Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

FlawQ: *Plugin* de Equalização Gráfica de Áudio

Autor:

Felipe Castro Vieira Martins

Orientador:

Prof. Luiz Wagner Pereira Biscainho, D.Sc.

Co-Orientador:

Leonardo de Oliveira Nunes, M.Sc.

Examinador:

Prof. Sergio Lima Netto, Ph.D.

Examinador:

Filipe Castello da Costa Beltrão Diniz, D.Sc.

DEL

Março de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

DEDICATÓRIA

Dedico este trabalho a todas as pessoas que confiaram em mim e me apoiaram durante toda a graduação.

AGRADECIMENTO

Nestes 5 anos de graduação de intenso estudo e dedicação, agradeço fortemente a todos que contribuíram para essa enorme conquista de formação pessoal e profissional.

Agradeço infinitamente aos meus pais, Carlos e Nádia, por ter tido a oportunidade de estudar e concluir minha graduação nesta universidade, pelo apoio financeiro e moral dado a mim durante todos esses anos.

A toda a minha família e aos amigos de Valença pela força e por compreenderem momentos de ausência da minha parte devido ao tempo necessário de dedicação aos estudos.

Ao meu irmão Fernando pela total amizade e cumplicidade com que eu sei que posso contar a vida toda.

Obrigado a todos os amigos que ganhei ao longo dessa jornada e que me apoiaram do início ao fim do curso, em especial aos amigos: Danilo, Marcelo Domingues, Ramón, Marcelo Larcher, André Mendes, Dmitri, Michel, Lyno e Thiago Signorelli.

Um agradecimento bastante especial ao meu orientador Luiz Wagner pela sua paciência, dedicação e amizade.

Ao meu professor e amigo Carlos Eduardo Belmonte por toda a ajuda e apoio moral dado ao longo desses anos, em assuntos pessoais ou acadêmicos.

Agradeço também à toda a equipe do LPS (Laboratório de Processamento de Sinais), em especial aos membros do GPA (Grupo de Processamento de Áudio).

Ao meu co-orientador, Leonardo Nunes, pela imensa ajuda dada em todo o projeto, assim como pela sua amizade.

Ao amigo Alan Tygel pelas imensas contribuições dadas a este trabalho e ao amigo Vicente Júnior por imensas contribuições na elaboração da arte do *Plugin*.

Agradeço ao Professor Sergio Lima Netto e ao Engenheiro Filipe Diniz por terem aceitado o convite para participarem da banca.

A todo o corpo docente que compõe o Departamento de Eletrônica e Computação da UFRJ, obrigado.

RESUMO

Este trabalho consiste na modificação e implementação do projeto de um equalizador gráfico digital de áudio destinado à operação até 20 kHz. A arquitetura se baseia num projeto industrial originalmente voltado para implementação em tempo real via DSP, com canais espaçados de forma aproximadamente linear por oitava. A solução modificada aqui proposta foi concebida como um *plugin* VST, com interface gráfica amigável, e inclui como facilidade adicional a visualização da resposta na frequência resultante do equalizador com os ganhos escolhidos pelo usuário. Outras modificações que foram realizadas tinham como objetivo melhorar o desempenho do equalizador e melhorar a sua resposta na frequência.

Palavras-Chave: equalizador, banco de filtros, plugin, áudio, multitaxa.

SIGLAS

CD - *Compact Disc*

FFB - *Fast Filter Bank*

FFT - *Fast Fourier Transform*

FFTW - *Fastest Fourier Transform in the West*

FIR - *Finite-Length Impulse Response*

FRM - *Frequency-Response Masking*

LP - *Long-Play*

PC - *Personal Computer*

VST - *Virtual Studio Technology*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Breve Histórico do Som Gravado

A história do som gravado remonta à segunda metade do século XIX, com a gravação em cilindros, que deram lugar aos discos no início do século XX. O próximo passo importante foi a gravação elétrica, na década de 1920. No pós-guerra, assistiu-se ao advento do LP e se difundiu a gravação em meio magnético. A partir de fins dos anos 1950, utilizou-se comercialmente a estereofonia. Daí em diante, na era analógica, tudo o que se viu foram aperfeiçoamentos de técnicas já conhecidas. O grande salto tecnológico-científico na história do áudio gravado foi o emprego do processamento digital de sinais, cujas bases vêm da primeira metade do século XX. Somente na década de 1970 a maior disponibilidade de recursos computacionais permitiu que ele fosse utilizado comercialmente em aplicações de áudio. Em 1982 foi lançado o CD, ainda hoje um padrão comercial de áudio de alta qualidade.

1.2 Processamento Digital de Áudio

A era digital tornou possível armazenar o áudio de forma confiável e repetível, com controle sobre o binômio qualidade *versus* quantidade; resgatar ou até superar a qualidade de material preexistente; extrair do som gravado informações de diversos níveis; e modificar e sintetizar som. Isso abriu para o processamento digital de áudio um espectro inesgotável de aplicações, associadas desde à música até às telecomunicações. O processamento digital de sinais envolve o uso intensivo do

ferramental genericamente agrupado sob o nome de Sinais e Sistemas [3] (representações de sinais e sistemas tanto no domínio do tempo quanto no da frequência e transformadas), com ênfase ao tratamento de sinais discretos [4]. Sua aplicação ao áudio [5] intensificou-se extraordinariamente nos últimos 20 anos.

1.3 Equalizadores

A audição de um ser humano jovem compreende a faixa de frequências de 20 Hz a 20 kHz, aproximadamente. A percepção de *pitch* (altura do som), por sua vez, é aproximadamente geométrica na frequência: o que se percebe como uma nota musical se repete a cada vez que dobra a frequência. Em sistemas de reprodução de áudio, é muito comum a necessidade de corrigir a resposta de frequências do ambiente em que se dá a audição (ou simplesmente modificar o conteúdo espectral de acordo com o gosto pessoal do ouvinte). Bancos de filtros com esse fim compõem os chamados "equalizadores". Estes devem abranger de preferência toda a faixa audível, bem como ajustar a seletividade em frequência de seus canais à percepção humana.

Equalizadores paramétricos permitem um ajuste mais fino na resposta em frequência desejada, pelo ajuste da largura e da frequência central da faixa de passagem e do ganho[1] de cada filtro. Equalizadores gráficos fornecem o controle de ganho sobre a faixa de passagem de cada filtro, de largura e frequência central fixas e pré-definidas. Os Equalizadores Gráficos são de menor complexidade de uso que os Paramétricos, o que torna sua aplicação mais adequada e eficiente quando utilizados em sistemas tradicionais de áudio.

O presente trabalho consiste na implementação de um Equalizador Gráfico. Projetos analógicos dessa natureza são conhecidos desde a segunda metade do século XX, implementados com filtros passivos e ativos. Mas a evolução tecnológica que revolucionou as aplicações de processamento digital de sinais também chegou ao áudio. Além de ter aplicação em som doméstico e automotivo, a equalização digital

[1] O ganho, geralmente em decibéis, determina quanto o sinal será amplificado/atenuado.

é componente essencial em sistemas de áudio tendo PCs como plataforma — quando requer uma interface gráfica adequada.

1.4 Objetivo do Projeto

Como vimos, a evolução dos processadores digitais no último quarto do século XX permitiu a aproximação entre as aplicações de ciência avançada e o usuário comum. Na área de áudio, o processamento digital permeia desde os equipamentos domésticos de som até os diversos aplicativos para manipulação e reprodução de áudio disponíveis para computadores pessoais. Este trabalho tem como objetivo mostrar o uso de uma ferramenta avançada de filtragem numa aplicação típica de áudio que possa ser facilmente utilizada por um profissional sem a necessidade de conhecimento especializado em processamento de sinais. Em particular, será apresentado o procedimento de projeto do FlawQ, um equalizador gráfico digital de 10 oitavas baseado em uma estrutura multitaxa. Esta solução foi escolhida por sua baixa complexidade computacional, uma vez que uma das especificações do equalizador era a operação em tempo real. A fim de permitir a fácil utilização e portabilidade do sistema, utilizou-se o padrão de *plugin* VST[2], amplamente aceito por fabricantes e usuários de aplicativos de áudio profissional.

