**UNIVERSIDADE FEDERAL DE SANTA MARIA**
**CENTRO DE TECNOLOGIA**
**PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA**

# IMPLEMENTAÇÃO E OTIMIZAÇÃO DE UMA ARQUITETURA DE REVERBERAÇÃO DIGITAL EMPREGANDO TÉCNICAS DE PROCESSAMENTO MULTITAXA SOBRE PLATAFORMA RECONFIGURÁVEL

**DISSERTAÇÃO DE MESTRADO**

**Leandro Roberto Sehn**

**Santa Maria, RS, Brasil**
**2009**

# IMPLEMENTAÇÃO E OTIMIZAÇÃO DE UMA ARQUITETURA DE REVERBERAÇÃO DIGITAL EMPREGANDO TÉCNICAS DE PROCESSAMENTO MULTITAXA SOBRE PLATAFORMA RECONFIGURÁVEL

**Por**

**Leandro Roberto Sehn**

Dissertação apresentada ao Programa de Pós-Gaduação em Informática,
da Universidade Federal de Santa Maria (UFSM, RS), como
requisito parcial para a obtenção do grau de
**Mestre em Computação.**

**Orientador: Prof. Dr. João Baptista dos Santos Martins**

**Santa Maria, RS, Brasil**
**2009**

**Universidade Federal de Santa Maria**
**Centro de Tecnologia**
**Programa de Pós-Graduação em Informática**

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

# IMPLEMENTAÇÃO E OTIMIZAÇÃO DE UMA ARQUITETURA DE REVERBERAÇÃO DIGITAL EMPREGANDO TÉCNICAS DE PROCESSAMENTO MULTITAXA SOBRE PLATAFORMA RECONFIGURÁVEL

Elaborada por
**Leandro Roberto Sehn**

Como requisito parcial para obtenção do grau de
**Mestre em Computação**

**COMISSÃO EXAMINADORA:**

**Prof. Dr. João Baptista dos Santos Martins**
(Presidente/Orientador)

**Prof. Dr. Alexandre Campos** (UFSM)

**Prof. Dr. Giovani Baratto** (UFSM)

Santa Maria, RS, Brasil
2009

# RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

## IMPLEMENTAÇÃO E OTIMIZAÇÃO DE UMA ARQUITETURA DE REVERBERAÇÃO DIGITAL EMPREGANDO TÉCNICAS DE PROCESSAMENTO MULTITAXA SOBRE PLATAFORMA RECONFIGURÁVEL

AUTOR: LEANDRO ROBERTO SEHN
ORIENTADOR: PROF. DR. JOÃO BAPTISTA DOS SANTOS MARTINS
Data e local da defesa: Santa Maria, 30/10/2009

Este trabalho apresenta uma proposta de otimização para uma arquitetura de reverberação digital, empregando técnicas de processamento multitaxa sobre uma plataforma reconfigurável. A reverberação é um dos efeitos acústicos de maior ocorrência em nossa vida. Porém, apesar de muito comum, este fenômeno é muitas vezes imperceptível. Destaca-se que a presença de reverberação é de extrema importância particularmente no meio musical, pois ela adiciona senso de espaço às gravações (ou execuções) de determinada música, proporcionando assim uma maior naturalidade. Devido a esta importância, os primeiros reverberadores artificiais surgiram muito antes dos computadores digitais. Estes simuladores eram dispositivos eletro-acústicos que simulavam a reverberação fazendo uso de molas ou chapas de aço equipadas com transdutores. Com o surgimento dos primeiros computadores digitais, técnicas de processamento digital de sinais começaram a ser utilizadas, dando origem aos primeiros reverberadores digitais que simulavam a reverberação através do uso de filtros lineares em tempo discreto. Tendo em vista a recente evolução experimentada na área da computação configurável, surge uma tendência natural à pesquisa e desenvolvimento de sistemas acústicos baseados em tal plataforma. O processamento de sinais multitaxa se caracteriza pela mudança da freqüência de amostragem de um sinal, a partir da remoção ou adição de amostras na seqüência de entrada original. Dependendo da aplicação, a mudança da freqüência de amostragem pode reduzir consideravelmente a complexidade dos algoritmos e do *hardware*. Como o efeito de reverberação digital se baseia em linhas de atraso cujo tamanho é proporcional à freqüência de amostragem, e as técnicas de processamento multitaxa possibilitam a redução desta freqüência, visualiza-se então a redução da quantidade de memória necessária para a implementação do efeito em questão. Neste sentido, a arquitetura de reverberação digital proposta por James A. Moorer foi escolhida como base de desenvolvimento e comparação. Dos resultados obtidos neste trabalho, destaca-se a redução do consumo de memória em 50% em relação à arquitetura de referência. No tocante a compatibilidade de resultados, a arquitetura proposta apresentou uma resposta satisfatória, sendo imperceptíveis as diferenças entre a arquitetura de referência e a arquitetura proposta. Por fim, destaca-se que a arquitetura proposta pode ser utilizada na construção de outros efeitos de áudio baseados em atrasos de tempo, que se beneficiarão com a redução do consumo de memória proporcionada pela proposta em questão. Essa redução considerável de memória possibilita o emprego da arquitetura proposta em um *chip* único (*single-chip*) de baixo custo, e apresenta uma nova maneira de gerenciar os recursos computacionais exigidos pelos reverberadores digitais.

Palavras–chave: reverberação; FPGA; processamento multitaxa.

# ABSTRACT

Master Dissertation
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

## IMPLEMENTATION AND OPTIMIZATION OF A DIGITAL REVERBERATOR ARCHITECTURE APPLYING MULTIRATE PROCESSING TECHNIQUES OVER A RECONFIGURABLE PLATFORM

AUTHOR: LEANDRO ROBERTO SEHN
SUPERVISOR: PROF. DR. JOÃO BAPTISTA DOS SANTOS MARTINS
Date and Place of Presentation: Santa Maria, 30/10/2009

The following work presents a optimization proposal for a digital reverberation architecture applying multirate processing techniques over a reconfigurable platform. Reverberation is one of the acoustic effects that most occur in our lives. Although very common, this phenomenon is often imperceptible. It is noteworthy that the presence of reverberation has a paramount importance, particularly in the musical environment, since it adds sense of space to the recordings (or executions) of a particular song, making it sounds more natural. Due to this importance, the first artificial reverbs came much time before digital computers. These simulators were electro-acoustic devices that simulated the reverberation making use of springs or steel plates equipped with transducers. With the appearance of the first digital computers, digital signal processing techniques began to be used, leading to the first digital reverbs that simulate the reverberation using linear filters in discrete time. Considering the recent developments experienced in the configurable computing field, there is a natural tendency to research and develop acoustic systems based on such a platform. The multirate signal processing is characterized by changing the signal sampling frequency from the removal or addition of samples in the original input sequence. Depending on the application, changing the frequency of sampling can greatly reduce the algorithms and hardware complexity. As the reverb effect is based on digital delay lines which size is proportional to the sampling frequency, and, since multirate processing techniques allow the frequency reduction, is possible visualise the reduction in memory needed to implement the effect in question. In this sense, the architecture of digital reverb proposed by James A. Moorer was chosen as the basis for development and comparison. From the results of this work, it is highlighted the reduction in memory consumption by 50% compared to the reference architecture. Regarding the results compatibility, the proposed architecture presented a satisfactory response, being imperceptible the differences between the reference architecture and the proposed one. At last, it is emphasized that the proposed architecture can be used to build other audio effects based on time delays, which will benefit from the reduction in memory consumption afforded by the proposal. This considerable reduction in memory enables the proposed architecture utilization on a single low-cost chip and presents a new way to manage computational resources required by digital reverberators.

Key-words: reverberator; FPGA; multirate processing.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| DSP | *Digital Signal Processor/Processing* |
| FPGA | *Field Programmable Gate Array* |
| DE2 | *Development and Education board 2* |
| ITD | *Initial Time Delay* |
| MFP | *Mean Free Path* |
| AD | *Analógico para Digital* |
| DA | *Digital para analógico* |
| IIR | *Infinite Impulse Response* |
| FIR | *Finite Impulse Response* |
| LE | *Logic Element* |
| PLL | *Phase-Locked Loops* |
| DC | *Divisor de Clock* |
| UA | *Unidade de Atraso* |
| CG | *Controle de Ganho* |
| FP_PBX | *Filtro Pente Passa Baixas* |
| F_PTD | *Filtro Passa-Tudo* |
| REFPRI | *Reflexões Primárias* |
| REFTAR | *Reflexões Tardias* |
| DEC | *Decimador* |
| INT | *Interpolador* |
| DCM | *Digital Clock Manager* |
| LPM | *Library of Parameterized Modules* |
| FIFO | *First In First Out* |
| SCFIFO | *Single-clock FIFO* |
| RAM | *Random Access Memory* |
| ROM | *Read-Only Memory* |

| | |
|---|---|
| ROM_RT | *Read-Only Memory das Reflexões Tardias* |
| ROM_RP | *Read-Only Memory das Reflexões Primárias* |
| VHSIC | *Very High Speed Integrated Circuits* |
| VHDL | *VHSIC Hardware Description Language* |
| LUT | *Look Up Table* |
| GCLK | *Global Clocks* |

# SUMÁRIO

# 1. INTRODUÇÃO

## 1.1. Apresentação

A acústica é a ciência que estuda a produção, transmissão e efeitos do som. O som não é apenas um fenômeno que provoca sensação de audição através do ar. Por este motivo, freqüências inferiores (infrasons) ou superiores (ultrasons) à gama audível por um ser humano são também entendidas como som. O som é uma forma de transmissão de informação, sendo que essa forma de transmissão é geralmente afetada pelos meios que percorre, bem como pelos obstáculos que encontra (FERNANDES, 2002) (SEHN, 2004).

No final do século XIX, a possibilidade de gravar músicas se tornou possível. Conseqüentemente, não se tornava mais necessário ir a uma casa de concertos ou auditórios para ouvir determinada música. Porém, um problema se tornou evidente: dificilmente era possível ouvir as músicas gravadas em ambientes de boa acústica. Ao contrário das salas de concerto ou auditórios que foram projetados especificadamente para tornar as execuções musicais mais agradáveis, a peça de uma residência, por exemplo, não possuía uma arquitetura que fosse adequada para reproduções musicais.

A solução encontrada foi adicionar à gravação o efeito acústico proporcionado pelos auditórios e salas de concerto. Uma maneira de fazer isto é realizar a gravação em um ambiente de boa qualidade acústica. Porém, esta solução acabou se mostrando muito dispendiosa financeiramente. Sendo assim, técnicas para simular as propriedades acústicas de ambientes começaram a ser desenvolvidas (FERREIRA, 2004).

Um dos efeitos acústicos mais importantes (e de maior ocorrência) é a reverberação. Apesar de muito comum, este fenômeno é muitas vezes imperceptível. As paredes de um auditório ou escritório, as construções em uma rua, enfim, todos os objetos a nossa volta refletem e/ou absorvem o som que é propagado através destes ambientes. Devido a estas reflexões e/ou absorções, o que ouvimos não é somente a informação que partiu da fonte sonora; alguns componentes adicionais são somados ao som original (BELTRÁN, 2007). As reflexões modificam a percepção do som, alterando suas características timbrais e espaciais. A reverberação está intimamente associada com a arquitetura e com as propriedades acústicas de

determinado ambiente. Destaca-se que a presença de reverberação é particularmente importante no meio fonográfico. Praticamente todo o áudio que ouvimos em gravações musicais, rádio, televisão, filmes, entre outros, possuem reverberação artificial adicionada. O uso de reverberadores acrescenta ao sinal processado a sensação de espaço (TOMA, 2006) e, conseqüentemente uma maior naturalidade às gravações (ou execuções) de determinada música (ou qualquer outro tipo de sinal de áudio processado).

Devido a sua importância, muito esforço foi feito no sentido de simular este efeito através de dispositivos eletro-acústicos (BELTRÁN, 2007). Com o surgimento dos primeiros computadores digitais, técnicas de processamento digital de sinais (*Digital Signal Processing* – DSP) começaram a ser utilizadas, e continuam sendo utilizadas até hoje com o mesmo propósito.

## 1.2. Motivação

A engenharia de áudio é uma das modalidades da engenharia ainda muito pouco difundida no Brasil. Acústica, eletrônica analógica e digital, psicoacústica, processamento digital de sinais, entre outros, são apenas algumas das áreas de conhecimento que são abrangidos pela engenharia de áudio.

Nas ultimas décadas, a demanda por dispositivos portáteis com sistemas de DSP embutidos tem aumentado drasticamente. Telefones celulares, fones de ouvidos, processadores de áudio digital, entre outros, são dispositivos que se beneficiam de técnicas de DSP. Estes dispositivos possuem severas restrições como, por exemplo, área, velocidade, consumo de potência e tempo de desenvolvimento (ELHOSSINI, 2007). Sendo assim, o ciclo de desenvolvimento destes dispositivos requer uma abordagem que ofereça um produto com a melhor relação “Custo x Benefício” em um menor tempo possível para serem lançadas no mercado.

Na área de áudio digital, o processador de efeitos é um dos dispositivos mais utilizados, podendo ser utilizado tanto para voz quanto para instrumentos musicais. Dentre os efeitos comumente disponibilizados pelos processadores de efeitos, o efeito de reverberação (ou simplesmente *Reverb*) é o mais conhecido e utilizado.

A principal característica procurada em um processador de efeitos de áudio é a capacidade de realizar simultaneamente diversos algoritmos complexos em tempo real.

Particularmente na implementação de algoritmos de reverberação, um eficiente gerenciamento de memória se faz necessário. Os algoritmos de reverberação digital se baseiam em linhas de atraso que utilizam posições de memória para sua implementação. Ou seja, quanto maior o tempo de atraso, mais memória será utilizada.

Ao longo do tempo, os *Field Progammable Gate Arrays* (FPGAs) vem aumentando consideravelmente sua performance, bem como quantidade de recursos disponibilizados ao projetista. Fatores como o aumento da densidade lógica, incorporação de elementos como memórias e multiplicadores, elevaram o *status* dos FPGAs de blocos construtores para plataforma de implementação (DESSBESELL, 2008) (DAVIS, 2005) (FLETCHER, 2005). Portanto, o desempenho e a flexibilidade oferecidos pelos FPGAs fornecem um cenário ideal para a pesquisa e o desenvolvimento de produtos na área de áudio digital.

## 1.3. Objetivo

Este trabalho tem como objetivo o desenvolvimento de uma arquitetura de *hardware* otimizada através do uso de técnicas de processamento multitaxa, específica para a implementação de um algoritmo de reverberação digital, utilizando dispositivos lógicos programáveis (FPGA) como ambiente de pesquisa e prototipação. A escolha do efeito de reverberação como estudo de caso se deve ao fato de ser este o mais importante e conhecido fenômeno da acústica (VALLE, 2006). Outro fator motivador desta escolha foi o fato de que praticamente todos os demais efeitos de áudio têm sua base de desenvolvimento nos algoritmos empregados na implementação do *Reverb* (SEHN, 2004).

Segundo Dattorro (1997), os reverberadores digitais são como pinturas: existem infinitas técnicas, com as mais diversas cores, sendo que cada indivíduo possui suas cores de preferência. Sendo assim, a idéia de uma arquitetura universal se torna praticamente impossível. Porém, é possível destacar algumas arquiteturas que se tornaram referência ao longo do tempo. Os trabalhos desenvolvidos por Schroeder (1961), Moorer (1979), Gardner (1992) e Dattorro (1997), são geralmente tomados como base desenvolvimento devido à qualidade sonora alcançada por suas arquiteturas e principalmente pela documentação técnica, muito rara nesta área (PIIRILÄ, 1998), publicada por estes autores.

Sendo assim, como estudo de caso neste trabalho, a arquitetura de reverberação proposta por James A. Moorer (1979) foi escolhida como base de desenvolvimento e

comparação. Esta arquitetura foi fruto de um aperfeiçoamento feito através da reciclagem de diversas arquiteturas anteriores, e pode ser considerada como ponto de transição entre as arquiteturas tradicionais e as novas abordagens de implementação. Como não é escopo deste trabalho desenvolver uma nova arquitetura de reverberação, mas sim, apresentar uma nova forma de gerenciar os recursos computacionais exigidos pelos reverberadores digitais tradicionais, a arquitetura de Moorer apresenta-se como a escolha ideal.

O processo de conversão da taxa de amostragem (processamento multitaxa) de um sinal é um tipo de processamento que utiliza certos operadores que alteram a taxa de amostragem de determinado sinal, a partir da remoção ou adição de amostras na seqüência de entrada original. A mudança da freqüência de amostragem tem sido muito utilizada em DSPs e aplicações de áudio. Dependendo da aplicação, a mudança da freqüência de amostragem pode reduzir a complexidade dos algoritmos e do *hardware* ou aumentar a resolução em determinadas operações de processamento de sinais através da introdução de novas amostras de sinais (SPANIAS, 2007). Os blocos básicos de um sistema de processamento de sinais multitaxa são os decimadores (redução da taxa de amostragem) e os interpoladores (aumento da taxa de amostragem).

Como o efeito de reverberação digital se baseia em linhas de atraso cujo tamanho é proporcional à freqüência de amostragem, e as técnicas de processamento multitaxa possibilitam a redução desta freqüência, visualiza-se então a redução da quantidade de memória necessária para a implementação do efeito em questão. Reduzindo a quantidade de memória utilizada pelo sistema, conseqüentemente reduz-se a área necessária para sua implementação física, possibilitando então o emprego da arquitetura um *chip* único (*single-chip*) de baixo custo com área reduzida. Sendo assim, é possível destacar os principais objetivos na otimização da arquitetura escolhida:

- Redução do consumo de memória;
- Redução de área;
- Apresentar uma nova maneira de gerenciar os recursos computacionais exigidos pelos algoritmos de efeitos de áudio baseados em atrasos de tempo.

Por fim, além do desenvolvimento da arquitetura em si, outro objetivo consiste na comparação entre o modelo de referência tradicional e a arquitetura proposta.

## 1.4. Metodologia

Inicialmente, um estudo detalhado sobre os principais algoritmos de reverberação digital foi realizado. Diversas propostas de implementação foram analisadas, visando à escolha da melhor arquitetura.

Após definida a arquitetura a ser utilizada como estudo de caso, a implementação da mesma foi primeiramente realizada utilizando uma abordagem baseada em *software* construída sobre uma plataforma x86 (*Windows XP*), com auxílio da ferramenta MATLAB™ Simulink™. O intuito da construção de tal versão era possuir um parâmetro para comparações posteriores com a arquitetura desenvolvida em *hardware*.

Com a implementação em *software* concluída, partiu-se para o desenvolvimento sobre uma plataforma reconfigurável (FPGA) como arquitetura dedicada. Para este fim, como ambiente de prototipação do sistema, o *software* Quartus™ II foi utilizado em conjunto com o *kit* de desenvolvimento DE2 da Altera™, sendo que neste encontra-se o FPGA Cyclone II EP2C35F672C6.

## 1.5. Contribuições do trabalho

A partir do estudo realizado neste trabalho, é possível destacar as seguintes contribuições como as mais importantes:

- A definição de uma arquitetura de reverberação digital em *hardware*, otimizada para redução do consumo de memória e redução de área;
- A definição de um método para a redução do consumo de memória para algoritmos amplamente utilizados na área de áudio digital;
- A definição de um conjunto de blocos de *hardware* otimizados, que podem ser utilizados na construção de outros efeitos de áudio (*Delay*, *Chorus*, *Flanger*, entre outros).

## 1.6. Organização do trabalho

Este trabalho está dividido em cinco capítulos. O Capítulo 2 apresenta a revisão bibliográfica bem como a fundamentação teórica sobre os temas a serem abordados ao decorrer do projeto. Seguindo a estrutura, o Capítulo 3 apresenta a arquitetura proposta, com a descrição dos blocos principais que irão compor o sistema, bem como as devidas justificativas. No Capítulo 4 é apresentada a implementação do sistema, bem como os resultados parciais obtidos. O Capítulo 5 apresenta os resultados finais obtidos, bem como as conclusões e sugestões para trabalhos futuros.

# 2. FUNDAMENTAÇÃO TEÓRICA

## 2.1 Introdução

A maior parte de nossa vida acontece em ambientes reverberantes. Indo a um concerto musical, conversando com colegas no escritório, caminhando na rua, ou até mesmo em uma floresta, o som que ouvimos é invariavelmente acompanhado por reflexões atrasadas proveniente das mais diversas direções. Mas ao invés de causar confusão, este efeito geralmente passa despercebido. Isto acontece porque nosso sistema auditivo está bem equipado para lidar com a reverberação. Se as reflexões ocorrem rapidamente após o som inicial, nosso sistema auditivo não interpreta estes eventos como sons separados (KAHRS, 2002). Porém, as reflexões modificam as características do som, alterando seu timbre e principalmente as características espaciais do som.

A presença de reverberação é particularmente interessante no ramo musical, pois ela adiciona vida e senso de espaço, proporcionando assim uma maior naturalidade às gravações (ou execuções) de determinada música. A reverberação no interior de um automóvel, por exemplo, pode não ser suficientemente boa para a reprodução do som de uma orquestra sinfônica, e ao utilizarmos um fone de ouvidos, não existe a possibilidade de reverberação natural. Desta maneira, uma fonte sonora sem reverberação soará seca e artificial.

## 2.2 A história do áudio (Revisão bibliográfica)

Nos teatros da Grécia antiga (conhecidos também como Epidauros), a audiência sentava-se em lugares de inclinação íngreme (Figura 2.1), em torno de um palco onde a voz dos atores radiava com suficiente amplitude sobre toda a platéia. Superfícies refletoras de som eram colocadas em locais próximos do palco, permitindo assim uma melhor propagação da voz dos atores (FERNANDES, 2002) (SEHN, 2004).

**Figura 2.1 - Teatros da Grécia antiga (Epidauros) (MICHAUD, 2009)**

Posteriormente, nos grandes espaços fechados da Idade Média, como eram as igrejas (Figura 2.2), a reverberação provocada pelas paredes reflexivas de pedra era responsável por distribuir o som facilmente a todos os ouvintes, proporcionando um reforço auditivo. Por outro lado, tempos de reverberação muito longos tornavam a fala difícil de compreender. Como resultado, as falas e os cânticos tinham de ser efetuados de forma lenta, para se tornarem perceptíveis. Daí pensa-se, o aparecimento do Canto Gregoriano e outros arranjos corais executados de forma lenta (SANCHIDRIÁN, 2006) (FERNANDES, 2002).

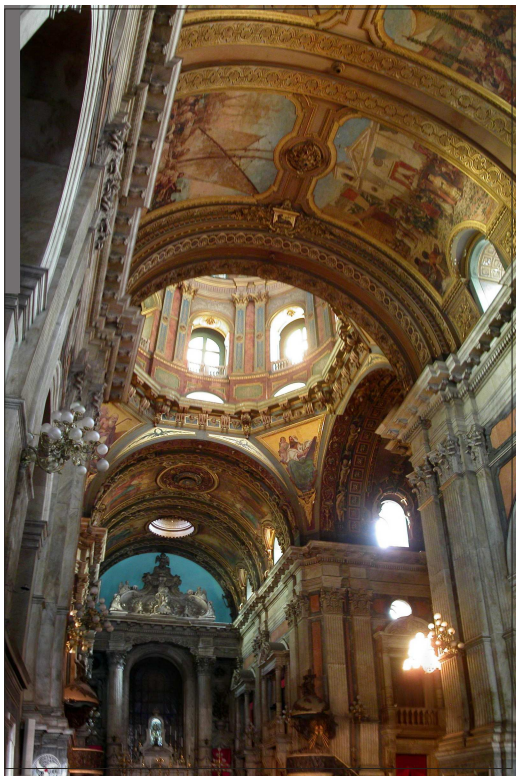

**Figura 2.2 - Estilo arquitetônico das igrejas da Idade Média (MICHAUD, 2009).**

A primeira teoria matemática da propagação sonora surge no livro *The Principia: Mathematical Principles of Natural Philosophy* de Isaac Newton (1686). A formulação de Newton, de pulsos de pressão transmitidos através das partículas de ar vizinhas, foi aperfeiçoada e corrigida em alguns pontos por Euler, d'Alembert e Lagrange, com base na consistência matemática então adquirida pela Mecânica dos Meios Contínuos (FREITAS, 2005).

A era da acústica moderna começa no princípio do século XX, com W.C. Sabine (MOORER, 1979) e as suas experiências que levaram à publicação da famosa fórmula de reverberação. Esta fórmula relaciona o volume de uma sala com a sua área de absorção, definindo assim o tempo de reverberação.

Uma vez que a reverberação é um efeito natural, a tarefa de produzir um dispositivo que simulasse este fenômeno com uma boa qualidade tornou-se extremamente importante (JOHANSSON, 2005). Devido a esta importância, os primeiros reverberadores artificiais surgiram muito antes dos computadores digitais, sendo chamados de reverberadores

analógicos. Estes simuladores eram dispositivos eletro-acústicos que simulavam a reverberação fazendo uso de molas ou chapas (placas) de aço equipadas com transdutores.

Um *reverb* de placa consiste de uma placa fina de aço ou outro metal resistente coberto com liga de ouro, que é posto a vibrar pelo sinal a ser processado (reverberado). Em outro ponto da placa o sinal é captado por um transdutor e então adicionado ao sinal original. O *reverb* de placa possui uma característica bastante natural porque as vibrações na placa são similares às vibrações do ar numa câmara de reverberação, sendo espalhadas em todas as direções da placa e refletidas quando atingem suas bordas, podendo-se distinguir bem as reflexões primárias das posteriores. O resultado é uma ambiência natural, mas o tempo de decaimento não pode ser modificado (WHITE, 2006).

O *reverb* de mola usa o mesmo princípio. Porém, o som reverberado possui uma qualidade inferior ao do sistema com placa. Sinais com muita dinâmica, como bateria, sofrem uma compressão alta quando reproduzidos num *reverb* de mola. Esses tipos de *reverb* podem ser encontrados ainda hoje em alguns amplificadores de guitarra. Pelo fato dos alto-falantes usados nesses amplificadores não reproduzirem freqüências muito altas, pode-se usar satisfatoriamente um *reverb* de baixa qualidade (e baixo custo) (RATTON, 2202).

Já os reverberadores analógicos mais modernos geralmente fazem uso de um circuito integrado conhecido como Tipo BBD (*Bucket Brigade Device*), que foi desenvolvido em 1969 por F. Sangster e K. Teer. O BBD é um dispositivo analógico, controlado por um *clock*. Ele é um semicondutor composto de uma linha comprida de transistores CMOS que vão passando o sinal de um para o outro a cada ciclo de *clock*. O BBD pode ser considerado como um registrador de deslocamento analógico, no qual a informação é enviada como pacotes de carga que passam ao longo de uma linha de capacitores de armazenamento sob o controle da interconexão de chaves implementadas pelos transistores (TOUMAZOU, 1993).

Com o surgimento dos primeiros computadores, técnicas de processamento digital de sinais começaram a ser utilizadas, dando origem aos primeiros reverberadores digitais que simulavam a reverberação através do uso de filtros lineares em tempo discreto (GARDNER, 1998). O primeiro reverberador digital foi desenvolvido por Schroeder (1962). Sua solução era baseada em uma estrutura recursiva, composta de quatro filtros Pente (*Comb Filters*), e dois filtros Passa-Tudo (*All-Pass Filters*) (PIIRILÄ, 1998).

Posteriormente, Moorer (1979), Gardner (1992) e Dattorro (1997), entre outros, deram continuidade às pesquisas sobre reverberação digital, oferecendo diferentes abordagens sobre como obter um algoritmo de reverberação com algumas características específicas como, por exemplo, naturalidade e controle do tempo de reverberação (JOHANSSON, 2005).

