UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Implementação em FPGA de módulo de efeito de reverberação

Diego Vinícius Cirilo do Nascimento

Orientador: Prof. Dr. Valentin Obac Roda

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de ordem do PPgEEC: M433
Natal, 2014

UFRN / Biblioteca Central Zila Mamede
Catalogação da Publicação na Fonte

Nascimento, Diego Vinícius Cirilo do.
Implementação em FPGA de módulo de efeito de reverberação / Diego Vinícius Cirilo do Nascimento. – Natal, RN, 2014.
79 f. : il.

Orientador: Prof. Dr. Valentin Obac Roda.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e de Computação.

1. Processamento digital de áudio – Dissertação. 2. Reverberação – Dissertação. 3. Sistemas digitais – Dissertação. 4. FPGA – Dissertação. I. Roda, Valentim Obac. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM CDU 681.3:78

Implementação em FPGA de módulo de efeito de reverberação

Diego Vinícius Cirilo do Nascimento

Dissertação de Mestrado aprovada em 4 de agosto de 2014 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Valentin Obac Roda (Orientador) ..... DEE/UFRN

Prof. Dr. Allan de Medeiros Martins (Examinador Interno) ..... DEE/UFRN

Prof. Dr. Emerson Carlos Pedrino (Examinador Externo) ..... DC/UFSCar

Prof. Dr. José Alberto Nicolau de Oliveira (Examinador Interno) . DEE/UFRN

Agradecimentos

Agradeço a meus pais e familiares por todo o apoio, incentivo e investimento.

Agradeço a Ludmila, pelo carinho e incentivo.

Aos *brothers-in-arms* do IFRN e aos amigos de sempre pela motivação e pelos momentos de descontração.

Ao meu orientador neste trabalho, professor Valentin Obac, e ao orientador "de sempre", professor Alberto Nicolau, pela paciência e tutoria nesse período.

A todos os professores que de forma prazerosa ou não me apresentaram o conhecimento.

Por fim agradeço também a CAPES pelo apoio financeiro.

“If you want to accomplish something in the world, idealism is not enough — you need to choose a method that works to achieve the goal. In other words, you need to be pragmatic.”
(Richard Stallman)

Resumo

O efeito de reverberação ocorre pela reflexão do som nas superfícies adjacentes à fonte sonora durante sua propagação até o ouvinte e a resposta ao impulso de um ambiente representa suas características de reverberação. Por ser dependente do ambiente, a reverberação leva ao ouvinte características do espaço onde o som está sendo reproduzido e comumente sua ausência não soa como "natural". Como nem sempre é possível ter características desejáveis de reverberação em gravações, métodos para reverberação artificial vêm sendo desenvolvidos há décadas, sempre buscando implementações mais eficientes e mais fiéis aos ambientes reais. Neste trabalho é apresentada uma implementação em FPGA (*Field Programmable Gate Array*) de uma estrutura de reverberação digital de áudio clássica, usando conjuntos de filtros *allpass* e *comb*, baseada na proposta de Manfred Schroeder, demonstrando a utilização de *hardware* reconfigurável como plataforma de desenvolvimento e implementação de efeitos digitais de áudio, com foco nas características de modularidade e de reuso.

Palavras-chave: Processamento digital de áudio. Reverberação. Sistemas digitais. FPGA.

Abstract

Reverberation is caused by the reflection of the sound in adjacent surfaces close to the sound source during its propagation to the listener. The impulsive response of an environment represents its reverberation characteristics. Being dependent on the environment, reverberation takes to the listener characteristics of the space where the sound is originated and its absence does not commonly sounds like “natural”. When recording sounds, it is not always possible to have the desirable characteristics of reverberation of an environment, therefore methods for artificial reverberation have been developed, always seeking a more efficient implementations and more faithful to the real environments. This work presents an implementation in FPGAs (Field Programmable Gate Arrays) of a classic digital reverberation audio structure, based on a proposal of Manfred Schroeder, using sets of all-pass and comb filters. The developed system exploits the use of reconfigurable hardware as a platform development and implementation of digital audio effects, focusing on the modularity and reuse characteristics.

Keywords: Digital audio processing. Reverberation. Digital systems. FPGAs.

Lista de ilustrações

Lista de quadros

Lista de abreviaturas e siglas

AC'97: *Audio* CODEC 97

ACM: *Association for Computing Machinery*

AES: *Audio Engineering Society*

ASIC: *Application Specific Integrated Circuit*

BBC: *British Broadcasting Corporation*

BBD: *Bucket-brigade device*

BRAM: *Block RAM*

CAPES: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CI: Circuito Integrado

CODEC: *Coder – Decoder*

DPRAM: *Dual Port RAM*

DSP: *Digital Signal Processor*

FIFO: *First In, First Out*

FPGA: *Field Programmable Gate Array*

IEEE: *Institute of Electrical and Electronic Engineers*

MFP: *Mean Free Path*

PLD: *Programmable Logic Device*

VCD: *Value Change Dump*

VHDL: VHSIC *Hardware Description Language*

VHSIC: *Very High Speed Integrated Circuit*

Sumário

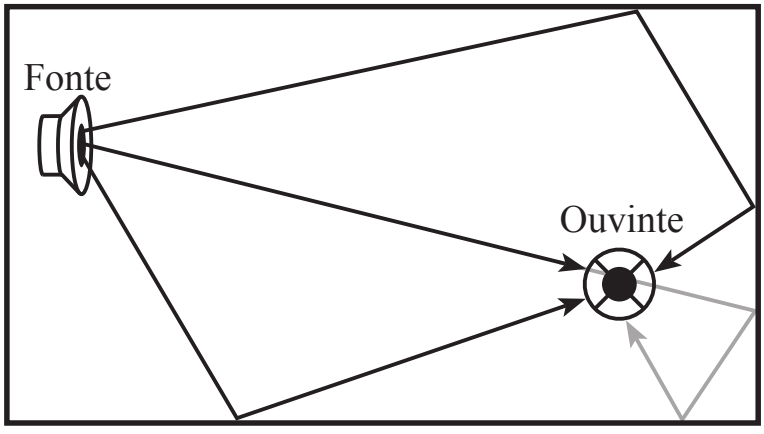
1 Introdução

Dentre os efeitos naturais que ocorrem com o áudio, o da reverberação é um dos mais comuns, e é percebido como o prolongamento do som após a sua produção. Esse efeito se dá pela reflexão do som nas superfícies próximas durante o trajeto entre a fonte sonora e o destino, de forma que o som recebido seja composto pela componente direta fonte – ouvinte, e todos os caminhos alternativos, gerados pela reflexão do som. Como cada um desses caminhos têm comprimentos diferentes, o atraso proporcionado pelos mesmos também é diferente, a exemplo da Figura 1.

A reverberação é bem mais complexa que o popular "eco", já que não apresenta apenas reflexões discretas e espaçadas, mas sim uma grande densidade de reflexões. Essas reflexões trazem consigo características do ambiente onde o som está sendo reproduzido, sendo dependente da distância entre as superfícies, dos materiais refletores, além de outros fatores, inclusive de características do meio de propagação do som, o ar. Assim, por essa relação direta com o ambiente, a reverberação é o efeito que nos dá a sensação auditiva de *ambiência* e é característico do local (ZÖLZER, 2011). É claramente possível diferenciar o som gerado dentro de uma catedral gótica, com suas grandes paredes de pedra, o som gerado dentro de um quarto de dormir, com suas paredes próximas e superfícies absorvedoras de som, como colchões, e o som gerado em um banheiro, com suas paredes de cerâmica e próximas.

Evolutivamente a reverberação tem importância crucial, pois é capaz de permitir que o cérebro compute a distância aproximada de alguma fonte sonora, e a sua ausência pode ser perturbadora pelo fato do cérebro humano precisar dessa informação no processo de decodificação de sons (BLAUERT, 1997). Justamente assim a reverberação é fator importante para que um som seja considerado agradável para os ouvintes (BELTRÁN; BELTRÁN, 2002). Porém, sendo

Figura 1 – Diferentes caminhos de reflexão do som.



Fonte: Autoria Própria.

excessiva, é capaz de dificultar a inteligibilidade de sons, inclusive sendo um grande problema da área de comunicações e reconhecimento de voz (SHI; MA, 2012). Deste modo, ambientes utilizados para gravação de áudio devem ser construídos de tal forma que a reverberação seja agradável aos ouvintes e não dificulte a inteligibilidade do material.

Para obter sons com reverberação natural e em níveis controlados, é necessário utilizar salas especiais, construídas exatamente com o fim de reverberar o som ao gosto do músico ou engenheiro de áudio, ou realizar gravações em ambientes externos ao estúdio, que já apresentem as características desejadas, como salas de concerto e catedrais. Infelizmente essas abordagens são na maioria das vezes inviáveis, dado o grande custo envolvido. Justamente pelas limitações citadas, estúdios de gravação utilizam métodos acústicos de redução de reverberação, de forma a obter uma gravação mais limpa, ou pura, com o mínimo possível de influências ambientais indesejadas. Em contrapartida, o resultado desse tipo de gravação é geralmente muito seco e sem vida, e não soam naturalmente já que os ambientes reais reverberam o som. A abordagem prática nesse momento é o uso de dispositivos de reverberação artificial que possam suprir essa deficiência (KAHRS; BRANDENBURG, 2002).

As primeiras técnicas artificiais de reverberação se baseavam no uso de equipamentos eletromecânicos, que reproduziam o efeito da reflexão e atraso de sons por meio de molas, placas de metal ou gravação repetida em fitas magnéticas. Todos esses métodos apresentam uma boa qualidade de áudio, tendo sido utilizados por muitos anos como padrão da indústria. Apresentam, porém, vários problemas inerentes aos dispositivos eletromecânicos como, por exemplo, maior necessidade de manutenção, menor confiabilidade e portabilidade, além de ter um custo alto de produção (VALIMAKI et al., 2012).

Com o desenvolvimento da eletrônica para o processamento de sinais, e mais especificamente dos sistemas digitais, a primeira proposta de uma unidade eletrônica de reverberação foi feita, por Schroeder e Logan (1961) em seu artigo "*Colorless Artificial Reverberation*", onde é apresentada a utilização de filtros *comb* e filtros *allpass* como elementos de reverberação. A partir desse artigo, várias técnicas e algoritmos para reverberação foram propostas, podendo ser classificadas basicamente em três categorias (VALIMAKI et al., 2012):

- Redes de atrasos: várias malhas com atrasos, filtros e realimentação parametrizados de forma a simular as reflexões e alterações do sinal causados pela reverberação;
- Convolucionais: é realizada a convolução do sinal original com a resposta ao impulso gravada ou estimada do ambiente desejado;
- Acústica computacional: o sinal é processado por um modelo computacional do ambiente em questão.

Cada uma das abordagens citadas tem seus pontos positivos e negativos. As redes de atrasos são as de implementação mais direta, e as mais flexíveis, pois permitem o controle

de seus parâmetros, e o algoritmo implementado neste trabalho se encaixa nessa categoria. Os algoritmos convolucionais são os mais fiéis aos ambientes reais, porém não apresentam flexibilidade para ajustes de parâmetros como tamanho do ambiente a ser simulado, além de uma grande carga computacional. Os modelos computacionais podem ser bem fiéis e ainda apresentar flexibilidade, porém apresentam alta complexidade de projeto que se reflete no alto consumo de recursos computacionais.

As técnicas de reverberação apresentadas podem ser implementadas por diversos dispositivos, desde o uso de computadores pessoais e processadores de uso geral, até a criação de *Application Specific Integrated Circuits* (ASICs), ou circuitos integrados de aplicação específica. As mais comuns atualmente são as implementações em *software*, que são executadas em processadores de uso geral ou *Digital Signal Processors* (DSPs), processadores de uso específico para o processamento digital de sinais.

Como os algoritmos de reverberação são comumente intensivos tanto no uso de processador como de memória, devido aos longos atrasos, abordagens em *software* nem sempre podem ser eficientes o suficiente para que sejam executadas em tempo real por processadores de baixo consumo de energia e baixo custo. E justamente os dispositivos portáteis usados por músicos e produtores musicais têm restrições de consumo, custo e tamanho. O uso de ASICs, um *hardware* dedicado, produzido especificamente para aquela aplicação, é limitado pelo fator custo de projeto. Normalmente ASICs são utilizados em projetos que possam ter uma alta escala de produção, o que não é o caso de equipamentos de um nicho específico de mercado, como os efeitos de áudio.

A partir dos anos 80 um meio termo entre os dispositivos de uso geral e os de aplicações específicas começou a se popularizar através do desenvolvimento de dispositivos de lógica programável, ou *Programmable Logic Devices* (PLDs).

PLDs sintetizam em *hardware* arquiteturas descritas através de linguagens de descrição de *hardware*, ou *Hardware Description Languages* (HDLs), a exemplo do *VHSIC Hardware Description Language* (VHDL). Essa característica de descrição de *hardware* permite o reuso de blocos já implementados, reduzindo o tempo de projeto e o retrabalho, além de possibilitar uma maior característica de modularidade.

Um dispositivo da família dos PLDs, o *Field Programmable Gate Array* (FPGA), vem se destacando como um dispositivo de baixo consumo, baixo custo e alta confiabilidade, sendo adequado para o uso em sistemas embarcados e portáteis que necessitem de eficiência no desempenho de suas funções (RODRIGUEZ-ANDINA; MOURE; VALDES, 2007). Foram inicialmente voltados para prototipagem rápida de sistemas digitais e atualmente são utilizados tanto como acelerador de execução em sistemas computacionais e até mesmo em dispositivos de consumo finais. A partir da popularização dos FPGAs e da lógica reconfigurável, essa plataforma vem se mostrando viável para aplicações de processamento digital de sinais, como o abordado neste trabalho (VILLASENOR; HUTCHINGS, 1998).

O presente trabalho apresenta uma implementação de um sistema completo de reverberação, desde as entradas analógicas, que podem ser provenientes de um microfone, instrumento ou qualquer outro dispositivo de áudio, passando pela unidade de efeito, responsável por executar o algoritmo de reverberação até a conversão digital – analógico (D/A) ao sinal de saída. O algoritmo utilizado será o proposto por Schroeder (1961). Tal sistema será implementado em FPGA, exceto pelas fases de conversão A/D – D/A, de forma a usufruir das características inerentes à plataforma, como já citadas, baixo consumo, baixo custo e em especial a possibilidade de reuso e modularidade, o que possibilitaria a implementação de outros algoritmos de reverberação ou até mesmo de outros efeitos na mesma plataforma.

Considerando-se as questões já apresentadas, o objetivo geral deste trabalho é apresentar o processo e os resultados da implementação de um módulo completo de reverberação de áudio em FPGA. O algoritmo utilizado para a implementação será o proposto por Schröeder. Como objetivos específicos temos:

- Estudar o *Coder – Decoder* (CODEC) de áudio LM4550 e o padrão *Intel* AC'97;
- Estudar o fluxo de projeto e implementação para FPGAs utilizando VHDL;
- Implementar o controle de tal CODEC em FPGA, conseguindo o fluxo completo de conversão A/D e D/A;
- Apresentar a teoria da reverberação de áudio e os métodos artificiais para sua obtenção;
- Apresentar o trabalho de Schröeder e seu algoritmo de reverberação;
- Implementar tal algoritmo em FPGA;
- Integrar o módulo do algoritmo ao módulo do CODEC;
- Realizar testes que demonstrem a compatibilidade entre os resultados esperados pelo algoritmo proposto e os resultados obtidos na prática.

Para fins de organização o presente trabalho está dividido em seis capítulos. A seguir é apresentada uma breve descrição dos mesmos.

- O capítulo atual apresentou a motivação, com uma breve introdução aos temas que serão abordados no decorrer do trabalho, os trabalhos relacionados e os objetivos;
- Já no Capítulo 2 é apresentada a fundamentação teórica e técnica sobre reverberação, focando na sua base acústica, importante para o entendimento dos algoritmos apresentados no capítulo subsequente;
- O Capítulo 3 apresenta os algoritmos de reverberação mais famosos, com foco especial no algoritmo de Schroeder, utilizado neste trabalho. Também são apresentados os trabalhos relacionados ao tema de reverberação em FPGA;

- O Capítulo 4 traz detalhes sobre o projeto e a implementação que foi realizada, apresentando todo o fluxo de trabalho;
- Os resultados do trabalho são apresentados no Capítulo 5;
- Por fim o Capítulo 6 faz as considerações finais, apresentando as contribuições deste trabalho e as possibilidades de desenvolvimento futuro.

2 Características da Reverberação Natural

Neste capítulo será apresentada a base teórica sobre reverberação e sobre alguns métodos artificiais para sua obtenção.

2.1 A reverberação

A reverberação é um efeito natural que ocorre quando ondas sonoras se propagam em um ambiente fechado. Por ambiente fechado entende-se qualquer ambiente que possa apresentar obstáculos à propagação do som, em contraste com o espaço aberto. É fator determinante para o processamento de informações auditivas pelo cérebro, até pela necessidade natural de tentar descobrir a distância dos sons gerados, para proteção contra predadores, etc (BLAUERT, 1997).

Desde a antiguidade o homem vem utilizando ambientes com características específicas de reverberação para melhoria da reprodução de áudio, seja com fins artísticos ou utilitários. Exemplos são os anfiteatros gregos, como apresentado na Figura 2, que permitiam, por meio das reflexões do som, um reforço no áudio gerado no palco. Igrejas góticas, com suas abóbadas de pedra apresentam uma reverberação excessiva, que dá um caráter sombrio ou místico ao ambiente, vide Figura 3. Até hoje casas de espetáculo se baseiam nos mesmos princípios acústicos para obter ambientes com características específicas de reverberação. Na Figura 4 vemos um teatro atual, e como sua construção é semelhante ao antigo teatro grego.

Figura 2 – Anfiteatro grego

Fonte: Hansueli Krapf. [1]

[1] Disponível em: <http://en.wikipedia.org/wiki/File:2007-05-10_Epidauros,_Greece_5.jpg>. Acesso em 30/10/2013

Figura 3 – Catedral Gótica

Fonte: Antoine[2]

Figura 4 – Teatro moderno

Fonte: Giovanni Sérgio[3]

2.2 Acústica de ambientes fechados

As bases da acústica moderna foram definas por Sabine (1906), tendo sido mais tarde complementadas por outros autores, inclusive Schroeder (1987).

