Louis Lumière
école nationale supérieure

Conception d'un Compresseur de Dynamique en Sigma-Delta

Antoine Ricoux

Mémoire de Master 2 - Spécialité Son

Directeur Interne : Mohammed Elliq
Directeur Externe : Paul Payen de La Garanderie
Responsable Universitaire : Corsin Vogel
Rapporteur : Laurent Millot

Juin 2021

Résumé :

Ce mémoire explore les moyens théoriques et techniques requis pour créer un compresseur de dynamique audionumérique utilisant la modulation Sigma-Delta (Σ/Δ).
Après avoir placé dans son contexte l'utilisation de la modulation Sigma-Delta en audionumérique, comme dans le SA-CD ou dans le DSD, on établira dans la première partie de ce mémoire, la relation entre dynamique et modulation Σ/Δ à 1 bit. Les enjeux technologiques qui rendent possible la mise en œuvre ce genre de traitement du signal, notamment le recours aux circuits logiques programmable de type FPGA serons également exposés.
Dans la deuxième partie, on présentera le synoptique d'un compresseur audio et l'on expliquera la réalisation de ses sous-blocs en utilisant la modulation Σ/Δ.

Finalement, on présentera le fonctionnement des outils logiciels et matériels nécessaires à la réalisation de notre compresseur de dynamique en modulation Σ/Δ. Cette explication sera suivie par la réalisation concrète d'un prototype de compresseur audio en modulation Σ/Δ utilisant un système sur puce (SoC).

Mots Clefs :

Sigma-Delta, Compresseur de Dynamique, FPGA, DSD, Traitement Numérique du Signal, Électronique

Abstract :

This master's thesis explores the theory and techniques required to create a digital audio dynamic compressor for Sigma-Delta modulation. After putting the use of Sigma-Delta modulation in its context, like in the SA-CD and DSD, we establish the relation between one bit audio and dynamics. We also take a look at the technological issues that makes this kind of digital signal processing possible, notably the choice of the FPGA.
In the second part, we will explain how every part of a dynamic compressor works and we will suggest a Sigma-Delta schematic for each part.

Finally, we will present the hardware and software tools used for implementing the Σ/Δ compressor in a system on chip. This presentation will be followed by a concrete implemeentation of said compressor.

Key Words :

Sigma-Delta, Compressor, FPGA, DSP, DSD, Electronics

Remerciements :

Je tiens à remercier chaleureusement les personnes dont l'aide précieuse m'a permis d'écrire ce mémoire :

Paul Payen de La Garanderie pour ses conseils avisés et sa disponibilité qui dépassent largement le cadre de directeur externe.

Mohammed Elliq pour sa confiance et son exigence.

Laurent Millot pour ses conseils utiles et son intérêt.

Corsin Vogel pour sa correction express et ses questionnements.

Charleyne, mes parents et ma sœur pour leur patience et leurs encouragements.

William Thenier et Valerian Fraisse pour leurs encouragements.

Florent Fajole pour son accueil au CDI.

Le corps enseignant de l'ENS Louis-Lumière pour un cadre de travail exceptionnel. Enfin, toute la promotion ENSLL Son 2021 pour une camaraderie sans précédent.

Table des matières

Acronymes

Σ/Δ Sigma-Delta.

Σ/Δ-M Sigma-Delta Modulator.

ADC Analog to Digital Converter.

AMBA Advanced Microcontroller Bus Architecture.

ARM Advanced RISC Machine.

AXI Advanced eXtensible Interface.

CB Connection Block.

CD-DA Compact Disc - Digital Audio.

DAC Digital to Analog Converter.

DAW Digital Audio Workstation.

DBCK DSD Bit Clock.

DSD Direct Stream Digital.

DSDL DSD Left.

DSDR DSD Right.

DSP Digital Signal Processor.

DST Direct Stream Transfert.

DXD Digital eXtreme Definition.

EEPROM Electrically-Erasable Programmable Read-Only Memory.

FIR Finite Impulse Response.

FPGA Field Programmable Gate Array.

GPIO General Purpose Input Output.

HDL Hardware Description Language.

HLS High Level Synthesis.

I^2C Inter Integrated Circuit.

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

IIR Infinite Impulse Response.

ILA Integrated Logic Analyzer.

IOB Input/Output Block.

IP Intellectual Property.

JTAG Joint Test Action Group.

LB Logic Block.

LED Light-Emitting Diode.

LSB Least Significant Bit.

LUT Look-Up Table.

MSB Most Significant Bit.

Mux Multiplexer.

NaN Not a Number.

PCI Peripheral Component Interconnect.

PCM Pulse Code Modulation.

PDM Pulse Density Modulation.

PL Programmable Logic.

PS Processing System.

PWM Pulse Width Modulation.

RISC Reduced Instruction Set Computer.

RMS Root Mean Square.

SA-CD Super Audio Compact Disc.

SB Switch Block.

SCL Serial Clock.

SDA Serial DAta.

SoC System on Chip.

SRAM Static Random Access Memory.

VHDL Very high speed integrated circuit HDL.

1 Introduction

L'une des innovations qui a permis la démocratisation globale de l'audionumérique est le convertisseur Sigma-Delta (Σ/Δ). Il propose une conversion très performante et peu coûteuse dès le début des années 1980. Vingt ans plus tard, Sony et Philips, avec la sortie du Super Audio Compact Disc, proposent de franchir le pas vers une chaîne du son entièrement Σ/Δ au détriment de la modulation PCM. Cependant, le domaine de l'audio professionnel, en grande partie, ne suit pas, notamment par manque de solutions professionnelles accessibles. En effet, le traitement du son, vital pour la plupart des étapes d'une production sonore, se révèle être un défi technologique particulièrement complexe en Σ/Δ.

Aujourd'hui, maintenant que la technologie du traitement numérique du signal a grandement évoluée. La question d'une chaîne du son cohérente entièrement en modulation Σ/Δ peut à nouveau être posée. Nous devons alors commencer par vérifier si les traitements numériques du son, devenus essentiels aujourd'hui, sont réalisables pour une modulation Σ/Δ. S'ils le sont, quels avantages et désavantages présentent ces traitements Σ/Δ par rapport aux traitements numériques utilisés aujourd'hui. Ainsi pourrons-nous déterminer la place qu'occupera la modulation Σ/Δ dans la chaîne audionumérique du futur.

Dans ce mémoire, nous allons étudier le traitement numérique de la dynamique d'un signal en modulation Sigma-Delta. L'une des raisons qui m'ont fait graviter autour de ce sujet est le paradoxe entre modulation 1 bit et évaluation du niveau. En effet, nous verrons dans quelles conditions nous pourrons évaluer le niveau d'un signal seulement composé de zéros et de uns.
Ce mémoire a pour volonté de s'inscrire dans la suite des mémoires de l'ENS Louis-Lumière traitants de la modulation Σ/Δ [Payen de La Garanderie 2015, et Montpied 2019]. On souhaite ici introduire un nouveau type de traitement sonore pour la modulation Σ/Δ et vérifier si ce modèle est viable pour une utilisation exigeante en matière de qualité sonore, comme la production musicale.

Dans la première partie, on définira les notions de modulation et modulateur Σ/Δ. Mais avant, on exposera l'histoire des innovations technologiques qui ont fait évoluer la modulation Σ/Δ et son utilisation.

Dans la partie suivante, on commencera par présenter la relation qui existe entre modulation Σ/Δ et la dynamique d'un système numérique. Puis, après avoir expliqué le fonctionnement d'un compresseur de dynamique, on proposera un modèle numérique de compresseur de dynamique pour la modulation Σ/Δ. Le modèle est éclaté en trois parties et chaque étape du traitement du signal est expliquée. Ainsi, les particularités du traitement de la modulation Σ/Δ seront abordées.
Enfin, dans la troisième partie, nous réaliserons le modèle de compresseur proposé dans la partie précédente dans un système sur puce. Les outils logiciels et matériels utilisés seront présentés et l'on montrera comment réaliser une interface utilisateur pour ce compresseur. L'interface comprendra des potentiomètres pour contrôler les paramètres du compresseur ainsi qu'un indicateur de niveau.

2 Modulation Sigma-Delta

2.1 Historique & Utilisations

Depuis 1982, le signal audionumérique est principalement représenté sous la forme de modulation par impulsion et codage, appelée PCM (*Pulse Code Modulation*). Dans le domaine de l'audio grand public, la résolution et fréquence d'échantillonnage retenue est celle du CD-DA (*Compact Disc - Digital Audio*), à savoir 16 bits et 44,1 kHz, respectivement. Cependant, dans le domaine de l'audio professionnel on rencontre couramment une quantification de 24 bits pour une fréquence d'échantillonnage entre 44,1 et 192 kHz. Le traitement en temps réel de ce signal PCM se fait à l'aide de processeurs de traitement numérique du signal ou DSP (*Digital Signal Processor*). Dans le domaine de l'informatique, c'est le microprocesseur de l'ordinateur, associé au coprocesseur mathématique, qui réalise le traitement du son en PCM.

Au début de l'audionumérique, toute la chaîne audio était en modulation PCM de façon cohérente : la conversion analogique-numérique, le traitement audio et la conversion numérique-analogique.

La recherche de convertisseurs analogique-numérique et numérique-analogique performants a conduit à l'utilisation de la modulation Σ/Δ. Cette situation a engendré le recours à deux conversions supplémentaires Σ/Δ vers PCM et PCM vers Σ/Δ.

Dans ce travail, on cherche à rendre la chaîne audio cohérente en effectuant les deux opérations de conversion et de traitement en utilisant uniquement la modulation Σ/Δ.

Dans cette partie, on va présenter le contexte historique pour comprendre comment la modulation Σ/Δ se place dans cette chaîne audio et ce qu'elle propose comme alternatives et évolutions.

2.1.1 Conception de la Modulation Sigma-Delta

La modulation Σ/Δ n'est pas une idée récente, elle fut théorisée dès 1952 et elle fut présentée comme une alternative plus performante et robuste que le PCM. Une illustration effectuée sous Matlab des apports de la modulation Σ/Δ par rapport au PCM est donnée ci-dessous :

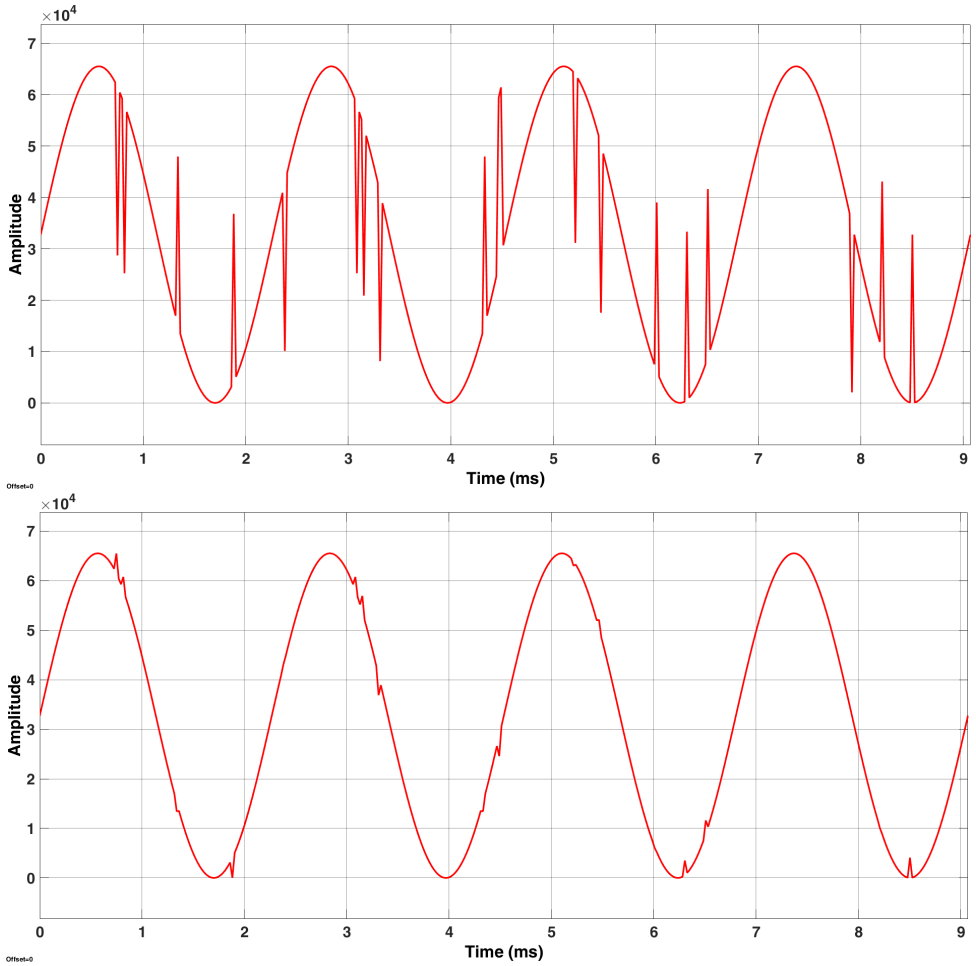


Figure 1 – Haut : Effet de la Modification du MSB (par Application de la Fonction Complément) de 5 % des Échantillons d'un Signal PCM (16 bits/44,1 kHz). Bas : Effet de la Modification des 12 bits de Poids les Plus Faibles de 5 % des Échantillons d'un Signal PCM (16 bits/44,1 kHz).

La figure 1 nous montre bien la vulnérabilité d'un signal PCM face à la corruption des bits de chacun de ses échantillons. La corruption du MSB des échantillons d'un signal PCM, l'affecte de manière considérable. En effet, le poids du MSB d'un échantillon représente la moitié de sa valeur totale.

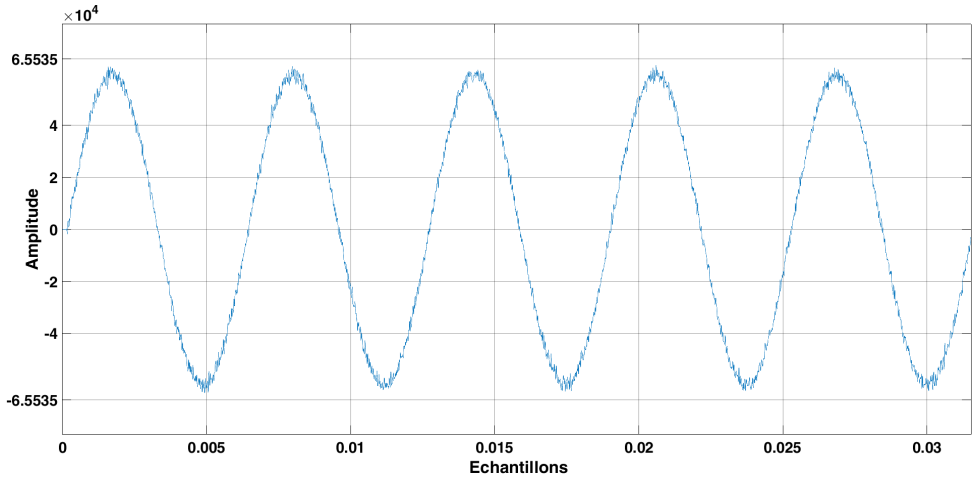


Figure 2 – Effet de la Modification du Bit (par Application de la Fonction Complément) de 5 % des Échantillons d'un Signal Σ/Δ

La figure 2 montre qu'en modulation Σ/Δ, la corruption d'un bit a moins d'effet sur l'intégrité du signal qu'en modulation PCM. En effet, en modulation Σ/Δ, il y a un bit par échantillon et comme tous les bits ont des poids équivalents, la corruption d'un bit a moins d'effet sur l'intégrité de la forme d'onde finale.

L'une des premières applications concrètes de la modulation Σ/Δ date de 1962 [Inose, Yasuda et Murakami 1962]. Avec celle-ci, la notion de mise en forme du bruit ainsi que le nom de modulation Σ/Δ furent introduits. Cependant, avec 40 dB de dynamique et 5 kHz de bande passante le système aura besoin de plus de développement et ne paraît pas attractif à son époque. La raison étant que l'expansion des systèmes numériques n'est pas encore arrivée, donc la recherche ne s'intéresse pas encore à de meilleurs moyens de numériser un signal analogique.

Dans les années qui suivent, le convertisseur Σ/Δ a fait l'objet de nouveaux travaux de recherche visant à améliorer ses performances. Ainsi, l'arrivée en 1982 du CD-DA, a nécessité le développement de convertisseurs performants, moins encombrants et à faible coût [Reefman et Janssen 2004]. Dès les années 1990, l'utilisation des convertisseurs Σ/Δ s'est donc démocratisée. Les apports des convertisseurs Σ/Δ par rapport aux convertisseurs PCM font que, la quasi-totalité des convertisseurs utilisés dans beaucoup de domaines, y compris en audiovisuel, sont basés sur les convertisseurs Σ/Δ. Les convertisseurs PCM cèdent le terrain face aux convertisseurs Σ/Δ de manière régulière.

2.1.2 Super Audio Compact Disc

Figure 3 – Logos du Support SA-CD et du Format DSD

En 1998, un support entièrement 1 bit fût présenté : le Super Audio Compact Disc (*SA-CD*). La fréquence d'échantillonnage du SA-CD, appelée DSD64, est 64 fois plus élevée que celle du CD-DA; elle vaut 2,8224 MHz. Le SA-CD a été présenté par Philips et Sony comme le successeur du CD-DA et propose une nouvelle chaîne d'écoute où le Σ/Δ n'est plus seulement une technologie interne des convertisseurs mais l'encodage de toute la chaîne du son. Le procédé de stockage de ce nouveau support est baptisé Direct Stream Digital (DSD).
En mars 1999, la première version du livret de spécification du SA-CD est éditée par Sony et Philips. Ce livret est nommé le *"Scarlet Book"*, en sachant que le livret de spécification du CD-DA est nommé *"Red Book"*, c'est encore une indication que le SA-CD était pensé comme une évolution du CD-DA.

Il existe trois types de SA-CD : simple couche, double couche et hybride [Janssen et Reefman 2003]. Le type le plus courant est l'hybride; il associe une couche CD-DA conventionnelle, avec une couche DSD haute définition placée au-dessus de celle-ci. Cela rend le SA-CD hybride rétrocompatible avec les lecteurs CD-DA conventionnels. Les deux autres types de SA-CD ne comporte que des pistes DSD, sur une ou deux couches. La couche DSD d'un SA-CD contient une piste multicanal 5.1 ainsi qu'une piste stéréo séparée.

En sachant qu'une couche DSD d'un SA-CD peut stocker 4,7 Go de données, voici le calcul de la durée en minutes d'un programme audio Σ/Δ à 1 bit en stéréo et en 5.1 avec une fréquence d'échantillonnage de 64 fois 44,1 kHz :

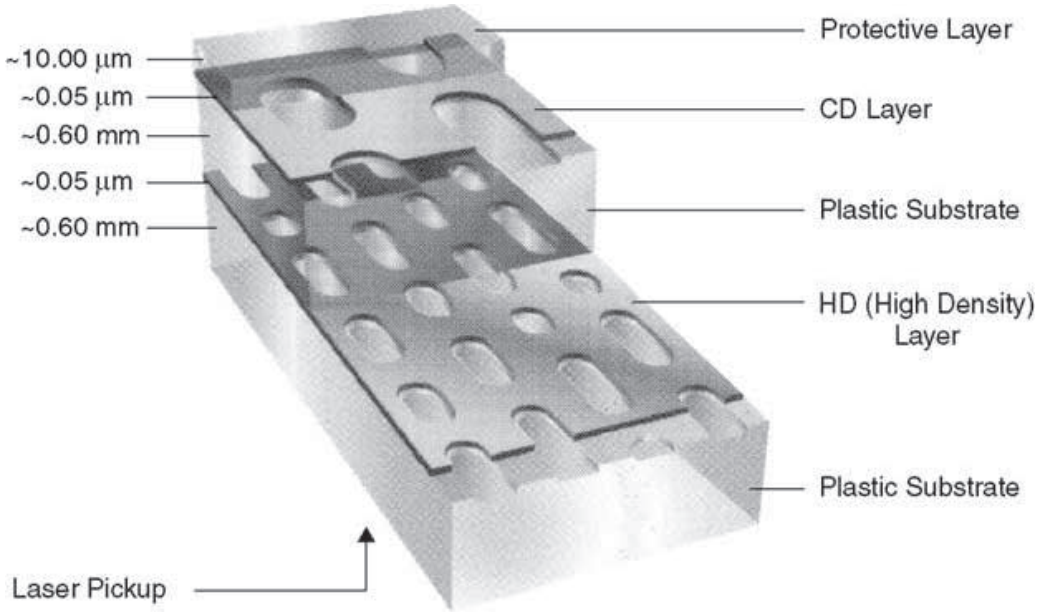


Figure 4 – Agencement des Couches d'un SA-CD Hybride [Janssen et Reefman 2003, Figure 1, p.1]

$$t_{pgm} = \frac{8 \cdot 4,7 \cdot 10^9}{1 \cdot 64 \cdot 44\,100 \cdot (6+2) \cdot 60} \approx 27\,min\,45\,sec$$

Cependant, chaque couche DSD d'un SA-CD propose 74 minutes de programme. Pour pouvoir stocker un programme de cette longueur, le débit doit être réduit sans perte (couramment appelé compression de donnée), avec un ratio d'environ 2,7. Ce procédé est appelé DST pour *Direct Stream Transfert*. L'encodage DST repose sur un filtre à prédiction linéaire et du codage entropique que nous ne présenterons pas ici [Janssen, Knapen et al. 2004]. Les procédés de compression sans pertes qui existent pour les formats PCM ne peuvent pas être utilisés efficacement sur un signal 1 bit. Souvent, la modulation Σ/Δ est appelée DSD par abus de langage, même si elle ne comporte de pas de compression DST.

2.1.3 Solutions de Traitement DSD en Temps-Réel

Le traitement en temps réel du flux DSD reste un problème difficile à régler. En effet, même si c'est théoriquement possible, la plupart des DSP ne sont pas assez performants pour cette application. Nous allons approfondir deux innovations qui contournent le problème de manière similaire : le procédé DSD-Wide de Sony [P. Thorpe et al. 2001] et le procédé Digital eXtreme Definition (*DXD*) de Merging Technologies [Vest 2004].

Ces deux solutions ont été développées avec le souci du maintien de l'ergonomie de travail des professionnels du son en tête. Elles font le constat que la production d'un SA-CD nécessite un environnement de travail qui permet d'éditer et de traiter des flux DSD de façon simple et sans faire de compromis sur la qualité originale du flux DSD.

2.1.3.1 DSD-Wide

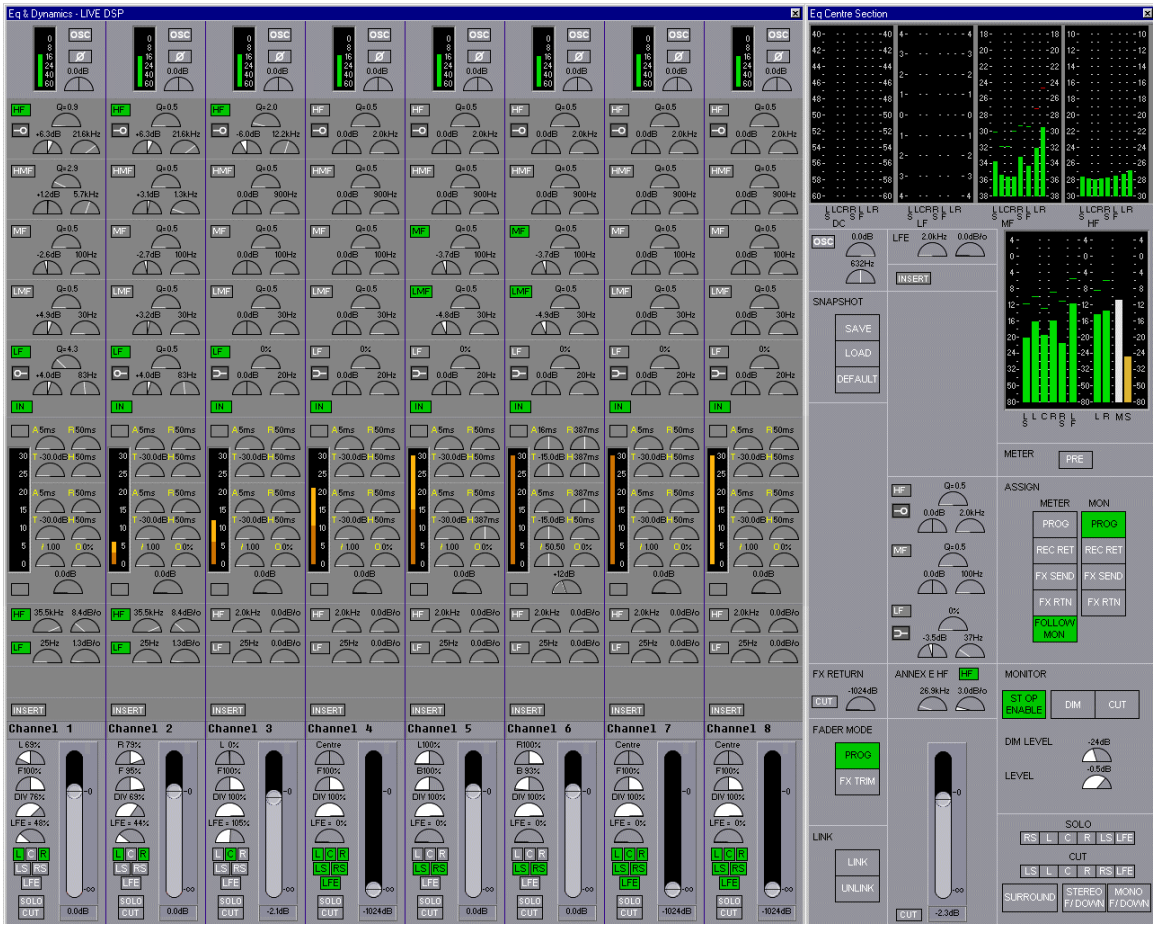


Figure 5 – L'Interface Utilisateur de la Console de Mastering Proposée par la Carte DSD-Wide [P. Thorpe et al. 2001, Figure 15, p.12]

Sony propose en 2001 le DSD-Wide qui est une matérielle (*hardware*). C'est une carte d'extension de type PCI (*Peripheral Component Interconnect*) équipée de sept puces de réseaux logiques programmables de type FPGA (*Field Programmable Gate Array*). Le format audionumérique utilisé à l'intérieur de la carte est de résolution 8 bits, avec la même fréquence d'échantillonnage que le DSD, soit $64 \times 44,1kHz = 2,8224MHz$. La caractéristique importante du format DSD-Wide est qu'il est parfaitement com-

patible avec le DSD. C'est-à-dire que la conversion, dans les deux sens, n'entraîne pas de perte de qualité.
Au final, comme dans la figure 5, la carte DSD-Wide peut proposer une console de mastering en 5.1 et en stéréo avec un jeu complet de traitements numériques : Niveau, Panoramique, Égalisation et Dynamique.

2.1.3.2 Digital Extreme Definition

Figure 6 – Le Logo du Format DXD [Merging Technologies p. d.(b)]

La solution de Merging Technologies intègre des éléments logiciels (*software*) et matériels (*hardware*). L'idée principale est de permettre aux utilisateurs de leur DAW Pyramix de produire un SA-CD en qualité DSD de l'enregistrement au mastering.
Pour ce faire, Merging Technologies a proposé un format propriétaire appelé DXD pour *"Digital eXtreme Definition"*. Le DXD est un format PCM 24 bits avec une fréquence d'échantillonnage de 352,8 kHz (8 fois 44,1 kHz). Comme le format interne du DSD-Wide, le but est de pouvoir convertir le DXD en DSD sans perte de qualité. Ainsi, le DXD est édité et traité de la même manière qu'un signal PCM ordinaire. L'ambiguïté de la dynamique du signal est levée, ce qui rend le contrôle du niveau au VU-mètre aisé.

Un des avantages de cette conversion PCM est qu'il suffit qu'un plug-in de traitement puisse accepter le format suréchantillonné du DXD pour pouvoir le traiter. Ainsi, Merging Technologies fournit une liste des plug-ins compatibles DXD [Merging Technologies p. d.(a)].

2.1.4 Évolution du FPGA

Les réseaux logiques programmables de type FPGA (*Field Programmable Gate Array*) ont étés introduits par l'entreprise américaine Xilinx en 1985. En résumé, un FPGA est une puce qui peut être programmée pour créer

n'importe quel circuit logique en son sein.
Au moment de la création du FPGA, les fonctions logiques étaient réalisées au moyen de circuits logiques discrets. La réalisation d'une nouvelle fonction nécessite la conception d'un nouveau circuit. Le FPGA a été développé comme solution à ce problème. Ainsi, si une modification de la fonction du circuit est à faire il suffit de reprogrammer le FPGA [Woods et al. 2017]. Dans le travail d'ingénieur électronique, cela permet de dissocier le travail de fabrication du circuit imprimé de celui de conception du circuit logique.

À ses débuts, le FPGA est donc vu comme un composant "glue" qui relie tous les composants d'un système pour le rendre plus complexe et configurable. Le succès du FPGA lui a permis de beaucoup évoluer depuis sa création. Il a commencé à être vu comme une solution complète pour une variété de problèmes. En effet, avec les avancées technologiques des semiconducteurs, le nombre de transistors par unité de surface (densité) dans une puce à grandement augmenté (loi de Moore). Á l'aube des années 2000, on à commencé à associer les FPGA aux microprocesseurs dans une même puce. De nos jours, ces plateformes hétérogènes sont appelées systèmes sur puces ou SoC (System on Chip).

Les FPGA ont un attrait particulier dans le domaine du traitement numérique du signal pour plusieurs raisons. En premier lieu, les FPGA modernes possèdent une grande puissance de calcul. De plus, contrairement aux processeurs, un FPGA peut effectuer un grand nombre de traitements en même temps [Eastty, Sleight et P. D. Thorpe 1997]. En effet, le FPGA est un circuit logique et non pas un processeur qui doit suivre un algorithme de manière linéaire. Un FPGA peut donc avoir différentes horloges non synchrones et faire des opérations complexes exactement en même temps.

Le recours à un FPGA dans un équipement audionumérique est justifié par le fait qu'il constitue une alternative intéressante vis-à-vis des DSP. Ainsi, le FPGA permettrait de réaliser une chaîne audio cohérente travaillant en modulation S/D allant de la captation à la diffusion.

2.2 Modulation Sigma-Delta

Tout d'abord, une mise au point du vocabulaire s'impose : le flux de sortie d'un modulateur Σ/Δ, désigné Σ/Δ-M (Sigma-Delta Modulator), est appelé par abus de langage modulation Σ/Δ. Cependant, il est aussi possible de rencontrer l'appellation modulation de densité d'impulsion (PDM pour Pulse Density Modulation). Cette PDM est équivalente, à fréquence d'échantillonnage constante, à la modulation de largeur d'impulsion (PWM pour Pulse Width Modulation).
Ainsi, dans le jargon technique, Σ/Δ, PDM, DSD et même PWM sont souvent utilisés de façon interchangeable.

Contrairement à la modulation par impulsion et codage, la forme d'onde de la modulation Σ/Δ ne ressemble pas à celle d'un signal analogique de prime abord. En effet, on peut se figurer l'amplitude de la forme d'onde originale seulement avec le rapport cyclique (*duty cycle*) de la modulation Σ/Δ. Comme suggéré par le nom "modulation à densité d'impulsion", lorsque le signal Σ/Δ est souvent au maximum c'est que l'amplitude doit être proche du maximum, de même pour le minimum. Lorsqu'il y a l'air d'avoir autant de 0 que de 1, c'est que l'amplitude est proche de 0.

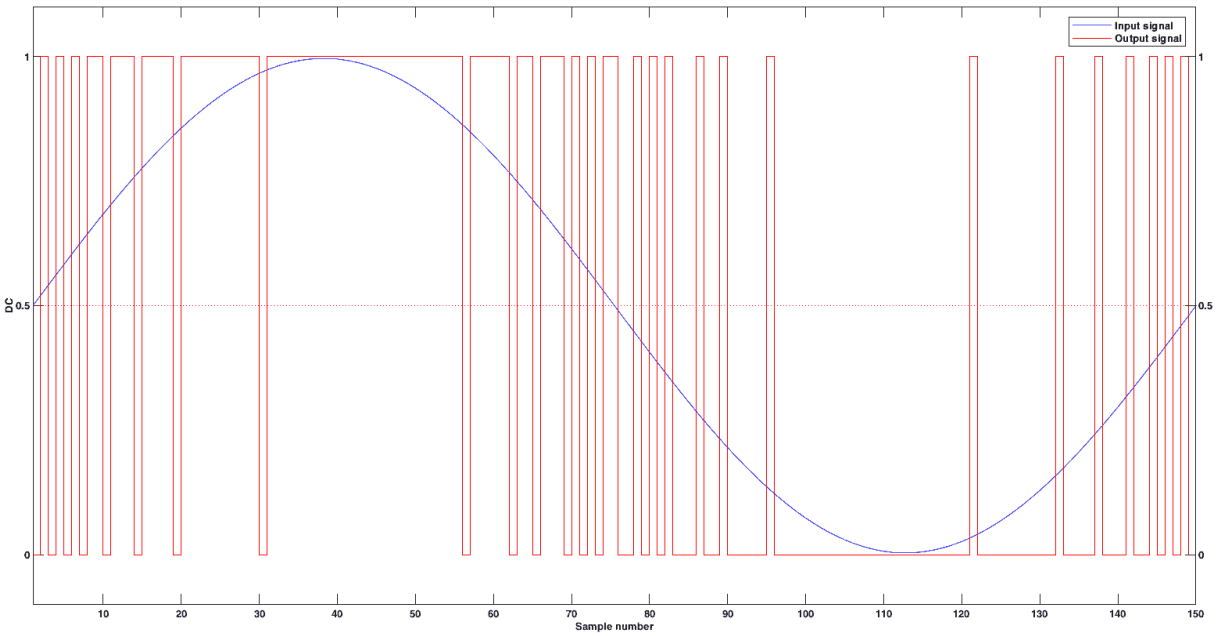


Figure 7 – Encodage d'un signal sinusoïdal en Σ/Δ 1 bit [Ogier 2021]

Appelé aussi convertisseur à suréchantillonnage, un convertisseur Σ/Δ utilise une fréquence d'échantillonnage f_s égale à celle d'un convertisseur PCM multipliée par un certain facteur N_0. Typiquement, ce facteur est 64, ce qui donne une fréquence d'échantillonnage de $N_0 \cdot f_s = 64 \times 44,1kHz = 2,8224MHz$. On rencontre aussi des facteurs de suréchantillonnage de 128 et 256.
La fréquence de Nyquist se retrouve donc dans la bande des ultrasons. Cependant, un repli spectral est toujours possible. Un filtrage des fréquences supérieures à la fréquence de Nyquist est donc toujours nécessaire, mais facilité par la fréquence d'échantillonnage qui est très haute par rapport à la bande audio à préserver.