Muitos pesquisadores de diferentes áreas da engenharia têm dedicado esforços ao estudo do áudio e suas propriedades, com o objetivo de melhor compreender os princípios pelos quais se regem. Isto tem lhes permitido projetar sistemas acústicos cada vez mais perfeitos, e melhorar aqueles que pelos mais diversos motivos não o são (FERNANDES, 2002).

Uma revisão mais aprofundada sobre os algoritmos de reverberação digital pode ser encontrada em (GARDNER, 1998) e (KAHRS, 2002).

## 2.3 A reverberação

A reverberação é o mais importante fenômeno da acústica, responsável por belezas e por fracassos, dependendo da quantidade e da conveniência (VALLE, 2006).

O efeito de reverberação é produzido quando a energia do som é refletida e/ou absorvida pelas diversas paredes e objetos de um espaço, simulando assim, o espaço acústico no qual o som é produzido. Em um ambiente qualquer, as ondas sonoras são refletidas ao encontrarem uma superfície refletora. Essas primeiras reflexões são seguidas de outras reflexões menos intensas e mais atrasadas em relação ao sinal inicial (Figura 2.3), sendo que os intervalos de tempo desses atrasos são extremamente curtos (milissegundos entre as reflexões) e quase aleatórios, formando o que se conhece como difusão sonora. Esta difusão acontece, portanto, no tempo e no espaço (VALLE, 2006).

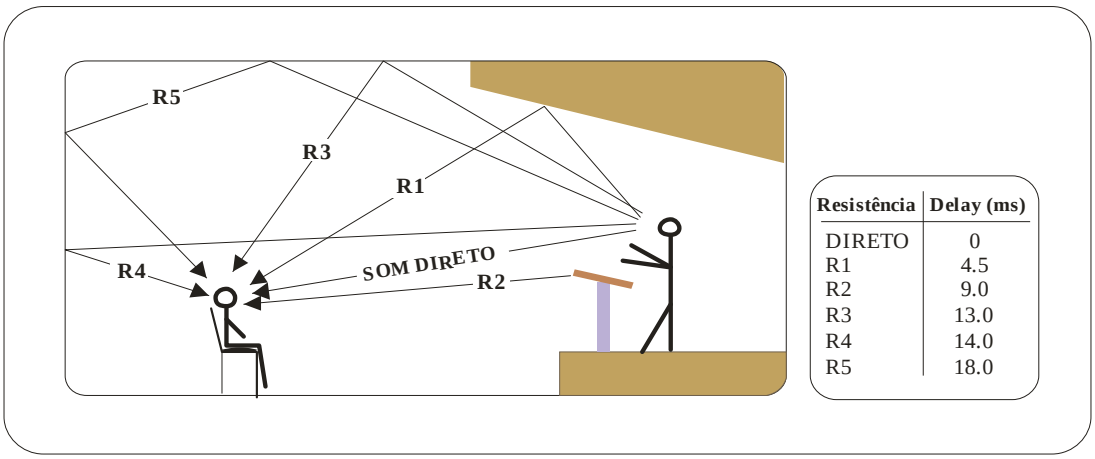


**Figura 2.3 - Reflexões que formam uma reverberação.**

Sendo assim, após a parada de um som fonte (um impulso, por exemplo), a reverberação produzida pelo ambiente causa a percepção de prolongamento do som original. Este prolongamento decai de forma suave e gradual; portanto o que se ouve não é somente a informação que parte da fonte sonora; alguns componentes adicionais são somados ao som original (BELTRÁN, 1999).

Existindo uma trajetória direta entre o sinal fonte e o ouvinte, o ouvinte ouvirá primeiramente o Som Direto (*Direct Sound*), seguido pelas reflexões das superfícies mais próximas. O tempo que a reverberação leva para desaparecer totalmente é chamado de Tempo de Reverberação (*Reverberation Time* ou $RT_{60}$).

### 2.3.1. Retardo inicial (*Initial Time Delay* – ITD)

O Retardo Inicial (ITD) é o tempo decorrido entre o som original e a primeira de todas as reflexões, incluindo as Reflexões Primárias (Figura 2.4). O ITD é uma medida segura do tamanho de um ambiente, pois a primeira de todas as reflexões virá do obstáculo mais próximo (VALLE, 2006).

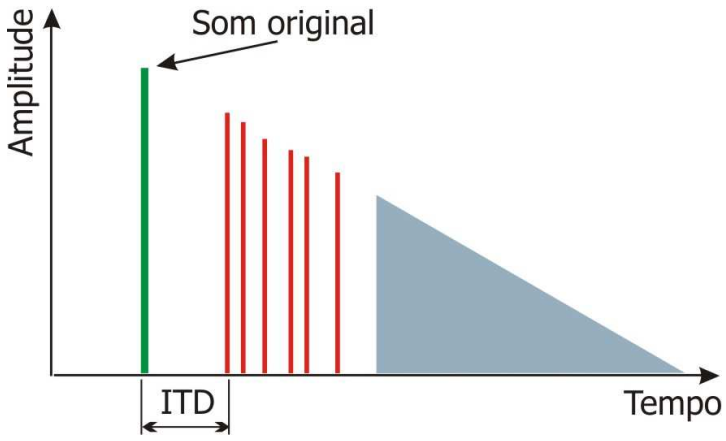


**Figura 2.4 – IDT, o retardo inicial até a primeira reflexão.**

### 2.3.2. Reflexões Primárias (*Early Reflections*)

Antes de o som se tornar difuso, espalhando-se por igual por todo o ambiente, ouvem-se algumas reflexões localizadas e distantes entre si no tempo e no espaço (VALLE, 2006). Essas primeiras reflexões são chamadas de Reflexões Primárias (Figura 2.5), e ocorrem normalmente entre 10 e 100 milissegundos após o Som Direto.

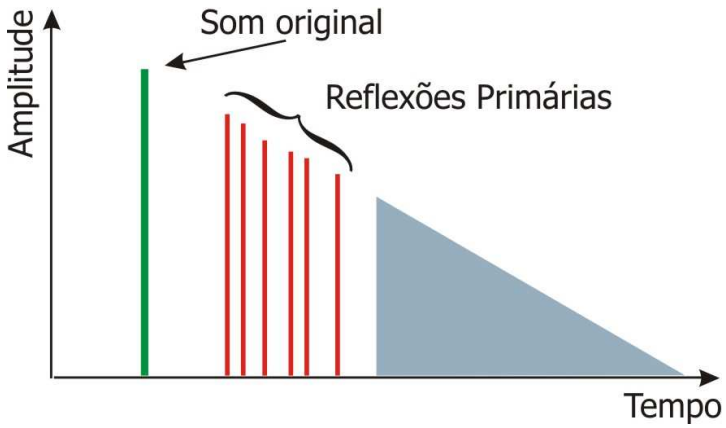


**Figura 2.5 – Reflexões primárias.**

As Reflexões Primárias são importantes porque são elas que propiciam ao ouvinte a sensação de direção e espaço, ou seja: onde se encontra a fonte sonora, qual o tamanho do ambiente e onde o ouvinte está posicionado (JOHANSSON, 2005). Sendo assim, o nível das Reflexões Primárias é uma informação psicoacústica sobre a distância de uma fonte sonora e sobre a característica acústica de determinado ambiente.

### 2.3.3. Pré-retardo (*Pre-delay*)

Após as Reflexões Primárias, o som se espalha pelo ambiente se tornando difuso, e assumindo a sonoridade semelhante a um prolongamento, que caracteriza a reverberação propriamente dita. Para que isso aconteça, é necessário que o ambiente seja grande o

suficiente para que dê tempo de essa difusão se formar, caso contrário, haverá somente um conjunto de Reflexões Primárias que, quando se extingue, deixa o local em silêncio.

Define-se, portanto, o Pré-retardo (Figura 2.6) como sendo o tempo decorrido entre o Som Direto (ou som original) e o início da reverberação. Portanto, somente após o Pré-retardo é que o som se tornará difuso.

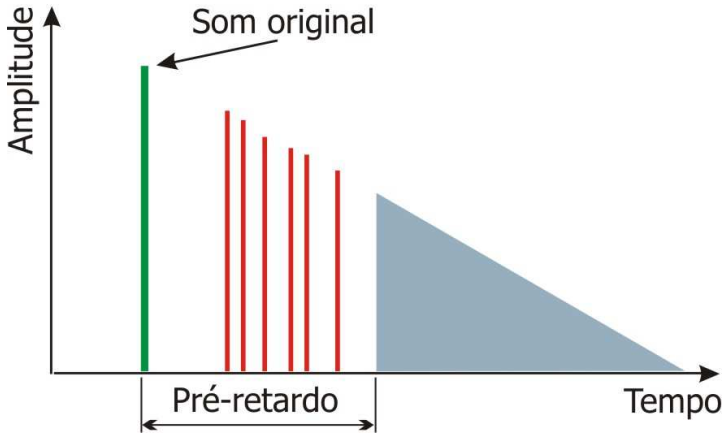


**Figura 2.6 – Pré-retardo, tempo até o início da reverberação difusa.**

O Pré–retardo é também um indicativo de tamanho de determinado ambiente, conforme especificado na Tabela 2.1 (VALLE, 2006).

**Tabela 2.1 – Pré-retardo, efeitos e características de tamanho de ambiente.**

| Pré-retardo | Efeito | Característica |
|---|---|---|
| Até 10 ms | Imperceptível | Ambiente muito pequeno |
| De 10 a 30 ms | Atraso imperceptível, mas a fonte se destaca pelo efeito Haas* | Ambiente pequeno |
| De 30 a 50 ms | Atraso perceptível onde a fonte se destaca muito bem | Ambiente médio |
| Mais de 50 ms | Grande atraso, onde há um período de silêncio entre o som original e a reverberação | Ambiente grande |

* Fenômeno psicoacústico que permite identificar corretamente a origem de um som percebido por ambos ouvidos, mas que os alcançam em momentos diferentes.

### 2.3.4. Reflexões Tardias (*Late Reflections*)

Após as Reflexões Primárias, o número de ondas sonoras refletidas torna-se muito grande, mas com uma redução suave da amplitude do som ao passar do tempo. Essas reflexões representam o denso conjunto de ecos que se deslocam em todas as direções e são chamadas de Reflexões Tardias (*Late Reflections*) ou Reflexões Difusas (*Diffuse Reverberation*) (Figura 2.7). São estas reflexões que caracterizam a reverberação propriamente dita (TOPA, 2008) (VALLE, 2006) (KAHRS, 2002).

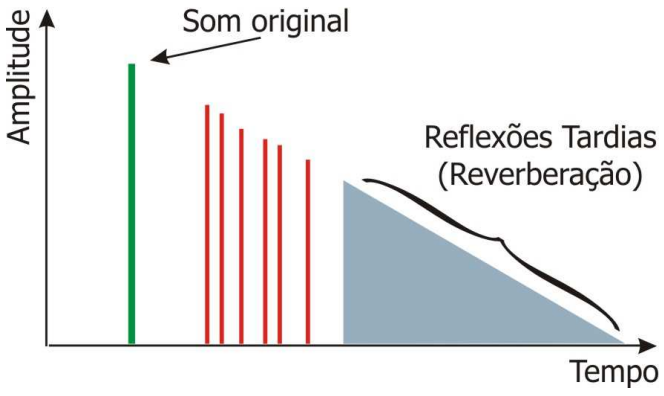


**Figura 2.7 – Reflexões tardias.**

### 2.3.5. Densidade da reverberação

A densidade de reverberação é definida como a quantidade de reflexões que forma o campo reverberante. Uma alta densidade produz uma reverberação suave e regular, soando como um sutil prolongamento do som original. Uma baixa densidade produz uma reverberação mais irregular, mais semelhante à sonoridade das Reflexões Primárias (VALLE, 2006).

### 2.3.6. Caminho Livre Médio (*Mean Free Path – MFP*)

O Caminho Livre Médio (MFP) é uma medida de "tamanho acústico" de uma sala. O MFP é medido em metros, correspondendo à distância média percorrida entre reflexões sucessivas do som. O MFP é calculado através da seguinte equação (2.1).

$$MFP = \frac{4V}{S_t} \qquad \textbf{(2.1)}$$

onde:

$V$ = volume da sala;

$S_t$ = superfície total da sala.

O significado do MFP como medida de "tamanho acústico" do ambiente é o seguinte: quanto maior o volume de uma sala, maior é o tempo decorrido entre cada reflexão sucessiva do som. Porém, para dois ambientes de mesmo volume e diferentes superfícies, a sonoridade será muito diferente. Quando a superfície total é menor, o tempo médio entre as reflexões é maior. Se duas salas possuem o mesmo volume e o mesmo $RT_{60}$, mas uma delas possui menor superfície, a sonoridade de cada uma será diferente, pois a de menor superfície possui maior índice de absorção. Neste caso, a sala de menor superfície possui um MFP mais longo, ou seja, tem-se em média mais tempo entre as reflexões sucessivas.

A conseqüência audível é de que a reverberação da sala de maior área é mais densa, ou seja, formada por um número maior de reflexões. Isto acontece devido ao fato de, no mesmo $RT_{60,}$ existir menos tempo entre as reflexões seguidas (VALLE, 2006).

### 2.3.7. Absorção em um ambiente

Os diferentes tipos de materiais presentes em um ambiente absorvem freqüências diferentes, dependendo da composição destes materiais. Portanto, o tempo de reverberação em determinado ambiente depende do volume da sala e das quantidades e espécies de materiais aplicados.

A Absorção Total (*A*) de som em um ambiente é a soma de todas as absorções parciais, providas por áreas revestidas com diferentes materiais. Cada material possui um índice de absorção ($\alpha$) que varia com a freqüência. Os valores de $\alpha$ para os diferentes tipos de materiais podem ser obtidos em livros especializados, ou então dos próprios fabricantes destes materiais.

A absorção parcial de um material é obtida multiplicando o índice de absorção $\alpha$ pela respectiva superfície (*s*) ocupada. A soma de todas as absorções parciais resulta na Absorção Total:

$$A = s_1\alpha + s_2\alpha_2 + s_3\alpha_3 + ...s_n\alpha_n \quad \textbf{(2.2)}$$

A Absorção Total é medida em Sabines (*sa*), unidade que significa “metros quadrados de absorção total”.

O índice médio de absorção ($\bar{a}$) é a média ponderada de todos os índices, ou seja, a soma das absorções parciais, divididas pela superfície total (VALLE, 2006):

$$\bar{a} = \frac{A}{S} = \frac{s_1\alpha_1 + s_2\alpha_2 + s_3\alpha_3 + ... + s_n\alpha_n}{s_1 + s_2 + s_3 + ... + s_n} \quad \textbf{(2.3)}$$

### 2.3.8. O tempo de reverberação (Fórmula de Sabine e correção)

Conforme mencionado anteriormente**,** o trabalho pioneiro de W.C. Sabine resultou na publicação da famosa fórmula de reverberação (SABINE, 1922). Seus experimentos iniciais consistiam de medir o decaimento do tempo de reverberação de uma sala, observando as mudanças que aconteciam conforme diferentes materiais absorventes eram adicionados ao ambiente. Sabine determinou então que o tempo de decaimento da reverberação era proporcional ao volume da sala, e inversamente proporcional a quantidade de absorção (SABINE, 1922) (KAHRS, 2002):

$$RT_{60} = \frac{0{,}16V}{A}(segundos) \quad \textbf{(2.4)}$$

onde:

$RT_{60}$ = tempo necessário para que o nível de pressão sonora (NPS) decaia 60dB em relação ao momento inicial;

$V$ = volume da sala;

$A$ = área total dos diferentes materiais absorventes presentes na sala (BOTTAZZINI, 2008).

Segundo Valle (2006), quando um ambiente possui pouca absorção, a fórmula de Sabine funciona perfeitamente. Porém, quando o $RT_{60}$ tende para zero, aparecem grandes erros. A razão para isto é que, para $RT_{60}$ igual a zero, o índice médio de absorção logicamente deveria ser de 100%, ou seja, todo o som é absorvido e não há o que reverberar. Mas ao utilizar-se $\bar{A}$ igual a 1 (um) na fórmula de Sabine, ainda assim haverá reverberação, o que é evidentemente incorreto.

O matemático Eyring (1930) descobriu a correção para este impasse, aplicando a seguinte correção ao valor do coeficiente de absorção:

$$\overline{A}_{Eyring} = -\ln(1-\overline{A}) \quad \textbf{(2.5)}$$

onde:

Sendo assim, se $\overline{A}$ é muito baixo, (1- $\overline{A}$) tende para 1 (um), cujo logaritmo é 0 (zero), e o $RT_{60}$ resultante é alto, aproximando-se do valor não corrigido. Porém, se $\overline{A}$ é igual a 1 (um), então (1- $\overline{A}$) é 0 (zero), e seu logaritmo é infinito negativo, fazendo com que $RT_{60}$ seja efetivamente 0 (zero).

### 2.3.9. Descrição modal da reverberação

No domínio das freqüências, uma sala pode ser caracterizada pelas suas freqüências características ou modos normais. O conhecimento das freqüências características de uma sala é essencial para o entendimento completo de suas propriedades acústicas, visto que ela age como um ressonador respondendo fortemente naqueles sons compostos com freqüências iguais ou próximas das freqüências características (KAHRS, 2002).

Quando um som é emitido em uma sala, ele excita um ou mais modos normais da mesma. Quando a fonte sonora é desligada, os modos continuam a ressonar com um decaimento determinado pelo seu fator de amortecimento que depende da absorção da sala. Isto é análogo a um circuito elétrico contendo inúmeras ressonâncias paralelas (BERANEK, 1986). Cada modo tem uma curva de ressonância associada, sendo que o fator de qualidade depende da constante de amortecimento.

O número $N_f$ de modos normais abaixo de uma freqüência $f_s$ é aproximadamente (KUTTRUFF, 1991):

$$N_f \approx \frac{4\pi V}{3c^3} f^3 \qquad \textbf{(2.6)}$$

onde:

$V$ = volume da sala ($V = L_x L_y L_z$);

$c$ = velocidade do som (m/seg).

Derivando a equação (2.6) em relação a $f$, obtém-se a densidade modal em função da freqüência (KAHRS, 2002):

$$\frac{dN_f}{df} \approx \frac{4\pi V}{c^3} f^2 \qquad \textbf{(2.7)}$$

Sendo assim, o número de modos por unidade de banda cresce proporcionalmente ao quadrado da freqüência.

### 2.3.10. Modelo estatístico para a reverberação

Acima de uma determinada freqüência crítica $f_c$ definida por:

$$f_c = 2000\sqrt{\frac{RT_{60}}{V}} \qquad \textbf{(2.8)}$$

onde:

$RT_{60}$ = tempo de reverberação em segundos;

$V$ = volume da sala em metros cúbicos.

a densidade de modos se torna tão alta que eles começam a se sobrepor. A separação média de máximos na resposta de magnitude será (VALLE, 2006):

$$\Delta f_{\max} \approx \frac{4}{RT_{60}} \tag{2.9}$$

Outro dado estatístico importante é o número de ecos que ocorrerão até o instante $t$ que é dado por (KAHRS, 2002):

$$N_t = \frac{4\pi(vt)^3}{3V} \tag{2.10}$$

onde:

$t$ = instante de tempo;

$v$ = velocidade do som.

Diferenciando a equação 2.10 em relação a $t$, obtém-se a densidade de ecos da sala. Ela é definida como sendo o número de ecos por segundo que chagam até o ouvinte, e cresce proporcionalmente ao quadrado do tempo:

$$\frac{dN}{dt} = \frac{4\pi v^3 t^2}{v} \tag{2.11}$$

### 2.3.11. A resposta ao impulso

De acordo com Topa (2006), a melhor maneira de se analisar o efeito de reverberação é através da resposta ao impulso. O conceito de resposta impulsiva pode ser definido como a resposta da pressão sonora registrada na posição de interesse do ouvinte, quando um sinal de excitação muito intenso e de curta duração é produzido.

A resposta impulsiva é particularmente atrativa por oferecer uma completa descrição da transmissão entre dois pontos. Sendo assim, através de um sinal de excitação emitido em determinado ambiente, é possível obter a curva de decaimento do som, bem como o tempo de reverberação (BOTTAZZINI, 2008). Portanto, a partir da resposta ao impulso fica evidente a possibilidade de parametrizar as reflexões de uma sala da forma como foi apresentada na Figura 2.3. A Figura 2.8 ilustra a resposta ao impulso parametrizada típica de um ambiente reverberante.

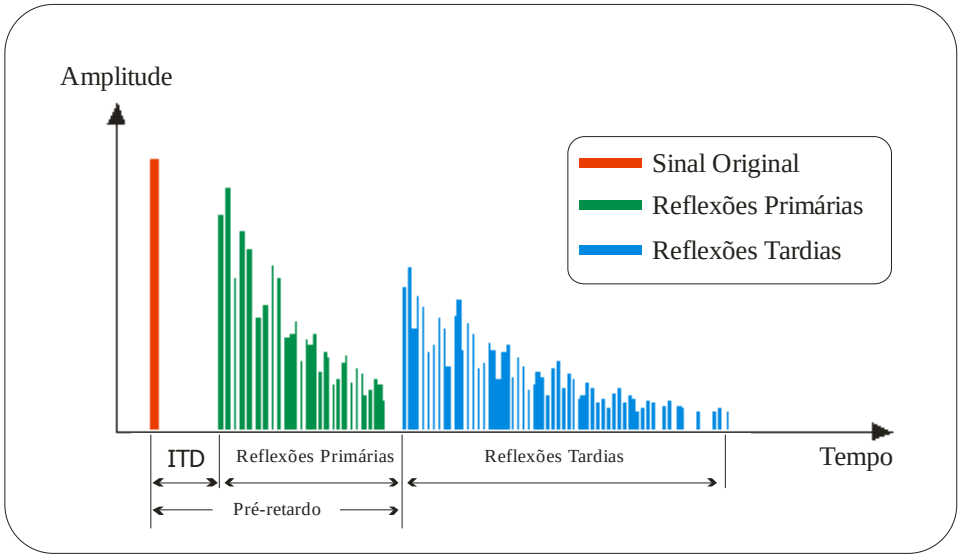


**Figura 2.8 - Resposta ao impulso parametrizada do Efeito *Reverb*.**

### 2.3.12. Tipos de reverberações e suas classificações

Os geradores de reverberação são classificados de acordo com o tipo de espaço simulado (*Room Type*). Sendo assim, para recriar o efeito de reverberação, algumas considerações a respeito do espaço acústico devem ser consideradas (SEHN, 2004), como por exemplo:

- Tamanho do espaço (*Room Size*);
- Características de absorção dos objetos e superfícies;
- Arquitetura do espaço simulado, etc.

Os efeitos de reverberação mais comuns são os efeitos do tipo Sala (*Room*) e do tipo Saguão (*Hall*). Na Tabela 2.2, têm-se as configurações básicas para os quatro tipos de efeitos de reverberação mais utilizados.

**Tabela 2.2 - Características de reverberação de diferentes espaços acústicos.**

| | **Pré-retardo (ms)** | **Tempo de reverberação (s)** | **Decaimento** |
|---|---|---|---|
| Concert Hall | 50 | 2,50 | Longo |
| Cathedral | 100 | 5 | Médio |
| Bathroom | 10 | 0,50 | Curto |
| Sitting room | 20 | 0,25 | Longo |

## 2.4. Filtros Digitais

A filtragem digital de sinais é uma operação básica da área de processamento digitais de sinais, sendo muito útil num conjunto vasto de aplicações, como por exemplo, sistemas de comunicações e processamento de áudio e imagem (FERREIRA, 2004). No ramo do áudio digital, os filtros são indispensáveis. Eles estão presentes em conversores AD/DA, processadores de efeitos, sistemas de transmissão, equalizadores, sistemas de compressão, entre outros.

Um filtro digital é um sistema temporal discreto projetado para passar o conteúdo espectral de um sinal de entrada em uma determinada banda de freqüências. Sendo assim, a função de transferência de um filtro forma uma janela espectral através da qual somente é permitida a passagem da parte desejada do espectro de entrada (SCANDELARI, 2008). De modo mais explícito, e especificamente de acordo com o assunto deste trabalho, os filtros digitais são utilizados para aumentar ou diminuir a quantidade de energia presente em certas freqüências ou faixas de freqüências de áudio (FERREIRA, 2004).

Os vários tipos de filtros (Figura 2.10) podem ser definidos de acordo com a seguinte classificação (DUTILLEUX, 1998):

- Passa-Altas (*High-Pass*);
- Passa-Baixas (*Low-Pass*);
- Passa-Banda (*Band-Pass*);
- Rejeita-Banda (*Band-Reject*);

sendo que outros tipos de filtros podem ser descritos através da combinação destes elementos básicos.

Os filtros digitais são geralmente implementados através uma das seguintes formas:

- Filtro de Resposta ao Impulso Infinita - IIR (*Infinite Impulse Response* );
- Filtro de Resposta ao Impulso Finita - FIR – (*Finite Impulse Response* ).

Nos capítulos seguintes, serão apresentados os conceitos básicos sobre os filtros utilizados neste trabalho.

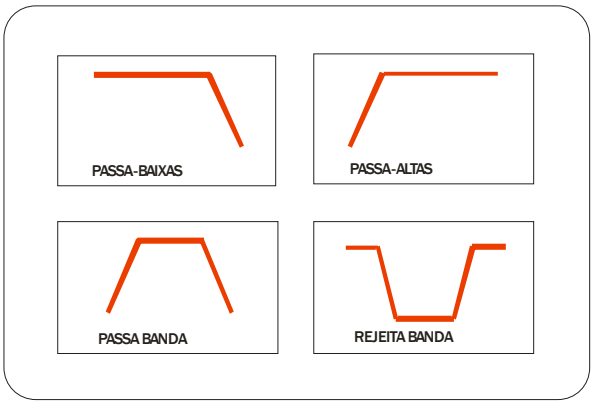


**Figura 2.9 - Classificação dos diferentes tipos de filtros.**

### 2.4.1. O filtro FIR *(Finite Impluse Response)*

Neste tipo de filtro, o valor da saída depende apenas do valor de entrada presente e de um número finito de entradas passadas. Devido ao fato de que os valores de saída passados não influenciam no cálculo dos valores de saída presentes, este filtro também é chamado de filtro não-recursivo (SCANDELARI, 2008).

Sua equação de diferenças é dada por (SHENOI, 2006):

$$y[n] = \sum_{k=0}^{L} a_k x[n-k] \tag{2.12}$$

que é a equação típica de um sistema não recursivo.

Sua resposta ao impulso é caracterizada por um número finito de impulsos com amplitude dada pelos coeficientes $a_k$, daí a razão do nome FIR (*Finite Impulse Response*).

Aplicando a transformada *z* à equação (2.12), tem-se a seguinte relação entrada-saída (DINIZ, 2004):

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{L} a_k z^{-1} = \sum_{k=0}^{L} h(k) z^{-1} \tag{2.13}$$

Na prática, a equação (2.13) pode ser implementada de várias maneiras distintas, utilizando blocos básicos (atraso, multiplicadores e somadores).

### 2.4.2. O filtro IIR *(Infinite Impulse Response)*

Neste tipo de filtro, o sinal de saída depende tanto dos valores de entrada quanto dos valores de saída anteriores. Por esta razão, estes filtros são conhecidos também como filtros recursivos (SCANDELARI, 2008).

A dependência das saídas passadas (recursividade) faz com que a duração da resposta seja infinita, mesmo quando cessam os sinais de entrada.

Sua equação de diferenças é definida por (SHENOI, 2006):

$$y[n] = -\sum_{k=1}^{N}(a_k y[n-k]) + \sum_{k=0}^{M}(b_k x[n-k]) \quad \textbf{(2.14)}$$

que é a equação típica de um sistema recursivo.

