PLATEFORME VERSATILE D'EXPERIMENTATION AUDIO : TRAITEMENT ET SYNTHESE NUMERIQUE

L. Barrandon
Institute of Microelectronics and Wireless Systems, National University of Ireland - Maynooth
lbarrandon@eeng.nuim.ie

S. Crand
Institut d'Electronique et de Télécommunications de Rennes
samuel.crand@univ-rennes1.fr

RÉSUMÉ

Nous présentons une plateforme électronique dédiée à la mise en œuvre de traitements numériques audio en temps réel. Ce système modulaire permet de numériser jusqu'à 8 signaux audio mono provenant de différentes sources, et de les restituer, après traitements, sur 8 sorties. Le caractère reprogrammable de son élément principal, un FPGA[1], permet d'utiliser cet outil à des fins d'exploration concernant le traitement et la synthèse du son. Il peut donc être employé aussi bien dans un cadre pédagogique, qu'expérimental et artistique.

Deux aspects principaux sont développés dans ce document. Une présentation générale de la constitution matérielle du prototype, de son fonctionnement et de sa mise en œuvre permet d'apprécier la versatilité de cet outil. Partant de cette base, quelques applications concrètes sont proposées et permettent une initiation pratique et progressive au traitement numérique du son suivant les différents niveaux hiérarchiques définis : le niveau opérateur définit les outils arithmétiques et les manipulations élémentaires disponibles, le niveau fonction met en œuvre les opérateurs afin de produire des effets et outils de synthèse, le niveau application vise à personnaliser la plateforme pour une finalité précise : l'interface homme-machine (paramétrage et couplage au geste) ainsi que les paramètres ainsi contrôlés sont programmables à volonté.

1. INTRODUCTION

Cet article présente la première phase d'un projet intitulé « Développement d'une plate-forme de prototypage pour applications audio » [1]. Ce projet s'inscrit dans la formation Master électronique préparée à l'université de Rennes 1. Cette plate-forme sera enrichie d'année en année dans le cadre de projets effectués par des étudiants de Master 1 ou de Master 2.

L'objectif final de ce projet était de mettre en œuvre une plate-forme reconfigurable dédiée à des applications de traitement du signal audio. Outre le fait que le développement de cette plate-forme est assuré par des étudiants, celle-ci sera aussi un support pour l'enseignement. En effet, cette plate-forme matérielle sera utilisée pour mettre en pratique les concepts et les méthodologies de conception de systèmes numériques ainsi qu'une mise en œuvre des principes du traitement du signal à travers la manipulation du son.

La section suivante présente la plateforme développée et donne une vue d'ensemble de ses possibilités opérationnelles au travers de sa conception matérielle, de ses performances intrinsèques, et des différentes étapes nécessaires à sa mise en œuvre.

La troisième partie décrit des applications possibles utilisant cette carte par une approche concrète, hiérarchique et évolutive répondant à d'éventuels besoins pédagogiques, les étudiants étant capables de constituer leur propre synthétiseur et de personnaliser leurs effets sonores en temps réel.

Nous aborderons finalement les perspectives ouvertes par cette approche et les différents objectifs fixés afin d'enrichir notre projet.

2. LA PLATE-FORME

Cette partie présente les caractéristiques physiques du prototype (conception, performances) et explique les différentes opérations nécessaires à son utilisation.

2.1. Principe

L'industrie de l'électronique propose depuis quelques années des composants de traitement numérique flexibles, performants et abordables, aussi bien financièrement que du point de vue de leur prise en main. D'une part, les FPGA, de part leur capacité à accéder et à manipuler de façon transparente des données binaires donnent une grande liberté aux concepteurs. Ils nécessitent l'utilisation d'un ensemble d'outils logiciels dédiés dont nous parlerons rapidement par la suite, et la description de l'application à l'aide de langages dits de description matérielle tels que VHDL [2, 3]. Les exemples donnés dans le troisième paragraphe seront écrits à l'aide de ce langage dont la syntaxe et l'utilisation sont très abordables tout en permettant la programmation d'algorithmes complexes.

L'objectif de la 1ère phase de ce projet était donc de développer une plate-forme de prototypage pour applications audio. Pour ce faire, nous avons sélectionné un CODEC audio, conçu et réalisé les cartes électroniques nécessaires, et créé l'interface de contrôle du système afin d'initialiser la plateforme et de rendre son utilisation transparente.

