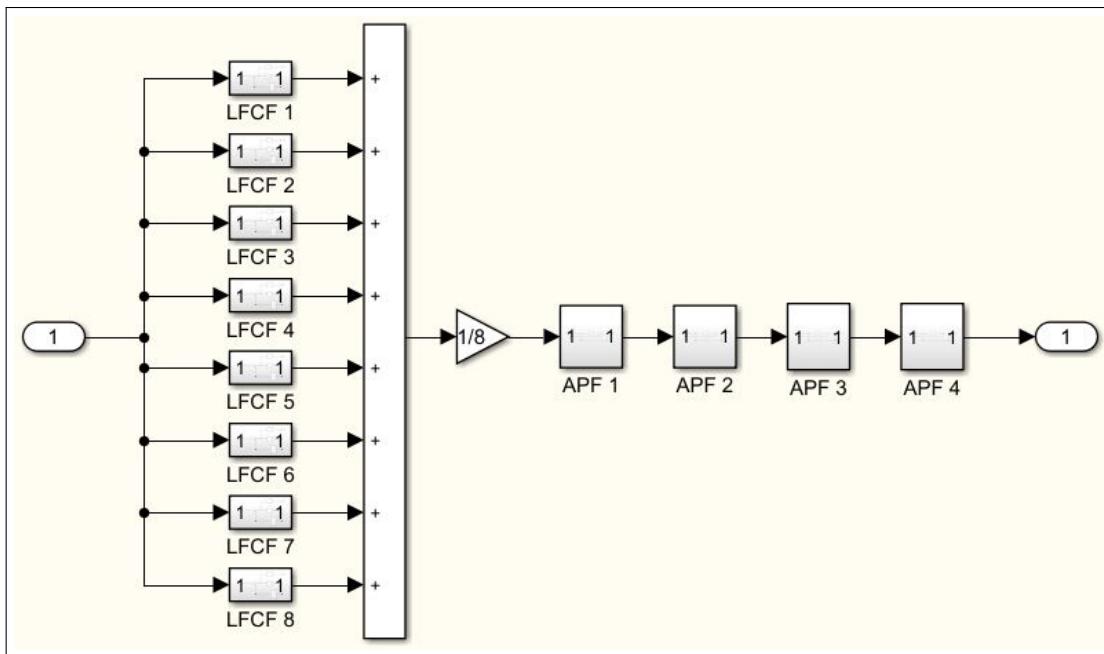


FIGURE 7 – Late Reverb

De même que précédemment, 3 bits sont ajoutés au niveau du sommateur et les 24 bits de poids fort sont récupérés sur la sortie. La dynamique au niveau du sommateur est donc inchangée.

Le détails des divers blocs sont présentés dans les sections suivantes.

Les blocs LFCF et APF comportent chacun un retard de N échantillons. D'une manière générale, plus N est grand plus on aura l'impression d'être dans une grande salle. Différentes valeurs peuvent être utilisées, les suivantes ont été testées en simulation puis utilisées dans l'implémentation matérielle :

- LFCF : 1857, 1917, 1791, 1722, 1577, 1656, 1488, 1416
- APF : 443, 997, 881, 701

#### 4.3.1 LFCF : Low-Pass Feedback Comb Filter

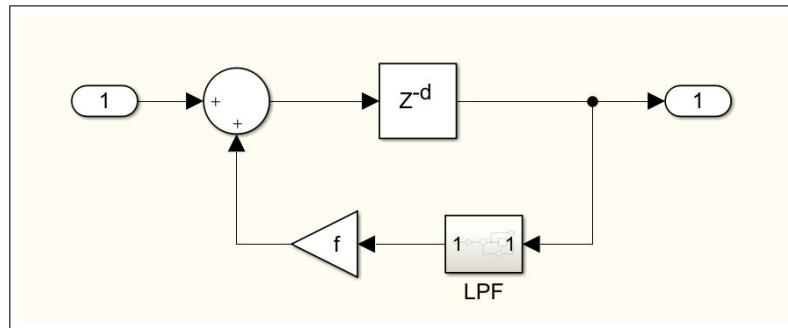


FIGURE 8 – Low-Pass Feedback Comb Filter block diagram

Ce bloc est le coeur de la génération de la queue de réverbération. La boucle de retour permet d'ajouter la valeur d'un échantillon antérieur (retard entre 1000 et 2000 en général) à l'échantillon actuel. Le poids de la boucle de retour dépend de la valeur du feedback qui correspond directement au paramètre **Decay** et également du **Damping** présent dans le bloc LPF 4.3.2.