No domínio *z*, sua função de transferência será do tipo:

$$H(z) = \frac{N(z)}{D(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} \quad \textbf{(2.15)}$$

### 2.4.3. O filtro Passa-Baixas *(Low-Pass)* – IIR

No contexto deste trabalho, os filtros Passa-Baixas são utilizados para simular a atenuação das altas freqüências que ocorrem nos ambientes devido à absorção dos materiais. Sendo assim, este tipo de filtro permite a passagem das componentes harmônicas de valor mais baixo, que correspondem às tonalidades mais graves do som. Portanto, após passar por um filtro Passa-Baixas, o som tende a ficar mais grave e abafado (BARBOSA, 1999).

Sua equação de diferenças é definida por (SHENOI, 2006):

$$y[n] = (1-g)x[n] + gy[n-1] \quad \textbf{(2.16)}$$

No domínio *z*, sua função de transferência será do tipo:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1-g}{1-gz^{-1}} \quad \textbf{(2.17)}$$

### 2.4.4. O filtro Passa-Tudo *(All Pass)* – IIR

Os filtros Passa-Tudo são sistemas que, para todas as freqüências, mantém a magnitude inalterada, sendo sujeitos somente a atrasos de fase. Este tipo de filtro pode ser definido como um meio onde sinais se propagam através de um atraso dependente da freqüência, sem sofrer nenhuma absorção ou amplificação.

Sua equação de diferenças é definida por (ZÖLZER, 2002):

$$y[n] = x[n-m] - gx[n] + gy[n-m] \quad \textbf{(2.18)}$$

No domínio *z*, sua função de transferência será do tipo:

$$H(z) = \frac{Y(z)}{Z(z)} = \frac{-g + z^{-m}}{1-gz^{-m}} \quad \textbf{(2.19)}$$

### 2.4.5. O filtro Pente *(Comb Filter)* – IIR

Os filtros Pente são muito utilizados para a construção de filtros redutores de ruídos, sendo inerente em efeitos digitais de áudio como *Delay*, *Chorus* e *Flanger*. Os filtros Pente trabalham somando ao sinal de entrada uma versão atrasada e escalonada do mesmo. Isto faz com que algumas freqüências venham a ser se atenuadas, e outras ampliadas. Este controle se dá através de adição ou subtração.

Sua equação de diferenças é definida por (ZÖLZER, 2002):

$$y[n] = x[n] + ay[n - D] \tag{2.20}$$

onde $D$ representa o atraso (número de amostras) e $a$ representa o coeficiente de ganho. Neste caso, é preciso destacar que $D$ é um atraso bem maior do que 1 (um).

No domínio $z$, sua função de transferência é dada por:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - az^{-D}} = \frac{z^D}{z^D - a} \tag{2.21}$$

### 2.4.6. Linha de atraso (*Delay line*)

A linha de atraso é uma estrutura básica de processamento de sinais, que pode ser utilizada tanto para filtrar um sinal quanto para produzir um efeito sonoro baseado em atrasos de tempo como *Chorus, Flanger, Reverb* e *Delay*.

O tempo de atraso de um sinal de áudio digital é determinado por:

- Tamanho do *Buffer* (*Buffer Size*): representado pela letra *D*, este parâmetro representa o número de palavras (endereços de armazenamento) definidos para o *buffer*;
- Taxa de Amostragem (*Sampling Rate*): geralmente determinado pelos conversores de áudio. A taxa de amostragem corresponde à freqüência com que dados são recebidos, processados e retornados pelo processador DSP.

A equação básica da linha de atraso é a seguinte:

$$y[n] = x[n - D] \tag{2.22}$$

Para determinar o tempo de atraso (em segundos), utiliza-se a seguinte equação:

$$\text{Atraso (segundos)} = \frac{D}{f_s} \tag{2.23}$$

Portanto, de acordo com a equação (2.23), para aumentar tempo de atraso de um sinal, o tamanho do *buffer* também deve ser aumentado para poder armazenar mais amostras, ou a freqüência de amostragem deve ser diminuída. Porém, diminuindo a freqüência de amostragem, o tamanho da banda também é reduzido. Em alguns casos isto não é necessariamente um problema. Por exemplo, a voz humana ou um instrumento de cordas requerem na maioria dos casos um tamanho de banda de somente 6 kHz. Entretanto, existem certos instrumentos que podem exigir 15 kHz de banda (Tabela 2.3), sendo indesejável a diminuição da freqüência de amostragem (TOMARAKOS, 1998).

**Tabela 2.3 – Faixa de freqüência dos instrumentos musicais**

| Instrumento | Fundamental | Harmônicos |
|---|---|---|
| Flauta | 261-2349 Hz | 3-8 KHz |
| Oboé | 261-1568 Hz | 2-12 KHz |
| Clarinete | 165-1568 Hz | 2-10 KHz |
| Fagot | 62-587 Hz | 1-7 KHz |
| Trompete | 165-988 Hz | 1-7.5 KHz |
| Trombone | 73-587 Hz | 1-4 KHz |
| Tuba | 49-587 Hz | 1-4 KHz |
| Bumbo | 30-147 Hz | 1-6 KHz |
| Pratos | 300-587 Hz | 1-15 KHz |
| Violino | 196-3136 Hz | 4-15 KHz |
| Viola | 131-1175 Hz | 2-8.5 KHz |
| Cello | 65-698 Hz | 1-6.5 KHz |
| Baixo acústico | 41-294 Hz | 1-5KHz |
| Baixo elétrico | 41-300 Hz | 1-7 KHz |
| Guitarra acústica | 82-988 Hz | 1-15 KHz |
| Guitarra elétrica (amplif.) | 82-1319 Hz | 1-3.5 KHz |
| Guitarra elétrica (direta) | 82-1319 Hz | 1-15 KHz |
| Piano | 28-4196 Hz | 5-8 KHz |
| Sax Soprano | 247-1175 Hz | 2-12 KHz |
| Sax alto | 175-698 Hz | 2-12 KHz |
| Sax tenor | 131-494 Hz | 1-12 KHz |
| Cantor | 87-392 Hz | 1-12 KHz |

A Figura 2.11 ilustra o diagrama de bloco responsável pela geração do atraso.

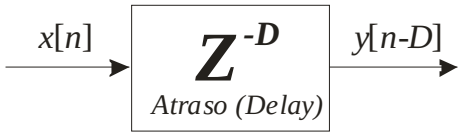


**Figura 2.10 – Diagrama de bloco representando a geração de atraso.**

Analisando a equação (2.23), conclui-se que a geração de um atraso de tempo está vinculada a freqüência de amostragem e ao tamanho do *buffer* utilizado. Sendo assim, tomando como exemplo um *buffer* com $2^{12}$ palavras, e uma freqüência de amostragem de 44,1 kHz, o tempo de atraso poderá variar entre 0 e 0,09 segundos.

### 2.4.7. *Buffer* circular

Na implementação de algoritmos de processamento de sinais em tempo real, para calcular o sinal de saída, é preciso ter acesso a um determinado número de amostras anteriores do sinal de entrada. Por exemplo, suponha a implementação de um filtro FIR que utilize oito coeficientes para sua implementação (Figura 2.12). Neste caso, as últimas oito amostras passadas são necessárias para o cálculo do sinal filtrado. Estas amostras devem estar armazenadas em memória, sendo continuamente alteradas à medida que novas amostras são adquiridas. Sendo assim, a melhor maneira de gerenciar estas amostras é a utilização de um *buffer* circular (SMITH, 1999).

A idéia principal de um *buffer* circular é a que o fim do vetor de entrada deve ser conectado à seu início. Para isto, faz-se uso de ponteiros (variáveis cujo valor é um endereço de memória). Quatro parâmetros são necessários para gerenciar um *buffer* circular:

- Um ponteiro que indique o início do *buffer* na memória;
- Um ponteiro que indique o fim do *buffer*;
- O tamanho de cada bloco de memória;

Estes três valores definem o tamanho e a configuração do *buffer* circular e serão constantes durante a operação do sistema.

- Um ponteiro que indica a posição da amostra mais recente.

Este parâmetro deve ser atualizado a cada nova amostra adquirida. Em outras palavras, deve haver uma lógica que controle como o valor deste ponteiro deve ser atualizado, baseado nos três primeiros parâmetros que de configuração do *buffer*. Esta lógica deve ser simples, porém muito rápida. Sendo assim, deve-se otimizar e gerenciar o *buffer* circular de maneira a acelerar a velocidade de processamento (SMITH, 1999).

Na Figura 2.13, é possível observar o exemplo de um *buffer* circular para o filtro FIR com oito coeficientes.

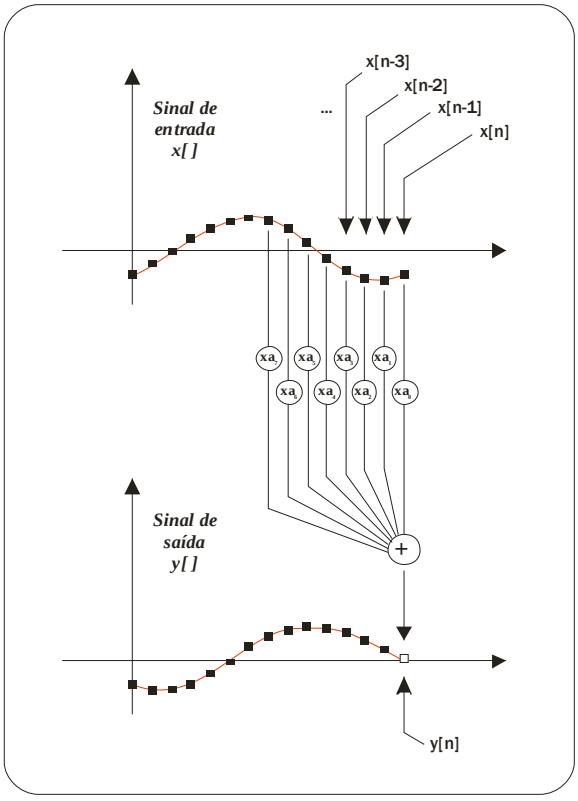


**Figura 2.11 - Filtro baseado em amostras passadas.**

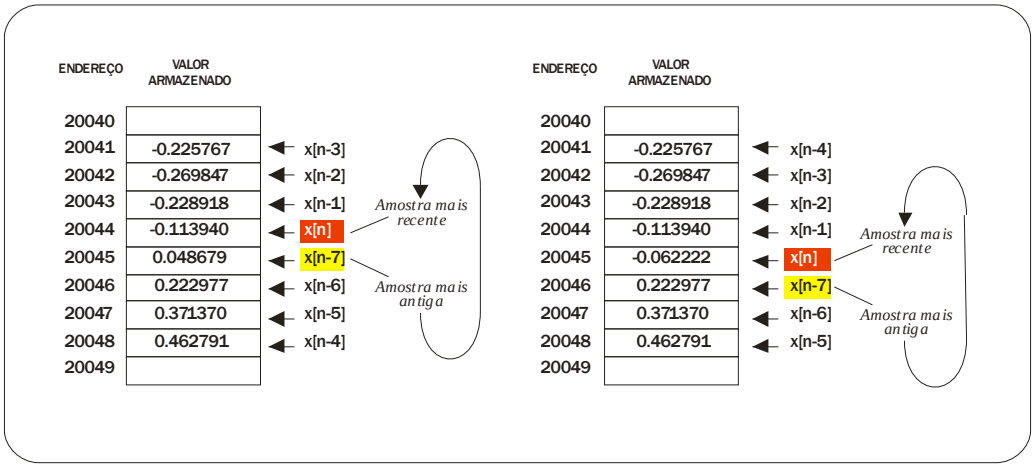


**Figura 2.12 – *Buffer* Circular.**

## 2.5. Processamento multitaxa (Decimação e interpolação)

O processo de conversão da taxa de amostragem de um sinal é um tipo de processamento que utiliza certos operadores que alteram a taxa de amostragem de determinado sinal, a partir da remoção ou adição de amostras na seqüência de entrada original. Este processo pode ser visto como uma operação de filtragem, conforme o esquema apresentado na Figura 2.14 (JESUS, 2002) (PROAKIS, 1996).

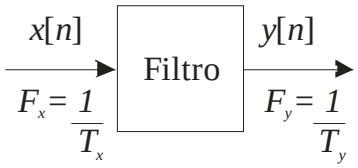


**Figura 2.13 – Conversão de taxa de amostragem visto como um processo de filtragem.**

onde $x[n]$ representa o sinal de entrada caracterizado por uma taxa de amostragem $F_x = 1/T_x$, e $y[n]$ representa o sinal de saída caracterizado por uma taxa de amostragem $F_y = 1/T_y$, sendo que $T_x$ e $T_y$ representam os respectivos intervalos de amostragem.

Os principais operadores multitaxa são os decimadores e os interpoladores, que operam em conjunto com filtros digitais, formando as estruturas básicas de filtragem digital multitaxa (GUEDES, 2002). Sendo assim, designa-se por decimação ao processo de reduzir a taxa de amostragem por um fator $M$. Ao processo de aumentar a taxa de amostragem por um fator $L$, dá-se o nome de interpolação.

### 2.5.1. Decimação por um fator *M*

O processo de decimar ou subamostrar um sinal digital é realizado pelo dispositivo decimador, ilustrado na Figura 2.15. Tal dispositivo, tendo um sinal de entrada $x[n]$ e um fator $M$, é responsável por gerar um sinal de saída $y[n]$ com uma taxa de amostragem $M$ vezes menor do que a taxa de amostragem do sinal de entrada.

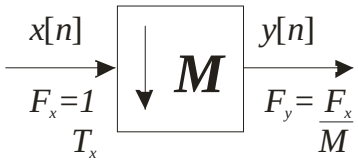


**Figura 2.14 – Diagrama de bloco representando a decimação por um fator *M*.**

Em outras palavras, o decimador retém todas as amostras com índice múltiplo de *M*, removendo as *M*-1 amostras entre as consecutivas amostras retidas. Matematicamente, a relação entre entrada e saída é descrita por (CARVALHO, 2008):

$$h[n] = x[nM] \tag{2.24}$$

O espectro de um sinal decimado possui relação estreita com o espectro do sinal original, a qual é dada pela equação (2.25), que estabelece que o espectro do sinal decimado, é dado por um somatório de versões do espectro original expandidas de *M* e deslocadas de 2π (FURTADO, 2006).

$$X_d(e^{jw}) = \frac{1}{M}\sum_{k=0}^{M-1} X(e^{j\frac{w-2\pi k}{M}}) \tag{2.25}$$

É sabido que reduzindo da taxa de amostragem através da seleção dos valores de $x[n]$ para múltiplos de *M*, o sinal resultante conterá *aliasing*, com duplicação em $F_x/2M$. Desta forma, para evitar o *aliasing*, torna-se necessário reduzir a largura de banda de $x[n]$ para $F_{\max} = F_x/2M$, ou equivalentemente, $\Omega_{\max} = \pi/M$. Sendo assim, utiliza-se um filtro Passa-Baixas (*anti-aliasing*) para eliminar o espectro de $X(\Omega)$ no intervalo $\pi/M < \Omega < \pi$. Este filtro é caracterizado pela resposta ao implulso $h[n]$ e uma resposta de freqüência $H_D(\Omega)$, que satisfaz a seguinte condição (PROAKIS, 1996):

$$H_D(\Omega) = \begin{cases} 1, & |\Omega| \leq \pi/\mathrm{D} \\ 0 & \text{demais casos} \end{cases} \quad \textbf{(2.26)}$$

A saída do filtro é uma seqüência $v[n]$ dada por:

$$v[n] = \sum_{k=0}^{\infty} h[k]x[n-k] \quad \textbf{(2.27)}$$

O processo completo de decimação está ilustrado na Figura 2.16.

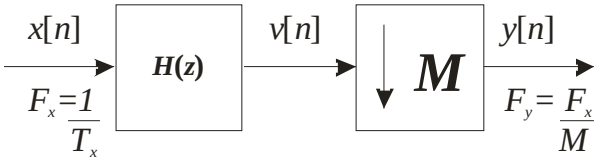


**Figura 2.15 – Operação geral de decimação.**

2.5.2. Interpolação por um fator *L*

A operação inversa da decimação é a interpolação ou sobreamostragem. Este dispositivo está graficamente representado na Figura 2.17.

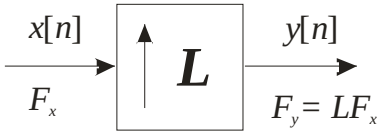


**Figura 2.16 - Diagrama de bloco representando a interpolação por um fator *L*.**

O interpolador tem por objetivo aumentar a taxa de amostragem de determinado sinal, através da interpolação de $L$-1 novas amostras entre cada par de amostras do sinal de entrada $x[n]$. Isto é feito através da inserção de $L$-1 zeros entre cada par de amostras do sinal de entrada $x[n]$. Sendo assim, para uma seqüência de entrada $x[n]$, teremos a seguinte saída (JAHROMI, 2007):

$$x[n] = \begin{cases} x[n/L] & n = 0, \pm L, \pm 2L, \ldots \\ 0 & \text{demais casos} \end{cases} \quad \textbf{(2.28)}$$

O espectro $X_i(e^{jw})$ do sinal interpolado está relacionado com o espectro do sinal original de acordo com a equação (2.29). Isso implica em que o espectro do sinal interpolado é uma versão comprimida por um fator $L$ do espectro do sinal original, apresentando um período de repetição de $2\pi/L$ (FURTADO JÚNIOR, 2006).

$$X_i(e^{jw}) = X(e^{jwL}) \quad \textbf{(2.29)}$$

Para eliminar as imagens replicadas do sinal original (advindas da inserção de zeros entre as amostras do sinal de entrada $x[n]$), torna-se necessário a inserção de um segundo sistema, com o objetivo de filtrar o sinal $y[n]$ através de um filtro Passa-Baixas (*anti-imaging*) com ganho $L$ e freqüência de corte em $\pi/L$. Sendo assim, o sinal voltará a ter o mesmo espectro original, apenas com uma taxa de amostragem $L$ vezes maior (GUEDES, 2002).

A Figura 2.18 apresenta o sistema final de interpolação.

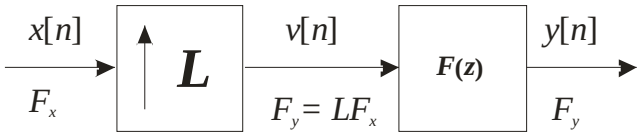


**Figura 2.17 - Operação geral de interpolação.**

### 2.5.3. Identidades Nobres

O processo de filtrar um sinal para posteriormente decimá-lo não é uma abordagem muito eficiente, visto que se filtram amostras que não serão utilizadas, aumentando com isto o tempo de processamento. No caso do interpolador será interessante tê-lo, sempre que possível, após o filtro, pois assim estaremos processando os sinais na taxa mais baixa do sistema (BARCELLOS, 2006). Para resolver estes problemas, têm-se as chamadas Identidades Nobres (Figura 2.19).

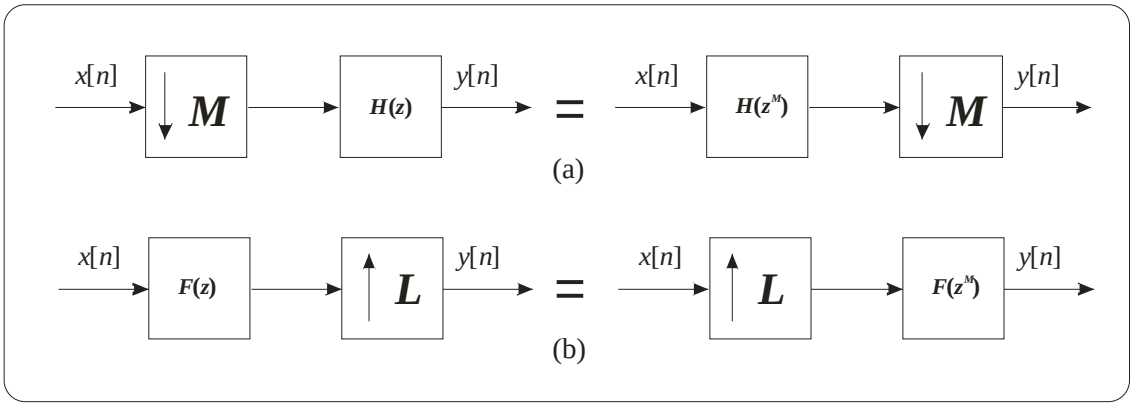


**Figura 2.18 – Identidades nobres: (a) decimação; (b) interpolação.**

A primeira identidade está relacionada à decimação, e estabelece que filtrar um sinal $x[n]$ com um filtro cuja transformada $z$ da resposta impulsiva é $H(z^M)$ e depois decimar o resultado por um fator $M$, é equivalente a decimar o sinal $x[n]$ por um fator $M$ e posteriormente filtrá-lo com um filtro $H(z)$. Sendo assim, é possível aplicar a seguinte relação (MEYER-BAESE, 2007):

$$(\downarrow M)H(z) = H(z^M)(\downarrow M) \quad \textbf{(2.30)}$$

Ou seja, fazendo a decimação antes da filtragem, reduz-se a extensão do filtro $F(z^M)$ por um fator $M$.

A segunda propriedade está relacionada à interpolação e estabelece que interpolar um sinal $x[n]$ por um fator $L$ e filtrá-lo com um filtro cuja resposta é $F(z^L)$, se equivale a filtrar esse mesmo sinal $x[n]$ com um filtro $F(z)$ e posteriormente interpolar o resultado. Sendo assim, é possível aplicar a seguinte relação (MEYER-BAESE, 2007):

$$F(z)(\uparrow L) = (\uparrow L)(F(z^L) \quad \textbf{(2.31)}$$

Ou seja, fazendo a filtragem antes da interpolação, reduz-se a extensão do filtro $F(z^L)$ por um fator $M$.

### 2.5.4. Decomposição polifásica de filtros decimadores e interpoladores

A decomposição polifásica de um filtro $h[n]$ cuja transformada $z$ é dada por $H(z)$ segue de acordo com a equação (2.32). Através desta equação, é possível perceber que o filtro original $H(z)$ foi reestruturado de maneira a ser representado como um somatório de polinômios $E_j(Z^n)$ (componentes polifásicas), $j$=0, ... , $M$-1, cada um com um devido atraso $z^{-j}$ (FURTADO JÚNIOR, 2006) (VAIDYANATHAN, 1993).

$$
\begin{aligned}
H(z) &= \sum_{k=-\infty}^{\infty} h(k) z^{-k} \\
&= \sum_{l=-\infty}^{\infty} h(lM) z^{-lM} + \sum_{l=-\infty}^{\infty} h(lM+1) z^{-(lM+1)} + \ldots \\
&\quad + \sum_{l=-\infty}^{\infty} h(lM+M-2) z^{-(lM+M-2)} + \sum_{l=-\infty}^{\infty} h(lM+M-1) z^{-(lM+M-1)} \\
&= \sum_{l=-\infty}^{\infty} h(lM) z^{-lM} + z^{-1} \sum_{l=-\infty}^{\infty} h(lM+1) z^{-(lM)} + \ldots \\
&\quad + z^{-(M-2)} \sum_{l=-\infty}^{\infty} h(lM+M-2) z^{-(lM)} + z^{-(M-1)} \sum_{l=-\infty}^{\infty} h(lM+M-1) z^{-(lM)} \\
&= \sum_{j=0}^{M-1} z^{-j} E_j(z^M).
\end{aligned}
\tag{2.32}
$$

onde o termo $E_j(z)$ em (2.32) é definido por:

$$
E_j(z) = \sum_{l=-\infty}^{\infty} h(lM + j) z^{-1} \tag{2.33}
$$

Esta decomposição do filtro $H(z)$ em $M$ subfiltros com resposta $E_j(z^M)$, $j$=0, ..., $M$-1, permite que a simples operação de filtragem seguida de decimação seja representada de uma nova maneira. A Figura 2.20(a) representa a operação básica de filtragem seguida de decimação. A Figura 2.20(b) representa a mesma operação, porém vista a partir das componentes polifásicas de $H(z)$, e finalmente, a Figura 2.20(c) representa esta mesma operação, aplicando a identidade nobre da decimação.

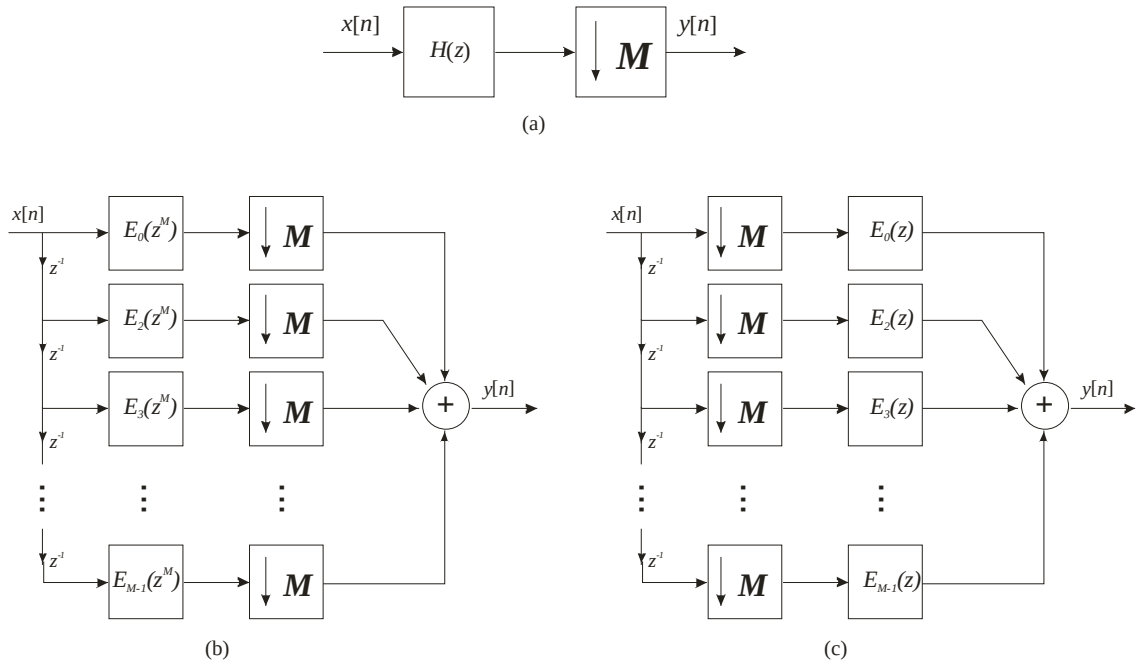


**Figura 2.19 – (a) Decimação por um fator *M*. (b) Decimação utilizando decomposições polifásicas. (c) Decimação utilizando decomposições polifásicas e identidade nobre.**

Destaca-se que, a partir da estrutura representada na Figura 2.20(c), somente amostras $x[n]$ que são múltiplas de $M$ mais um inteiro $k$ são filtradas pelo filtro de componente polifásica $E_k(z)$, k=0, ..., $N$-1 (FURTADO JUNIOR, 2006) (OPPENHEIM, 1999). Ou seja:

$$h_k[n] = \begin{cases} h[n+k], & n = \text{múltiplo inteiro de } M, \\ 0, & \text{demais casos} \end{cases} \qquad \textbf{(2.34)}$$

Sendo assim, cada subfiltro recebe como entrada uma versão deslocada (atrasada) e decimada do sinal $x[n]$, e nenhuma amostra desse sinal servirá de entrada para mais do que um único ramo da estrutura.

Já no caso da interpolação, a decomposição polifásica do filtro $F(z)$ que sucede o interpolador (Figura 2.21(b)) é feita através da definição das componentes polifásicas $R_j(z)=E_{L-1-j}(z)$. Sendo assim, a decomposição polifásica de $F(z)$ se torna (DINIZ, 2004):

$$F(z) = \sum_{j=0}^{L-1} z^{-(L-1-j)} R_j(z^L) \qquad \textbf{(2.35)}$$

Igualmente ao caso da decimação, o bloco interpolador pode ser reorganizado em uma estrutura de *L* ramos. A Figura 2.21(a) representa a operação básica de interpolação. A Figura 2.21(b) representa a mesma operação, porém vista a partir das componentes polifásicas de *F*(*z*), e finalmente, a Figura 2.21(c) representa esta mesma operação, aplicando a identidade nobre da interpolação.