La sortie du Σ/Δ-M est déterminée par l'erreur de conversion accumulée.c'est-à-dire que l'écart entre la valeur analogique et la valeur numérique.
Cet écart représente l'erreur de quantification calculée à chaque échantillon. Puis, elle est accumulée, on l'additionne aux erreurs de quantification précédentes. Si cette accumulation est positive, l'échantillon sera un 1. Il sera un 0 si l'accumulateur a une valeur négative. Ainsi, par nature, la valeur d'un échantillon dans un flux Σ/Δ dépend de la valeur des échantillons précédents. Cette caractéristique est autre une particularité d'un convertisseur Σ/Δ qui est absente dans un convertisseur PCM qui quantifie les échantillons individuellement.

2.3 Modulateur Sigma-Delta

Pour qu'un système numérique puisse traiter un signal audio, nous devons transformer ce signal en une suite d'échantillons quantifiés. En audio, ces échantillons sont prélevés à une fréquence constante F_s. Le rôle du modulateur est donc de convertir le signal d'entrée en une modulation numérique spécifique, en vue d'être interprétée par un système numérique.

Un convertisseur analogique-numérique ou ADC (*Analog to Digital Converter*) à suréchantillonnage moderne opère selon le principe décrit par le schéma de la figure 8.
On fait passer le signal d'entrée par un filtre analogique d'ordre faible afin de ne convertir aucune fréquence supérieure à la fréquence de Nyquist du modulateur Σ/Δ. On peut se permettre d'utiliser un filtre de faible ordre, car la fréquence d'échantillonnage du Σ/Δ-M est un multiple de la fréquence

d'échantillonnage cible, la fréquence de Nyquist est donc bien plus élevée. Ensuite, le flux Σ/Δ est passé à travers un filtre numérique avec un ordre élevé, son rôle est de filtrer les fréquences supérieures à la fréquence de Nyquist de la fréquence d'échantillonnage cible.

Enfin, la sortie du filtre est quantifiée sur le nombre de bits souhaité, à la fréquence d'échantillonnage cible. On retrouve alors une modulation PCM en sortie.
Employer un Σ/Δ-M pour déporter le filtrage anti-repliement d'ordre élevé dans le domaine numérique est une technique qui a permis d'une part d'améliorer la qualité des convertisseurs et d'autre part de réduire leurs coûts de fabrication.

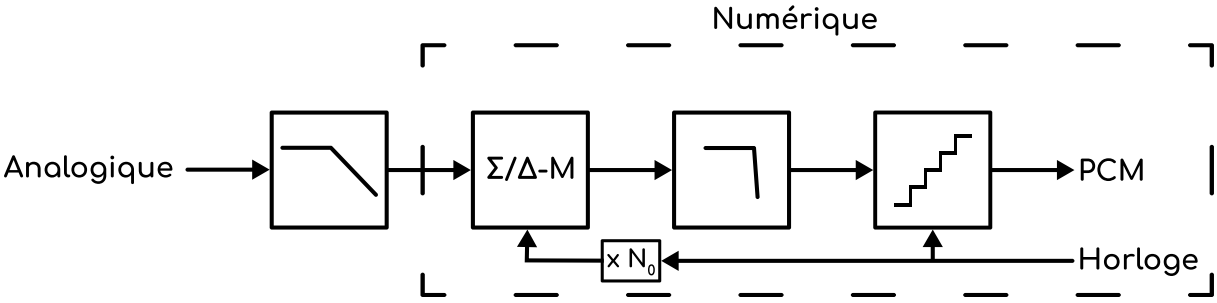


Figure 8 – Schéma Fonctionnel d'un ADC à Suréchantillonnage avec une Sortie PCM

Maintenant, nous allons nous intéresser seulement au modulateur Σ/Δ, car il est l'élément principal de l'ADC si on souhaite un flux Σ/Δ en sortie.

Comme son analogue PCM, le modulateur Σ/Δ (Σ/Δ-M) est basé autour d'un quantificateur. Le quantificateur est un système non linéaire qui est défini par ses caractéristiques statiques d'entrée/sortie [Schreier et Temes 2005]. L'entrée d'un quantificateur est généralement appelée Y et la sortie V.

En d'autres termes, c'est l'élément qui est chargé de transformer un signal en une suite d'échantillons d'une valeur quantifiée en un nombre de bits quelconque. L'audio en 1 bit sous-entend un quantificateur 1 bit avec des caractéristiques similaires à la Figure 10.

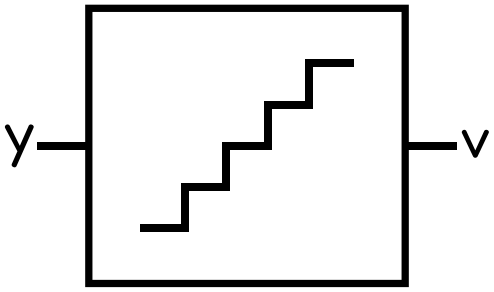


Figure 9 – Schéma du quantificateur

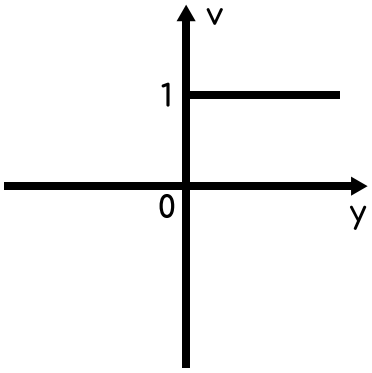


Figure 10 – Caractéristiques y/v d'un quantificateur bipolaire 1 bit

Un modulateur PCM est constitué uniquement d'un quantificateur. Dans le cas d'un Σ/Δ-M, nous devons ajouter les parties qui lui confèrent son nom. Pour toutes les explications suivantes, nous allons nous limiter à un Σ/Δ-M 1 bit du premier ordre.

Lorsqu'un modulateur Σ/Δ est utilisé dans un ADC, il est composé d'un quantificateur 1 bit, d'un filtre intégrateur et d'un convertisseur numérique-analogique 1 bit ou DAC (*Digital to Analog Converter*). Le DAC 1 bit consiste seulement à un multiplexeur qui commute entre la valeur maximale et minimale de l'entrée U.

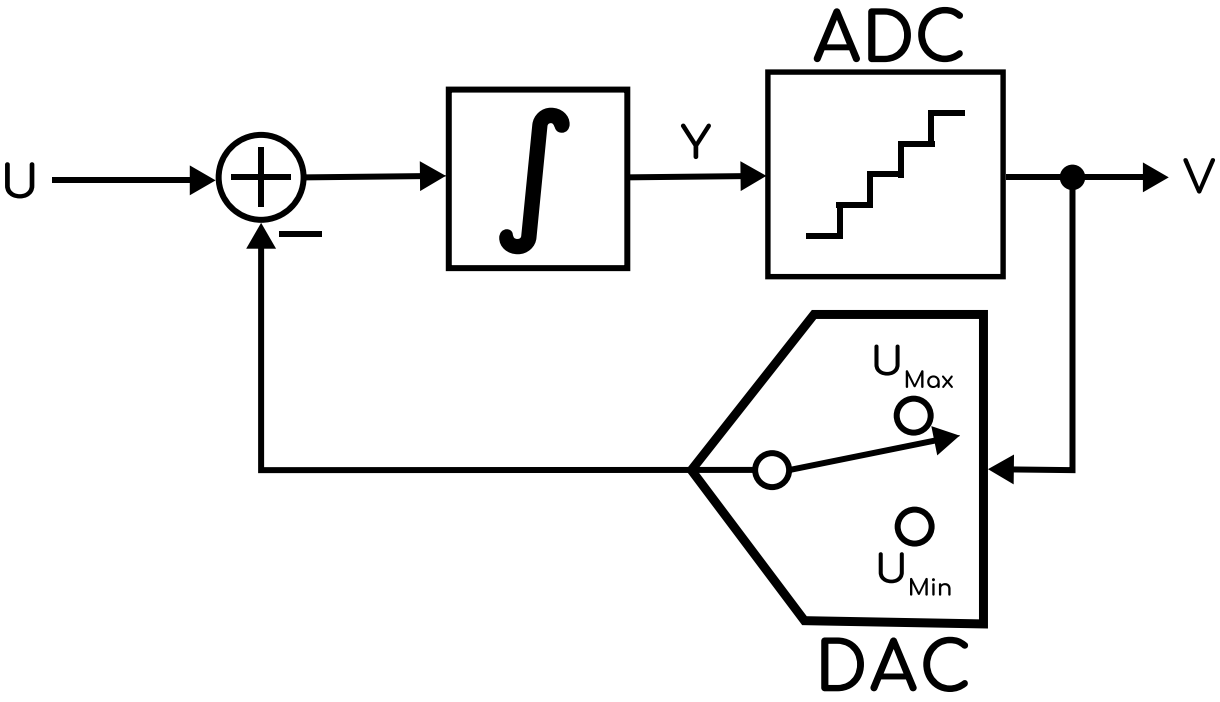


Figure 11 – Schéma d'un Σ/Δ-M utilisé comme ADC [d'après Schreier et Temes 2005, Figure 2.12 a, p.30]

On simplifie le schéma en supposant que $U_{Max} = VMax$ et en passant dans le domaine z. En domaine z, le filtre intégrateur peut être remplacé par un simple accumulateur. Le schéma du Σ/Δ-M devient :

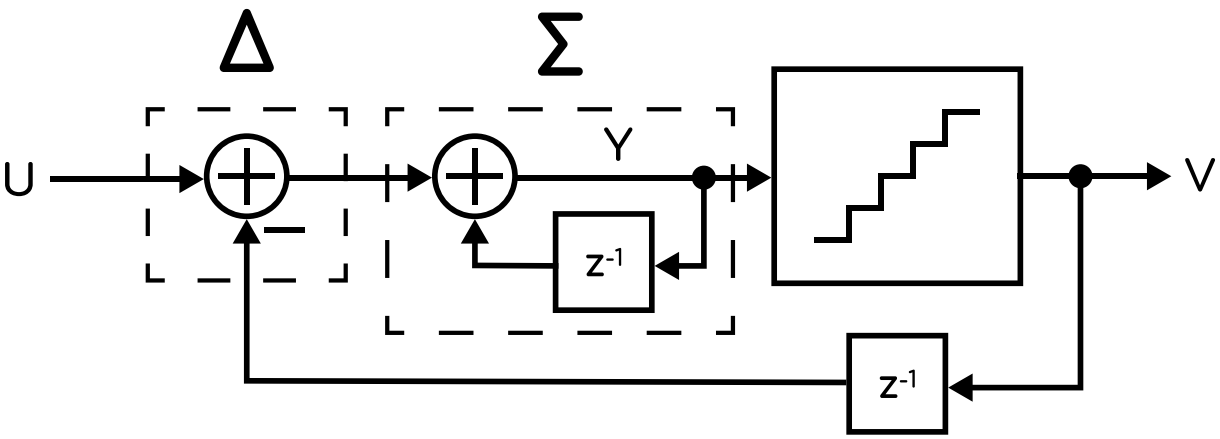


Figure 12 – Schéma d'un Σ/Δ-M 1 bit de premier ordre dans le domaine z [d'après Schreier et Temes 2005, Figure 2.16, p.33]

La figure 12 nous montre le système qui s'articule autour du quantificateur. La partie Δ est la soustraction de l'entrée par la sortie de l'échantillon précédent. Cette opération donne l'erreur de quantification.
Quand on parle de l'ordre du Σ/Δ-M, on parle de l'ordre du filtre de sa partie Σ. Dans notre cas, c'est un filtre de premier ordre, mais un convertisseur du marché utiliserait un filtre du cinquième ou sixième ordre. Le quantificateur a ici les caractéristiques de la Figure 10. Il a donc 0 en sortie quand l'erreur accumulée est négative et 1 quand elle est positive.

3 Traitement Dynamique en Sigma-Delta

3.1 Dynamique en Sigma-Delta

3.1.1 Mise en Forme du Bruit

D'après le théorème de quantification de Widrow [Widrow 1961], un quantificateur peut-être modélisé de manière linéaire par la somme d'un bruit blanc au signal d'entrée. Cette approximation est valide dans le cas où l'amplitude du signal d'entrée est grande et que le bruit n'est pas corrélé avec ce dernier. Dans ces conditions, on peut considérer la quantification comme un ajout aléatoire de bruit, appelé bruit de quantification. Le bruit de quantification est noté E(z) dans la Figure 13 et dans les calculs suivants. Nous allons pouvoir analyser le niveau du bruit de fond d'un Σ/Δ-M du premier ordre.

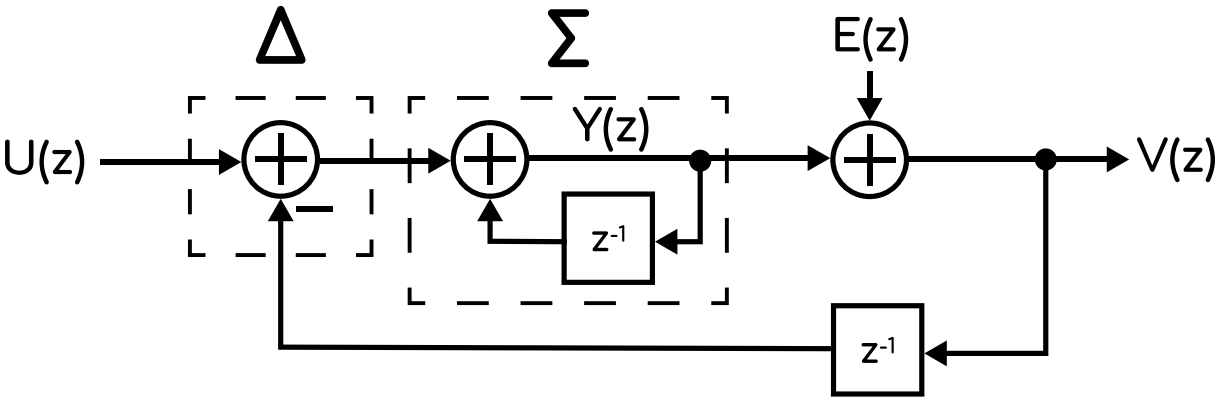


Figure 13 – Modèle linéaire d'un Σ/Δ-M de premier ordre dans le domaine z [d'après Schreier et Temes 2005, Figure 2.18, p.36]

D'après la Figure 13 on a :

$$Y(z) = U(z) - z^{-1}V(z) + z^{-1}Y(z)$$

Or :

$$\begin{aligned} V(z) &= Y(z) + E(z) \\ &= U(z) - z^{-1}V(z) + z^{-1}Y(z) + E(z) \\ &= U(z) + E(z) - z^{-1}\big(V(z) - Y(z)\big) \\ &= U(z) + E(z) - z^{-1}E(z) \\ &= U(z) + (1 - z^{-1})E(z) \end{aligned}$$

On remarque que la sortie d'un Σ/Δ-M linéarisé n'est d'autres que son

entrée et le bruit de quantification filtré. En effet, $(1 - z^{-1})E(z)$ représente le bruit filtré par un filtre coupe-bas du premier ordre. Pour mettre cela en évidence, voici le calcul de la fonction de transfert du filtre appliqué au bruit.

$$H_{bruit}(z) = 1 - z^{-1}$$

Pour avoir $H_{bruit}(f)$ il suffit de substituer z à $e^{j \cdot \frac{2 \cdot \pi \cdot f}{N_0 \cdot Fs}}$. Avec j le nombre complexe tel que $j^2 = -1$, N_0 le facteur de suréchantillonnage et F_s la fréquence d'échantillonnage de base.

$$H_{bruit}(f) = 1 - e^{-j \cdot \frac{2 \cdot \pi \cdot f}{N_0 \cdot Fs}}$$

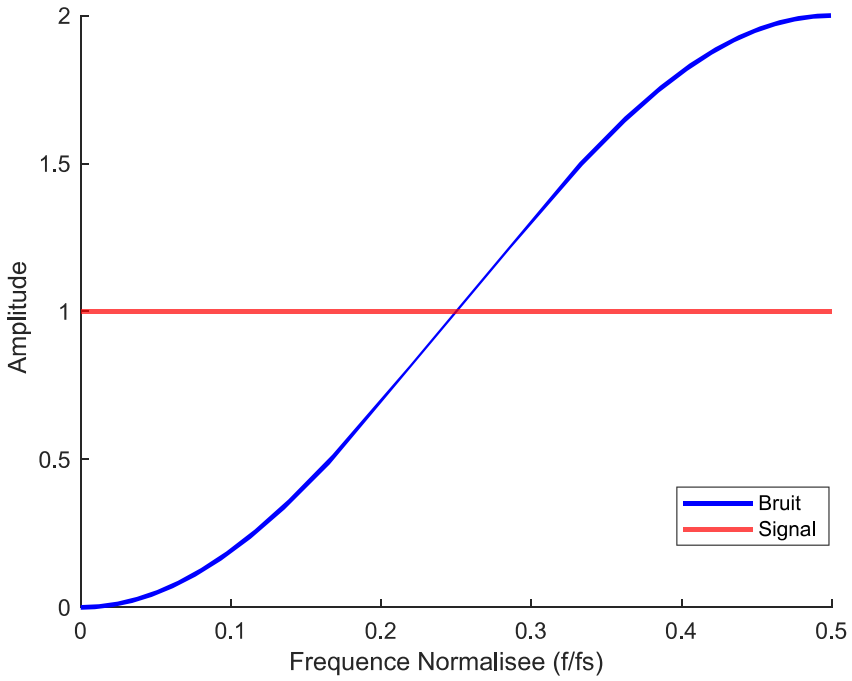


Figure 14 – Fonctions de transfert du signal et du bruit de quantification dans un Σ/Δ-M du premier ordre

Une des grandes caractéristiques du Σ/Δ-M est mise en évidence ici, le bruit de fond n'est pas de niveau uniforme sur tout le spectre. Cette caractéristique est appelée mise en forme du bruit ou *"Noise-Shaping"*, le bruit est donc concentré dans la partie haute du spectre. Par conséquent, la dynamique du Σ/Δ-M dépend de la fréquence. Ceci n'est pas le cas en

PCM.
De plus, cela met en avant l'importance de l'ordre du filtre présent dans le Σ/Δ-M, plus l'ordre est élevé plus le Noise-Shaping est intense. De cette manière, on réduit le niveau du bruit présent dans la plage fréquentielle utile au signal.

3.1.2 Calcul de Dynamique

On définira la dynamique d'un système numérique par le rapport signal sur bruit exprimé en décibels. Le calcul de la dynamique est donc :

$$Dyn = 10 \cdot \log_{10} \left(\frac{P_{EffMax}}{P_{Bruit}^{f_0}} \right)$$

Avec P_{EffMax} étant la puissance efficace d'un signal d'entrée au maximum de l'amplitude permise par le modulateur. P_{Bruit}, est la puissance du bruit de fond.
Avec un signal d'entrée sinusoïdal d'amplitude V_{Max}, l'expression de la dynamique devient :

$$\begin{aligned} Dyn &= 10 \cdot \log_{10} \left(\frac{\frac{V_{Max}^2}{2}}{P_{Bruit}^{f_0}} \right) \\ &= 20 \cdot \log_{10} \left(\frac{V_{Max}}{\sqrt{2}} \right) - 10 \cdot \log_{10}(P_{Bruit}^{f_0}) \end{aligned}$$

Dans le modèle linéaire de la figure 13, la seule source de bruit est le bruit de quantification, nous allons donc calculer sa puissance.

Prenons un quantificateur qui sature au-delà de l'intervalle $[-V_{Max}, V_{Max}]$. Au sein de cet intervalle, le modulateur a un pas de discrétisation de valeur Δ. Dans le cas d'un Σ/Δ-M 1 bit, le quantificateur a les caractéristiques de la figure 10. On a donc :

$$\Delta = V_{Max}$$

Le bruit d'un quantificateur vient de la différence d'amplitude entre le

signal continu d'entrée et le signal échantillonné de sortie. L'intervalle d'erreur possible est donc : $\left[-\frac{\Delta}{2}, \frac{\Delta}{2}\right]$. Si l'on suppose cette erreur aléatoire, on peut exprimer la densité de probabilité d'erreur suivante :

$$p_X(u) = \frac{1}{\Delta} \cdot \prod\left(\frac{u}{\Delta}\right)$$

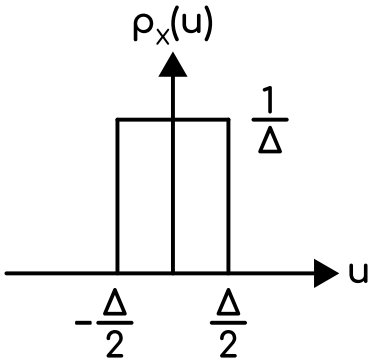


Figure 15 – Densité de Probabilité de l'Erreur de Quantification

Avec u l'erreur de quantification et $\prod\left(\frac{u}{\Delta}\right)$ la fonction rectangle centrée en 0 et de largeur Δ.

La puissance du bruit de fond σ_X^2 est calculée grâce à la variance de la densité de probabilité $p_X(u)$. Le calcul de la variance d'une variable aléatoire X est :

$$Var(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \sigma_X^2$$

Avec $\mathbb{E}[X^k]$ le moment d'ordre k de la variable aléatoire X dont le calcul est :

$$\mathbb{E}[X^k] = \int_{-\infty}^{\infty} u^k \cdot p_X(u)\, du$$

Dans le cas d'une variable aléatoire distribuée uniformément les deux premiers ordres de moments sont donnés par [Zölzer 2008] :

$$\begin{aligned}\mathbb{E}[X] &= \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} u \cdot p_X(u)\, du \\ &= \frac{1}{\Delta} \cdot \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} u\, du \\ &= 0\end{aligned}$$

$$\begin{aligned}\mathbb{E}[X^2] &= \int_{-\infty}^{\infty} u^2 \cdot p_X(u)\, du \\ &= \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \frac{u^2}{\Delta}\, du \\ &= \frac{\Delta^2}{12}\end{aligned}$$

Donc :

$$Var(X) = \mathbb{E}[X^2] = \sigma_X^2 = \frac{\Delta^2}{12}$$

La puissance du bruit de quantification σ_X^2 est répartie sur toute la période spectrale $[-N_0 \cdot F_s, N_0 \cdot F_s]$. Avec N_0 le facteur de suréchantillonnage, et F_s la fréquence d'échantillonnage de base. C'est l'un des avantages du suréchantillonnage, le niveau du bruit de quantification est plus bas car la puissance du bruit est répartie sur un intervalle plus grand.
Le calcul de la répartition spectrale du bruit de quantification $P_{BB}(f)$ est donc :

$$\begin{aligned}P_{BB}(f) &= \frac{\sigma_X^2}{N_0 \cdot F_s} \\ &= \frac{\frac{\Delta^2}{12}}{N_0 \cdot F_s} \\ &= \frac{\Delta^2}{12 \cdot N_0 \cdot F_s}\end{aligned}$$

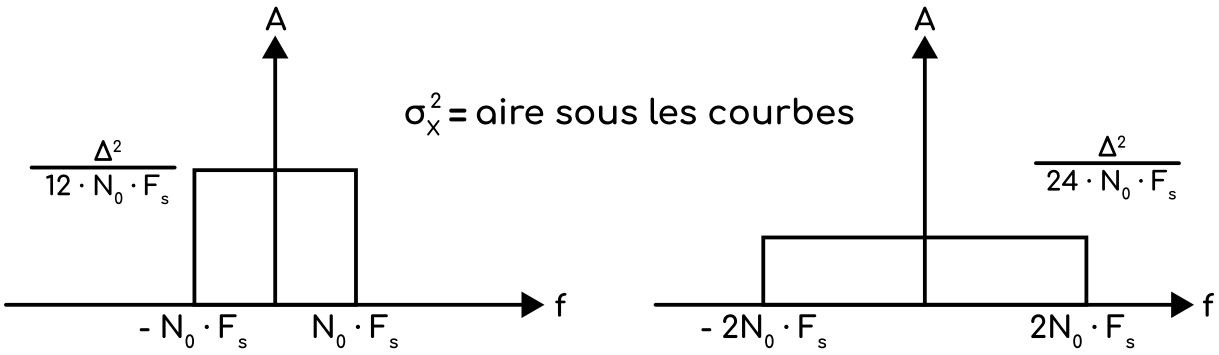


Figure 16 – Niveau du Bruit de Quantification en Fonction du Facteur de Sur-Échantillonnage [d'après Laroche 1995, p.10]

Il faut maintenant prendre en compte la caractéristique de mise en forme du bruit, propre au Σ/Δ-M. Pour cela il suffit de multiplier la répartition spectrale du bruit par le module au carré de la fonction de transfert du bruit $H_{bruit}(f)$. Cela nous donne la densité spectrale du bruit $D_{Bruit}(f)$.

$$\begin{aligned} P_{Bruit}(f) &= P_{BB}(f) \cdot |H_{bruit}(f)|^2 \\ &= \frac{\Delta^2}{12 \cdot N_0 \cdot F_s} \cdot |H_{bruit}(f)|^2 \end{aligned}$$

Intégrer la densité spectrale du bruit sur $[-f_0, f_0]$ permet de récupérer la puissance du bruit de fond dans la bande $[0, f_0]$. Avec $f_0 = 0,5 \cdot F_s$ on pourra calculer la puissance du bruit de fond dans la bande de fréquences utile. C'est cette puissance qui intervient dans le calcul de la dynamique.

$$\begin{aligned} P_{Bruit}^{f_0} &= \int_{-f_0}^{f_0} P_{Bruit}(f)\, df \\ &= \int_{-f_0}^{f_0} P_{BB}(f) \cdot |H_{bruit}(f)|^2\, df \\ &= \frac{\Delta^2}{12 \cdot N_0 \cdot F_s} \cdot \int_{-f_0}^{f_0} |H_{bruit}(f)|^2\, df \end{aligned}$$

Le détail des calculs peuvent être trouvé dans [Payen de La Garanderie 2015, p23-25]. On obtient :

$$|H_{bruit}(f)|^2 = \left(\frac{2 \cdot \pi \cdot f}{N_0 \cdot F_s}\right)^2$$

$$P_{Bruit}^{f_0} = \frac{\Delta^2}{12} \cdot \frac{\pi^2}{3} \cdot \left(\frac{2 \cdot f_0}{N_0 \cdot F_s} \right)^3$$

$$Dyn = 6,02 \cdot \big(N_b - 1 + 1,5 \cdot (r-1) \big) + 2,61 - 30 \cdot \log_{10} \left(\frac{f_0}{F_s} \right)$$

Avec r un facteur de suréchantillonnage tel que $N_0 = 2^r$ et N_b le nombre de bits du signal de sortie du modulateur tel que $\Delta = \frac{2 \cdot V_{Max}}{2^{N_b} - 1}$.

Faisons l'application numérique avec les paramètres suivants :

— Quantification de 1 bit, $N_b = 1\,bit$.

— Facteur de suréchantillonnage de 64, $r = log_2(64) = 6$.

— Fréquence d'échantillonnage de base de 44,1 kHz, $F_s = 44\,100\,Hz$.

— Une bande passante de $\left[0, \frac{F_s}{2}\right]$, $f_0 = 22\,050\,Hz$.

On trouve :

$$Dyn \approx 56,8\,dB$$

C'est un résultat bien inférieur à la dynamique du CD-DA qui est aux alentours de 96 dB. Il y a plusieurs solutions pour pallier à ce problème :

— Augmenter le facteur de suréchantillonnage. En effet, doubler le facteur de suréchantillonnage N_0 (ajouter 1 à r) augmente la dynamique d'environ 9 dB.

— On pourrait faire une quantification sur plus de bits. Chaque bit supplémentaire représente environ 6 dB de dynamique supplémentaire. C'est la solution qu'adopte certains ADC en incorporant des étages multi bits dans leur Σ/Δ-M d'ordre élevé. Par souci de simplicité, ce n'est pas une solution que nous explorerons dans ce mémoire.

— Augmenter l'ordre du modulateur. Avoir un filtre d'ordre supérieur permettrait d'accentuer la propriété de mise en forme du bruit du

Σ/Δ-M. Ainsi, la densité spectrale du bruit de quantification est encore plus concentrée dans les hautes fréquences et dégage le bruit de la bande passante.

Autant de raisons pour lesquelles la plupart des puces de convertisseurs comportent des modulateurs d'ordre 4, 5 ou supérieurs. De plus, il est commun de voir des facteurs de sur échantillonnage de 128 ou 256. Cependant, étudier le cas d'un Σ/Δ-M de premier ordre est plus simple.

3.2 Fonctionnement d'un Compresseur Dynamique

3.2.1 Fonction

Un compresseur est un outil de traitement du signal audio qui permet de réduire la plage dynamique d'un son. Le compresseur réalise cette tâche en réduisant le niveau sonore lors des moments où le signal d'entrée est le plus fort.
Ses réglages peuvent lui permettent d'intervenir sur la micro dynamique ou la macro-dynamique du son. C'est-à-dire des temps très courts comme un coup de caisse claire ou des temps assez longs comme les différents mouvements d'une symphonie.

Aujourd'hui le compresseur est un élément essentiel de beaucoup de productions sonores. En production musicale, beaucoup de sonorités modernes sont définies en grande partie par le rôle adopté par le compresseur. Ainsi, la diversité des modèles de compresseurs est à l'image de la grande diversité d'applications qui lui sont attribuées.
C'est pour cela qu'il n'est pas aisé de résumer la totalité des fonctionnalités qui ont été ajoutées au compresseur. Cependant, nous allons nous concentrer sur les fonctions principales et celles qui seront implémentées dans notre compresseur Σ/Δ.

3.2.2 Seuil & Ratio

Le seuil de compression (*threshold* en anglais) peut être considéré comme le paramètre principal d'un compresseur dynamique. Le seuil de compression indique le niveau à partir duquel la compression est enclenchée. Quand le niveau du signal appliqué à l'entrée du compresseur dépasse ce

seuil, le niveau sonore est baissé de manière proportionnelle à un ratio.

Ce ratio est souvent exprimé en fonction de combien de décibels en entrée, une fois passé le seuil, sont nécessaires pour augmenter le niveau de sortie de 1 dB. On peut donc associer le ratio à l'agressivité de la compression.
Par exemple, pour un ratio de 2 : 1, un seuil de -20 dB et un niveau continu à -18 dB en entrée, le compresseur réduira le niveau sonore du signal traité de 1 dB (Figure 17). Si le ratio est était de 4 : 1, la réduction serait de 1,5 dB. Si le ratio est infini, le compresseur devient un limiteur, la réduction du gain devient alors égale au niveau dépassant le seuil.

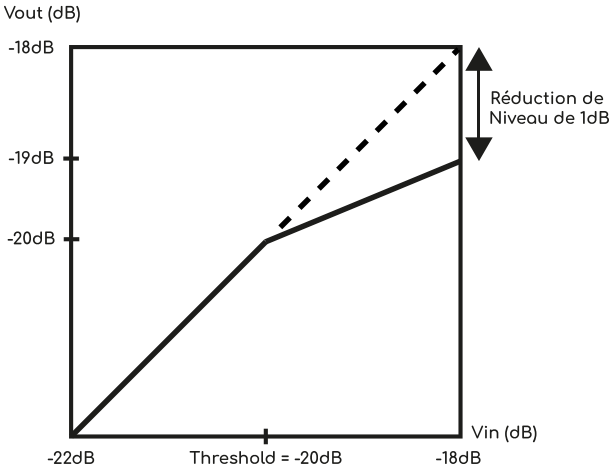


Figure 17 – Réponse dynamique d'un compresseur avec un ratio de 2 : 1 et un seuil de -20 dB

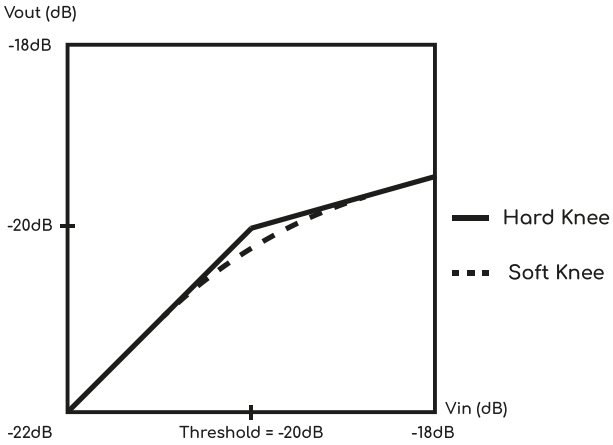


Figure 18 – Comparaison des réponses dynamiques d'un compresseur *Hard Knee* et *Soft-Knee*

On peut faire en sorte d'avoir un ratio qui évolue en fonction du niveau d'entrée du compresseur. Cette caractéristique s'appelle le *knee*. Ce nom fait référence à la jointure de la réponse dynamique au niveau du seuil. Dans la figure 18, on voit deux façons de traiter la transition de la compression au niveau du seuil.
Le "*hard knee*" compresse le signal, avec le ratio maximum, dès que le niveau détecté a dépassé le seuil. Le "*soft knee*" quant à lui fait une transition "douce" et augmente le ratio de compression un peu avant le seuil et n'est pas encore au ratio indiqué directement après le seuil. Certains compresseurs permettent de régler la largeur du *knee*.

3.2.3 Détection

La façon dont le niveau d'entrée du compresseur est évalué influe sur la totalité du comportement d'un compresseur. Le circuit de détection initie un signal de niveau détecté (appelé *sidechain* en anglais). Le niveau détecté peut être évalué de façons différentes.
On distingue ici les compresseurs dits "*peak*" et "RMS" (Root Mean Square ou valeur efficace).
Les compresseurs *peak* détectent le niveau absolu des crêtes du signal. Ils sont plus facilement réalisables en numérique et peuvent servir à effectuer des corrections très rapides.
Une détection RMS est facilement intégrée dans un compresseur analogique, ici on détecte la valeur efficace du signal. On agit donc plus facilement sur la macro-dynamique.