D'un point de vue applicatif, ce système permet de numériser jusqu'à 8 signaux audio mono provenant de

[1] FPGA : les « Field Programmable Gate Array », ou réseaux de portes programmables, constituent une famille de composants électroniques numériques apparue dans les années 90.

différentes sources (instruments de musique, lecteur MP3, oscillateurs…), et de les restituer, après traitements, sur 8 sorties (par exemple des hauts parleurs pré amplifiés, une table de mixage...). Il est donc possible de traiter des données audio, et éventuellement mixer différentes sources de signaux audio. La qualité d'écoute est de fait un critère déterminant dans la conception de notre système. Cet aspect pourra être pris en compte lors de la caractérisation du module « mise en forme » principalement constitué de filtres analogiques.

2.2. Conception matérielle

La figure 1 présente l'organisation de la plate-forme. Ce projet s'articule autour de la carte de prototypage SPARTAN-3 Starter Board de la société DIGILENT [4] et de deux cartes réalisées pour notre application.

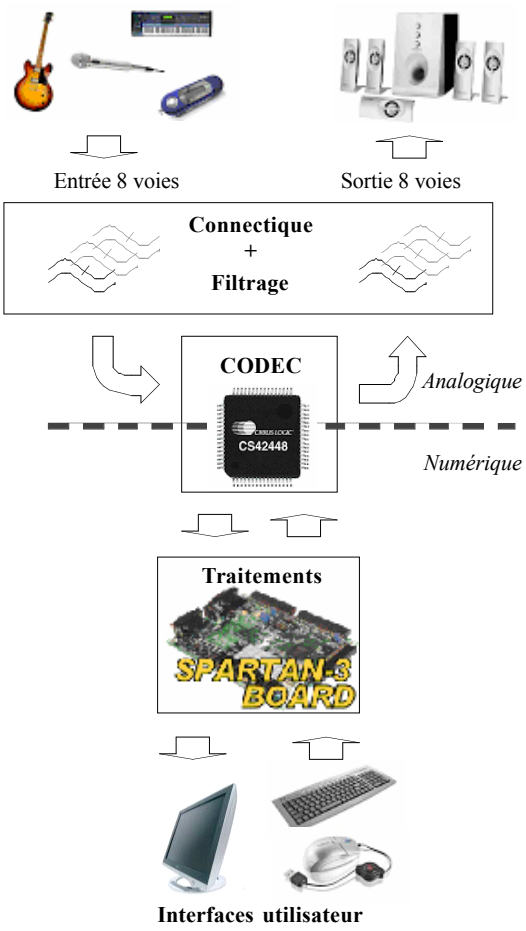

Figure 1. : Synoptique de la plate-forme audio.

L'utilisateur peut agir en temps réel sur l'application par le biais des 4 boutons poussoirs et des 8 interrupteurs présents sur la carte. Afin d'étendre l'ergonomie, le port PS2 permet de brancher soit un clavier, soit une souris dont les données peuvent être interprétées par l'application. Par exemple, la position de la souris est codée par deux nombres sur 8 bits. Ces deux nombres peuvent par exemple représenter de manière relativement directe un volume sonore ou une fréquence d'oscillation.

Ci-dessous, une photographie du prototype indiquant l'encombrement total du dispositif actuel (carte de filtrage 23x 20 cm, carte Codec 10 x 7cm.

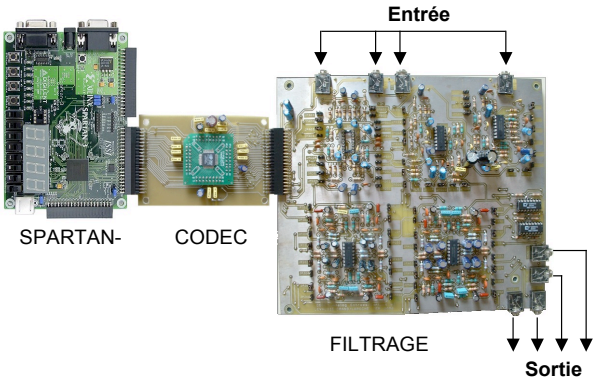

Figure 2. Présentation des 3 cartes.

2.2.1. Carte d'acquisition et de restitution audio

La carte dite « carte CODEC » est constituée du CODEC AUDIO CS42448-CQZ de la société CIRRUS LOGIC [5]. Ce composant permet d'acquérir jusqu'à 8 entrées mono (soit 4 entrées stéréo) et 8 sorties mono (soit 4 sorties stéréo). Il possède 3 convertisseurs analogique/numérique (CAN) stéréo, dont un pouvant être multiplexé en interne pour gérer 4 entrées mono (2 voies stéréo), et 4 convertisseurs numérique/analogique (CNA) stéréo indépendants. La fréquence d'échantillonnage des convertisseurs est comprise entre 32 KHz et 192 KHz et leur résolution est de 16 à 24 bits (voir table 1). Ces paramètres peuvent être sélectionnés par l'utilisateur en cours de fonctionnement.