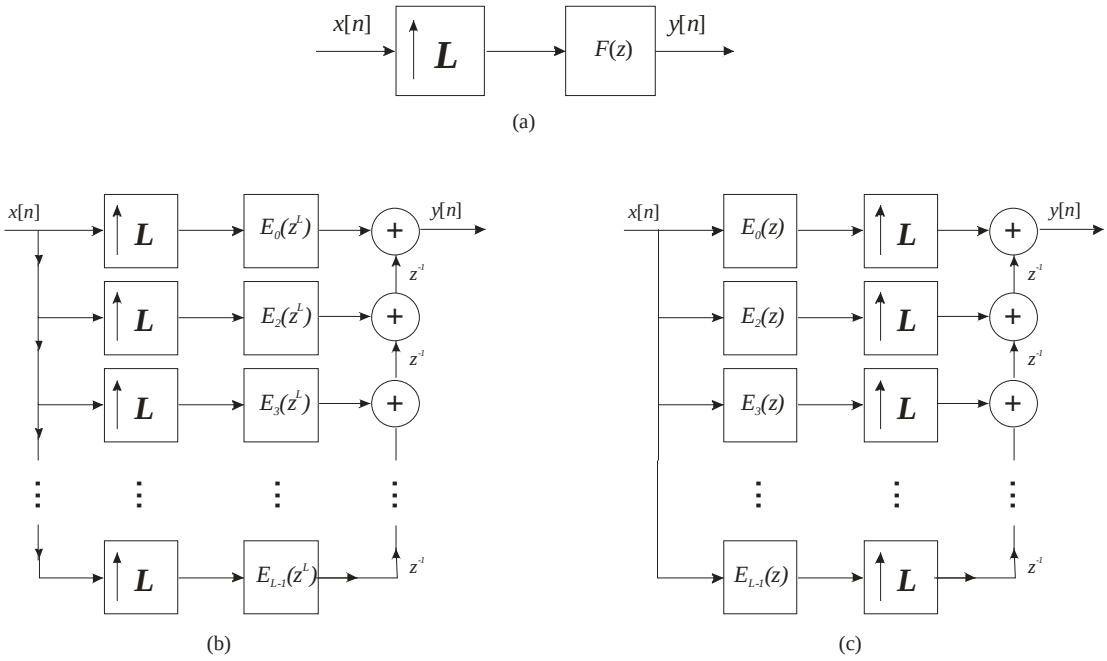


**Figura 2.20 - (a) Interpolação por um fator *L*. (b) Interpolação utilizando decomposições polifásicas. (c) Interpolação utilizando decomposições polifásicas e identidade nobre.**

## 2.6. A estrutura de reverberação de Moorer

Segundo Dattorro (1997), a escolha por determinada arquitetura de reverberação é simplesmente uma questão gosto, assim como acontece com as obras de arte. Dattorro afirma que não existe um reverberador universal que satisfaça a todas as pessoas e aplicações. Ele especula também que provavelmente, esse reverberador universal nunca existirá (DATTORRO, 1997).

Para o projeto em questão, a arquitetura proposta por James A. Moorer (1979) foi escolhida como base de desenvolvimento e comparação. A arquitetura proposta por Moorer abrange todos os estágios que compõem o efeito de reverberação: o Retardo Inicial, as Reflexões Primárias e as Reflexões Tardias, somadas ao Som Direto.

Tendo em vista que a proposta deste trabalho consiste na implementação da arquitetura de Moorer (referência) e na implementação de uma proposta de otimização para tal arquitetura, esta seção apresenta a arquitetura de reverberação proposta por Moorer de forma detalhada, com base nos parâmetros propostos pelo mesmo. Na Figura 2.22, é possível observar o diagrama de blocos simplificado da estrutura de reverberação de Moorer, simulando o espaço acústico de um auditório. Como a arquitetura de referência utiliza uma freqüência de amostragem de 44,1 kHz, todos os cálculos são realizados tendo como base esta freqüência.

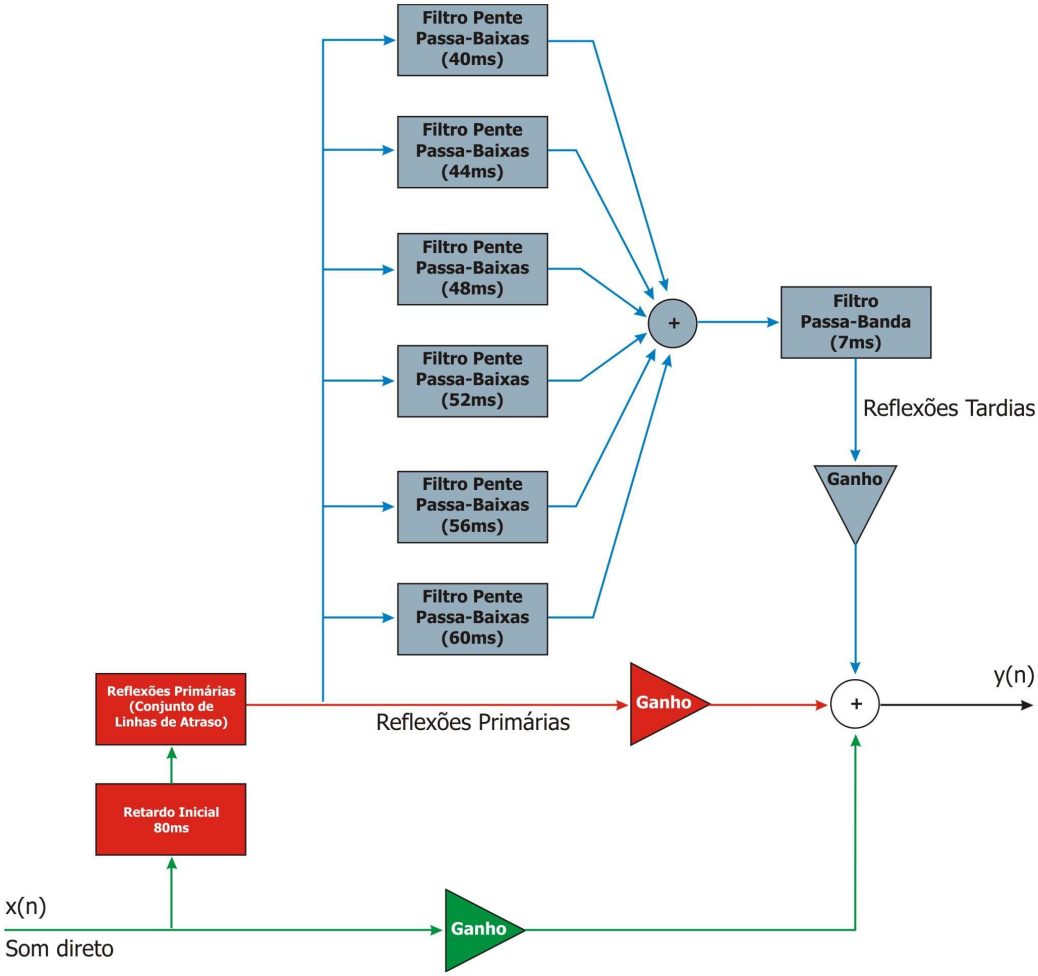


**Figura 2.21 - Estrutura de reverberação digital proposta por James A. Moorer.**

### 2.6.1. O Retardo Inicial e as Reflexões Primárias

Com o objetivo de simular o Retardo Inicial (ITD), Moorer propôs a utilização de um *buffer* com 3520 posições de memória, gerando um atraso de 80 milissegundos. Este sinal atrasado é enviado a um conjunto de linhas de atraso com diferentes coeficientes de atraso e de ganho, produzindo então as Reflexões Primárias.

Para a simulação das Reflexões Primárias, Moorer propôs as seguintes configurações (Tabela 2.4):

**Tabela 2.4 – Configurações propostas por Moorer para a simulação das Reflexões Primárias.**

| Tamanho do *Buffer* | Tempo de Atraso (ms) | Coeficiente de Ganho |
|---|---|---|
| 190 | 4,31 | 0,841 |
| 759 | 17,21 | 0,504 |
| 44 | 0,99 | 0,490 |
| 190 | 4,31 | 0,379 |
| 9 | 0,20 | 0,380 |
| 123 | 2,79 | 0,346 |
| 706 | 16 | 0,289 |
| 119 | 2,70 | 0,272 |
| 384 | 8,71 | 0,192 |
| 66 | 1,50 | 0,193 |
| 35 | 0,79 | 0,217 |
| 75 | 1,70 | 0,181 |
| 419 | 9,50 | 0,180 |
| 4 | 0,09 | 0,181 |
| 79 | 1,79 | 0,176 |
| 66 | 1,50 | 0,142 |
| 53 | 1,20 | 0,167 |
| 194 | 1,40 | 0,134 |

A Figura 2.23 apresenta o diagrama de blocos das Reflexões Primárias, de acordo com as configurações propostas por Moorer.

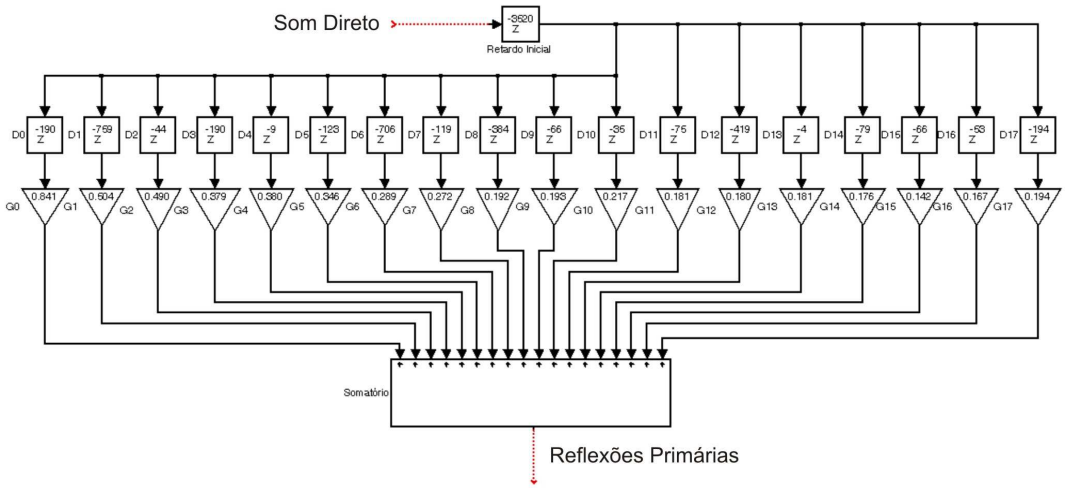


**Figura 2.22 - Diagrama de blocos das Reflexões Primárias proposto por Moorer.**

### 2.6.2. As Reflexões Tardias

Para a simulação da reverberação propriamente dita (Reflexões Tardias), Moorer propôs a utilização de 6 (seis) filtros Pente Passa-Baixas interligados paralelamente, visando aumentar a densidade da resposta. O filtro Pente Passa-Baixas proposto por Moorer consiste simplesmente na adição de um filtro Passa-Baixas na saída de um filtro Pente, antes de sua realimentação. A Figura 2.24 ilustra o Pente Passa-Baixas proposto por Moorer.

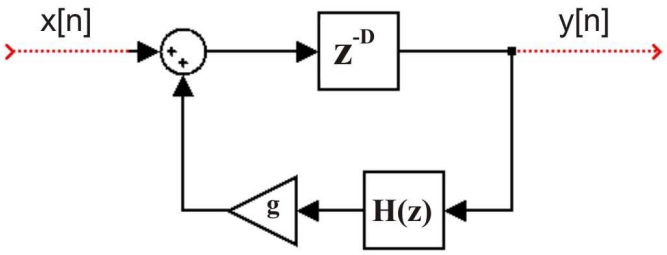


**Figura 2.23 – Filtro Pente Passa-Baixas proposto por Moorer.**

Os filtros Passa-Baixas introduzidos em cada filtro Pente reduzem os sons metálicos cortando o tempo de reverberação das altas freqüências (simulando o efeito de um auditório). A principal idéia por traz da adição destes filtros é simular o efeito da absorção das altas freqüências pelo ar (MOORER, 1979). De acordo com Moorer, a resposta destes filtros não reproduz exatamente estas propriedades, porém, suas configurações foram escolhidas visando o compromisso com a eficiência, visto que somente uma multiplicação e uma adição são necessárias.

Seguindo a estrutura proposta por Moorer para as Reflexões Tardias, o sinal resultante da saída dos filtros Pente Passa-Baixas é enviada a um filtro Passa-Tudo (Figura 2.25), cuja saída é somada ao sinal original (MOORER, 1979) (KUO, 2001). O filtro Passa-Tudo é utilizado para adicionar um pouco mais de realismo ao efeito através das variações de fase proporcionada pelo mesmo (MOORER, 1979).

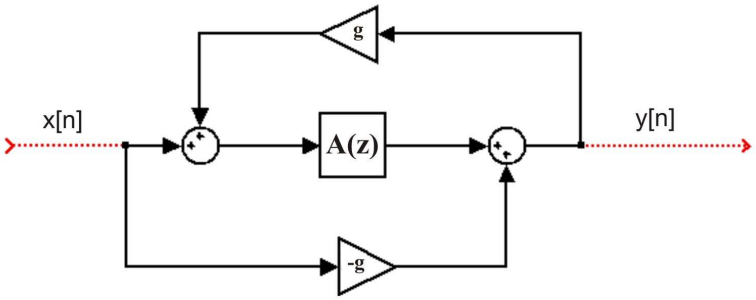


**Figura 2.24 – Diagrama de blocos do filtro Passa-Tudo.**

A Tabela 2.5 apresenta as configurações propostas por Moorer para a configuração do conjunto de filtros Pente Passa-Baixas. A Tabela 2.6 apresenta as configurações para o filtro Passa-Tudo.

**Tabela 2.5 – Configurações propostas por Moorer para a configuração do conjunto de filtros Pente Passa-Baixas.**

| Filtro | Tamanho do *Buffer* | Tempo de Atraso (ms) | Coeficiente de Ganho (Passa-Baixas) | Coeficiente de Ganho (Pente) |
|---|---|---|---|---|
| Pente Passa-Baixas 1 | 1759 | 40 | 0,941 | 0,46 |
| Pente Passa-Baixas 2 | 1949 | 44 | 0,818 | 0,48 |
| Pente Passa-Baixas 3 | 2113 | 48 | 0,635 | 0,50 |
| Pente Passa-Baixas 4 | 2293 | 52 | 0,719 | 0,52 |
| Pente Passa-Baixas 5 | 2467 | 56 | 0,267 | 0,53 |
| Pente Passa-Baixas 6 | 2647 | 60 | 0,242 | 0,55 |

**Tabela 2.6 – Configurações propostas por Moorer para o filtro Passa-Tudo.**

| Filtro | Tamanho do *Buffer* | Tempo de Atraso (ms) | Coeficiente de Ganho |
|---|---|---|---|
| Passa-Tudo | 307 | 7 | 0,7 |

A Figura 2.26 ilustra o diagrama de blocos completo do sistema que simula as Reflexões Tardias, de acordo com a proposta de Moorer.

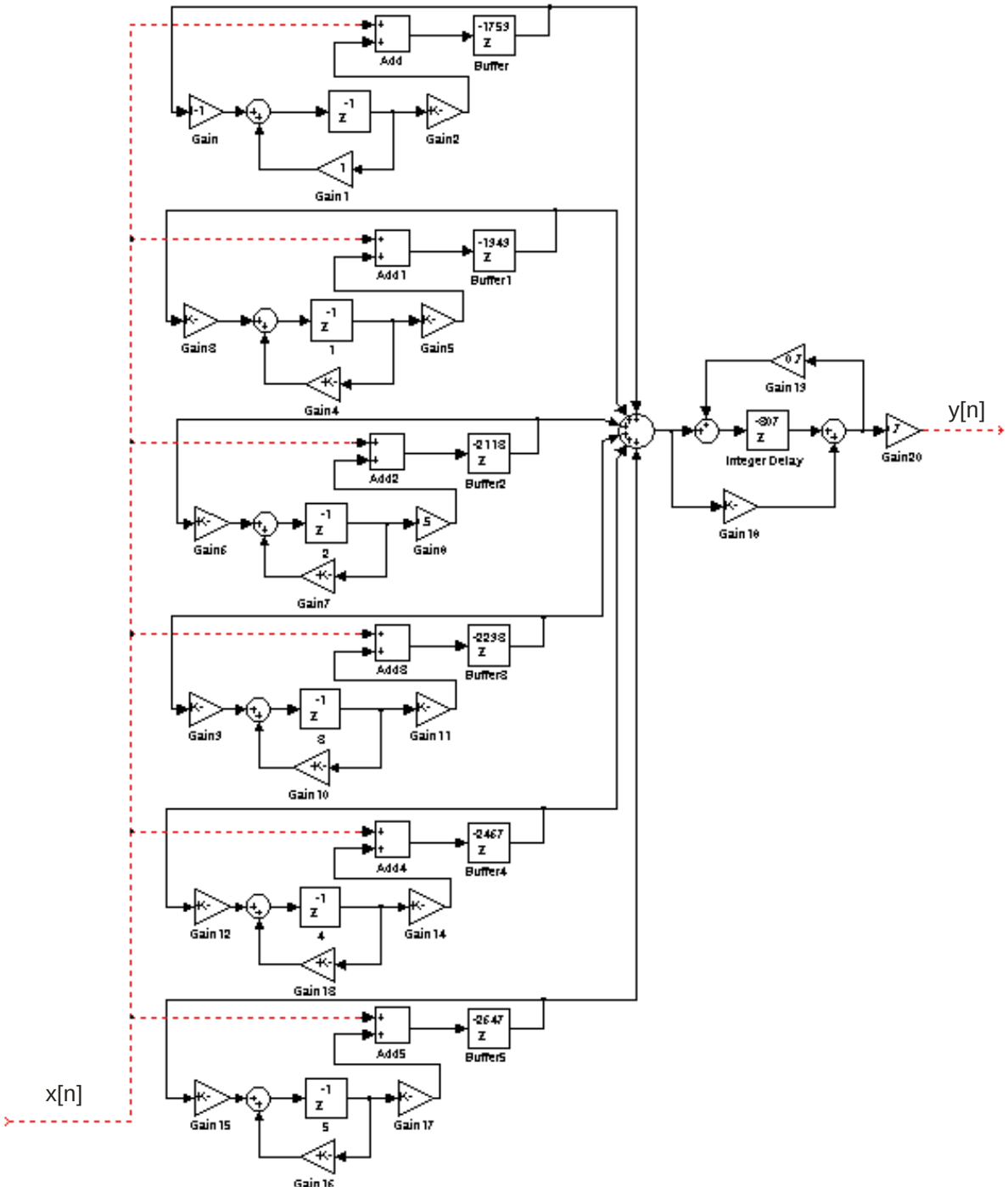


**Figura 2.25 – Diagrama de blocos das Reflexões Tardias de acordo com a proposta de Moorer.**

### 2.6.3. A arquitetura final

Através da interconexão das partes responsáveis pela simulação das Reflexões Primárias, Reflexões Tardias e Som Direto, obtém-se a arquitetura final proposta por Moorer

(Figura 2.27). Observando esta arquitetura, é possível perceber que cada estágio do sistema (Reflexões Primárias, Reflexões Tardias e Som Direto) possui um controle de ganho individual. Variando os coeficientes de ganho de cada estágio, diferentes possibilidades de efeito podem ser experimentadas.

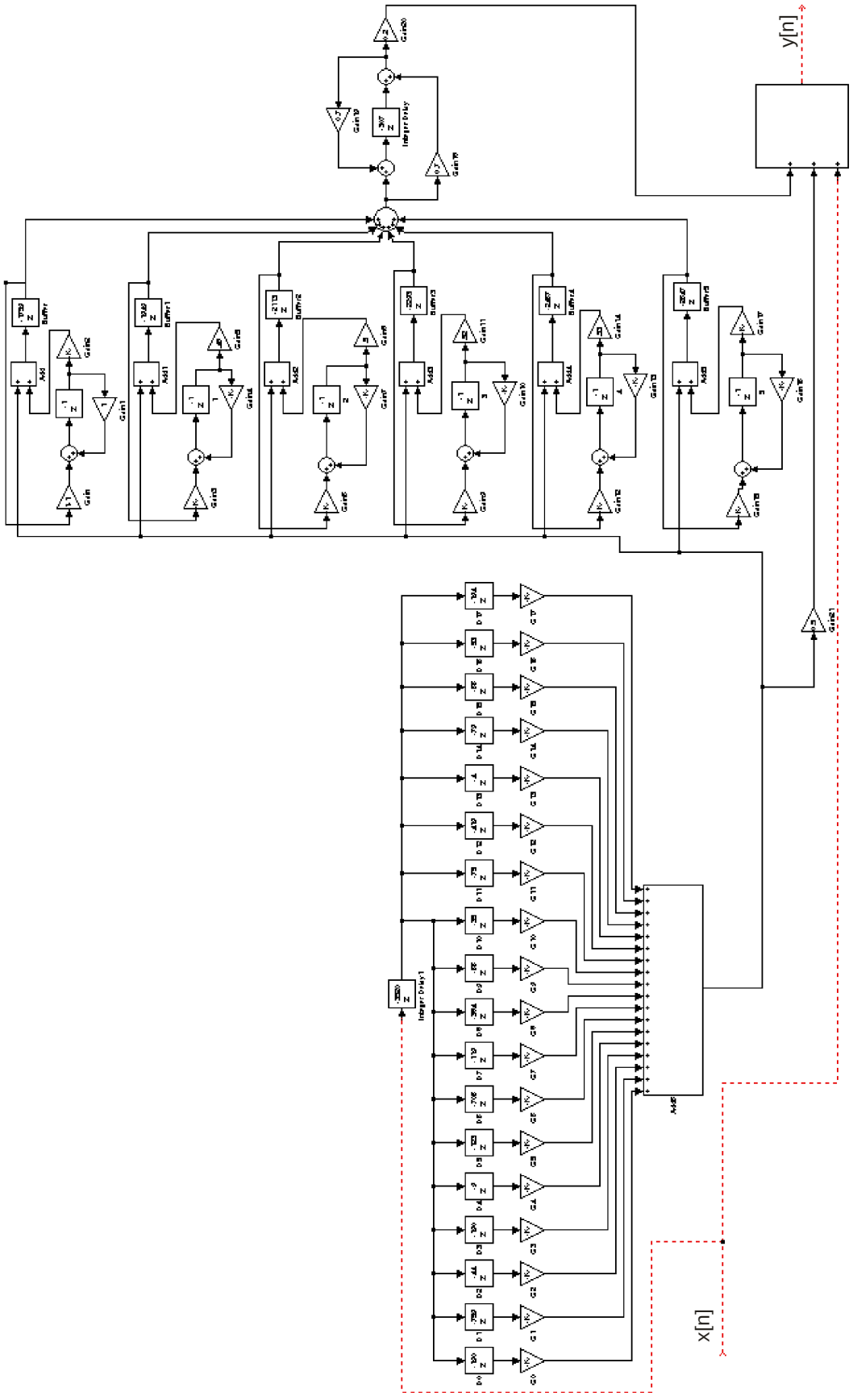


**Figura 2.26 – Arquitetura completa do sistema de reverberação digital proposta por Moorer.**

## 2.7. Conclusão

Este capítulo apresentou de forma teórica os conceitos relativos à reverberação, bem como as estruturas básicas de processamento de sinais utilizadas na construção de algoritmos de reverberação digital. Os conceitos de processamento multitaxa também foram apresentados, bem como a arquitetura de reverberação proposta por James A. Moorer que é utilizada como base de referência neste trabalho.

# 3. SISTEMA PROPOSTO E METODOLOGIA

## 3.1. Introdução

Conforme apresentado brevemente no capítulo introdutório, a proposta deste trabalho consiste no desenvolvimento de uma arquitetura de *hardware* otimizada através do uso de técnicas de processamento multitaxa, específica para a implementação de um algoritmo de reverberação digital utilizando dispositivos lógicos programáveis (FPGA) como ambiente de pesquisa e prototipação. Com o foco centrado na redução do consumo de memória e de área, a proposta em questão apresenta-se como uma nova forma de gerenciar os recursos computacionais exigidos pelos algoritmos de efeito de áudio baseados em atraso de tempo. Dentre os efeitos de áudio que utilizam esta metodologia, o efeito de reverberação é sem dúvida o mais importante. Esta importância se dá devido ao fato de ser este o mais ocorrente e conhecido fenômeno da acústica (VALLE, 2006). Outro fator motivador desta escolha deve-se ao fato de que praticamente todos os demais efeitos de áudio têm sua base de desenvolvimento nos algoritmos empregados na implementação do *Reverb* (SEHN, 2004).

Este capítulo apresenta detalhadamente o sistema proposto, bem como a metodologia utilizada para implementação do mesmo.

## 3.2. A Proposta de otimização

Conforme apresentado no Capítulo 2 (Equação 2.23), a quantidade de posições de memória necessária para gerar atrasos de tempo (princípio básico da reverberação digital) está vinculada a freqüência de amostragem do sistema. Sendo assim, o trabalho em questão propõe a utilização de técnicas de processamento multitaxa, tendo como objetivos a redução do consumo de memória e de área física despendidos pela arquitetura.

O processo de conversão da taxa de amostragem (processamento multitaxa) de um sinal é um tipo de processamento que utiliza certos operadores que alteram a taxa de amostragem de determinado sinal, a partir da remoção ou adição de amostras na seqüência de

entrada original. A mudança da freqüência de amostragem tem sido muito utilizada em DSPs e aplicações de áudio. Dependendo da aplicação, a mudança da freqüência de amostragem pode reduzir consideravelmente a complexidade dos algoritmos e do *hardware* ou aumentar a resolução em determinadas operações de processamento de sinais através da introdução de novas amostras de sinais (SPANIAS, 2007). Os blocos básicos de um sistema de processamento de sinais multitaxa são os decimadores (redução da taxa de amostragem) e os interpoladores (aumento da taxa de amostragem).

Como o efeito de reverberação digital se baseia em grandes linhas de atraso cujo tamanho é proporcional à freqüência de amostragem, e as técnicas de processamento multitaxa possibilitam a redução desta freqüência, visualiza-se então a redução da quantidade de memória necessária para a implementação do efeito em questão. Reduzindo a quantidade de memória utilizada pelo sistema, conseqüentemente reduz-se a área necessária para sua implementação física, possibilitando então o emprego da arquitetura um *chip* único (*single-chip*) de baixo custo com área reduzida.

Neste sentido, a proposta deste trabalho consiste em adicionar um sistema decimador na entrada de uma arquitetura de reverberação digital referencial (Moorer), decimando o sinal de entrada antes que o mesmo entre na arquitetura de reverberação. Após passar pela arquitetura de reverberação (com uma taxa de amostragem mais baixa), o sinal é enviado a um sistema interpolador que, adicionado a saída da arquitetura, tem a função de restabelecer a freqüência de amostragem original. Portanto, reduzindo a freqüência de amostragem do sistema de reverberação, conseqüentemente têm-se uma redução da quantidade de memória necessária para a implementação do mesmo. Para este trabalho, optou-se pela utilização de um fator de decimação e interpolação igual a 2 (dois). A escolha deste fator deu-se baseada no compromisso com a qualidade sonora. Fatores mais elevados poderiam ser utilizados, porém, isto resultaria em uma possível perda de qualidade.