Certains compresseurs permettent de choisir où est prélevé le signal du détecteur. Si le signal de *sidechain* est prélevé avant le circuit de réduction de niveau, le compresseur est de type "*feed-forward*". Si le *sidechain* détecte le niveau après le circuit de réduction de niveau le compresseur de type "*feedback*". Les compresseurs *feed-forward* sont privilégiés pour un traitement précis et rapide du son, un modèle classique de compresseur *feed-forward* étant l'Urei 1176LN. Les compresseurs *feedback*, quant à eux, sont privilégiés pour un traitement doux et lent du signal, un modèle classique de compresseur *feedback* étant le Teletronix LA-2A.

De plus, des options avancées d'un compresseur peuvent inclure un filtrage du signal avant la détection de niveau. C'est une option qui permet de sculpter la réponse du compresseur en fonction du contenu fréquentiel du signal d'entrée. Par exemple, couper les basses du signal de *sidechain* voudra dire que le compresseur réagira moins fortement aux composantes graves du signal sonore.

3.2.4 Réponse Temporelle

La rapidité de réaction du compresseur peut être réglée. Ainsi, le temps d'attaque (*attack*) détermine le temps que mettra le compresseur à s'activer et le temps de retour (*release*) détermine le temps que met le compresseur à revenir à son état initial. Par exemple, si le niveau du détecteur dépasse le seuil pendant un temps plus court que le temps d'attaque, le

compresseur ne réagira pas.

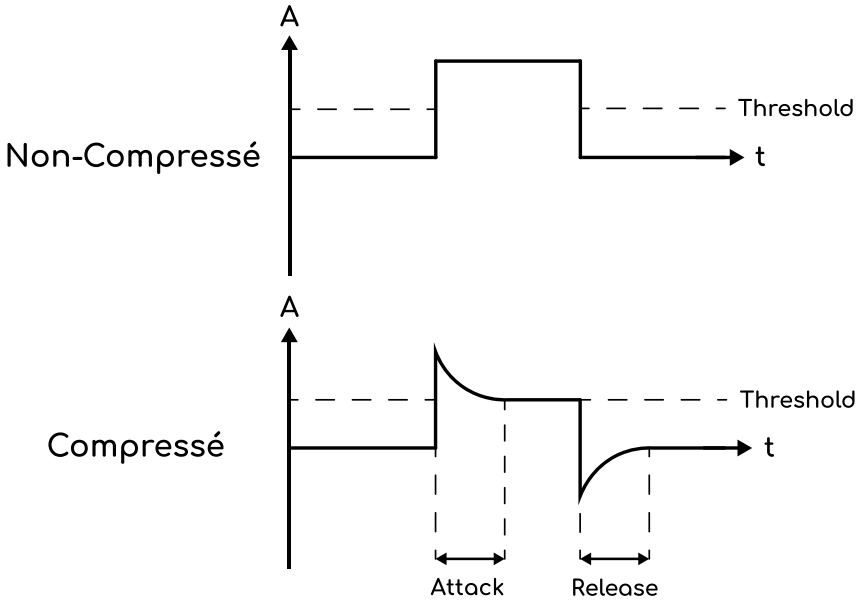


Figure 19 – Effet d'un Compresseur sur un Signal Rectangulaire

La figure 19 met en évidence le temps d'attaque et de retour pour un compresseur avec un ratio infini. On remarquera que dans ce cas, les temps d'attaque et de retour sont similaires.

Les compresseurs numériques ont introduit une façon de pouvoir détecter le signal en avance appelée le "*look-forward*". Il s'agit juste d'introduire du délai au signal par rapport au *sidechain*, mais cela permet de décaler à volonté le signal de réduction de gain par rapport à l'audio.

3.2.5 Signal de Sortie

Comme un compresseur peut réduire considérablement le niveau du signal traité, il est essentiel de pouvoir augmenter le niveau de sortie afin de compenser la réduction de niveau sonore perçu. En effet, le but principal d'un compresseur est seulement de réduire la plage dynamique du signal traité, pas forcément de réduire son niveau sonore général. Cette compensation de gain est souvent appelée *makeup gain*.

Certains compresseurs proposent en sortie un mixeur entre le signal d'entrée et le signal de sortie du compresseur. Mélanger ces deux signaux s'appelle de la compression parallèle.

3.2.6 Couplage

Quand un compresseur traite un signal stéréo, il est souvent utile de coupler les compressions de chaque canal entre elles (*stereo linking*). C'est-à-dire que, par exemple, pour les mêmes réglages de détection, si seulement le canal gauche dépasse le seuil et déclenche la compression, le canal droit est compressé de la même façon. C'est une façon de procéder pour avoir une image stéréo plus stable. Il est aussi possible d'utiliser une seule chaîne de détection pour la somme des deux canaux, mais cette technique supprime l'option de découpler les paramètres des deux canaux.

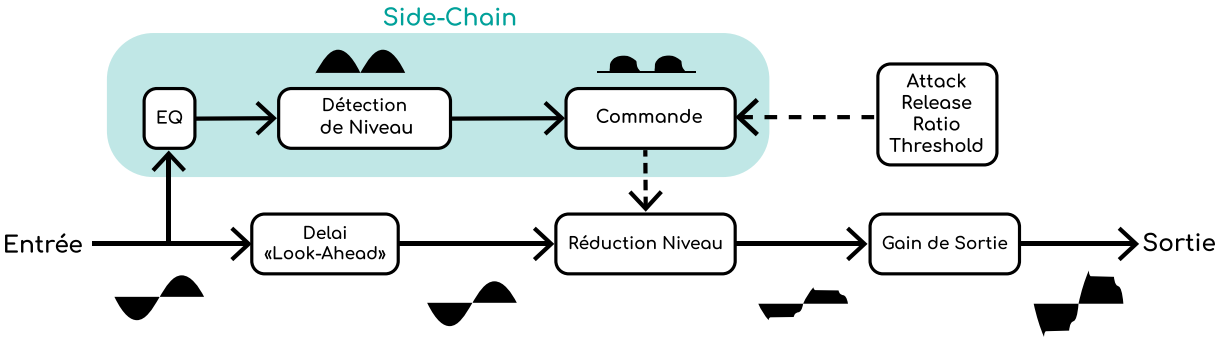


Figure 20 – Schéma Fonctionnel d'un compresseur *Feed-Forward* avec Equalisation de *Side-Chain* et *"Look-Ahead"*

3.3 Conception du Compresseur Dynamique Sigma-Delta

Réaliser un compresseur Σ/Δ pourrait présenter des caractéristiques intéressantes par rapport à un compresseur PCM. La fréquence d'échantillonnage est bien supérieure, ce qui permet d'avoir un temps de réduction de gain extrêmement rapide. Cela peut être une caractéristique recherchée pour un limiteur.
De plus, la latence introduite par le traitement numérique d'un signal Σ/Δ est souvent très faible. Ce qui peut aussi être une caractéristique recherchée dans les applications où la rapidité d'exécution est essentielle.

Le modèle de compresseur que nous souhaitons réaliser est un compresseur capable d'exploiter les capacités d'un compresseur numérique et du format DSD c'est-à-dire un modèle proche de celui de la Figure 20, notamment une détection de crête très efficace grâce à l'architecture *feed-forward*.

C'est pour cette raison que la fréquence d'échantillonnage n'est pas réduite dans le *sidechain*. Ainsi, chaque échantillon DSD est associé à un gain de compression.
Par souci de compréhension, le compresseur est divisé en trois blocs fonctionnels qui seront présentés successivement (Figure 21).

Tous les modèles sont réalisés dans *System Generator for DSP*, nous expliciterons l'utilité de ce programme dans la partie suivante. Nous verrons les modèles comme des schémas fonctionnels, certains blocs seront donc omis par souci de simplicité, car leur présence n'altère pas la compréhension du circuit. Les blocs de fonction utilisés sont définis dans l'annexe A.

Les graphes de cette partie ont été créés grâce à un modèle de test dans *System Generator for DSP* (Figure 22). Le signal appliqué à l'entrée du compresseur est un signal sinusoïdal de fréquence 160 Hz et d'amplitude maximale et quantifiée sur 16 bits. Ce sinus est modulé en Σ/Δ grâce à un Σ/Δ-M de premier degré. La sortie du compresseur est traitée de la même manière que dans le détecteur de niveau du compresseur (figure 23) afin de récupérer un signal PCM facile à interpréter.
Afin réduire au minimum les artefacts et le bruit de fond engendrés par un simple Σ/Δ-M de premier ordre, le facteur de suréchantillonnage est augmenté à 256 pour le test du modèle. On pourra tout de même remarquer quelques imprécisions dans la forme d'onde du signal. Ce problème n'apparaîtra normalement pas avec une puce ADC du marché.
Pour visualiser le graphe d'un signal à un moment de son traitement, il suffit de le connecter à un bloc de sortie suivi d'un oscilloscope *Simulink*.

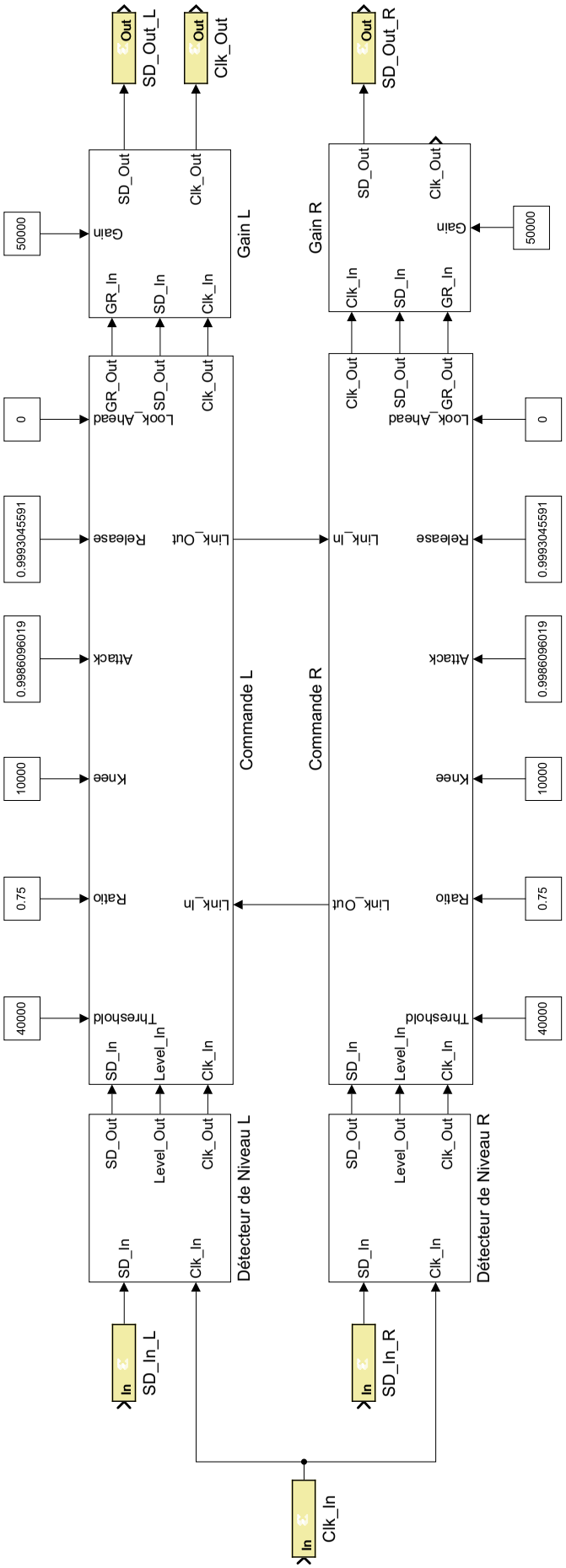


Figure 21 – Connexion des Trois Blocs pour Réaliser un Compresseur Stéréo dans *System Generator for DSP*

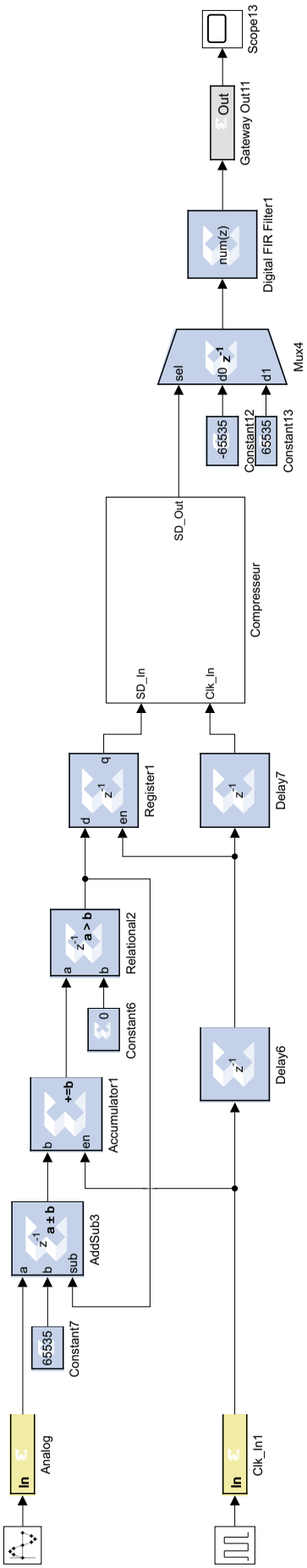


Figure 22 – Conditions de Test du Compresseur dans *System Generator for DSP*

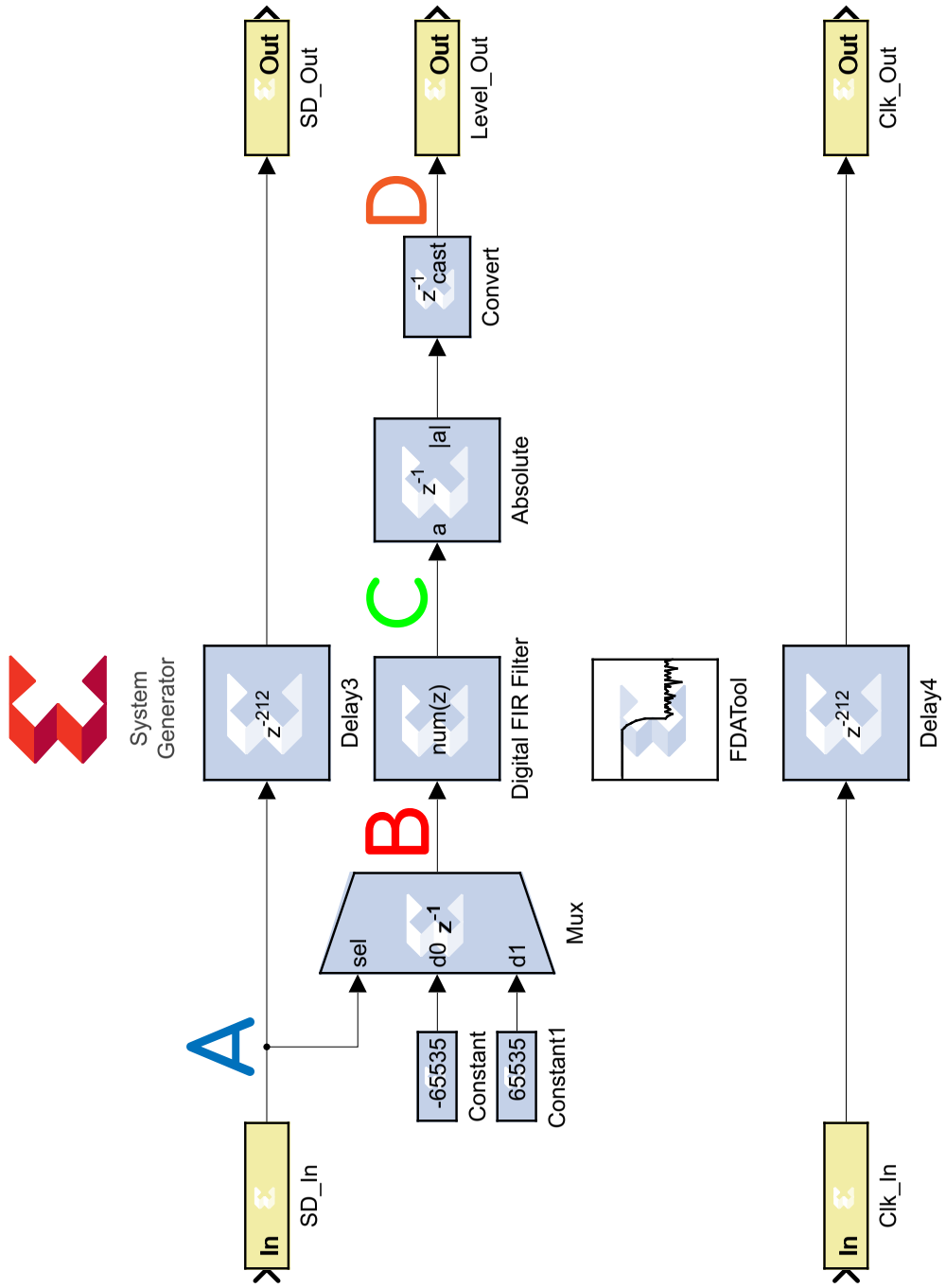


Figure 23 – Modèle *System Generator for DSP* du Bloc de Détecteur de Niveau

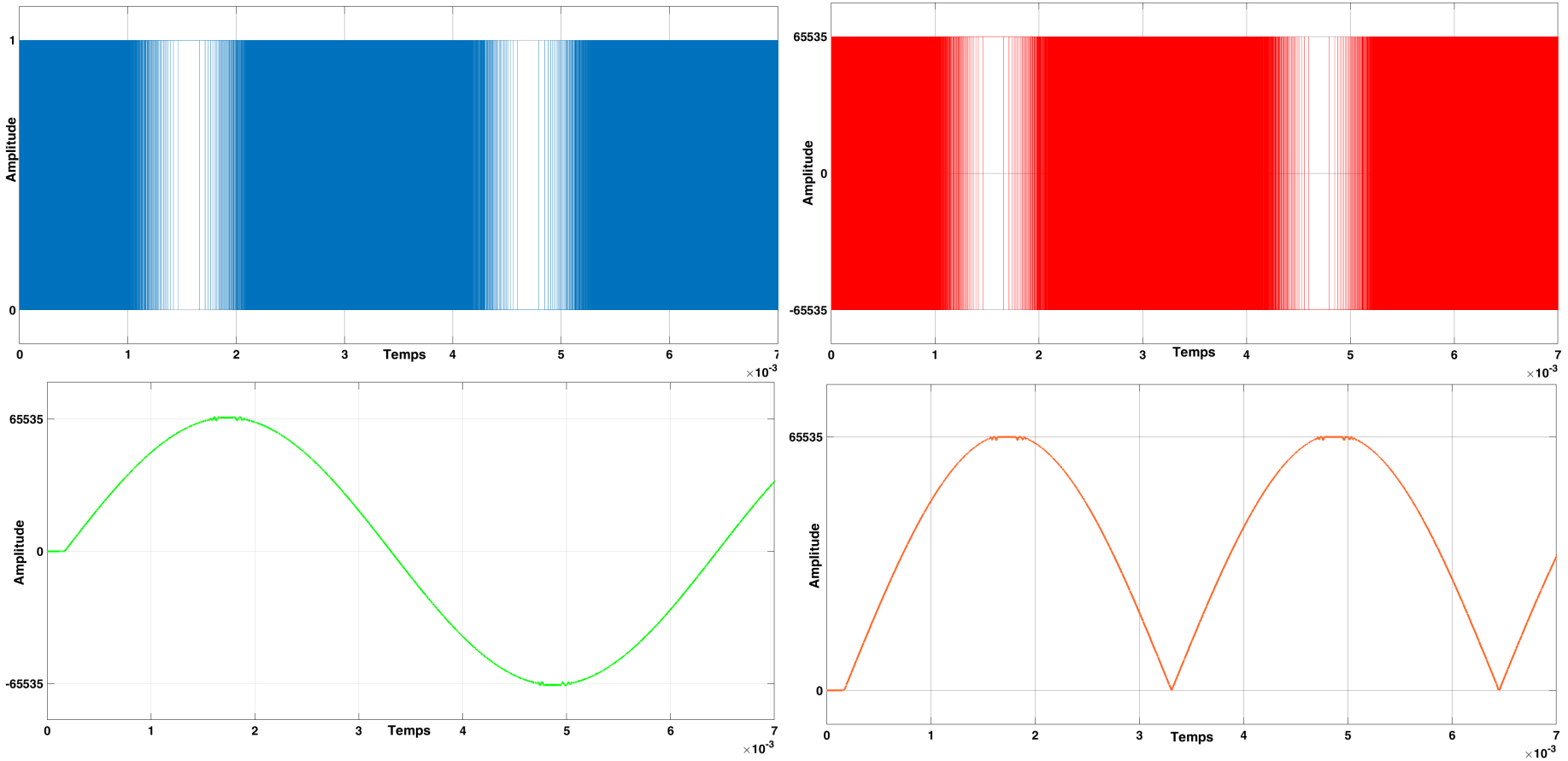


Figure 24 – Signal du Détecteur de Niveau aux Points A (en haut à gauche), B (en haut à droite), C (en bas à gauche) et D (en bas à droite)

3.3.1 Détecteur de Niveau

La première partie du compresseur permet d'évaluer le niveau du signal entrée. Ce procédé est proche d'un convertisseur Σ/Δ vers PCM. En effet, il est basé autour d'un filtre à réponse impulsionnelle finie ou FIR (*Finite Impulse Response*).

En premier lieu, les deux blocs d'entrées acheminent le signal Σ/Δ et le signal d'horloge, provenant de l'ADC, dans le système. Ensuite, l'amplitude du signal Σ/Δ à 1 bit est étendue à $\pm$65535. Ceci est fait grâce au bloc multiplexeur (Mux) et aux deux blocs de constantes.

Le signal est alors filtré par le bloc FIR configuré par le FDATool. Le filtre est configuré de manière à filtrer le bruit de quantification concentré dans les hautes fréquences et de récupérer un signal de type PCM.
Puis, on récupère la valeur absolue de ce signal PCM. Les crêtes négatives deviennent positives, on a redressé le signal pour récupérer le niveau de crête.

D'après ce calcul simplifié (où le bruit de fond est ignoré, 1 bit = 6 dB), un signal 16 bits non signé nous donne une dynamique de réduction de niveau de 96dB.

$$Dyn = 20log_{10}(2^{16}) \approx 96dB$$

C'est une dynamique bien suffisante pour l'utilisation dans le *sidechain* d'un compresseur. C'est pour cela qu'une amplitude de $\pm$65535 a été choisie lors de la première étape. La valeur maximale d'un nombre 16 bits non signé est de 65535. Ainsi, le niveau détecté, une fois redressé (Figure 24 C), à pour maximum 65535. Ce signal peut donc être converti sans pertes par le bloc convertisseur en un nombre non signé exprimé sur 16 bits.

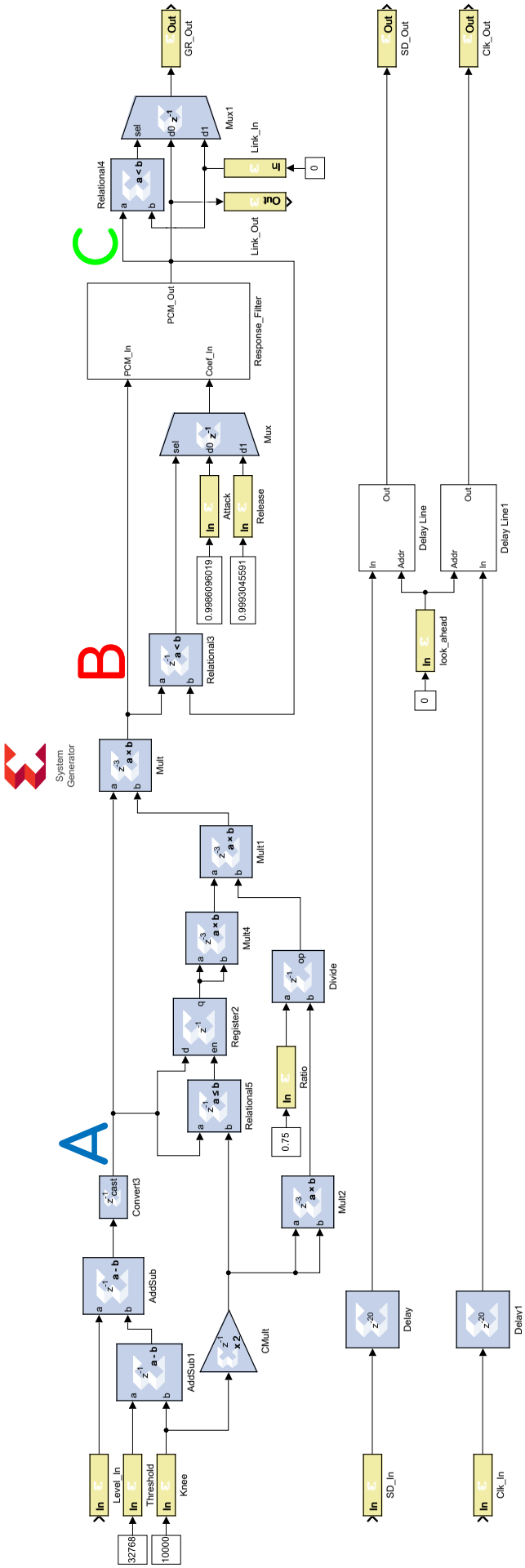


Figure 25 – Modèle *System Generator for DSP* du Bloc de Commande

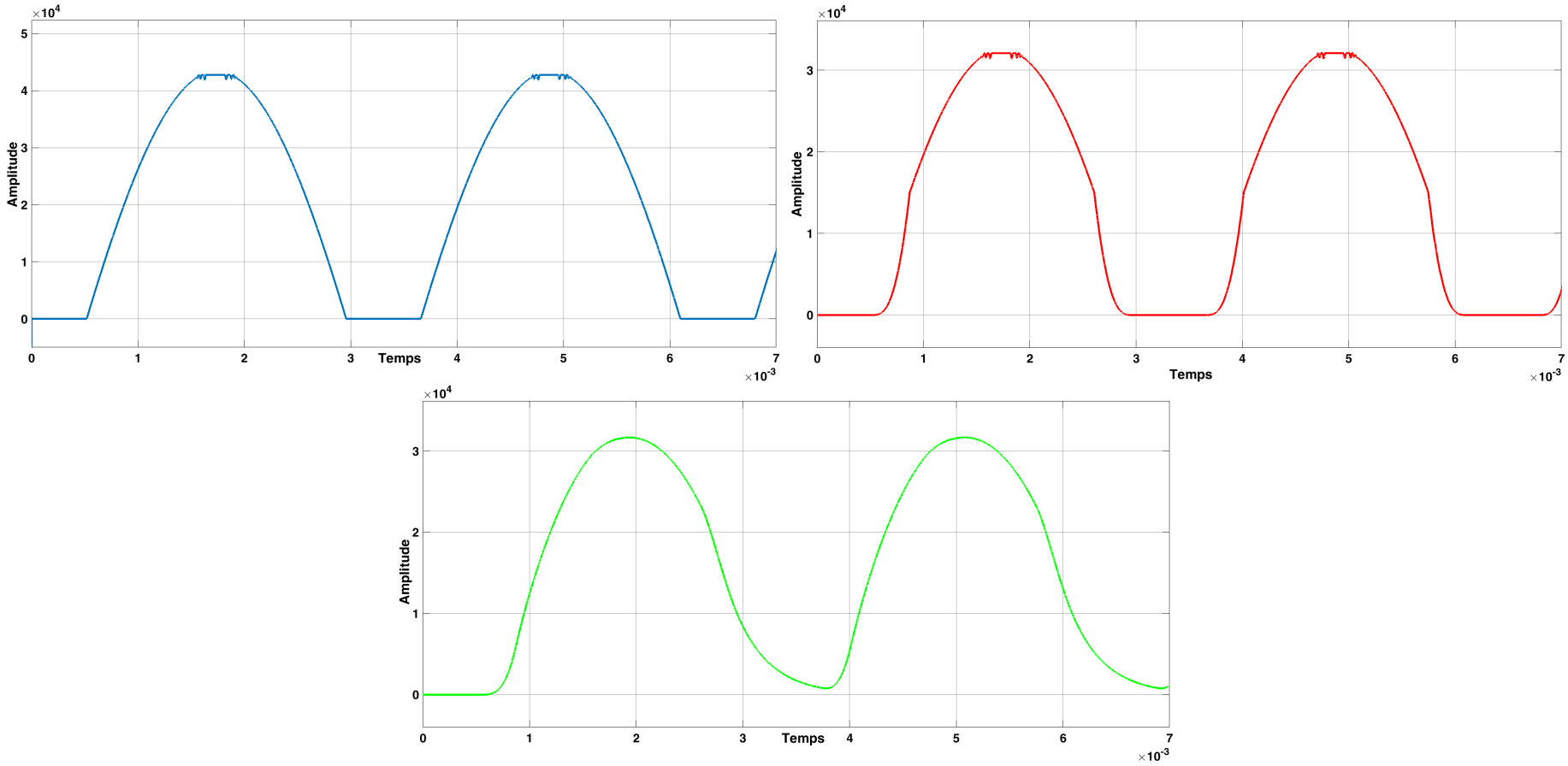


Figure 26 – Signal de la Commande aux Points A (en haut à gauche), B (en haut à droite) et C (en bas à gauche)

3.3.2 Section de Commande

Le système de commande reçoit le niveau d'entrée évalué par le détecteur et a pour but de calculer la réduction de gain nécessaire à la compression, en fonction des paramètres qui lui sont donnés. À une exception près, la totalité des contrôles du compresseur s'adresse au système de commande.

La première étape du calcul de réduction de gain est de déterminer si le niveau d'entrée dépasse le seuil de compression. Pour ce faire, on soustrait la valeur du *threshold* au niveau d'entrée. Puis, un bloc de convertisseur convertis le signal en format non signé. Avec la saturation, cela permet de ramener toutes les valeurs négatives à zéro (Figure 26 a).

Ensuite, on multiplie par $1 - \frac{1}{Ratio}$ avec un bloc de multiplication. Le compresseur comprend un *knee*, le ratio n'est donc pas constant, il dépend du niveau qui dépasse le seuil. On donne en entrée du paramètre de *knee* la moitié de sa largeur. Le calcul du *knee* est explicité dans la partie suivante.
On peut tout de même remarquer que le seuil de compression est abaissé de la moitié de la largeur du *knee* car il est nécessaire de commencer à compresser avant le seuil. Les blocs suivants permettent de réaliser le calcul du *knee*.

Après la multiplication par le ratio, le signal de réduction du gain est passé à travers un filtre de premier ordre qui lui donne les caractéristiques d'attaque et de retour (Figure 26 C). Ce filtre est expliqué par la suite.
Le coefficient du filtre doit changer en fonction de si le compresseur voit une attaque ou un retour. Pour ce faire, on utilise un bloc de comparaison qui contrôle un multiplexeur avec les coefficients d'attaque et de retour à chaque entrée. On compare entre l'entrée et la sortie du filtre de réponse, l'entrée du filtre peut être vue comme un gain cible et la sortie comme le gain actuel. Si le gain cible est plus bas que le gain actuel, on détecte une attaque. Le contraire est un retour.

Après le filtre de réponse, on envoie le signal de réduction dans une sortie *link*. C'est pour pouvoir coupler ce compresseur mono avec un autre pour avoir un compresseur stéréo (Figure 21). Il y a donc aussi une entrée *link* pour récupérer le signal de réduction du gain d'une autre instance

du compresseur. Pour réaliser la fonction de couplage la plus simple, on choisit le signal de réduction qui présente l'amplitude la plus haute. On présente donc les deux signaux avec un bloc de comparaison qui commande un multiplexeur qui, à son tour, commute entre les deux signaux de réduction de gain.

3.3.3 *Knee* : Interpolation du Ratio

Le principe d'un *knee* en compression dynamique est d'avoir un ratio de compression qui s'ajuste progressivement avant et après le seuil, en fonction du niveau d'entrée (Figure 18). Le *knee* est donc caractérisé par sa largeur, qui est centrée sur le seuil du compresseur. Cela sous-entend qu'une partie du signal qui n'a pas encore dépassé le seuil de compression est tout de même compressée, mais avec un ratio plus doux. Dans ce modèle, on remédie à cela en soustrayant la moitié de la largeur du *knee* au seuil.

Afin de récupérer les valeurs intermédiaires du ratio, il faut réaliser une interpolation qui connecte les deux extrémités de la largeur du *knee*. On part de la plage dynamique où il n'y a pas de compression jusqu'à la plage dynamique où le ratio est à son maximum (la valeur réglée par l'utilisateur). La façon la plus simple de réaliser une interpolation est de la faire de manière linéaire. On divise le ratio par la largeur du *knee* pour obtenir une augmentation linéaire du ratio sur toute la largeur du *knee*.
On rappelle que, pour notre modèle, on multiplie le signal qui dépasse le seuil de compression par un facteur pour obtenir le signal de réduction du gain de compression. On nommera ce facteur R_K et voici son expression pour une interpolation linéaire :

$$R_{K1} = \begin{cases} 0 & pour \quad x < T - \dfrac{K}{2} \\ \dfrac{\left(1 - \frac{1}{R}\right) \cdot x}{K} & pour \quad T - \dfrac{K}{2} < x < T + \dfrac{K}{2} \\ 1 - \dfrac{1}{R} & pour \quad x > T + \dfrac{K}{2} \end{cases}$$

Avec x l'amplitude du signal d'entrée avant la soustraction du seuil, R le ratio maximal réglé, T le *threshold* et K la largeur du *knee*.