Le CODEC est constitué de deux parties principales : le port de *contrôle* et le port de *données*. Des fonctions de configuration, d'ajustement de volume en entrée et en sortie sont accessibles directement par le *port de contrôle*. L'acquisition et l'envoi d'échantillons audio sont assurés par le *port de données* par une liaison série. Au lieu d'utiliser un microcontrôleur pour le contrôle et un DSP pour le traitement des échantillons audio, nous avons choisi un FPGA. Les avantages d'une telle utilisation sont d'une part que le FPGA permet de remplir ces deux rôles et d'autre part d'illustrer l'enseignement portant sur l'implantation matérielle d'applications décrites à haut niveau.

	Min	Max
Entrées	1 voie	8 voies mono
Sorties	1 voie	8 voies mono
Fréquence échant.	32	192 kHz
Résolution	16	24 bits

Table 1. Performances d'acquisition/restitution.

2.2.2. Carte de traitement et de configuration

La carte Xilinx Spartan-3 [6], avec son FPGA associé et ses ports périphériques est tout à fait adaptée à ce type de plate-forme de prototypage[1]. Elle nous permet de configurer et de piloter le CODEC audio tout en effectuant des traitements numériques sur les échantillons audio, grâce au FPGA. De plus, les liaisons VGA et série permettent l'implantation d'une interface homme/machine facilitant l'ergonomie de ce type de plate-forme. Cette interface permet à la fois d'interagir sur le fonctionnement de la plate-forme en assurant la configuration, la génération de signaux audio, le suivi des traitements et la récupération de données traitées.

Fréquence d'horloge	50 MHz
Ressources logiques	200 000 portes
Multiplieurs 18b x 18b	12
Mémoire FPGA	216kbits
Mémoire externe	1Mo

Table 2. Caractéristiques de la carte de traitement.

Les ressources logiques dites « distribuées », relatives aux 200 000 portes peuvent prendre en charge tout type d'opération numérique élémentaire. Des opérations plus complexes, telles que la multiplication ou la fonction mémoire, peuvent être mises en œuvres en utilisant des blocs « précâblés » du FPGA. L'intérêt d'avoir des multiplieurs et de la mémoire dédiés consiste à optimiser le système tout en économisant les ressources logiques distribuées.
Environ 3% des ressources sont utilisés pour l'interfaçage (port souris) et la configuration du CODEC.

2.2.3. Carte d'interface connecteur et filtrage.

La carte dite de filtrage intègre les connectiques audio, des filtres actifs et passifs (anti-repliement et adaptation d'impédance) sur chacune des entrées et des sorties, œ qui explique la taille de cette carte. Le choix entre les deux types de filtres est assuré par des cavaliers. Nous avons choisi de séparer le CODEC audio des filtres et connectiques pour pouvoir augmenter la modularité de notre plate-forme de prototypage et faciliter son utilisation dans un cadre pédagogique. En effet, la carte de filtrage pourra être utilisée en licence 2 et/ou licence 3 afin d'illustrer l'enseignement sur les filtres passifs et actifs. L'intérêt est de pouvoir caractériser simplement les filtres en leur fournissant un signal audio et en connectant les sorties à des haut-parleurs pré amplifiés.

2.3. Mise en œuvre, utilisation

Toutes les opérations d'initialisation et de configuration du CODEC, de récupération et de restitution des données sont les premières fonctions déjà réalisées sur la base de ce système et sont organisées selon le schéma présenté sur la figure suivante :

[1] La société Altera propose des produits comparables, et le site [7] peut être consulté. Les logiciels proposés sont également comparables en première approximation. Les codes VHDL et les principes fondamentaux sont donc transposables sans distinction de marque.

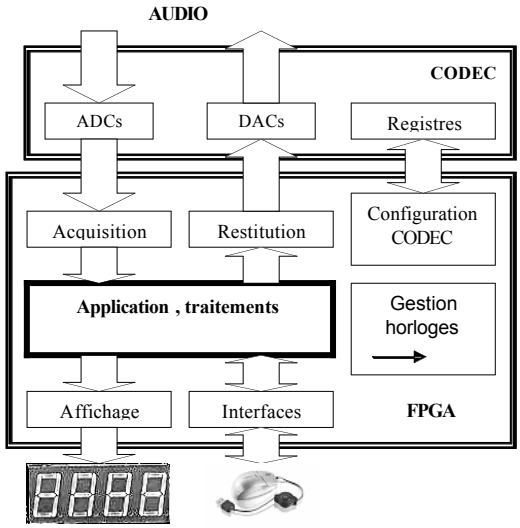

Figure 3. Organisation fonctionnelle de la plate-forme.