1.5 Estrutura do Texto

Após esta introdução, toda a descrição teórica envolvida no projeto desde o projeto dos filtros até a equalização em multitaxa está descrita no Capítulo 2. Neste capítulo são discutidos o projeto do equalizador que divide o espectro de frequências linearmente, a estrutura em multitaxa e a formação da estrutura completa.

No Capítulo 3 são apresentadas as alterações propostas para o projeto do equalizador linear visando à melhora do desempenho e da qualidade do sistema, além de uma modificação para correção de um problema estrutural do sistema completo.

[2]A marca VST (*Virtual Studio Technology*) é propriedade da Steinberg Co.

O Capítulo 4 trata da implementação de todo o projeto, desde a construção das classes necessárias para compor cada bloco do sistema até a criação do correspondente *plugin* VST com interface gráfica.

Por fim, o Capítulo 5 apresenta as conclusões do projeto e propõe trabalhos futuros.

Capítulo 2

Descrição Teórica

Neste capítulo será abordada a teoria envolvida na implementação do FlawQ desde o projeto das estruturas em multitaxa, passando pelo equalizador que divide o espectro de frequências linearmente, até o projeto dos filtros antiimagem e *antialiasing*. Este capítulo é baseado no artigo [6], coescrito pelo autor do presente trabalho.

A Seção 2.1 irá abordar a descrição da estrutura em multitaxa adotada no projeto. A Seção 2.2 irá tratar sobre o equalizador que divide o espectro de frequências de forma linear utilizando um banco de filtros e por fim, na Seção 2.3, trata-se da estrutura completa formada pelo equalizador em multitaxa operando em conjunto com o equalizador linear proposto.

2.1 Equalização em Multitaxa

Essa seção realiza uma breve descrição da estrutura do equalizador implementado neste trabalho, originalmente proposta em [1], e de algumas modificações realizadas sobre aquele projeto inicial.

A seletividade frequencial da audição humana é aproximadamente logarítmica com a frequência, seguindo a mesma organização das chamadas bandas críticas [7]. Assim sendo, pode ser vantajoso dividir o espectro de frequências de forma logarítmica, favorecendo o uso de processamento em multitaxa. O equalizador deste projeto foi concebido [1] na forma de uma estrutura modular de banco de filtros em multitaxa, mostrada nas Figuras 2.1 e 2.2 para um número arbitrário K de sub-bandas.

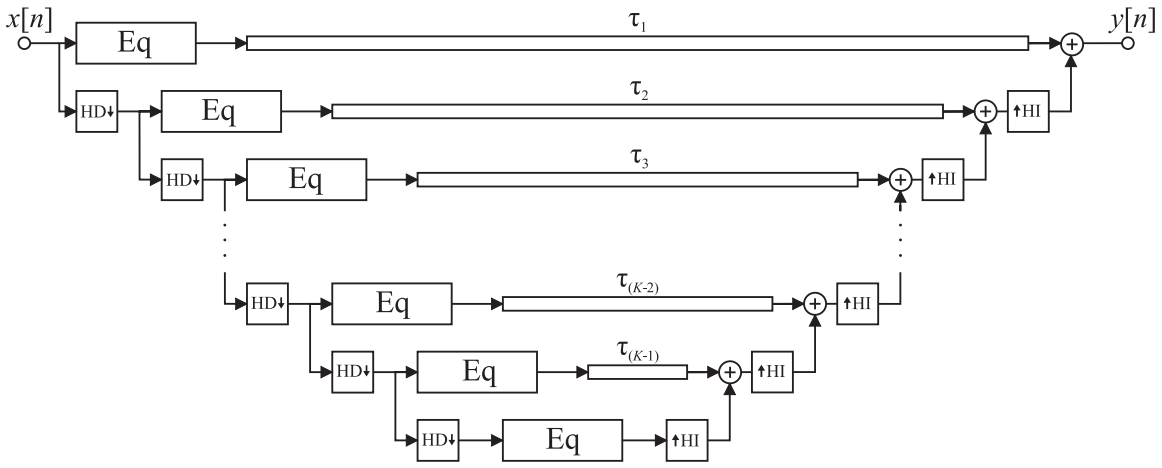

Figura 2.1: Diagrama de blocos da estrutura em multitaxa. Adaptado de [1]. Os blocos HD↓ e ↑HI são expandidos na Figura 2.2.

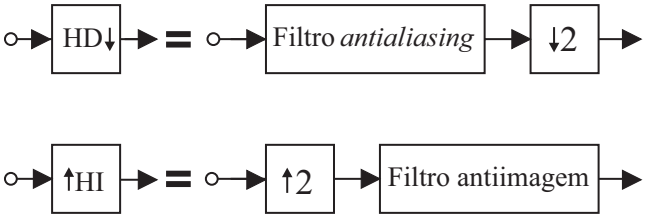

Figura 2.2: Expansão dos blocos HD↓ e ↑HI da Figura 2.1.

Cada um dos ramos da estrutura em multitaxa (exceto o primeiro) é composto por:

filtro *antialiasing* – filtro FIR (do inglês, *finite-length impulse response*) passa-baixas com frequência de corte igual a 0,5π rad/amostra[1];

subamostrador por 2 ($\downarrow 2$) – módulo que remove uma de cada duas amostras do sinal no domínio do tempo;

equalizador linear (Eq) – equalizador gráfico linear com os centros de suas bandas de atuação espaçados linearmente na frequência;

bloco de atraso (τ_k) – módulo que atrasa o sinal no tempo de um número de amostras pré-especificado para o ramo k;

superamostrador por 2 ($\uparrow 2$) – módulo que insere um zero entre cada duas amostras do sinal no tempo;

filtro antiimagem – filtro FIR passa-baixas com frequência de corte igual a 0,5π rad/amostra.

Partindo de um primeiro ramo que opera sobre todo o espectro do sinal $x[n]$, o sistema é desdobrado em ramos aninhados que sequencialmente separam e processam a metade inferior do espectro. Assim, a partir do primeiro, cada ramo opera exatamente sobre a metade inferior do espectro do sinal do ramo imediatamente superior. O sinal de saída $y[n]$ é a soma dos sinais processados em cada nível.

O equalizador linear (indicado na figura por `Eq`) presente em cada ramo possui 9 canais, o primeiro e o último com largura de banda igual à metade da largura de banda dos demais (Figura 2.4). A faixa total de operação desses equalizadores é função do nível ao qual pertence, que também define sua taxa de processamento. Para um equalizador no k-ésimo ramo, onde $k = 1$ representa o ramo que opera na taxa mais alta, a faixa total de operação vai de 0 a $2^{-(k-1)}\pi$ rad/amostra.

[1]As referências a frequências aqui correspondem à frequência angular normalizada em rad/amostra: 2π rad/amostra representa a frequência de amostragem.

Essa estrutura consome um número reduzido de operações aritméticas, pois cada ramo opera apenas na metade da taxa do ramo imediatamente superior. Além disso, sua modularidade permite que o número de subdivisões em 'meias-bandas' possa ser facilmente aumentado, com o consequente aumento na resolução de frequência disponibilizada ao usuário para montar as curvas de equalização.

Todos os filtros da estrutura foram projetados como FIR de fase linear [4].