Cependant, à l'instar d'un potentiomètre de volume, pour avoir une réduction de sonie linéaire, il est préférable d'abaisser l'amplitude d'un signal avec une forme logarithmique. Ainsi, il est nécessaire d'employer une interpolation du second degré qui remplace l'évolution du ratio par une section de parabole [Giannoulis, Massberg et Reiss 2012]. L'expression de R_K devient :

$$R_{K2} = \begin{cases} 0 & pour \quad x < T - \frac{K}{2} \\ \frac{\left(1 - \frac{1}{R}\right) \cdot x^2}{K^2} & pour \quad T - \frac{K}{2} < x < T + \frac{K}{2} \\ 1 - \frac{1}{R} & pour \quad x > T + \frac{K}{2} \end{cases}$$

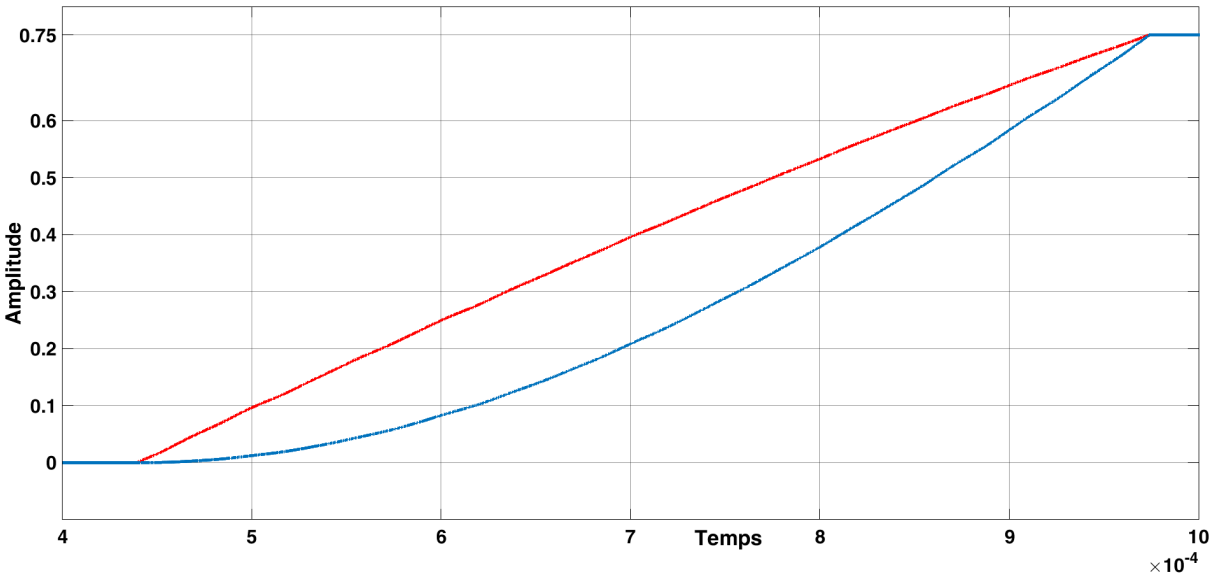


Figure 27 – Comparaison de l'Interpolation Linéaire (rouge) et l'Interpolation de Deuxième Degrés (bleu) d'un Ratio de Compression de 1 : 4

3.3.4 Filtre Passe-Bas du Premier Ordre

3.3.4.1 Calcul du Coefficient

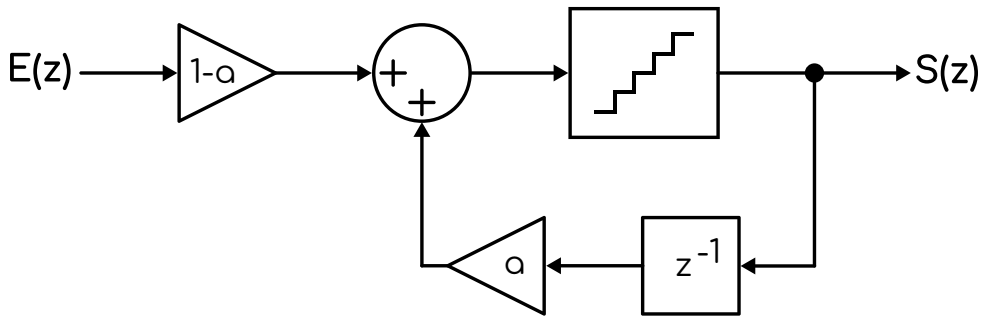


Figure 28 – Schéma d'un Filtre IIR du Premier Ordre en PCM

Faire un filtre à réponse impulsionnelle infinie ou IIR (*Infinite Impulse Response*), de type passe-bas du premier ordre en PCM, repose sur une boucle de rétroaction et un coefficient a. La figure 28 nous montre que le signal est multiplié par $1-a$ avant d'être ajouté à la sortie du filtre, qui est retardée d'un échantillon et multipliée par a. Le quantificateur permet de récupérer le même nombre de bits en sortie qu'en entrée.
Le coefficient a doit être inférieur à 1 pour préserver la stabilité du filtre. En effet, avec une boucle de rétroaction positive, une multiplication par un coefficient supérieur à 1 fera tendre la sortie vers l'infini.

Si on linéarise le quantificateur et on ignore le bruit de quantification, on trouve une fonction de transfert :

$$S(z) = E(s) \cdot (1-a) + a \cdot z^{-1} \cdot S(z) = \frac{E(z) \cdot (1-a)}{1 - a \cdot z^{-1}}$$

$$H_{LP}(z) = \frac{S(z)}{E(z)} = \frac{1-a}{1 - a \cdot z^{-1}}$$

Le détail des calculs peut être trouvé dans [Payen de La Garanderie 2015, Annexe 3, p.90-92]. On obtient alors le calcul du coefficient a en fonction de la fréquence de coupure du filtre et de la fréquence d'échantillonnage :

$$a = 2 - \cos\left(\frac{2 \cdot \pi \cdot f_c}{N_0 \cdot F_s}\right) - \sqrt{\left(\cos\left(\frac{2 \cdot \pi \cdot f_c}{N_0 \cdot F_s}\right) - 2\right)^2 - 1}$$

3.3.4.2 Temps de Montée

Pour un filtre de réponse de compresseur, ce n'est pas la fréquence de coupure qui nous intéresse, mais le temps de montée. En effet, on veut régler le temps que prend le compresseur à ajuster son gain. Nous allons donc voir quel est le rapport entre la fréquence de coupure d'un filtre de premier ordre et son temps de montée.
Il y a plusieurs façons de qualifier le temps de montée d'un filtre. Dans le cas du compresseur celle qui semble être à privilégier est le temps de montée jusqu'à 63 % de la valeur du régime permanent. Néanmoins, beaucoup de compresseurs ne calculent pas le temps de montée de cette façon. Il n'est pas rare de voir le temps de montée de 10 à 90 % ou le temps de compresser un nombre de décibels arbitraire.

Le temps de montée est mise en évidence par la sortie du filtre quand un échelon est appliqué en entrée ; cela s'appelle la réponse indicielle. La constante de temps τ correspond au temps de montée de 63 % ; τ est égal à l'inverse de la pulsation de coupure ω_c. On a donc :

$$\tau = t_{63\%} = \frac{1}{\omega_c} = \frac{1}{2 \cdot \pi \cdot f_c} \approx \frac{0,16}{f_c}$$

On peut s'intéresser au temps de montée de 10 à 90 % qui est égal à $2,2 \cdot \tau$. Pour un temps total de montée, on peut considérer que le régime permanent est atteint au bout de $5 \cdot \tau$. On peut les exprimer facilement en fonction de la fréquence de coupure :

$$t_{10\% \to 90\%} = 2,2 \cdot \tau = \frac{2,2}{\omega_c} = \frac{2,2}{2 \cdot \pi \cdot f_c} \approx \frac{0,35}{f_c}$$

$$t_{total} = 5 \cdot \tau = \frac{5}{\omega_c} = \frac{5}{2 \cdot \pi \cdot f_c} \approx \frac{0,8}{f_c}$$

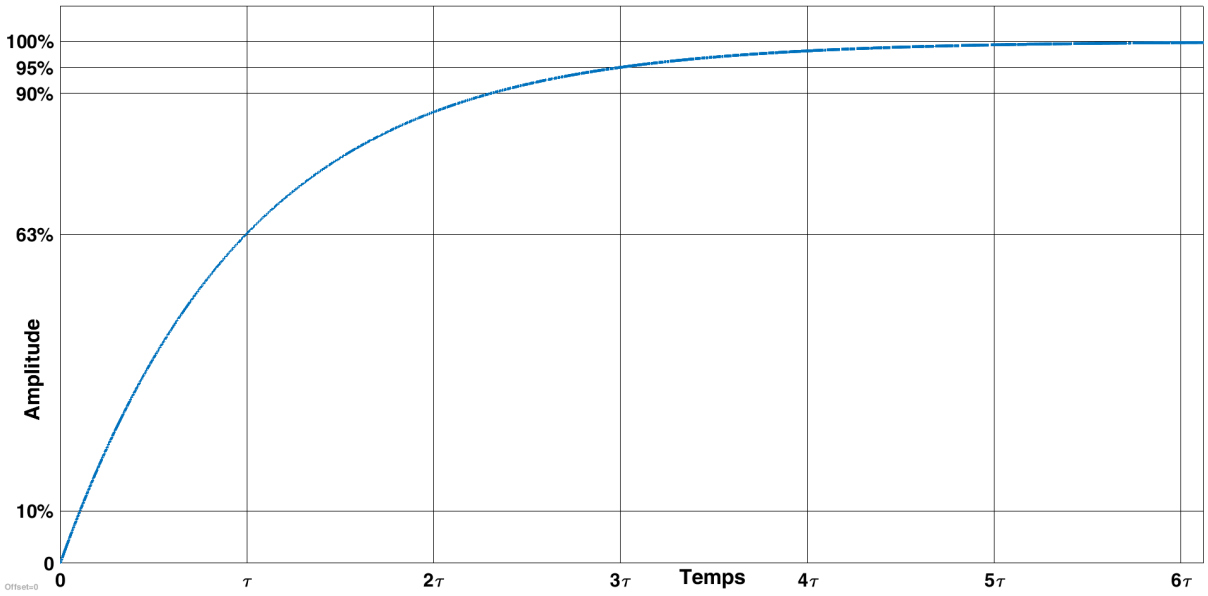


Figure 29 – Réponse Indicielle du Modèle de Filtre Passe-Bas Réalisé dans *System Generator for DSP*

3.3.4.3 Modèle du Filtre

Pour réaliser le filtre de premier ordre dans *System Generator for DSP* on peut s'inspirer directement de la figure 28. Un bloc de conversion sert de quantificateur.
Il faut faire attention à la gestion des retards et faire en sorte qu'il y ai un retard d'un échantillon dans la boucle de contre-réaction, mais pas de retard dans les blocs d'addition et de quantification.
Comme on doit multiplier la valeur des échantillons du signal d'entrée par des coefficients précis, on effectue tous les calculs en virgule flottante (simple précision). Le bloc de conversion en entrée est utilisé pour convertir le signal 16 bits virgule fixe non-signé en *float*. Le bloc de conversion de sortie fait l'opération contraire.
Avec les différents blocs de multiplication, on peut faire une version du filtre avec coefficient constant et un autre avec un coefficient variable.

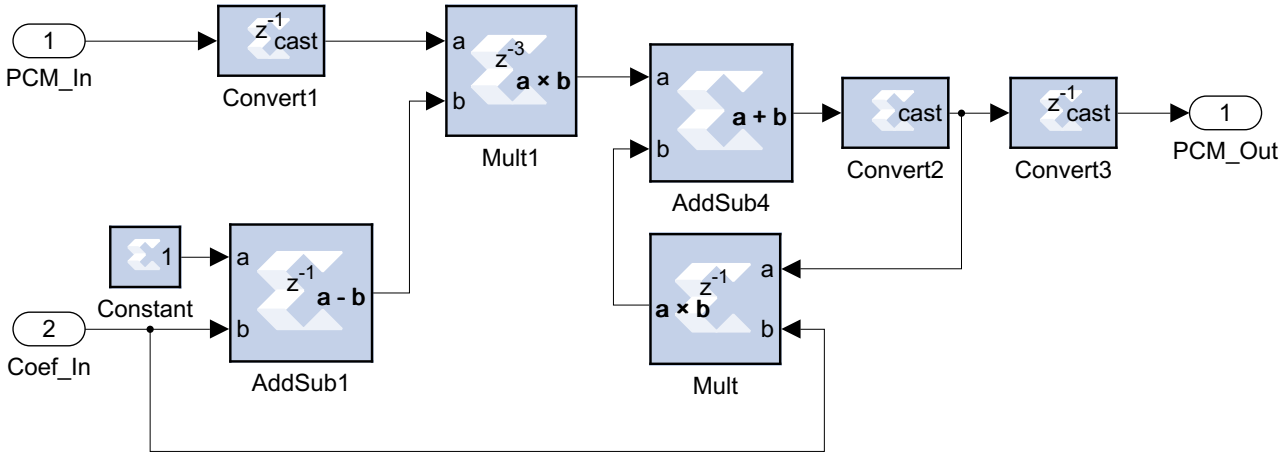


Figure 30 – Modèle *System Generator for DSP* du Filtre Passe-Bas PCM du Premier Ordre à Coefficient Variable

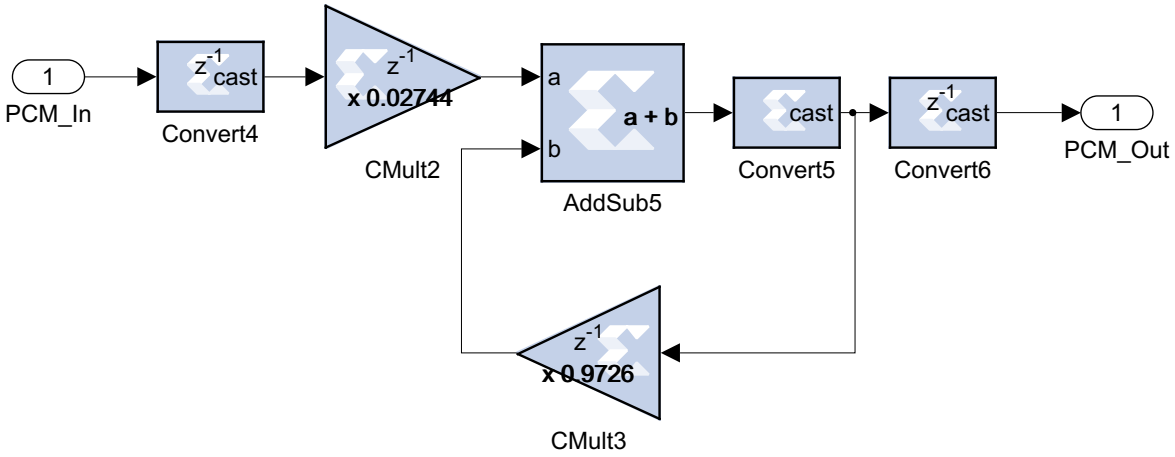


Figure 31 – Modèle *System Generator for DSP* du Filtre Passe-Bas PCM du Premier Ordre à Coefficient Fixe

3.3.5 Ligne à Retard du Look-Ahead

La fonction *look-ahead* d'un compresseur repose sur le fait d'introduire un retard au signal à traiter par rapport au signal détecté du *sidechain* afin d'avoir un compresseur qui "voit" les changements dynamiques en avance.
Dans un système numérique, une ligne à retard variable sous-entend une mémoire tampon à longueur réglable. Le bloc de registre adressable est une mémoire à longueur variable avec un maximum de 1024 bits. En se référant à d'autres compresseurs avec *look-ahead*, il semble qu'un maximum de délai de 10 ms est suffisant. 10 ms d'audio en DSD128 représente une mémoire de :

$$Nb_{10ms} = 128 \cdot 44\,100 \cdot 1 \cdot 10 \cdot 10^{-3} = 56\,448\,bits$$

En divisant ce nombre par 1024, on trouve que 55 blocs de mémoire sont suffisants pour stocker environ 10 ms d'audio Σ/Δ. Il suffit alors de les relier en série et de transmettre la même adresse de mémoire à tous les blocs. Ainsi, chaque incrémentation de l'adresse entraînera 55 bits de retard (ce qui correspond à environ 9,5 µs).

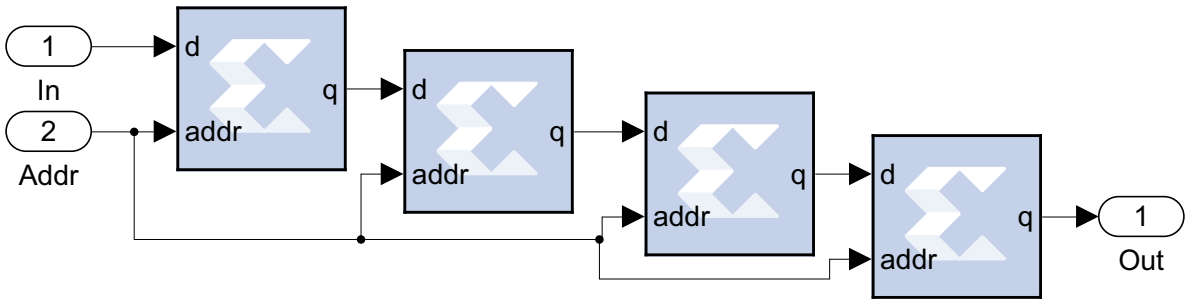


Figure 32 – Modèle Réduit de la Ligne à Retard dans *System Generator for DSP*

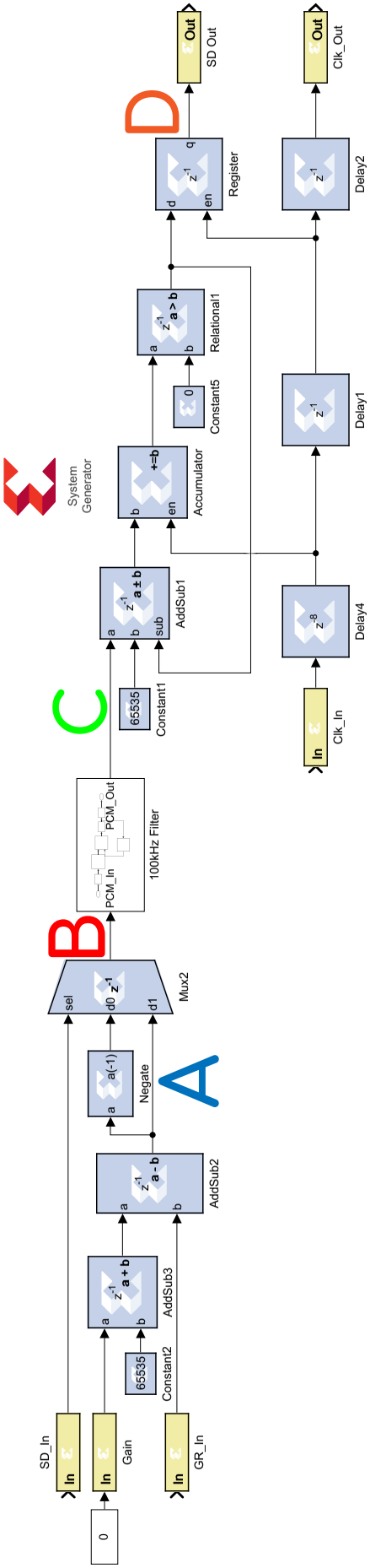


Figure 33 – Modèle *System Generator for DSP* du Bloc de Gain

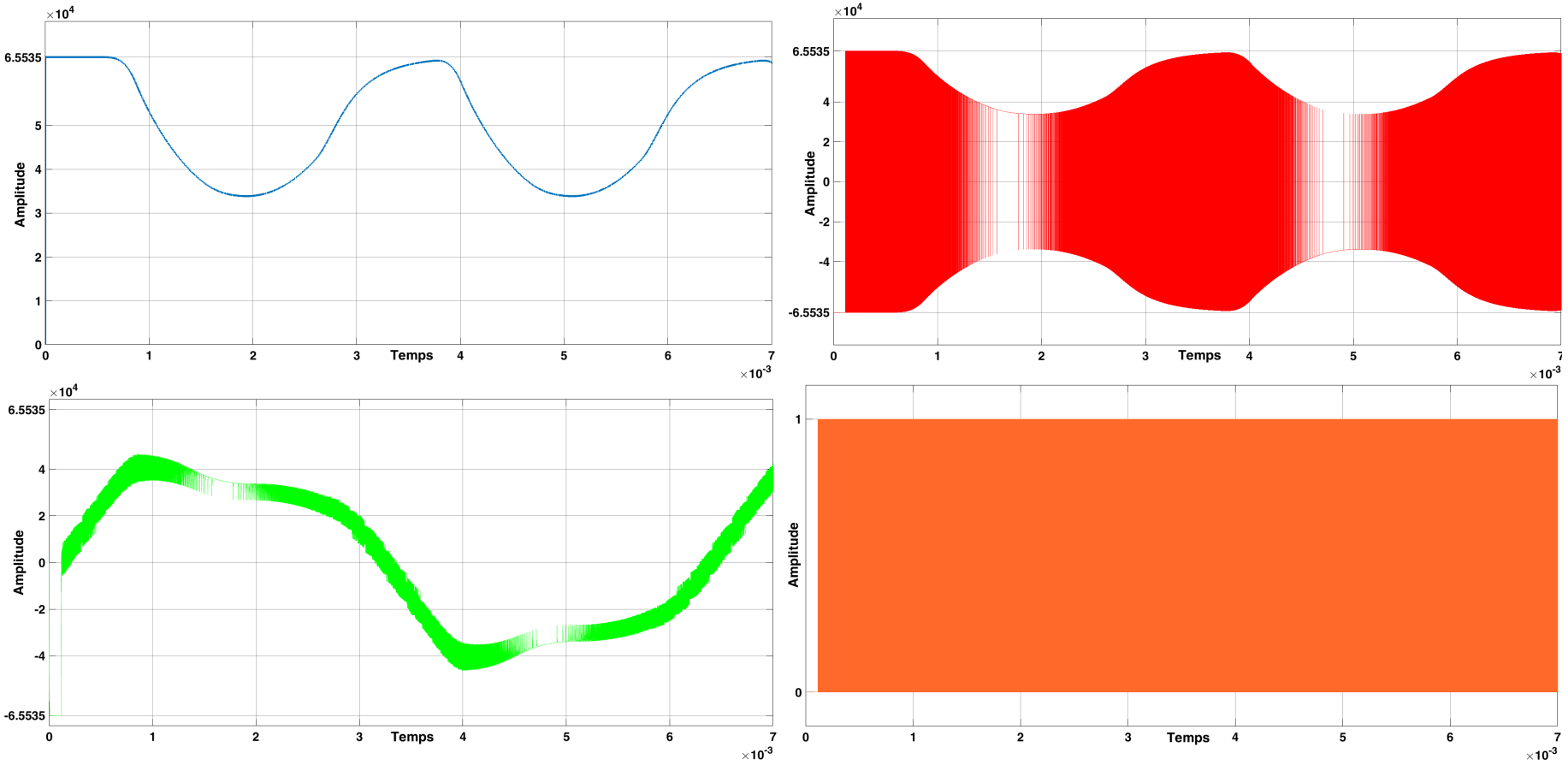


Figure 34 – Signal du Gain aux Points A (en haut à gauche), B (en haut à droite), C (en bas à gauche) et D (en bas à droite)

3.3.6 Module de Gain

Une fois que le niveau de réduction appropriée est calculé, il faut l'appliquer comme gain au flux Σ/Δ.
On commence par soustraire le signal de réduction, calculé dans le système de commande, au gain maximum qu'on souhaite avoir. Ici, le gain maximum souhaité est 65535 (gain unitaire) plus un *make-up gain* choisi par l'utilisateur (Figure 34 A).
Pour appliquer le gain, on donne l'amplitude de ±Gain au signal Σ/Δ (Figure 34 B). On réalise cette opération avec un bloc multiplexeur et un bloc de négation. Avec une amplitude maximale de ±65535, le signal est alors codé sur 17 bits signé.

Pour récupérer un signal sur 1 bit, il faut utiliser un modulateur Σ/Δ. Au préalable, on fait passer le signal multi bit dans un filtre de premier ordre (Figure 31) avec une fréquence de coupure de 100 kHz. Le rôle de ce filtre est de stabiliser le Σ/Δ-M. En effet, si on injecte un signal de fréquence trop élevée dans le Σ/Δ-M cela engendre des distorsions, comme mit en évidence dans la figure 35.

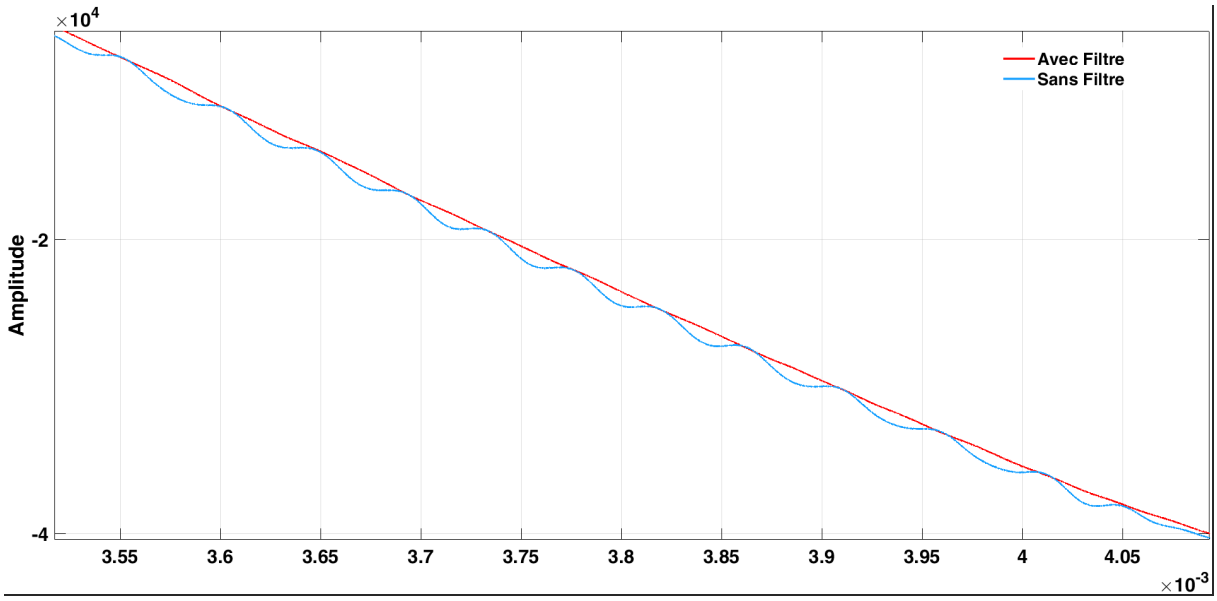


Figure 35 – Comparaison du Signal de Sortie Avec et Sans le Filtre Précédent le Modulateur Σ/Δ

La saturation du format à virgule fixe signé 17 bits fait en sorte que la sortie du filtre ne peut pas être supérieure à ± 65535. Ainsi, même si l'on donne un gain de compensation très fort en entrée, la saturation limitera l'amplitude du signal d'entrée du Σ/Δ-M. Ceci est important, car un Σ/Δ-M

basique pourrait facilement se déstabiliser avec une amplitude d'entrée trop grande. Il est donc préférable de saturer en entrée.

Une fois filtré, on entre donc dans un modulateur Σ/Δ d'ordre 1. Le premier bloc d'addition-soustraction est équivalent à adapter l'amplitude de la sortie 1 bit à celle de l'entrée et de la soustraire à l'entrée. Quand la sortie du modulateur est 1, on soustrait le maximum de l'amplitude d'entrée (65535) au signal d'entrée. Quand la sortie du modulateur est 0 on soustrait le minimum de l'amplitude d'entrée (-65535), ce qui revient à ajouter 65535.
Le bloc suivant accumule l'erreur de quantification résultante. L'entrée *enable* permet de s'assurer que l'accumulation est faite de manière synchrone afin d'avoir une valeur d'erreur de quantification accumulée par échantillon.
Le bloc *relational* à la suite vérifie si l'erreur accumulée est supérieure à 0, ce qui donne un flux Σ/Δ en sortie. Le bloc *register* en sortie est ajouté pour s'assurer que le flux Σ/Δ est transmis de manière synchrone.

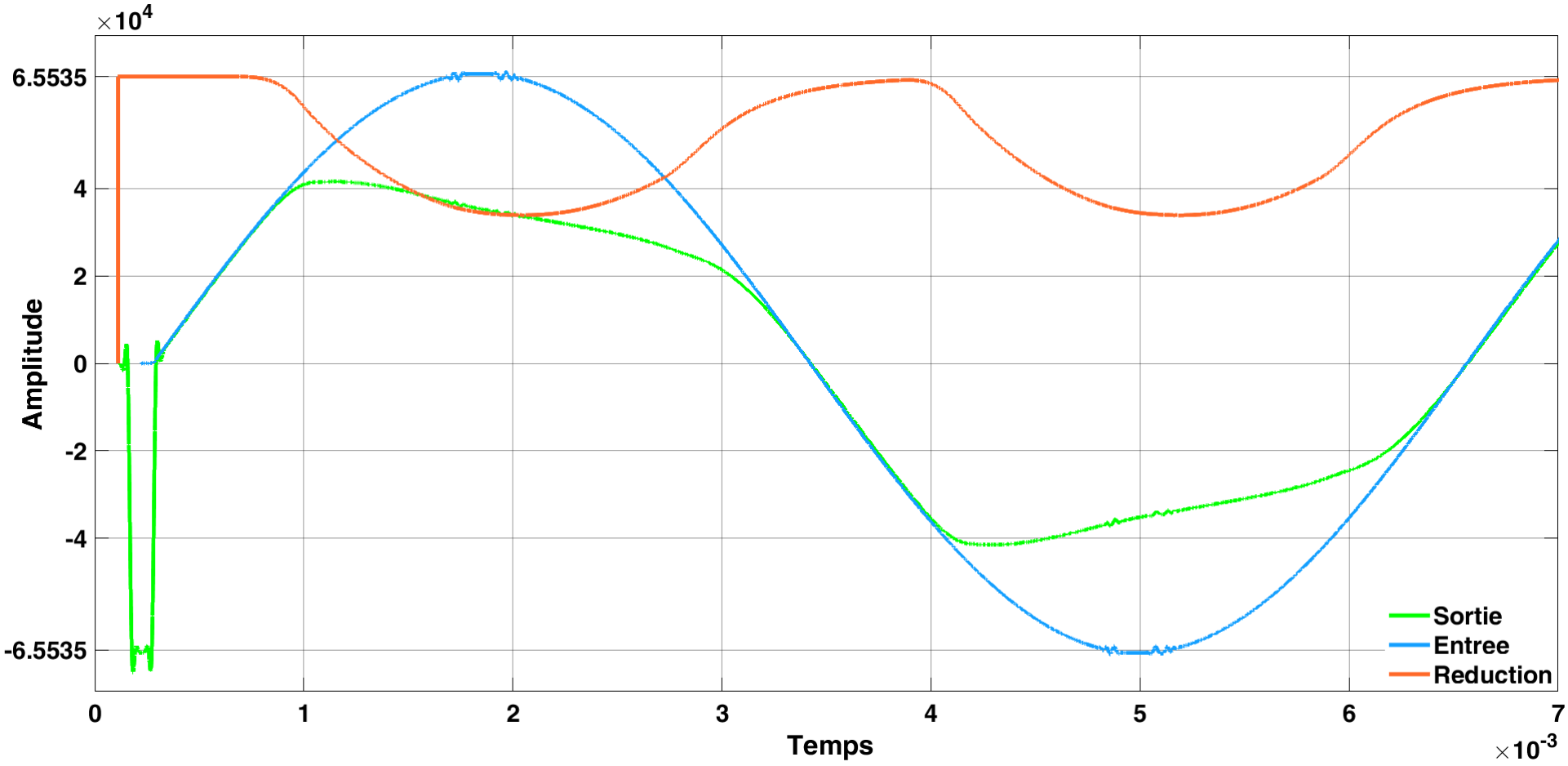


Figure 36 – Comparaison du Signal du Signal de Détecté, du Signal de Sortie Compressé et du Signal de Réduction de Gain (le Délai est Compensé)

4 Réalisation du Compresseur Sigma-Delta sur SoC

4.1 Architecture d'un Système sur Puce

4.1.1 Architecture de Base d'un FPGA

L'un des points forts d'un FPGA est le fait que son circuit peut être reconfiguré entièrement en quelques secondes, pour passer d'une fonction à une autre. La majorité des fabricants choisissent la technologie de mémoire vive statique ou SRAM (*Static Random Access Memory*) pour stocker le programme du FPGA. L'avantage de la SRAM est sa rapidité, le nombre illimité de programmations et sa place réduite sur la puce [Pang et Membrey 2017]. Le principal désavantage vient du fait que c'est une mémoire volatile, ce qui veut dire que le programme du FPGA doit être chargé dans la SRAM depuis une mémoire non volatile au moment de la mise en route. C'est un procédé lent par rapport à une puce programmée dans une mémoire non volatile. La plupart des FPGA possèdent donc une mémoire à lecture seule effaçable électriquement ou EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) qui est prévue pour charger la mémoire vive au démarrage.
Il existe d'autres technologies de programmation que la SRAM, comme la mémoire Flash ou Antifuse qui sont non-volatile. Mais elles n'égalent pas la performance de la SRAM et sont utilisées dans des contextes spécifiques.