La suite de cet article sera consacrée aux fonctions qui peuvent être intégrées au bloc nommé « application, traitements ». Nous allons nous intéresser de plus près aux étapes nécessaires à la mise en œuvre de la plate-forme :

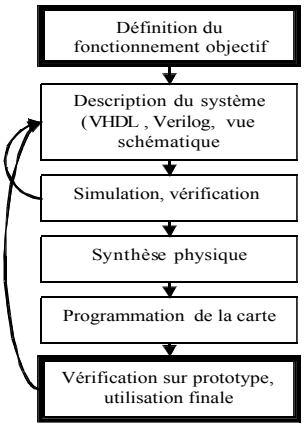

Figure 4 Flot de développement d'une application sur FPGA

La première étape de tout projet est la définition du fonctionnement à atteindre. L'objectif est de décrire de façon matérielle, mathématique et algorithmique les opérations qui devront être réalisées. Cette description est ensuite formalisée dans un langage informatique puis le fonctionnement global vérifié par simulation comportementale. Une fois cette étape validée, l'opération de synthèse logique et physique peut être effectuée. Elle consiste à utiliser un logiciel permettant de traduire la description du fonctionnement d'un système en une association de fonctions logiques élémentaires. Elle aboutit à la génération d'un fichier de configuration du FPGA qui est envoyé sur la carte de traitement typiquement via un câble parallèle ou USB. Si le résultat final n'est pas satisfaisant, il faut identifier l'erreur ou l'imperfection en corrigeant la description du système.

Un des intérêts considérables de cette approche matérielle, c'est-à-dire basée sur l'utilisation d'un prototype, plutôt qu'une approche purement simulatoire, est le gain de temps au niveau de l'exploration de l'espace de conception. En effet, dès lors qu'une application fonctionne mais nécessite des améliorations, on peut se passer de faire des simulations coûteuses en temps, il suffit donc de configurer le FPGA et de lui fournir du son pour valider ou décider de modifier l'application considérée.

3. APPLICATIONS

3.1. Une approche multi niveau

Dans l'approche que nous proposons, afin de tirer parti de la plate forme, nous allons distinguer différentes couches hiérarchiques dans le but de découper les problèmes et d'aborder le prototypage de traitement du signal en temps réel de manière progressive :

- le niveau *opérateur* définit les outils arithmétiques de base ainsi que les manipulations élémentaires disponibles ;
- le niveau *fonction* met en œuvre les opérateurs afin de produire des effets et des outils de synthèse du son ;
- le niveau *application* vise à personnaliser la plateforme pour un cadre d'utilisation précis.

Dans la suite de cette section, différents exemples, loin d'être exhaustifs, sont proposés et explicités pour chaque niveau, par ordre de complexité croissante.

3.2. Niveau opérateur

Les opérateurs, tels que l'addition, la multiplication ou la mémoire, constituent le niveau de base en termes de conception de traitements du signal. Ce niveau va nous permettre d'illustrer certains principes élémentaires et d'aborder leur utilisation au sein du prototype audio.

Trois exemples sont détaillés ici : la balance, le trémolo et la distorsion.

3.2.1. Le tremolo

Le principe du tremolo consiste a moduler un son en multipliant celui-ci par un oscillateur basse fréquence [2 chap. 30].

Figure 5. Schéma d'un tremolo (source: [2]).

En VHDL, cela peut se traduire par le code suivant :

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

ENTITY tremolo IS
PORT(
  entree:IN STD_LOGIC_VECTOR(15 DOWNTO 0);
  trem:IN STD_LOGIC_VECTOR(15 DOWNTO 0);
  sortie:OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END tremolo;