Afin de savoir combien de bits supplémentaires sont nécessaires au sein de ce bloc, il est intéressant de tracer le diagramme de bode. On se place dans le pire des cas, c'est à dire dans le cas où le **Damping** d vaut 0 (cf 4.3.2), c'est à dire que rien n'est atténué par le bloc LPF dans la boucle de retour (équivalent à un fil à la place du bloc).

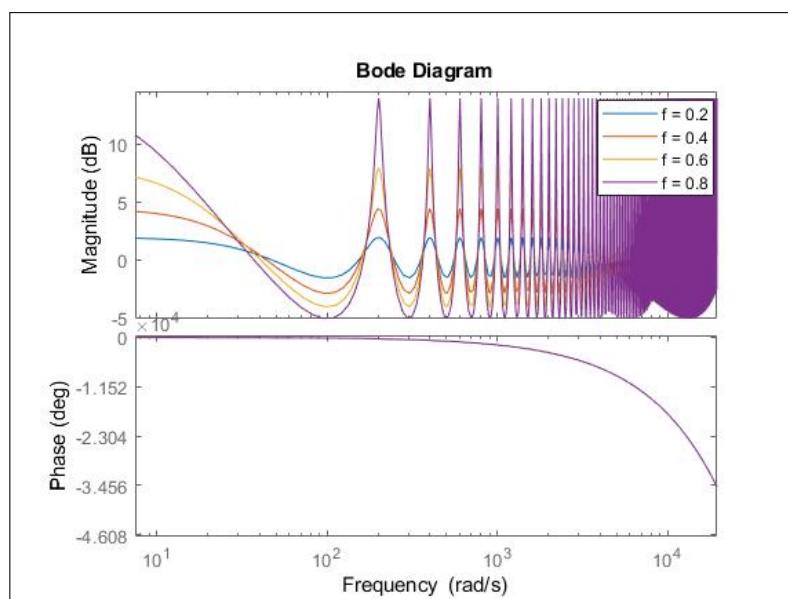


FIGURE 9 – Diagramme de Bode du bloc LFCF en fonction du feedback f (**Decay**)

D'une part, le gain dépend du feedback  $f$  et d'autre part il dépend de la fréquence. Analytiquement, le gain maximal (atteint à chaque pic) est de  $\frac{1}{1-f}$ . Une première solution pour obtenir un gain maximal de 1 serait donc de rajouter un gain de  $1 - f$  en sortie du bloc. Le problème est que la majorité du temps le gain en fonction de la fréquence est bien plus faible que le gain maximal, donc la dynamique de sortie en est fortement réduite en appliquant cette méthode.

La méthode retenue pour combler ce problème est de calculer un gain moyen en fonction de la fréquence, et ceci pour une valeur de  $f$  assez grande (potentiellement la valeur maximale de  $f$  que l'on souhaitera utiliser). Il suffit ensuite d'ajouter un gain d'atténuation en sortie du filtre égal à l'inverse de ce gain calculé. La dynamique reste donc la même en entrée qu'en sortie "en moyenne".

Le gain moyen pour  $f$  de 0.2 à 0.8 est de 1.01 dB à 1.27 dB. Ces valeurs sont clairement plus faibles que les valeurs maximales, ce qui montre bien que la plupart du temps le gain est assez bas. Bien évidemment, cette méthode n'assure pas un gain inférieur à 1 pour chaque fréquence, ce qui peut toujours entraîner une saturation numérique lorsque  $f$  est très grand. Néanmoins, étant donné les fortes variations de gain imposées par la structure de ce bloc, il en est donc de la responsabilité de l'utilisateur de baisser le niveau d'entrée s'il entend une saturation à l'oreille (provoqué par la partie réverbération).

Le gain moyen qui a été retenu est de 1.93 dB (pour  $d = 0$  et  $f = 0.98$ ), soit un gain d'atténuation en sortie du filtre de -1.93 dB (0.8). Le nombre de bits supplémentaires au sein du bloc est alors de 1 bit.