De acordo com o Teorema da Amostragem (Nyquist), a taxa de amostragem deve ser de pelo menos duas vezes a maior freqüência que se deseja registrar (OPPENHEIM, 1999). Sendo assim, tendo em vista que a arquitetura de referência trabalha a uma freqüência de 44,1 kHz, a freqüência mais alta registrada por esta amostragem é 22,05 kHz. Considerando que a proposta deste trabalho prevê um fator de decimação/interpolação igual a 2 (dois), a freqüência de amostragem da arquitetura de reverberação passa a ser 22,05 kHz, e a maior freqüência registrada pela arquitetura é 11,025 kHz.

Conforme discutido no Capítulo 2, a inserção de filtros *anti-aliasing* e *anti-image* torna-se necessária para garantir a integridade do sinal. Sendo assim, a Figura 3.1 apresenta o diagrama de blocos simplificado da solução proposta.

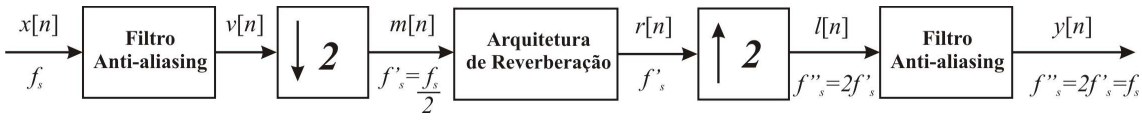


**Figura 3.1 – Diagrama de blocos da arquitetura proposta.**

Para o sistema de decimação, a implementação polifásica utilizando a identidade nobre apresentada no Capitulo 2 (dois) foi empregada. Esta escolha se deu devido ao fato de ser esta a implementação mais eficiente (FURTADO JUNIOR, 2006) (OPPENHEIM, 1999) (VAIDYANATHAN, 1990), visto que o fato de filtrar um sinal para posteriormente decimá-lo aumenta o tempo de processamento devido ao fato de se filtrar amostras que não serão utilizadas. Sendo assim, fazendo a decimação antes da filtragem aplicando a decomposição polifásica, reduz-se a extensão do filtro por um fator *M*.

Com relação ao sistema de interpolação, igualmente ao sistema de decimação, a implementação polifásica utilizando a identidade nobre apresentada no Capitulo 2 (dois) também foi utilizada. Esta abordagem também é considerada a mais eficiente, visto que os sinais serão processados na taxa mais baixa do sistema. Sendo assim, fazendo a filtragem antes da interpolação, reduz-se a extensão do filtro por um fator *M*.

Após as considerações supracitadas, tem-se então o novo diagrama de blocos do sistema proposto (Figura 3.2)

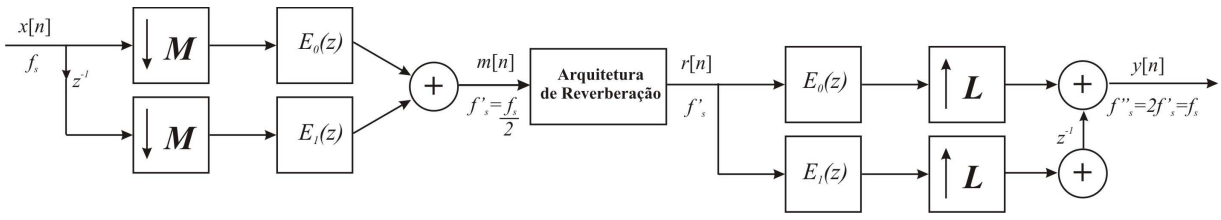


**Figura 3.2 – Diagrama de blocos da arquitetura proposta, utilizando a decomposição polifásica.**

### 3.2.1. As Reflexões Primárias

Conforme apresentado no Capítulo 2 (Seção 2.6.1), Moorer propôs a utilização de uma linha de atraso com 3520 posições de memória (80 milissegundos) para a simulação do ITD, sendo que este sinal atrasado é enviado a um conjunto de linhas de atraso com diferentes coeficientes de atraso e de ganho. Porém, sua arquitetura é baseada em uma freqüência de amostragem de 44,1 kHz.

Como a proposta de otimização faz uso da decimação por um fator 2 (dois), têm-se então a freqüência de amostragem de 22,05 kHz. Sendo assim, fazendo uso da equação (2.4.8), obtém-se a nova tabela de configurações para as Reflexões Primárias (Tabela 3.1). Da mesma forma, levando em conta a nova freqüência da amostragem, têm-se para a simulação do ITD um total de 1760 posições de memória para gerar os mesmos 80 milissegundos propostos por Moorer.

**Tabela 3.1 – Novas configurações para a simulação das Reflexões Primárias.**

| **Tamanho do *Buffer*** | **Tempo de Atraso (ms)** | **Coeficiente de Ganho** |
|---|---|---|
| 95 | 4,31 | 0,841 |
| 380 | 17,21 | 0,504 |
| 22 | 0,99 | 0,490 |
| 95 | 4,35 | 0,379 |
| 5 | 0,20 | 0,380 |
| 62 | 2,79 | 0,346 |
| 353 | 16 | 0,289 |
| 60 | 2,70 | 0,272 |
| 192 | 8,71 | 0,192 |
| 33 | 1,50 | 0,193 |
| 18 | 0,79 | 0,217 |
| 38 | 1,70 | 0,181 |
| 210 | 9,50 | 0,180 |
| 2 | 0,09 | 0,181 |
| 40 | 1,79 | 0,176 |
| 33 | 1,50 | 0,142 |
| 27 | 1,20 | 0,167 |
| 97 | 1,40 | 0,134 |

### 3.2.2. As Reflexões Tardias

Conforme citado no Capitulo 2 (Seção 2.6.2), a arquitetura de Moorer utiliza um banco de filtros composto de seis filtros Pente Passa-Baixas interligados paralelamente, sendo que a saída destes filtros é enviada a um filtro Passa-Tudo cuja saída é somada ao sinal original.

Com a freqüência de amostragem reduzida pelo processo de decimação, novas configurações devem ser adotadas para os filtros utilizados na simulação das Reflexões Tardias. A Tabela 3.2 apresenta as novas configurações para o conjunto de filtros Pente Passa-Baixas. A Tabela 3.3 apresenta as novas configurações para o filtro Passa-Tudo.

**Tabela 3.2 – Novas configurações o conjunto de filtros Pente Passa-Baixas.**

| Filtro | Tamanho do *Buffer* | Tempo de Atraso (MS) | Coeficiente de Ganho (Passa-Baixas) | Coeficiente de Ganho (Pente) |
|---|---|---|---|---|
| Pente Passa-Baixas 1 | 880 | 40 | 0,941 | 0,46 |
| Pente Passa-Baixas 2 | 975 | 44 | 0,818 | 0,48 |
| Pente Passa-Baxias 3 | 1057 | 48 | 0,635 | 0,50 |
| Pente Passa-Baixas 4 | 1147 | 52 | 0,719 | 0,52 |
| Pente Passa-Baixas 5 | 1234 | 56 | 0,267 | 0,53 |
| Pente Passa-Baixas 6 | 1324 | 60 | 0,242 | 0,55 |

**Tabela 3.3 – Novas configurações para o filtro Passa-Tudo.**

| Filtro | Tamanho do *Buffer* | Tempo de Atraso (ms) | Coeficiente de Ganho |
|---|---|---|---|
| Passa-Tudo | 154 | 7 | 0,7 |

## 3.3. Metodologia

Buscando validar a proposta deste trabalho, um estudo detalhado sobre os principais algoritmos de reverberação digital foi realizado, visando obter uma arquitetura de referência que pudesse ser tomada como base de desenvolvimento e comparação. Das diversas arquiteturas analisadas, concluiu-se que a arquitetura de Moorer seria a escolha ideal. A opção

por esta arquitetura se deu devido ao fato de ser esta o fruto de um aperfeiçoamento feito através da reciclagem de diversas arquiteturas anteriores. Outro fator motivador foi a documentação técnica disponível sobre esta arquitetura, bem como a qualidade sonora apresentada pela mesma (BELTRÁN, 2007).

Após definida a arquitetura a ser utilizada como estudo de caso, partiu-se então para a implementação da mesma. Primeiramente uma abordagem baseada em *software* foi construída sobre uma plataforma x86 (*Windows XP*), com auxílio da ferramenta MATLAB™ Simulink™. O intuito da construção de tal versão era possuir um parâmetro para comparações durante o desenvolvimento da arquitetura em *hardware*. O ambiente MATLAB™ Simulink™ fornece uma plataforma de desenvolvimento de fácil manipulação, onde resultados podem ser obtidos em um curto espaço de tempo. Porém, este ambiente não pode ser considerado levando em conta sistemas de tempo real. Sendo assim, a implementação em *software* foi utilizada somente para comparações durante o desenvolvimento do trabalho em questão.

Com a implementação em *software* concluída e validada, partiu-se então para o desenvolvimento do sistema como arquitetura dedicada. O fato de se buscar uma arquitetura dedicada vem da necessidade de processamento em tempo real. Como os efeitos de áudio baseados em atrasos de tempo necessitam de um constante acesso à memória, um sistema eficiente para gerenciar este comportamento se torna indispensável. Sendo assim, optou-se pela utilização de *Field Progammable Gate Arrays* (FPGAs) como ambiente de prototipação. Ao longo do tempo, os FPGAs vêm aumentando consideravelmente sua performance, bem como quantidade de recursos disponibilizados ao projetista. Portanto, o alto desempenho e a flexibilidade oferecidos pelos FPGAs fornecem um cenário ideal para a pesquisa e o desenvolvimento de produtos na área de áudio digital, justificando então esta escolha.

No que diz respeito à plataforma de desenvolvimento, o *software* Quartus™ II foi utilizado em conjunto com o *kit* de desenvolvimento DE2 da Altera™ , onde encontra-se o FPGA Cyclone™ II EP2C35F6726. Com relação à linguagem de descrição de *hardware*, optou-se pela utilização do VHDL (*VHSIC Hardware Description Language*). Visto que não é escopo deste trabalho apresentar diferenças de implementação entre arquiteturas reconfiguráveis, destaca-se que qualquer outro FPGA poderia ser utilizado.

Considerando que a proposta deste trabalho está centrada na otimização de uma arquitetura já existente (Moorer), o processo de desenvolvimento se deu em duas etapas: primeiramente, a arquitetura de Moorer foi construída sem nenhuma modificação. Posteriormente, após a validação da arquitetura de referência, partiu-se então para o

desenvolvimento da arquitetura proposta, empregando as técnicas de processamento multitaxa.

## 3.4. Conclusão

Este capítulo apresentou de forma teórica os conceitos necessários para a implementação da arquitetura de reverberação digital proposta, bem como a metodologia utilizada para a implementação desta arquitetura. Os conceitos para a incorporação dos sistemas decimador/interpolador também foram apresentados, bem como sua aplicação na proposta de otimização. De acordo com esta proposta, visualiza-se então a redução do consumo de memória bem como a redução de área física ocupada pelo sistema.

# 4. IMPLEMENTAÇÃO E RESULTADOS

## 4.1. Introdução

Este capítulo descreve em detalhes a implementação das arquiteturas de reverberação digital, visando demonstrar a possibilidade de desenvolvimento de algoritmos de efeitos de áudio empregando técnicas de processamento multitaxa sobre plataforma reconfigurável. Sendo assim, os blocos implementados e a descrição funcional do sistema são apresentados, bem como os resultados parciais referentes às implementações.

A implementação completa da arquitetura se deu em duas etapas. A primeira etapa consistiu na implementação da arquitetura proposta por Moorer sem nenhuma alteração, visando a obtenção de uma base para comparações. Na segunda etapa, a arquitetura proposta foi desenvolvida conforme a metodologia apresentada no Capítulo 3.

## 4.2. Ambiente de desenvolvimento

Para a implementação das arquiteturas, o *software* Quartus™ II foi utilizado em conjunto com o *kit* de desenvolvimento DE2 da Altera™, sendo que neste encontra-se o FPGA Cyclone™ II EP2C35F6726. As principais características do FPGA em questão são (ALTERA, 2008):

- 33.216 Elementos Lógicos (LEs);
- 35 18x18 (ou 70 9x9) multiplicadores;
- 483.840 bits de memória, divididos em 105 MK4 blocos;
- 475 pinos de entrada e saída;
- 4 PLLs.

## 4.3. Os módulos de *hardware*

A arquitetura de *hardware* do sistema de reverberação é composta por um conjunto de cinco módulos básicos (DC, UA, CG, FP_PBX, F_PTD) e quatro módulos principais (REFPRI, REFTAR, DEC e INT). Dentre as tarefas realizadas pelo conjunto de módulos básicos estão a geração de *clock*, geração de atrasos de tempo, controle de ganho e filtragem. Já os blocos principais são responsáveis pela decimação (DEC), interpolação (INT) e geração das etapas de reverberação através da utilização dos diversos módulos básicos, sendo o módulo REFPRI responsável pela simulação das Reflexões Primárias e o módulo REFTAR responsável pela simulação das Reflexões Tardias.

### 4.3.1. O conjunto de módulos básicos

No intuito de prover funções de controle e filtragem, bem como possibilitar a validação do sistema, inicialmente o conjunto de blocos ilustrado na Figura 4.1 foi construído e validado. Tal conjunto é composto por um circuito de geração de *clock* (DC), unidade de atraso (UA), controle de ganho (CG), filtro Pente Passa-Baixas (FP_PBX) e filtro Passa-Tudo (F_PTD).


**Figura 4.1 - Diagrama de blocos do conjunto de módulos básicos.**

Para a implementação do divisor de *clock*, utilizou-se o *Digital Clock Manager* (DCM) disponibilizado como recurso do próprio FPGA. Dentre outras funções, este dispositivo pode derivar diferentes freqüências a partir da multiplicação e divisão de um *clock* principal. Conforme proposto no Capítulo 3, a freqüência principal utilizada para o trabalho em questão é de 44,1 kHz.

O módulo UA (Unidade de Atraso) é um dos elementos básicos do sistema, sendo empregado tanto para a construção de filtros como para a implementação das linhas de atraso utilizadas pelo algoritmo de reverberação. Neste sentido, tendo em *sinal_entrada* o sinal a ser processado, *tam_buffer* define a quantidade de posições de memória utilizada pelo UA, sendo que esta quantidade caracteriza o tempo de atraso.

Descendo um nível na hierarquia do módulo UA (Figura 4.2), torna-se possível a visualização do conteúdo de seus sub-módulos. O principal deles, *Single-clock FIFO* (SCFIFO), é um componente da biblioteca de módulos parametrizados (LPM) fornecido pela Altera™. Com este componente torna-se possível implementar uma FIFO, que desempenha o papel necessário para a construção da unidade de atraso, conforme pode-se observar no esquema da Figura 4.3.

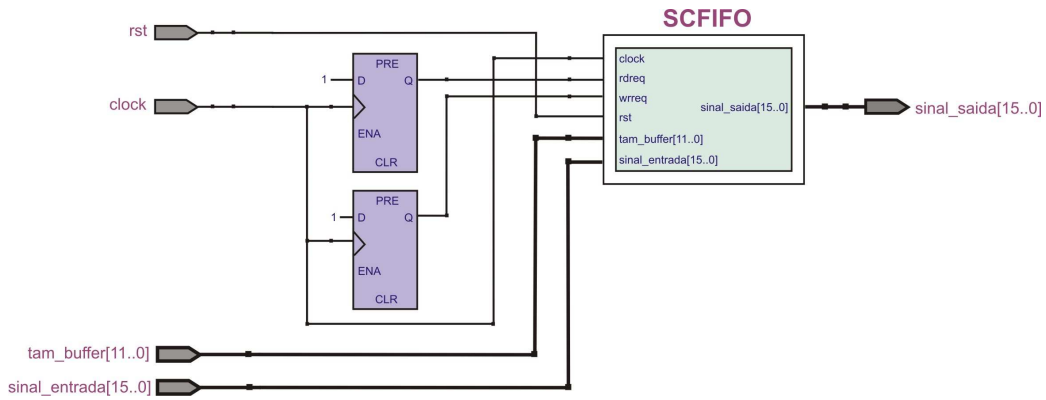


**Figura 4.2 – Diagrama de blocos da unidade de atraso.**

O componente SCFIFO utiliza a memória volátil disponíveis no FPGA, onde parâmetros como tipo de memória (RAM/ROM, acesso simples/duplo), largura de palavra, número de palavras disponíveis, entre outros, podem ser especificados. Sendo assim, para gerenciar o comportamento do componente SCFIFO, uma arquitetura de controle também foi desenvolvida.

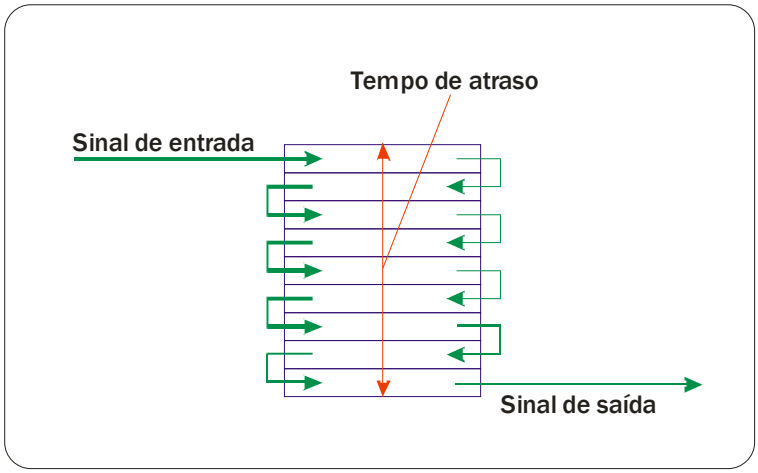


**Figura 4.3 – Unidade de atraso implementada através de uma FIFO.**

No que diz respeito ao módulo CG (Controle de Ganho), o mesmo foi desenvolvido tendo em vista o controle da amplitude do sinal a ser processado. Este módulo, tendo em *sinal_entrada* o sinal a ser processado, multiplica o mesmo pelo fator de redução aplicado na entrada *fator_redução*. Emprega-se o termo "fator de redução" devido ao fato de, para o trabalho em questão, o coeficiente de ganho ser sempre inferior ao valor 1 (um).

O módulo FP_PBX (Filtro Pente Passa-baixas) tem como objetivo executar a filtragem do sinal, de acordo com a arquitetura proposta por Moorer (Seção 2.6), permitindo a passagens das componentes harmônicas de valor mais baixo. Tendo em *sinal_entrada* o sinal a ser processado, *coeficiente_pente* e *coeficiente_pb* recebem os coeficientes de ganho propostos por Moorer para os filtros Pente Passa-Baixas e Passa-Tudo respectivamente, e *tam_buffer* recebe o tamanho do *buffer* (número de endereços) para o filtro Pente. Estes valores estão armazenados em uma ROM (ROM_RT) (Seção 4.3.2).

Da mesma maneira, o módulo F_PTD (Filtro Passa-Tudo) executa a filtragem do sinal, de acordo com a proposta de Moorer. Tendo em *Sinal_entrada* o sinal a ser processado, *coeficiente_pb* recebe o valor de ganho, e *tam_buffer* recebe o tamanho do *buffer*, que também se encontra armazenado em ROM_RT.

### 4.3.2. O conjunto de módulos principais

Os módulos REFPRI (Reflexões Primárias), REFTAR (Reflexões Tardias), DEC (Decimador), INT (Interpolador), ROM_RP (ROM das Reflexões Primárias) e ROM_RT (ROM das Reflexões Tardias) formam o conjunto de módulos principais (Figura 4.4).

No que diz respeito ao módulo REFPRI, o mesmo foi construído através da interligação dos módulos básicos descritos anteriormente e de um somador fornecido pela Altera™ através de suas bibliotecas de mega-funções. Sendo assim, a arquitetura apresentada no Capítulo 3 (Seção 3.3.5.1) pode ser implementada.

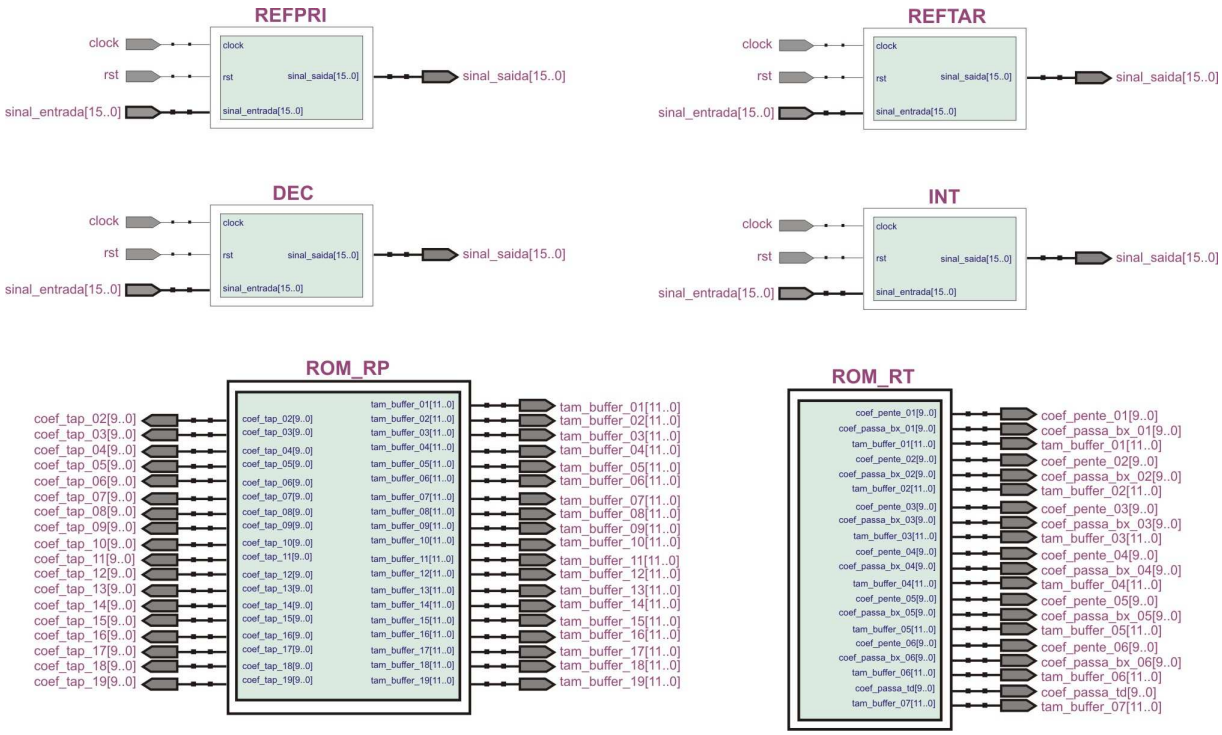


**Figura 4.4 – Diagrama de blocos do conjunto de módulos principais.**

Descendo um nível na hierarquia do módulo REFPRI, é possível observar a maneira como os sub-módulos foram instanciados (Figura 4.5).

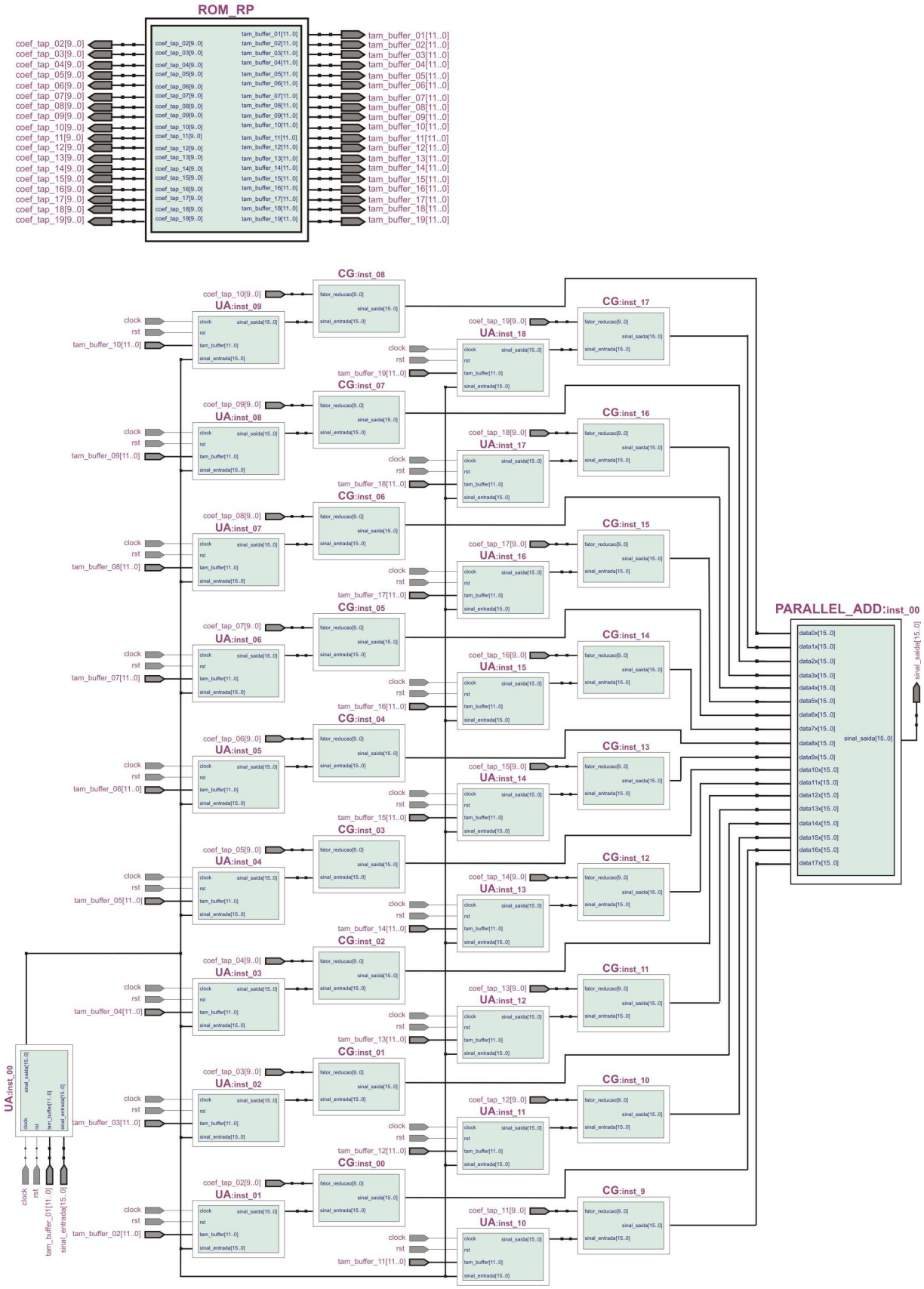


**Figura 4.5 – Diagrama de blocos dos sub-módulos do módulo principal REFPRI.**

O módulo ROM_RP é responsável por alimentar as configuração dos parâmetros associados a cada sub-módulo de REFPRI, onde *tam_buffer_xx[]* contém as informações referentes ao tempo de atraso de cada módulo UA, e *coef_tap_xx[]* contém as informações referentes ao ganho de cada UA.

Já o módulo REFTAR, conforme teoria apresentada no Capítulo 3 (Seção 3.2.2), foi implementado mediante a utilização de seis filtros Pente Passa-baixas paralelos, com sua saída enviada a um filtro Passa-Tudo. Descendo um nível na hierarquia de REFTAR, tem-se o diagrama de blocos ilustrado na Figura 4.6, apresentando como os sub-módulos foram instanciados.