Os filtros utilizados para prevenção do *aliasing* (causado pela subamostragem) e eliminação das imagens replicadas (causadas pela superamostragem) são filtros passa-baixas idênticos com as seguintes especificações:

- faixa de passagem até 0,4535 rad/amostra;
- faixa de rejeição a partir de 0,5 rad/amostra;
- ordem 140.

O projeto foi feito por otimização *least-squares*, e atingiu a resposta mostrada na Figura 2.3, com uma atenuação na faixa de rejeição maior que 96 dB e um *ripple* na faixa de passagem menor que 0,46 dB. Maior detalhamento sobre o projeto deste filtro pode ser encontrado na Seção A.1.

A operação em diferentes taxas gera atrasos diferentes em cada nível da estrutura multitaxa. Para que a soma dos sinais oriundos de dois ramos diferentes seja coerente, é necessário sincronizar a saída do ramo com a saída do ramo imediatamente abaixo na estrutura. Por isso, é necessário aplicar um atraso adequado na saída dos equalizadores lineares em cada ramo, dependente do atraso resultante do ramo imediatamente abaixo e do atraso já implícito nos demais blocos do ramo em questão. Como todos os filtros envolvidos são filtros FIR de fase linear, esses atrasos são facilmente calculados. A expressão abaixo mostra o cálculo do atraso τ_k do k-ésimo ramo em função dos atrasos τ_{k+1} do ramo seguinte, τ_{eq} do equalizador, τ_{aa} do filtro *antialiasing* e τ_{ai} do filtro antiimagem:

$$\tau_k = 2\tau_{k+1} + \tau_{\mathrm{eq}} + \tau_{\mathrm{aa}} + \tau_{\mathrm{ai}}. \tag{2.1}$$

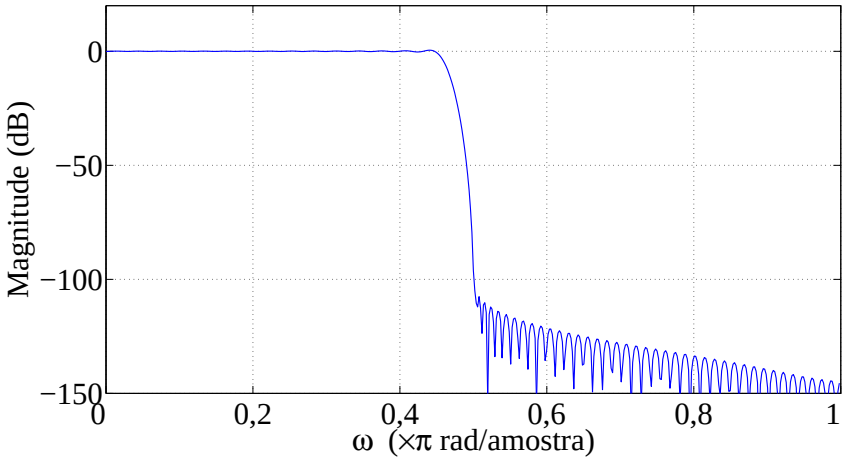

Figura 2.3: Resposta em frequência de magnitude do filtro utilizado como *antialiasing* e antiimagem.

Deve-se observar que o atraso do último ramo pode ser feito nulo, isto é, $\tau_K = 0$. Neste projeto, $\tau_{\text{aa}} = \tau_{\text{ai}} = 70$, que é metade da ordem dos filtros em questão.

O número de operações aritméticas por amostra do sinal de entrada por ramo pode ser obtido a partir do comprimento dos filtros *antialiasing* e antiimagem e do número de operações realizadas pelo equalizador. O número de adições reais por amostra do sinal de entrada no ramo k é

$$\nu_{\text{A},k} = \frac{\nu_{\text{A,eq}} + \nu_{\text{A,aa}} + \nu_{\text{A,ai}} + 1}{2^{k-1}}, \tag{2.2}$$

onde $\nu_{\text{A,eq}}$ é o número de adições realizadas pelo equalizador, que será calculado na próxima seção; e $\nu_{\text{A,aa}}$ e $\nu_{\text{A,ai}}$ são os números de adições realizadas pelos filtros *antialiasing* e antiimagem, respectivamente. Já o número de multiplicações reais por amostra do sinal de entrada no ramo k pode ser definido como

$$\nu_{\text{M},k} = \frac{\nu_{\text{M,eq}} + \nu_{\text{M,aa}} + \nu_{\text{M,ai}}}{2^{k-1}}, \tag{2.3}$$

onde $\nu_{\text{M,eq}}$ é o número de multiplicações realizadas pelo equalizador, que será calculado na próxima seção; e $\nu_{\text{M,aa}}$ e $\nu_{\text{M,ai}}$ são os números de multiplicações realizadas pelos filtros *antialiasing* e antiimagem, respectivamente, que são filtros simétricos.

O número de adições realizadas $\nu_{\mathrm{A,x}}$ pode ser calculado pela expressão

$$\nu_{\mathrm{A,x}} = N_{\mathrm{x}} - 1, \text{com x = aa ou ai}, \tag{2.4}$$

sendo N_{x} o comprimento do filtro x em amostras. O número de multiplicações $\nu_{\mathrm{M,x}}$ realizadas por estes filtros é

$$\nu_{\mathrm{M,x}} = N_{\mathrm{x}} - \frac{N_{\mathrm{x}} - 1}{2} = \frac{N_{\mathrm{x}} + 1}{2}, \tag{2.5}$$

onde o termo $\frac{N_{\mathrm{x}}-1}{2}$ pôde ser subtraído do total de multiplicações devido à simetria do filtro. Neste projeto, $\nu_{\mathrm{A,ai}} = \nu_{\mathrm{A,aa}} = 140$ adições por amostra e $\nu_{\mathrm{M,ai}} = \nu_{\mathrm{M,aa}} = 71$ multiplicações por amostra. Os números de adições e multiplicações da estrura completa são, então, respectivamente,

$$\nu_{\mathrm{A}} = \sum_{k=1}^{K} \nu_{\mathrm{A},k} \tag{2.6}$$

e

$$\nu_{\mathrm{M}} = \sum_{k=1}^{K} \nu_{\mathrm{M},k}. \tag{2.7}$$

Percebe-se que a complexidade computacional cresce, aproximadamente, de forma linear com o comprimento dos filtros *antialiasing* e antiimagem e a complexidade do equalizador. Este projeto utiliza 6 ramos na estrutura multitaxa, fazendo com que a faixa espectral de operação do equalizador seja subdividida sucessivamente em cinco meias-bandas. Isso implica um pequeno acréscimo na complexidade do sistema em relação ao projeto original [1], que se compunha de apenas 4 ramos.

Resumidamente, a estrutura completa do equalizador é composta pelos blocos de decimação e interpolação (HD$\downarrow$ e $\uparrow$HI), pelos equalizadores lineares (que dividem o espectro de frequências em 9 bandas e serão detalhados na Seção 2.2) e pelos atrasos adicionados em cada ramo. A Figura 2.1 mostra a estrutura em multitaxa completa com K ramos, identificando todos os blocos descritos. Sua filosofia é utilizar equalizadores idênticos que dividem o espectro de forma linear operando em diferentes taxas, subordinados à macrodivisão logarítmica do espectro. A Seção 2.2 irá abordar o projeto e o funcionamento do equalizador linear proposto em [1].