#### 4.3.2 LPF : Low-Pass Filter

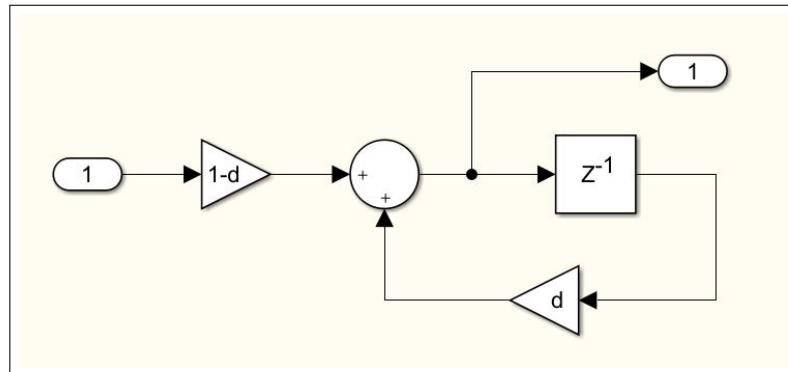


FIGURE 10 – Low-Pass Filter block diagram

Ce bloc est tout simplement un filtre passe-bas d'ordre 1 paramétrable par la valeur du **Damping**  $d$ . Pour  $d = 0$ , tout passe, tandis que pour  $d = 1$ , rien ne passe.

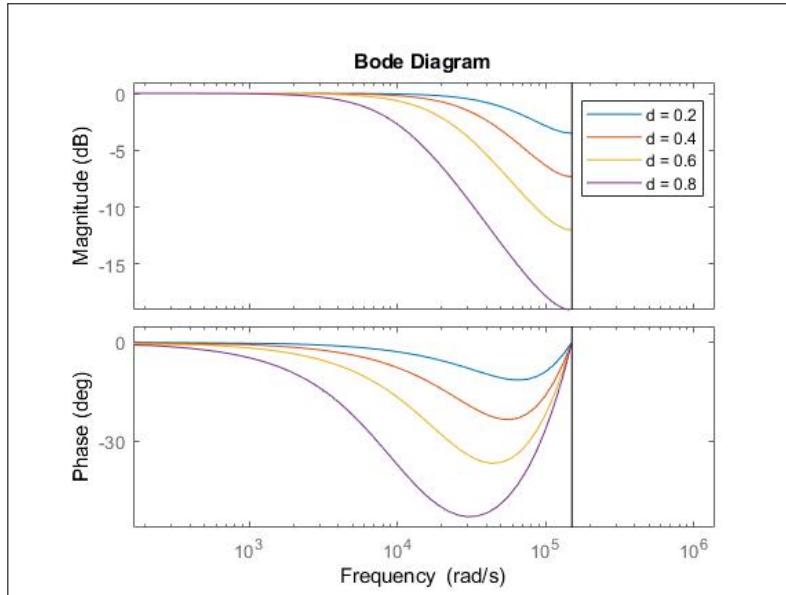


FIGURE 11 – Diagramme de Bode du bloc Low-Pass Filter en fonction du **Damping** d

Ce bloc ne nécessite aucun bit supplémentaire en raison des coefficients d et 1-d.

#### 4.3.3 APF : All-Pass Filter

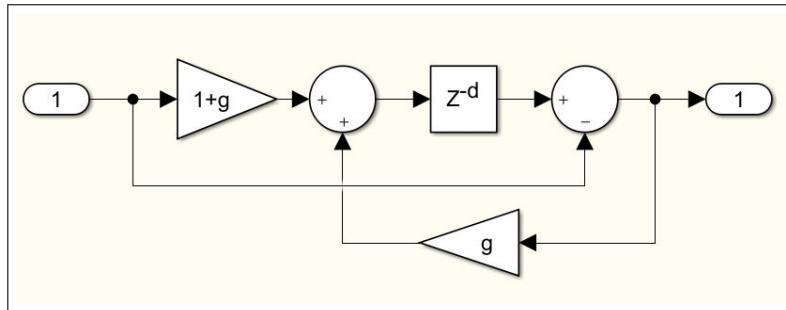


FIGURE 12 – All-Pass Filter block diagram ( $g = 0.5$ )

Ce bloc peut être interprété comme un bloc de diffusion de la réverbération. Plus la valeur du retard est importante, plus la réverbération semblera longue et donnera une impression de salle plus grande.

Il s'agit en réalité d'un "pseudo" All-Pass Filter puisque le gain n'est pas exactement constant pour une valeur de  $g = 0.5$ . C'est ce type de filtre qui est implémenté dans la Freeverb.