As características de reverberação de um ambiente podem ser descritos pela sua resposta ao impulso, apresentada na Figura 5. Tal resposta é composta por três momentos definidos:

- Sinal direto: É o som que se propaga diretamente da fonte ao ouvinte, sendo o sinal de menor atraso e normalmente de maior intensidade;

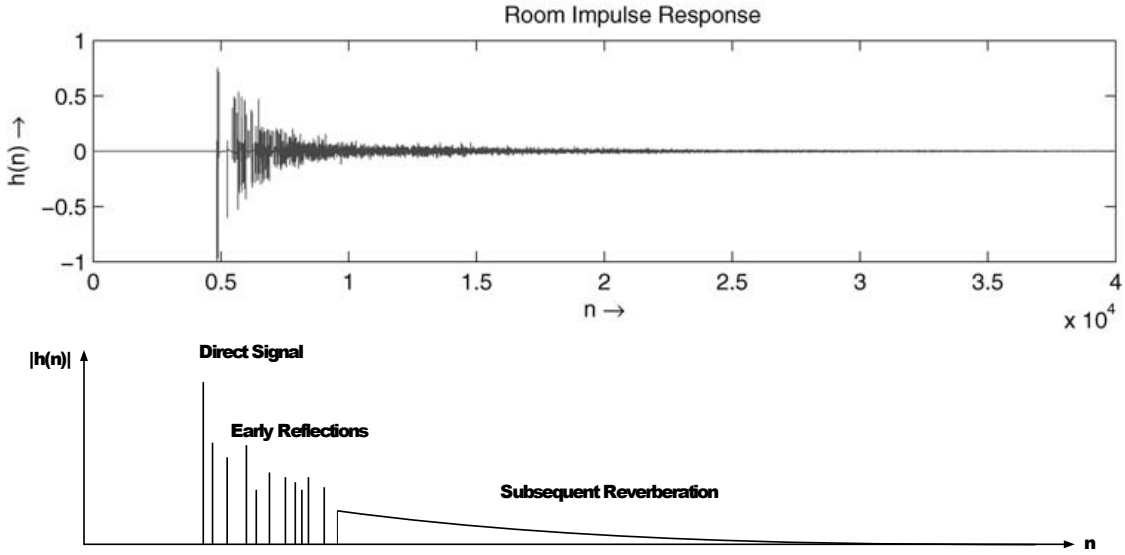
[2] Disponível em: <http://en.wikipedia.org/wiki/File:Interieur_cathedrale_de_wells.JPG>. Acesso em 30/10/2013

[3] Disponível em: <http://teatroriachuelo.com.br/fotos.php>. Acesso em 30/10/2013

- Reflexões iniciais: São os caminhos mais curtos de reflexão, ocorrem pela reflexão do som nas superfícies adjacentes à fonte sonora. Possuem um atraso mais longo que o sinal direto;

- Reflexões finais: Principal característica da reverberação e o foco da maioria dos algoritmos. São causadas pela composição de todas as reflexões secundárias, que ocorrem entre duas ou mais superfícies do ambiente. A densidade de reflexões é crescente, enquanto sua amplitude decai. Sua densidade pode ser tão alta que idealmente seria possível ser modelado por um ruído branco decaindo exponencialmente, segundo Moorer (1979).

Figura 5 – Resposta ao impulso de uma sala e sua versão simplificada decomposta em sinal direto, reflexões iniciais e reflexões finais.



Fonte: Zölzer (1997)

A intensidade do sinal direto pode ser calculada pela Equação 2.1 (HOWARD; ANGUS, 2009):

$$I_{direto} = \frac{QW_{fonte}}{4\pi r^2} \tag{2.1}$$

A intensidade é definida em W/m^2, Q é uma constante de diretividade relativa à uma esfera, W_{fonte} é a potência da fonte, em *Watts*, e r é a distância entre a fonte e o ouvinte, em metros. A constante de diretividade é 1 quando a fonte é omnidirecional, ou seja, o som pode se propagar em todas as direções, como uma esfera. Seu valor é inversamente proporcional à frações da esfera. O Quadro 1 ilustra o conceito.

Quadro 1 – Exemplos de coeficientes de diretividade

Geometria	**Q**
Nenhuma superfície adjacente, situação onde o som pode se propagar em todas as direções, como um som gerado no espaço livre.	1
Som gerado próximo a uma superfície plana, onde o som pode se propagar como em um hemisfério.	2
Som gerado próximo a duas superfícies planas perpendiculares, formando um quarto de esfera.	4
Som gerado próximo a três superfícies planas perpendiculares, como um canto de parede, formando um oitavo de esfera.	8

E o atraso de cada sinal que o ouvinte recebe é simplesmente calculado através da Equação 3.1 (HOWARD; ANGUS, 2009).

$$Atraso = \frac{r}{c} \tag{2.2}$$

Onde o atraso é dado em segundos, r é a distância percorrida pelo som e c a velocidade do som no meio, em m/s. Para o ar seco tal velocidade é de aproximadamente $344m/s$.

Outro parâmetro importante para a reverberação é o cálculo do caminho médio que o som percorre entre a fonte e o ouvinte, considerando as reflexões. Tal parâmetro é conhecido como *Mean Free Path* (MFP) e é apresentado na Equação 2.3

$$MFP = \frac{4V}{S} \tag{2.3}$$

Onde o MFP é dado em metros, V é o volume da sala e S a área de superfície da sala. O tempo entre as reflexões (τ) pode ser calculado dividindo-se a Equação 2.3 pela velocidade do ar (c):

$$\tau = \frac{4V}{Sc} \tag{2.4}$$

2.3 Tempo de reverberação

Sabine (1906) definiu em seu trabalho o chamado tempo de reverberação, que é o tempo até que ocorra um decaimento de 60 dB na intensidade sonora. O cálculo de tal parâmetro, apresentado na Equação 2.5, foi desenvolvido empiricamente e é realizado com base na geometria do ambiente, nas áreas parciais que absorvem o som e seus respectivos coeficientes de absorção.

$$T_{60} = 0.163\frac{V}{\alpha S} = 0.163\frac{V}{\sum_n \alpha_n S_n} \tag{2.5}$$

Onde T_{60} é o tempo de reverberação, em segundos, V é o volume do ambiente em m^3, α_n o coeficiente de absorção da área parcial S_n, que no caso da absorção total é 1. É importante notar que a Equação 2.5 é dependente da frequência, já que o coeficiente de absorção também o é. E justamente essa é a maior falha de vários algoritmos de reverberação artificial, apresentados no Capítulo 3, não conseguir simular tempos de decaimento diferentes para diferentes bandas de frequência. O Quadro 2 apresenta alguns exemplos de valores para esse coeficiente.

Quadro 2 – Coeficientes de absorção para alguns materiais

Material	**Frequência**					
	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
Gesso sobre madeira	0.14	0.10	0.06	0.05	0.04	0.03
Carpete sobre concreto	0.02	0.06	0.14	0.37	0.60	0.65
Piso de madeira	0.15	0.11	0.10	0.07	0.06	0.07
Gesso pintado	0.01	0.01	0.02	0.02	0.02	0.02
Parede (1,27 cm de gesso)	0.29	0.10	0.05	0.04	0.07	0.09
Janela (vidro)	0.35	0.25	0.18	0.12	0.07	0.04
Painel de madeira	0.30	0.25	0.20	0.17	0.15	0.10
Cortina de algodão	0.07	0.31	0.49	0.81	0.66	0.54

Fonte: Howard e Angus (2009)

A Equação 2.5 foi posteriormente melhorada por Eyring-Norris, visando a correção da fórmula de Sabine, dado que em um ambiente totalmente absorvente, ou seja $\alpha = 1$, a fórmula não dá como resultado um tempo de reverberação nulo, já que teoricamente não há reverberação (EVEREST; POHLMANN, 2009). A proposição de Eyring-Norris é apresentada na Equação 2.6, e é possível observar que no caso de $\alpha = 1$, $ln(1-\alpha)$ tende ao infinito, trazendo o tempo de reverberação para zero, como se espera. Finalmente, o termo $4mV$ pode ser somado ao denominador da equação para que seja considerada a absorção do ar, onde m é o coeficiente de absorção do ar, dependente da frequência e da umidade do ar, e V o volume da sala.

$$T_{60} = -0.163 \frac{V}{\sum_n ln(1-\alpha_n) S_n} \tag{2.6}$$

Everest e Pohlmann (2009) afirmam que para um coeficiente médio de absorção menor que 0.25 a equação de Sabine é uma aproximação perfeitamente válida, sendo a equação de Eyring-Norris utilizada apenas para ambientes muito absorventes, devido à complexidade adicional desnecessária.

2.4 Reverberação em termos de frequência

Uma outra abordagem é a descrição do ambiente em termos de frequência, ou uma descrição *modal*. Considerando uma sala ideal, perfeitamente retangular com paredes rígidas, é possível descrever matematicamente seu comportamento em termos de frequências ressonantes, ou modos normais, calculados pela Equação 2.7 (ZÖLZER, 2011).

$$f_n = \frac{c}{2}\sqrt{\left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 + \left(\frac{n_z}{L_z}\right)^2} \tag{2.7}$$

Onde f_n é a n-ésima frequência normal, em Hz; n_x, n_y e n_z são números inteiros de 0 a ∞ que definem o modo normal; L_x, L_y e L_z são as dimensões do ambiente em metros e c a velocidade do som em m/s.

O número de modos normais abaixo de uma frequência f pode ser aproximado pela Equação 2.8, considerando V como o volume da sala e c como a velocidade do som (GARDNER, 2002).

$$N_f \approx \frac{4\pi V}{3c^3} f^3 \tag{2.8}$$

A densidade modal em função da frequência pode ser encontrada diferenciando-se a Equação 2.8 em f, obtendo-se:

$$\frac{dN_f}{df} \approx \frac{4\pi V}{3c^3} f^2 \tag{2.9}$$

Gardner (2002) aponta que tais características são muito difíceis de se obter analiticamente em um ambiente real e irregular, no entanto a Equação 2.9 é normalmente verdadeira para ambientes irregulares. Pode ser mostrado (GARDNER, 2002) que em altas frequências, há um grande número de modos sobrepostos, sendo possível modelar a resposta em frequência do ambiente por meio de variáveis aleatórias. Essa afirmativa possibilita a extração de diversos dados estatísticos, porém só pode ser considerada a partir da frequência crítica f_g:

$$f_g \approx 2000\sqrt{\frac{T_{60}}{V}} Hz \tag{2.10}$$

Onde T_{60} é o tempo de reverberação, dado pela equação 2.5 e V o volume do ambiente. A separação média dos máximos é dada por:

$$\Delta f_{max} \approx \frac{4}{T_{60}} Hz \tag{2.11}$$

Gardner (2002) ainda mostra mais dois dados, o número de reflexões que ocorrerão até o momento t, após a produção do som, calculada através da Equação 2.12, e finalmente a

densidade temporal de reflexões, obtida diferenciando-se a Equação 2.12 em t, apresentada na Equação 2.13. É importante notar como a densidade de reflexões cresce quadraticamente com o tempo, sendo também uma das metas a serem alcançadas nos algoritmos de reverberação artificial.

$$N_t = \frac{4\pi(ct)^3}{3V} \tag{2.12}$$

$$\frac{dN_t}{dt} = \frac{4\pi c^3}{V} t^2 \tag{2.13}$$

2.5 Conclusão

O presente capítulo apresentou as bases da acústica e da reverberação para que seja possível identificar os parâmetros importantes, que serão exatamente os objetivos dos algoritmos de reverberação artificial. O Capítulo 3 apresentará algumas abordagens para reverberação artificial, com foco no algoritmo de Schroeder.

3 Reverberação Artificial

O presente capítulo traz uma visão geral das técnicas de reverberação artificial, com foco na arquitetura desenvolvida por Schroeder e utilizada no decorrer do trabalho.

3.1 Histórico

Considerando o início da era do som gravado, a partir da invenção do cilindro fonográfico por Thomas Edison e posteriormente do som transmitido por rádio, foi possível que ouvintes pudessem ter acesso a apresentações artísticas e musicais que antes eram apenas disponíveis *in loco* nos teatros e auditórios (GARDNER, 2002). A grande diferença era justamente o fato do som ser reproduzido em casa, que dificilmente teria as mesmas características acústicas de um teatro, ficando a reprodução prejudicada, por não poder contar com as características agradáveis de reverberação que os ambientes próprios apresentariam.

A primeira abordagem para transmitir sons mais ricos, foi a de utilizar os próprios teatros como estúdios de gravação. Além da complexidade do posicionamento de microfones para captar de forma natural o som ambiente havia também o fator custo, já que não havia um público pagante no local que justificasse o uso da estrutura. Com a tendência dos estúdios se tornarem cada vez menores, as primeiras técnicas de reverberação artifical foram desenvolvidas.

Segundo Axon, Gilford e Shorter (1955), existiam salas especiais para reverberação, construídas ou adaptadas apenas para esse propósito, porém ainda havia a barreira do custo. Em 1949 os engenheiros da *British Broadcasting Corporation* (BBC) apresentaram como solução o uso de um tubo de atraso, basicamente um tubo fechado com um microfone em uma ponta e um alto falante em outra, longo o suficiente para que o atraso no áudio fosse perceptível. O equipamento era então conectado à uma malha de realimentação elétrica, de forma a se obter uma maior densidade de reflexões.

Como o uso de tubos de atraso não trazia resultados satisfatórios, a pesquisa se voltou para o uso de unidades de gravação de fita magnética, que poderiam ser escritas e lidas em posições diferentes, gerando consequentemente um atraso. A primeira dessas unidades entrou em uso em julho de 1952, e mesmo assim ainda não apresentava características muito agradáveis de som (AXON; GILFORD; SHORTER, 1955).

3.1.1 Reverberadores Eletromecânicos

Um dos primeiros modelos de reverberadores eletromecânicos foi criado pela Hammond, empresa que produzia orgãos também eletromecânicos na época, e utilizava molas para repercutir o som. A Figura 6 mostra um dos chamados tanques de molas. Em uma das extre-

midades temos um solenóide, que é conectado a uma fonte de áudio, e na outra extremidade um tipo de microfone, que capta o sinal proveniente das molas. Pela própria característica oscilatória, a mola consegue reverberar o sinal aplicado na entrada, porém apresenta um som bem distinto de um ambiente real, mas obteve êxito e continua sendo utilizado até hoje (VALIMAKI et al., 2012).

Figura 6 – Tanque de molas

Fonte: Adaptado de Accu-Bell Sound Inc.[1]

Outro reverberador eletromecânico de sucesso foi o modelo *plate*, ou placa, onde uma grande placa de metal suspenso pelas extremidades possui, assim como o reverberador de mola, um solenóide que excita o sistema a partir de uma entrada de áudio e um microfone que capta o som final. O reverberador placa apresenta características mais próximas de um ambiente real (VALIMAKI et al., 2012). Além dos modelos apresentados outros também foram desenvolvidos, com menor sucesso, como tanques de óleo ou água utilizados para atrasar o sinal.

3.1.2 Reverberadores eletrônicos

A primeira proposta de um reverberador totalmente eletrônico, e mais especificamente digital foi feita por (SCHROEDER, 1961), e a partir de então várias arquiteturas foram desenvolvidas, sendo a grande maioria baseada em eletrônica digital ou em *software*, sendo as mais importantes apresentadas na Seção 3.3.

Uma abordagem alternativa para a reverberação, baseada em eletrônica analógica surgiu a partir de 1968, com o dispositivo *bucket-brigade device* (BBD), ou brigada de baldes, desenvolvido pelo engenheiro da Philips J. Sangster (RAFFEL; SMITH, 2010). Tal dispositivo é composto por uma grande quantidade de elementos capacitivos chaveados, que permitem a amostragem do sinal baseado em um *clock*. A amostra é então passada sequencialmente por

[1] Original disponível em: <http://www.accutronicsreverb.com/design/default/images/sub_01/ing_03.jpg>. Acesso em 30/10/2013.

cada um dos elementos a cada ciclo de *clock*, gerando assim um atraso no sinal de forma análoga à uma brigada de baldes para transporte de água. Essa linha de atraso analógica, associada a diversos circuitos de realimentação e filtros permite a implementação de estruturas de reverberação como a de Schroeder em um circuito analógico, que obtiveram sucesso na época já que eram mais baratos que as implementações em sistemas computacionais. Como é notável, essa tendência foi superada, tendo os dispositivos BBD perdido o mercado, sendo atualmente utilizado apenas por entusiastas de dispositivos analógicos.

3.2 Objetivos da reverberação artificial

Axon, Gilford e Shorter (1955) definem as características esperadas na reverberação natural em termos de frequência e tempo:

- Resposta em função do tempo: sons gerados no ambiente são refletidos constantemente, e a cada reflexão são atenuados. Em ambientes com características naturais a taxa de decaimento se assemelha a um decaimento exponencial. Além disso, mesmo para entradas impulsivas após o período inicial o ouvinte não deve ser capaz de identificar as reflexões individuais.
- Resposta em função da frequência: como em certas frequências as reflexões de pontos diferentes chegam ao ouvinte em momentos iguais, tais sons se somam, sendo então reforçados, sendo tais frequências chamadas de "modos normais" ou "frequências de ressonância". Os modos mais altos apresentam um espaçamento mútuo curto, onde não é possível haver distinção entre eles. Já nos modos mais baixos, a distância é suficiente para que o som seja percebido como desagradável, pelo efeito de filtro *comb*, ou pente, onde há alguns picos de frequência ressonante. Esse efeito desagradável ocorre principalmente em ambientes pequenos e deve ser evitado.

Já Schroeder e Logan (1961) apresentam seis condições que devem ser satisfeitas para a reverberação artificial:

1. A resposta em frequência deve ser plana quando medida com bandas estreitas de ruído, onde essa largura corresponde ao transiente do som a ser reverberado. Essa condição é satisfeita por estruturas que têm uma resposta plana mesmo para excitação senoidal;
2. Os modos normais do reverberador devem se sobrepor e cobrir toda banda de frequência de áudio;
3. Os tempos de reverberação dos modos individuais devem ser iguais ou aproximadamente iguais, de forma que componentes distintos de frequência decaiam com taxas iguais;

4. A densidade de reflexões em um curto intervalo após a excitação deve ser alta o suficiente para que não seja possível identificar as reflexões individuais;

5. A resposta da reverberação deve ser livre de periodicidades;

6. Por fim, a resposta em frequência deve ser livre de picos periódicos, ou respostas em formato de pente. Tal comportamento se traduz em sons ocos e metálicos, como se o som estivesse se propagando dentro de um duto.

Os autores afirmam que mesmo que uma boa parte dos parâmetros possa ser definida, não é possível precisar de outra maneira, que não sensitivamente, a qualidade da reverberação, sendo mais tarde reforçada essa afirmação por Jot e Chaigne (1991).

Portanto as técnicas de reverberação artificial buscam atender inicialmente características básicas da acústica de ambientes fechados, para que então sejam adicionadas nuances que se mostrem como agradáveis para os autores e para os envolvidos em testes sensoriais.

3.3 Estruturas digitais para reverberação artificial

A primeira estrutura digital para reverberação artificial foi proposta por Schroeder (1961), onde o autor buscava solucionar os problemas relacionados às técnicas aplicadas em sua época, especialmente o fato da resposta em frequência não ser plana e a densidade de reflexões ser muito baixa. A Seção 3.4.1 apresenta em detalhes tal estrutura, que será utilizada neste trabalho pelo seu menor uso de memória, devido às limitações do dispositivo FPGA utilizado, e por atender diretamente os pontos necessários para uma reverberação artificial adequada.