ARCHITECTURE exemple OF tremolo IS
  SIGNAL mult: STD_LOGIC_VECTOR(31 DOWNTO 0)
BEGIN
  mult <= entree * trem;
  sortie <= mult(31 DOWNTO 16);
END exemple;
```

Tout comme les exemples qui vont suivre, les mots clés du VHDL sont indiqués en lettres majuscules. L'ENTITY représente l'interface d'un bloc, c'est-à-dire ses entrées et sorties, et l'ARCHITECTURE définit le comportement ou la structure du bloc. STD_LOGIC désigne basiquement un bit et STD_LOGIC_VECTOR un signal numérique de 16 bits dont les bits sont numérotés, dans cet exemple, entre 15 et 0.

`Entree` est un signal mono numérisé sur 16 bits, c'est-à-dire prélevé directement sur l'une des sorties du CODEC après désérialisation. Ce signal est envoyé sur `Sortie` après multiplication par la valeur de `trem`.

`Mult` est un signal intermédiaire recevant le résultat d'une multiplication, c'est-à-dire des données dont la taille, en bits, est la somme des tailles des opérandes (soit 32 bits). Il faut donc le redimensionner avant de l'associer à la sortie 16 bits.

Le signal `trem` peut être un LFO (de l'ordre de quelques Hertz) branché sur une entrée mono analogique de la carte, aussi bien qu'une source numérique générée à l'intérieur du FPGA, l'amplitude et la valeur moyenne de ce signal modulant devant être adaptées à l'effet souhaité.

Un exercice intéressant peut consister à partir de ce système à réaliser un modulateur en anneau.

3.2.2. Balance

La balance consiste à associer une amplitude relative à chaque sortie audio. Le système de la figure 6 utilise une source unique (`input`), répartie sur deux sorties. La sortie d'un oscillateur que nous nommerons « `bal` » pilote l'amplitude associée à chaque sortie.

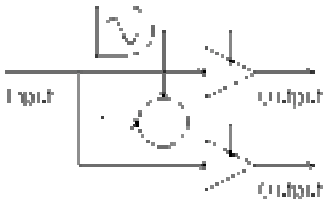

Figure 6. Schéma d'une balance (source: [7] chap. 30).

Ce traitement est très proche de l'exemple précédent du point de vue de son implémentation et consiste à changer la gamme de fréquences et éventuellement la nature signal modulant. Afin de générer le signal sur la sortie « `output` » à laquelle est affectée `input*(1-bal)`, une astuce simple et efficace consiste à écrire la ligne VHDL suivante :

```
Mult <= Input * not(bal) ;
```

`Not(bal)` étant le complément à 2 de `bal`, et `Mult` un signal intermédiaire suivant le principe du code du tremolo.

Tout comme dans l'exemple précédent on peut envisager de remplacer le LFO par une source extérieure, ou de moduler la valeur de bal en temps réel grâce à la souris, comme évoqué dans la partie 2.

Rappelons que la plate-forme dispose de 8 sorties mono qui peuvent être mises à profit. Il est donc possible, par exemple, d'employer trois signaux équivalents à `bal` et répartir le son selon les trois dimensions sur 6 ou 8 haut-parleurs.

3.2.3. Saturation, distorsion

La saturation et la distorsion visent à déformer un signal (et peuvent être parfois des effets indésirables…).

Voici un exemple de code permettant de saturer un signal en limitant volontairement son amplitude.

```
{les appels à bibliothèques sont identiques}
ENTITY saturation IS
PORT(
  entree:IN STD_LOGIC_VECTOR(15 DOWNTO 0);
  sortie:OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END saturation;

ARCHITECTURE exemple OF saturation IS
  CONSTANT   sat:STD_LOGIC_VECTOR(15   DOWNTO   0)   :=
"0100000000000000";
BEGIN
PROCESS(entree)
BEGIN
  IF entree > sat THEN
    sortie <= sat;
  ELSIF entree < -sat THEN
    sortie <= -sat;
  ELSE
    Sortie <= entree;
  END IF;
END PROCESS;
END exemple;
```

Ce code spécifie un bloc permettant de limiter le signal d'entrée entre les valeurs définies par `sat` et `-sat`. Une fois de plus, cette valeur peut être une donnée externe sélectionnée par boutons poussoirs ou définie avec la souris.

Suivant un principe plus générique mais partant de la même idée, de la mémoire doit être utilisée dans le but de faire du « waveshaping », aussi appelé distorsion non-linéaire ou processus non-linéaire. Au lieu de définir trois zones dans la définition de la fonction d'entrée/sortie du code précédent, à chaque code binaire de l'entrée numérisée correspond une case mémoire dont le contenu est attribué à la sortie.

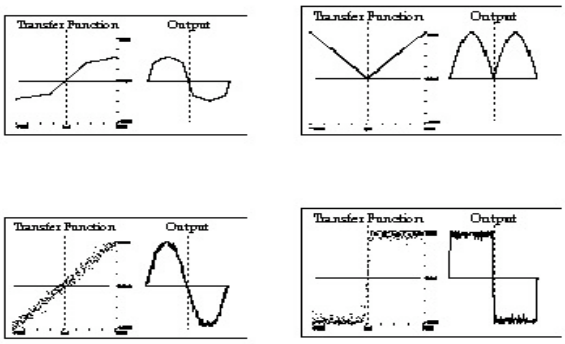

Figure 7. Exemples de waveshaping d'une sinusoïde [2].

Voici le code VHDL pouvant répondant au principe défini :