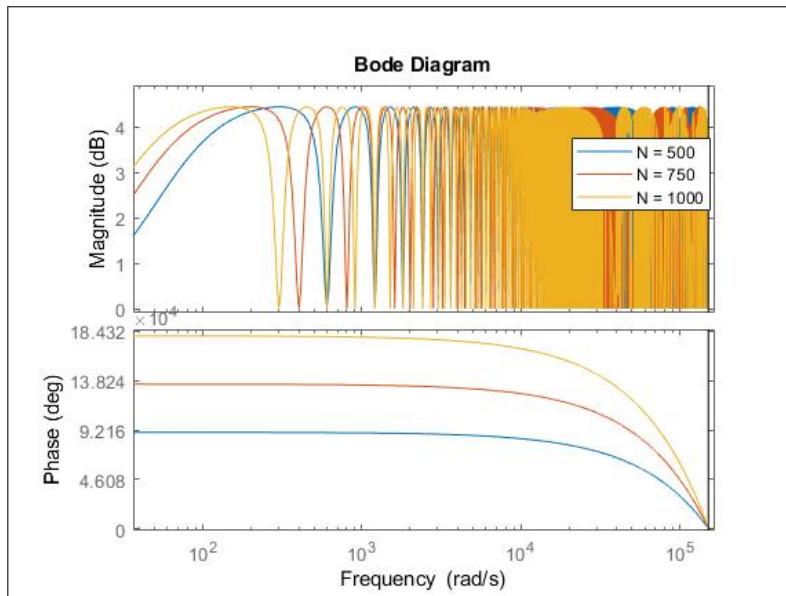


FIGURE 13 – Diagramme de Bode du bloc All-Pass Filter en fonction du retard N

Les mêmes commentaires que pour le filtre LFCF 4.3.1 sont valables. Cette fois-ci, le gain moyen ne dépend d'aucun paramètre. Le gain d'atténuation ajouté en sortie du filtre est de -1.42 dB (0.84).

En prenant en compte le gain moyen de 1.42 dB, le nombre de bits supplémentaires est de 2 au sein du bloc.

## 5 Implémentation d'une ligne à retard

### 5.1 Contraintes

Dans presque tous les blocs vus précédemment, il est nécessaire de décaler le signal d'entrée de plusieurs échantillons (retard  $z^{-N}$ ).

Pour ce faire, une première solution a été envisagée : utiliser des registres à décalage, ou autrement dit une succession de bascules D cadencées sur le signal audioL/R\_IN\_valid et ainsi former la ligne à retard. Le problème de cette méthode est qu'il faut beaucoup trop de registres pour stocker chaque échantillon, les blocs logiques du FPGA sont donc très vite tous utilisés.

La seconde méthode, celle retenue, consiste à effectuer des lectures/écritures dans des instances de blocs de RAM internes au FPGA, afin d'effectuer le décalage de la ligne entre deux audioL/R\_IN\_valid :

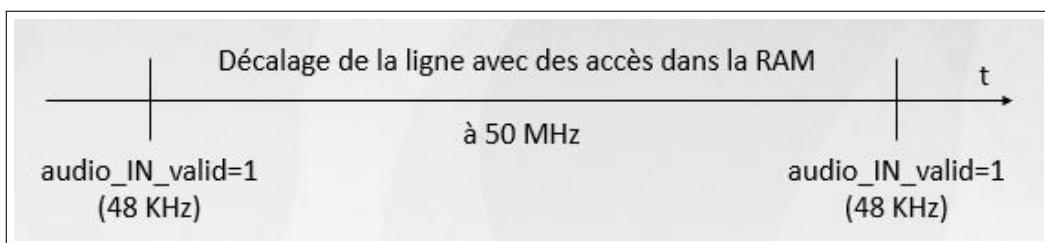


FIGURE 14 – Principe du décalage de la ligne à retard avec accès RAM

Si l'on considère qu'il faut autant de cycles à 50 MHz que la taille de la ligne pour effectuer un décalage complet (ce qui est presque le cas, voir 16), il est donc possible de générer une ligne à retard de 50M/48k échantillons maximum, soit 1041 au maximum. Une marge est définie et on choisira alors 1000 comme valeur maximale dans tout le projet.

Ceci dit, il sera toujours possible d'avoir un retard de plus de 1000 en associant en série plusieurs instances de ligne à retard.