Vários outras estruturas foram propostos por autores como Moorer (1979), Stautner e Puckette (1982), Smith (1985), Gardner (1992), Jot e Chaigne (1991) e Datorro (1997). Cada um desses algoritmos apresenta características sonoras e níveis de complexidade distintos, sendo, como afirma Datorro (1997), a escolha da estrutura de reverberação uma questão de gosto por ser uma análise muito subjetiva.

3.4 Estrutura de reverberação de Schroeder

3.4.1 Atraso simples

A estrutura proposta por Schroeder (1961) é baseada em atrasos. A resposta ao impulso de atraso básico pode ser representado por:

$$H(t) = \delta(t - \tau) \tag{3.1}$$

Onde τ representa o atraso ao qual o sinal está submetido. A resposta em frequência desse atraso pode ser obtida através da transformada de Fourier e se o impulso aplicado for ideal, temos:

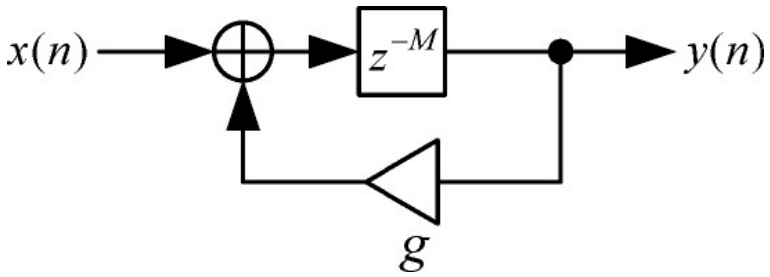
$$H(\omega) = e^{-j\omega\tau} \tag{3.2}$$

Onde $\omega = 2\pi f$, que é a frequência angular. O valor absoluto de 3.2 é unitário, ou seja, a resposta em frequência do atraso simples é plana, sem nenhuma alteração nas frequências do sinal original.

3.4.2 Filtro *Comb*

Partindo do atraso simples para que sejam obtidas mais repetições é necessário que sejam utilizados vários atrasos ou que um atraso seja realimentado. Obviamente a segunda alternativa é mais viável, dada a quantidade de memória necessária para a implementação de vários atrasos independentes. A Figura 7 apresenta o diagrama de blocos desse arranjo que é conhecido como **Filtro *Comb***, devido a forma de sua resposta em frequência, apresentada no decorrer desta seção.

Figura 7 – Atraso realimentado: Filtro *Comb*



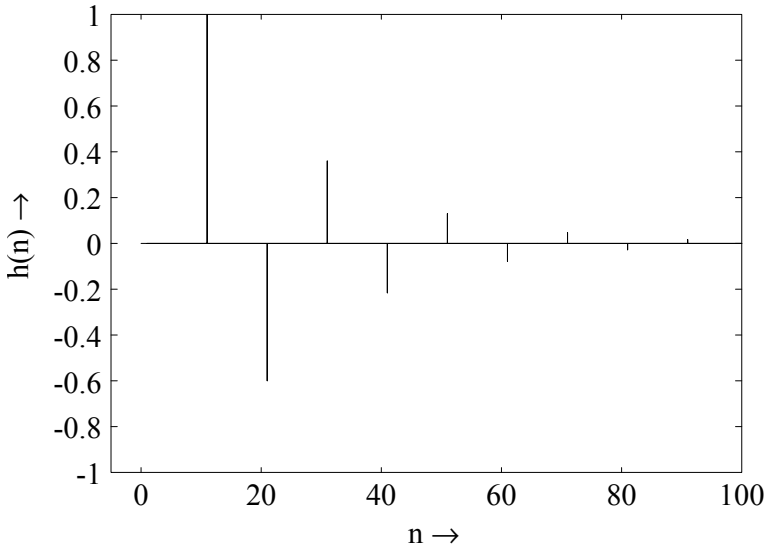
Fonte: Valimaki et al. (2012)

Na Figura 8 temos a resposta ao impulso do sistema, em forma de pulsos discretos com amplitudes que apresentam um decaimento exponencial. Essa sequência de impulsos está representada na Equação 3.3 seguida de sua resposta em frequência, na Equação 3.4.

$$h(t) = \delta(t-\tau) + g\delta(t-2\tau) + g^2\delta(t-3\tau) + \ldots \tag{3.3}$$

$$H(\omega) = e^{-j\omega\tau} + ge^{-2j\omega\tau} + g^2e^{-3j\omega\tau} + \ldots \tag{3.4}$$

Figura 8 – Resposta ao impulso: Filtro *Comb*



Fonte: Zölzer (1997)

A Equação 3.4 pode ser reescrita a partir da fórmula da soma de uma série geométrica:

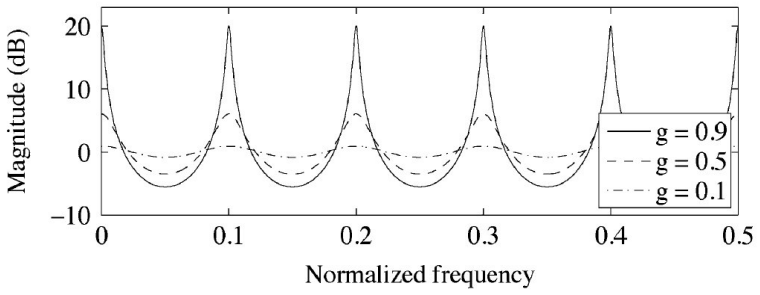
$$H(\omega) = \frac{e^{-j\omega\tau}}{1 - ge^{-j\omega\tau}} \tag{3.5}$$

Por fim a Equação 3.6 apresenta o módulo de $H(\omega)$:

$$|H(\omega)| = \frac{1}{\sqrt{1 + g^2 - 2gcos(\omega\tau)}} \tag{3.6}$$

Como é possível notar, o módulo da resposta do filtro não é mais independente da frequência, tendo na realidade uma característica periódica, com picos em certas frequências, como apresenta a Figura 9. A denominação filtro *comb* ou pente é decorrente do formato da resposta.

Figura 9 – Resposta em frequência: Filtro *Comb*



Fonte: Valimaki et al. (2012)

Pode ser implementada por meio da equação de diferenças na forma direta I, apresentada a seguir (SMITH, 2010):

$$y[n] = x[n] + gy[n-M] \tag{3.7}$$

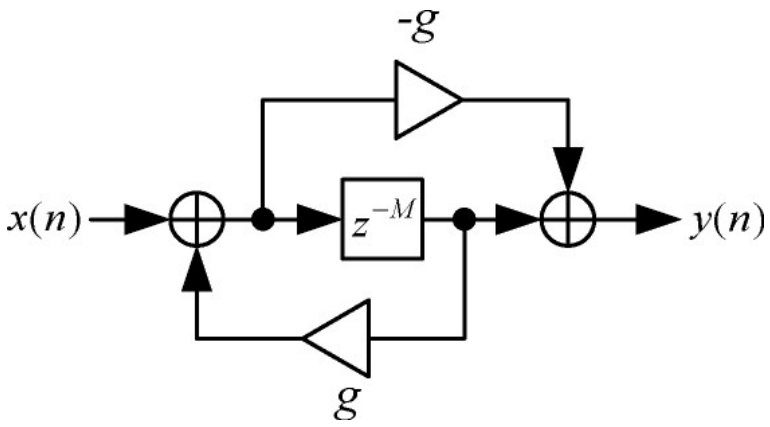
Onde $x[n]$ e $y[n]$ são respectivamente a entrada e a saída, g é o ganho e M o atraso.

3.4.3 Filtro *Allpass*

Schroeder (1961) buscando uma resposta de frequência plana introduziu uma malha *feedforward*, ou de alimentação direta, na arquitetura, conforme apresentado na Figura 10. Usando ganhos $-g$ para o sinal sem atraso e $1-g^2$ para o sinal com atraso a resposta do sistema é unitária, assim como ocorria no atraso simples. A resposta ao impulso é dada por:

$$h(t) = -g\delta(t) + (1-g^2)[\delta(t-\tau) + g\delta(t-2\tau) + \ldots] \tag{3.8}$$

Figura 10 – Filtro *Allpass*



Fonte: Valimaki et al. (2012)

A resposta em frequência pode ser expressada realizando-se a transformação de Fourier na Equação 3.9:

$$H(\omega) = e^{-j\omega\tau}\frac{1-ge^{j\omega\tau}}{1-ge^{-j\omega\tau}} \tag{3.9}$$

O autor aponta que o primeiro fator da Equação 3.9 apresenta valor absoluto unitário, e a fração à direita é o quociente de dois números complexos conjugados, logo também apresenta valor absoluto unitário. A análise dessa equação valida a proposta inicial do autor, uma estrutura sem alterações no espectro do sinal original. Tal estrutura é conhecida como filtro *allpass* ou "passa-tudo".

Pode ser implementada na forma direta I, de acordo com a Equação 3.10. A implementação na forma direta I apresenta a desvantagem de precisar de duas linhas de atraso indepen-

dentes, ocupando mais memória, porém a maior vantagem é não ser passível de estouro nos registradores internos quando implementada (SMITH, 2010).

$$y[n] = gx[n] + x[n-M] - gy[n-M] \tag{3.10}$$

3.4.4 Cálculo do tempo de reverberação

Baseado nos princípios de Sabine, Schroeder define a relação entre o ganho g e o atraso τ das suas estruturas e o tempo de reverberação T_{60} como:

$$T_{60} = \frac{60}{-20log|g|}\tau = \frac{3}{log|1/g|}\tau \tag{3.11}$$

Ainda há um déficit de densidade de reflexões na estrutura, que o autor soluciona com o uso de mais filtros em série, de forma que quantidade de reflexões por segundo seja próxima de 1000, que é o valor onde o autor considera que a reverberação soe natural (SCHROEDER, 1961). No artigo o autor consegue tal densidade com o uso de cinco filtros *allpass* em série, sendo o primeiro filtro com 0,1 segundos de atraso, e os filtros subsequentes com um terço do valor do filtro anterior. Os ganhos foram fixados em 0,7.

3.4.5 Estrutura final

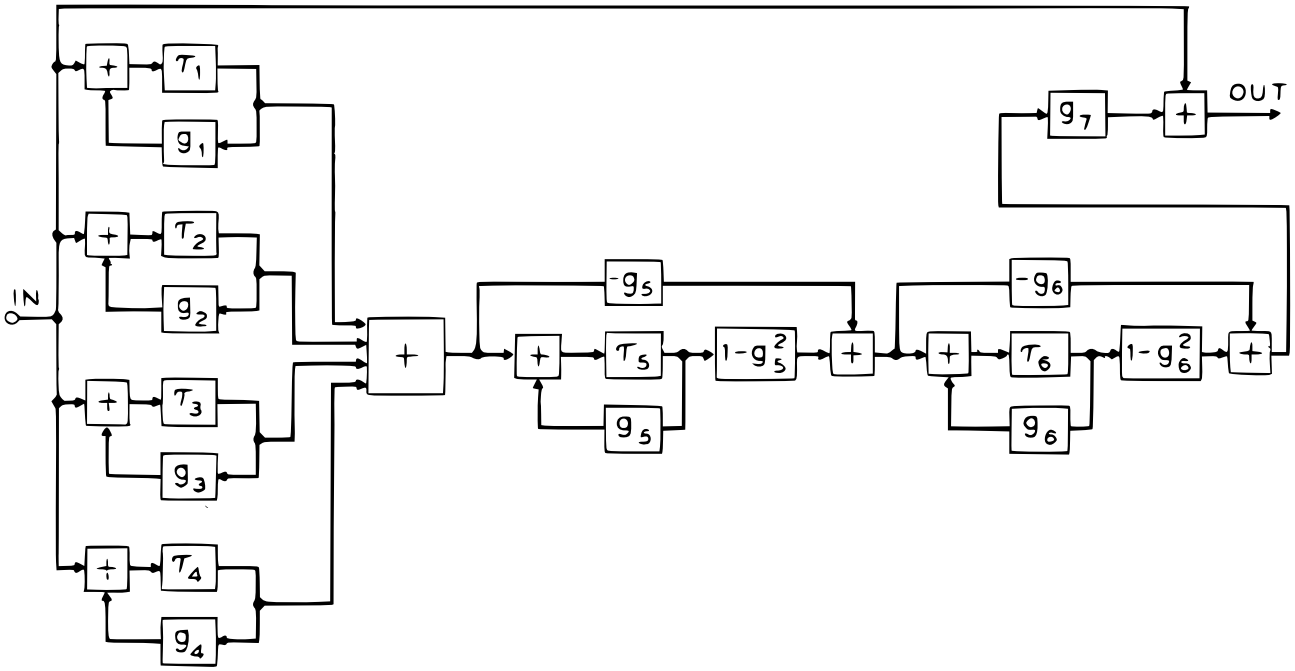
A estrutura final envolve o uso de filtros *comb* e *allpass*, com os parâmetros sendo definidos empiricamente através de testes sensoriais. O autor justifica o uso de filtros *comb* apesar do problema da resposta em frequência com o fato de ambientes reais também apresentarem esta característica. Porém se a densidade de picos for alta o suficiente o som não se mostra como desagradável. Tal densidade é obtida com o uso de quatro filtros *comb* em paralelo que então alimentam uma malha de mais dois filtros *allpass* em série, responsáveis por aumentar a densidade de reflexões. O autor faz a análise de que um filtro *comb* com um atraso de $40ms$ apresenta uma densidade de 25 reflexões por segundo, portanto, quatro unidades em paralelo apresentam uma densidade de 100. Os filtros *allpass* na sequência conseguem multiplicar por 3 essa densidade cada, para que no final a densidade seja próxima de 1000 reflexões por segundo.

Para maximizar a densidade e diminuir a possibilidade de reflexões periódicas os atrasos devem ser descorrelacionados. No seu artigo Schroeder usa atrasos para os filtros *comb* a uma razão de 1:1,5. Os ganhos são então calculados considerando o tempo de reverberação (T_{60}) esperado, a partir da Equação 3.11. Para os filtros *allpass* o autor usa atrasos de $5ms$ e $1,7ms$ com os ganhos em 0,7.

Assim o reverberador completo de Schroeder, apresentado na Figura 11, consegue satisfazer os pontos apresentados na Seção 3.2 pela união dos dois tipos de filtro. Schroeder em seu artigo ainda propõe o uso de filtros na malha de realimentação caso haja necessidade de

simular a absorção diferenciada para diferentes frequências e afirma que testes auditivos extensivos foram realizados e que não foi possível distinguir o som de seu reverberador do som de ambientes reais (SCHROEDER, 1961).

Figura 11 – Reverberador proposto por Schroeder



Fonte: Schroeder (1961)

3.5 Trabalhos relacionados

Utilizando as bases de dados do *Institute of Electrical and Electronic Engineers* (IEEE), *Association for Computing Machinery* (ACM), *Audio Engineering Society* (AES), Periódicos CAPES e *Google Scholar* foi realizada uma pesquisa de trabalhos especificamente relacionados à implementação em FPGA de algoritmos de reverberação. A grande maioria de tais trabalhos apresenta a implementação dos algoritmos em DSP, como Mohammed e Bijoy (2011), que não utiliza algoritmos de reverberação em si, apenas alguns atrasos no áudio, já Wang, Yin e Chen (2008) apresenta em DSP uma implementação simplificada do algoritmo de reverberação de Gardner (1992), entre outros trabalhos.

Há trabalhos que envolvem a implementação de um *core* DSP em FPGA, como Byun et al. (2011), Byun et al. (2009) e Pfaff et al. (2007), envolvendo o *co-design* de *hardware* e *software*, uma técnica que visa a otimização dos sistemas (TEICH, 2012).

E, finalmente, implementações diretas para FPGA, que são exatamente o foco do presente trabalho, também foram realizadas por Catuna, Szopos e Topa (2010), Dornean, Topa e Kirei (2008), Bota et al. (2011) e Sehn (2009). Tais trabalhos são apresentados a seguir.

3.5.1 Dornean, Topa e Kirei (2008)

Este artigo apresenta implementações em *Verilog*, uma linguagem de descrição de *hardware*, para os algoritmos de Schroeder (1961), Moorer (1979) e Gardner (1992). Em paralelo também foi feita a implementação de tais algoritmos no ambiente *Simulink* do *Matlab*, que permite a composição de sistemas por meio de uma ferramenta gráfica. Por fim é feita uma comparação entre a simulação do *Matlab* e uma simulação utilizando o *Modelsim*, um ambiente de desenvolvimento e simulação de sistemas digitais. Tal comparação foi feita com base na resposta ao impulso dos sistemas, como é comum na área. O trabalho é focado na simulação, não sendo feita nenhuma consideração quanto ao uso de FPGAs reais.

3.5.2 Sehn (2009)

Em sua dissertação de mestrado, Sehn (2009) apresenta uma implementação em FPGA, utilizando o dispositivo EP2C35F672C6 da *Altera*, do algoritmo de Moorer (1979). O autor faz ainda uma contribuição ao algoritmo, empregando a técnica de processamento multitaxa como forma de reduzir o tamanho dos atrasos necessários para execução do algoritmo. Tal técnica se baseia na diminuição da taxa de amostragem antes do algoritmo principal, por meio da decimação, e ao final da estrutura é realizada a interpolação para trazer o sinal de volta para a mesma taxa de amostragem.

Sehn (2009) realizou a implementação do algoritmo original e de sua versão otimizada, obtendo uma redução de cerca de 50% no tamanho dos atrasos necessários. O documento não apresenta módulo A/D – D/A, e os testes, também com impulso foram realizados, onde a saída do algoritmo modificado é bem parecida com o original, apenas com um atraso de aproximadamente 320 ns, não sendo perceptível aos ouvidos humanos. Não fica claro, no entanto, se os resultados foram obtidos pela simulação funcional da estrutura ou utilizando algum circuito auxiliar externo para excitação e aquisição dos dados.

3.5.3 Catuna, Szopos e Topa (2010)

O artigo apresenta uma implementação em FPGA do algoritmo de Schroeder (1961) por meio do *software LabView*, que também é uma ferramenta gráfica que permite a descrição e simulação de sistemas digitais. O sistema foi avaliado por meio de ferramentas da própria *National Instruments*, empresa responsável pelo *LabView*, tendo o FPGA desempenhado o papel de apenas executar o algoritmo, sem controle de interfaces A/D – D/A. Os resultados, obtidos por meio de resposta ao impulso apresentam as formas de onda clássica esperadas no algoritmo de Schroeder.

3.5.4 Bota et al. (2011)

O último artigo relevante no tópico apresenta uma implementação completa em FPGA de um sistema de reverberação baseado no algoritmo de Schroeder (1961), inclusive com o controle de um CODEC também baseado no padrão AC'97. O sistema foi implementado no FPGA *Xilinx* XC5VLTX110T, da família *Virtex-5*, por meio do *System Generator*, uma ferramenta que utiliza o *Simulink* para descrição gráfica de arquiteturas digitais. São apresentados apenas os resultados de simulações, não havendo formas de onda para os sinais obtidos a partir do CODEC de áudio. Também, segundo os autores, não foi possível implementar a estrutura completa, por restrições de memória.