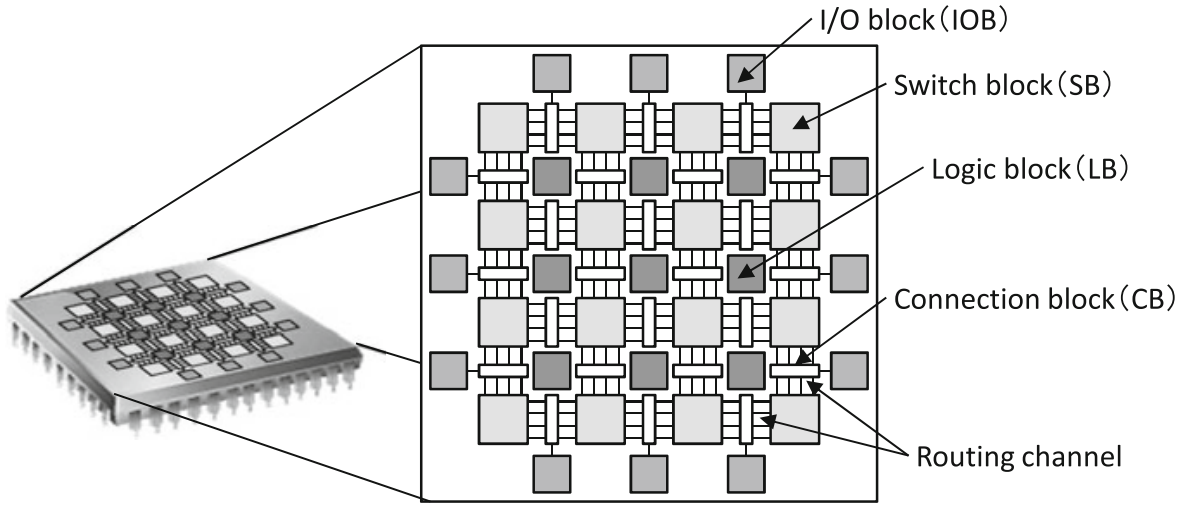


Figure 37 – Architecture Classique "en Îles" d'un FPGA [Amano 2018, Firgure 2.1, p.24]

L'architecture d'un FPGA repose sur quatre blocs élémentaires :

— Le bloc logique ou LB (*Logic Block*) : C'est une porte logique programmable dont l'élément principal est une table de correspondance ou LUT (*Look-Up Table*). D'autres éléments peuvent être présents, comme un circuit de bascule pour pouvoir synchroniser le bloc à une horloge, ou un multiplexeur pour choisir le signal à présenter en sortie (Figure 38).

— Le bloc d'entrée/sortie ou IOB (*Input/Output Block*) : C'est le bloc responsable de gérer les connexions entre le circuit interne et les broches de la puce. Ils sont aussi chargés de configurer chaque broche comme une entrée ou une sortie ainsi que de connecter des résistances entre l'alimentation et la broche si nécessaire.

— Le bloc de connexion ou CB (*Connection Block*) : Ce sont des connexions qui sont chargées de relier les LB entre eux et avec les IOB.

— Le bloc commutateur ou SB (*Switch Block*) : Les SB sont des commutateurs programmables qui servent à acheminer les signaux aux bons endroits. Ce sont eux qui gèrent toutes les connexions faites par les CB.

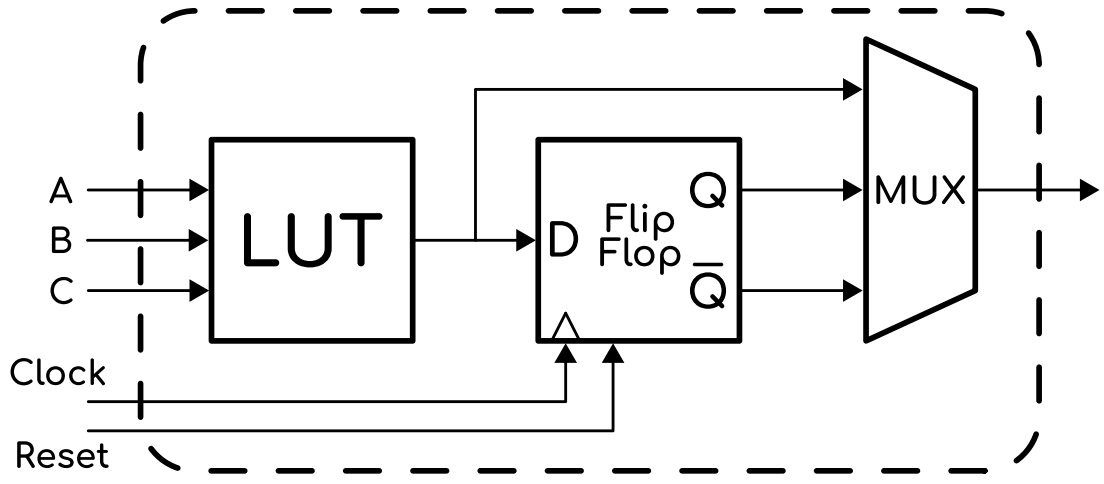


Figure 38 – Schéma Simplifié d'un Bloc Logique [d'après Pang et Membrey 2017, Figure 1.2, p.5]

Un exemple permet de mieux comprendre les rôles joués par chacun des blocs. De plus, nous on verra quelles parties ont besoin d'êtres programmées dans le FPGA. Si nous souhaitons réaliser un circuit logique qui allume une diode électroluminescente ou LED (*Light-Emitting Diode*) lorsqu'une majorité d'interrupteurs est appuyée.

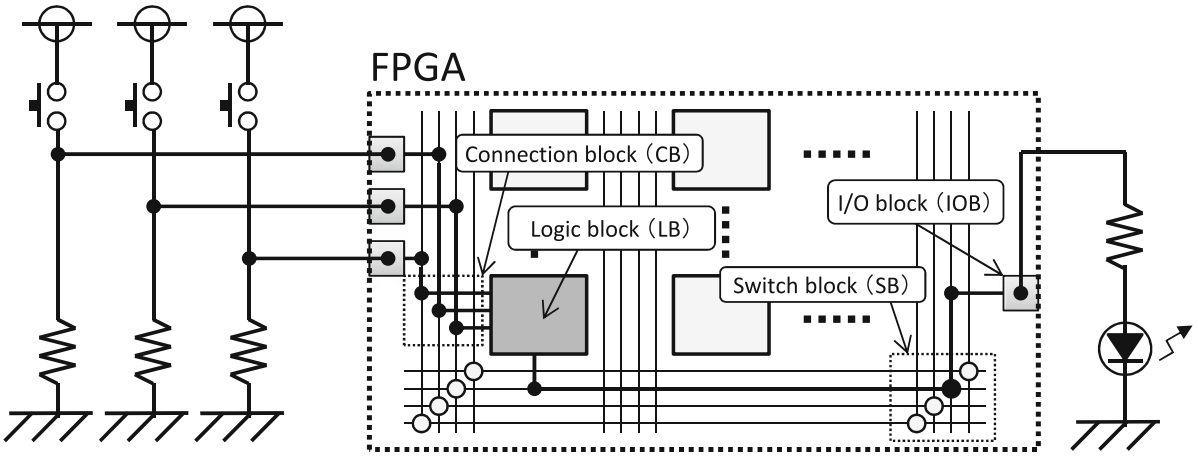


Figure 39 – Circuit Logique de Majorité à Travers un FPGA [Amano 2018, Figure 2.11, p.35]

La figure 39 nous montre la place occupée par le circuit logique "de majorité" dans le FPGA. Les interrupteurs, forcément extérieurs à la puce, sont connectés aux broches du FPGA de façon à présenter un état logique haut lorsqu'ils sont appuyés.
Les broches du FPGA sont chacune contrôlées par un bloc d'entrée/sortie. Puis, les signaux sont acheminés à un bloc logique. Nous verrons qu'il n'y a pas besoin de plus d'un LB pour réaliser le circuit logique souhaité. La sortie du LB est alors acheminée via un bloc commutateur à la broche connectée à la LED.
On voit que dans le FPGA il faudra configurer les IOB pour configurer trois entrées et une sortie. Le SB doit être programmé afin d'acheminer la sortie du circuit logique à la bonne broche. Enfin, le LB dois être programmé pour réaliser l'opération logique désirée, c'est-à-dire, donne une sortie dont l'état logique représente celui de la majorité de ses entrées.

Comme le montre la figure 40, une LUT est une table de correspondance. Dans notre cas, elle représente l'évolution des valeurs prise par la sortie (f) en fonction des valeurs prises par les entrées (A,B,C). La figure 40 a représente la table de vérité de la fonction logique (f) en fonction des variables logiques (A,B,C). La figure 40 b montre son implémentation.

Dans un FPGA, les résultats possibles (colonne f de la figure 40 a) sont stockés dans la mémoire vive. C'est de cette manière qu'il est possible d'avoir de la logique programmable. Pour avoir une porte logique qui rend le résultat de la majorité de ses trois entrées il suffit donc de programmer la LUT du bloc logique comme dans la figure 40.

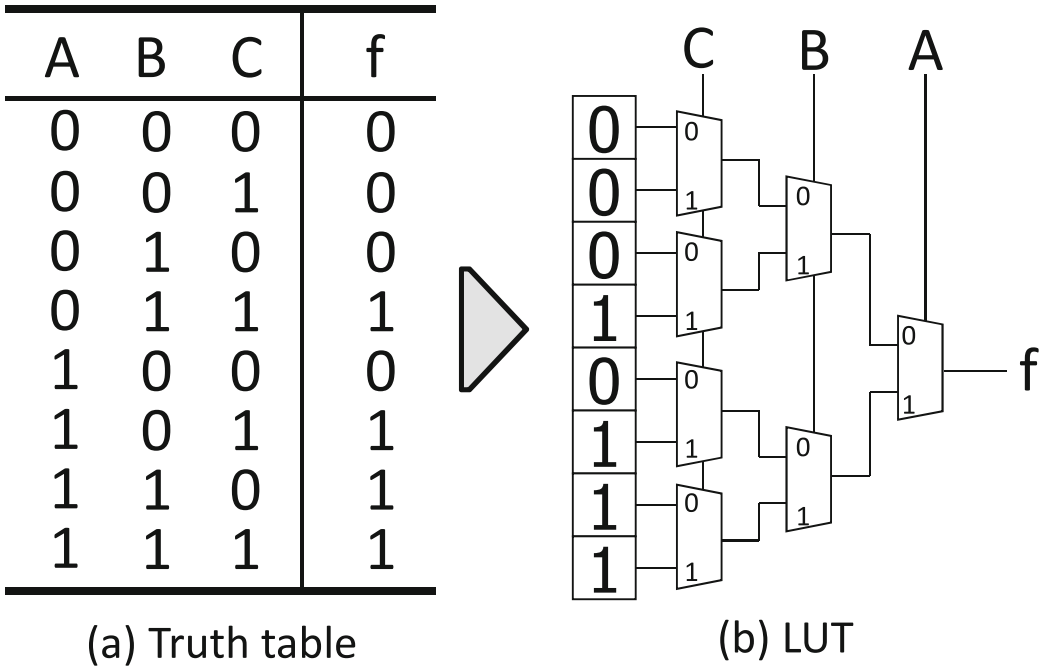


Figure 40 – LUT Programmable dans le Circuit de Majorité [Amano 2018, Figure 2.16, p.37]

4.1.2 Bloc DSP48E1

Il n'est pas rare de voir des blocs spécialisés incorporés dans l'architecture d'un FPGA. C'est le cas du FPGA utilisé dans ce mémoire. Il est équipé de 220 blocs DSP48E1. Comme leur nom l'indique, ce sont des calculateurs destinés au traitement numérique du signal. Le bloc comporte quatre entrées et peut faire ce genre d'opération sur elles :

$$P = \big((A \pm D) \times B\big) \pm C$$

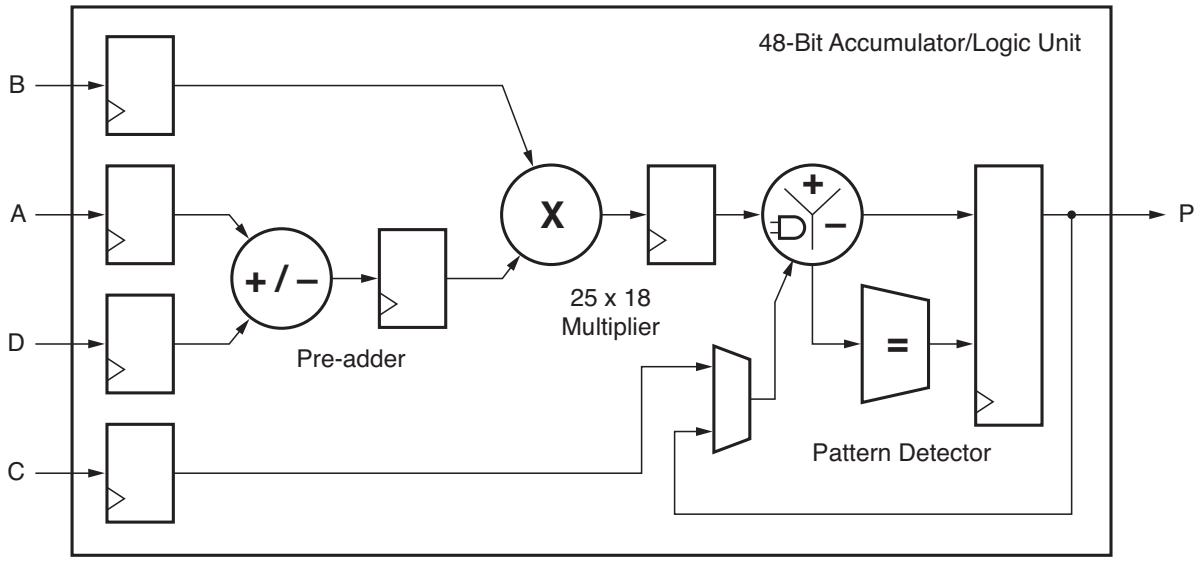


Figure 41 – Schéma Fonctionnel d'un Bloc DSP48E1 [*7 Series DSP48E1 User Guide* 2018, Figure 1.1, p.9]

Dans le cas du traitement numérique du signal, il est pertinent d'ajouter des blocs spécialisés de multiplication-addition. En effet, réaliser un multiplicateur avec des portes logiques programmables est coûteux en termes de nombre de blocs logiques et en temps nécessaires. Un bloc DSP, disposant de son circuit de multiplication et d'addition dédié, sera plus rapide et économisera l'utilisation des blocs logiques.
C'est particulièrement utile dans le cas des opérations rencontrées en traitement du signal qui nécessitent souvent des opérations de multiplication-addition, comme un filtre FIR, par exemple. Le filtre FIR du circuit de détection du compresseur en utilise environ 85 blocs DSP.

4.1.3 Architecture de Base d'un SoC

La particularité des SoC de la série Zynq-7000 de Xilinx est d'associer en une seule puce, une partie de logique programmable ou PL(*Programmable Logic*) et une partie processeur ou PS (un processeur ARM désigné PS pour *Processing System*). Ces deux parties sont reliées par un bus AXI (*Advanced eXtensible Interface*).

Dans notre cas, la partie PL sert à traiter le flux DSD et la partie PS sert à créer une interface utilisateur afin de régler les paramètres du compresseur. On utilisera également la partie PS pour initialiser les puces de convertisseurs.

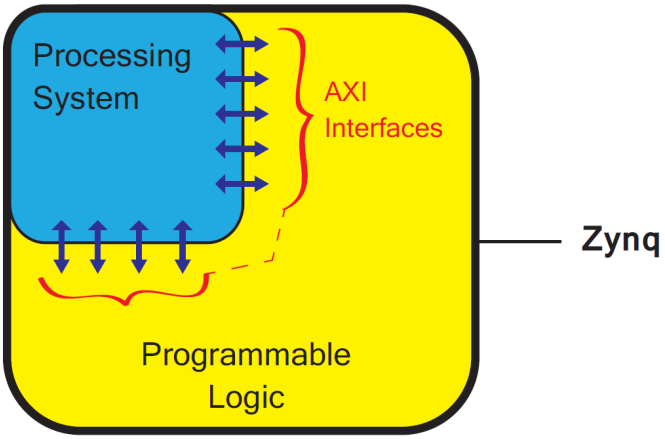


Figure 42 – Schéma Simplifié de l'Architecture du Zynq-7000 [Crockett et al. 2014, Figure 1.2, p.4]

4.2 Présentation de la Chaîne d'Outils

4.2.1 Plateforme de Développement : Zedboard

Afin de faciliter le développement sur une puce aussi complexe, nous utiliserons la plateforme de développement Zedboard proposée par Digilent. En effet, la Zedboard comprend les alimentations nécessaires au SoC, un port JTAG pour la programmation et le debug. De plus, les ports d'extension "P-Mod" vont nous permettre de connecter facilement nos circuits de conversion au SoC.
Le modèle exact du SoC utilisé est XC7Z020-CLG484-1 de la série Zynq 7000 de Xilinx. La partie PS est constituée d'un processeur à deux cœurs ARM Cortex-A9.
ARM est une entreprise qui vend aux fabricants d'électronique des licences d'exploitation de propriétés intellectuelles ou IP (*Intellectual Property*). Ces IP sont des architectures de processeurs. Le Cortex-A (A pour *Application*) est très populaire, par exemple il est au cœur d'une grande partie des produits commercialisés par Apple.

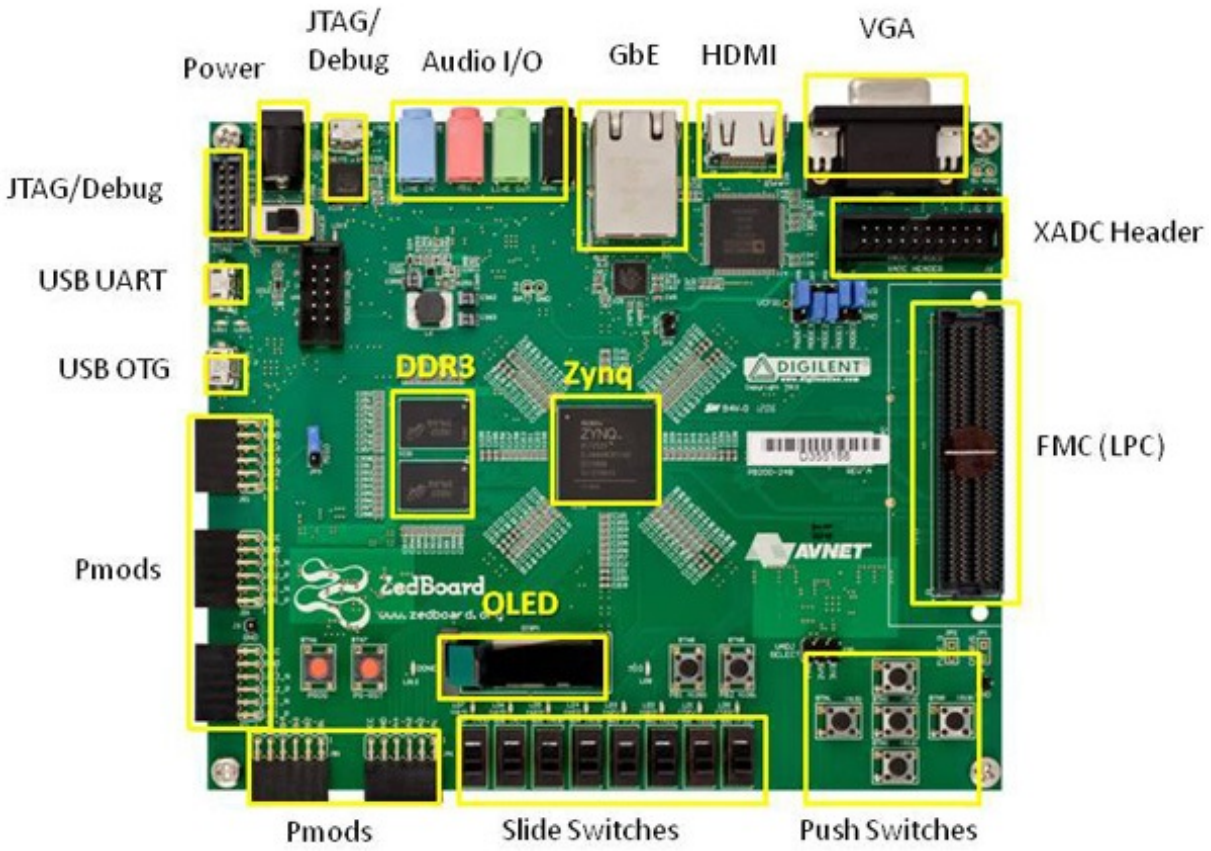


Figure 43 – La Plateforme de Développement Zedboard [Avnet p. d.]

4.2.2 Environnement Xilinx

4.2.2.1 Vitis & Vivado

Depuis octobre 2019, Xilinx a regroupé son environnement de développement intégré ou IDE (emphIntegrated Development Environment) sous le nom *Vitis*.
Vitis lui-même est chargé du développement de la partie PS des SoC et est basé sur le projet open source *Eclipse*. On code en C ou C++ des applications qui seront exécutées par le processeur.

L'autre logiciel important est *Vivado* qui s'occupe du développement de la partie PL.
Vivado supporte plusieurs façons d'intégrer des fonctions aux FPGA. Il est possible de coder directement dans un des langages de description matérielle ou HDL (*Hardware Description Language*). *Vivado* supporte le VHDL (*Very high speed integrated circuit HDL*) ou le Verilog.
Les HDL sont, comme leur nom l'indique, des langages qui permettent de créer des circuits électroniques numériques seulement en les décrivant en

code. Ces circuits vont être implémentés au sein du FPGA.

Une autre façon plus rapide de coder une FPGA dans *Vivado* est de connecter de façon graphique des blocs de fonctions. Ces blocs sont des IP qui représentent chacun un circuit numérique de complexité variable. Nous verrons que les blocs fournis par *Vivado* sont souvent des blocs qui se connectent au bus AXI d'un processeur et agissent comme des périphériques. La partie PS est aussi représentée en tant que bloc dans *Vivado*, mais ce bloc ne représente pas de circuit, il sert à échanger des signaux avec le FPGA. Par exemple, l'horloge, le *reset* et les entrées/sorties du bus AXI.

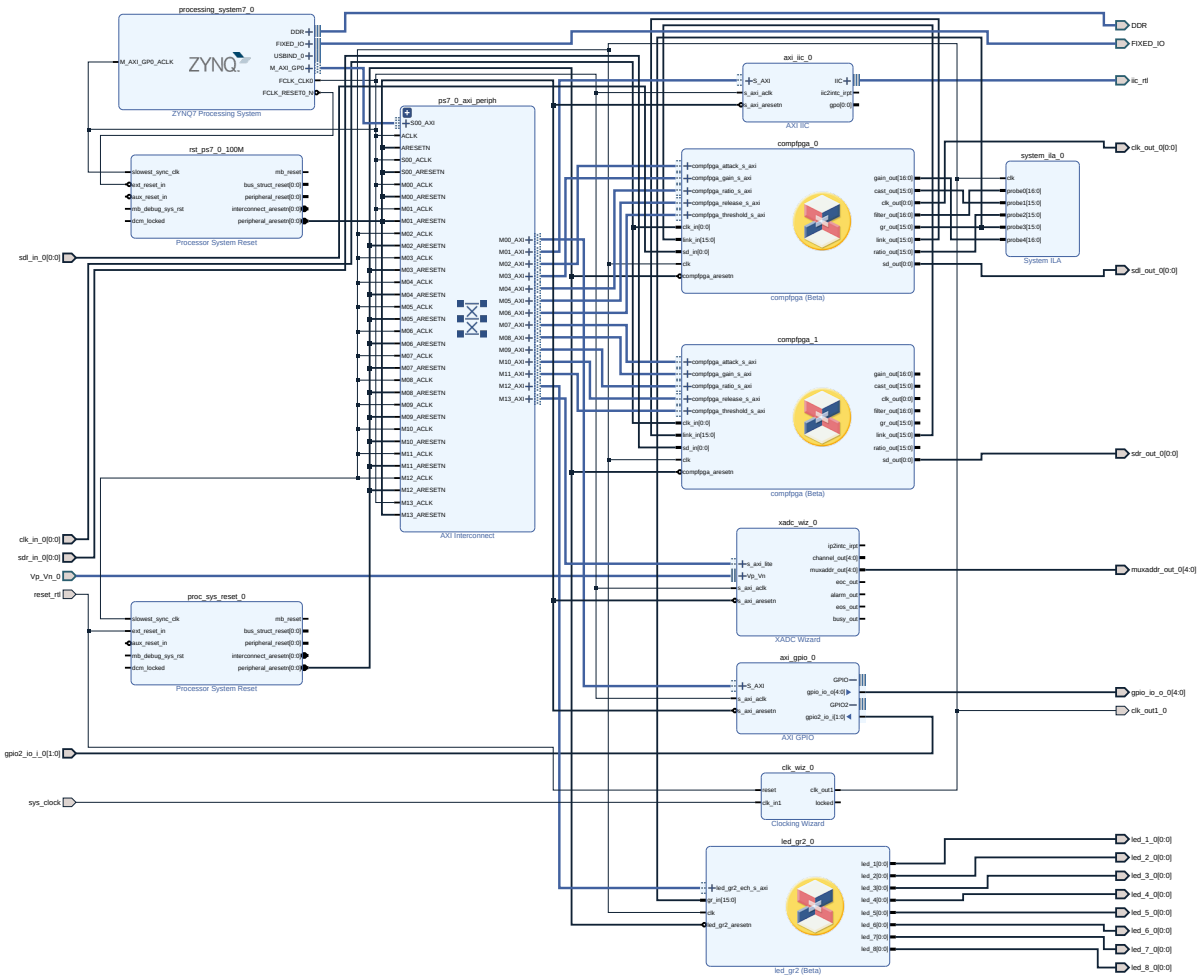


Figure 44 – Système de Blocs de Fonctions du Compresseur dans *Vivado*

4.2.2.2 System Generator for DSP

Il est possible de créer ses propres blocs de fonctions afin que le FPGA réalise une opération spécifique absente du catalogue d'IP Xilinx. Une possibilité est de coder le bloc en langage C. Cela s'appelle HLS (High Level Synthesis) et c'est une façon d'exploiter les avantages des langages de haut niveau comme le C pour créer des circuits logiques complexes sans passer par le HDL.

Pour coder un bloc de façon graphique, Xilinx met à disposition une extension de *Simulink* intitulée *System Generator for DSP*.
Simulink est un logiciel développé par MathWorks en complément de Matlab. C'est un logiciel de modélisation physique qui fonctionne par blocs de fonctions à connecter. De manière simple, *System Generator for DSP* est un jeu de blocs de fonctions supplémentaires dans *Simulink*. Ces blocs Xilinx correspondent à des circuits électroniques logiques élémentaires. De plus, comme dans la figure 45, un système de blocs peut être directement testé avec les blocs de base de *Simulink*, comme un oscilloscope ou un analyseur de spectre. Une fois compilé, un système réalisé grâce à *System Generator for DSP*, peut être importé dans *Vivado* en tant que bloc de fonctions. Dans la figure 44 les deux blocs estampillés du logo de *System Generator for DSP* ont été créés dans *Simulink*

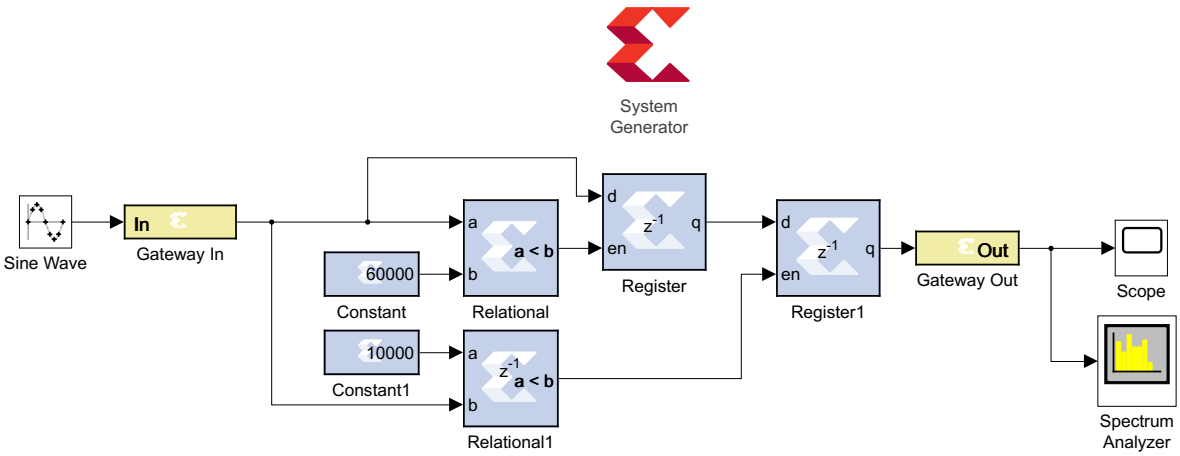


Figure 45 – Un Circuit d'Écrêtage Réalisé avec des Blocs *System Generator for DSP* et *Simulink*

4.3 Circuits de Conversion

La Zedboard est équipée de convertisseurs dans le SoC ainsi que sur puce externe. Cependant, il n'est pas possible de les utiliser pour convertir le signal d'entrée du compresseur. En effet, nous avons besoin de transformer le signal audio analogique en un flux Σ/Δ. Les puces de convertisseurs conçues pour l'audio et qui proposent une sortie Σ/Δ ne sont pas communes. Nous allons utiliser la PCM4202 [*24 Bits, 216 kHz Stereo Audio ADC* 2004] ainsi que la DSD1793 [*24 Bits 192 kHz Audio Stereo DAC* 2006].

Pour connecter nos circuits de conversion à la Zedboard, nous allons utiliser les ports P-Mod qui sont une façon rapide d'accéder aux broches du SoC ainsi qu'à une alimentation 3,3 V. J'ai la chance de pouvoir réutiliser le circuit mis au point par Paul Payen de la Garanderie dans le cadre de son travail mémoire de 2015 [Payen de La Garanderie 2015]. Nous allons tout de même exposer le contenu de ce circuit et ses fonctionnalités.

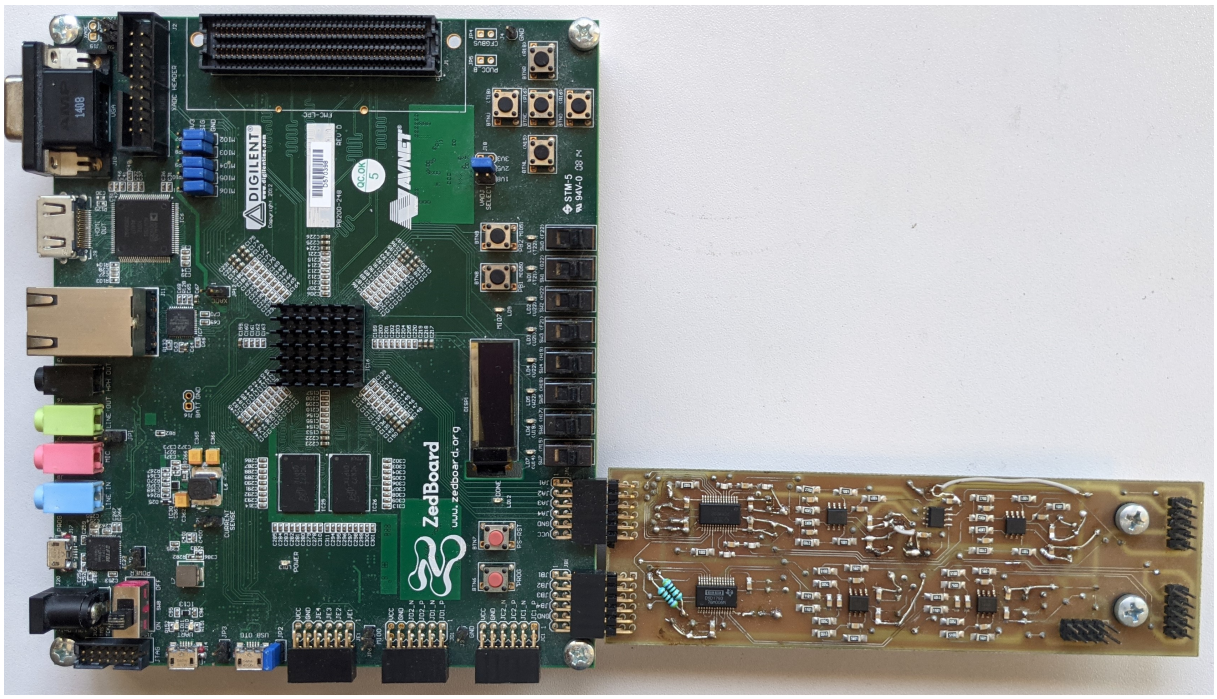


Figure 46 – La Zedboard Connectée aux Convertisseurs Via les Ports P-Mod

4.3.1 ADC

4.3.1.1 Électronique

Avant de faire entrer le signal analogique à convertir dans la puce de convertisseur, il faut préparer le signal. C'est-à-dire faire une adaptation d'impédance, mais aussi contrôler la bande passante et l'amplitude. C'est le rôle de l'étage d'entrée de la figure 47. Ce schéma est celui proposé dans la fiche technique de la puce d'ADC [*24 Bits, 216 kHz Stereo Audio ADC* 2004, Figure 13-14, p.22].

L'entrée du convertisseur est différentielle, on utilise donc un amplificateur opérationnel différentiel OPA1632 comme base de l'étage d'entrée. Il est tout de même facile de fournir une source asymétrique en entrée, il suffit de relier l'entrée négative à la masse.

L'étage d'entrée accepte une amplitude d'entrée maximale de $\pm$13,79 V (+22 dBu) et réduit cette amplitude à $\pm$3 V, soit un gain de 0,27.
Pour pouvoir utiliser des sources sonores plus faibles on pourra modifier le gain du circuit en modifiant la valeur des résistances R5 et R6. Pour une source à +4 dBu (3,47 V_{pp}) un gain de 1,73 est préférable (R5 = R6 = 1730 Ω).
L'entrée V_{COM} de l'OPA1632 est utilisée pour ajouter une tension continue à ses deux sorties. C'est le convertisseur qui fournit la tension en entrée de V_{COM}, celle-ci est acheminée par un circuit de suiveur, réalisé avec un amplificateur opérationnel OPA2134. Cette tension continue est ajoutée en sortie, car la plage de signal d'entrée acceptée par l'ADC est 0-6 V. La tension V_{COM} permet donc de passer d'un signal de $\pm$3V à un signal de 0-6V.

Dans la figure 48 on voit que l'ADC utilise 2 alimentations asymétriques différentes. +3,3 V pour la partie numérique et +5 V pour la partie analogique. Les sorties audionumériques ainsi que les broches de configuration sont connectées aux broches du connecteur P-Mod. Le signal d'horloge, indispensable à la conversion, provient du P-Mod dédié au DAC.