```
{le bloc ENTITY est identique au précédent}

ARCHITECTURE exemple OF distortion IS
  TYPE   waveshp   IS   ARRAY   (0   TO   2^16-1)   OF
STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT   tab:   waveshp:=   ("1111001100110101l0",   {…),
"0111001100110101l0");
BEGIN
  Sortie <= tab(conv_integer(entree)) ;
END exemple;
```

La fonction `conv_integer` permet d'interpréter le signal d'entrée (qui est un vecteur de bits) comme un entier, ce dernier étant associé au contenu de la case mémoire correspondante dans le tableau `tab`, conformément à la figure 7.

3.3. Niveau fonction

Le niveau fonction, comme il a déjà été évoqué, peut se définir comme l'association de différents opérateurs afin de réaliser un traitement plus complexe. Deux exemples, l'écho et le filtrage sont traités dans cette partie. D'autres effets tels que le vibrato, la compression,… auraient également pu avoir leur place ici.

3.3.1. Echo

L'écho est typiquement un effet utilisant une ligne à retard, ce qui se traduit en électronique numérique par de la *mémoire*. Le signal est donc transmis de manière directe de la source vers la sortie, et *additionné* avec une ou plusieurs versions de lui-même retardées et filtrées (ou du moins atténuées, c'est-à-dire *multipliées* par un coefficient).

L'exemple suivant présente un écho dont l'amplitude de l'entrée est divisée par 2 à chaque passage dans une ligne à retard dont le nombre de point mémoire est `delay`, soit 4410, c'est-à-dire un dixième de seconde à une fréquence d'échantillonnage de 44100 Hz.

```
{le bloc ENTITY est identique au précédent}

ARCHITECTURE exemple OF echo IS
  CONSTANT delay : integer := 4410;
```

```
  TYPE mem IS ARRAY (0 TO delay-1) OF
STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL ligne:mem;
BEGIN
PROCESS(entree)
BEGIN
  Sortie <= ligne(delay-1);
  FOR i IN delay-1 DOWNTO 0 LOOP
    ligne(i) <= ligne(i-1);
  END LOOP ;
  ligne(0) <= entree/2 + ligne(delay-1)/2 ;
END exemple;
```

Des raffinements de cette exemple basique peuvent être menés dans le développement d'effets tels que le « slapback echo », « multitap echo » avec utilisation de la stéréo ou éventuellement de plus de 2 voies.

3.3.2. Filtrage FIR

Le filtrage est très utilisé dans le traitement du son et il paraît pertinent d'aborder cet question afin d'appréhender certaines problématiques liées à l'utilisation des FPGA.

Le code VHDL basique d'un filtre à réponse impulsionnelle finie (RIF) est proposé sur la colonne suivante. Celui-ci correspond à l'architecture directe bien connue présentée sur la figure 8 et au gabarit de filtrage indiqué sur la figure 9, mais il peut être modifié en recréant la liste (en binaire ou hexadécimal) des coefficients indiqués ici sous le nom `coef`, c'est-à-dire en modifiant leur nombre et leur valeur. Ce type de filtre utilise essentiellement les opérateurs multiplication et addition. Des points mémoire, représentés par les opérandes de type VARIABLE, sont également nécessaires à la sauvegarde des calculs intermédiaires. Toutes les opérations sont synchrones à l'horloge `clk`.

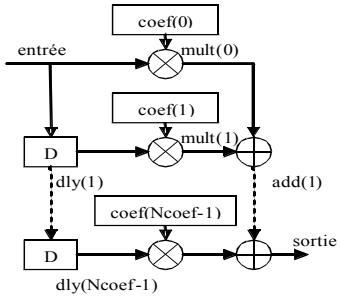

Figure 8. Schéma bloc d'un filtre FIR.

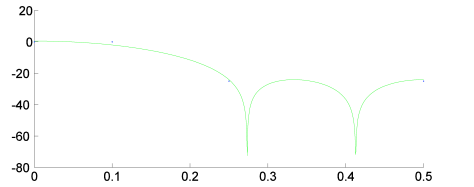

Figure 9. Fonction de transfert en dB du filtre proposé en fonction des fréquences normalisées.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

ENTITY fir_dir IS
PORT(
  clk:IN STD_LOGIC;
  entree:IN STD_LOGIC_VECTOR(15 DOWNTO 0);
  sortie:OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END fir_dir;

ARCHITECTURE exemple OF fir_dir IS

TYPE mem IS ARRAY (0 TO 4) OF STD_LOGIC_VECTOR(15
DOWNTO 0);
SIGNAL dly:mem;
CONSTANT coef : mem := ("0011001100110101",
"0110011001101000", "0111111111111111",
"0110011001101000", "0011001100110101");

BEGIN
fir : PROCESS(clk)
  VARIABLE add:mem;
  VARIABLE mult:mem;
  VARIABLE vmult : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
IF rising_edge(clk) THEN
  sortie<=add(Ncoef-1);
  FOR i IN 0 TO Ncoef-1 LOOP
    vmult:=coef(i)*dly(i);
    mult(i):=sxt(vmult(31 DOWNTO 17),16);
  END LOOP;
  add(0):=mult(0);
  FOR i IN 1 TO Ncoef-1 LOOP
    add(i):=ext((add(i-1)+mult(i)),16);
  END LOOP;
  FOR i IN 1 TO Ncoef-1 LOOP
    dly(i)<=dly(i-1);
  END LOOP;
  dly(0)<=entree;
END IF;
END PROCESS fir;
END exemple;
```