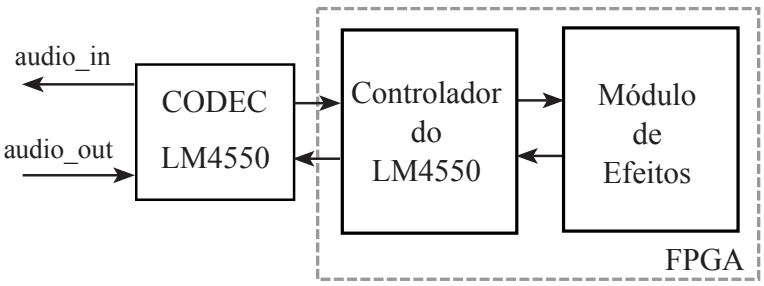
4 O Projeto

Neste capítulo será apresentado o procedimento utilizado para o projeto e implementação do reverberador de Schroeder em FPGA.

4.1 Visão geral

Baseado nos conceitos apresentados nos capítulos anteriores, o sistema objeto desse trabalho foi implementado em FPGA, sendo descrito na linguagem VHDL, como apresentado no Capítulo 1. A Figura 12 apresenta uma visão geral do sistema em questão.

Figura 12 – Diagrama de blocos do sistema



Fonte: Autoria Própria

O sistema é composto por três blocos básicos:

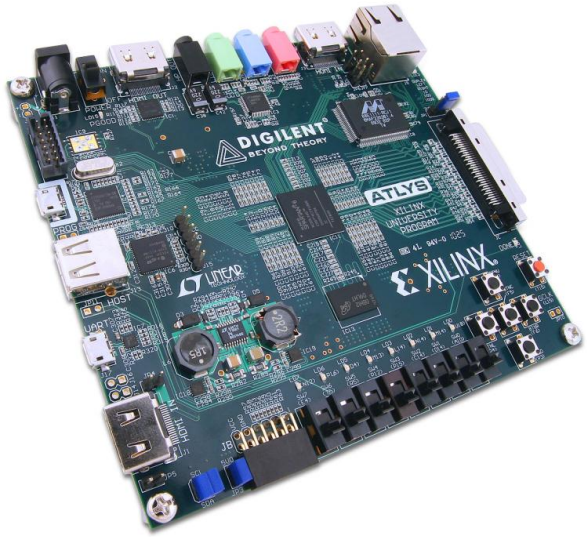
- CODEC de áudio LM4550;
- Controlador do CODEC de áudio;
- Módulo de efeitos;

O CODEC é responsável pela conversão entre o áudio analógico e o seu equivalente digital, além da adequação dos níveis de sinal de entrada e saída do áudio analógico. O bloco de controle do CODEC é responsável pela configuração dos parâmetros de aquisição de dados do CODEC, além de implementar seu protocolo de comunicação com o FPGA. E, finalmente, o módulo de efeitos é responsável por executar o algoritmo de reverberação proposto por Schroeder e Logan (1961) e apresentado no Capítulo 3.

Todo o sistema foi implementado em um *kit* de desenvolvimento produzido pela *Digilent Inc.* o *Atlys*, visto na Figura 13. O FPGA presente no *kit* é o *Xilinx Spartan 6* XC6SLX45 - CSG324C, que dispõe, entre outras características, de 2,1 Mbits de BRAM (*Block RAM*). A

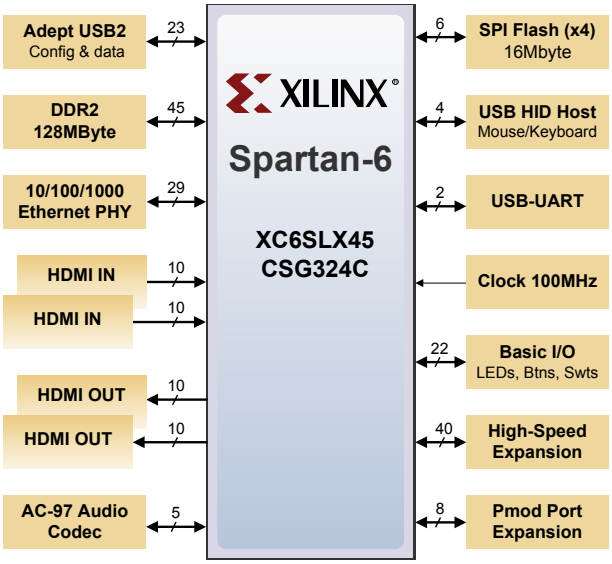
Figura 14 apresenta o diagrama dos periféricos conectados ao FPGA, onde o CODEC AC'97 utilizado neste projeto aparece.

Figura 13 – Digilent Atlys Board



Fonte: Digilent Incorporated (2012)

Figura 14 – Diagrama de componentes da Digilent Atlys



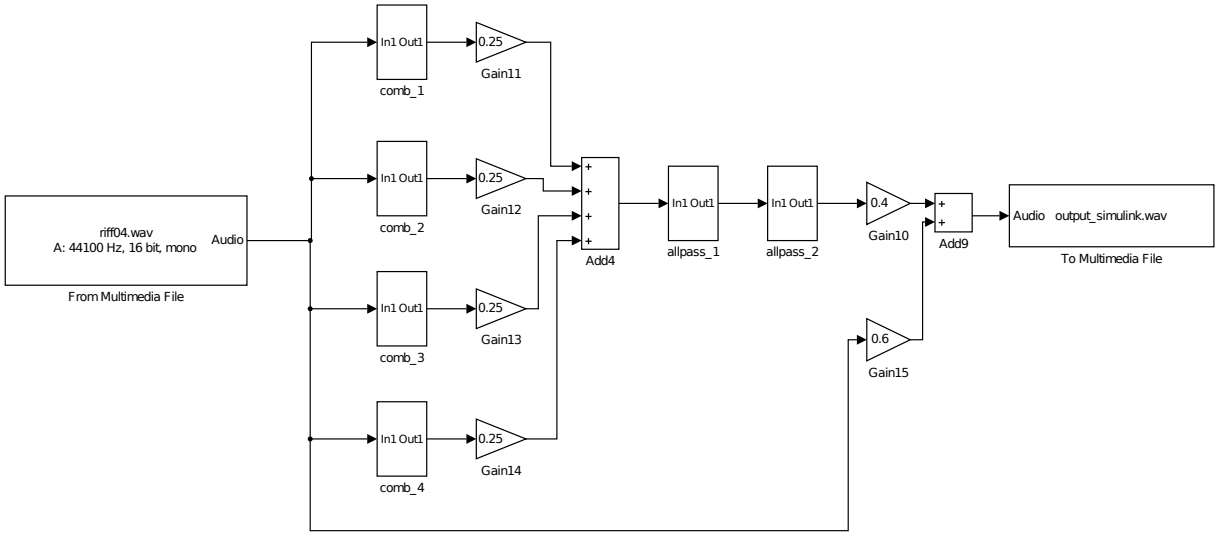
Fonte: Digilent Incorporated (2012)

4.1.1 Implementação em MATLAB

O desenvolvimento do sistema se iniciou em uma implementação em MATLAB, no ambiente Simulink, passo importante para melhor entendimento da estrutura e que permitiu testes rápidos já com áudio. A Figura 15 apresenta o sistema completo, baseado no sistema original

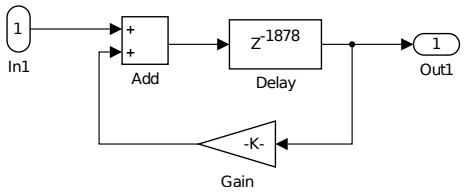
de Schroeder. Os blocos "`comb_n`" e "`allpass_n`" aparecem respectivamente nas Figuras 16 e 17.

Figura 15 – Reverberador de Schroder - Simulink



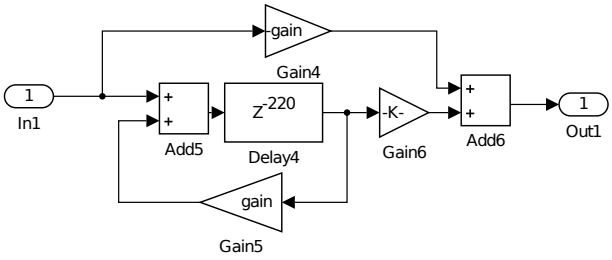
Fonte: Autoria própria.

Figura 16 – Bloco `comb_n` - Simulink



Fonte: Autoria própria.

Figura 17 – Bloco `allpass_n` - Simulink



Fonte: Autoria própria.

4.1.2 Parâmetros do reverberador

Os ganhos dos filtros *comb* foram definidos para um $T_{60} = 2s$ pela Equação 3.11. Os atrasos foram selecionados a partir dos valores do artigo original de Schroeder, mantendo-se a razão de 1:1,5 entre o maior e o menor. Para os filtros *allpass* os atrasos foram 5 e 1,7 *ms* e os ganhos fixados em 0,7, como sugerido já apresentado no Capítulo 3. A Tabela 3 apresenta os valores utilizados no projeto, não só na implementação em MATLAB, como em FPGA.

Quadro 3 – Parâmetros do projeto

	Comb 1	**Comb 2**	**Comb 3**	**Comb 4**	**Allpass 1**	**Allpass 2**
Ganho	0,8632	0,8740	0,8917	0,9016	0,7	0,7
Atraso	42,6 ms	39 ms	33,2 ms	30 ms	5ms	1,7ms

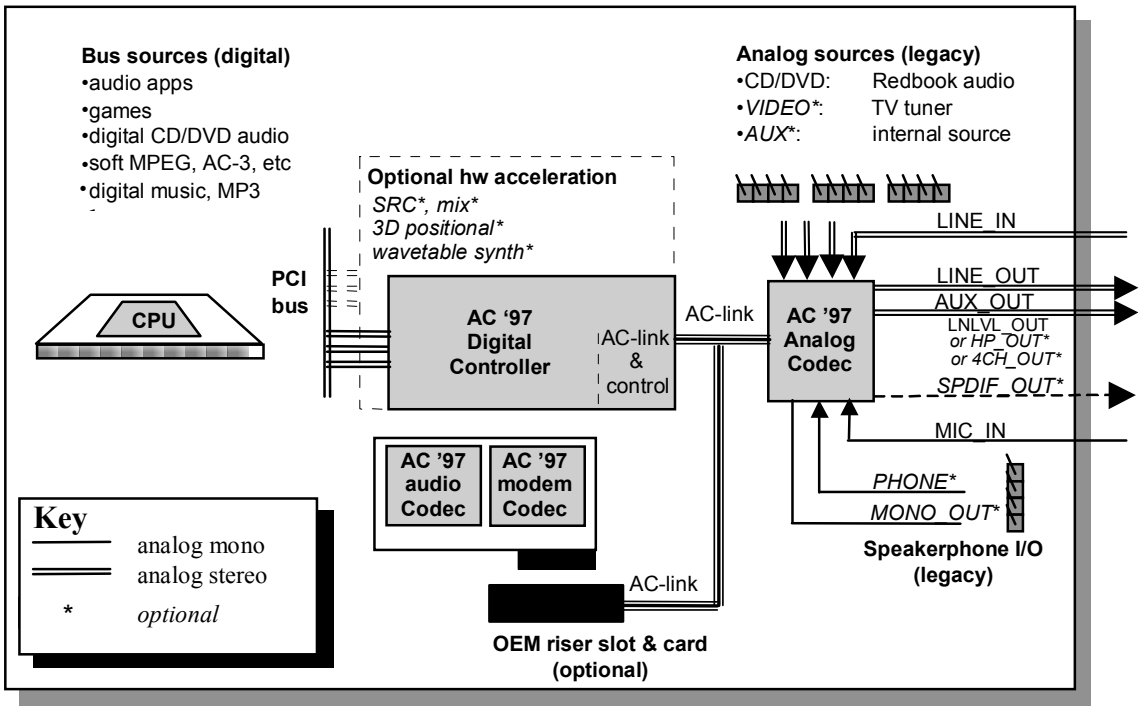
4.2 CODEC LM4550

Para executar as funções de conversão A/D – D/A foi utilizado o CODEC de áudio integrado LM4550, produzido pela *National Instruments* e compatível com o padrão Intel AC'97. Tal padrão foi desenvolvido pela Intel em 1997 como um CODEC de alta qualidade de áudio com foco em computadores pessoais (Intel Corporation, 2002). A popularização do padrão ocasionou uma grande disponibilidade de CIs compatíveis e consequente redução de preços, permitindo a expansão do seu uso também para dispositivos embarcados.

O LM4550 é capaz de trabalhar com 48 KHz de amostragem e 18 bits de resolução.

A Figura 18 apresenta o diagrama de blocos de um sistema que utilize o CODEC. O *hardware* disponível no *kit* de desenvolvimento utilizado implementa apenas o bloco *"Analog Codec"*, sendo então necessária a implementação no FPGA do bloco de controle do *Analog Codec* e da comunicação AC-Link.

Figura 18 – Diagrama do AC'97

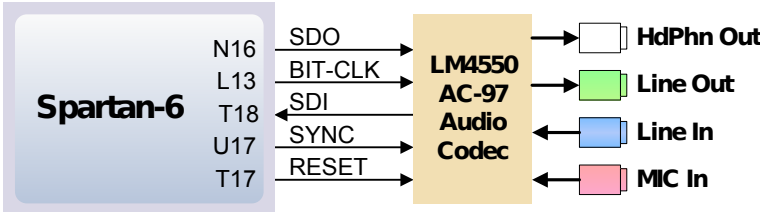


Fonte: Intel Corporation (2002)

A comunicação entre o dispositivo de controle, no caso o FPGA, e o CODEC é feita por meio de uma conexão serial com dois sinais de dados (SDI e SDO), um sinal de sincronismo (SYNC), *clock* (BIT-CLK) e *reset*, como apresentado na Figura 19 e descrito no Quadro 4.

A configuração dos parâmetros de operação é feita através de 27 registradores específicos.

Figura 19 – Conexões LM4550 - Spartan 6



Fonte: Digilent Incorporated (2012)

4.2.1 Interface *AC-Link*

Baseando-se no trabalho de Mayen (2011) e nos exemplos disponibilizados *online* por Javier Valcarce[1], foi possível desenvolver a interface de comunicação *AC-Link*, como requerida

[1] Disponível em: <http://www.javiervalcarce.eu/wiki/VHDL_Macro:_DC97>. Acesso em 28/10/2011.

Quadro 4 – Sinais de comunicação com o CODEC LM4550

Sinal	Pino	Direção(E/S)	Função
SDO	5	E	Entrada para o CODEC, recebe os sinais de controle e de áudio digital.
BIT-CLK	6	E/S	Atua como entrada quando no modo escravo, recebendo o *clock* do dispositivo mestre, e como saída no modo mestre, disponibilizando o *clock* de 12.288 kHz gerado pelo CODEC.
SDI	8	S	Saída do CODEC, retorna tanto sinais de controle, como áudio digitalizado das entradas.
SYNC	10	E	Sinal de sincronismo, utilizado para marcar o início de um bloco.
RESET	11	E	Utilizado para reiniciar o CODEC.

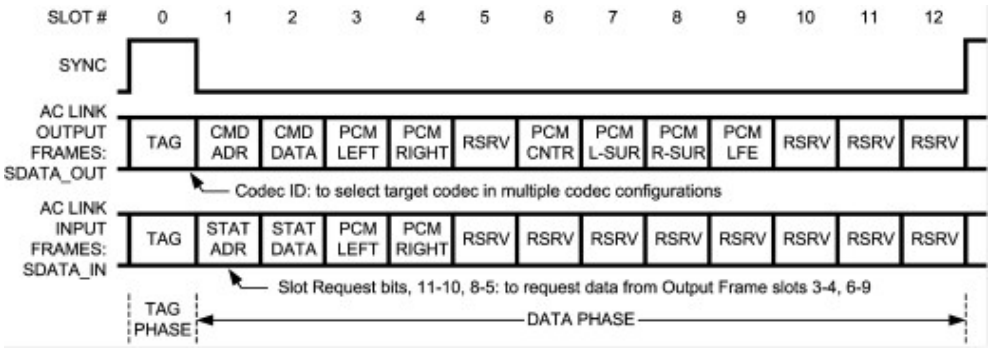
Fonte: Digilent Incorporated (2012)

pelo CODEC.

4.2.1.1 Protocolo da Interface

A interface *AC Link* utiliza 256 bits por *frame*, divididos em 13 *slots*, o primeiro de sinalização, com 16 bits e os 12 seguintes de dados, com 20 bits. Toda a comunicação é sincronizada por BIT-CLK, em 12.288 kHz, e o início dos *frames* é sinalizado pelo sinal SYNC que ocorre a uma taxa de 48 kHz, exatamente os 12.288 kHz de BIT-CLK divididos pelos 256 bits. O sinal SYNC é gerado pelo controlador da interface, e deve ser síncrono ao BIT-CLK, que é gerado pelo CODEC. A Figura 20 mostra o diagrama temporal de um *frame* completo, apresentando todos os *slots*.

Figura 20 – *Slots* de comunicação da interface *AC Link*



Fonte: National Semiconductor (2004)

Os Quadros 5 e 6 descrevem os *slots* de comunicação relevantes para o projeto.

Quadro 5 – *Slots* de comunicação: SDO

Slot	**Bits**	**Descrição**
0	0-15	Tag: Indica o ID do dispositivo, se o *frame* é válido, e quais *slots* contém áudio válido.
1	16-35	Endereço de comando: Indica o endereço do registrador e se a operação será de leitura ou escrita.
2	36-55	Corpo do comando.
3-4	56-95	Canais esquerdo e direito de áudio, respectivamente.
5-12	96-255	Não utilizados no projeto.

Fonte: Digilent Incorporated (2012)

Quadro 6 – *Slots* de comunicação: SDI

Slot	**Bits**	**Descrição**
0	0-15	Tag: Retorna o status do CODEC e a validade dos próximos *slots*.
1	16-35	Retorna informações sobre a existência de dados nos próximos *slots*.
2	36-55	Resultado da leitura de algum registro.
3 - 4	56-95	Canais esquerdo e direito de áudio, respectivamente.
5-12	96-255	Não utilizados no projeto.

Fonte: Digilent Incorporated (2012)

4.2.2 Bloco de controle

Para realizar a comunicação foi implementado um processo sensível a BIT-CLK, com um contador incrementado a cada borda de subida deste sinal. Esse contador é que mantém o controle da quantidade de bits já enviados ou recebidos, possibilitando assim a separação dos *frames*.

Ao primeiro pulso de BIT-CLK, o controlador gera um sinal alto em SYNC que se mantém por 16 ciclos, sinalizando assim o início do *frame*. Durante esses 16 primeiros ciclos, os 16 bits do *slot* 0 são transmitidos e recebidos concomitantemente, respectivamente via SDO e SDI. Nos 20 ciclos de BIT-CLK subsequentes, o *slot* 1 é transmitido via SDO e recebido em SDI, e assim sucessivamente até que todos os 255 bits do *frame* sejam transmitidos. Ao final do processo, um novo sinal SYNC é gerado e o processo inicializado para um novo *slot*.