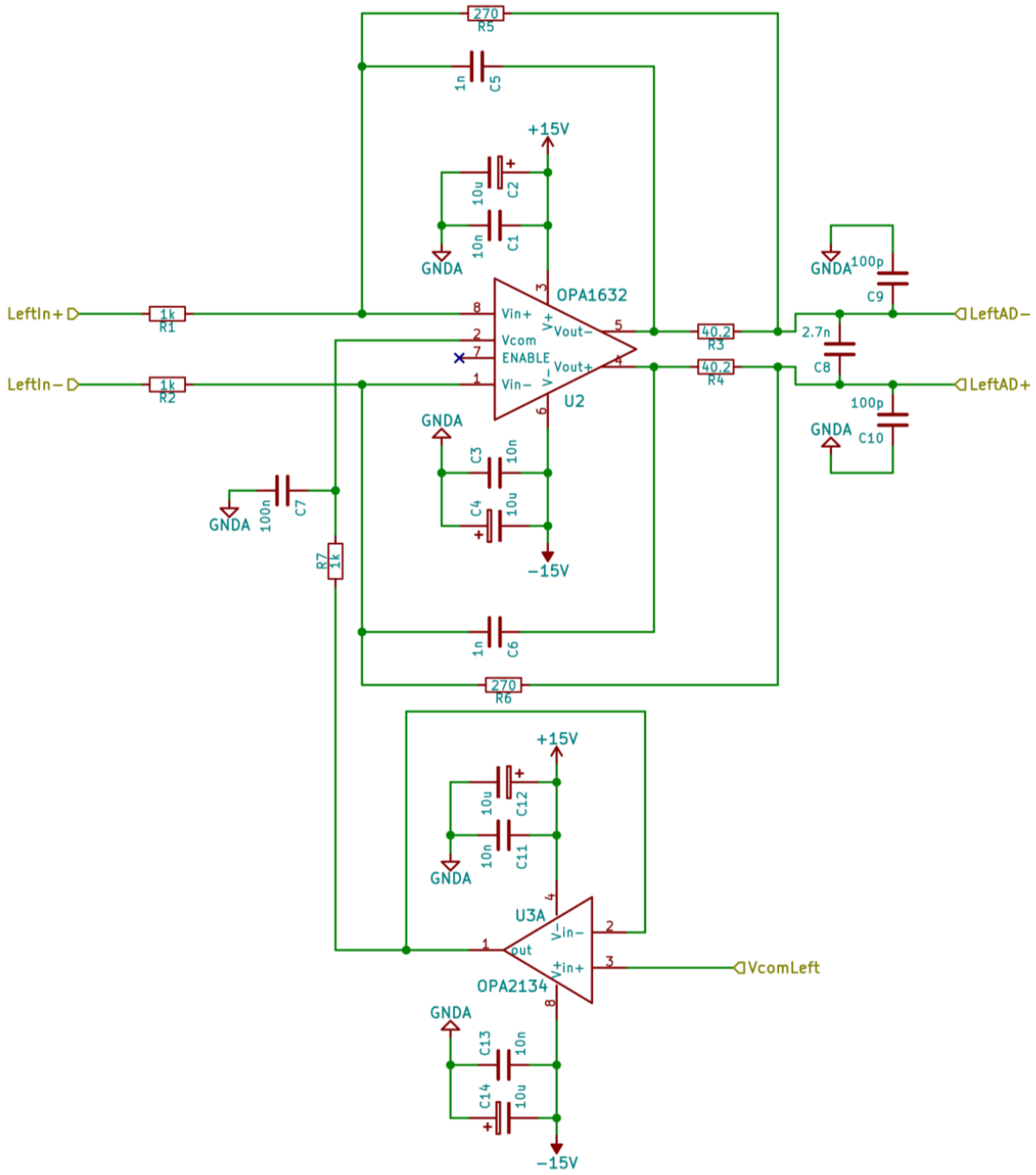


Figure 47 – Étage d'Entrée de l'ADC [Payen de La Garanderie 2015, Figure 39, p.50]

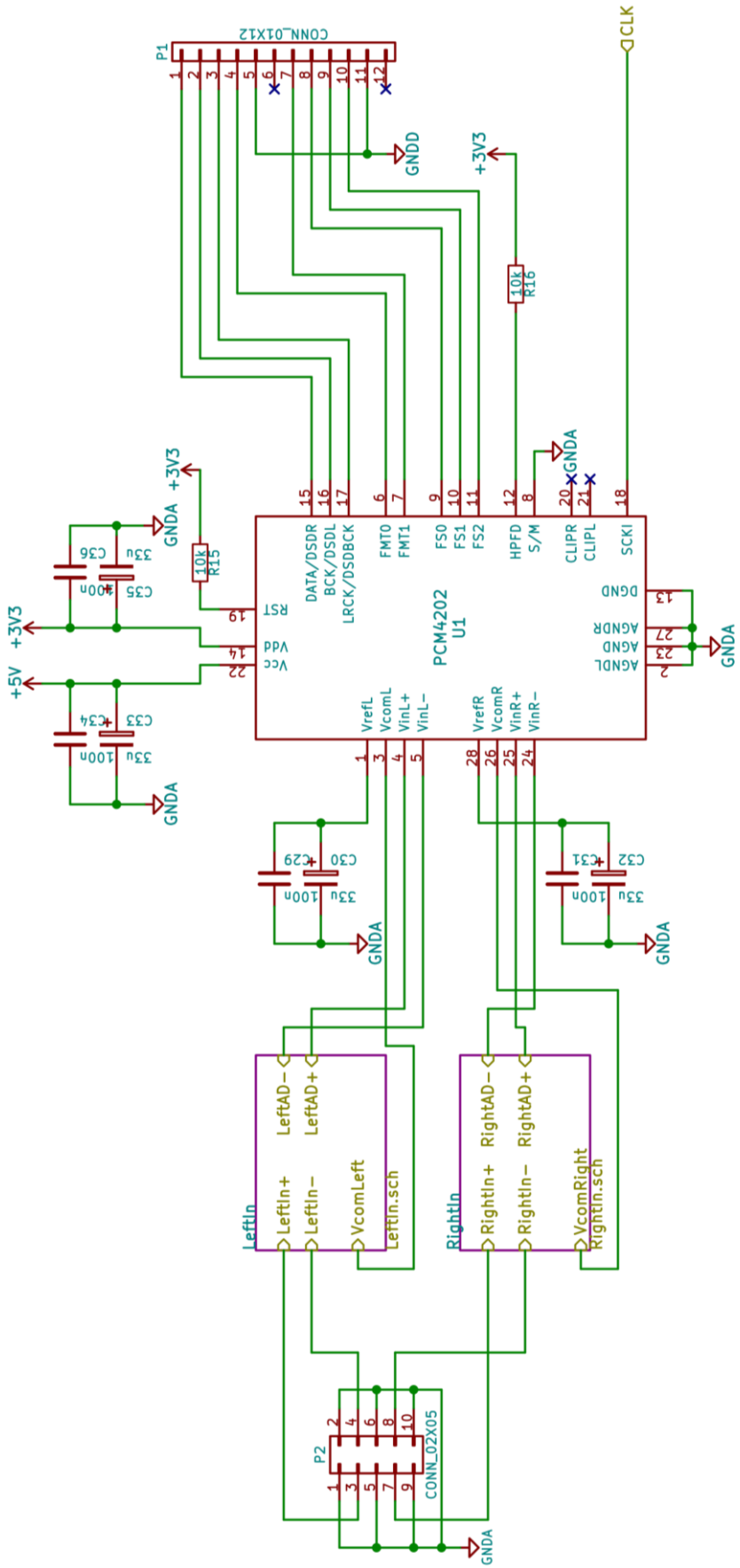


Figure 48 – Circuit de la Puce d'ADC PCM4202 [Payen de La Garanderie 2015], Figure 40, p.51

4.3.1.2 Configuration

La configuration de la puce PCM4202 peut être effectuée simplement avec l'état logique appliqué à cinq de ses broches. L'état logique des broches $FMT[0:1]$ sélectionne le format audio entre DSD et plusieurs types de PCM. L'état des broches $FS[0:2]$ sélectionne le mode d'échantillonnage; c'est-à-dire qu'il fixe le facteur de suréchantillonnage du flux DSD et celui de l'horloge système.

FMT1	FMT0	AUDIO DATA FORMAT
0	0	24-bit Left Justified
0	1	24-bit I^2S
1	0	24-bit Right Justified
1	1	1-bit DSD (Master Mode Only)

FS2	FS1	FS0	SAMPLING MODE
0	0	0	$128f_S$ DSD Output Rate with $f_{SCKI} = 768f_S$
0	0	1	$128f_S$ DSD Output Rate with $f_{SCKI} = 512f_S$
0	1	0	$128f_S$ DSD Output Rate with $f_{SCKI} = 384f_S$
0	1	1	$128f_S$ DSD Output Rate with $f_{SCKI} = 256f_S$
1	0	0	$64f_S$ DSD Output Rate with $f_{SCKI} = 384f_S$
1	0	1	$64f_S$ DSD Output Rate with $f_{SCKI} = 256f_S$
1	1	0	Reserved
1	1	1	Reserved

Figure 49 – Tableaux de configuration du PCM4202 [*24 Bits, 216 kHz Stereo Audio ADC* 2004]

On doit régler la puce en mode DSD avec un facteur de suréchantillonnage de 128 pour le flux DSD (5,6448 MHz) et 768 pour l'horloge système (33,8688 MHz). Pour faire cela, on a créé une fonction d'initialisation exécutée par le processeur du SoC lors du démarrage du système. Il s'agit juste de mettre les broches adéquates à l'état logique haut ou bas.

Cette action demande d'utiliser un périphérique d'entrée/sortie à usage général ou GPIO (*General Purpose Input Output*). Il s'agit un bloc IP, disponible dans Vivado, qu'on connecte au processeur via le bus AXI. Un pilote informatique (ou *driver*) appelé *xgpio* est mis à disposition par Xilinx pour

contrôler le GPIO.
On écrit la fonction *PCM4202_init* qui permet d'initialiser l'ADC en choisissant un taux de suréchantillonnage pour le bus DSD. On remarquera que chaque bit des nombres *INIT64* et *INIT128* correspond à l'état d'une broche du GPIO.

Voici le code en C dans *Vitis* utilisé pour initialiser la puce PCM4202 :

```c
#include "xparameters.h"
#include "xgpio.h"

//On va chercher le numéro du périphérique GPIO dans xparameters
#define GPIO_ID     XPAR_GPIO_0_DEVICE_ID

//On définit les états nécessaire à l'initialisation
#define INIT64      0b10011
#define INIT128     0b00011

//Création du pointeur pour le périphérique GPIO
XGpio Gpio;

void PCM4202_init(char oversampling)
{
    //Initialisation du périphérique GPIO
    XGpio_Initialize(&Gpio, GPIO_ID);

    //Toutes les broches en mode "sortie"
    XGpio_SetDataDirection(&Gpio, 1, 0);

    //On écrit l'état d'initialisation en fonction
    //du facteur de sur-échantillonnage
    if (oversampling == 64) {XGpio_DiscreteWrite(&Gpio, 1, INIT64);}
    else {XGpio_DiscreteWrite(&Gpio, 1, INIT128);}
}
```

4.3.2 DAC

4.3.2.1 Électronique

Tout comme les signaux d'entrée de l'ADC, les signaux de sortie du DAC nécessitent un étage de sortie qui adapte l'impédance, applique un gain et filtre les signaux de sortie.
Le schéma de la figure 50 est proposé dans la fiche technique de la puce DSD1793 [*24 Bits 192 kHz Audio Stereo DAC* 2006, Figure 32, p.32]. On utilise encore l'amplificateur opérationnel OPA1632 car il est symétrique, ce qui correspond à la sortie du DAC.
Le circuit comprend aussi un filtre de type passe-bas avec une fréquence de coupure à 77 kHz.
Le DSD1793 présente un signal d'amplitude ± 3,2 V (environ 3,3 dBu) pour une entrée numérique à 0 dBFS. Le circuit donne un gain de 1,83 au signal. Pour une sortie normalisée à +4 dBu, on préférera un gain de 1,08. Ce gain peut être modifié en altérant les valeurs des résistances R104 et R108.

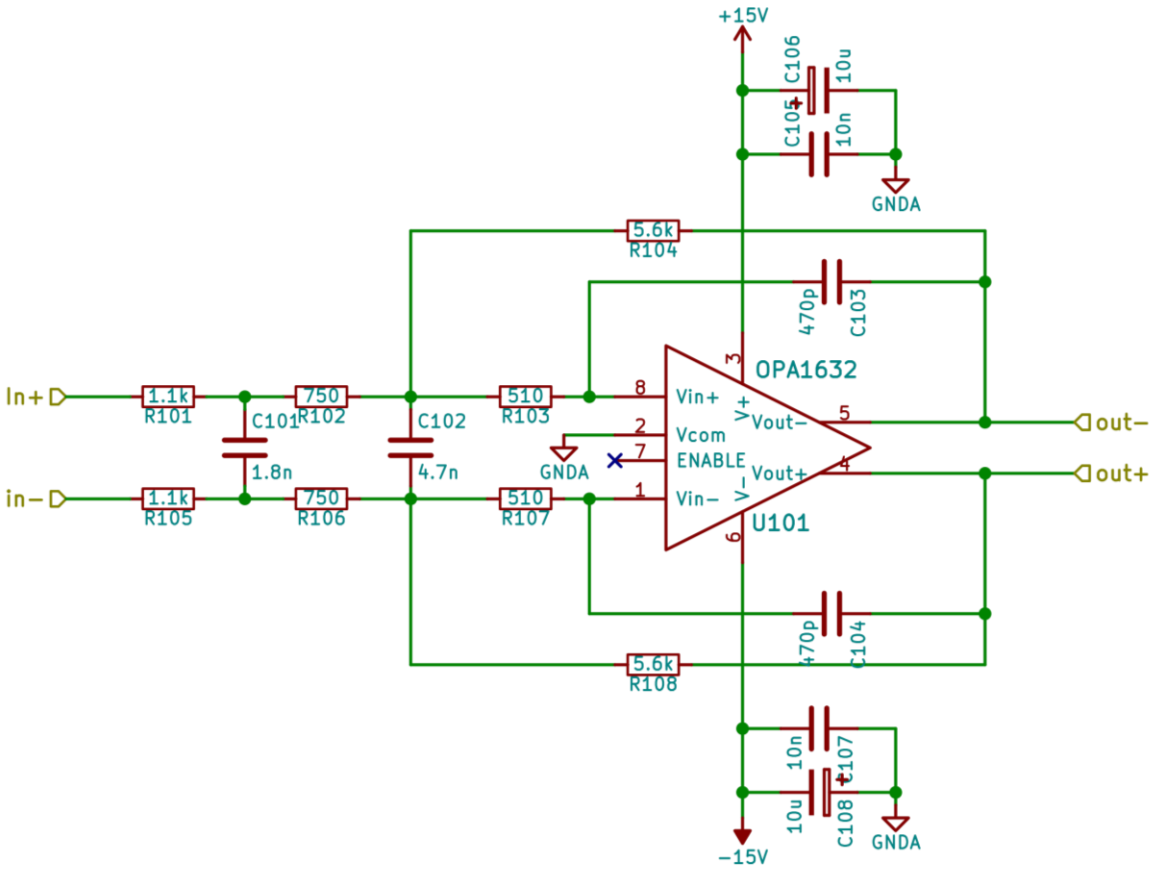


Figure 50 – Filtre de Sortie du DAC [Payen de La Garanderie 2015, Figure 41, p.52]

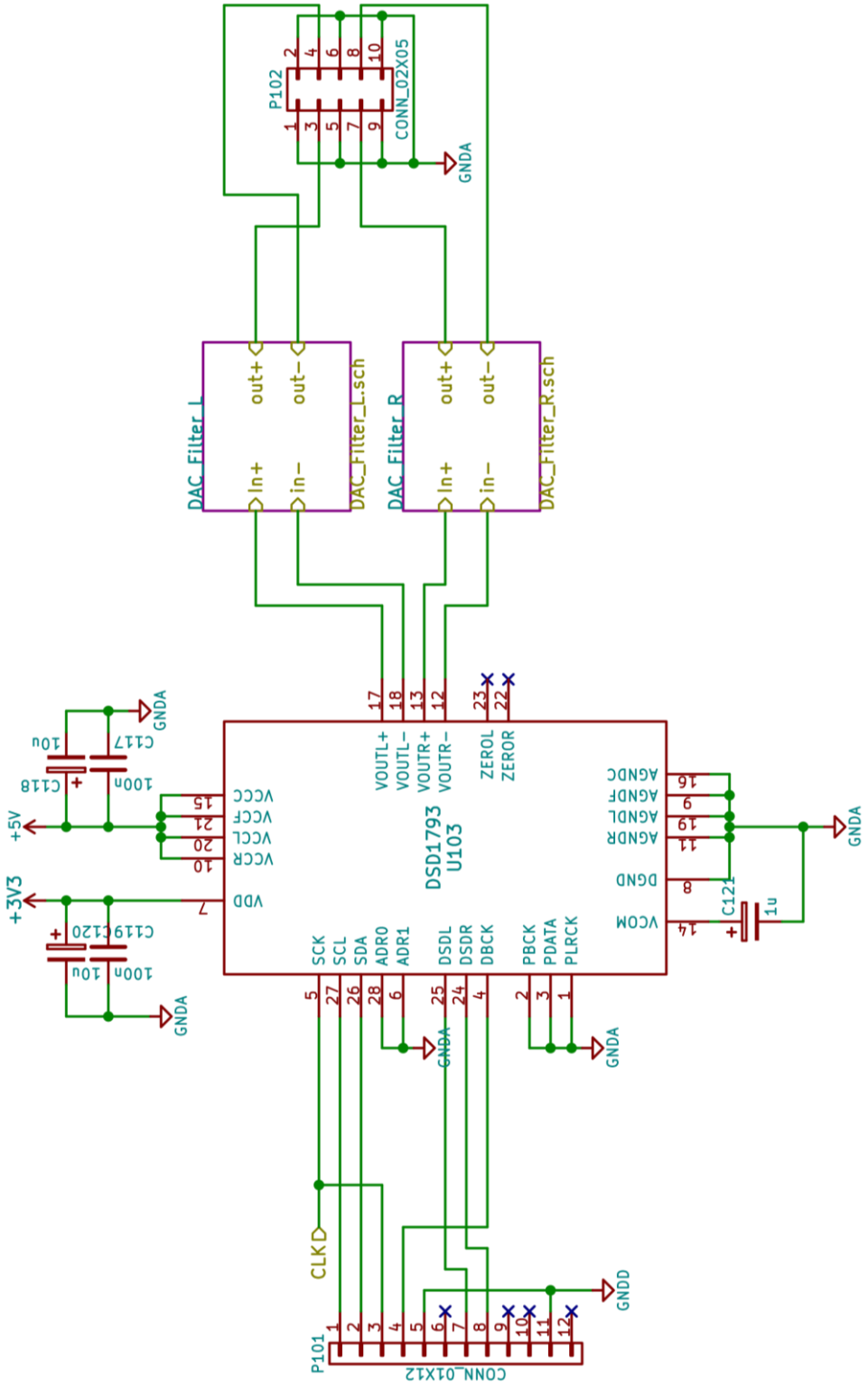


Figure 51 – Circuit de la Puce de DAC DSD1793 [Payen de La Garanderie 2015, Figure 42, p.53]

4.3.2.2 Configuration

Tout comme pour la puce de ADC, la puce de DAC DSD1793 peut fonctionner en modes PCM ou DSD. La configuration de notre DAC contient beaucoup d'options; c'est pour cela qu'elle se fait via un bus I^2C.
Il est possible de faire varier deux des sept bits de l'adresse I^2C du DAC en faisant varier l'état logique des broches "ADDR0" et "ADDR1". Cette option est utile si on connecte plusieurs puces de même modèle sur un même bus I^2C.

Afin de faire fonctionner le DAC en mode DSD, nous devons seulement changer l'état logique d'un des bits du registre numéro 20 du DSD1793. Pour faire cela, nous allons utiliser un autre bloc IP du FPGA appelé "*axi_iic*" qui permet au SoC de communiquer via un bus I^2C. Ce bloc permet de passer du bus AXI, contrôlé par le processeur ARM, vers un message I^2C compréhensible par le DAC.
Le contrôle du bloc "*axi_iic*" se fait dans des des registres via le bus AXI. C'est un autre *driver* qui s'occupe du contrôle du bloc I^2C; la configuration du DAC peut donc se faire en une seule ligne de code. Cette ligne envoie l'ordre à l'adresse I^2C du DAC de modifier le bit de fonctionnement DSD dans le registre numéro 20 où il se trouve.

4.4 Interface

4.4.1 Principe du Contrôle du FPGA par le Processeur

Contrôler les paramètres du compresseur est essentiel pour pouvoir l'utiliser convenablement. Comme évoqué précédemment, cette tâche repose sur le processeur du SoC. Comme le compresseur se trouve dans la partie de logique programmable du SoC, le moyen d'échanger des informations entre les parties PS et PL est de passer par le bus AXI prévu à cet effet.
Lors de la création du modèle *System Generator for DSP* du compresseur, on fait en sorte que tous les paramètres réglables soient introduits par une entrée compatible AXI-lite. Une fois que le bloc *System Generator* est intégré au circuit dans *Vivado*, on peut connecter les bus AXI du processeur Zynq aux bus AXI-lite de chaque paramètre du compresseur en passant par un bloc périphérique prévu à cet effet.

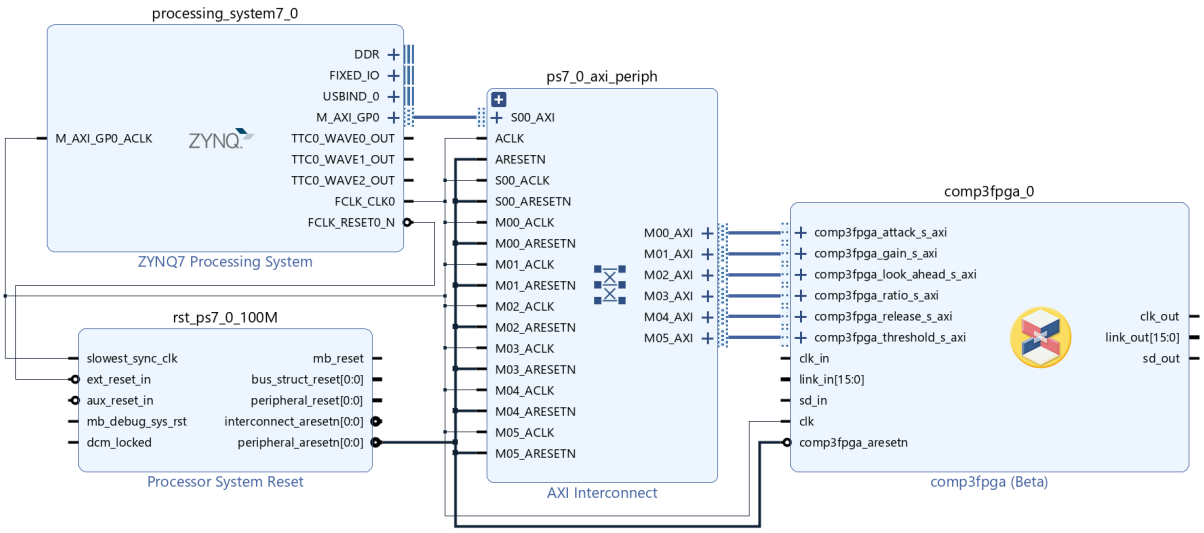


Figure 52 – Connexion des Paramètres du Compresseur au Bus AXI du Processeur

Une fois passé à *Vitis* pour programmer le processeur, on peut se servir du pilote que *Vivado* a créé spécifiquement pour s'adresser au bloc du compresseur. Ce pilote permet d'initialiser le compresseur ainsi que d'envoyer et de recevoir des informations pour chaque paramètre du compresseur

Voici le code en C dans *Vitis* utilisé pour modifier un paramètre du bloc de compresseur :

```c
//On inclut le driver du compresseur
#include "compresseur.h"
#include "xparameters.h"

//On récupère les identifiants des deux blocs de compresseur dans xparameters
#define COMPL_ID XPAR_COMPRESSEUR_0_DEVICE_ID
#define COMPR_ID XPAR_COMPRESSEUR_1_DEVICE_ID

//On donne un pointeur pour chaque instance du compresseur
compresseur CompL;
compresseur CompR;

int main(void)
{
    //On initialise les deux compresseurs
```

```
16      compresseur_Initialize(&CompL, COMPL_ID);
17      compresseur_Initialize(&CompR, COMPR_ID);
18
19      //On change la valeur du threshold pour chaque instance du compresseur
20      compresseur_threshold_write(&CompL, 50000);
21      compresseur_threshold_write(&CompR, 50000);
22  }
```

4.4.2 Principe du Contrôle du Processeur par des Potentiomètres

Le SoC de la Zedboard, est équipé de convertisseurs analogique-numérique 12 bits. L'une des fonctions principales de ces convertisseurs est de surveiller les températures du SoC et de déclencher des alarmes si celles-ci dépassent un seuil critique. Néanmoins, il est aussi possible de présenter un signal analogique extérieur en entrée de l'ADC et de le convertir.
C'est grâce à cette fonction de l'ADC que nous allons pouvoir contrôler des variable dans le processeur par des potentiomètres.

Les entrées de ADC peuvent être configurées en mode bipolaire pour convertir des tensions entre -0,5 et 0,5 V. On utilisera le mode unipolaire, avec une tension maximale convertie par l'ADC de 1 V [Engeler 2017]. En 12 bits, Cela donne un pas de discrétisation de : $\Delta = \frac{1}{2^{12}} \approx 0,24\ mV$
Chaque entrée est symétrique avec une fil positif (avec un nom finissant par P) et un fil négatif (avec un nom finissant par N). Nous connecterons systématiquement l'entrée négative à la masse pour pouvoir moduler seulement l'entrée positive.

L'ADC supporte l'utilisation d'un multiplexeur analogique externe dans le but de convertir plusieurs signaux analogiques via une seule broche (Figure 53).

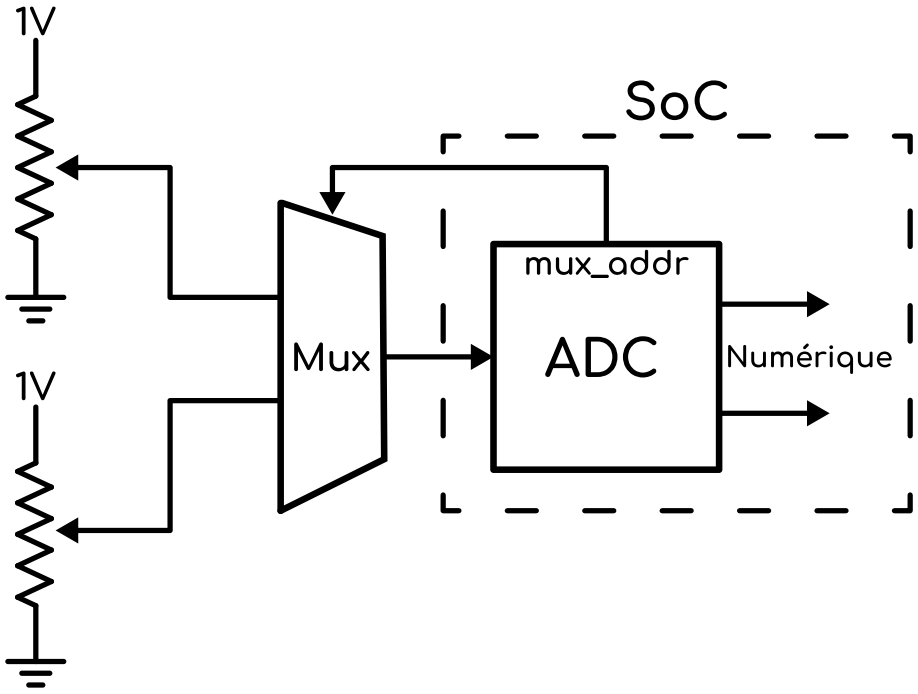


Figure 53 – Schéma Fonctionnel de l'Utilisation de l'ADC du SoC avec un Multiplexeur Analogique Externe pour 2 Canaux de Conversion

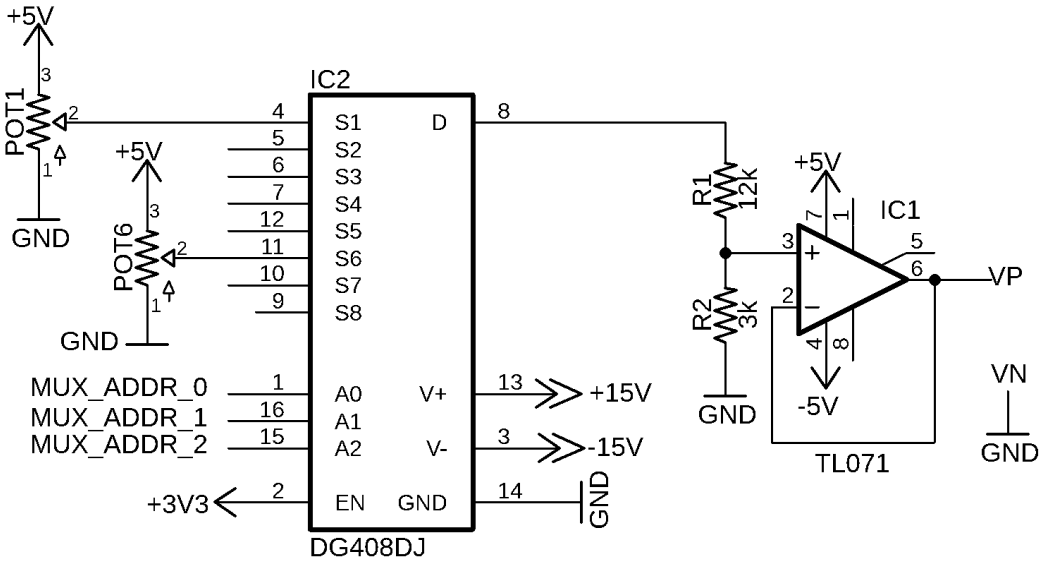


Figure 54 – Schéma Électronique de la Carte de Contrôle du Compresseur (Représenté avec 2 Potentiomètres sur 7)

Pour réaliser le circuit on utilisera la puce de multiplexeur analogique DG408 [*8-Ch CMOS Analog Multiplexer* 2013], fabriquée par Vishay. Elle permet de faire un multiplexage temporel entre huit sources différentes, il est donc possible de brancher jusqu'à huit potentiomètres au SoC. On alimentera le multiplexeur avec la même alimentation symétrique ±15 V utilisée dans le circuit des convertisseurs.

Chaque potentiomètre fournit une tension variable entre 0 et 5 V au multiplexeur. La tension de sortie du multiplexeur doit donc être ramenée à un maximum de 1 V. On réalise cela avec un diviseur de tension. On ajoute un circuit suiveur après le diviseur de tension pour faire une adaptation d'impédance. Sans cette adaptation d'impédance, les données sont mal lues et les valeurs de sortie des potentiomètres influent les unes sur les autres.

La Zedboard propose un connecteur spécifique pour les broches du SoC qui concernent l'ADC. On connectera alors notre carte de contrôle à ce connecteur pour recevoir les signaux de commutation du multiplexeur et envoyer le signal de sortie du multiplexeur au SoC.
La figure 55 nous montre la carte de contrôle du compresseur réalisée dans le cadre de ce mémoire, avec 7 potentiomètres, le multiplexeur, l'amplificateur opérationnel, le connecteur 8 broches d'alimentation et le connecteur 20 broches de la Zedboard.

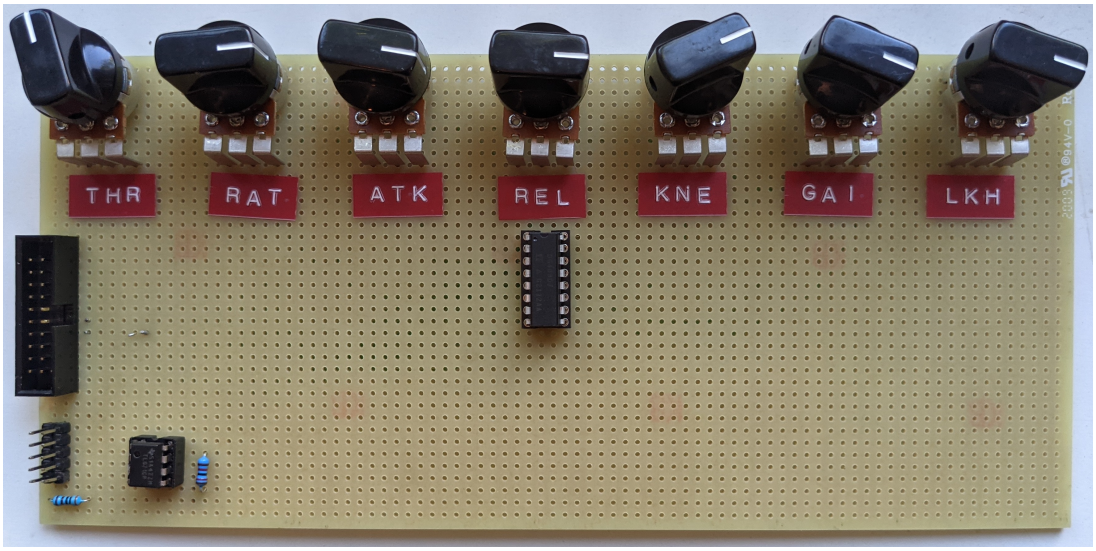


Figure 55 – La Carte de Contrôle du Compresseur Réalisée Lors de ce Mémoire

L'ADC peut-être facilement paramétré dans *Vivado* via un bloc appelé *"XADC_Wizard"*. Une fois l'ADC configuré dans *Vivado*, il suffit d'utiliser les fonctions du pilote *"xadcps"* dans *Vitis* pour récupérer les valeurs de conversion stockées dans les registres de l'ADC. Même si la précision de conversion est de 12 bits, ces valeurs sont codées sur 16 bits entre 0 et 65535.