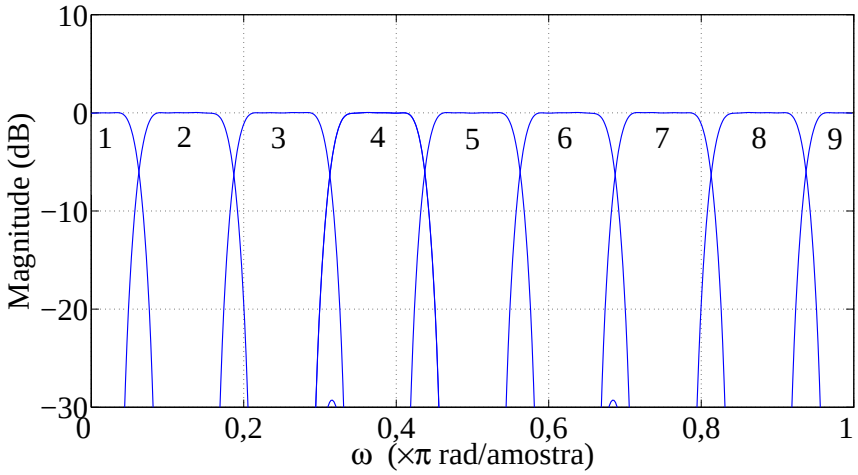

Figura 2.4: Canais do equalizador linear.

2.2 Equalizador Linear - Banco de Filtros

Nesta seção, o chamado 'equalizador linear' utilizado na estrutura multitaxa e descrito em [1] é brevemente revisto. Seu projeto foi inteiramente preservado, a menos da especificação do filtro-protótipo, como se verá adiante.

O banco de filtros proposto divide o espectro de frequências linearmente em 9 canais de igual largura, à exceção do primeiro e do último, que têm metade da largura dos demais. O projeto de cada um dos canais se dá por um arranjo em árvore de filtros que partem sempre de um mesmo filtro-protótipo. Além disso, os canais resultantes possuem fase linear. A magnitude da resposta em frequência dos 9 canais deste equalizador pode ser vista na Figura 2.4.

É utilizada a estrutura em árvore da Figura 2.5, em que os ganhos de saída de A_1 a A_9 são controláveis pelo usuário.

Em seu projeto, que se baseia na técnica de FRM [4] (do inglês *Frequency Response Masking*), cada filtro-protótipo $F_{\mathrm{B}}(z)$ é interpolado de modo que ele e o seu filtro complementar gerem a divisão desejada do espectro. As réplicas indesejadas na resposta do filtro que gera determinada banda, decorrentes da interpolação, são

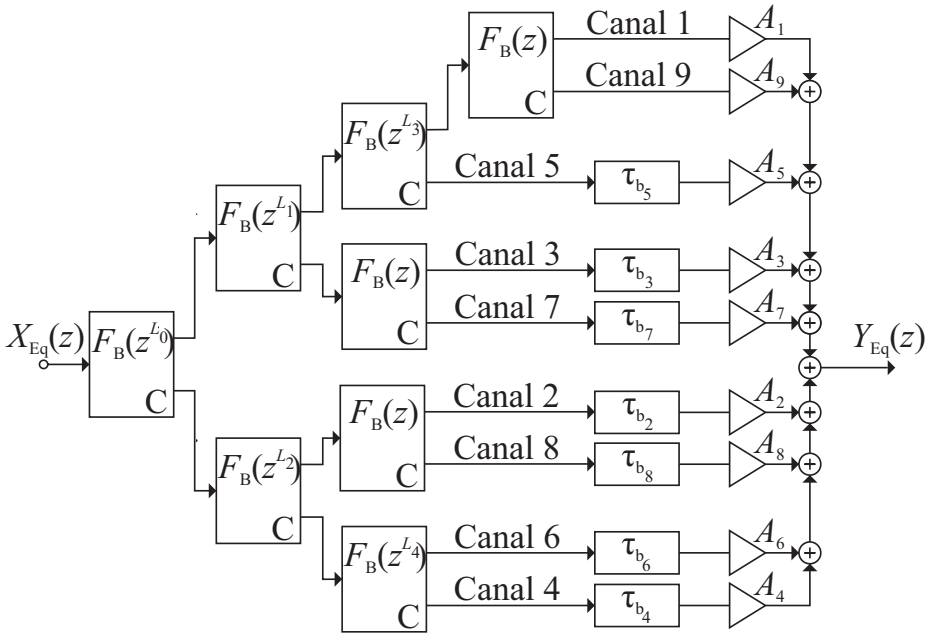

Figura 2.5: Diagrama de blocos do equalizador linear. "C" indica a saída do filtro complementar. Adaptado de [1].

estruturalmente eliminadas nos níveis subsequentes da árvore. A Figura 2.6, como exemplo, ilustra o processo de geração do primeiro canal do equalizador linear.

Deve-se observar que o equalizador linear possui ganho unitário se os ganhos na saída de todos os canais forem escolhidos iguais a 1.

Para reduzir a complexidade computacional, o equalizador utiliza filtros de meia-banda simétricos com ordem par. Apenas metade dos coeficientes desses filtros são não-nulos, o que permite reduzir o número de multiplicações e adições necessárias a um quarto da ordem do filtro. Além disso, o uso de filtros complementares, relacionados pela expressão

$$F_{\mathrm{B}}(z) + \overline{F_{\mathrm{B}}}(z) = 1,$$

evita operações redundantes. A saída $\overline{y}[n]$ do filtro complementar $\overline{F_{\mathrm{B}}}(z)$ para uma entrada $x[n]$ pode ser obtida através de:

$$\overline{y}[n] = x[n] - y[n],$$

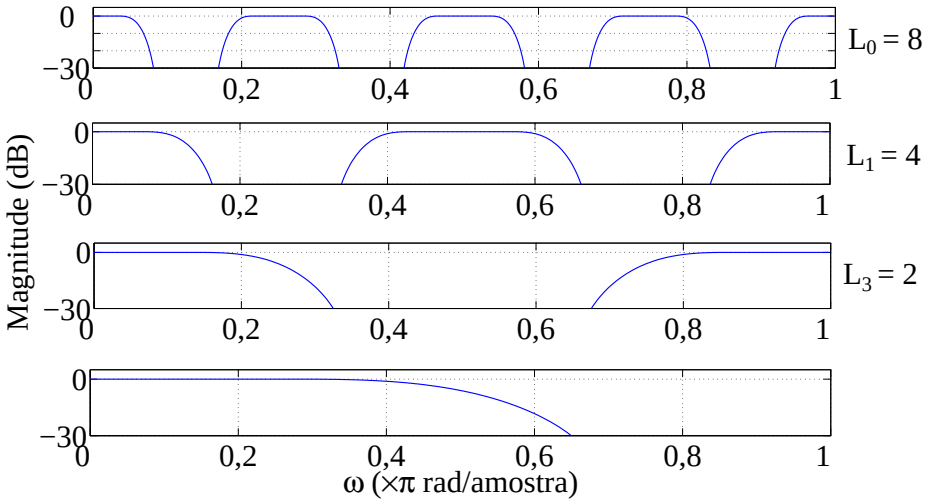

Figura 2.6: Construção do Canal 1 do equalizador linear a partir das versões modificadas do filtro-protótipo. Os gráficos representam, de cima para baixo, as respostas de magnitude na frequência dos filtros $F_{\mathrm{B}}(z^{L_0})$, $F_{\mathrm{B}}(z^{L_1})$, $F_{\mathrm{B}}(z^{L_3})$ e $F_{\mathrm{B}}(z)$. O filtro resultante é o Canal 1 da Figura 2.4.

onde

$$y[n] = (f_{\mathrm{b}} * x)[n]$$

é a própria a saída do filtro $F_{\mathrm{B}}(z)$.

Em lugar de realizar o projeto do filtro-protótipo de meia banda por janelamento [4] como em [1], o presente trabalho optou por um projeto *equiripple* de ordem 14 com especificações mais exigentes. A resposta na frequência da magnitude do filtro utilizado pode ser vista na Figura 2.7. Maior detalhamento do projeto deste filtro pode ser encontrado na Seção A.2.

As diversas versões de $F_{\mathrm{B}}(z)$ utilizadas nos diversos ramos da árvore são interpoladas pelos fatores [1] mostrados na Tabela 2.1.