### 5.2 Fonctionnement interne

Afin d'indiquer au synthétiseur d'instancier des blocs de RAM présents dans le Cyclone V, il faut écrire un module RAM respectant des règles de codage définies dans la documentation de Quartus. Le module créé correspond à une "Single clock dual-port RAM with Old Data Read-during-Write" (module le plus simple pour lire et écrire en même temps dans la mémoire à 50 MHz) :

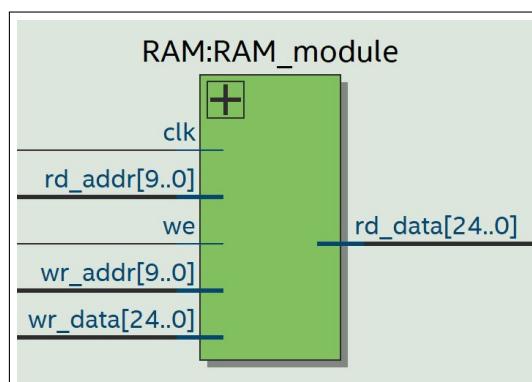


FIGURE 15 – Module "Single clock dual-port RAM with Old Data Read-during-Write"

Le chronogramme suivant détaille le fonctionnement du module VHDL gérant la ligne à retard (il instancie un module de RAM) :

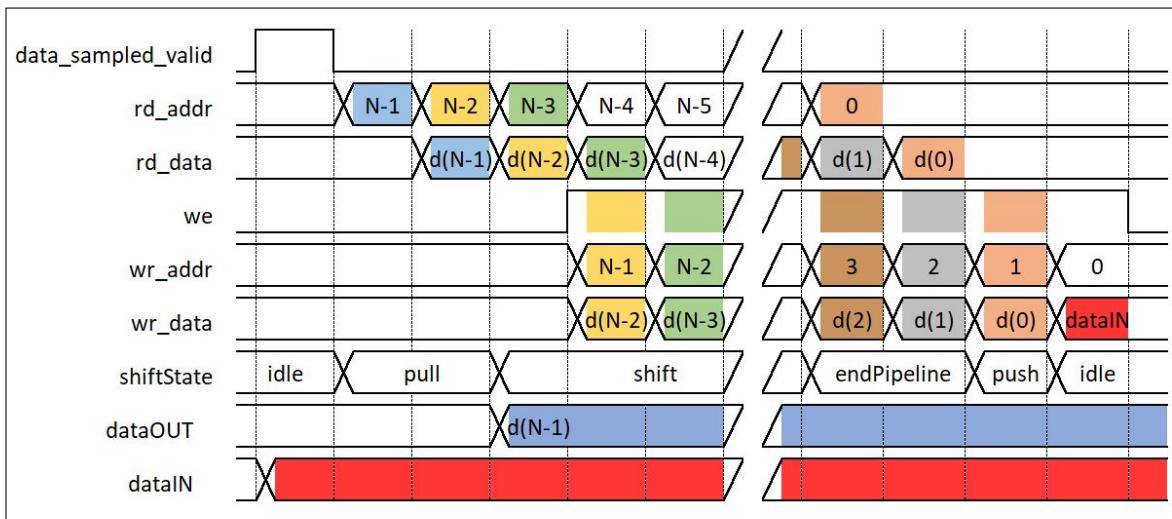


FIGURE 16 – Chronogramme de la ligne à retard de taille fixe N

La ligne à retard est décalée de gauche à droite, c'est à dire de l'adresse 0 à l'adresse N-1.

Ainsi les parties bleu correspondent à la sortie de la dernière valeur de la ligne (N-1), tandis que la partie rouge correspond à l'insertion de la valeur d'entrée au début de la ligne (0).

Entre les deux, la ligne est décalée en utilisant les accès à la RAM sous forme de pipeline, ce qui réduit considérablement le nombre de cycles nécessaires (à 50 MHz) pour décaler la ligne entière.

Chaque couleur correspond à : une lecture à l'adresse A, puis l'écriture de la valeur lue à l'adresse A+1.

Le nombre de cycles nécessaires sous cette implémentation est de N+3, au lieu de environ 3N pour une implémentation sans pipeline, ce qui est considérablement bénéfique. Cependant, une taille minimale de 4 est nécessaire pour le bon établissement du pipeline (pour que l'on puisse au moins rentrer dans l'état shift durant un cycle d'horloge).