Após o recebimento do bit 95, último do *slot* 4, em SDI o controlador gera um pulso de sinalização para o módulo de efeito, indicando que a amostra de áudio está pronta para uso. Tal pulso tem a duração de um ciclo de clock e ocorre no bit 96, logo após a finalização do *slot* 4. A listagem do código que executa essas funções pode ser vista no Apêndice A.

A configuração dos parâmetros do CODEC foi feita por meio da escrita nos registradores específicos. Para tanto foi implementada uma máquina de estados que escreve nos registradores de configuração continuamente, permitindo, por exemplo, a variação do ganho de entrada em tempo de execução. Há 27 registradores de configuração, e para o projeto em questão só foi necessário alterar os padrões de cinco deles, consequentemente em cinco estados. O Quadro 7 apresenta os sinais e registradores em questão. No Apêndice B temos a listagem do código que implementa essa máquina de estados.

Quadro 7 – Configuração do CODEC

Estado	**Registrador**	**Descrição**
0	0x04	"*Headphone Volume*" - Habilita a saída de fone de ouvido e configura sua atenuação para 0dB.
1	0x18	"*PCM Input Volume*" - Habilita a conversão de áudio e configura sua atenuação para 0dB.
2	0x1A	"*Record Select*" - Seleciona "*Line_in*" como entrada para o CODEC.
3	0x1C	"*Record Gain*" - Configura o ganho de gravação em 22.5dB.
4	0x20	"*General Purpose*" - Desabilita o áudio 3D, não necessário no projeto e configura um sinal de bypass do CODEC, utilizado para fins de comparação entre o áudio processado e o original.

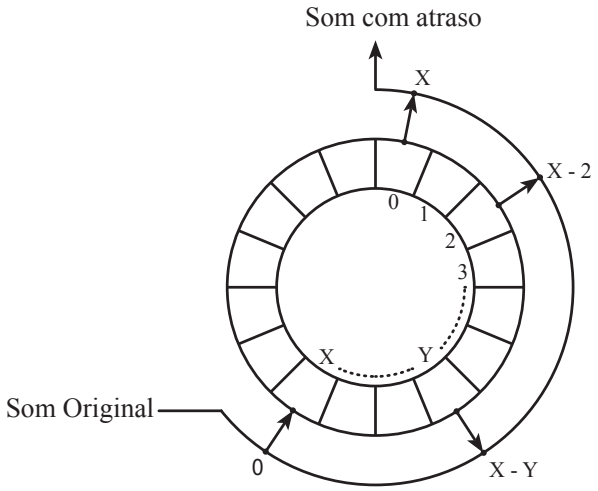
4.3 Módulo de Efeitos

O bloco de efeito é responsável por executar o algoritmo utilizado, no caso deste trabalho, o reverberador de Schroeder. O bloco tem como portas as linhas de entradas e saída de dados, o sinal de "*frame* pronto" do controlador do CODEC e um sinal de reset. Sua descrição em VHDL pode ser visualizada no Apêndice E. Os parâmetros de ganhos e atrasos utilizados para construção da estrutura são os mesmos utilizados na implementação em MATLAB do início do Capítulo atual, e foram baseados no artigo original de Schroeder, assim como apresentado no Capítulo 3.

4.3.1 Blocos de Atraso

Finalmente para a implementação em FPGA, o bloco básico seria o atraso, z^{-M}. Para tanto foi utilizada a estrutura de *buffer* circular. Tal estrutura é um tipo de FIFO (*First-In, First-Out*) onde a saída é conectada à entrada. Dessa forma é possível escrever dados em uma posição X e ler dados de momentos anteriores em posições diferentes, $X - M$. A figura 21 representa graficamente a estrutura.

Figura 21 – *Buffer* circular



Os dados entram na estrutura a uma taxa de $48KHz$, a frequência de amostragem (f_s) do CODEC, e portanto a conversão entre atraso em milissegundos e o número de posições Y é dada por:

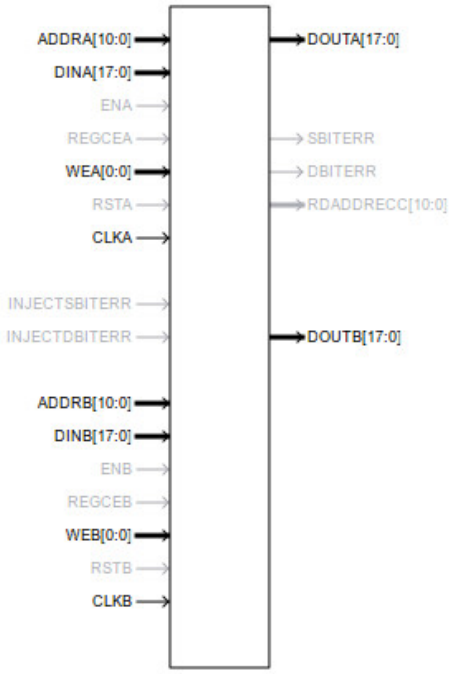
$$Atraso(s) = \frac{Y}{f_s} \tag{4.1}$$

O *buffer* circular foi implementado no projeto como uma DPRAM (*Dual-Port RAM*), uma memória com duas portas, que permite a leitura e a escrita concomitantemente. O componente foi gerado a partir de um núcleo disponibilizado pela *Xilinx* no seu *CORE Generator*. Na Figura 22 é possível ver as portas utilizadas em preto. O Quadro 8 descreve a funcionalidade de cada uma das portas no projeto. Para controlar os endereços do *buffer*, que devem ser cíclicos, foi implementado um contador simples, também utilizando o *Xilinx CORE Generator*. Sua entrada de clock é conectada ao sinal de "*frame* pronto" gerado pelo controlador do CODEC, e sua saída conectada a entrada *addra* do *buffer*. Dessa forma, cada vez que houver um novo *frame* de amostras prontos, os valores são atualizados.

4.3.2 Filtro *Comb*

Na implementação do filtro *comb* foi utilizada a equação de diferenças descrita em 3.7. O filtro usa como componentes um bloco de atraso como descrito na subseção anterior, com 2048 posições, possibilitando um atraso de $42,6$ ms, que é o atraso máximo utilizado na estrutura. Todas as operações são realizadas em ponto fixo sinalizado, numa representação Q1.17, totalizando 18 bits. Tal representação resulta em uma uma resolução de $1/2^{17} = 7,62 \times 10^{-6}$. O tamanho total de memória para cada bloco é de 2048×18 bits, num total de 36,864 Kbits. A implementação é bem direta e é apresentada no Apêndice F.

Figura 22 – Bloco de DPRAM usado como *buffer* circular



Fonte: Xilinx CORE Generator.

Quadro 8 – Portas do *buffer* circular

Porta	Descrição
clka	Clock.
wea	Habilitação de escrita em A, habilitado.
addra	Endereço de A, recebe o endereço da posição atual.
dina	Entrada de dados A, recebe a amostra de áudio atual.
douta	Saída de dados A, não utilizada.
clkb	Clock, o mesmo de clka.
web	Habilitação de escrita em B, desabilitado, já que B é utilizado apenas para leitura.
addrb	Endereço de B, recebe o endereço da posição com atraso que se deseja ler.
dinb	Entrada de dados B, não utlizada.
doutb	Saída de dados B, retorna o valor da amostra atrasada.

Fonte: Autoria própria.

4.3.3 Filtro *Allpass*

Já o filtro *allpass* usa dois blocos de atraso por ter sido implementado na forma direta I, como apresentado na Equação 3.10. Os blocos de atraso são diferenciados por serem bem menores que os dos filtros *comb*, dado o atraso menor necessário, em um total de 256 posições em cada bloco. As questões de precisão são as mesmas do filtro *comb*, logo a largura da memória é 18 bits, perfazendo um total de 4,608 Kbits nos dois blocos. A listagem está disponível no Apêndice G.

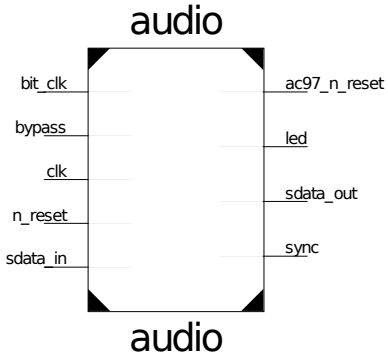
4.3.4 Estrutura Final

A estrutura final é a mesma da Figura 11, com quatro filtros *comb* em paralelo alimentando dois filtros *comb* em série. O total de memória utilizada nos atrasos é de 156,672 Kbits, cerca de 7% do total de memória disponível no FPGA, 2,1 Mbits.

4.4 Entidade Geral (*Top-level*)

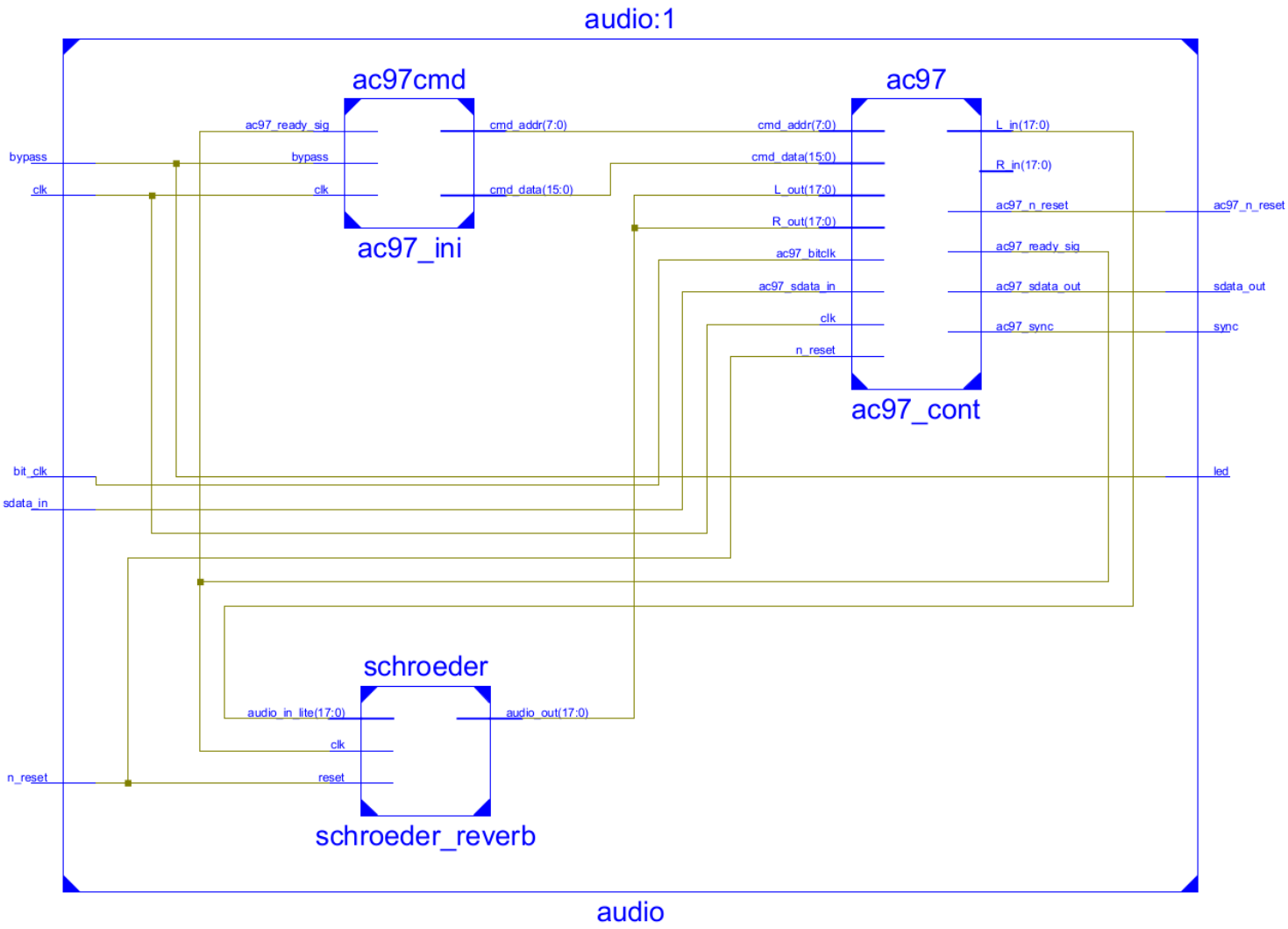
A entidade *top-level* é a junção do bloco de efeito, bloco de configuração do CODEC e bloco de controle. Está transcrita no Apêndice D. Sua representação simbólica, exibindo as entradas e saídas, pode ser vista na Figura 23. Já a Figura 24 é a visão interna de suas interconexões. Como a quantidade de bits da representação numérica do sistema é grande e por lidar com aritmética de ponto fixo muitos barramentos e componentes intermediários foram gerados no momento da síntese, tornando o esquemático dos blocos mais internos do sistema muito grandes para serem representados neste trabalho.

Figura 23 – Entidade *top-level* do projeto



Fonte: Xilinx Schematic View.

Figura 24 – Entidade *top-level* do projeto, visão interna



Fonte: Xilinx Schematic View.

5 Resultados

Para a avaliação do funcionamento do sistema vários testes foram realizados com seus diversos blocos, tanto a nível de simulação como testes práticos.

5.1 CODEC

O primeiro bloco a ser validado foi o de comunicação com o CODEC a partir de simulações e finalmente a verificação de seu funcionamento com áudio.

Os sinais do protocolo de comunicação foram avaliados por meio de simulação utilizando o *software ISim* da própria *Xilinx*. Para gerar os estímulos de entrada para simulação foi escrito um *testbench* em VHDL, presente no Apêndice C. O gráfico foi gerado no *GTKWave* a partir de um arquivo VCD (*Value Change Dump*) gerado pelo *ISim*, por permitir uma maior flexibilidade nas configurações de visualização das formas de onda.

A Figura 25 apresenta os primeiros *slots* do primeiro quadro da comunicação. É possível perceber que SYNC se mantém alto por 16 pulsos de BIT-CLK, correspondentes ao primeiro *slot* e durante esse período é transmitida a palavra "1111100000000000", onde os cinco primeiros bits em nível alto indicam ao CODEC, respectivamente, que o quadro é válido, que há um endereço válido no frame de endereço de comando, que há um comando válido em seu respectivo *slot* e que há dados válidos de áudio para os canais esquerdo e direito (National Semiconductor, 2004).

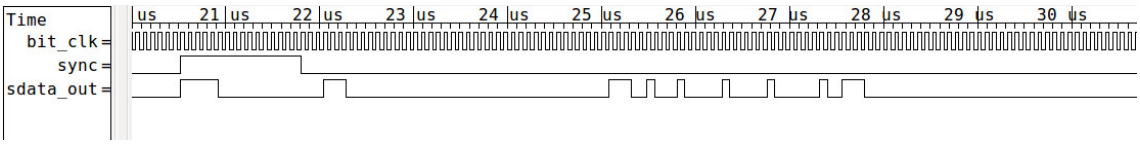
Os vinte bits que sucedem a descida do sinal SYNC formam o *slot* 1 do quadro, e transmite o endereço do comando de controle. No caso é transmitida a palavra "00011100000000000000", onde os oito primeiros bits são o endereço *0x1C*, utilizado pelo CODEC para configuração do ganho de gravação.

Continuando no sinal, os vinte bits seguintes formam o *slot* 2 do quadro, que traz o dado que deve ser escrito no registrador indicado pelo endereço do quadro anterior. No caso a palavra é "00000000000000000000".

O próximos dois *slots* de vinte bits são os canais esquerdo e direito de áudio, respectivamente, e as palavras são "01110010001000010"e "00100000100111000", valores utilizados para teste apenas e que constam no *testbench* do Apêndice C.

Os demais bits que completam os 256 bits do quadro não são utilizados nessa aplicação e são mantidos em nível baixo. Na sequência há a repetição do processo com a escrita dos próximos quadros.

Figura 25 – Quadro 1 da comunicação



Fonte: Autoria própria.

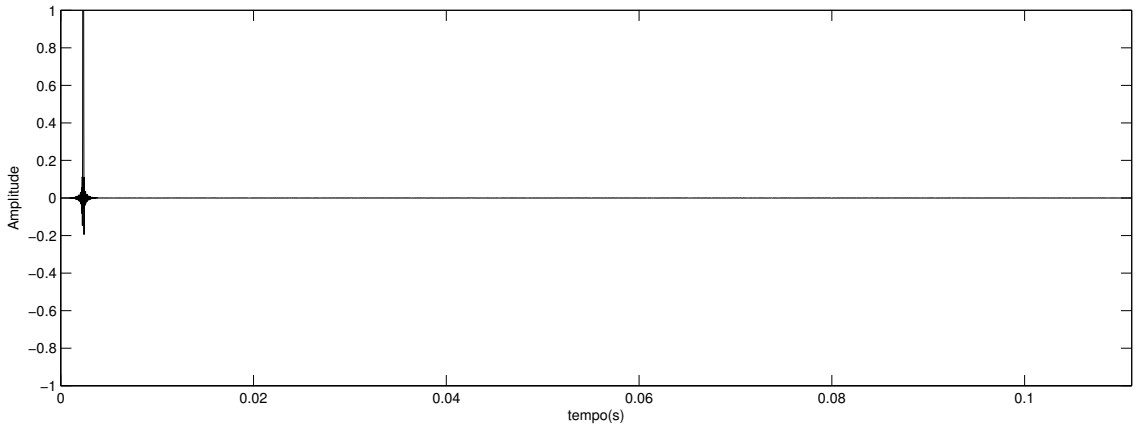
5.2 Estruturas de reverberação

A avaliação do funcionamento das estruturas de reverberação foi feita a partir da medição das respostas ao impulso das mesmas, em busca das características apresentadas nos capítulos anteriores. A resposta ao impulso é uma boa técnica para tal por conseguir descrever o sistema, e pela simplicidade de obtenção (GARDNER, 2002). A estrutura utilizada como referência foi a implementada em MATLAB.

5.3 *Setup* de testes

O sinal de entrada utilizado foi similar a um impulso unitário e é apresentado na Figura 26.

Figura 26 – Sinal de entrada para testes



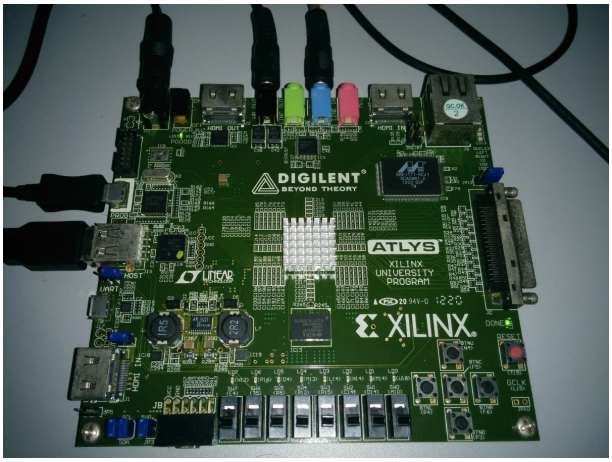
Fonte: Autoria própria.