Ce code peut être adapté en fonction de la réponse de filtrage désirée. Une attention toute particulière doit être apportée à l'aspect « matériel » issu de ce code et qui peut apporter quelque éclairage quand à l'utilisation des FPGA. En effet, ce filtre comprenant 5 coefficients, 5 multiplieurs, sur les 12 disponibles (cf. table 2) sont mis en œuvre. Cela fait une proportion importante, d'autant plus que ce filtre présente une sélectivité très faible. Cependant, le logiciel de conception utilisé pour la synthèse de ce filtre estime à 80 MHz la fréquence d'horloge maximale utilisable, soit plus de 1000 fois les fréquences utilisées typiquement pour les débits audio. Il faut donc avoir à l'esprit qu'il est possible d'optimiser l'algorithme de filtrage afin d'associer un seul multiplieur à plusieurs dizaines voire quelques centaines de coefficients. L'architecture MAC (Multiplier / ACumulator), fournie par le constructeur [6], permet de répondre à ce besoin. Elle peut être utilisée en tant que boîte noire, de même que de nombreux systèmes de traitement optimisés (Transformée de Fourier Rapide – FFT, filtres CIC, algorithme CORDIC…), il n'est donc pas nécessaire ici de rentrer dans les détails concernant ces aspects, l'idée principale étant l'initiation.

3.4. Niveau application

Les principes essentiels de traitements du son dans le cadre de notre projet ont été définis. Le niveau application va consister à assembler différentes fonctions et des opérateurs afin de créer une plate-forme adaptée à une application donnée.

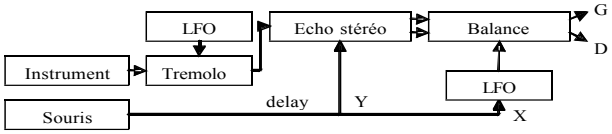

Figure 10. Vue schématique d'un exemple d'application.

Les logiciels de conception proposent un mode schématique qui donne la possibilité de définir une application en utilisant des fonctions sous forme de boîtes noires. Il suffit donc pour créer un système de récupérer des blocs dans une bibliothèque prédéfinie, de connaître l'influence des différents paramètres et la relation entrée(s)/sortie(s) et d'assembler ces fonctions. Signalons que le flot mis en œuvre est toujours conforme à ce qui est présenté sur la figure 4.

4. PERSPECTIVES

4.1. Mise en place de travaux pratiques et de projets

Des travaux pratiques, éventuellement par binômes, sur la base de cette plate-forme peuvent être mis en place de façon relativement aisée. Il nécessite, pour un poste, le prototype de traitement audio, un ordinateur et le logiciel ISE, proposé gratuitement sur [6]. Cette version est apparemment limitée, mais elle est utilisable pour le FPGA de la carte sans restriction.

La constitution d'une bibliothèque comme évoqué plus haut, sur la base des exemples proposés, paraît un préliminaire relativement incontournable afin de se familiariser avec les outils de développement dans un premier temps, sans avoir à apprendre un langage type VHDL d'emblée.