Il est à noter que la réelle taille de la ligne à retard ne sera pas de N mais de N+1 car au moment où la donnée dataIN est prise en compte, la donnée dataOUT correspond toujours à l'ancienne valeur. Ce petit détail est géré dans le module pour que la valeur de la taille en entrée de celui-ci soit effectivement la bonne.

Une fois ce module implémenté, il a été modifié pour accepter une taille de ligne variable (signal d'entrée supplémentaire pour la taille de la ligne) tout en gardant une valeur générique de taille maximale (pour instancier correctement le module RAM). Ainsi, le retard pour le **Pre-delay** a pu devenir un paramètre variable en temps réel.

## 6 Partie logicielle

### 6.1 Interface utilisateur

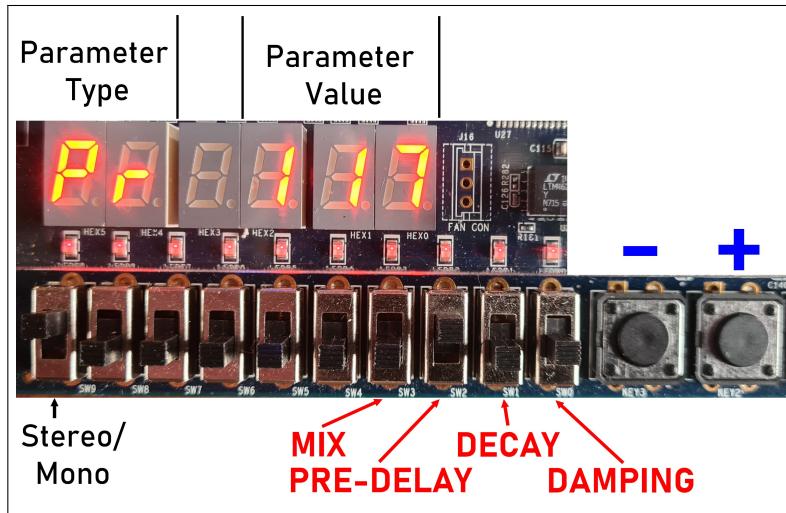


FIGURE 17 – Interface avec l'utilisateur

Tout d'abord, l'utilisateur choisit le paramètre à régler en utilisant les 4 switchs. Il suffit de mettre le bon switch en haut et les autres en bas pour sélectionner le paramètre voulu.

Une fois un paramètre sélectionné, l'utilisateur peut le modifier en appuyant sur les boutons poussoirs.

Afin d'indiquer à l'utilisateur la valeur actuelle du paramètre, les 6 afficheurs 7 segments sont utilisés ; les deux premiers affichent les deux premières lettres du paramètre sélectionné, les trois derniers affichent la valeur du paramètre. La valeur correspond à un pourcentage pour tous les paramètres sauf pour le **Pre-delay** où la valeur affichée correspond au retard en ms. Des valeurs minimales et maximales ont été définies pour chaque paramètre.

Le switch tout à gauche sélectionne le mode stéréo en haut et le mode mono en bas. En mode stéréo, la réverbération est ajoutée sur les deux canaux gauche/droite d'entrée. En mode mono, seul le canal de gauche (correspondant au seul canal des jacks TS mono) est pris en compte et les deux sorties gauche/droite renvoient le même signal de la voie gauche traité, ce qui est très utile pour un guitariste par exemple, ou un chanteur, souhaitant entendre sa source mono sur l'ensemble des enceintes gauche/droite en sortie.

Tout ceci (sauf le switch stereo/mono) est géré par le processeur en mode polling, étant donné que celui-ci n'a pas d'autres tâches à exécuter. Il permet de calculer les différents paramètres (**Mix**, **Pre-delay**, **Decay**, **Damping**) à partir du choix de l'utilisateur et de les communiquer sous la bonne forme à la partie FPGA via les IP PIO (*Parallel Input Output*) :