No MATLAB foram utilizados blocos de entrada e saída de arquivos de áudio do tipo WAV, com taxa de amostragem de 48 KHz e 16 bits de resolução. O sinais foram aplicados e recuperados da estrutura apresentada na Figura 15.

Para os testes no FPGA, os sinais de áudio foram enviados pela saída de áudio de um computador pessoal até a entrada do *kit* de desenvolvimento, de forma analógica. O áudio da saída do sistema foi gravado no *software* Audacity a uma taxa de 48 KHz, 16 bits, através da entrada de microfone do computador. Na Figura 27 é possível perceber os cabos de áudio

conectados no topo da *Atlys*, sendo o conector preto o de saída e o azul de entrada. Os dados adquiridos no Audacity foram então importados no MATLAB para geração dos gráficos em melhor qualidade.

Figura 27 – Atlys e conexões de áudio

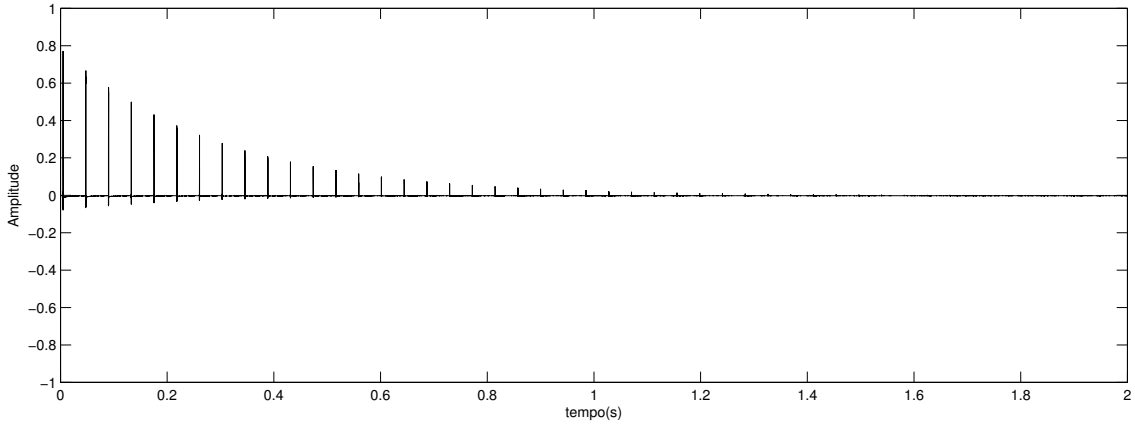


Fonte: Autoria própria.

5.4 Formas de onda resultantes

A estrutura de reverberação foi avaliada incrementalmente, com cada um de seus blocos e algumas combinações que pudessem demonstrar o efeito de cada uma no áudio. A partir do bloco mais simples, o filtro *comb*, é possível perceber a repetição do sinal de entrada e seu decaimento, como esperado, na Figura 28. Como também já previsto, a densidade de reflexões se mantém fixa e é muito baixa. O tempo de atraso utilizado foi de $42,6ms$ com um ganho de $0,8632$.

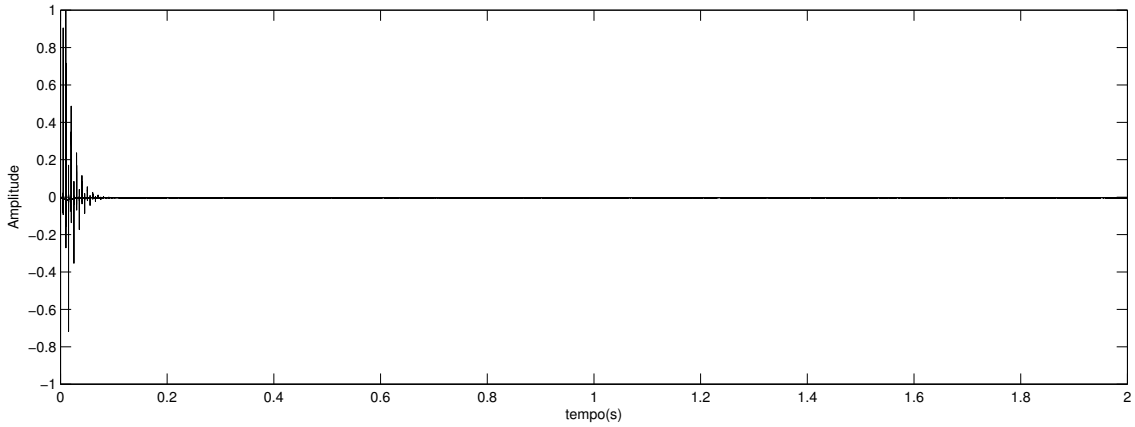
Figura 28 – Resposta ao impulso do sistema com apenas um filtro *comb*



Fonte: Autoria própria.

A próxima estrutura avaliada é o filtro *allpass*, na Figura 29, com um atraso de $5ms$ e um ganho de $0,7$. Sua resposta ao impulso é semelhante ao do filtro *comb* exceto pela inversão do sinal a cada repetição. Como também esperado sua densidade de reflexões é baixa.

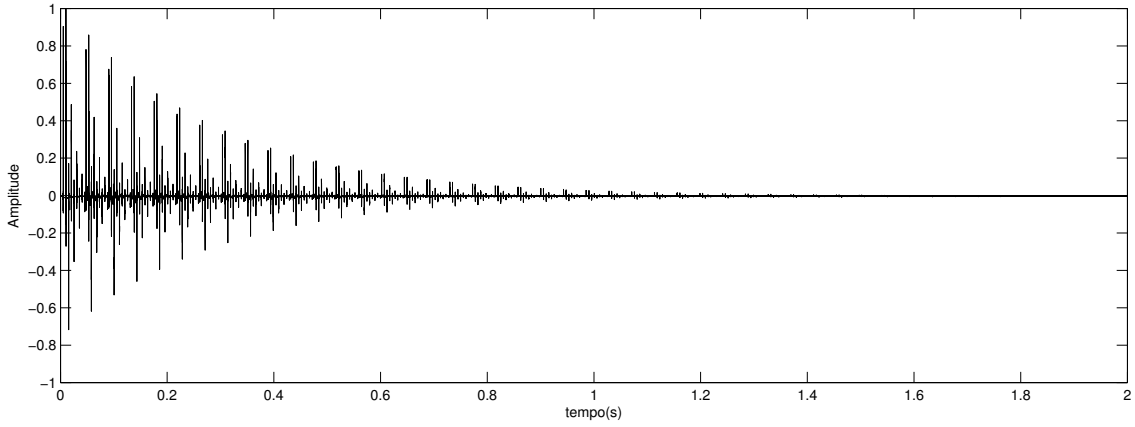
Figura 29 – Resposta ao impulso do sistema com apenas um filtro *allpass*



Fonte: Autoria própria.

Partindo para a combinação dos filtros, utilizando um filtro *comb* em série com um filtro *allpass* a densidade de reflexões já se mostra mais alta, vide Figura 30, porém ainda há uma característica periódica muito forte.

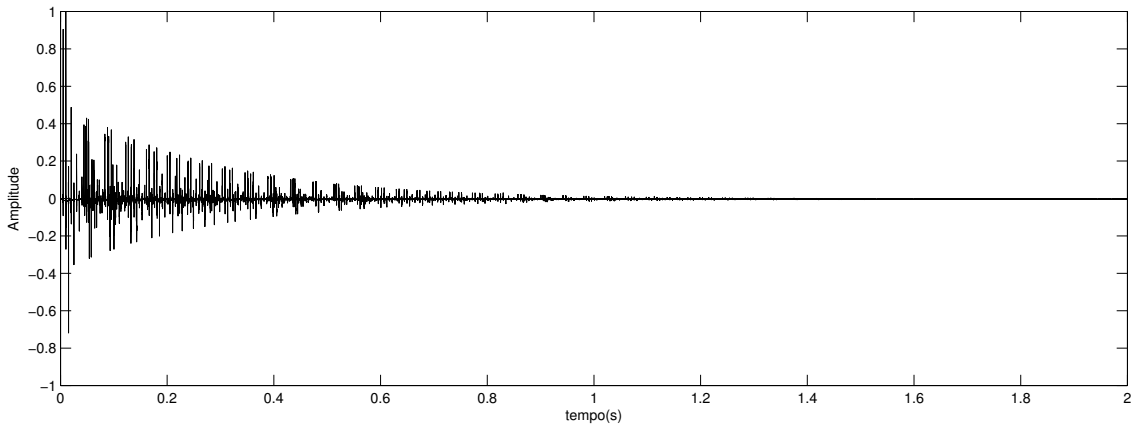
Figura 30 – Resposta ao impulso do sistema com um filtro *comb* e um *allpass*



Fonte: Autoria própria.

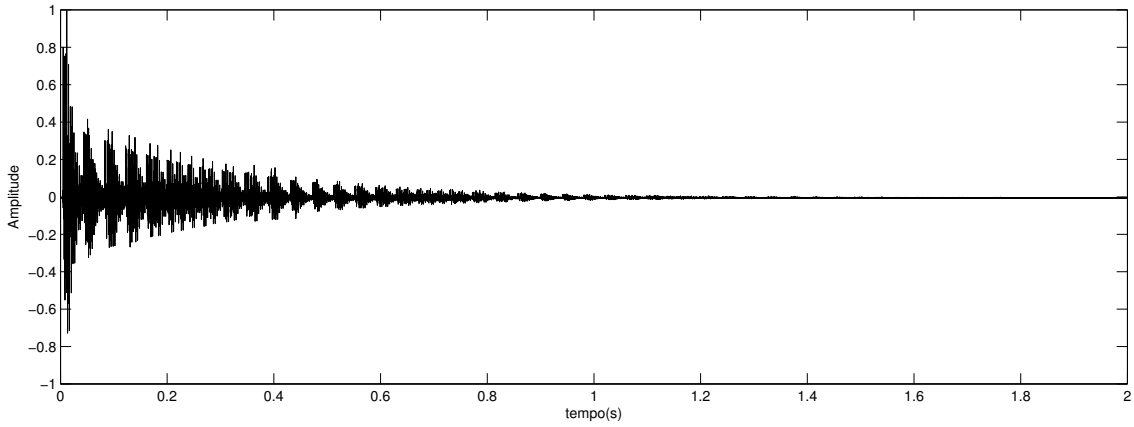
A Figura 31 é o resultado do sistema com dois filtros *comb* em paralelo e um filtro *allpass* em série. Já a Figura 32 é a combinação de dois filtros *comb* em paralelo e dois filtros *allpass* em série. É possível perceber o grande aumento da densidade de reflexões se comparadas com as estruturas anteriores, efeitos já previsíveis e apresentados nos Capítulos anteriores deste trabalho. Os tempos e ganhos dos filtros são os mesmos apresentados na Tabela 3.

Figura 31 – Resposta ao impulso do sistema com dois filtros *comb* e um *allpass*



Fonte: Autoria própria.

Figura 32 – Resposta ao impulso do sistema com dois filtros *comb* e dois *allpass*



Fonte: Autoria própria.

Finalmente as figuras a seguir apresentam as formas de onda da resposta ao impulso do sistema completo, como proposto por Schroeder, tanto na implementação em MATLAB como na implementação no FPGA.

Na Figura 33, resultante da execução do algoritmo no MATLAB, é possível perceber o claro efeito da reverberação com o prolongamento do sinal, se comparado ao impulso original de entrada da Figura 26. As características esperadas da reverberação estão presentes, com a boa densidade de reflexões e o decaimento exponencial. Se comparada às arquiteturas anteriores a característica periódica do sinal é menos aparente. A Figura 34 apresenta a resposta em fase do sistema.

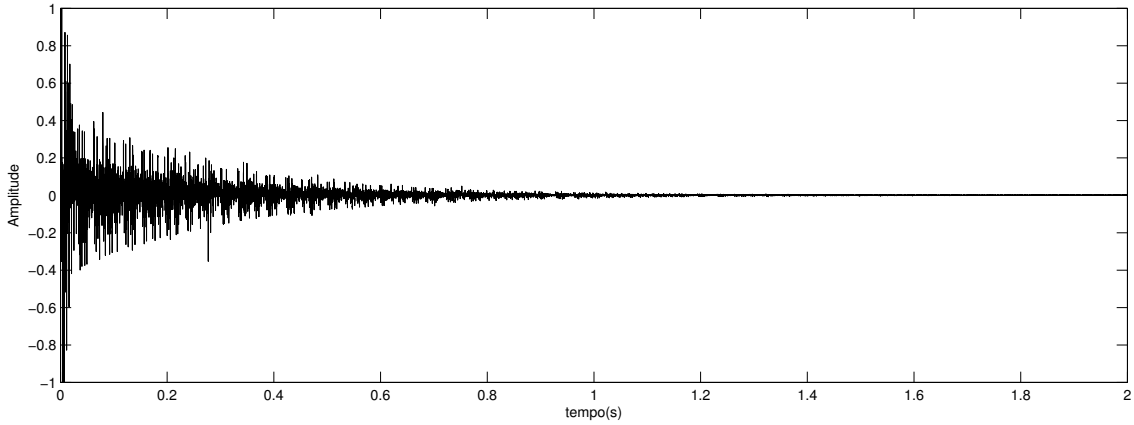
A Figura 35 apresenta a saída do reverberador implementado em FPGA. Tal figura confirma o funcionamento do sistema, sendo bem semelhante ao resultado do MATLAB. O processo de gravação de áudio, transmissão analógica por meio de cabos e a dupla conversão A/D - D/A sofrida pelo sinal acaba resultando no ruído que era inexistente no caso da simulação,

além da alteração da resposta em fase, apresentada na Figura 36.

Para melhor visualização de ambos os sinais, a Figura 37 as respostas ao impulso sobrepostas e a Figura **??** as respostas em fase sobrepostas. Obviamente há algumas diferenças, provavelmente decorrentes do uso de ponto fixo nos cálculos em FPGA e do processo de gravação do áudio, sofrendo dupla conversão A/D - D/A, porém a tendência geral é mantida em ambos os casos.

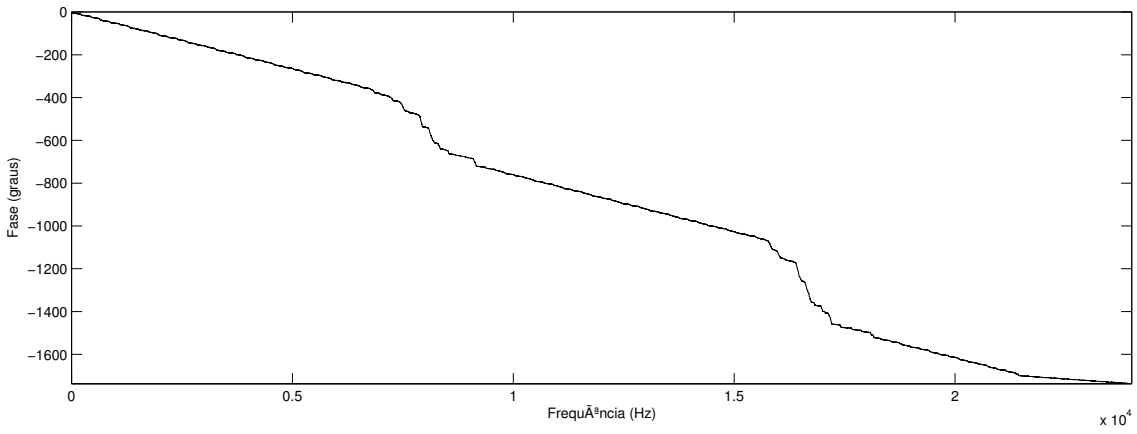
Convém notar o aparente decaimento total do sinal antes do tempo $T_{60} = 2$, como apresentado no Capítulo 4. O tempo T_{60} é definido como o decaimento de 60 dB no sinal, em termos de tensão, $0,001$, um valor muito pequeno para visualização no tipo de gráfico apresentado.

Figura 33 – Resposta ao impulso da estrutura em MATLAB



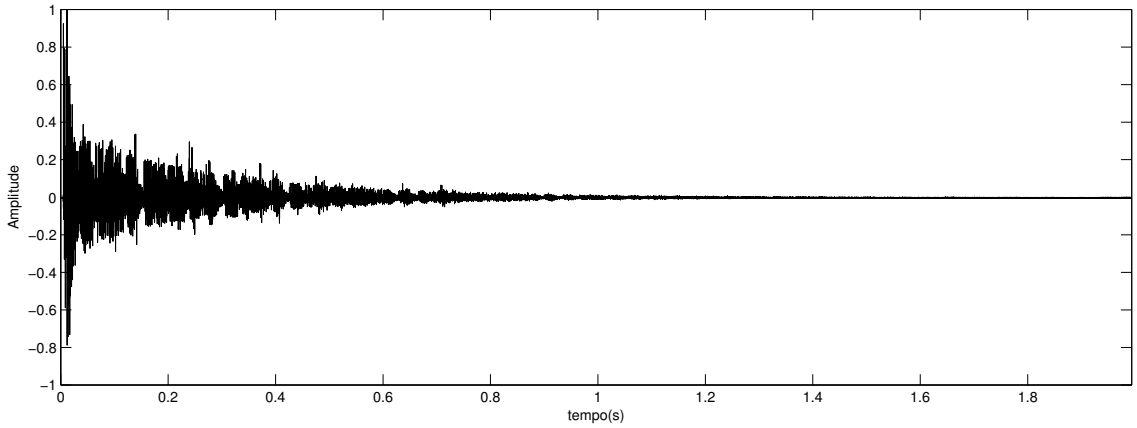
Fonte: Autoria própria.

Figura 34 – Resposta em fase da estrutura em MATLAB



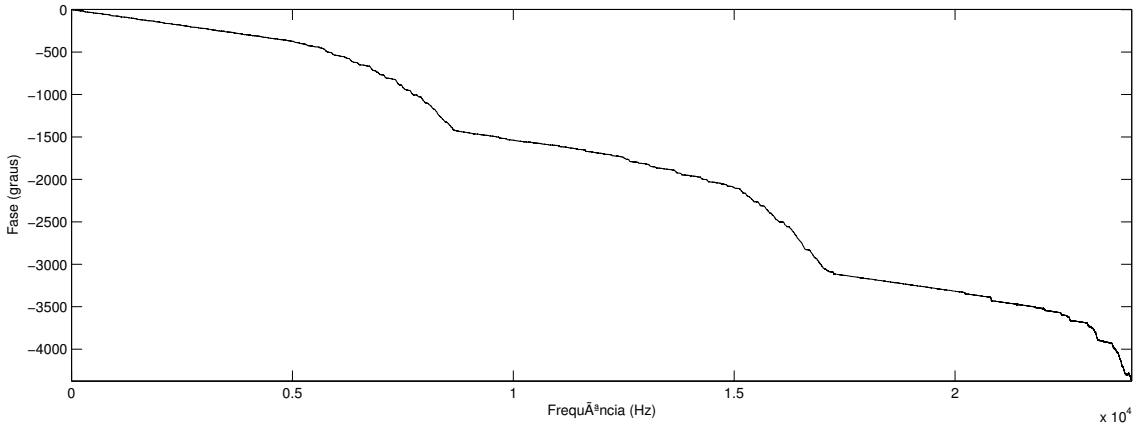
Fonte: Autoria própria.