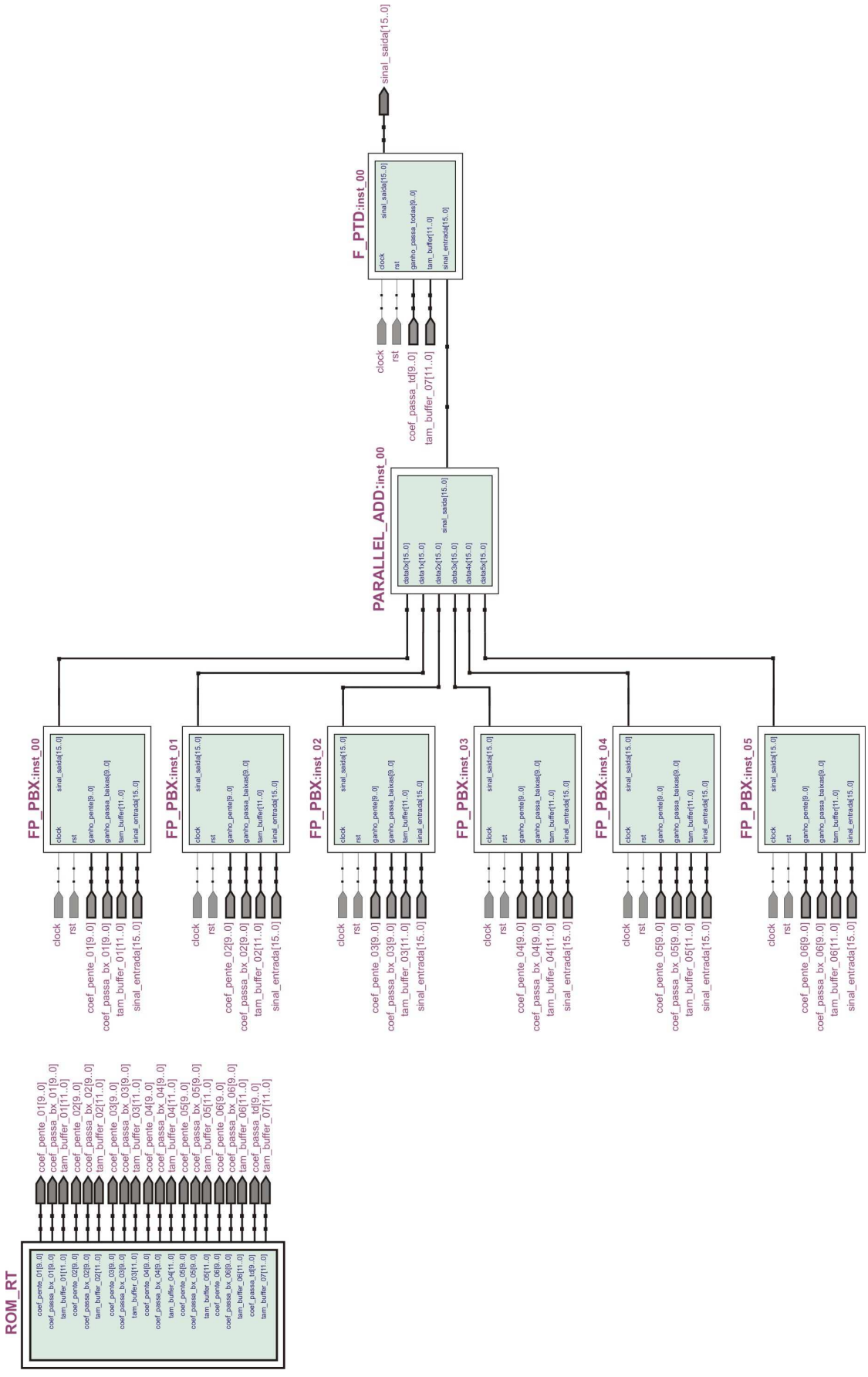


**Figura 4.6 – Diagrama de blocos dos sub-módulos do módulo principal REFTAR.**

O módulo ROM_RT é responsável por alimentar as configuração dos parâmetros associados a cada sub-módulo de REFTAR, onde *tam_buffer_xx[]* contém as informações referentes ao tempo de atraso de cada filtro Pente encapsulado em FP_PBX, *coef_pente_xx[]* contém as informações referentes ao ganho destes filtros, e *coef_passa_bx_xx[]* contém as informações referentes ao ganho dos filtros Passa-Baixas também encapsuladoes em REFTAR.

Com relação aos módulos responsáveis pela decimação (DEC) e interpolação (INT), os mesmos forma implementados através do uso da ferramenta *FIR Compiler* disponibilizada pelo *software* Quartus™ II através do aplicativo *MegaCore™ Function Generator*.

O *FIR Compiler* oferece um ambiente de desenvolvimento de filtros FIR otimizados para uso em FPGAs da Altera™. Esta otimização se dá através do controle eficiente dos recursos disponibilizados pelo *chip* (ALTERA, 2009). O *FIR Compiler* possui um gerador de coeficientes que calcula os valores ideais de acordo com especificações definidas pelo usuário (tipo de filtro, freqüência de amostragem, freqüências de corte, estrutura aritmética, entre outros), sendo que esses coeficientes determinam a resposta do filtro.

Especificamente com relação aos decimadores e interpoladores, o *FIR Compiler* gera os filtros *anti-aliasing* e *anti-imaging* de forma integrada. Sendo assim, o módulo DEC possui integrado à sua arquitetura o filtro *anti-aliasing*, e o módulo INT possui integrado à sua arquitetura o filtro *anti-imaging*. Estes coeficientes são gerados levando em consideração a freqüência de corte especificada para cada filtro.

Conforme apresentado no Capítulo 3, a arquitetura proposta prevê uma decimação e interpolação por um fator 2 (dois), e uma freqüência de amostragem principal de 44,1 kHz. Sendo assim, a freqüência de amostragem da arquitetura de reverberação (decimada) é de 22,050 kHz. Portanto, considerando o Teorama da Amostragem de Nyquist, a freqüência máxima do sinal a ser reverberado é de 11,025 kHz, sendo este o ponto de corte do filtro decimador/interpolador.

Com relação ao módulo DEC, a Figura 4.7 apresenta as configurações especificadas no *FIR Compiler* para o sistema decimador, bem como sua resposta de freqüência. Percebe-se então que este é um decimador Passa-Baixas, visto que o filtro *anti-aliasing* é incorporado à arquitetura, com uma freqüência de corte em 11,025 kHz.

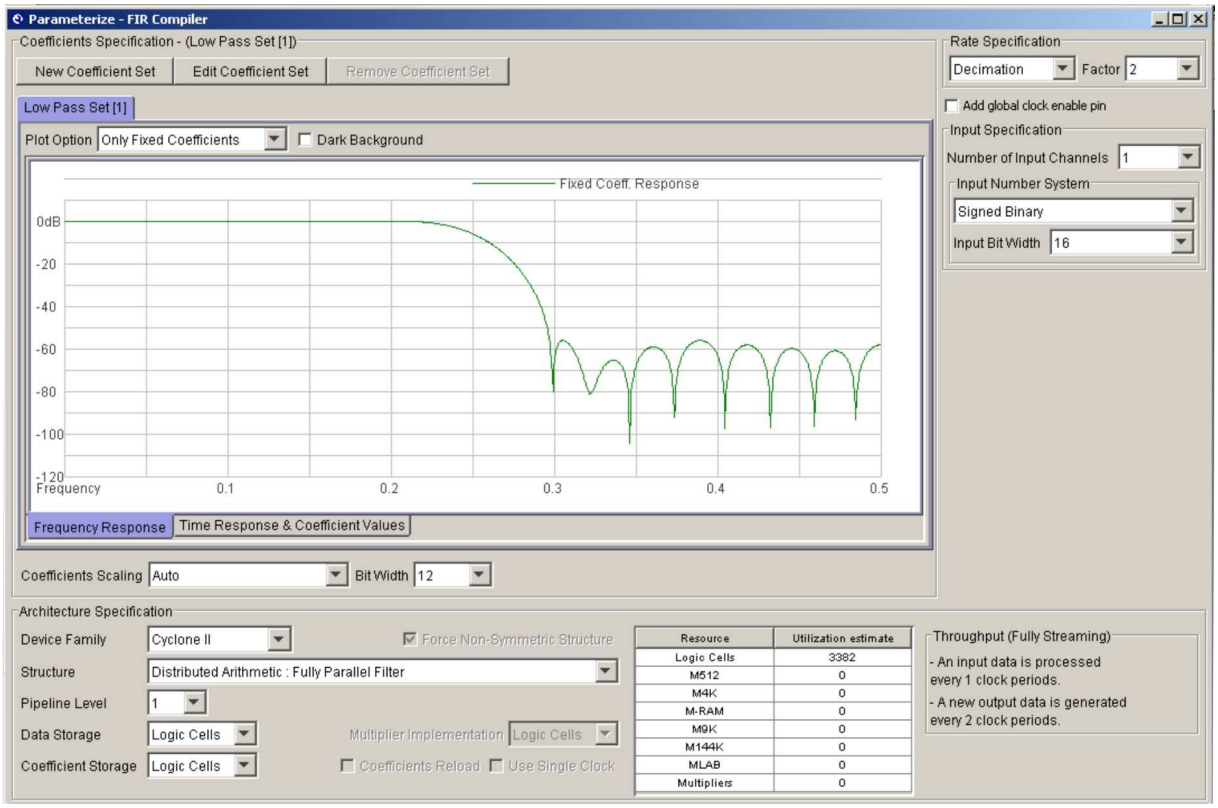


**Figura 4.7 – Configurações do *FIR Compiler* para o sistema decimador e sua resposta de freqüência.**

Já com relação ao módulo INT, a Figura 4.8 apresenta as configurações especificadas para o sistema interpolador, bem como sua resposta de freqüência. Percebe-se que o filtro *anti-imaging* é também incorporado à arquitetura, caracterizando o sistema interpolador completo.

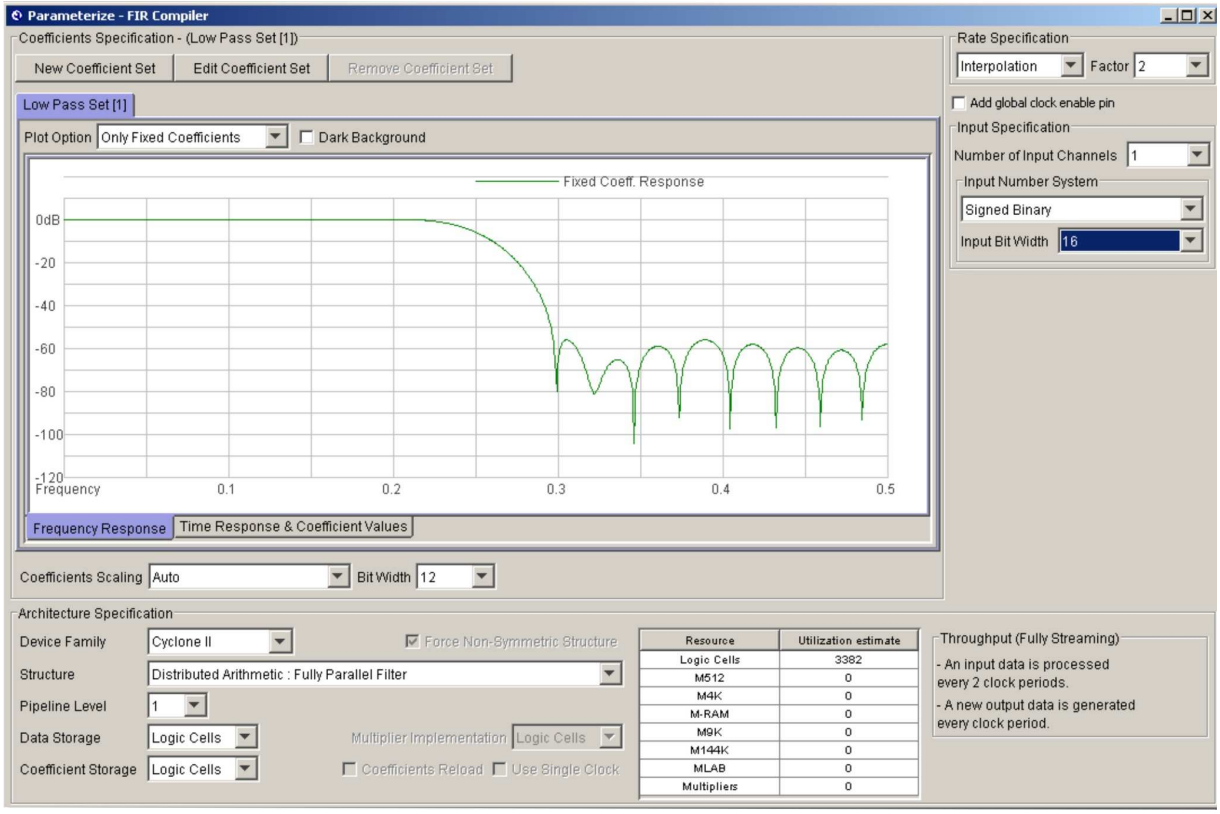


**Figura 4.8 – Configurações do *FIR Compiler* para o sistema interpolador e sua resposta de freqüência.**

Durante o processo de geração de código, o *FIR Compiler* disponibiliza uma grande variedade de arquivos secundários (VHDL, Verilog e Matlab™) para simulação e validação da arquitetura desenvolvida, fornecendo assim um amplo conjunto de possibilidades para *testbenches*. Para uma melhor visualização (gráfica), os *testbenches* fornecidos para o ambiente Matlab™ foram utilizados.

## 4.4. A arquitetura final

### 4.4.1. Introdução

Ao término da implementação e validação individual dos módulos básicos e principais, os mesmos foram então agregados. Esta seção apresenta os diagramas de blocos para ambas as arquiteturas (referência e proposta), bem como a utilização de recursos de cada arquitetura com relação ao FPGA utilizado neste trabalho.

### 4.4.2. A arquitetura de Moorer

A implementação da arquitetura de Moorer deu-se através da instanciação dos vários módulos apresentados nas seções anteriores (REFPRI, REFTAR, CG, DC), bem como um somador pertencente às bibliotecas de mega-funções da Altera™.

Na Figura 4.9 tem-se o diagrama de blocos final implementado de acordo com a proposta de Moorer.

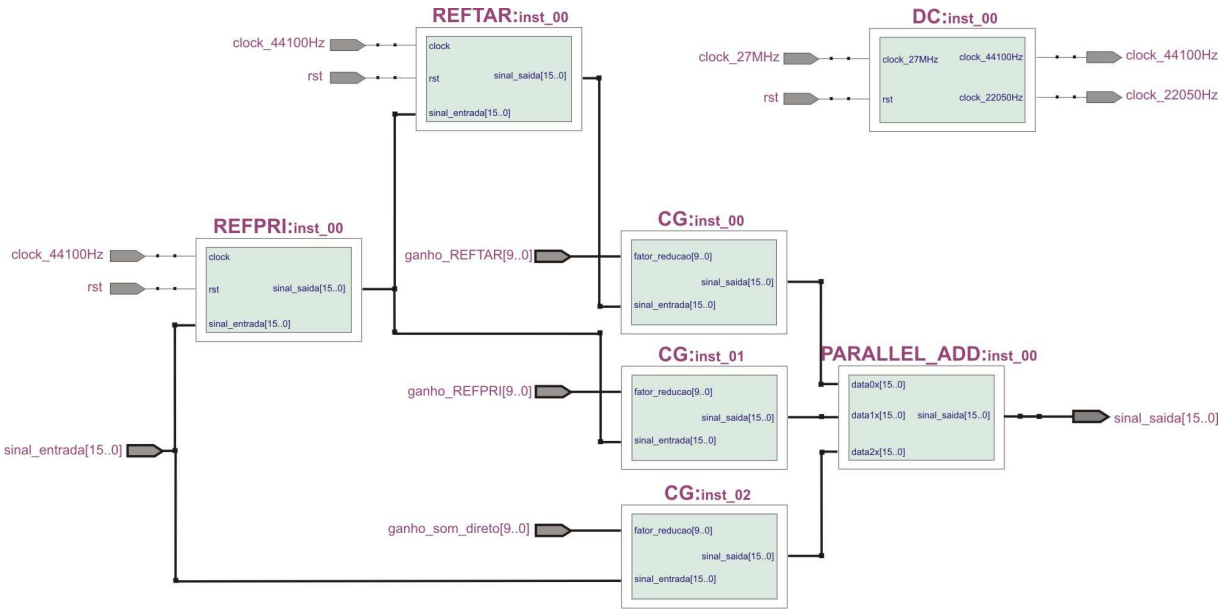


**Figura 4.9 – Diagrama de blocos da arquitetura de Moorer.**

Analisando a Figura 4.9, observa-se que cada estágio da arquitetura de reverberação (Som Direto, Reflexões Primárias e Reflexões Tardias) possui controle de ganho individual. Sendo assim, diferentes combinações podem ser simuladas, no intuito de obterem-se diversas variantes do efeito de reverberação.

#### 4.4.2.1. Utilização de recursos

Considerando o FPGA utilizado na implementação, um resumo de utilização de recursos do mesmo com relação à arquitetura de Moorer é apresentado na Tabela 4.1.

**Tabela 4.1 – Utilização de recursos do FPGA para a arquitetura de Moorer.**

| **Recursos** | **Utilizados** | **Disponíveis** | **Utilização** |
|---|---|---|---|
| Número de elementos lógicos (LE's) | 18.753 | 33.216 | 56,5% |
| Número de LUT's de 4 entradas | 18.753 | 33.216 | 56,5% |
| Número de registradores lógicos | 1.365 | 33.216 | 4,1% |
| **Número de blocos de RAM (MK4)** | **104** | **105** | **99,0%** |
| Número de multiplicadores de 9x9 bits | 58 | 70 | 82,9% |
| Número de clocks globais (GCLK's) | 2 | 16 | 12,5% |
| Número de PLL's | 1 | 4 | 25,0% |

Analisando a tabela, é possível visualizar que a grande quantidade de memória utilizada (99,0%) pela arquitetura de referência torna quase impossível sua implementação no FPGA utilizado neste trabalho.

### 4.4.3. A arquitetura proposta

Para a implementação da arquitetura proposta, a mesma metodologia utilizada na implementação da arquitetura de Moorer foi empregada. Porém, tendo como objetivo a otimização, módulos de decimação e interpolação foram agregados à arquitetura de referência.

A Figura 4.11 apresenta o diagrama de blocos final da arquitetura implementada de acordo com a proposta deste trabalho.

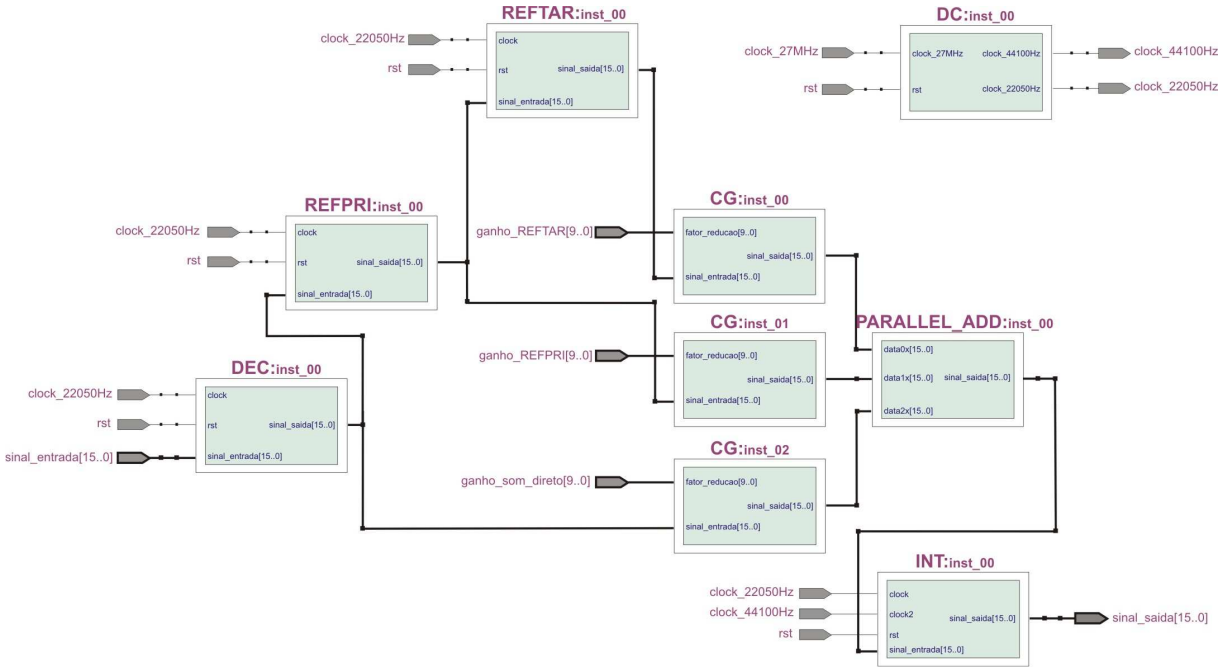


**Figura 4.10 – Diagrama de blocos da arquitetura proposta.**

Analisando a Figura 4.11, observa-se que o mesmo conceito utilizado na arquitetura de Moorer foi mantido, ou seja, cada estágio da arquitetura de reverberação (Som Direto, Reflexões Primárias e Reflexões Tardias) possui controle de ganho individual. Paralelo a isto, os módulos de decimação e interpolação foram adicionados.

### 4.4.3.1. Utilização de recursos

Considerando o FPGA utilizado na implementação, um resumo de utilização de recursos do mesmo com relação à arquitetura de proposta neste trabalho é apresentado na Tabela 4.2.

**Tabela 4.2 – Utilização de recursos do FPGA para a arquitetura proposta.**

| Recursos | Utilizados | Disponíveis | Utilização |
|---|---|---|---|
| Número de elementos lógicos (LE's) | 24.776 | 33.216 | 74,6% |
| Número de LUT's de 4 entradas | 24.776 | 33.216 | 74,6% |
| Número de registradores lógicos | 7.131 | 33.216 | 21,5% |
| **Número de blocos de RAM (MK4)** | **52** | **105** | **49,5%** |
| Número de multiplicadores de 9x9 bits | 64 | 70 | 91,4% |
| Número de clocks globais (GCLK's) | 2 | 16 | 12,5% |
| Número de PLL's | 1 | 4 | 25,0% |

Analisando a tabela, é possível visualizar que a quantidade de memória utilizada pela arquitetura proposta (49,5%) é significativamente menor em relação a arquitetura de referência.

## 4.5. Metodologia de apuração dos resultados

### 4.5.1. Introdução

Findada a implementação de ambas as arquiteturas, partiu-se então para a apuração dos resultados proporcionados pelas mesmas. Conforme mencionado na Seção 4.2 o *software* Quartus™ II foi utilizado como ferramenta de síntese e simulação, e o dispositivo para a implementação das arquiteturas foi o FPGA Cyclone™ II EP2C35F6726. As simulações realizadas foram: funcional, para detectar o correto funcionamento do sistema; *timing*, para observar o comportamento do sistema com relação aos atrasos proporcionados pelas arquiteturas.

Desta forma, inicialmente é apresentado o método e as considerações envolvidas na tarefa de apuração de resultados, sendo estes últimos discutidos na seqüência.

### 4.5.2. Os dados de entrada

Conforme apresentado no Capítulo 2 (seção 2.3.11), a melhor maneira de se analisar o efeito de reverberação é através da resposta ao impulso. O conceito de resposta impulsiva pode ser definido como a resposta da pressão sonora registrada na posição de interesse do ouvinte, quando um sinal de excitação muito intenso e de curta duração é produzido.

A resposta impulsiva é particularmente atrativa por oferecer uma completa descrição da transmissão entre dois pontos. Sendo assim, através de um sinal de excitação emitido em determinado ambiente, é possível obter a curva de decaimento do som, bem como o tempo de reverberação (BOTTAZZINI, 2008).

Sendo assim, buscando validar as arquiteturas implementadas, utilizou-se como dado de entrada um impulso unitário, visando obter a resposta da arquitetura em relação ao impulso. É evidente que a utilização de um sinal de áudio real como dado de entrada seria mais interessante. Porem, para validar a arquitetura no FPGA em relação à um sinal de áudio real, seria necessário um estudo mais aprofundado sobre os conversores AD/DA da placa de prototipação, bem como a maneira de integrá-los ao sistema. Visto que não é escopo desta proposta trabalhar com conversores, optou-se então pela utilização da resposta impulsiva como forma de validação.

Neste sentido, analisando a Figura 4.10, é possível visualizar graficamente a resposta ao impulso obtida pela arquitetura de referência (Moorer).

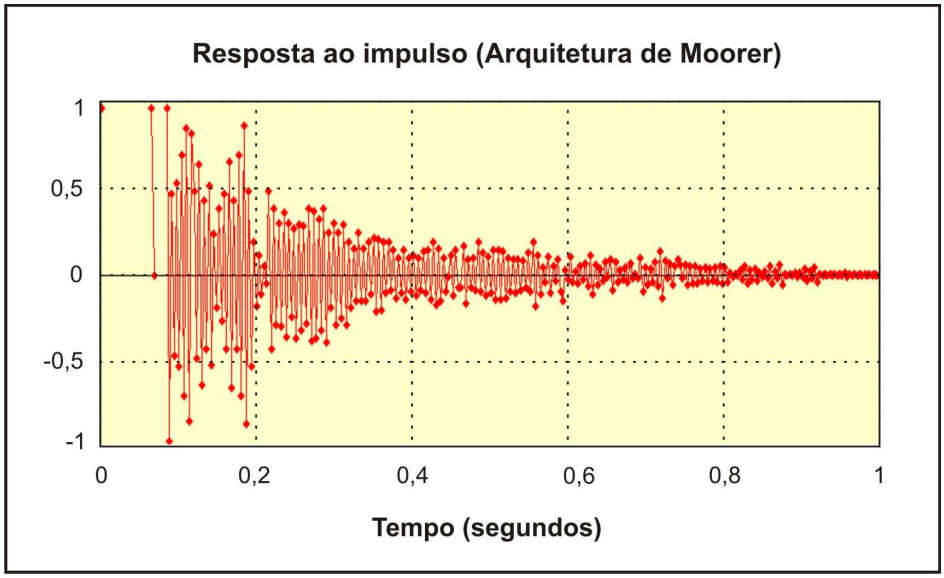


**Figura 4.11 – Resposta ao impulso da arquitetura de Moorer.**

Já com relação à arquitetura proposta, têm-se a Figura 4.12, onde é possível visualizar graficamente a resposta ao impulso obtida pela arquitetura proposta.

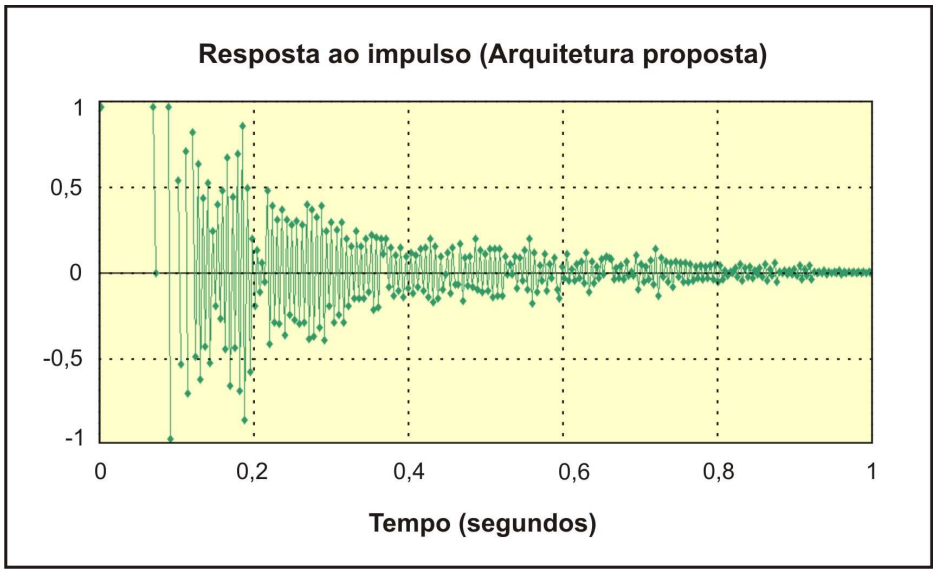


**Figura 4.12 – Resposta ao impulso da arquitetura proposta.**

Finalmente, têm-se a Figura 4.13, onde é possível visualizar graficamente a comparação entre as duas arquiteturas com relação à resposta ao impulso.