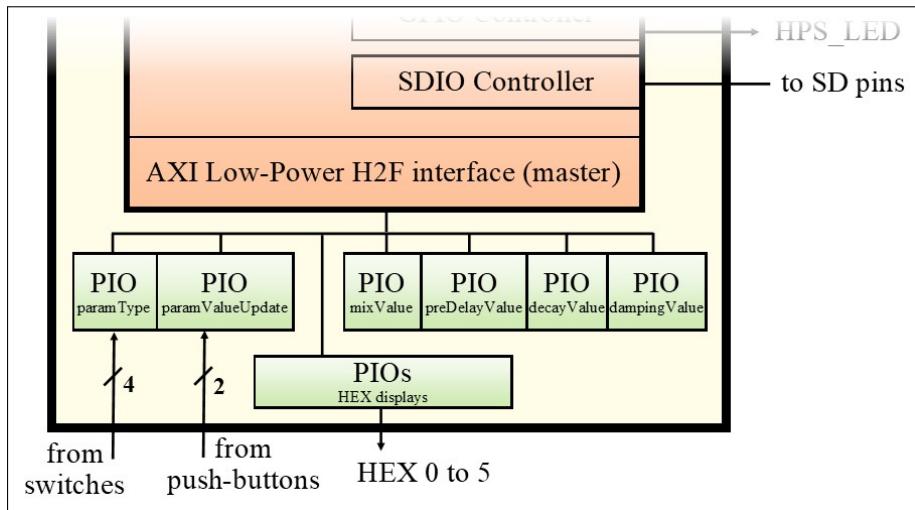


FIGURE 18 – Interface utilisateur + interface avec le traitement de la réverbération dans le FPGA

## 6.2 Chargement du programme sur carte SD

Tout d'abord, avant de pouvoir exécuter le programme sur le processeur intégré, il faut configurer le FPGA avec le design correspondant. Pour ce faire, le chargeur flash série EPCQ256 est utilisé afin de programmer le FPGA à chaque démarrage de la carte. Le mode de configuration du FPGA (switch MSEL de la carte DE1-SoC) doit être sur *Active Serial Configuration* correspondant à la valeur  $MSEL[4 :0]=10010$ . Le bistream pour programmer le FPGA est alors envoyé dans la flash du chargeur sous la forme d'un fichier JIC (*Jtag Indirect Configuration file*) sous Quartus Programmer.

Ensuite, il faut que le processeur puisse charger le programme dans la SDRAM (mémoire RAM utilisée par le HPS). Pour cela, on utilisera comme espace de stockage une micro SD (d'où l'utilisation d'un contrôleur SDIO, cf 18). Elle contiendra d'abord le preloader, qui viendra charger le programme, puis le programme lui-même.

Diverses séquences de boot sont possibles mais la suivante a été utilisée pour démarrer "simplement" un programme bare-metal sur le processeur. Le fonctionnement est le suivant : la bootROM du HPS permet de charger le preloader depuis la carte SD dans la OCRAM (*On-Chip RAM*) du HPS, puis celui-ci est exécuté dans le but d'initialiser la SDRAM (mémoire externe de plus grande capacité) et d'initialiser les périphériques du HPS. Il finit ensuite par charger l'image suivante présent sur la carte SD, à savoir dans notre cas le programme bare-metal dans la SDRAM.

Le preloader est généré grâce aux outils fournis par Altera SoC-EDS. Il suffit de spécifier le dossier *hps\_handoff* généré par la synthèse Quartus et regroupant la configuration matérielle de l'instance du HPS pour obtenir le preloader associé.

Maintenant, la dernière étape consiste à mettre les images binaires sur la carte SD. Une partition de type 0xA2 (partition inconnue sur la plupart des systèmes d'exploitation) doit être créée, par exemple avec l'utilitaire fdisk sous linux.

Ensuite il faut copier l'image "preloader-mkpimage.bin" au début de cette partition puis copier l'image "app-mkimage.bin" à l'offset 0x40000 (soit à l'octet 262144) de cette partition, avec la commande dd sous linux par exemple.

En s'assurant que les pins BSEL sont bien configurés sur 101 (configuration par défaut sur la carte DE1-SoC), le programme doit se lancer correctement. Un cold reset (bouton présent sur la carte) permet de relancer toute la procédure de démarrage.

## 7 Gestion du projet

Un dépôt GitHub a été utilisé pour gérer l'avancement du projet : <https://github.com/vvincent02/reverbFPGA>

Une vidéo finale de démonstration du système a également été tournée afin de valider son bon fonctionnement.

## Références

- [1] [https://ccrma.stanford.edu/~jos/pasp/Artificial\\_Reverberation.html](https://ccrma.stanford.edu/~jos/pasp/Artificial_Reverberation.html), *Artificial Reverberation - Center for Computer Research in Music and Acoustics - Stanford*