L_0	L_1	L_2	L_3	L_4
8	4	2	2	3

Tabela 2.1: Fatores de interpolação do equalizador linear.

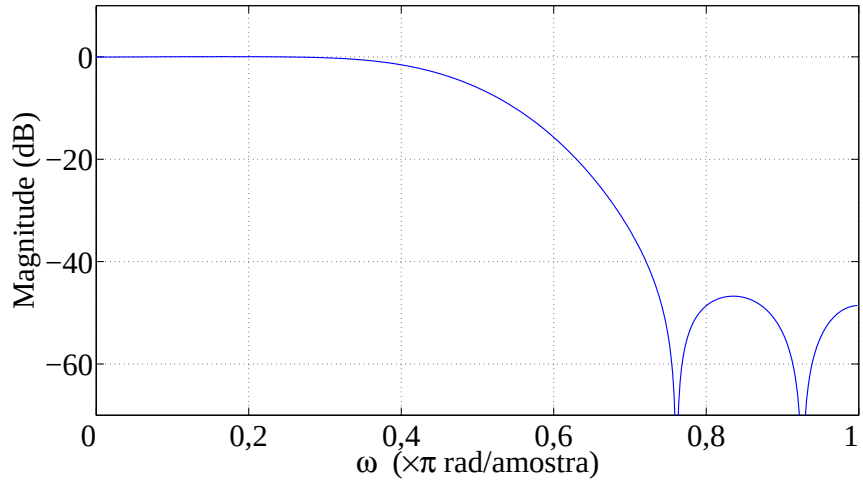

Figura 2.7: Magnitude da resposta de frequência do filtro-protótipo, $F_{\mathrm{B}}(z)$.

É necessário inserir atrasos adequados (τ_{b_2} a τ_{b_8}) nos canais, uma vez que o sinal pode percorrer números desiguais de filtros, e estes podem ter diferentes comprimentos devido às diversas taxas de interpolação. Para a estrutura adotada no projeto, o atraso provocado por um filtro é dado por $L_i\tau_{\mathrm{b}}$, onde L_i é o fator de interpolação do i-ésimo filtro e

$$\tau_{\mathrm{b}} = \frac{N_{\mathrm{b}} - 1}{2} \tag{2.8}$$

é o atraso do filtro-protótipo, suposto de comprimento N_{b}. Neste projeto, $\tau_{\mathrm{b}} = 7$ amostras, o que implica um atraso total para o equalizador de $\tau_{\mathrm{eq}} = 105$ amostras. A Tabela 2.2 exibe os valores obtidos para os atrasos de cada canal em amostras.

τ_{b_2}	τ_{b_3}	τ_{b_4}	τ_{b_5}	τ_{b_6}	τ_{b_7}	τ_{b_8}
28	14	14	7	14	14	28

Tabela 2.2: Valor do atraso de cada canal em amostras.

O número de operações aritméticas (adições e multiplicações reais) realizadas pelo equalizador linear é função do número de operações realizadas pelo filtro-protótipo. O número de adições realizadas pelo filtro-protótipo é

$$\nu_{\mathrm{A,b}} = \frac{N_{\mathrm{b}} + 1}{2}, \tag{2.9}$$

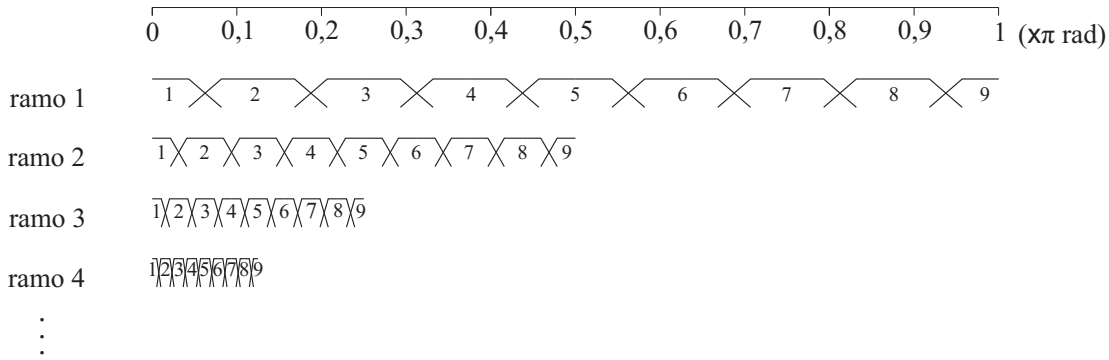

Figura 2.8: Divisão do espectro entre os diferentes canais dos diferentes ramos da estrutura multitaxa.

e o número de multiplicações é

$$\nu_{\mathrm{M,b}} = \frac{N_{\mathrm{b}} + 1}{4}. \tag{2.10}$$

O total de adições por amostra do sinal de entrada no equalizador linear é, então, $8\nu_{\mathrm{A,b}} + 8 + 8$, onde 8 adições são necessárias para se obter a saída dos filtros complementares e 8 adições são necessárias para obtenção do sinal de saída do equalizador. E o número de multiplicações é $8\nu_{\mathrm{M,b}} + 9$, onde 9 multiplicações são relativas aos ganhos em cada um dos canais. Como $N_{\mathrm{b}} = 15$ para este projeto, o equalizador linear realiza 80 adições e 41 multiplicações por amostra do sinal de entrada.

2.3 Estrutura Completa

Combinando-se os resultados obtidos nas duas seções anteriores, pode-se determinar que:

- os atrasos da estrutura da Figura 2.1 são aqueles listados na Tabela 2.3;
- o atraso total do equalizador é de 7700 amostras;
- os números totais de adições e multiplicações por amostra do sinal de entrada são 710,72 e 360,28, respectivamente.

τ_1	τ_2	τ_3	τ_4	τ_5
7595	3675	1715	735	245

Tabela 2.3: Atrasos da estrutra multitaxa em amostras.

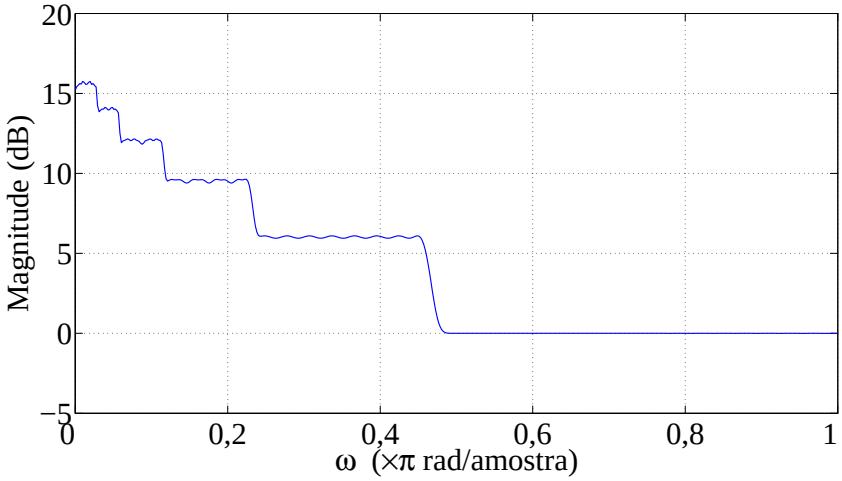

Figura 2.9: Magnitude da resposta em frequência da estrutura em multitaxa utilizando ganho unitário em todos os canais dos equalizadores lineares.

A estrutura multitaxa descrita anteriormente proporciona uma divisão do eixo das frequências conforme exibe a Figura 2.8. Como se pode ver, as faixas de operação dos equalizadores lineares de ramos diferentes da estrutura se sobrepõem parcialmente. Com isso, o ganho de uma determinada faixa pode ser influenciado por mais de um canal, cada um proveniente de um equalizador linear distinto.