Voici le code en C utilisé dans *Vitis* pour récupérer les valeurs utiles des conversions de l'ADC :

```c
//On inclue les librairies nécessaires
#include "xparameters.h"
#include "xadcps.h"
#include "math.h"

//On trouve l'identification de l'ADC dans xparameters
#define ADC_ID XPAR_PS7_XADC_0_DEVICE_ID
//On indique un pointeur pour l'ADC
XAdcPs Adc;

//On entre dans la fonction principale
int main(void)
{
    //On déclare les variables nécessaires
    unsigned int threshold, rawratio;
    float ratio;

        //On initialise l'ADC
        XAdcPs_Config * cfg = XAdcPs_LookupConfig (ADC_ID);
        XAdcPs_CfgInitialize(&Adc,cfg,cfg->BaseAddress);

        //On entre dans une boucle infinie
        while(1)
        {
            //On récupère les valeurs des conversions de chaque potentiomètre
            //dans leur canal respectif
            threshold = XAdcPs_GetAdcData(&Adc, XADCPS_CH_AUX_MIN);
            rawratio = XAdcPs_GetAdcData(&Adc, XADCPS_CH_AUX_MIN + 2);

        //On adapte la valeur du ratio entre 0 et 1
            ratio = ((float) rawratio)/65535.f;
            //Les valeurs sont maintenant prêtes à être envoyées au compresseur
        }
}
```

4.4.3 Indicateur de Réduction de Gain

Pour correctement utiliser un compresseur, visualiser la réduction de gain en temps réel peut être très important. Par conséquent, on veut ajouter une bande de LED qui s'allument en fonction du signal de réduction du compresseur.

La Zedboard comprend une bande de huit LED déjà reliées à différentes broches du SoC. Nous allons donc créer un bloc logique dans *System Generator for DSP* qui prend en entrée le signal de réduction de gain codé sur 16 bits et dont les sorties contrôlent l'état des huit LED.
Comme on a un nombre de LED limité, on peut ajouter un paramètre d'échelle qui adapte le nombre de décibels de réduction que représente chaque LED. On va utiliser quatre échelles différentes, avec une plage dynamique de réduction de : 4, 8, 16 et 32 dB. Ainsi, la plupart des utilisations d'un compresseur seront couvertes.

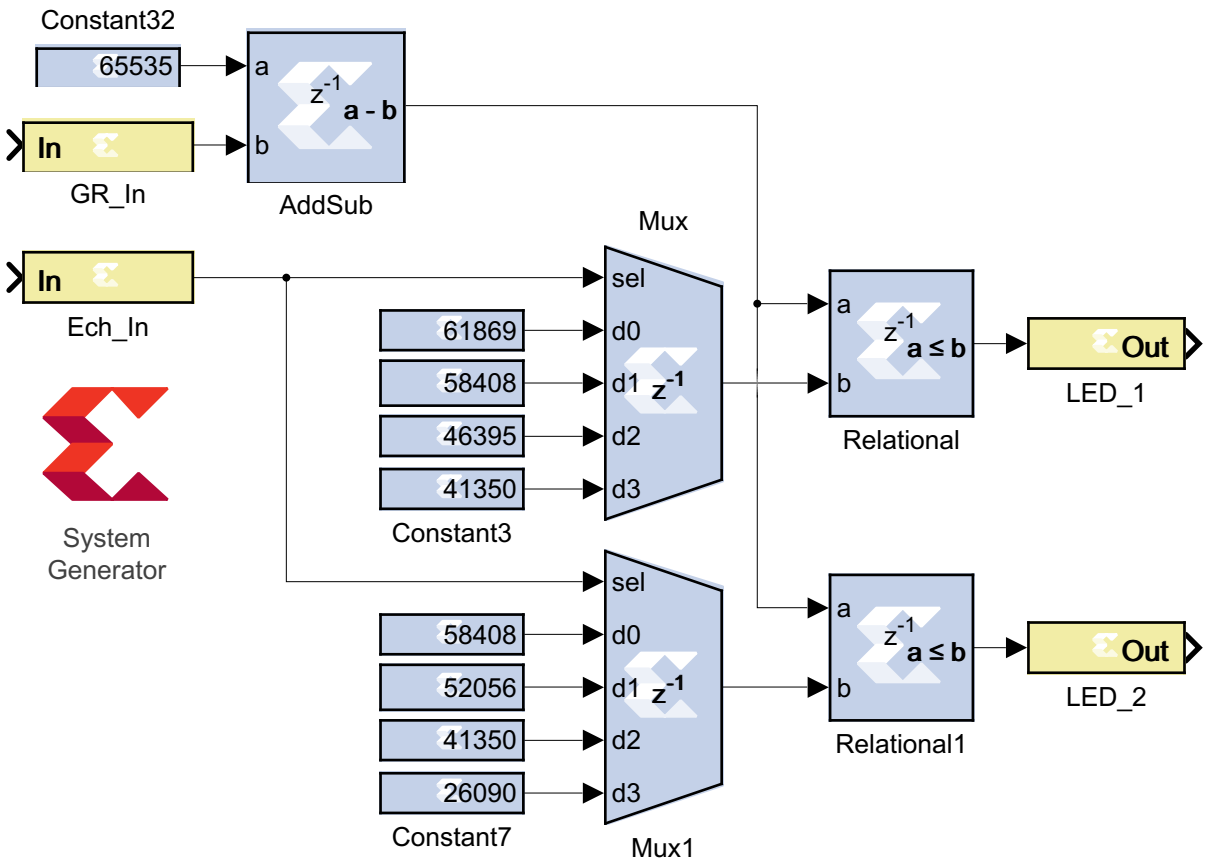


Figure 56 – Modèle Réduit *System Generator for DSP* du Circuit de l'Indicateur de Réduction de Gain

Comme le montre la figure 56, pour indiquer le niveau de réduction, il

suffit de comparer le signal de réduction avec des paliers progressifs pour chaque LED. Cette comparaison est faite avec un bloc *relational* par LED. Quand le signal de réduction de gain est inférieur au palier, le bloc *relational* présente un état logique haut en sortie, ce qui allume la LED.
Pour choisir entre les différentes échelles de dynamique, on fait varier les constantes de comparaison. On commute entre les blocs de constantes grâce a des multiplexeurs. La commande de ces multiplexeurs est gérée par l'entrée d'échelle.
Les seuils ont étés calculés avec cette formule simplifiée :

$$Seuil_R = 10^{\frac{20\log(2^{16}-1)-R}{20}}$$

Avec R la réduction à indiquer en décibels.

Pour connecter le bloc logique qui contrôle les LED, on commence par ajouter une sortie au bloc de compresseur pour pouvoir récupérer le signal de réduction du gain sur 16 bits (avant l'application du *make-up gain*). L'échelle, quant à elle, pourra être contrôlée par le processeur via un bus AXI-lite, comme les autres paramètres du compresseur. Pour indiquer au processeur qu'on désire changer d'échelle, on utilisera les boutons-poussoirs présents sur la Zedboard. Deux entrées sont donc ajoutées sur le second canal du module GPIO.

Voici le code en C utilisé dans *Vitis* pour contrôler le bloc d'indicateur de réduction de gain :

```
1  //On importe les pilotes necessaires
2  #include "led_gr.h"
3  #include "xparameters.h"
4  #include "xgpio.h"
5  //On récupère les identifiants du GPIO et de l'indicateur
6  #define LEDGR_ID XPAR_LED_GR_0_DEVICE_ID
7  #define GPIO_ID        XPAR_GPIO_0_DEVICE_ID
8  //On donne un pointeur pour le GPIO et l'indicateur
9  XGpio Gpio;
10 led_gr Ledgr;
11
12 //On entre dans la fonction principale
13 int main (void)
```

```
{
    //On declare une variable pour l'echelle
    char ech = 0;
    //On déclare une variable pour vérifier si un des boutons est appuyé
    char pushed;

    //On initialise les deux blocs
    led_gr_Initialize(&Ledgr, LEDGR_ID);
    XGpio_Initialize(&Gpio, GPIO_ID);

    //On écrit 0 dans le sélecteur d'échelle
    led_gr_ech_in_write(&Ledgr, ech);

    //On entre dans une boucle infinie
    while(1)
    {
        //Si le bouton de réduction vient d'être appuyé
        if (XGpio_DiscreteRead(&Gpio, 2)==1 && pushed == 0) {
            //On réduit l'échelle si elle n'est pas au minimum
                        if (ech != 0) {
                        ech--;
                        led_gr_ech_in_write(&Ledgr, ech);
                        pushed = 1;}
                }

                //Si le bouton d'augmentation vient d'être appuyé
            if (XGpio_DiscreteRead(&Gpio, 2)==2 && pushed == 0) {
                    //On augmente l'échelle si elle n'est pas déjà au maximum
                        if (ech != 3) {
                                ech++;
                                led_gr_ech_in_write(&Ledgr, ech);
                                pushed = 1;}
                }

                //Quand le bouton est relâché
                //un nouveau changement d'échelle est possible
                if (XGpio_DiscreteRead(&Gpio, 2) == 0) {pushed = 0;}
    }}
```

4.5 Test du Modèle

4.5.1 Dans *Vivado*

Il est possible d'observer, depuis *Vivado*, les différents signaux disponibles dans le FPGA. On utilise pour cela un bloc appelé ILA (Integrated Logic Analyzer). À l'image d'un oscilloscope, ce bloc dispose de plusieurs sondes à connecter au circuit. Pendant le fonctionnement, il suffit d'engager la capture depuis *Vivado* pour afficher un nombre d'échantillons prélevés à chaque sonde.

Pour pouvoir observer les signaux internes au compresseur pendant son fonctionnement, il suffit donc de les rendre disponibles en ajoutant une sortie au modèle *System Generator for DSP* et d'y connecter une sonde d'un ILA dans *Vivado*.

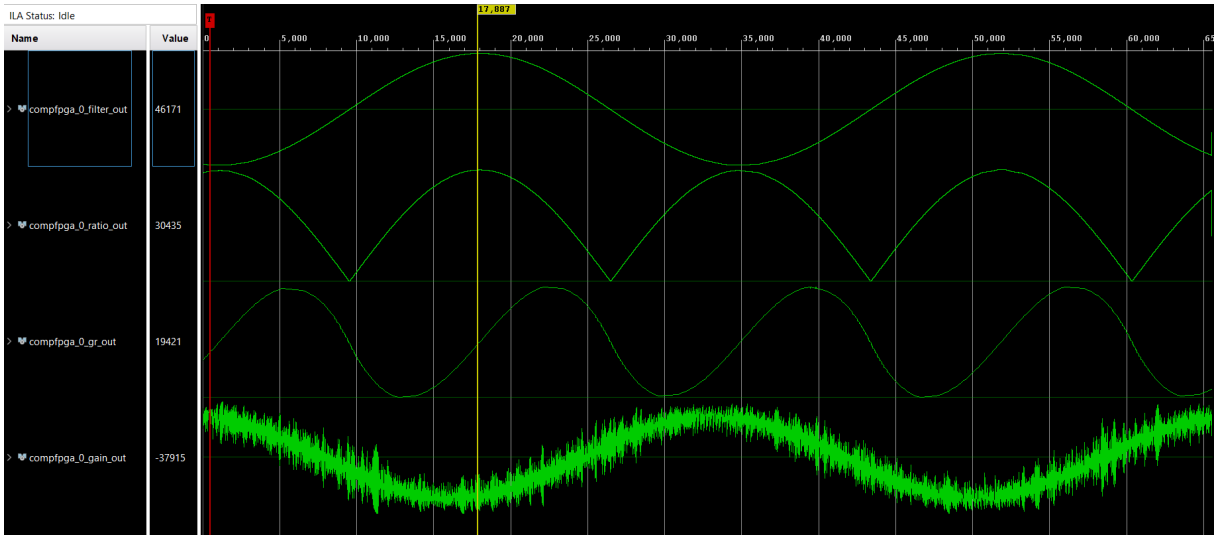


Figure 57 – Observation de Plusieurs Étapes du Traitement du Signal dans le Compresseur avec un ILA dans *Vivado*

La figure 57 nous montre différentes étapes du traitement du signal dans le compresseur. De haut en bas on retrouve la sortie du filtre de détection (Figure 24 C), le signal de réduction de gain après la multiplication par le ratio (Figure 26 B), le signal de réduction de gain après le filtre de réponse (Figure 26 C) et la sortie du filtre qui précède le modulateur Σ/Δ (Figure 34 C).

4.5.2 Mesure Audio

Maintenant qu'on s'est assuré que le circuit fonctionne bien en interne, on peut utiliser du matériel de test audio pour vérifier les caractéristiques du compresseur. On commence par envoyer un échelon de niveau d'un signal sinusoïdal de 10 kHz. Le niveau du signal commence à -10 dBFS, puis passe à 0 dBFS et enfin reviens à -10 dBFS. Le seuil et ratio du compresseur sont réglés de façon à compresser le signal de 4 dB quand le signal est à son amplitude maximale. Le temps d'attaque et de retour est réglé à 1 ms.

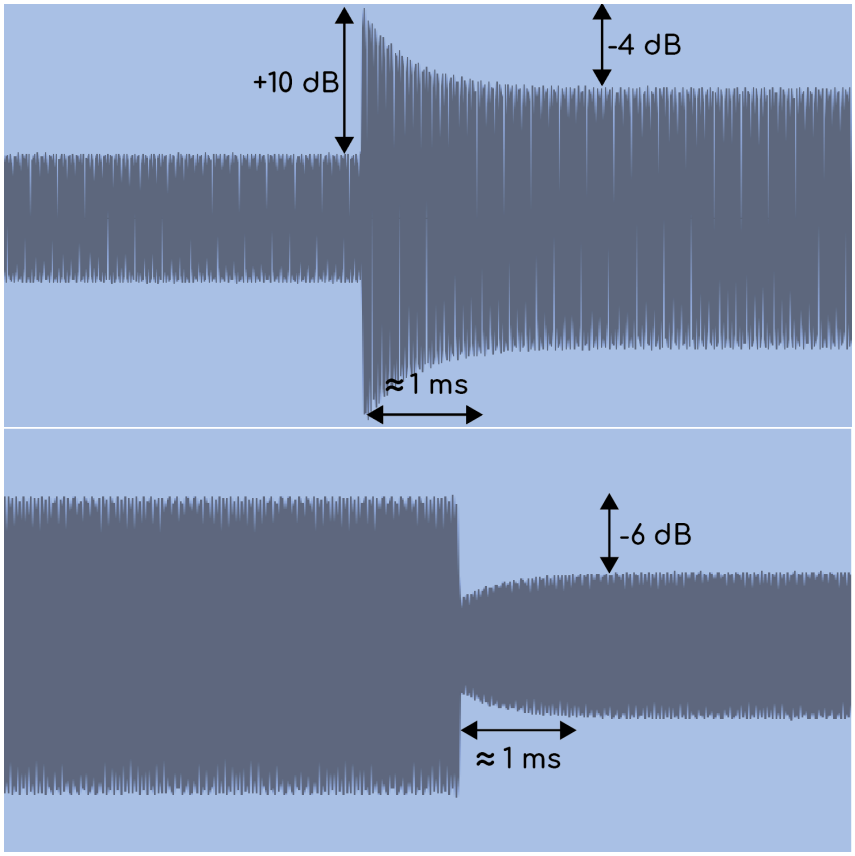


Figure 58 – Réponse en Dynamique du Compresseur avec un Échelon de Niveau en Entrée

La figure 58 nous montre bien que l'effet du compresseur Σ/Δ est typique d'un compresseur de dynamique. On retrouve le même genre de signal que la figure 19. La compression indiquée par les LED est la même que celle réellement appliquée à l'audio.

La prochaine mesure est de comparer le niveau de sortie par rapport au niveau d'entrée. Cela met en évidence la réponse en dynamique du compresseur. Le compresseur est réglé avec un ratio de 4 : 1.

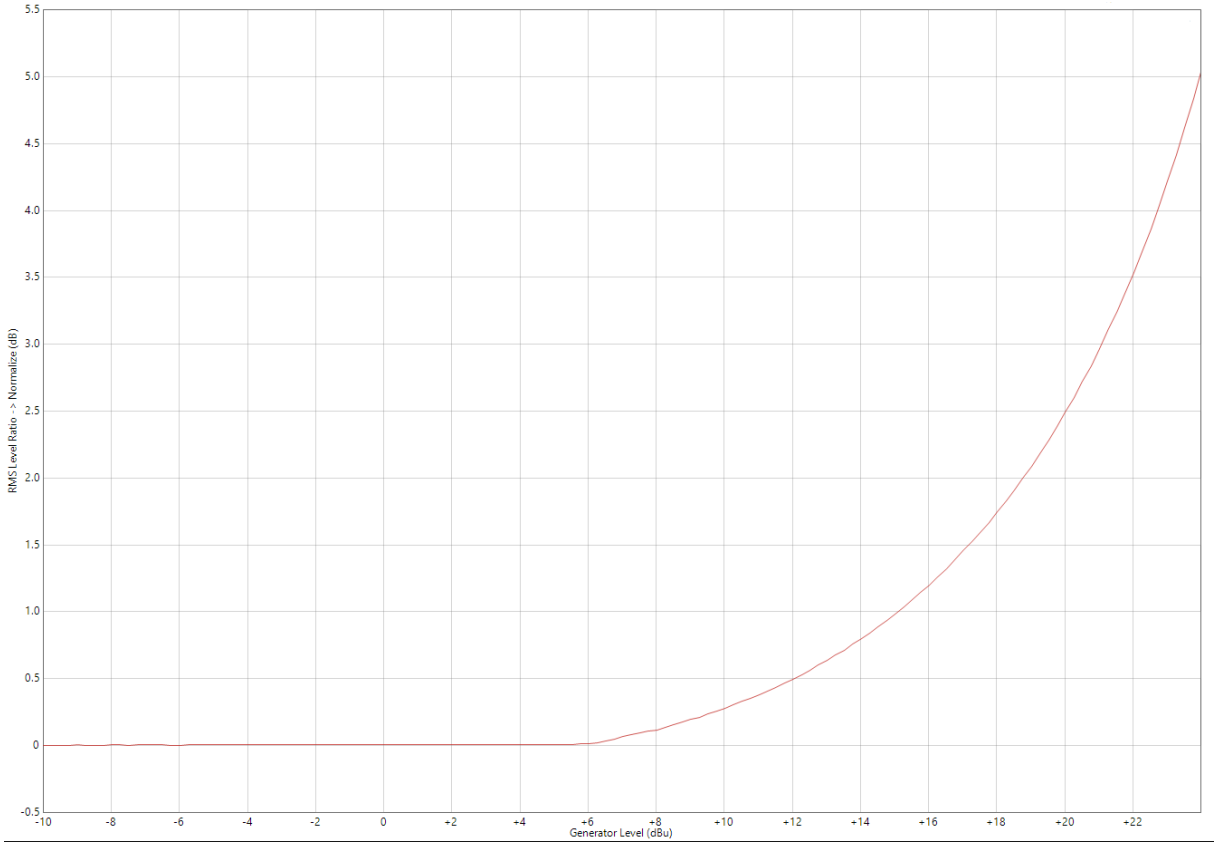


Figure 59 – Mesure de la Réduction de Gain pour un Ratio de 4 : 1

La figure 59 révèle un problème avec le modèle de compresseur réalisé dans ce mémoire, le ratio appliqué au signal compressé n'est pas constant. En effet, avec le *knee* désactivé, on devrait retrouver une ligne droite de réduction de niveau et non pas une courbe.
Cette caractéristique ne provient pas du filtre de réponse. Une piste est donc de vérifier comment la courbe de gain de réduction est appliquée au signal d'entrée. Néanmoins, ce n'est pas une barrière au bon fonctionnement du compresseur, juste un problème de précision dans l'application du gain.

5 Conclusion

Dans ce mémoire, nous avons résumé les bases théoriques et techniques nécessaires au traitement de dynamique de la modulation Σ/Δ en temps réel. Après avoir donné le contexte d'utilisation de la modulation Σ/Δ, on a justifié le choix du FPGA pour son traitement numérique.
Par la suite, on a rappelé le rapport complexe entre dynamique audionumérique et la modulation Σ/Δ. On propose alors un modèle de compresseur de dynamique en modulation Σ/Δ. Ce modèle prend pour base un compresseur audionumérique courant, mais pour chaque bloc de fonction on explique les différences liées au traitement numérique de la modulation Σ/Δ.
Dans la réalisation pratique, on à utilisé des outils logiciels et matériels modernes de développement sur SoC et présenté leurs utilisations. On finit par obtenir un compresseur de dynamique en Σ/Δ fonctionnel avec un panel de contrôles ajustables en temps réel. Les fonctionnalités ne sont pas réduites par le fait que le compresseur soit conçu pour la modulation Σ/Δ.
L'objectif de ce mémoire est de vérifier si un compresseur de dynamique peut être utilisé en modulation Σ/Δ avec une intégrité du signal préservée. Les mesures réalisées sur le modèle de compresseur semblent indiquer qu'une utilisation exigeante en intégrité du signal, comme dans la production musicale, est possible, même si certains défauts de conception restent à corriger.

Le DSD est aujourd'hui surtout consommé par un public audiophile. L'idée d'une chaîne du son seulement en format 1 bit semble difficile à atteindre. Néanmoins, de nombreux types de traitements numériques du signal en modulation Σ/Δ restent à développer et la modulation Σ/Δ reste utile dans certaines utilisations précises. On peut penser par exemple, au traitement des microphones numériques quand une latence minimale doit être maintenue.
Comme décrit précédemment, le modèle de compresseur réalisé dans la partie pratique de ce mémoire présente une qualité audio légèrement dégradée. Ceci est lié au besoin de raffiner le modèle du compresseur, mais aussi aux outils utilisés. En effet, *System Generator for DSP* est un outil facile d'accès, mais les langages HLS ou HDL restent des outils de conception plus puissants utilisés dans un contexte professionnel. Même si la conception du compresseur dans ce mémoire est surtout une preuve de

concept, une conception plus robuste aurait permis de proposer des tests perceptifs sur l'utilisation du compresseur.

D'autres types de traitement numériques essentiels restent à développer pour la modulation Σ/Δ, comme la réverbération ou des filtres communément utilisés dans les égaliseurs. Peut-être que des avantages propres aux traitements en modulation Σ/Δ encourageront à l'utilisation plus courante de ce type de traitement. La chaîne du son utilisée dans le futur de l'ingénierie audio reste tout de même à imaginer et la modulation Σ/Δ est seulement une des propositions avancées.

Sigma-Delta

Eastty, P. C., C. Sleight et P. D. Thorpe (mar. 1997). « Research on Cascadable Filtering, Equalization, Gain Control, and Mixing of 1-Bit Signals for Professional Audio Applications ». In : *Audio Engineering Society Convention 102*. url : http://www.aes.org/e-lib/browse.cfm?elib=7335.

Inose, H., Y. Yasuda et J. Murakami (1962). « A Telemetering System by Code Modulation - Δ/Σ Modulation ». In : *IRE Transactions on Space Electronics and Telemetry*.

Reefman, D. et E. Janssen (2004). « One-Bit Audio : An Overview ». In : *J. Audio Eng. Soc* 52.3, p. 166-189. url : http://www.aes.org/e-lib/browse.cfm?elib=12989.

Schreier, R. et G. C. Temes (2005). *Understanding Delta-Sigma Data Converters*. Sous la dir. de Wiley-Interscience.

Widrow, B. (1961). « Statistical Analysis of Amplitude-Quantized Sampled-Data Systems ». In : *Transactions of the American Institute of Electrical Engineers, Part II : Applications and Industry* 79.6, p. 555-568. doi : 10.1109/TAI.1961.6371702.

FPGA

Amano, H., éd. (2018). *Principles and Structures of FPGAs*. Springer.

Crockett, L. H. et al. (2014). *The Zynq Book*. url : http://www.zynqbook.com.

Pang, A. et P. Membrey (2017). *Beginning FPGA : Programming Metal, Your Brain on Hardware*. Apress.

Woods, R. et al. (2017). *FPGA-based Implementation of Signal Processing Systems*. Wiley.

Traitement du Signal

Giannoulis, D., M. Massberg et J. D. Reiss (2012). « Digital Dynamic Range Compressor Design—A Tutorial and Analysis ». In : *J. Audio Eng. Soc* 60.6, p. 399-408. url : http://www.aes.org/e-lib/browse.cfm?elib=16354.

Laroche, J. (1995). *Traitement des Signaux Audio-Fréquences*. TELECOM Paris.

Montpied, E. (2019). « Conception et Réalisation d'un Microphone à Directivité Contrôlable et Orientable ». Mém. de mast. École Nationale Supérieure Louis-Lumière. url : https://www.ens-louis-lumiere.fr/sites/default/files/2021-04/ENSLL_2019_Son_Montpied_M%C3%A9moire.pdf.

Payen de La Garanderie, P. (2015). « Traitement du Signal Audionumérique à Partir des Modulations Sigma-Delta et PCM ». Mém. de mast. École Nationale Supérieure Louis-Lumière. url : https://www.ens-louis-lumiere.fr/sites/default/files/2019-02/ENSLL_2015_Son_Payen_BD.pdf.

Zölzer, U. (2008). *Digital Audio Signal Processing*. Wiley. doi : 10.1002/9780470680018.

DSD

Janssen, E., E. Knapen et al. (2004). « Lossless Compression of One-Bit Audio ». In : *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*. T. 3, p. iii-1020. doi : 10.1109/ICASSP.2004.1326721.

Janssen, E. et D. Reefman (2003). « Super-audio CD : an Introduction ». In : *IEEE Signal Processing Magazine* 20.4, p. 83-90. doi : 10.1109/MSP.2003.1226728.

Merging Technologies (p. d.[a]). *DXD Compatible Plugins*. url : https://confluence.merging.com/display/PUBLICDOC/DXD+Compatible+Plugins.

Merging Technologies (p. d.[b]). *High Resolution Audio*. url : https://www.merging.com/highlights/high-resolution.

Thorpe, P. et al. (mai 2001). « DSD-Wide. A Practical Implementation for Professional Audio. » In : *Audio Engineering Society Convention 110*. url : http://www.aes.org/e-lib/browse.cfm?elib=9998.

Vest, M. (2004). « The Advantages of DXD for SACD ». In : *Resolution*.

Documents Techniques

24 Bits 192 kHz Audio Stereo DAC (nov. 2006). SLES075B. Rev.B. Texas Instruments. url : https://www.ti.com/lit/ds/symlink/dsd1793.pdf.

24 Bits, 216 kHz Stereo Audio ADC (sept. 2004). SBAS290B. Texas Instruments. url : https://www.ti.com/lit/ds/symlink/pcm4202.pdf.

7 Series DSP48E1 User Guide (mar. 2018). UG479. v1.10. Xilinx. url : https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

8-Ch CMOS Analog Multiplexer (déc. 2013). 70062. Vishay. url : https://www.vishay.com/docs/70062/dg408.pdf.

AMBA AXI and ACE Protocol Specification (2013). IHI0022E. ARM. url : https://developer.arm.com/documentation/ihi0022/e?_ga=2.67820049.1631882347.1556009271-151447318.1544783517.

Avnet (p. d.). *ZedBoard*. url : https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/.

Engeler, P. (2017). *Using the Zynq-7000 XADC and Signal Pre-Conditioning*. Rapp. tech. ETH Zürich - Trapped Ion Quantum Information Group. url : https://ethz.ch/content/dam/ethz/special-interest/phys/quantum-electronics/tiqi-dam/documents/semester_theses/vacationthesis-Pascal_Engeler.

I^2C - Bus Specification and User Manual (avr. 2014). UM10204. Rev.6. NXP. url : https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

Ogier, E. (2021). *Delta-Sigma Modulator*. MATLAB Central File Exchange. url : https://www.mathworks.com/matlabcentral/fileexchange/56166-delta-sigma-modulator.

Appendices

A Fonctions de *System Generator for DSP*

Voici un résumé des blocs de fonctions de *System Generator for DSP* qui sont utilisés dans ce mémoire.

A.1 System Generator

Figure 60 – Bloc *System Generator* de *System Generator for DSP*

Ce bloc est essentiel à tout montage sur *System Generator*. Il permet d'invoquer une instance *System Generator for DSP* dans *Simulink*. On peut y régler la fréquence d'horloge du SoC et la fréquence de simulation au sein de *Simulink*. De plus, c'est d'ici qu'on peut exporter notre système vers *Vivado* en choisissant notre type de SoC. Les blocs d'IP Xilinx ne peuvent pas fonctionner sans le bloc *System Generator*.

A.2 Constant

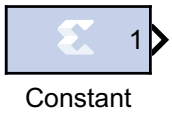


Figure 61 – Bloc de Constante dans *System Generator for DSP*

Ce bloc très simple permet de donner une valeur constante aux autres blocs. Pour une opération mathématique ou logique (une comparaison, par exemple).
Dans *System Generator for DSP*, il faut être vigilant au format des informations qui sont transmises entre les blocs. Ce format peut être modifié dans

beaucoup de blocs. Ainsi, on peut choisir un format à virgule flottante ou fixe ou un format booléen, ainsi que le nombre de bits qui définissent ces formats.

A.3 Gateway In/Out

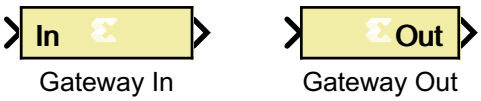


Figure 62 – Blocs d'entrée et sortie dans *System Generator for DSP*

Les blocs *Gateway* permettent de créer des ports d'entrées et de sorties de notre système. C'est avec ces mêmes ports que nous allons connecter le système dans *Vivado*.
Ces blocs permettent aussi d'interagir avec les blocs *Simulink* (comme les oscilloscopes ou les générateurs de fréquence). De cette manière, on peut diagnostiquer plus facilement le système dans *Simulink*.
Une option permet de rendre les entrées compatibles avec un bus AXI-lite pour recevoir des informations d'un processeur. C'est de cette manière qu'on peut paramétrer le compresseur depuis le processeur.

A.4 Delay

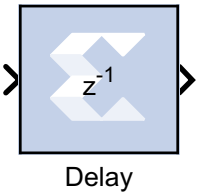


Figure 63 – Bloc Délai dans *System Generator for DSP*

Le bloc *delay* permet d'introduire un retard correspondant à un certain nombre d'échantillons. Dans le cas de la figure 63, c'est un délai d'un échantillon. La plupart des blocs comprennent un délai réglable (indiqué par le z^{-n} au centre du bloc).

Un bloc de délai réglable est utile lorsque l'on doit synchroniser des informations. Typiquement, une horloge au signal qui lui est associée.

A.5 Mux

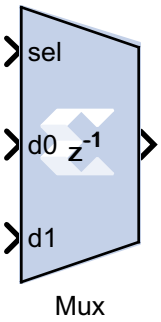


Figure 64 – Bloc Multiplexeur dans *System Generator for DSP*

Le multiplexeur délivre en sortie la valeur d'une de ses entrées [d0 : dn], en fonction de la valeur de son entrée de sélection "sel". Dans le cas de la Figure 64, l'entrée de sélection reçoit un seul état logique, car il n'y a que deux possibilités de commutation. On trouve donc la valeur de l'entrée "d0" en sortie si "sel" est à 0 et "d1" lorsque "sel" est égal à 1.
Cette fonction est très utile pour choisir entre deux valeurs ou deux chemins du signal. Dans ce mémoire, il est utilisé par exemple pour donner une amplitude différente au signal Σ/Δ.

A.6 Register

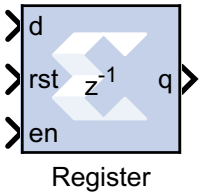


Figure 65 – Bloc de Bascule dans *System Generator for DSP*

Le bloc *Register* est équivalent à un circuit de bascule (*flip-flop*). Le nom *Register* vient du fait qu'un circuit de bascule peut être vu comme une mémoire de 1 bit. En effet, à chaque crête montante du signal d'horloge (équi-

valent à l'entrée "en" pour *enable*), l'état logique de l'entrée "d" est reporté dans la sortie principale notée "q". L'état logique de "q" est maintenu jusqu'à la prochaine crête montante de "en" ; à ce moment-là, l'état logique de "q" est réévalué en fonction de l'état logique de "d".
Quand l'entrée "rst" (*reset*) est d'état logique 1, la sortie "q" est forcée à l'état logique 0. Ceci peut être fait indépendamment des signaux "en" et "d", donc de manière asynchrone.

Certains circuits de bascule possèdent aussi une sortie complémentée notée "$\bar{q}$". Cette sortie est, à tout moment, l'exact complément de la sortie "q". Il existe d'autres types de bascules avec des caractéristiques différentes, mais nous ne les expliciterons pas dans ce mémoire.

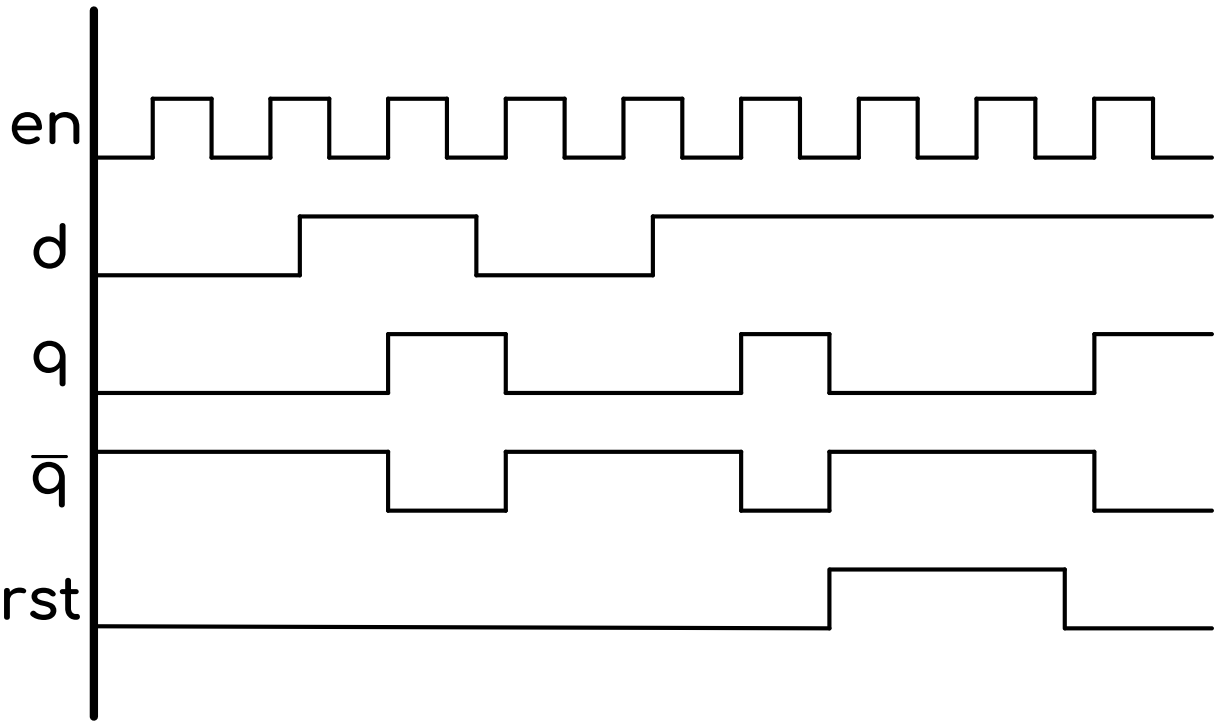


Figure 66 – Fonctionnement Logique d'un Circuit de Bascule D avec Reset Asynchrone

A.7 Addressable Shift Register

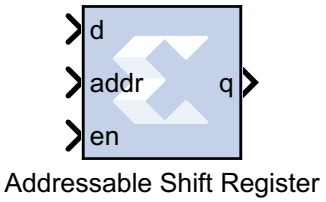


Figure 67 – Bloc de Registre Adressable dans *System Generator for DSP*

Le registre à décalage est similaire à une série de 1024 blocs de bascules, avec une mémoire allant jusqu'à 1024 bits. On peut l'imaginer comme une ligne de 1024 délais en série. L'adresse sert à faire varier la durée du délai.
On se sert de ce bloc pour introduire un délai variable dans un signal.