Figura 35 – Resposta ao impulso da estrutura em FPGA



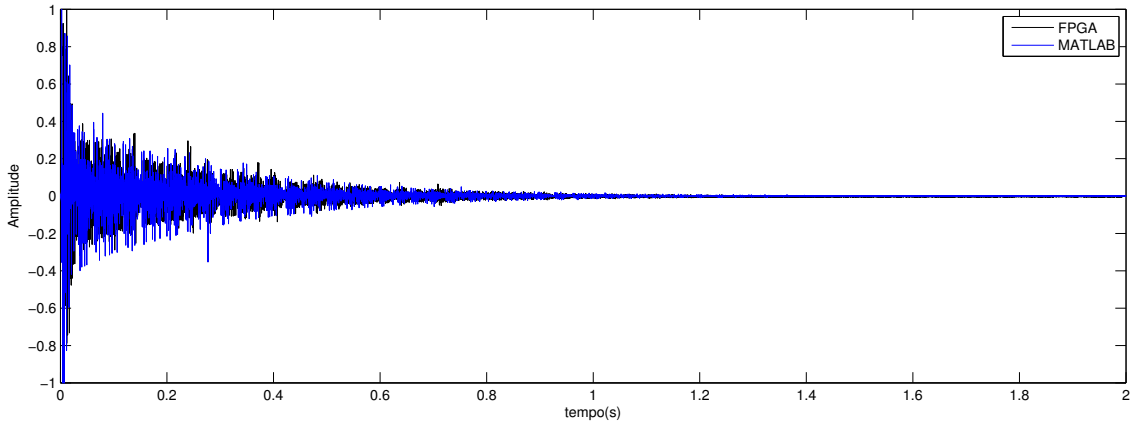
Fonte: Autoria própria.

Figura 36 – Resposta em fase da estrutura em FPGA



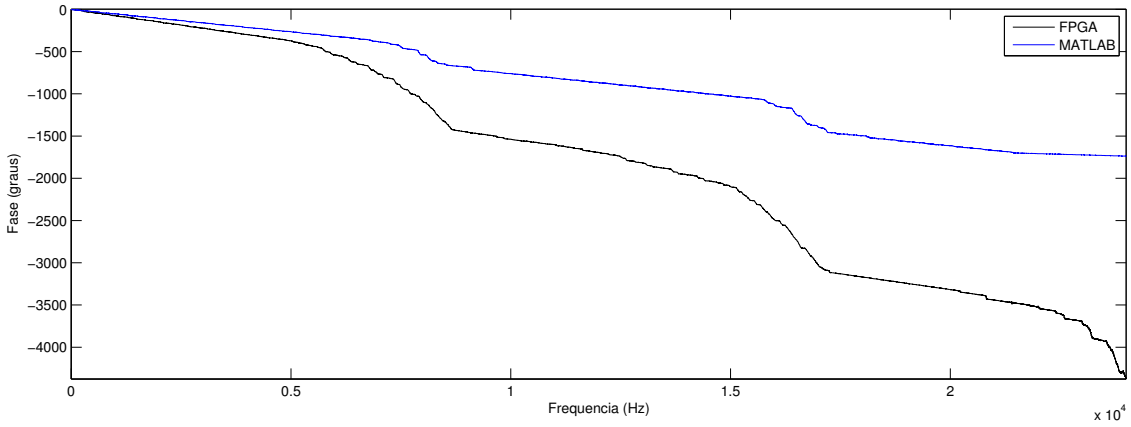
Fonte: Autoria própria.

Figura 37 – Comparação das respostas do FPGA e MATLAB



Fonte: Autoria própria.

Figura 38 – Comparação das respostas em fase do FPGA e MATLAB



Fonte: Autoria própria.

5.5 Consumo de recursos no FPGA

A tabela 9 apresenta o número de blocos do FPGA utilizado e seu percentual em relação ao total disponível, tais dados foram obtidos a partir do relatório gerado após o processo de síntese do sistema. A quantidade de recursos utilizada é moderada, permitindo por exemplo a expansão da arquitetura, aumento dos tempos de atraso ou a implementação de arquiteturas mais complexas.

Quadro 9 – Uso de recursos do FPGA

Uso da Lógica	**Usados**	**Disponível**	**Percentual**
Slice Registers	307	54576	0%
Slice LUTs	775	27288	2%
Fully Used LUT-FF pair	179	903	19%
Bonded IOBs	9	218	4%
Block RAM/FIFO	10	116	8%
BUFG/BUFGCTRLS	3	16	18%
DSP48A1s	14	58	24%

Fonte: Autoria própria.

6 Conclusões

Neste trabalho foi apresentada a implementação de um sistema completo de reverberação de áudio em FPGA, tendo como resultado um dispositivo capaz de receber dados analógicos, processa-los e retornar um sinal com o efeito de reverberação.

No decorrer do texto são apresentadas as bases para o entendimento do efeito de reverberação e as técnicas utilizadas para reverberação artificial, com foco na estrutura proposta por Schroeder. Tendo como base tais conceitos, foi descrito o procedimento para implementação da estrutura, detalhando cada um dos seus módulos, o CODEC de áudio, o controlador para o CODEC de áudio e o bloco de execução do efeito.

A avaliação do funcionamento do sistema foi feita através de simulações e testes práticos. Para o controlador do CODEC sinais foram simulados e seus diagramas de temporização comparados com as definições do padrão AC'97. O funcionamento do algoritmo de reverberação foi validado por testes de áudio e pela comparação da sua resposta ao impulso à resposta ao impulso do mesmo algoritmo implementado no *software* de matemática computacional e simulação MATLAB e com as características apresentadas por Schroeder em seu artigo original.

Como resultado o controle de CODEC se mostrou funcional ao permitir a comunicação com o CODEC LM4550 e efetuar as conversões de áudio. As respostas ao impulso apresentadas no Capítulo 5 são compatíveis com a teoria apresentada por Schroeder, assim como a resposta ao impulso do sistema completo foi também compatível com a resposta ao impulso do sistema implementado em *MATLAB*, utilizado como prova.

Com relação a implementação do projeto, a organização em forma modular dos dispositivos permite o reuso das estruturas com o mínimo de trabalho. Dessa forma o sistema se torna flexível, sendo capaz de servir como plataforma para implementação de outros algoritmos de processamento de áudio, especialmente os baseados em atraso.

Não foi possível realizar uma comparação satisfatória do desempenho do sistema ao apresentado por Bota et al. (2011), já que artigo original não apresenta dados suficientes. Porém, foi possível perceber que o presente trabalho consegue ser executado em tempo real a uma taxa de amostragem de 48 KHz, ao passo que o trabalho de Bota et al. (2011) opera a apenas 8 KHz, limitando sua aplicação a sistemas de baixa fidelidade.

Dentre as possíveis funcionalidades futuras do sistema é possível destacar:

- Implementação de outras arquiteturas de reverberação a partir da mesma plataforma;
- Com base nos mesmos blocos de atraso, implementação de outros efeitos também utilizados na indústria de áudio, como *phasing* (variação da fase), *chorus* (simulação de efeito de

coro de vozes), trêmolo (modulação em amplitude), vibrato (modulação em frequência), entre outros;

- Melhor utilização dos recursos disponíveis do *kit* de desenvolvimento, como as memórias DDR, possibilitando a implementação de algoritmos com grande consumo de memória, como a reverberação por convolução, ou a interface HDMI, para uso diretamente em sistemas de áudio digital;

- Criação de interface para usuário, permitindo o controle dos parâmetros como tempo de reverberação e decaimento em tempo real;

Referências

AXON, P.; GILFORD, C.; SHORTER, D. Artificial reverberation. *Proceedings of the IEE-Part B: Radio and Electronic Engineering*, IET, v. 102, n. 5, p. 624–640, 1955.

BELTRÁN, J. R.; BELTRÁN, F. A. Matlab implementation of reverberation algorithms. *Journal of New Music Research*, v. 31, n. 2, p. 153–161, 2002. ISSN 0929-8215. Disponível em: <http://www.tandfonline.com/doi/abs/10.1076/jnmr.31.2.153.8096>.

BLAUERT, J. *Spatial Hearing: The Psychophysics of Human Sound Localization*. MIT Press, 1997. ISBN 9780262024136. Disponível em: <http://books.google.com.br/books?id=wBiEKPhw7r0C>.

BOTA, C. et al. The implementation of schroeder reverberator on an fpga platform using xilinx system generator. *Acta Technica Napocensis*, Universitatea Tehnica Cluj- Napoca, v. 52, n. 4, 2011.

BYUN, K. et al. Development of portable sound effector. In: *2011 IEEE International Conference on Multimedia and Expo (ICME)*. [S.l.: s.n.], 2011. p. 1 –4.

BYUN, K. et al. Implementation of digital audio effect SoC. In: *IEEE International Conference on Multimedia and Expo, 2009. ICME 2009*. [S.l.: s.n.], 2009. p. 1194 –1197.

CATUNA, I.; SZOPOS, E.; TOPA, M. D. Labview fpga implementation of digital reverberators. *Acta Technica Napocensis*, Universitatea Tehnica Cluj- Napoca, v. 51, n. 3, 2010.

DATORRO, J. Effect design. part 1: Reverberator and other filters. *Journal of the Audio Engineering Society*, Audio Engineering Society, v. 45, n. 9, p. 660–684, 1997.

Digilent Incorporated. *Atlys Board Reference Manual*. [S.l.: s.n.], 2012.

DORNEAN, I.; TOPA, M.; KIREI, B. S. Digital implementation of artificial reverberation algorithms. *Acta Technica Napocensis*, v. 49, n. 4, 2008.

EVEREST, F.; POHLMANN, K. *Master Handbook of Acoustics*. Mcgraw-hill, 2009. ISBN 9780071603331. Disponível em: <http://books.google.com.br/books?id=6tiJ1cwnwxoC>.

GARDNER, W. G. *The virtual acoustic room*. Thesis — Massachusetts Institute of Technology, 1992. Thesis (M.S.)–Massachusetts Institute of Technology, Dept. of Architecture, 1992. Disponível em: <http://dspace.mit.edu/handle/1721.1/66351>.

GARDNER, W. G. Reverberation algorithms. In: KAHRS, M.; BRANDENBURG, K. (Ed.). *Applications of Digital Signal Processing to Audio and Acoustics*. [S.l.]: Springer, 2002. p. 85–131. ISBN 9780792381303.

HOWARD, D.; ANGUS, J. *Acoustics and psychoacoustics*. [S.l.]: Taylor & Francis US, 2009.

Intel Corporation. *AC '97 Component Specification*. [S.l.: s.n.], 2002.

JOT, J.-M.; CHAIGNE, A. Digital delay networks for designing artificial reverberators. In: *Audio Engineering Society Convention 90*. [s.n.], 1991. Disponível em: <http://www.aes.org/e-lib/browse.cfm?elib=5663>.

KAHRS, M.; BRANDENBURG, K. (Ed.). [S.l.]: Springer, 2002. ISBN 9780792381303.

MAYEN, J. R. S. *Implementing an AC97 Audio Controller IP*. Dissertação (Master of Science Thesis in Integrated Electronic System Design) — Chalmers University of Technology - University of Gothenburg, 2011. Disponível em: <http://publications.lib.chalmers.se/records/fulltext/146445.pdf>. Acesso em: 23 abr. 2013.

MOHAMMED, M.; BIJOY, K. Analysis of digital audio effects using simulink and c6713 dsk. In: *Innovations in Emerging Technology (NCOIET), 2011 National Conference on*. [S.l.: s.n.], 2011. p. 55–60.

MOORER, J. A. About this reverberation business. v. 3, p. 13, 1979. ISSN 01489267. Disponível em: <http://www.jstor.org/stable/3680280>.

National Semiconductor. *LM4550 Datasheet*. [S.l.: s.n.], 2004.

PFAFF, M. et al. Implementing digital audio effects using a hardware/software co-design approach. In: *Int. Conf. Digital Audio Effects (DAFx-07)*. [s.n.], 2007. p. 125–132. Disponível em: <http://dafx.labri.fr/main/papers/p125.pdf>.

RAFFEL, C.; SMITH, J. Practical modeling of bucket-brigade device circuits. In: *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx'10)*. [S.l.: s.n.], 2010.

RODRIGUEZ-ANDINA, J.; MOURE, M.; VALDES, M. Features, design tools, and application domains of FPGAs. *IEEE Transactions on Industrial Electronics*, v. 54, n. 4, p. 1810 –1823, ago. 2007. ISSN 0278-0046.

SABINE, W. C. Architectural acoustics. In: JSTOR. *Proceedings of the American Academy of Arts and Sciences*. [S.l.], 1906. v. 42, n. 2, p. 51–84.

SCHROEDER, M.; LOGAN, B. Colorless artificial reverberation. *IRE Transactions on Audio*, AU-9, n. 6, p. 209 – 214, dez. 1961. ISSN 0096-1981.

SCHROEDER, M. R. Natural sounding artificial reverberation. In: *Audio Engineering Society Convention 13*. [s.n.], 1961. Disponível em: <http://www.aes.org/e-lib/browse.cfm?elib=343>.

SCHROEDER, M. R. Statistical parameters of the frequency response curves of large rooms. *Journal of the Audio Engineering Society*, Audio Engineering Society, v. 35, n. 5, p. 299–306, 1987.

SEHN, L. R. *Implementação e otimização de uma arquitetura de reverberação digtal empregando técnicas de processamento multitaxa sobre plataforma reconfigurável*. 111 p. Dissertação (Mestrado em Computação) — Universidade Federal de Santa Maria, Santa Maria, 2009.

SHI, G.; MA, C. Subband dereverberation algorithm for noisy environments. In: *2012 IEEE International Conference on Emerging Signal Processing Applications (ESPA)*. [S.l.: s.n.], 2012. p. 127–130.

SMITH, J. *Physical Audio Signal Processing: For Virtual Musical Instruments and Audio Effects*. W3K Publishing, 2010. ISBN 9780974560724. Disponível em: <http://books.google.com.br/books?id=lDUIfAEACAAJ>.

A New Approach to Digital Reverberation Using Closed Waveguide Networks. 47-53 p. Disponível em: <https://ccrma.stanford.edu/files/papers/stanm31.pdf>.

STAUTNER, J.; PUCKETTE, M. Designing multi-channel reverberators. *Computer Music Journal*, The MIT Press, v. 6, n. 1, p. pp. 52–65, 1982. ISSN 01489267. Disponível em: <http://www.jstor.org/stable/3680358>.

TEICH, J. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, v. 100, n. Special Centennial Issue, p. 1411–1430, 2012. ISSN 0018-9219.

VALIMAKI, V. et al. Fifty years of artificial reverberation. *Audio, Speech, and Language Processing, IEEE Transactions on*, v. 20, n. 5, p. 1421 –1448, jul. 2012. ISSN 1558-7916.

VILLASENOR, J.; HUTCHINGS, B. The flexibility of configurable computing. *IEEE Signal Processing Magazine*, v. 15, n. 5, p. 67 –84, set. 1998. ISSN 1053-5888.

WANG, L.; YIN, F.; CHEN, Z. An "out of head" sound field enhancement system for headphone. In: *Neural Networks and Signal Processing, 2008 International Conference on*. [S.l.: s.n.], 2008. p. 517–521.

ZÖLZER, U. *Digital Audio Signal Processing*. [S.l.]: John Wiley & Sons, 1997. ISBN 9780471972266.

ZÖLZER, U. *DAFX: Digital Audio Effects*. [S.l.]: John Wiley & Sons, 2011. ISBN 9780470979679.