A resposta na frequência da magnitude do equalizador completo quando se escolhe ganho unitário na saída de todos os canais pode ser vista na Figura 2.9. As respostas dos diferentes canais são somadas, em particular nas faixas de frequência onde ocorre sobreposição. Além de não resultar em ganho unitário, essa sobreposição dificulta muito a escolha dos ganhos, por aumentar excessivamente o número de parâmetros de controle oferecidos ao usuário. A modelagem de um *notch* (uma rejeição sintonizada de uma faixa) em baixas frequências nessas condições pode se tornar um exercício extenuante.

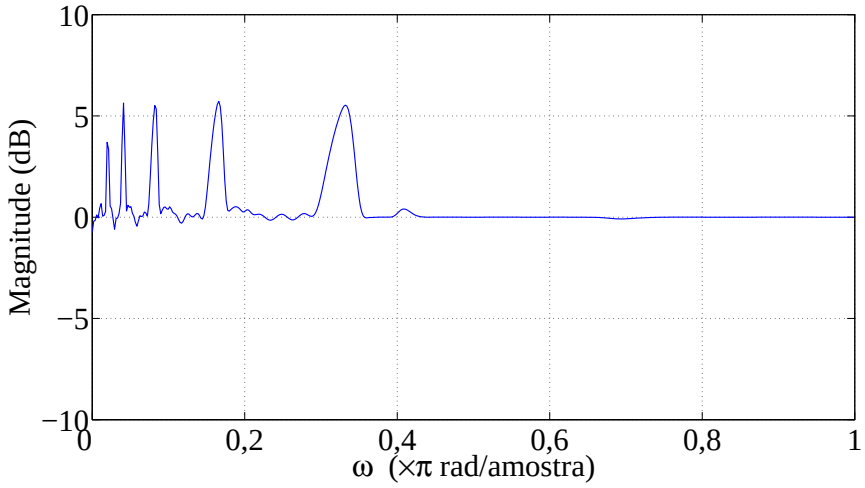

Figura 2.10: Magnitude da resposta em frequência da estrutura em multitaxa utilizando ganho unitário nos canais de 1 a 6 no ramo mais inferior; 4, 5 e 6 nos quatro ramos seguintes; e de 4 a 9 no primeiro ramo.

Para contornar este problema, em todos os casos em que ocorre sobreposição o efeito das bandas superiores foi eliminado pela atribuição de ganho nulo a elas. Assim, são utilizados apenas os canais: de 1 a 6 no ramo mais inferior; 4, 5 e 6 nos quatro ramos seguintes; e de 4 a 9 no primeiro ramo. Como efeito colateral, perde-se a resposta plana na situação de ganho unitário, uma vez que a complementaridade entre filtros adjacentes de ramos diferentes não é garantida pelo projeto original. A Figura 2.10 mostra a resposta do equalizador com as configurações citadas. Este problema da não-complementaridade é um problema estrutural que será abordado com maior detalhe no capítulo seguinte.

Capítulo 3

Alterações no Projeto

Na sua concepção original, a arquitetura do equalizador possui dois níveis: no primeiro, o espectro é dividido sucessivamente em duas metades (geometricamente) por uma estrutura multitaxa; no segundo, divide-se cada uma dessas faixas em sub-canais de mesma largura (linearmente).

Como uma tentativa de se melhorar o desempenho do sistema, o banco de filtros que forma o segundo nível do equalizador (Seção 2.2) foi alterado. A nova estrutura de filtros foi baseada na técnica chamada FFB (do inglês *Fast Filter Bank*), como descrito em [8]. O objetivo desta mudança foi prover boa seletividade nas bandas lineares do equalizador, com menor esforço computacional.

Outra alteração, descrita na Seção 3.2, foi feita no projeto dos filtros *antialiasing* e antiimagem de forma a se tentar minimizar a sobreposição de canais adjacentes na estrutura completa, a fim de se obter uma resposta em frequência do sistema mais plana. Este capítulo irá abordar essas alterações feitas no projeto.

A Seção 3.1 irá tratar sobre as alterações feitas sobre o equalizador linear que compõe a estrutura em multitaxa. A Seção 3.2 tratará das alterações feitas sobre os filtros *antialiasing* e antiimagem a fim de se corrigir um problema estrutural do projeto.

3.1 Alterações sobre o Equalizador Linear

Esta seção irá abordar as alterações feitas sobre o bloco do equalizador linear (bloco `Eq` da Figura 2.1), que visaram a melhorar o desempenho do sistema pelo aumento da seletividade dos canais e pela redução na complexidade computacional. A Seção 3.1.1 irá abordar a descrição teórica da nova estrutura proposta em [8] e a Seção 3.1.2 irá tratar de sua adaptação e utilização no projeto.

3.1.1 FFB - *Fast Filter Bank*

Esta seção descreve o FFB - *Fast Filter Bank*, que é a estrutura adotada em [8] para implementação do equalizador linear.

Conforme descrito em [8], a ferramenta mais popular de análise espectral para sinais discretos no tempo é a *Discrete Fourier Transform* (DFT) [3], que pode ser definida como

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi kn}{N}},$$

onde $x[n]$ é o sinal no tempo e $X[k]$ é sua representação no domínio da frequência, na forma de um par (módulo, fase) associado à componente $e^{-j\frac{2\pi kn}{N}}$. As chamadas *Fast Fourier Transform* (FFT) são implementações rápidas da DFT, das quais as mais usuais são de raiz 2.

A FFT pode ser representada na forma de um banco de filtros e, conforme pode ser visto na Figura 3.1, nessa estrutura cada amostra da entrada origina N amostras na saída, uma para cada canal na saída. O j-ésimo filtro de cada nível i da árvore é obtido pela modificação do filtro-*kernel*

$$H(z) = 1 + z^{-1} \tag{3.1}$$

de acordo com a expressão

$$H_{ij}(z) = H(W_N^{\tilde{j}} z^{2^{L-i-1}}), \tag{3.2}$$

onde $L = \sqrt{N}$, $W_N = e^{-j\frac{2\pi}{N}}$ e $\tilde{j}$ é j com os *bits* na ordem reversa [4]. Com isso, o filtro-*kernel* é deslocado na frequência e estreitado por interpolação dos seus coeficientes, de acordo com sua posição na árvore. As réplicas indesejadas na resposta

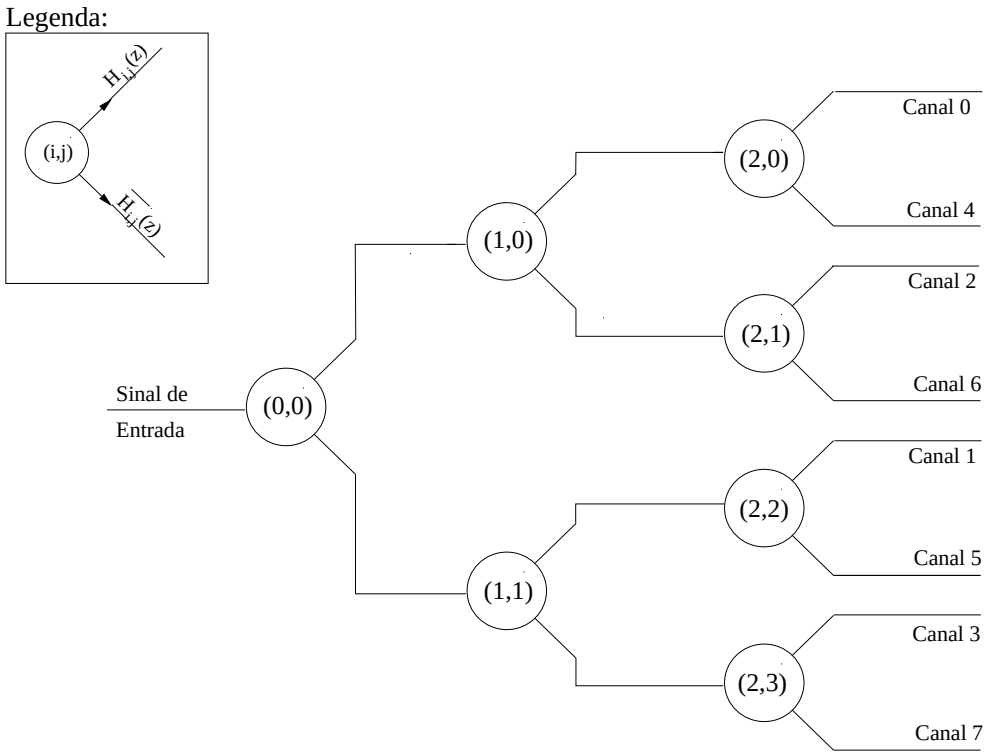

Figura 3.1: Diagrama de blocos da FFT, comum ao equalizador linear composto pela estrutura FFB [8].