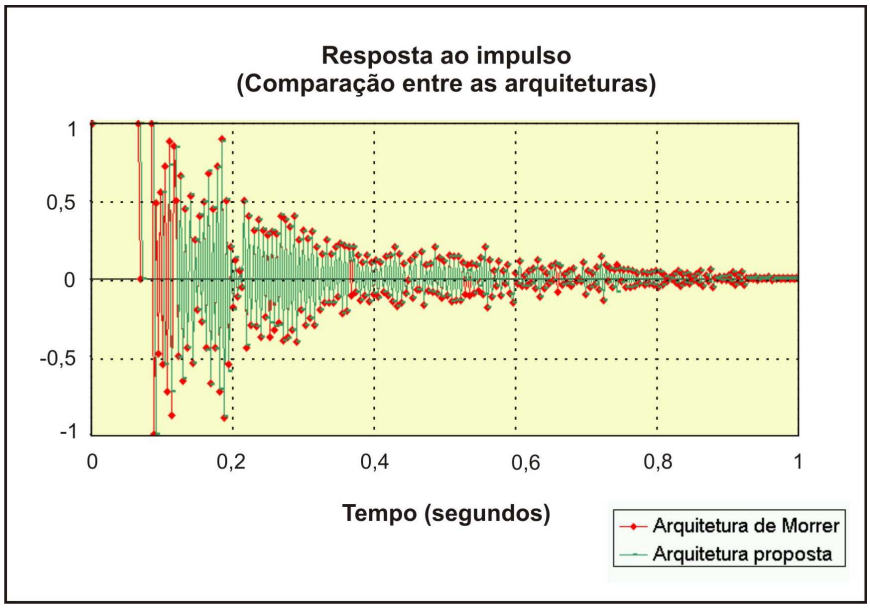


**Figura 4.13 - Comparação entre as arquiteturas (resposta ao impulso)**

Analisando a Figura 4.13, é possível perceber que a arquitetura proposta apresentou um leve atraso em relação à arquitetura de Moorer. Este atraso já era esperado, sendo o mesmo atribuído à inserção dos componentes de *hardware* adicionais (arquitetura de decimação/interpolação), bem como aos arredondamentos feitos para diminuição dos tamanhos de *buffer*. O atraso proporcionado pela arquitetura proposta foi de aproximadamente 320 ns, sendo que o mesmo pode ser considerado insignificante do ponto de vista de percepção do sistema auditivo humano, que não consegue diferenciar atrasos de tempo tão pequenos

## 4.6. Conclusão

Este capítulo apresentou os detalhes da implementação da arquitetura de reverberação digital (arquitetura de referência e arquitetura proposta), bem como os resultados referentes às implementações. Os blocos implementados e a descrição funcional do sistema também foram apresentados, sendo as conclusões finais apresentadas no próximo capítulo.

# 5. CONCLUSÕES

Tendo como principal motivação a performance e quantidade de recursos disponibilizadas pelos dispositivos lógicos programáveis (FPGA), bem como a redução dos recursos computacionais proporcionadas pelas técnicas de processamento multitaxa, o intuito deste trabalho foi desenvolver uma arquitetura de *hardware* otimizada através do uso de técnicas de processamento multitaxa, específica para a implementação de um algoritmo de reverberação digital, utilizando dispositivos lógicos programáveis (FPGA) como ambiente de pesquisa e prototipação. Como resultado deste estudo, têm-se duas arquiteturas de reverberação digital (arquitetura de referência e arquitetura proposta) implementadas sobre um FPGA.

No que diz respeito ao processo de desenvolvimento, inicialmente um estudo detalhado sobre os principais algoritmos de reverberação foi realizado. Diversas propostas de implementação foram analisadas, visando à escolha da melhor arquitetura.

Após definida a arquitetura a ser utilizada como estudo de caso, a implementação da mesma foi primeiramente realizada utilizando uma abordagem baseada em *software* construída sobre uma plataforma x86 (*Windows XP*), com auxílio da ferramenta MATLAB™ Simulink™. O intuito da construção de tal versão era possuir um parâmetro para comparações posteriores com a arquitetura desenvolvida em *hardware*. Como não é objetivo deste trabalho apresentar uma solução em *software*, a mesma foi utilizada somente para comparações durante o desenvolvimento da arquitetura de *hardware*. Sendo assim, a implementação em *software* não foi detalhada neste trabalho, visto que foge do escopo do mesmo. Com a implementação em *software* concluída e validada, partiu-se então para o desenvolvimento sobre uma plataforma reconfigurável (FPGA) como arquitetura dedicada.

Com relação ao uso de técnicas de processamento multitaxa, o mesmo se justifica, visto que a reverberação digital se baseia em linhas de atraso cujo tamanho é proporcional à freqüência de amostragem, e as técnicas de processamento multitaxa possibilitam a redução desta freqüência. A escolha do fator de decimação/interpolação deu-se baseada no compromisso com a qualidade sonora. Como este trabalho constitui a primeira versão do sistema, a escolha do fator 2 se mostrou a mais lógica, haja vista que fatores mais elevados poderiam ser utilizados, porém, isto resultaria em perda de qualidade.

Dentre as conclusões obtidas a partir da avaliação dos resultados, a mais importante foi a redução no consumo de memória da arquitetura proposta em relação à arquitetura de referência. Esta redução foi de 50% conforme pode ser observado analisando as tabelas 4.1 e 4.2, sendo que este foi o principal objetivo buscado na proposta de otimização.

Já com relação à área ocupada, a arquitetura proposta apresentou um aumento de 18% em relação à arquitetura de Moorer (tabelas 4.1 e 4.2). Este fato já era esperado, haja vista que novos componentes de *hardware* foram incorporados à arquitetura de referência. Porém, destaca-se que este aumento de área não leva em consideração a área ocupada pela memória (que neste caso foi consideravelmente reduzida), visto que esta ocupa uma parte específica do FPGA, e não está relacionada com os LE's. Outro fator a ser ressaltado é que este trabalho apresenta somente 1 (um) efeito de áudio (reverberação) para um ambiente específico, e que a inclusão de novos efeitos se beneficiará da arquitetura adicional, sem a necessidade de incorporação de novos componentes de *hardware* para redução de memória. Sendo assim, o aumento na área ocupada não se torna um fator preocupante, haja vista que, à medida que novos efeitos forem agregados à arquitetura proposta, a redução no consumo de memória acabará por reduzir a área ocupada.

Com relação ao sistema como um todo, percebeu-se que a arquitetura proposta apresentou um leve atraso em relação à arquitetura de Moorer, conforme pode ser observado na Figura 4.13. Este atraso já era esperado, sendo o mesmo atribuído à inserção dos componentes de *hardware* adicionais (arquitetura de decimação/interpolação), bem como aos arredondamentos feitos para diminuição dos tamanhos de *buffer*. O atraso proporcionado pela arquitetura proposta foi de aproximadamente 320 ns, sendo que o mesmo pode ser considerado insignificante do ponto de vista de percepção do sistema auditivo humano, que não consegue diferenciar atrasos de tempo tão pequenos

Através dos resultados obtidos, pode-se concluir também que a utilização de técnicas de processamento multitaxa se apresenta como uma solução atrativa para o desenvolvimento de aplicações de áudio digital, mais especificamente, efeitos sonoros baseados em atrasos de tempo.

Finalizando a discussão e este capítulo, seguem abaixo sugestões de trabalhos futuros que certamente contribuirão com o aprimoramento desta primeira versão:

- Avaliação da qualidade de áudio proporcionada pelo sistema quando da utilização de fatores de decimação/interpolação mais elevados;

- Incorporação de outros efeitos baseados em atrasos de tempo (*Delay*, *Chorus*, entre outros) e avaliação dos resultados obtidos;
- Avaliação do consumo de potência entre a arquitetura proposta e a arquitetura de referência;
- Implementação de outras arquiteturas de reverberação (Datorro, Gardner, Schroeder, entre outros), utilizando os conceitos de processamento multitaxa, bem como sua comparação com a arquitetura proposta;
- Elaboração de novas arquiteturas para efeitos de reverberação baseadas nos conceitos de processamento multitaxa apresentados neste trabalho;
- Implementação de uma arquitetura em que o Som Direto seja enviado paralelamente à arquitetura de decimação/interpolação, sem passar por nenhum processamento, ou seja, mantendo sua freqüência de amostragem original. Este Som Direto seria posteriormente somado ao som processado pela arquitetura multitaxa através de um *mix* dos dois sinais. Isto possivelmente aumentaria a qualidade final do áudio gerado pelo reverberador. Neste sentido, uma investigação mais aprofundada com relação ao atraso proporcionado pela arquitetura de decimação/interpolação deve ser feita, visto que, *a priori*, o Som Direto não sofreria atraso, sendo então necessária a inclusão de um controle de atraso por parte da arquitetura paralela responsável pelo Som Direto (sem processamento) para que o mesmo entrasse em sincronia com o som processado pela arquitetura multitaxa.

# BIBLIOGRAFIA

ALTERA **Cyclone II device handbook, vol 1** – Documentação da família de dispositivos Cyclone II da Altera, 2008.

ALTERA **FIR Compiler User Guide** – Documentação do *IP Core FIR Compiler* da Altera, 2009.

BARBOSA, ÁLVARO M. – **Edição digital de som: uma abordagem aos fundamentos da escultura sonora orientada para criadores** – Portugal: Universidade Católica Portuguesa, 1999.

BARCELLOS, LUIZ C. R. DE – **Estruturas eficientes de transmultiplexadores e de banco de filtro modulados por cossenos** – Tese de doutorado – Universidade Federal do Rio de Janeiro, 2006.

BELTRÁN, FERNANDO A.; BELTRÁN, JOSÉ R.; ARREGUI, JAVIER; GRACIA, BELÉN – **A real-time DSP-based reverberation system with computer interface** – Proceedings of 1st COST-G6 Workshop on Digital Audio Effects (DAFX98) - Barcelona, 1999.

BELTRÁN, FERNANDO A..; BELTRÁN, JOSÉ R.; HOLZEM, NICOLAS; GOGY, ADRIAN – **Matlab Implementation of Reverberation Algorithms** – Proceedings of 2007 IEEE International conference on Reconfigurable Computing and FPGAs - ReConfig , pp. 1-8, 2007.

BERANEK, LEO L. – **Acoustics** – Journal of Acoustic Society of America – Vol. 83, Issue 3, pp. 1206-1207, 1986.

BOTTAZZINI, MARCELO CARVALHO; BERTOLI, STELAMARIS ROLLA – **Acústica de igrejas barrocas, arquitetura que faz a diferença** – VI Congresso Iberoamericano de acústica – FIA 2008 – Buenos Aires, 2008.

BRAZIL, VINÍCIUS – Perspectivas do áudio digital – Disponível em: http://www.discoverstudio.com.br – Acesso em: 29 jan 2004.

CARVALHO, JANILSON R. DE – **Estimação de harmônicos/inter-harmônicos: uma abordagem multitaxa** – Universidade Federal de Juiz de Fora, 2008.

DATTORRO, JON – **Effect Design – Part 1: Reverberator and other filters** - JAES Volume 45 Issue 9 pp. 660-684; September 1997.

DAVIS, D.; SRINIVAS, B.; RANJESH, J. – **Hardware/software codesign for platform FPGAs** – 2005 Embedded Systems Conference – San Francisco – Disponível em: http://www.techonline.com/learnign/techpaper/193193703 - Acesso em: 22 nov 2008.

DESSBESELL, GUSTAVO FERNANDO – **Extração e reconhecimento de caracteres ópticos a partir do co-projeto de *hardware* e *software* sobre plataforma reconfigurável** – Dissertação de mestrado – Universidade Federal de Santa Maria, 2008.

DINIZ, PAULO S. R.; SILVA, EDUARDO A. B.; NETTO, SÉRGIO L. – Processamento digital de sinais: projeto e análise de sistemas – Editora Bookman, 2004.

DUTILLEUX, PIERRE – **Filters, delays, modulations and demodulations: a tutorial** – Proceedings of Cost-G6 conference in Digital Audio Effects (DAFX98), pp 4-11, Barcelona, 1998.

ELHOSSINI, AHMED; AREIBI, SHAWKI; DONY, ROBERT – **An FPGA implementation of the LMS adaptive filter for áudio processing** – Proceedings of 2007 IEEE international conference on Reconfigurable Computing and FPGAs, 2006 – pp. 1-8.

FERNANDES, GABRIEL F. P. – **Implementação em DSP de um sistema "real-time analyzer" aplicação à igualização adaptativa** – Dissertação de Mestrado – Faculdade de Engenharia da Universidade do Porto, 2002

FERREIRA, CRISTIANO OLIVEIRA – **Reverberação artificial** – Trabalho de Integração, Pontifícia Universidade Católica do Rio Grande do Sul, 2004

FLETCHER, B. H. – **FPGA embedded processors – Revealing the true system performance** – 2005 Embedded systems conference – San Francisco – Disponível em: http://www.techonline.com/learning/techpaper/193103682 - Acesso em 22 nov 2007.

FURTADO JÚNIOR, GILSON G. DA SILVA – **Banco de filtros e wavelets sobre corpos finitos** – Dissertação de mestrado – Universidade Federal de Pernambuco, 2006.

FURTADO, MIGUEL B. - **Otimização de transceptores FIR baseados em bancos de filtros aplicados à comunicação digital móvel e fixa** – Tese de doutorado – Universidade Federal do Rio de Janeiro, 2006.

FREITAS, MARCO A. B. – **Medindo a velocidade do som com o microfone do PC** – Projeto final de curso – Instituto de Física – Universidade Federal do Rio de Janeiro, 2005.

GARDNER, W. G. – **Reverberations algorithms** – in Applications of Digital Signal Processing to Audio and Acoustics – Norwell: Kluwer Academic Publishers, 1998.

GUEDES, KARLA BASTOS – **Simulação eficiente do sistema elétrico de potência utilizando filtragem digital multitaxa** – Tese de doutorado – Pontifícia Universidade Católica do Rio de Janeiro, 2002.

JAHROMI, OMID. S. – **Multirate statistical processing** – EUA: Springer, Netherlands, 2007

JESUS, RUI – Operação de zooming em processamento de imagem – Disponível em: http://www.deetc.isel.ipl.pt/comunicacoesep/disciplinas/pds/Apontamentos_zoom.pdf - Acesso em: 23 de set 2002.

JOHANSSON, HAKAN; SMITH, DANNY; STROMDAH, ERIK; SVENSSON, HELENE – **Algorithms in signal processors audio applications** – DSP Project course – Dept. of Eletroscience, Lund University, Sweden, 2005

KAHRS, MARK – **Applications of digital signal processing to audio and acoustics** – EUA: Kluwer Academic Publishers, 2002.

KUO, SEN M.; LEE, BOB H. – **Implementations, applications and experiments with TMS320C55X** – EUA: John Wiley& Sons, Inc., 2001.

KUTTRUFF, H. – **Room acoustics** – EUA: Applied Science Publisher Ltd., 1991.

MEYER-BAESE, UWE – **Digital signal processing with field programmable gate arrays** – 3nd ed., EUA: Springer-Verlag, 2007.

MICHAUD, JOSEPH-FRANÇOIS – Glória da Idade Média – Disponível em: http://gloriadaidademedia.blogspot..com – Acesso em 23 de Jan 2009.

MOORER, JAMES A. – **About this reverberation business** – Computer music journal, Vol. 3, N. 2, pp 13-28, 1979.

OPPENHEIM, ALAN V.; SCHAFER, RONALD W. - **Discrete-time signal processing** – 2nd ed., EUA: Prentice-Hall, 1999.

PIIRILÄ, E.; LOKKI , T.; VALIMAKI, V. - **Digital Signal Processing Techniques for Non-exponentially Decaying Reverberation –** Proceedings of 1st COST-G6 Workshop on Digital Audio Effects (DAFX98) - Barcelona, Spain, 1998.

PROAKIS, JOHN G.; MANOLAKIS, DIMITRIS G. – **Digital Signal Processing: Principles, algorithms and applications** – 3nd ed., EUA: Prentice-Hall, 1996.

RATTON, MIGUEL – **Processamento de Efeitos** – Artigo publicado em WWW.music-center.com.br, 2002.

SABINE, W. C. – **Collected papers on acoustics** – Harvard University Press – London, 1922.

SANCHIDRIÁN, CÉSAR DÍAS; GONZALES, ANTONIO P. – **La acústica de la iglesia del monarterio de Santo Domingo de Silos y su adecuación a La práctica de canto gregoriano** – TecniAcustica, Madrid, 2006.

SCANDELARI, LUCIANO – **Filtros digitais** – Disponível em: http://pessoal.cefetpr.br/luciano/DSP.htm - Acesso em 29 de nov 2008.

SEHN, LEANDRO R.; MOLZ, ROLF F. – **Um processador de efeitos de audio *stand-alone* empregando FPGA** – In: Anais do IX Seminário de Iniciação Científica e VIII Jornada de Ensino, Pesquisa e Extensão - Santa Cruz do Sul : UNISC, 2004.

SHENOI, B. A. – **Introduction to digital signal processing and filter design** – EUA: John Wiley & Sons, Inc., 2006.

SMITH, STEVEN W. – **The scientist and engineer's guide to digital signal processing** - 2nd ed., EUA: California Techinical Publishing, 1999.

SPANIAS, ANDREAS; PAINTER, TED; ATTI, VENKATRAMAN – **Audio signal processing and coding** – EUA: John Wiley & Sons, Inc., 2007.

TOMA, NORBERT; TOPA, MARIAN D.; POPESCU, VICTOR; SZOPOS, ERWIN – **Comparative performance analysis of artificial reverberation algorithms -** Proceedings of

IEEE international conference on Automation, Quality and Testing, Robotisc – Vol. 1, pp. 138-142, 2006.

TOMARAKOS, JOHN – **Using the low-cost, high performance ADSP-21065L digital signal processor for digital audio applications** – EUA: Analog Devices, 1998.

TOPA, MARINA DANA; TOMA, NORBERT; POPESCU, VICTOR; TOPA, VASILE – **Evaluation of all-pass reverberators** – Proceedings of IEEE international conference on Eletronic, Circuits and Systems (ICECS) – pp.339-342, 2008.

TOUMAZOU, C.; HUGHES, J. B.; BATTERSBY , N. C. – **Switched-Current: An Analogue Technique for Digital Technology** – IEE Peter Peregrinus Ltd, London, UK, 1993.

WHITE, PAUL – **Choosing The Right Reverb , how Best to use the different reverbs** - http://www.soundonsound.com/sos/mar06/articles/usingreverb.htm - Acessado em 21 de agosto de 2009.

VAIDYANATHAN, P. P. – **Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial** - IEEE Proceedings, Vol.78, pp. 56-93, 1990.

VALLE, SÓLON DO – **Manual prático de acústica** – 1nd ed., Rio de Janeiro: Música & Tecnologia, 2006.

# APÊNDICE A – LISTAGEM DOS CÓDIGOS FONTE

**Listagem A.1: Código VHDL utilizado para implementação da Unidade de Atraso.**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library lpm;
use lpm.lpm_components.all;

entity delay is
generic
(
	tam_palavra:integer :=16; --determina o tamanho do sinal (bits)
	num_enderecos:integer:=5--determina o tamanho do buffer (tempo de atraso) (bits)
);
port
(
	clk:in std_logic;
	sinal_entrada: in unsigned(tam_palavra-1 downto 0);
	sinal_saida:out unsigned(tam_palavra-1 downto 0)
);
end delay;

---------------------------------------------------------------------------------------
architecture delay_arc of delay is
---------------------------------------------------------------------------------------
	COMPONENT scfifo
	GENERIC
	(
		intended_device_family: STRING;
		lpm_width: NATURAL;
		lpm_numwords: NATURAL;
		lpm_type: STRING;
		lpm_showahead: STRING;
		overflow_checking: STRING;
		underflow_checking: STRING;
		use_eab: STRING
	);
	PORT
	(
			rdreq: IN STD_LOGIC ;
			clock: IN STD_LOGIC ;
			q: OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
			wrreq: IN STD_LOGIC ;
			data: IN STD_LOGIC_VECTOR (15 DOWNTO 0)
	);
	END COMPONENT;
---------------------------------------------------------------------------------------
	signal ler_fifo, escrever_fifo:std_logic;
	shared variable ssaida:std_logic_vector(tam_palavra-1 downto 0);
---------------------------------------------------------------------------------------
begin

	process(clk)
```

```
		variable cont:integer:=1;
	begin
		if clk'event and (clk='1') then
			if cont>=(num_enderecos-1) then
				ler_fifo<='1';
			else
				ler_fifo<='0';
			end if;
			escrever_fifo<='1';
			cont:=cont+1;
		end if;
	end process;

	scfifo_component : scfifo
	GENERIC MAP (
		intended_device_family => "CYCLONEII",
		lpm_width => tam_palavra,
		lpm_numwords => num_enderecos,
		lpm_type => "scfifo",
		lpm_showahead => "OFF",
		overflow_checking => "ON",
		underflow_checking => "ON",
		use_eab => "ON"
	)
	PORT MAP (
		rdreq => ler_fifo,
		clock => clk,
		wrreq => escrever_fifo,
		data =>conv_std_logic_vector(sinal_entrada,16),
		q => ssaida
	);

	sinal_saida<=unsigned(ssaida);