APÊNDICE A – Listagem do código do controlador AC'97

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity ac97 is
6    port (
7      n_reset         : in  std_logic;
8      clk             : in  std_logic;
9      ac97_sdata_out  : out std_logic;
10     ac97_sdata_in   : in  std_logic;
11     ac97_sync       : out std_logic;
12     ac97_bitclk     : in  std_logic;
13     ac97_n_reset    : out std_logic;
14     ac97_ready_sig  : out std_logic;
15     L_out           : in  std_logic_vector(17 downto 0);
16     R_out           : in  std_logic_vector(17 downto 0);
17     L_in            : out std_logic_vector(17 downto 0);
18     R_in            : out std_logic_vector(17 downto 0);
19     cmd_addr        : in  std_logic_vector(7 downto 0);
20     cmd_data        : in  std_logic_vector(15 downto 0));
21 end ac97;
22
23 architecture arch of ac97 is
24
25   signal bit_count         : integer range 0 to 255;
26   signal rst_counter       : integer range 0 to 1000;
27
28   signal latch_cmd_addr    : std_logic_vector(19 downto 0);
29   signal latch_cmd_data    : std_logic_vector(19 downto 0);
30
31   signal latch_left_data   : std_logic_vector(19 downto 0);
32   signal latch_right_data  : std_logic_vector(19 downto 0);
33
34   signal left_data       : std_logic_vector(19 downto 0);
35   signal right_data      : std_logic_vector(19 downto 0);
36   signal left_in_data    : std_logic_vector(19 downto 0);
37   signal right_in_data   : std_logic_vector(19 downto 0);
38
39 begin
40
```

```
41 -- Concatena bits para completar os 20 necessarios para transmissao.
42 -- Os conversores trabalham com 18 bits, mas o protocolo de comunicacao
43 -- usa 20.
44   left_data  <= L_out & "00";
45   right_data <= R_out & "00";
46
47 -- Recupera os 18 bits mais significativos, que sao os unicos com dados.
48 -- Os dois bits menos significativos sao '0' e desnecessarios.
49   L_in <= left_in_data(19 downto 2);
50   R_in <= right_in_data(19 downto 2);
51
52 -- Processo de RESET. Segura o pino RESET em baixa por 10us
53 -- para um clock de 100MHz
54   process (clk, n_reset)
55   begin
56     if (clk'event and clk = '1') then
57       if n_reset = '0' then
58         rst_counter <= 0;
59         ac97_n_reset <= '0';
60       elsif rst_counter = 1000 then
61         ac97_n_reset <= '1';
62         rst_counter <= 1000;
63       else
64         ac97_n_reset <= '0';
65         rst_counter <= rst_counter + 1;
66       end if;
67     end if;
68   end process;
69
70 -- Gera o sinal de "dados prontos" no bit 98.
71   process (clk, n_reset, bit_count)
72   begin
73     if(clk'event and clk = '1') then
74       if(bit_count = 98) then
75         ac97_ready_sig <= '1';
76       else
77         ac97_ready_sig <= '0';
78       end if;
79     end if;
80   end process;
81
82 -- Processo de envio de dados via SDO (ac97_sdata_out).
83   process (n_reset, bit_count, ac97_bitclk)
84   begin
85
86 -- No reset, retorna ao inicio da contagem de bits.
87     if(n_reset = '0') then
```

```
88        bit_count <= 0;
89      end if;
90
91      if (ac97_bitclk'event and ac97_bitclk = '1') then
92
93  -- Recupera os dados para transmissao do proximo quadro.
94        if bit_count = 255 then
95          latch_cmd_addr   <= cmd_addr & "000000000000";
96          latch_cmd_data   <= cmd_data & "0000";
97          latch_left_data  <= left_data;
98          latch_right_data <= right_data;
99        end if;
100
101 -- Gera o sinal de SYNC entre os bits 255 e 15.
102       if (bit_count >= 255) and (bit_count <= 15) then
103         ac97_sync <= '1';
104       else
105         ac97_sync <= '0';
106       end if;
107
108 -- Entre o bit 0 e bit 15 - Slot 0 - TAG
109       if (bit_count >= 0) and (bit_count <= 15) then
110         case bit_count is
111           when 0      => ac97_sdata_out <= '1';
112           when 1      => ac97_sdata_out <= '1';
113           when 2      => ac97_sdata_out <= '1';
114           when 3      => ac97_sdata_out <= '1';
115           when 4      => ac97_sdata_out <= '1';
116           when others => ac97_sdata_out <= '0';
117         end case;
118
119 -- Slot 1 (16 - 35) - CMD_ADDR
120       elsif (bit_count >= 16) and (bit_count <= 35) then
121         ac97_sdata_out <= latch_cmd_addr(35 - bit_count);
122 -- Slot 2 (36 - 55) - CMD_DATA
123       elsif (bit_count >= 36) and (bit_count <= 55) then
124         ac97_sdata_out <= latch_cmd_data(55 - bit_count);
125 -- Slot 3 (56 - 75) - Canal Esquerdo.
126       elsif ((bit_count >= 56) and (bit_count <= 75)) then
127         ac97_sdata_out <= latch_left_data(19);
128         latch_left_data <= latch_left_data(18 downto 0) & latch_left_data
          (19);
129 -- Slot 4 (76 - 95) - Canal Direito.
130       elsif ((bit_count >= 76) and (bit_count <= 95)) then
131         ac97_sdata_out <= latch_right_data(95 - bit_count);
132 -- Outros Slots (96 - 255) recebem 0.
133       else
```

```
134         ac97_sdata_out <= '0';
135       end if;
136 -- Incrementa o contador de bits.
137       if bit_count = 255 then
138         bit_count <= 0;
139       else
140         bit_count <= bit_count + 1;
141       end if;
142     end if;
143   end process;
144
145 -- Recupera em left_in_data e right_in_data os sinais adquiridos
146 -- pelo CODEC e transmitidos para o controlador via ac97_sdata_in.
147   process (ac97_bitclk)
148   begin
149     if (ac97_bitclk'event and ac97_bitclk = '0') then
150 -- Slot 3 (57 - 76) de sdata_in, canal esquerdo.
151       if (bit_count >= 57) and (bit_count <= 76) then
152         left_in_data <= left_in_data(18 downto 0) & ac97_sdata_in;
153 -- Slot 4 (77 - 96) de sdata_in, canal direito.
154       elsif (bit_count >= 77) and (bit_count <= 96) then
155         right_in_data <= right_in_data(18 downto 0) & ac97_sdata_in;
156       end if;
157     end if;
158   end process;
159
160 end arch;
```

APÊNDICE B – Listagem do código de inicialização do AC'97

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity ac97cmd is
6    port (
7       clk             : in  std_logic;
8       ac97_ready_sig   : in  std_logic;
9       cmd_addr        : out std_logic_vector(7 downto 0);
10      cmd_data        : out std_logic_vector(15 downto 0);
11      bypass          : in  std_logic);
12 end ac97cmd;
13
14 architecture arch of ac97cmd is
15   type state_type is (S0, S1, S2, S3, S4);
16   signal cur_state, next_state : state_type;
17 begin
18   -- Processo que faz a FSM andar.
19   process(clk, next_state, cur_state)
20   begin
21     if(clk'event and clk = '1') then
22       if ac97_ready_sig = '1' then
23         cur_state <= next_state;
24       end if;
25     end if;
26   end process;
27
28   -- FSM que escreve nos registradores de configuracao.
29   process (next_state, cur_state, bypass)
30   begin
31     case cur_state is
32 -- Headphone volume.
33       when S0 =>
34         cmd_addr <= X"04";
35         cmd_data <= X"0000";
36         next_state <= S1;
37 -- PCM Input Volume
38       when S1 =>
39         cmd_addr <= X"18";
40         cmd_data <= X"0808";
```

```
41          next_state <= S2;
42  -- Record select.
43        when S2 =>
44          cmd_addr <= X"1A";
45          cmd_data <= X"0404";
46          next_state <= S3;
47  -- Record gain.
48        when S3 =>
49          cmd_addr <= X"1C";
50          cmd_data <= X"0000";
51          next_state <= S4;
52  -- Bypass
53        when S4 =>
54          cmd_addr <= X"20";
55          cmd_data <= X"80" & bypass & "0000000";
56          next_state <= S0;
57        end case;
58    end process;
59  end arch;
```

APÊNDICE C – Listagem do código de *testbench* do AC'97

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity testbench is
6  end testbench;
7
8  architecture behavior of testbench is
9    signal clk : std_logic;
10   signal n_reset : std_logic;
11   signal sdata_in : std_logic;
12   signal bit_clk : std_logic;
13   signal sync : std_logic;
14   signal sdata_out : std_logic;
15   signal ac97_n_reset : std_logic;
16   signal l_bus, r_bus : std_logic_vector(17 downto 0);
17   signal l_bus_out, r_bus_out: std_logic_vector(17 downto 0);
18   signal bypass : std_logic;
19
20   -- Periodo para clock de 100MHz
21   constant period : time := 10 ns;
22   -- Periodo para bit clock de 12.288MHz
23   constant bit_clk_period : time := 81.380 ns;
24
25 begin
26
27 -- Componente a ser testado.
28   audiocodec : entity work.audio_codec
29     port map(clk => clk,
30              n_reset => n_reset,
31              sdata_in => sdata_in,
32              bit_clk => bit_clk,
33              sync => sync,
34              sdata_out => sdata_out,
35              ac97_n_reset => ac97_n_reset,
36              l_bus => l_bus,
37              r_bus => r_bus,
38              l_bus_out => l_bus_out,
39              r_bus_out => r_bus_out,
40              bypass => bypass);
```

```
41
42
43    bypass <= '0';
44
45  -- Palavras que simulam a existencia de um sinal de audio a ser escrito na
    saida.
46    l_bus <= "0111001000100000010";
47    r_bus <= "0010000001001110000";
48
49  -- Gerador de clock.
50    clock : process
51    begin
52      clk <= '0';
53      wait for period/2;
54      clk <= '1';
55      wait for period/2;
56    end process clock;
57
58  -- Gerador de BIT-CLOCK
59    bit_clock : process
60    begin
61      bit_clk <= '0';
62      wait for bit_clk_period/2;
63      bit_clk <= '1';
64      wait for bit_clk_period/2;
65    end process bit_clock;
66
67  -- Gera um sinal de reset aos 10k ciclos de bit_clk com duracao de 2
    periodos de bit_clk.
68    reset_sgn : process
69    begin
70      n_reset <= '1';
71      wait for 10000*bit_clk_period;
72      n_reset <= '0';
73      wait for 2*bit_clk_period;
74      n_reset <= '1';
75      wait;
76    end process reset_sgn;
77
78  -- Gera um sinal peridico para simular uma entrada de dados proveniente do
    CODEC.
79
80    audio_sgn : process
81    begin
82      wait for bit_clk_period;
83      sdata_in <= '0';
84      wait for 2*bit_clk_period;
```

```
85      sdata_in <= '1';
86    end process audio_sgn;
87
88  end;
```

APÊNDICE D – Listagem do código da entidade principal

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library ieee_proposed;
5  use ieee_proposed.fixed_pkg.all;
6
7  entity audio is
8      port(clk : in  std_logic;
9           n_reset : in  std_logic;
10          sdata_in : in std_logic;
11          bit_clk : in std_logic;
12          bypass : in std_logic;
13          sync : out std_logic;
14          sdata_out : out std_logic;
15          ac97_n_reset : out std_logic;
16          led : out std_logic
17         );
18 end audio;
19
20 architecture arch of audio is
21
22   signal l_bus, r_bus, l_bus_out, r_bus_out: std_logic_vector(17 downto 0);
23   signal cmd_addr : std_logic_vector(7 downto 0);
24   signal cmd_data : std_logic_vector(15 downto 0);
25   signal ready : std_logic;
26
27 begin
28
29 -- Saida para indicar o estado de bypass.
30   led <= bypass;
31
32 -- Canal esquerdo recebe o valor de canal direito,
33 -- ja que a arquitetura e mono.
34   r_bus <= l_bus;
35
36 -- Bloco de controle do CODEC.
37   ac97_cont : entity work.ac97
38     port map(n_reset => n_reset,
39           clk => clk,
40           ac97_sdata_out => sdata_out,
```

```
41            ac97_sdata_in => sdata_in,
42            ac97_sync => sync,
43            ac97_bitclk => bit_clk,
44            ac97_n_reset => ac97_n_reset,
45            ac97_ready_sig => ready,
46            l_out => l_bus,
47            r_out => r_bus,
48            l_in => l_bus_out,
49            r_in => r_bus_out,
50            cmd_addr => cmd_addr,
51            cmd_data => cmd_data);
52
53 -- Bloco de configuracao do CODEC.
54    ac97_ini : entity work.ac97cmd
55     port map(clk => clk,
56            ac97_ready_sig => ready,
57            cmd_addr => cmd_addr,
58            cmd_data => cmd_data,
59            bypass => bypass);
60
61 -- Bloco do efeito de reverberacao
62   schroeder_reverb : entity work.schroeder
63     port map(clk => ready,
64            reset => n_reset,
65            audio_in => to_sfixed(l_bus_out, 0, -17),
66            audio_out => l_bus);
67 end arch;
```

APÊNDICE E – Listagem do código do Reverberador de Schroeder

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library ieee_proposed;
5  use ieee_proposed.fixed_pkg.all;
6
7  entity schroeder is
8      port( clk : in  std_logic;
9            reset : in  std_logic;
10           audio_in : in sfixed(0 downto -17);
11           audio_out : out std_logic_vector(17 downto 0)
12         );
13 end schroeder;
14
15 architecture arch of schroeder is
16   signal sum, cf1_out, cf2_out, cf3_out, cf4_out, apf1_out, audio_out_sgn:
     sfixed(0 downto -17);
17   constant delay_cf1 : unsigned(10 downto 0) :=to_unsigned(2044,11);
18   constant delay_cf2 : unsigned(10 downto 0) :=to_unsigned(1872,11);
19   constant delay_cf3 : unsigned(10 downto 0) :=to_unsigned(1593,11);
20   constant delay_cf4 : unsigned(10 downto 0) :=to_unsigned(1440,11);
21
22   constant delay_apf1 : unsigned(7 downto 0) :=to_unsigned(240,8);
23   constant delay_apf2 : unsigned(7 downto 0) :=to_unsigned(81,8);
24
25   constant gain_cf1 : sfixed(0 downto -17) :=to_sfixed(0.8632, 0, -17);
26   constant gain_cf2 : sfixed(0 downto -17) :=to_sfixed(0.8740, 0, -17);
27   constant gain_cf3 : sfixed(0 downto -17) :=to_sfixed(0.8917, 0, -17);
28   constant gain_cf4 : sfixed(0 downto -17) :=to_sfixed(0.9016, 0, -17);
29   constant gain_apf : sfixed(0 downto -17) :=to_sfixed(0.7, 0, -17);
30
31   constant gain_schroeder : sfixed(0 downto -17) :=to_sfixed(0.5, 0, -17);
32   constant gain_direct : sfixed(0 downto -17) :=to_sfixed(0.5, 0, -17);
33   constant gain_comb : sfixed(0 downto -17) :=to_sfixed(0.25, 0, -17);
34
35 begin
36 -- Somatorio dos sinais de saida dos filtros comb.
37   sum <= resize(cf1_out*gain_comb + cf2_out*gain_comb + cf3_out*gain_comb +
      cf4_out*gain_comb, 0, -17);
38
```

```
39 -- Saida de audio.
40   audio_out <= to_slv(resize(audio_in*gain_direct+audio_out_sgn*
     gain_schroeder, 0, -17));
41
42 -- Filtros Comb.
43   comb_filter_1 : entity work.comb_filter
44     port map(clk => clk,
45           reset => reset,
46           delay => delay_cf1,
47           gain => gain_cf1,
48           audio_in => audio_in,
49           audio_out => cf1_out);
50
51   comb_filter_2 : entity work.comb_filter
52     port map(clk => clk,
53           reset => reset,
54           delay => delay_cf2,
55           gain => gain_cf2,
56           audio_in => audio_in,
57           audio_out => cf2_out);
58
59   comb_filter_3 : entity work.comb_filter
60     port map(clk => clk,
61           reset => reset,
62           delay => delay_cf3,
63           gain => gain_cf3,
64           audio_in => audio_in,
65           audio_out => cf3_out);
66
67   comb_filter_4 : entity work.comb_filter
68     port map(clk => clk,
69           reset => reset,
70           delay => delay_cf4,
71           gain => gain_cf4,
72           audio_in => audio_in,
73           audio_out => cf4_out);
74
75 -- Filtros Allpass.
76   allpass_filter_1 : entity work.allpass_filter
77     port map(clk => clk,
78           reset => reset,
79           delay => delay_apf1,
80           gain => gain_apf,
81           audio_in => sum,
82           audio_out => apf1_out);
83
84   allpass_filter_2 : entity work.allpass_filter
```

```
85      port map(clk => clk,
86            reset => reset,
87            delay => delay_apf2,
88            gain => gain_apf,
89            audio_in => apf1_out,
90            audio_out => audio_out_sgn);
91
92  end arch;
```

APÊNDICE F – Listagem do código do Filtro Comb

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library ieee_proposed;
5  use ieee_proposed.fixed_pkg.all;
6
7  entity comb_filter is
8      port( clk : in  std_logic;
9            reset : in std_logic;
10           delay : in unsigned(10 downto 0);
11           gain : in sfixed(0 downto -17);
12           audio_in : in sfixed(0 downto -17);
13           audio_out : out sfixed(0 downto -17)
14         );
15 end comb_filter;
16
17 architecture arch of comb_filter is
18
19   signal actual_addr, shifted_addr : unsigned(10 downto 0);
20   signal result_signal, delayed_signal : sfixed(0 downto -17);
21   signal delayed_signal_slv : std_logic_vector(17 downto 0);
22
23 begin
24 -- Conversao para representacao em ponto fixo do sinal de saida do buffer.
25   delayed_signal <= to_sfixed(delayed_signal_slv, 0, -17);
26
27 -- Processo
28   process(clk, reset, audio_in)
29   begin
30     if (clk'event and clk = '1') then
31       if reset = '0' then
32         audio_out <= (others => '0');
33       elsif(clk = '1') then
34 -- Equacao de diferencas do comb.
35         result_signal <= resize(audio_in + delayed_signal*gain, 0, -17);
36         audio_out <= result_signal;
37       end if;
38     end if;
39   end process;
40
```

```
41 -- Endereco com atraso.
42   shifted_addr <= actual_addr - delay;
43
44 -- Contador do buffer circular.
45 counter : entity work.cntr
46   port map (
47     clk => clk,
48     std_logic_vector(q) => actual_addr
49   );
50
51 -- Bloco de memoria do buffer circular.
52 buffer_circular : entity work.delay_fifo
53   port map (
54     clka => clk,
55     wea => "1",
56     addra => std_logic_vector(actual_addr),
57     dina => to_slv(result_signal),
58     clkb => clk,
59     web => "0",
60     addrb => std_logic_vector(shifted_addr),
61     dinb => "0000000000000000",
62     doutb => delayed_signal_slv
63   );
64 end arch;
```

APÊNDICE G – Listagem do código do Filtro Allpass

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  library ieee_proposed;
5  use ieee_proposed.fixed_pkg.all;
6
7  entity allpass_filter is
8      port( clk : in  std_logic;
9            reset : in std_logic;
10           delay : in unsigned(7 downto 0);
11           gain : in sfixed(0 downto -17);
12           audio_in : in sfixed(0 downto -17);
13           audio_out : out sfixed(0 downto -17)
14         );
15 end allpass_filter;
16
17 architecture arch of allpass_filter is
18
19   signal actual_addr, shifted_addr : unsigned(7 downto 0);
20   signal result_signal, delayed_signal, delayed_input_signal: sfixed(0
     downto -17);
21   signal delayed_signal_slv, delayed_input_signal_slv : std_logic_vector(17
      downto 0);
22
23 begin
24
25 -- Conversao dos sinais de saida dos buffers para ponto fixo.
26   delayed_signal <= to_sfixed(delayed_signal_slv, 0, -17);
27   delayed_input_signal <= to_sfixed(delayed_input_signal_slv, 0, -17);
28
29   process(clk, reset, audio_in)
30   begin
31     if (clk'event and clk = '1') then
32       if reset = '0' then
33         audio_out <= (others => '0');
34       elsif(clk = '1') then
35 -- Equacao de diferencas do allpass.
36         result_signal <= resize(audio_in*gain + delayed_input_signal -
           delayed_signal*gain, 0, -17);
37         audio_out <= result_signal;
```

```
38        end if;
39      end if;
40    end process;
41
42  -- Endereco com atraso.
43    shifted_addr <= actual_addr - delay;
44
45  -- Contador para o buffer circular.
46  counter : entity work.cntr_short
47    port map (
48      clk => clk,
49      std_logic_vector(q) => actual_addr
50    );
51
52  -- Memorias dos buffers circulares.
53  buffer_circular_1 : entity work.delay_fifo_short
54    port map (
55      clka => clk,
56      wea => "1",
57      addra => std_logic_vector(actual_addr),
58      dina => to_slv(result_signal),
59      clkb => clk,
60      web => "0",
61      addrb => std_logic_vector(shifted_addr),
62      dinb => "0000000000000000",
63      doutb => delayed_signal_slv
64    );
65
66   buffer_circular_2 : entity work.delay_fifo_short
67    port map (
68      clka => clk,
69      wea => "1",
70      addra => std_logic_vector(actual_addr),
71      dina => to_slv(audio_in),
72      clkb => clk,
73      web => "0",
74      addrb => std_logic_vector(shifted_addr),
75      dinb => "0000000000000000",
76      doutb => delayed_input_signal_slv
77    );
78  end arch;
```