de um dado filtro, decorrentes da interpolação, são estruturalmente eliminadas nos níveis subsequentes da árvore.

A Figura 3.2 mostra a resposta na frequência de um canal do banco de filtros correspondente à FFT. Como se pode observar, a FFT apresenta uma baixa atenuação na faixa de rejeição, da ordem de 13 dB. Para melhorar essa característica, o FFB substitui o filtro-*kernel* da FFT por filtros de ordem mais alta e mais seletivos. Além disso, o FFB admite a utilização de filtros-*kernel* diferentes para cada nível, sendo denominado $H_i(z)$ o filtro-*kernel* associado ao i-ésimo nível.

Para reduzir sua complexidade computacional, o FFB utiliza filtros de meia-banda simétricos de ordem par em que apenas metade dos coeficientes são não-nulos, o que permite reduzir o número de multiplicações necessárias a um quarto da ordem do filtro. Além disso, o uso de filtros complementares$H_{ij}(z)$ e $\overline{H_{ij}}(z)$, isto é, com $H_{ij}(z) + \overline{H_{ij}}(z) = 1$, evita o uso de operações redundantes, da mesma forma que o

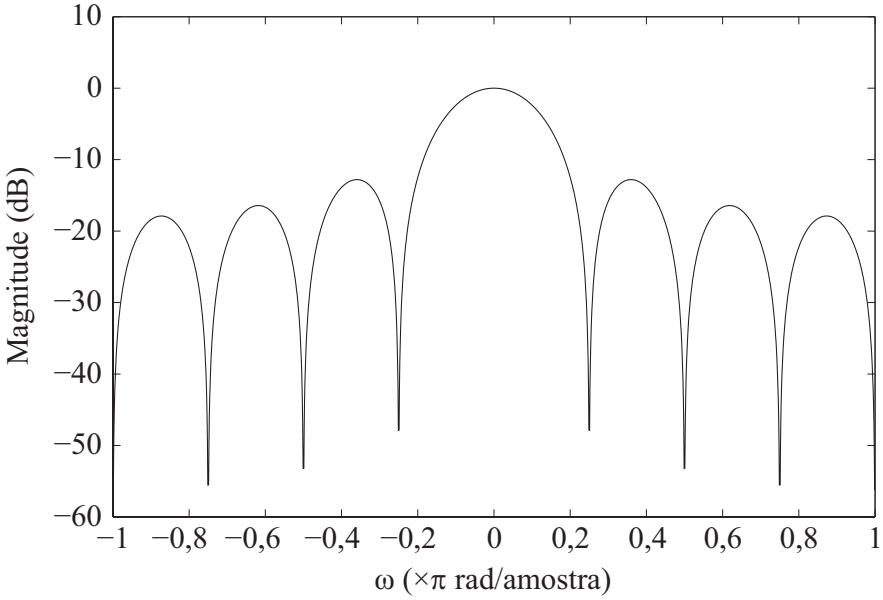

Figura 3.2: Resposta em frequência de um filtro da FFT [8].

banco de filtros da Seção 2.2.

O projeto dos filtros pode ser feito através do método FRM [9], similarmente ao equalizador linear descrito na Seção 2.2. Em [10] pode ser encontrada uma discussão detalhada sobre o projeto dos filtros e a complexidade do FFB.

Para exemplificar, a Figura 3.3 mostra a composição do canal 0 de um FFB de 8 canais, representado pela árvore da Figura 3.1. Os filtros utilizados no FFB são mais exigentes nos primeiros níveis (filtros mais estreitos) e mais relaxados (filtros mais largos) nos últimos níveis da árvore, como pode ser visto na figura.

3.1.2 Utilização do FFB

O objetivo de se usar o FFB neste trabalho foi tornar a estrutura do equalizador mais seletiva mantendo a mesma ordem de complexidade do sistema. Para preservar a semelhança com as especificações do projeto original [1], foi utilizado um FFB de 16 canais complexos (8 canais reais).

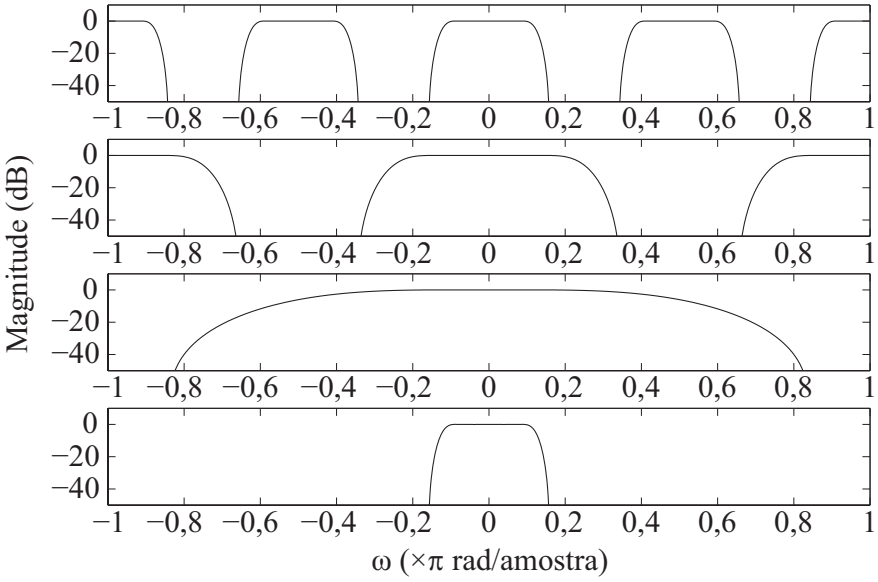

Figura 3.3: Composição do canal 0 de um FFB de 8 canais [8].

A Figura 3.4 mostra a resposta na frequência de cada canal do novo equalizador linear formado pelo FFB, que divide o espectro de frequências em 9 bandas (7 bandas completas e 2 meias-bandas).

A reconstrução do sinal após passar pela estrutura multitaxa requer o cálculo dos atrasos associados aos diversos ramos da estrutura, da mesma maneira que no projeto descrito na Seção 2.2. Desta forma, é necessário calcular o atraso provocado pelo FFB. Chamando de N_i o número de coeficientes de cada filtro utilizado em cada nível i da árvore, o fator de interpolação para cada nível é definido por

$$L_i = 2^{(T-i-1)}, \tag{3.3}$$

onde 2^T é o número total de canais do FFB.