4.2. Recherche, création

Dans un contexte de performance artistique, ce type d'approche implique l'artiste dans la création même de son instrument ou de ses effets. L'aspect d'exécution en temps réel paraît essentiel à ce cadre-là. La modularité de la plateforme ainsi que ses différents type d'accès laissent une grande liberté quant à la mise en place d'interaction entre l'interprète et l'instrument.

Pour la recherche sonore et acoustique, nous avons ici un outil supplémentaire d'expérimentation. Dans une perspective large de modélisation et d'expérimentation de physique acoustique, notons que les traitements implantés dans le FPGA peuvent être assimilés à leur environnement par le biais de description de phénomènes analogiques. Les langages Verilog-A et VHDL-AMS [3], extensions du Verilog et du VHDL à la modélisation en temps continu, sont des leviers puissants permettant cette approche-là. Les traitements numériques sont alors un des maillons d'un ensemble de phénomènes physiques.

4.3. Utilisation d'une plateforme de traitement hybride

Une version compacte du prototype présenté sur la figure 2 est en cours de réalisation. Cette nouvelle plateforme présentera les mêmes caractéristiques et des performances identiques avec un encombrement réduit. Une surface totale de l'ordre de 30 à 50% de la première version devrait être atteinte grâce à l'utilisation de composants montés en surface (CMS).

Une autre carte de traitements plus puissante peut avantageusement remplacer celle que nous avons proposée ici. Par exemple, la carte XUPV2P [4] utilise des FPGA Virtex 2 Pro [6] proposant des processeurs internes. Il serait donc possible ici de bénéficier de la flexibilité inhérente aux FPGAs tout en ayant la puissance de calcul nécessaire à certains algorithmes complexes, comme par exemple des calculs matriciels, ou des opérations devant être effectuées en virgule flottante. Nous entrons ici dans un domaine qui nécessite des compétences électroniques et informatiques dépassant le pur cadre pédagogique initialement associé à cet article et concerne plutôt la mise en place de projets spécifiques. Il est cependant intéressant voire indispensable de se pencher sur ces aspects pour ceux qui veulent suivre cette voie au-delà de ce que nous présentons ici, mais cela nécessite par ailleurs une licence payante du logiciel ainsi que le logiciel EDK [6] pour l'utilisation de la partie processeur. Une plateforme présentant un ou plusieurs processeurs permettrait d'approcher les performances offertes par des outils informatiques [8, 9]. Il y a de fait une convergence sensible entre les dispositifs électroniques autonomes et les logiciels de synthèse et de traitement. Ce dispositif, inspiré à l'origine par les effets pour guitare du commerce, dont [10] est un exemple de multi-effet reprogrammable via connexion USB, pourrait donc présenter des fonctionnalités comparables mais avec des solutions d'interfaces audio et numériques plus flexibles.

5. CONCLUSION

Le système de traitement sonore reprogrammable présenté est un outil d'exploration temps réel pour la traitement ou la synthèse du son. Nous avons vu le fonctionnement et les performances de cette plateforme et quelques exemples simples permettant d'en illustrer les possibilités.

D'un point de vue électronique, les deux extensions envisagées pour ce projet sont la réduction de l'encombrement de la plateforme et l'utilisation d'une autre carte de traitements numériques plus puissante.

Le côté applicatif quant à lui peut bénéficier d'une souplesse d'utilisation par la constitution d'une bibliothèque d'opérations et de fonctions afin d'aborder sa mise en œuvre au niveau application.

6. REFERENCES

[1] Laronche, D., Barrandon, L., Crand, S., Lelu, H., Lecomte, S., Mallet, B., Levallois, D. "Développement d'une Plate-forme de Prototypage pour Applications Audio", Actes des 9èmes Journées Pédagogiques du CNFM Saint-Malo, France, 2006.

[2] IEEE standard. *VHDL reference manual, Std 1076-1993*. IEEE, 1993.

[3] Houzet, D., Barrandon, L. *Conception de Circuits en VHDL et VHDL-AMS – Principes et Méthodologie,* 2ème édition, ISBN 2.85428.527.1. Editions Cépaduès, Toulouse, 2006.

[4] Site Internet de la sté Digilent : www.digilentinc.com

[5] Site Internet de la sté Cirrus : www.cirrus.com.

[6] Site Internet de la sté Xilinx : www.xilinx.com

[7] Site Internet de la sté Altera : www.altera.com

[8] Boulanger, R. et al. *The Csound Book - Perspectives in Software Synthesis, Sound Design, Signal Processing and Programming*. MIT Press, Boston, MA., USA.

[9] http://www.cycling74.com/products/maxmsp

[10] http://line6.com/podxtlive