A.8 AddSub

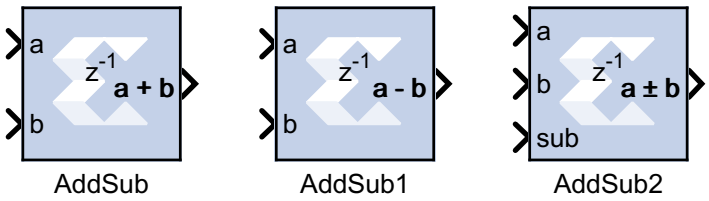


Figure 68 – Blocs Addition-Soustraction dans *System Generator for DSP*

Dans *System Generator for DSP*, le bloc AddSub est chargé d'effectuer des opérations d'addition ou soustractions. Il peut être mis dans un mode où le bloc réalise une soustraction quand son entrée "sub" est à l'état 1. Dans le cas contraire, il réalise une addition.

A.9 Accumulator

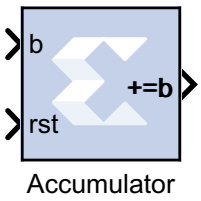


Figure 69 – Bloc Accumulateur dans *System Generator for DSP*

La fonction d'accumulateur effectue la somme de la valeur présente à son entrée avec la valeur précédente qu'il contient. On peut activer une entrée "*reset*" qui remet à zéro le contenu de l'accumulateur. Dans le Σ/Δ-M cela permet de calculer l'erreur de quantification accumulée.

A.10 Relational

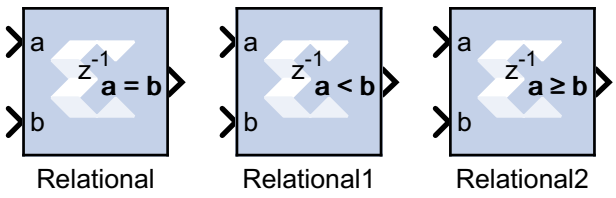


Figure 70 – Blocs Relation dans *System Generator for DSP*

La fonction *relational* compare ses deux entrées et émet un 1 à sa sortie si le résultat de la comparaison est vrai et un 0 dans le cas contraire. Les fonctions disponibles sont : $=, \neq, <, >, \leq$ et $\geq$.
Dans notre cas on l'utilise pour vérifier si le niveau est au-dessus du seuil de compression, par exemple.

A.11 Mult

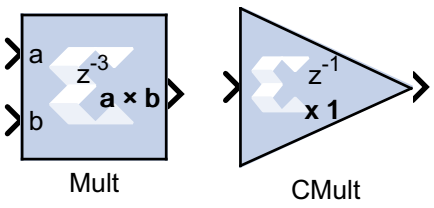


Figure 71 – Les Blocs de Multiplication dans *System Generator for DSP*

Le bloc *Mult* permet de multiplier ses deux entrées. C'est avec ce bloc que l'on applique le ratio du compresseur.
CMult est un bloc équivalent, mais qui permet de multiplier une entrée par une constante au format de notre choix.

A.12 Divide

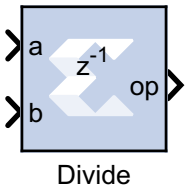


Figure 72 – Bloc de Division dans *System Generator for DSP*

Ce bloc permet de réaliser facilement une division de deux valeurs.

A.13 Negate

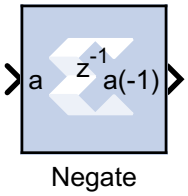


Figure 73 – Bloc Négation dans *System Generator for DSP*

Le bloc *Negate* multiplie l'entrée par -1. Si le chiffre d'entrée n'est pas signé, cela n'a pas d'importance et un chiffre signé sortira du bloc.

A.14 Absolute

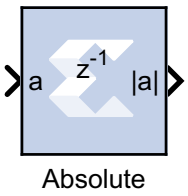


Figure 74 – Bloc Valeur Absolue dans *System Generator for DSP*

Le bloc *absolute* permet de récupérer la valeur absolue du chiffre d'entrée. C'est grâce à lui que l'on redresse le signal d'entrée pour la détection du niveau.

A.15 Convert

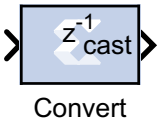


Figure 75 – Bloc de Conversion dans *System Generator for DSP*

Le bloc *convert* permet de convertir les données dans différents formats : Virgule fixe ou flottante, signé ou non. On trouve aussi des options d'arrondis quand on passe d'un format à virgule à un format sans virgule. De plus, il y a des options de saturation quand le maximum du format de sortie est atteint.

A.16 Digital FIR Filter & FDATool

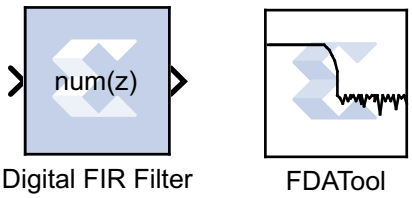


Figure 76 – Blocs de Filtre Numérique FIR et FDATool dans *System Generator for DSP*

Le bloc *Digital FIR Filter* permet de passer facilement le signal dans un filtre FIR. Son réglage principal est un champ pour entrer les coefficients du filtre.
Ainsi, une des façons simples de configurer le filtre est d'utiliser la fonction Matlab *fir1()*. Elle retourne les coefficients d'une réponse impulsionnelle FIR. L'autre façon est de le coupler avec un bloc *FDATool*. *FDATool* propose une interface de création de filtres. Cette interface permet de paramétrer les caractéristiques du filtre à réaliser.

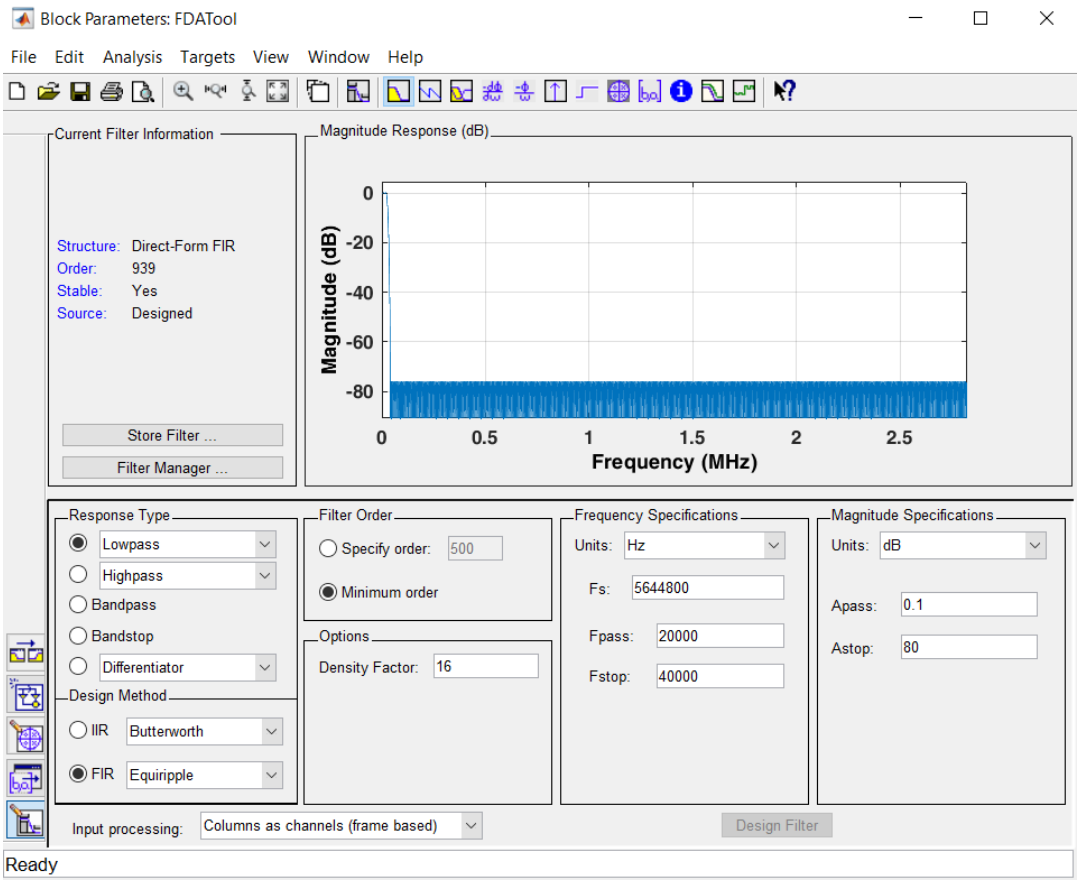


Figure 77 – L'Interface *FDATool* dans *System Generator for DSP*

B Calcul en Binaire

Dans un système numérique, toutes les informations doivent être représentées en base 2. Dans le cas des chiffres, deux familles de représentations sont majoritairement utilisées. Elles correspondent à la notation en virgule fixe et en virgule flottante.

B.1 Virgule Fixe

B.1.1 Représentation

La représentation à virgule fixe est la représentation la plus simple à comprendre. C'est aussi la première qui a été utilisée dans les microprocesseurs, les microprocesseurs et les DSP. La notation à virgule fixe repose sur le fait de choisir le nombre de bits qui représentera les nombres après la virgule. Plus il y a de de bits à droite de la virgule plus le nombre peut être précis. Les poids de ces bits situés à droite de la virgule correspondent à des puissances négatives de deux tels que 2^{-bit}.

Ce format peut être appelé le format Q. Pour un nombre de 16 bits avec 8 bits à droite de la virgule, son format est Q16.8. Quand le nombre total de bits est sous-entendu, dans un processeur dont tous les mots sont de 16 bits par exemple, on écrit seulement le nombre de bits après la virgule : Q8.

L'avantage de cette représentation est qu'elle est relativement facile à traiter par un système numérique. Le désavantage principal est que pour des nombres entiers beaucoup de bits peuvent se retrouver inutiles. De plus, on retrouve un antagonisme entre la précision possible à droite de la virgule et la grandeur maximale qui peut être représentée.

Binaire	0	1	0	0	1	0	1	0	1	0	0	1	1	1
Poids	±	128	64	32	16	8	4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$

Décimal : $128 + 16 + 4 + 1 + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} = 149,21875$

Figure 78 – Représentation d'un Nombre Signé de Format Q14.5

Si on désire pouvoir donner un signe à notre nombre binaire, on uti-

lise une opération appelée "complément de deux". Elle consiste à rajouter un bit de signe à gauche du MSB, 0 pour un nombre positif et 1 pour un nombre négatif.
Cependant, on ne peut pas seulement faire cette opération pour représenter des nombres négatifs, car les opérations mathématiques se trouveraient fausses. Pour représenter un nombre négatif en complément à deux, il faut d'abord complémenter tous ses bits (remplacer les 0 par des 1 et les 1 par 0) et ajouter un un au bit de poids le plus faible. Le format Q, quand il est signé, sous-entend que le nombre est représenté en complément de 2. Voici un exemple :

$$-9,625_{10} = -1001,101_2 = 10110,011_2$$

B.1.2 Saturation

Lorsqu'on dépasse le nombre maximal représentable dans un nombre de bits en virgule fixe, deux options sont possibles :

— L'enroulement (ou *warp*) : Le nouveau MSB créé par le dépassement est simplement ignoré. Ainsi, la valeur suivante du maximum est le minimum. Par exemple avec un nombre 4 bits :

$$1111_2 + 1 = 0000_2$$
$$0000_2 - 1 = 1111_2$$

— La saturation : Une fois le maximum ou le minimum atteint, une saturation est en place qui empêche le dépassement de ses valeurs. Donc en 4 bits :

$$1111_2 + 1 = 1111_2$$
$$0000_2 - 1 = 0000_2$$

B.1.3 Addition

Pour montrer comment une addition au format virgule fixe signé est réalisée, on va procéder par exemple. On veut faire l'addition suivante : $-5,25 + 9,09375$. On rappelle que le complément de deux : $-5,25 = -101,01 = 1010,11$.

$$\begin{array}{r} \,1010,11 \\ +\;01001,00011 \\ \hline \end{array}$$

On remarque que les deux nombres ne partagent pas le même format. La première étape sera donc d'ajuster les formats de chaque nombre pour qu'ils soient identiques. Quand le nombre de bits avant la virgule est trop court on ajoute des bits de signes avant et quand le nombre de bits après la virgule est trop court on rajoute des 0 après. L'addition se fait normalement, à l'exception qu'elle est réalisée en base 2. Donc quand deux 1 sont ajoutés on écrit 0 et l'on retient 1.

$$\begin{array}{r} \,{\scriptstyle +1\,\leftarrow\!\urcorner} \\ \,11010,11000 \\ +\;01001,00011 \\ \hline \not{1}00011,11011 \end{array}$$

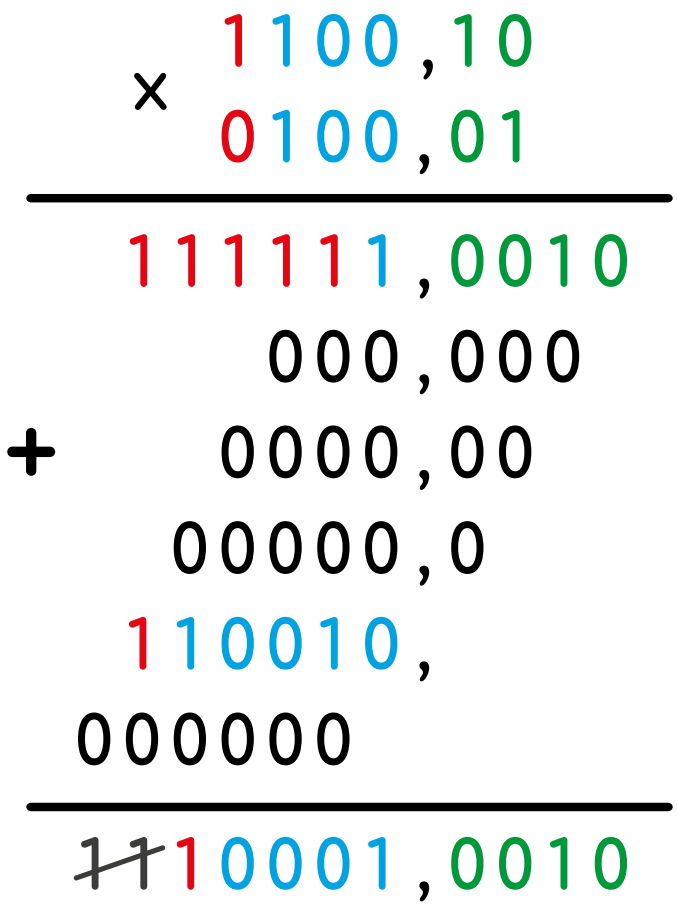
On trouve le bon résultat en complément de 2 qui est $11,11011 = 3,34375$. On ignore le bit supérieur au bit de signe, car le bit de signe fonctionne en modulo 2. On trouve 2 en signe (10 en binaire), et $2\%2 = 0$.
On remarquera que l'opération de serait pas juste si on n'utilise pas le format complément de 2 ou l'extension du bit de signe.

B.1.4 Multiplication

Les multiplications dans le format Q signé sont tout à fait possibles. La procédure à suivre est de multiplier chaque bit du multiplicateur par le multiplicande. Puis, on décale le résultat en fonction de la position du

bit du multiplicateur. Enfin, on additionne le tout pour obtenir le résultat. Lors de l'addition finale, il faut toujours réaliser l'extension du bit de signe.

Nous allons réaliser en exemple la multiplication suivante : $-3,5 \times 4,25 = -14,875$.
Ce qui donne en complément de deux : $1100,10 \times 0100,01 = 10001,001$



B.2 Virgule Flottante

B.2.1 Représentation

La représentation à virgule flottante s'est imposée dans un grand nombre de systèmes numériques récents, car c'est une représentation très précise et flexible. C'est un équivalent à la notation scientifique en mathématiques, avec un nombre à virgule (appelé mantisse) multiplié par une puissance de deux (appelée exposant).
En 1985, l'IEEE (pour Institute of Electrical and Electronics Engineers) établis la norme IEEE-754 qui normalise la représentation des nombres binaires à virgule flottante. Cette représentation prend la forme :

Signe (S)	Exposant (E)	Mantisse (M)
1 bit	e bits = 8 bits	m bits = 23 bits

Figure 79 – Trame d'un Nombre 32 bits à Virgule Flottante Normalisé IEEE-754

$$Décimal = (-1)^S \cdot \left(1 + \frac{M}{2^m}\right) \cdot 2^{E-(2^{e-1}-1)}$$

Avec S la valeur du bit de Signe, E la valeur de l'exposant, e le nombre de bits de l'exposant, M la valeur de la mantisse et m le nombre de bits de la mantisse.

Les formats les plus courants sont la simple précision de 32 bits, couramment appelée *float* (avec e = 8 et m = 23), et la double précision de 64 bits (avec e = 11 et m = 52), couramment appelée *double*.
En simple précision, on remarque que l'exposant représente une puissance de deux entre -126 et 128. C'est là que se trouve la flexibilité de la virgule flottante, on peut représenter, avec la même précision, des valeurs très grandes ou très petites.

Voici une application numérique pour un *float* simple :

$$\begin{aligned}
Binaire &= 0 \quad 10000011 \quad 00110100000000000000000 \\
S &= 0 \\
E &= 131 \\
M &= 1703936
\end{aligned}$$

$$\begin{aligned}
Décimal &= (-1)^0 \cdot \left(1 + \frac{1703936}{2^{23}}\right) \cdot 2^{131-(2^{8-1}-1)} \\
&= 1,203125 \cdot 2^4 \\
&= 19,25
\end{aligned}$$

Le format IEEE-754 à virgule flottante laisse la place pour certaines valeurs particulières dont voici la liste :

Signe	Exposant	Mantisse	Valeur Particulière
0	0	0	+0
1	0	0	-0
X	0	$\neq$0	Valeur Dénormalisée
0	Max	0	+∞
1	Max	0	-∞
X	Max	$\neq$0	Pas un Nombre (NaN)

X signifie Valeur Sans Importance

Figure 80 – Valeurs Particulières de la Norme IEEE-754

— Les valeurs infinies sont utilisées, par exemple, pour montrer que le nombre a dépassé les limites de saturation de la notation.

— La valeur NaN (pour *Not a Number*) indique une valeur erronée, comme la racine d'un nombre négatif par exemple.

— Il y a $2^{m+1} - 1$ valeurs dénormalisées possibles. Elles sont utilisées pour montrer qu'un nombre est sorti de l'intervalle autorisé par la représentation.

B.2.2 Addition

La première étape pour faire une addition en virgule flottante est de ramener les deux nombres sur le même exposant. On décale vers la gauche la virgule de la mantisse du nombre à l'exposant le plus faible jusqu'à atteindre le même exposant que l'autre nombre. Voici un exemple :

$$4,75_{10} + 2,25_{10}$$
$$1,0011_2 \cdot 2^2 + 1,0010_2 \cdot 2^1$$
$$1,0011_2 \cdot 2^2 + 0.1001_2 \cdot 2^2$$

Il ne reste plus qu'à additionner les mantisses et faire en sorte de normaliser le résultat.

$$1,0011_2 \cdot 2^2 + 0.1001_2 \cdot 2^2 = 1,1100_2 \cdot 2^2 = 7_{10}$$

B.2.3 Multiplication

Pour la multiplication en virgule flottante, on commence par déterminer le signe du résultat. Puis on additionne les exposants avant de multiplier les mantisses entre elles. Reprenons l'exemple précédent avec une multiplication :

$$
\begin{aligned}
4,75_{10} \times 2,25_{10} \\
1,0011_2 \cdot 2^2 \times 1,0010_2 \cdot 2^1 \\
1,0011_2 \times 1,0010_2 \cdot 2^{2+1} \\
1,0011_2 \times (1 + 0,001_2) \cdot 2^3 \\
1.0101011_2 \cdot 2^3 \approx 10.687_{10}
\end{aligned}
$$

B.3 Arrondi

Lorsqu'un nombre ne peut pas être représenté dans sa précision originale, on a besoin de l'arrondir. De façon courte, cela consiste à décider comment traiter les bits de poids faible qui sont en trop afin de conformer le nombre trop précis au format utilisé. Il existe plusieurs solutions que voici :

— La Troncature : On ignore simplement les bits de précision en trop.

— Arrondi au plus proche : C'est l'arrondi par défaut dans beaucoup de cas. On arrondit à la valeur représentable la plus proche de la valeur originale. En cas d'égalité c'est la valeur paire qui l'emporte (qui finit par 0 en binaire).

— Arrondi vers $+\infty$: On arrondit à la valeur la plus proche la plus grande.

— Arrondi vers $-\infty$: On arrondit à la valeur la plus proche la plus petite.

C Bus I2C

Le I^2C (Inter Integrated Circuit) est un bus série *half-duplex* synchrone de deux fils [*I^2C - Bus Specification and User Manual* 2014]. On peut y connecter jusqu'à une dizaine de puces qui peuvent chacune être maître ou esclave. Une grande diversité de puces compatibles I^2C est disponible : des capteurs, écrans, potentiomètres...
Chaque membre du bus I^2C est désigné par une adresse de 7 ou 10 bits. De cette façon, on peut s'adresser à une seule puce en particulier, voire plusieurs ou toutes si le besoin se présente.

Les données peuvent être échangées à différentes vitesses allant du mode standard à 100 kbits/s au mode *ultra fast* à 5 Mbits/s. Dans notre cas, le DSD1793 est compatible avec le mode standard et le *fast* mode de 400 kbits/s.

Un bus I^2C comporte deux fils :

— SCL : Serial Clock. C'est le fil d'horloge qui rythme les différentes phases de la transmission.
— SDA : Serial DAta. C'est le fil qui transporte les informations dans les deux sens.

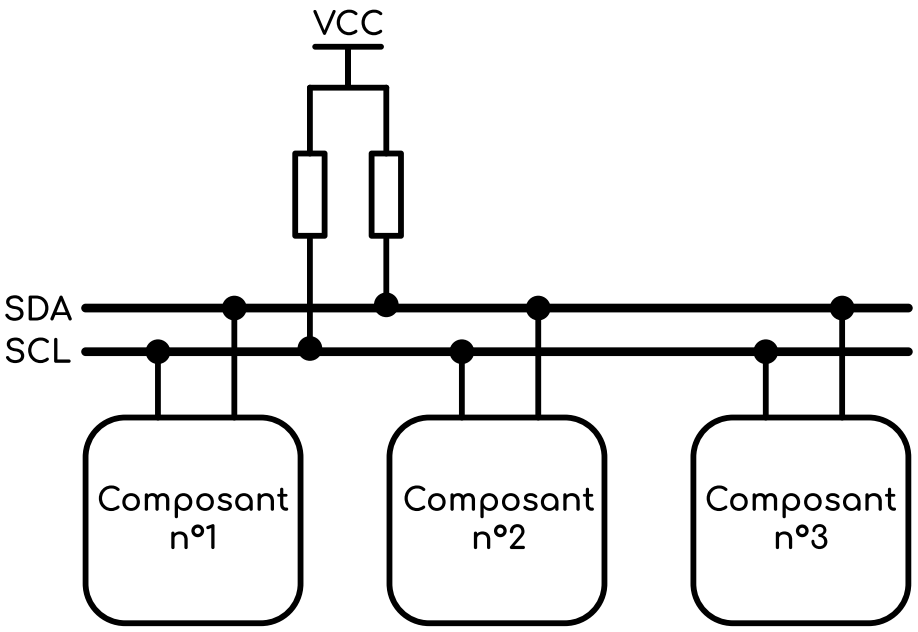
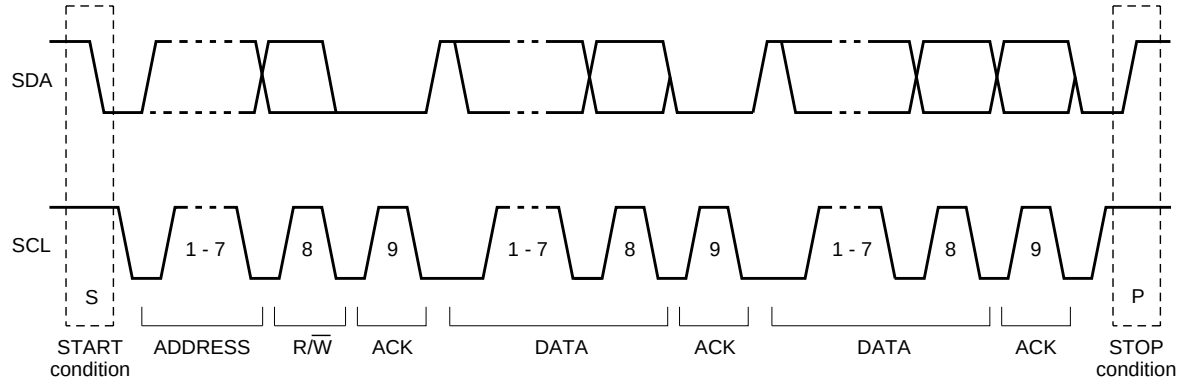


Figure 81 – Schéma d'un bus I^2C

La hiérarchie maître(s)/esclave(s) permet d'organiser la façon d'échanger des informations entre les composants d'un même bus I^2C. En effet, c'est le maître qui initie l'échange de donnée, choisis d'écrire ou de lire des données et enfin, il met fin à la conversation.



ADDRESS : Adresse 7 bits de l'esclave
R/$\overline{W}$: Bit de Lecture ou Écriture
DATA : Octet d'Information
ACK : Accusé de Réception

Figure 82 – Une Transmission Complète en I^2C [*I^2C - Bus Specification and User Manual* 2014]

D Bus DSD

Le bus DSD n'est pas normalisé comme le bus I^2C, mais il transporte les informations essentielles pour une liaison audio de type DSD. Cette liaison comporte 3 fils :

— DBCK : DSD Bit Clock. C'est l'horloge qui indique quand chaque nouveau bit d'information est disponible. Sa fréquence est la même que la fréquence d'échantillonnage du DSD, typiquement 64 fois 44,1 kHz.

— DSDL : DSD Left. C'est le fil qui transporte le flux Σ/Δ du canal de gauche.

— DSDR : DSD Right. C'est le fil qui transporte le flux Σ/Δ du canal de droite.

Grâce à la figure 83, on remarque que les fronts montants de l'horloge se trouvent au centre des échantillons et que les échantillons changent en même temps que les fronts descendants de l'horloge. C'est une façon de limiter la corruption d'information lors de la lecture du bus. En effet, on peut faire en sorte de lire l'information DSD sur le front montant de l'horloge lorsque l'échantillon DSD est à son état le plus stable.

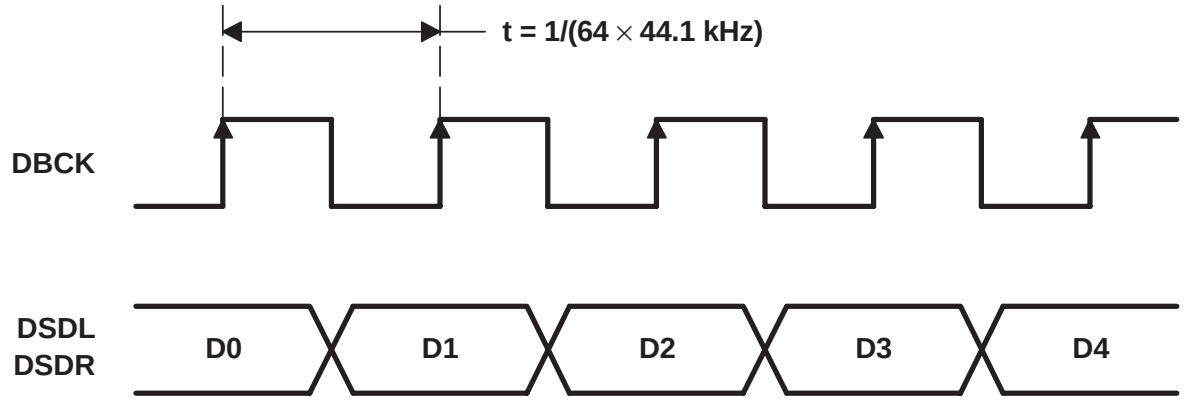


Figure 83 – Informations Transmises dans le bus DSD [*24 Bits 192 kHz Audio Stereo DAC* 2006, Figure 38, p.39]

E Bus AXI

Le bus AXI est développé par l'entreprise ARM dans le cadre de son format ouvert AMBA (Advanced Microcontroller Bus Architecture). C'est un protocole de communication haute performance principalement créé pour la communication au sein d'une puce.
On peut comprendre pourquoi une entreprise telle que ARM voudrait développer ce genre de bus. En effet, avec une batterie de bus puissants, les fabricants de systèmes numériques peuvent intégrer les IP ARM à leurs produits plus facilement.

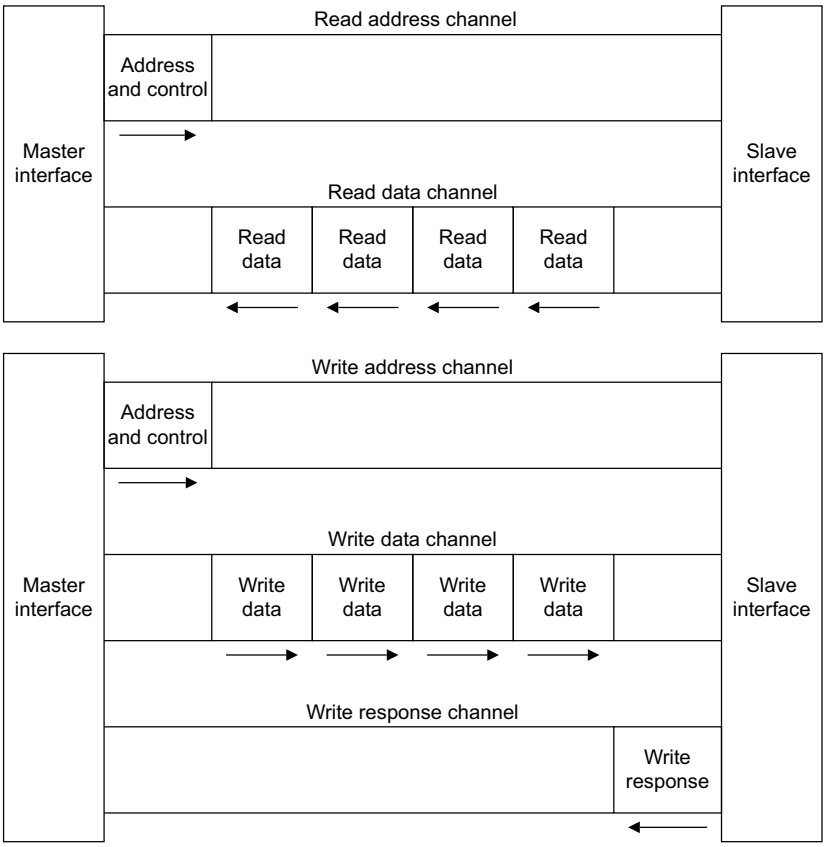


Figure 84 – Détail des Bus d'une Connexion AXI4 en Lecture (en haut) et en Écriture (en bas) [*AMBA AXI and ACE Protocol Specification* 2013, p.24, Figures A1-1 et A1-2]

Le SoC de la Zedboard utilise le bus AXI4 sorti en 2010. Dans son principe, le bus AXI généralise la façon d'écrire et de lire une mémoire, ce qui le rend compatible avec un grand nombre de systèmes. Il est synchrone, full-duplex et autorise plusieurs maîtres et plusieurs esclaves.
Le principe repose sur des requêtes émises par le maître et exécutées par

l'esclave. Les requêtes peuvent être émises à la suite sans attendre de réponse immédiate. Les réponses seront alors émises dans le même ordre que les requêtes.
Une connexion typique d'un bus AXI4 comporte trois bus de connexions pour l'écriture et deux bus de connexion pour la lecture.

Voici quelques caractéristiques du bus AXI :

— Un composant d'interconnexion est nécessaire pour connecter un maître à un esclave. Ce composant se comporte comme une interface esclave pour le maître et comme une interface maître pour un esclave. Il est chargé de faire correspondre les registres du maître à celui de chaque esclave.

— Chaque bus de données peuvent transporter simultanément de 1 à 128 octets de données. Une ligne par octet est ajoutée pour confirmer que l'octet est valide.

— L'AXI tiens compte du fait que le transfert puisse échouer.

— Les échanges peuvent être faits par paquets appelés *bursts*. Ainsi, il est possible de demander d'intervenir sur une plage d'adresses dans la mémoire et de les traiter en un seul *burst*.

— Le maître peut indiquer quand il est prêt à lire les informations. Dans certains bus plus simples, il est supposé que le maître soit toujours prêt à recevoir des informations quand il les demande.

L'AXI-lite est une version simplifiée de l'AXI qui limite les bursts à une information de 4 octets maximum.