end delay_arc;
---------------------------------------------------------------------------------------
```

**Listagem A.2: Código VHDL utilizado para implementação do Decimador.**

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;
ENTITY decimador IS
  PORT( clk                       :  IN    std_logic;
        clk_enable                :  IN    std_logic;
        reset                     :  IN    std_logic;
        filter_in                 :  IN    std_logic_vector(15 DOWNTO 0); -- sfix16_En15
        filter_out                :  OUT   std_logic_vector(33 DOWNTO 0); -- sfix34_En31
        ce_out                    :  OUT   std_logic
        );

END decimador;


-------------------------------------------------------------
--Module Architecture: decimador
-------------------------------------------------------------
ARCHITECTURE rtl OF decimador IS
  -- Local Functions
  -- Type Definitions
  TYPE input_pipeline_type IS ARRAY (NATURAL range <>) OF signed(15 DOWNTO 0); -- sfix16_En15
  -- Constants
  CONSTANT coeffphase1_1              : signed(15 DOWNTO 0) := to_signed(-60, 16); -- sfix16_En16
  CONSTANT coeffphase1_2              : signed(15 DOWNTO 0) := to_signed(240, 16); -- sfix16_En16
  CONSTANT coeffphase1_3              : signed(15 DOWNTO 0) := to_signed(502, 16); -- sfix16_En16
  CONSTANT coeffphase1_4              : signed(15 DOWNTO 0) := to_signed(400, 16); -- sfix16_En16
  CONSTANT coeffphase1_5              : signed(15 DOWNTO 0) := to_signed(-1108, 16); -- sfix16_En16
  CONSTANT coeffphase1_6              : signed(15 DOWNTO 0) := to_signed(1360, 16); -- sfix16_En16
  CONSTANT coeffphase1_7              : signed(15 DOWNTO 0) := to_signed(-68, 16); -- sfix16_En16
  CONSTANT coeffphase1_8              : signed(15 DOWNTO 0) := to_signed(-3891, 16); -- sfix16_En16
  CONSTANT coeffphase1_9              : signed(15 DOWNTO 0) := to_signed(20317, 16); -- sfix16_En16
  CONSTANT coeffphase1_10             : signed(15 DOWNTO 0) := to_signed(4403, 16); -- sfix16_En16
  CONSTANT coeffphase1_11             : signed(15 DOWNTO 0) := to_signed(1793, 16); -- sfix16_En16
  CONSTANT coeffphase1_12             : signed(15 DOWNTO 0) := to_signed(-2244, 16); -- sfix16_En16
  CONSTANT coeffphase1_13             : signed(15 DOWNTO 0) := to_signed(1143, 16); -- sfix16_En16
  CONSTANT coeffphase1_14             : signed(15 DOWNTO 0) := to_signed(23, 16); -- sfix16_En16
  CONSTANT coeffphase1_15             : signed(15 DOWNTO 0) := to_signed(-504, 16); -- sfix16_En16
  CONSTANT coeffphase1_16             : signed(15 DOWNTO 0) := to_signed(1137, 16); -- sfix16_En16
  CONSTANT coeffphase1_17             : signed(15 DOWNTO 0) := to_signed(-163, 16); -- sfix16_En16
  CONSTANT coeffphase2_1              : signed(15 DOWNTO 0) := to_signed(-178, 16); -- sfix16_En16
  CONSTANT coeffphase2_2              : signed(15 DOWNTO 0) := to_signed(895, 16); -- sfix16_En16
  CONSTANT coeffphase2_3              : signed(15 DOWNTO 0) := to_signed(-430, 16); -- sfix16_En16
  CONSTANT coeffphase2_4              : signed(15 DOWNTO 0) := to_signed(909, 16); -- sfix16_En16
  CONSTANT coeffphase2_5              : signed(15 DOWNTO 0) := to_signed(-584, 16); -- sfix16_En16
  CONSTANT coeffphase2_6              : signed(15 DOWNTO 0) := to_signed(-806, 16); -- sfix16_En16
  CONSTANT coeffphase2_7              : signed(15 DOWNTO 0) := to_signed(3132, 16); -- sfix16_En16
  CONSTANT coeffphase2_8              : signed(15 DOWNTO 0) := to_signed(-5394, 16); -- sfix16_En16
  CONSTANT coeffphase2_9              : signed(15 DOWNTO 0) := to_signed(28184, 16); -- sfix16_En16
  CONSTANT coeffphase2_10             : signed(15 DOWNTO 0) := to_signed(-5394, 16); -- sfix16_En16
  CONSTANT coeffphase2_11             : signed(15 DOWNTO 0) := to_signed(3132, 16); -- sfix16_En16
  CONSTANT coeffphase2_12             : signed(15 DOWNTO 0) := to_signed(-806, 16); -- sfix16_En16
  CONSTANT coeffphase2_13             : signed(15 DOWNTO 0) := to_signed(-584, 16); -- sfix16_En16
  CONSTANT coeffphase2_14             : signed(15 DOWNTO 0) := to_signed(909, 16); -- sfix16_En16
  CONSTANT coeffphase2_15             : signed(15 DOWNTO 0) := to_signed(-430, 16); -- sfix16_En16
  CONSTANT coeffphase2_16             : signed(15 DOWNTO 0) := to_signed(895, 16); -- sfix16_En16
  CONSTANT coeffphase2_17             : signed(15 DOWNTO 0) := to_signed(-178, 16); -- sfix16_En16
  CONSTANT coeffphase3_1              : signed(15 DOWNTO 0) := to_signed(-163, 16); -- sfix16_En16
  CONSTANT coeffphase3_2              : signed(15 DOWNTO 0) := to_signed(1137, 16); -- sfix16_En16
```

```
CONSTANT coeffphase3_3              : signed(15 DOWNTO 0) := to_signed(-504, 16); -- sfix16_En16
CONSTANT coeffphase3_4              : signed(15 DOWNTO 0) := to_signed(23, 16); -- sfix16_En16
CONSTANT coeffphase3_5              : signed(15 DOWNTO 0) := to_signed(1143, 16); -- sfix16_En16
CONSTANT coeffphase3_6              : signed(15 DOWNTO 0) := to_signed(-2244, 16); -- sfix16_En16
CONSTANT coeffphase3_7              : signed(15 DOWNTO 0) := to_signed(1793, 16); -- sfix16_En16
CONSTANT coeffphase3_8              : signed(15 DOWNTO 0) := to_signed(4403, 16); -- sfix16_En16
CONSTANT coeffphase3_9              : signed(15 DOWNTO 0) := to_signed(20317, 16); -- sfix16_En16
CONSTANT coeffphase3_10             : signed(15 DOWNTO 0) := to_signed(-3891, 16); -- sfix16_En16
CONSTANT coeffphase3_11             : signed(15 DOWNTO 0) := to_signed(-68, 16); -- sfix16_En16
CONSTANT coeffphase3_12             : signed(15 DOWNTO 0) := to_signed(1360, 16); -- sfix16_En16
CONSTANT coeffphase3_13             : signed(15 DOWNTO 0) := to_signed(-1108, 16); -- sfix16_En16
CONSTANT coeffphase3_14             : signed(15 DOWNTO 0) := to_signed(400, 16); -- sfix16_En16
CONSTANT coeffphase3_15             : signed(15 DOWNTO 0) := to_signed(502, 16); -- sfix16_En16
CONSTANT coeffphase3_16             : signed(15 DOWNTO 0) := to_signed(240, 16); -- sfix16_En16
CONSTANT coeffphase3_17             : signed(15 DOWNTO 0) := to_signed(-60, 16); -- sfix16_En16

-- Signals
SIGNAL ring_count                : unsigned(2 DOWNTO 0); -- ufix3
SIGNAL phase_0                   : std_logic; -- boolean
SIGNAL phase_1                   : std_logic; -- boolean
SIGNAL phase_2                   : std_logic; -- boolean
SIGNAL ce_out_reg                : std_logic; -- boolean
SIGNAL input_register            : signed(15 DOWNTO 0); -- sfix16_En15
SIGNAL input_pipeline_phase0          : input_pipeline_type(0 TO 15); -- sfix16_En15
SIGNAL input_pipeline_phase1          : input_pipeline_type(0 TO 16); -- sfix16_En15
SIGNAL input_pipeline_phase2          : input_pipeline_type(0 TO 16); -- sfix16_En15
SIGNAL product_phase0_1             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp                  : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_2             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_1                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_3             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_2                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_4             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_3                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_5             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_4                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_6             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_5                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_7             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_6                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_8             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_7                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_9             : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_8                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_10            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_9                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_11            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_10               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_12            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_11               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_13            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_12               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_14            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_13               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_15            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_14               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_16            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_15               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase0_17            : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_16               : signed(31 DOWNTO 0); -- sfix32_En31
```

```
SIGNAL product_phase1_1          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_17               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_2          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_18               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_3          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_19               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_4          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_20               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_5          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_21               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_6          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_22               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_7          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_23               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_8          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_24               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_9          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_25               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_10         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_26               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_11         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_27               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_12         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_28               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_13         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_29               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_14         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_30               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_15         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_31               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_16         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_32               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase1_17         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_33               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_1          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_34               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_2          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_35               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_3          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_36               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_4          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_37               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_5          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_38               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_6          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_39               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_7          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_40               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_8          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_41               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_9          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_42               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_10         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_43               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_11         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_44               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_12         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_45               : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_13         : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_46               : signed(31 DOWNTO 0); -- sfix32_En31
```

```
SIGNAL product_phase2_14          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_47                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_15          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_48                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_16          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_49                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL product_phase2_17          : signed(30 DOWNTO 0); -- sfix31_En31
SIGNAL mul_temp_50                : signed(31 DOWNTO 0); -- sfix32_En31
SIGNAL quantized_sum              : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL sum1                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp                   : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum2                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_1                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum3                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_2                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum4                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_3                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum5                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_4                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum6                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_5                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum7                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_6                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum8                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_7                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum9                     : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_8                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum10                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_9                 : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum11                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_10                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum12                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_11                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum13                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_12                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum14                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_13                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum15                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_14                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum16                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_15                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum17                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_16                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum18                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_17                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum19                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_18                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum20                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_19                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum21                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_20                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum22                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_21                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum23                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_22                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum24                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_23                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum25                    : signed(33 DOWNTO 0); -- sfix34_En31
SIGNAL add_temp_24                : signed(34 DOWNTO 0); -- sfix35_En31
SIGNAL sum26                    : signed(33 DOWNTO 0); -- sfix34_En31
```

```
  SIGNAL add_temp_25                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum27                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_26                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum28                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_27                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum29                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_28                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum30                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_29                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum31                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_30                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum32                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_31                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum33                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_32                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum34                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_33                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum35                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_34                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum36                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_35                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum37                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_36                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum38                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_37                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum39                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_38                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum40                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_39                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum41                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_40                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum42                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_41                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum43                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_42                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum44                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_43                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum45                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_44                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum46                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_45                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum47                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_46                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum48                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_47                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum49                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_48                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL sum50                      : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL add_temp_49                : signed(34 DOWNTO 0); -- sfix35_En31
  SIGNAL output_typeconvert         : signed(33 DOWNTO 0); -- sfix34_En31
  SIGNAL output_register            : signed(33 DOWNTO 0); -- sfix34_En31


BEGIN

  -- Block Statements
  ce_output : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      ring_count <= to_unsigned(1, 3);
```

```
  ELSIF clk'event AND clk = '1' THEN
    IF clk_enable = '1' THEN
      ring_count <= ring_count(0) & ring_count(2 DOWNTO 1);
    END IF;
  END IF;
END PROCESS ce_output;

phase_0 <= ring_count(0)  AND clk_enable;

phase_1 <= ring_count(1)  AND clk_enable;

phase_2 <= ring_count(2)  AND clk_enable;

--   ------------------ CE Output Register ------------------

ce_output_register : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    ce_out_reg <= '0';
  ELSIF clk'event AND clk = '1' THEN
    ce_out_reg <= phase_2;

  END IF;
END PROCESS ce_output_register;

input_reg_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_register <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF clk_enable = '1' THEN
      input_register <= signed(filter_in);
    END IF;
  END IF;
END PROCESS input_reg_process;

Delay_Pipeline_Phase0_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase0(0 TO 15) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_2 = '1' THEN
      input_pipeline_phase0(0) <= input_register;
      input_pipeline_phase0(1 TO 15) <= input_pipeline_phase0(0 TO 14);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;


Delay_Pipeline_Phase1_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase1(0 TO 16) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      input_pipeline_phase1(0) <= input_register;
      input_pipeline_phase1(1 TO 16) <= input_pipeline_phase1(0 TO 15);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase1_process;
```

```
Delay_Pipeline_Phase2_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase2(0 TO 16) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_1 = '1' THEN
      input_pipeline_phase2(0) <= input_register;
      input_pipeline_phase2(1 TO 16) <= input_pipeline_phase2(0 TO 15);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase2_process;


mul_temp <= input_register * coeffphase1_1;
product_phase0_1 <= mul_temp(30 DOWNTO 0);

mul_temp_1 <= input_pipeline_phase0(0) * coeffphase1_2;
product_phase0_2 <= mul_temp_1(30 DOWNTO 0);

mul_temp_2 <= input_pipeline_phase0(1) * coeffphase1_3;
product_phase0_3 <= mul_temp_2(30 DOWNTO 0);

mul_temp_3 <= input_pipeline_phase0(2) * coeffphase1_4;
product_phase0_4 <= mul_temp_3(30 DOWNTO 0);

mul_temp_4 <= input_pipeline_phase0(3) * coeffphase1_5;
product_phase0_5 <= mul_temp_4(30 DOWNTO 0);

mul_temp_5 <= input_pipeline_phase0(4) * coeffphase1_6;
product_phase0_6 <= mul_temp_5(30 DOWNTO 0);

mul_temp_6 <= input_pipeline_phase0(5) * coeffphase1_7;
product_phase0_7 <= mul_temp_6(30 DOWNTO 0);

mul_temp_7 <= input_pipeline_phase0(6) * coeffphase1_8;
product_phase0_8 <= mul_temp_7(30 DOWNTO 0);

mul_temp_8 <= input_pipeline_phase0(7) * coeffphase1_9;
product_phase0_9 <= mul_temp_8(30 DOWNTO 0);

mul_temp_9 <= input_pipeline_phase0(8) * coeffphase1_10;
product_phase0_10 <= mul_temp_9(30 DOWNTO 0);

mul_temp_10 <= input_pipeline_phase0(9) * coeffphase1_11;
product_phase0_11 <= mul_temp_10(30 DOWNTO 0);

mul_temp_11 <= input_pipeline_phase0(10) * coeffphase1_12;
product_phase0_12 <= mul_temp_11(30 DOWNTO 0);

mul_temp_12 <= input_pipeline_phase0(11) * coeffphase1_13;
product_phase0_13 <= mul_temp_12(30 DOWNTO 0);

mul_temp_13 <= input_pipeline_phase0(12) * coeffphase1_14;
product_phase0_14 <= mul_temp_13(30 DOWNTO 0);

mul_temp_14 <= input_pipeline_phase0(13) * coeffphase1_15;
product_phase0_15 <= mul_temp_14(30 DOWNTO 0);
```

```
mul_temp_15 <= input_pipeline_phase0(14) * coeffphase1_16;
product_phase0_16 <= mul_temp_15(30 DOWNTO 0);

mul_temp_16 <= input_pipeline_phase0(15) * coeffphase1_17;
product_phase0_17 <= mul_temp_16(30 DOWNTO 0);

mul_temp_17 <= input_pipeline_phase1(0) * coeffphase2_1;
product_phase1_1 <= mul_temp_17(30 DOWNTO 0);

mul_temp_18 <= input_pipeline_phase1(1) * coeffphase2_2;
product_phase1_2 <= mul_temp_18(30 DOWNTO 0);

mul_temp_19 <= input_pipeline_phase1(2) * coeffphase2_3;
product_phase1_3 <= mul_temp_19(30 DOWNTO 0);

mul_temp_20 <= input_pipeline_phase1(3) * coeffphase2_4;
product_phase1_4 <= mul_temp_20(30 DOWNTO 0);

mul_temp_21 <= input_pipeline_phase1(4) * coeffphase2_5;
product_phase1_5 <= mul_temp_21(30 DOWNTO 0);

mul_temp_22 <= input_pipeline_phase1(5) * coeffphase2_6;
product_phase1_6 <= mul_temp_22(30 DOWNTO 0);

mul_temp_23 <= input_pipeline_phase1(6) * coeffphase2_7;
product_phase1_7 <= mul_temp_23(30 DOWNTO 0);

mul_temp_24 <= input_pipeline_phase1(7) * coeffphase2_8;
product_phase1_8 <= mul_temp_24(30 DOWNTO 0);

mul_temp_25 <= input_pipeline_phase1(8) * coeffphase2_9;
product_phase1_9 <= mul_temp_25(30 DOWNTO 0);

mul_temp_26 <= input_pipeline_phase1(9) * coeffphase2_10;
product_phase1_10 <= mul_temp_26(30 DOWNTO 0);

mul_temp_27 <= input_pipeline_phase1(10) * coeffphase2_11;
product_phase1_11 <= mul_temp_27(30 DOWNTO 0);

mul_temp_28 <= input_pipeline_phase1(11) * coeffphase2_12;
product_phase1_12 <= mul_temp_28(30 DOWNTO 0);

mul_temp_29 <= input_pipeline_phase1(12) * coeffphase2_13;
product_phase1_13 <= mul_temp_29(30 DOWNTO 0);

mul_temp_30 <= input_pipeline_phase1(13) * coeffphase2_14;
product_phase1_14 <= mul_temp_30(30 DOWNTO 0);

mul_temp_31 <= input_pipeline_phase1(14) * coeffphase2_15;
product_phase1_15 <= mul_temp_31(30 DOWNTO 0);

mul_temp_32 <= input_pipeline_phase1(15) * coeffphase2_16;
product_phase1_16 <= mul_temp_32(30 DOWNTO 0);

mul_temp_33 <= input_pipeline_phase1(16) * coeffphase2_17;
product_phase1_17 <= mul_temp_33(30 DOWNTO 0);

mul_temp_34 <= input_pipeline_phase2(0) * coeffphase3_1;
product_phase2_1 <= mul_temp_34(30 DOWNTO 0);
```

```
mul_temp_35 <= input_pipeline_phase2(1) * coeffphase3_2;
product_phase2_2 <= mul_temp_35(30 DOWNTO 0);

mul_temp_36 <= input_pipeline_phase2(2) * coeffphase3_3;
product_phase2_3 <= mul_temp_36(30 DOWNTO 0);

mul_temp_37 <= input_pipeline_phase2(3) * coeffphase3_4;
product_phase2_4 <= mul_temp_37(30 DOWNTO 0);

mul_temp_38 <= input_pipeline_phase2(4) * coeffphase3_5;
product_phase2_5 <= mul_temp_38(30 DOWNTO 0);

mul_temp_39 <= input_pipeline_phase2(5) * coeffphase3_6;
product_phase2_6 <= mul_temp_39(30 DOWNTO 0);

mul_temp_40 <= input_pipeline_phase2(6) * coeffphase3_7;
product_phase2_7 <= mul_temp_40(30 DOWNTO 0);

mul_temp_41 <= input_pipeline_phase2(7) * coeffphase3_8;
product_phase2_8 <= mul_temp_41(30 DOWNTO 0);

mul_temp_42 <= input_pipeline_phase2(8) * coeffphase3_9;
product_phase2_9 <= mul_temp_42(30 DOWNTO 0);

mul_temp_43 <= input_pipeline_phase2(9) * coeffphase3_10;
product_phase2_10 <= mul_temp_43(30 DOWNTO 0);

mul_temp_44 <= input_pipeline_phase2(10) * coeffphase3_11;
product_phase2_11 <= mul_temp_44(30 DOWNTO 0);

mul_temp_45 <= input_pipeline_phase2(11) * coeffphase3_12;
product_phase2_12 <= mul_temp_45(30 DOWNTO 0);

mul_temp_46 <= input_pipeline_phase2(12) * coeffphase3_13;
product_phase2_13 <= mul_temp_46(30 DOWNTO 0);

mul_temp_47 <= input_pipeline_phase2(13) * coeffphase3_14;
product_phase2_14 <= mul_temp_47(30 DOWNTO 0);

mul_temp_48 <= input_pipeline_phase2(14) * coeffphase3_15;
product_phase2_15 <= mul_temp_48(30 DOWNTO 0);

mul_temp_49 <= input_pipeline_phase2(15) * coeffphase3_16;
product_phase2_16 <= mul_temp_49(30 DOWNTO 0);

mul_temp_50 <= input_pipeline_phase2(16) * coeffphase3_17;
product_phase2_17 <= mul_temp_50(30 DOWNTO 0);

quantized_sum <= resize(product_phase2_1, 34);

add_temp <= resize(quantized_sum, 35) + resize(product_phase2_2, 35);
sum1 <= add_temp(33 DOWNTO 0);

add_temp_1 <= resize(sum1, 35) + resize(product_phase2_3, 35);
sum2 <= add_temp_1(33 DOWNTO 0);

add_temp_2 <= resize(sum2, 35) + resize(product_phase2_4, 35);
sum3 <= add_temp_2(33 DOWNTO 0);

add_temp_3 <= resize(sum3, 35) + resize(product_phase2_5, 35);
```

```
sum4 <= add_temp_3(33 DOWNTO 0);

add_temp_4 <= resize(sum4, 35) + resize(product_phase2_6, 35);
sum5 <= add_temp_4(33 DOWNTO 0);

add_temp_5 <= resize(sum5, 35) + resize(product_phase2_7, 35);
sum6 <= add_temp_5(33 DOWNTO 0);

add_temp_6 <= resize(sum6, 35) + resize(product_phase2_8, 35);
sum7 <= add_temp_6(33 DOWNTO 0);

add_temp_7 <= resize(sum7, 35) + resize(product_phase2_9, 35);
sum8 <= add_temp_7(33 DOWNTO 0);

add_temp_8 <= resize(sum8, 35) + resize(product_phase2_10, 35);
sum9 <= add_temp_8(33 DOWNTO 0);

add_temp_9 <= resize(sum9, 35) + resize(product_phase2_11, 35);
sum10 <= add_temp_9(33 DOWNTO 0);

add_temp_10 <= resize(sum10, 35) + resize(product_phase2_12, 35);
sum11 <= add_temp_10(33 DOWNTO 0);

add_temp_11 <= resize(sum11, 35) + resize(product_phase2_13, 35);
sum12 <= add_temp_11(33 DOWNTO 0);

add_temp_12 <= resize(sum12, 35) + resize(product_phase2_14, 35);
sum13 <= add_temp_12(33 DOWNTO 0);

add_temp_13 <= resize(sum13, 35) + resize(product_phase2_15, 35);
sum14 <= add_temp_13(33 DOWNTO 0);

add_temp_14 <= resize(sum14, 35) + resize(product_phase2_16, 35);
sum15 <= add_temp_14(33 DOWNTO 0);

add_temp_15 <= resize(sum15, 35) + resize(product_phase2_17, 35);
sum16 <= add_temp_15(33 DOWNTO 0);

add_temp_16 <= resize(sum16, 35) + resize(product_phase1_1, 35);
sum17 <= add_temp_16(33 DOWNTO 0);

add_temp_17 <= resize(sum17, 35) + resize(product_phase1_2, 35);
sum18 <= add_temp_17(33 DOWNTO 0);

add_temp_18 <= resize(sum18, 35) + resize(product_phase1_3, 35);
sum19 <= add_temp_18(33 DOWNTO 0);

add_temp_19 <= resize(sum19, 35) + resize(product_phase1_4, 35);
sum20 <= add_temp_19(33 DOWNTO 0);

add_temp_20 <= resize(sum20, 35) + resize(product_phase1_5, 35);
sum21 <= add_temp_20(33 DOWNTO 0);

add_temp_21 <= resize(sum21, 35) + resize(product_phase1_6, 35);
sum22 <= add_temp_21(33 DOWNTO 0);

add_temp_22 <= resize(sum22, 35) + resize(product_phase1_7, 35);
sum23 <= add_temp_22(33 DOWNTO 0);

add_temp_23 <= resize(sum23, 35) + resize(product_phase1_8, 35);
```

```
sum24 <= add_temp_23(33 DOWNTO 0);

add_temp_24 <= resize(sum24, 35) + resize(product_phase1_9, 35);
sum25 <= add_temp_24(33 DOWNTO 0);

add_temp_25 <= resize(sum25, 35) + resize(product_phase1_10, 35);
sum26 <= add_temp_25(33 DOWNTO 0);

add_temp_26 <= resize(sum26, 35) + resize(product_phase1_11, 35);
sum27 <= add_temp_26(33 DOWNTO 0);

add_temp_27 <= resize(sum27, 35) + resize(product_phase1_12, 35);
sum28 <= add_temp_27(33 DOWNTO 0);

add_temp_28 <= resize(sum28, 35) + resize(product_phase1_13, 35);
sum29 <= add_temp_28(33 DOWNTO 0);

add_temp_29 <= resize(sum29, 35) + resize(product_phase1_14, 35);
sum30 <= add_temp_29(33 DOWNTO 0);

add_temp_30 <= resize(sum30, 35) + resize(product_phase1_15, 35);
sum31 <= add_temp_30(33 DOWNTO 0);

add_temp_31 <= resize(sum31, 35) + resize(product_phase1_16, 35);
sum32 <= add_temp_31(33 DOWNTO 0);

add_temp_32 <= resize(sum32, 35) + resize(product_phase1_17, 35);
sum33 <= add_temp_32(33 DOWNTO 0);

add_temp_33 <= resize(sum33, 35) + resize(product_phase0_1, 35);
sum34 <= add_temp_33(33 DOWNTO 0);

add_temp_34 <= resize(sum34, 35) + resize(product_phase0_2, 35);
sum35 <= add_temp_34(33 DOWNTO 0);

add_temp_35 <= resize(sum35, 35) + resize(product_phase0_3, 35);
sum36 <= add_temp_35(33 DOWNTO 0);

add_temp_36 <= resize(sum36, 35) + resize(product_phase0_4, 35);
sum37 <= add_temp_36(33 DOWNTO 0);

add_temp_37 <= resize(sum37, 35) + resize(product_phase0_5, 35);
sum38 <= add_temp_37(33 DOWNTO 0);

add_temp_38 <= resize(sum38, 35) + resize(product_phase0_6, 35);
sum39 <= add_temp_38(33 DOWNTO 0);

add_temp_39 <= resize(sum39, 35) + resize(product_phase0_7, 35);
sum40 <= add_temp_39(33 DOWNTO 0);

add_temp_40 <= resize(sum40, 35) + resize(product_phase0_8, 35);
sum41 <= add_temp_40(33 DOWNTO 0);

add_temp_41 <= resize(sum41, 35) + resize(product_phase0_9, 35);
sum42 <= add_temp_41(33 DOWNTO 0);

add_temp_42 <= resize(sum42, 35) + resize(product_phase0_10, 35);
sum43 <= add_temp_42(33 DOWNTO 0);

add_temp_43 <= resize(sum43, 35) + resize(product_phase0_11, 35);
```

```
  sum44 <= add_temp_43(33 DOWNTO 0);

  add_temp_44 <= resize(sum44, 35) + resize(product_phase0_12, 35);
  sum45 <= add_temp_44(33 DOWNTO 0);

  add_temp_45 <= resize(sum45, 35) + resize(product_phase0_13, 35);
  sum46 <= add_temp_45(33 DOWNTO 0);

  add_temp_46 <= resize(sum46, 35) + resize(product_phase0_14, 35);
  sum47 <= add_temp_46(33 DOWNTO 0);

  add_temp_47 <= resize(sum47, 35) + resize(product_phase0_15, 35);
  sum48 <= add_temp_47(33 DOWNTO 0);

  add_temp_48 <= resize(sum48, 35) + resize(product_phase0_16, 35);
  sum49 <= add_temp_48(33 DOWNTO 0);

  add_temp_49 <= resize(sum49, 35) + resize(product_phase0_17, 35);
  sum50 <= add_temp_49(33 DOWNTO 0);

  output_typeconvert <= sum50;

  output_register_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      output_register <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF phase_2 = '1' THEN
        output_register <= output_typeconvert;
      END IF;
    END IF;
  END PROCESS output_register_process;

  -- Assignment Statements
  ce_out <= ce_out_reg;
  filter_out <= std_logic_vector(output_register);
END rtl;
```

**Listagem A.3: Código VHDL utilizado para implementação do Interpolador.**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


library auk_dspip_lib;
use auk_dspip_lib.auk_dspip_lib_pkg_fir_90.all;


entity interpolador_ast is

  port(
    clk              : in  std_logic;
    reset_n          : in  std_logic;
    ast_sink_ready   : out std_logic;
    ast_source_data  : out std_logic_vector (29 -1 downto 0);
    ast_sink_data    : in  std_logic_vector (16 -1 downto 0);
    ast_sink_valid   : in  std_logic;
    ast_source_valid : out std_logic;
    ast_source_ready : in  std_logic;
    ast_sink_error   : in  std_logic_vector (1 downto 0);
    ast_source_error : out std_logic_vector (1 downto 0)
    );
attribute altera_attribute : string;
attribute altera_attribute of interpolador_ast:entity is "-name MESSAGE_DISABLE 15400; -name MESSAGE_DISABLE 14130; -name MESSAGE_DISABLE 12020; -name MESSAGE_DISABLE 12030; -name MESSAGE_DISABLE 12010; -name MESSAGE_DISABLE 12110; -name MESSAGE_DISABLE 14320; -name MESSAGE_DISABLE 13410; -name MESSAGE_DISABLE 10036";
end interpolador_ast;


-- Warnings Suppression On
-- altera message_off 10036


architecture struct of interpolador_ast is

  signal sink_packet_error   : std_logic_vector(1 downto 0);
  signal data_in             : std_logic_vector(16 -1 downto 0);
  signal data_out            : std_logic_vector(29 -1 downto 0);
  signal core_out            : std_logic_vector(29 -1 downto 0);
  signal ready               : std_logic;
  signal reset_fir           : std_logic;
  signal sink_ready_ctrl     : std_logic;
  signal sink_stall          : std_logic;
  signal source_packet_error : std_logic_vector(1 downto 0);
  signal source_stall        : std_logic;
  signal source_valid_ctrl   : std_logic;
  signal stall               : std_logic;
  signal valid               : std_logic;
  signal core_valid          : std_logic;
  signal enable_in           : std_logic;
  signal stall_delayed       : std_logic;
  constant ENABLE_PIPELINE_DEPTH_c : natural := 0;

  component interpolador_st_wr is
    port (
      rst        : in std_logic;
      clk        : in std_logic;
```

```
    clk_en      : in std_logic;
    rdy_to_ld   : out std_logic;
    done        : out std_logic;
       data_in      : in  std_logic_vector(16 - 1 downto 0);
    fir_result   : out std_logic_vector(29 - 1 downto 0));
 end component interpolador_st_wr;

begin
 sink : auk_dspip_avalon_streaming_sink_fir_90
  generic map (
    WIDTH_g         => 16,
    PACKET_SIZE_g    => 1,
    FAMILY_g         => "Cyclone",
    MEM_TYPE_g       => "Auto")
  port map (
    clk            => clk,
    reset_n         => reset_n,
    data           => data_in,
    sink_ready_ctrl => sink_ready_ctrl,
    sink_stall      => sink_stall,
    packet_error    => sink_packet_error,
    at_sink_ready   => ast_sink_ready,
    at_sink_valid   => ast_sink_valid,
    at_sink_data    => ast_sink_data,
    at_sink_error   => ast_sink_error);

 source : auk_dspip_avalon_streaming_source_fir_90
  generic map (
    WIDTH_g         => 29,
    packet_size_g   => 1)
  port map (
    clk              => clk,
    reset_n           => reset_n,
    data             => data_out,
    source_valid_ctrl => source_valid_ctrl,
    design_stall      => stall_delayed,
    source_stall      => source_stall,
    packet_error      => source_packet_error,
    at_source_ready   => ast_source_ready,
    at_source_valid   => ast_source_valid,
    at_source_data    => ast_source_data,
    at_source_error   => ast_source_error);

 intf_ctrl : auk_dspip_avalon_streaming_controller_fir_90
  port map (
    clk               => clk,
    ready              => ready,
    reset_n            => reset_n,
    sink_packet_error   => sink_packet_error,
    sink_stall          => sink_stall,
    source_stall        => source_stall,
    valid              => valid,
    reset_design        => reset_fir,
    sink_ready_ctrl     => sink_ready_ctrl,
    source_packet_error => source_packet_error,
    source_valid_ctrl   => source_valid_ctrl,
    stall              => stall);

 fircore: interpolador_st_wr
  port map (
```

```
    rst          => reset_fir,
    clk          => clk,
    clk_en       => enable_in,
    rdy_to_ld    => ready,
    done         => core_valid,
       data_in      => data_in,
    fir_result   => core_out);


 data_out <= core_out;
  valid <= core_valid;

enable_in <= not stall;

no_enable_pipeline: if ENABLE_PIPELINE_DEPTH_c = 0 generate
   stall_delayed <= stall;
end generate no_enable_pipeline;

enable_pipeline: if ENABLE_PIPELINE_DEPTH_c > 0 generate
 delay_core_enable : process (clk, reset_n)
   variable stall_delay : std_logic_vector(ENABLE_PIPELINE_DEPTH_c downto 0);
 begin  -- process delay_core_enable
   if reset_n = '0' then
     stall_delay := (others => '0');
   elsif rising_edge(clk) then
     stall_delay := stall_delay(stall_delay'high-1 downto 0) & stall;
   end if;
   stall_delayed <= stall_delay(stall_delay'high);
 end process delay_core_enable;
end generate enable_pipeline;
end struct;
```