Após interpolação pelo fator L_i, a expressão

$$N_i^{\text{coefs}} = (L_i - 1)(N_i - 1) + N_i \tag{3.4}$$

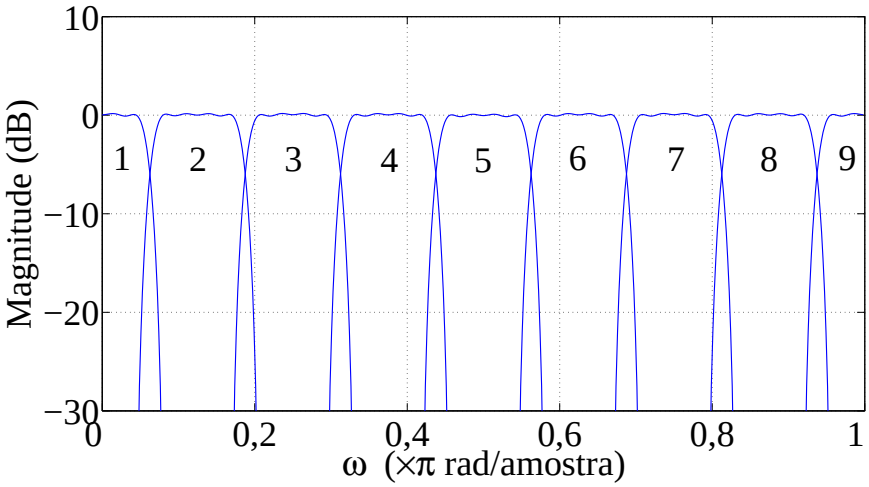

Figura 3.4: Divisão do espectro de frequências para o novo equalizador linear formado pelo FFB, onde cada canal é indicado pelo seu respectivo número.

determina o comprimento (N_i^{coefs}) da memória do filtro de cada nível. O atraso do i-ésimo nível, então, pode ser obtido através de

$$\tau_i = \frac{N_i^{\text{coefs}} - 1}{2}. \tag{3.5}$$

O atraso total da árvore é a soma dos atrasos de cada nível.

A estrutura do FFB possui o mesmo atraso de grupo para todos os canais resultantes do equalizador linear. Com isso, da mesma forma que ocorre com o atraso global do banco de filtros da Seção 2.2, o único efeito significativo sobre a fase do sinal do FFB é um atraso de grupo constante.

A ordem de cada filtro do FFB determina, naturalmente, a complexidade global do sistema. Os filtros utilizados neste trabalho têm, aproximadamente, 40 dB de atenuação na faixa de rejeição, resultando em filtros-*kernel* com 8, 6, 6, e 4 coeficientes, em ordem crescente de i. Os coeficientes dos filtros em cada nível são apresentados na Seção A.3. Como o FFB (ver Figura 3.1) leva em conta a estrutura em árvore da FFT vista como um banco de filtros, tendo apenas como diferença a ordem dos filtros protótipos de cada nível da árvore i, seu custo computacional é

comparável ao da FFT[1].

A Tabela 3.1 apresenta o número de coeficientes não-nulos por nível do FFB utilizada. Para cada filtro $H_{ij}(z)$ a estrutura requer um complementar; entretanto, este não impacta a complexidade geral do sistema, já que sua saída pode ser calculada diretamente a partir da saída de $H_{ij}(z)$.

A Tabela 3.2 apresenta o cálculo da complexidade computacional do FFB para determinados números de canais.

Nível da árvore (i)	**Coeficientes por filtro**	**Quantidade de filtros**	**Coeficientes por nível**	**Total de coeficientes**
1	4	1	4	4
2	3	2	6	10
3	3	4	12	22
4	2	8	16	38

Tabela 3.1: Quantidade de coeficientes não-nulos e distintos por nível na estrutura da FFB. Adaptado de [2].

Total de canais da FFB	**Total de coeficientes**	**Multiplicações complexas por canal por amostra**
2	4	4 / 2 = 2
4	10	10 / 4 = 2,5
8	22	22 / 8 = 2,75
16	38	38 / 16 = 2,375

Tabela 3.2: Complexidade computacional da FFB: número de multiplicações complexas por amostra por canal.

A título de comparação, uma multiplicação complexa equivale a 5 operações reais (3 multiplicações e 2 adições). Logo, como no projeto considera-se que o FFB tem

[1]O custo computacional da FFT é de 1 multiplicação complexa por canal por amostra [11].

canais reais, somente metade das operações são realizadas, o número de adições relacionadas às multiplicações complexas realizadas pelo novo equalizador linear é de

$$\nu_{\mathrm{A,complex}} = \frac{38 \times 2}{2} = 38. \tag{3.6}$$

O número de somas realizadas pela estrutura é relacionado somente com o número de coeficientes não-nulos. Logo, para a estrutura do equalizador com o FFB, são realizadas $\nu_{\mathrm{A},f} = 46$ somas. Com isso, o número total de adições realizadas passa a ser

$$\nu_{\mathrm{A}} = \nu_{\mathrm{A,complex}} + \nu_{\mathrm{A},f} + 8 + 8 = 100 \tag{3.7}$$

adições por amostra, onde 8 adições são necessárias para se obter as saídas dos filtros complementares e 8 adições são necessárias para a obtenção do sinal de saída do FFB.

O número de multiplicações reais relacionadas às multiplicações complexas realizadas pelo FFB é de

$$\nu_{\mathrm{M,complex}} = \frac{38 \times 3}{2} = 57. \tag{3.8}$$

O número total de multiplicações realizadas pelo FFB é

$$\nu_{\mathrm{M}} = \nu_{\mathrm{M,complex}} + 8 = 65 \tag{3.9}$$

multiplicações por amostra, onde 8 multiplicações são relativas aos ganhos aplicados sobre as saídas de cada canal.

Em comparação com o banco de filtros da Seção 2.2, a estrutura do FFB utilizada no equalizador completo demonstrou um pequeno aumento no número de operações (750,09 adições por amostra do FFB contra 710,72 adições por amostra do equalizador linear da Seção 2.2 e 407,53 multiplicações por amostra do FFB contra 360,28 multiplicações por amostra do equalizador da Seção 2.2) a reboque do aumento na seletividade dos canais. Estruturalmente não houve alteração significativa no projeto. A Figura 3.5 mostra a resposta da estrutura completa utilizando o FFB no equalizador linear com ganhos zerados nos canais com sobreposição. A Figura 3.6 mostra a resposta do equalizador completo com ganho unitário em todos os canais.

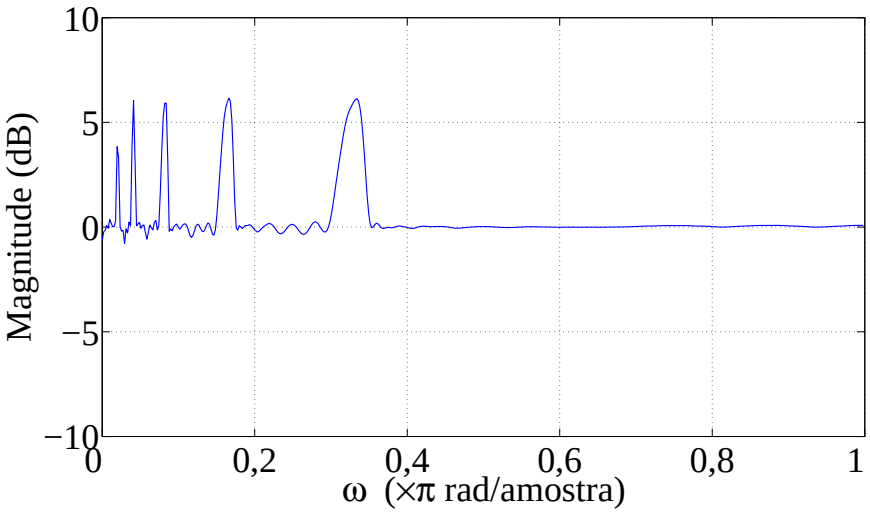

Figura 3.5: Resposta em frequência do equalizador completo utilizando a FFB com os canais que não se sobrepõem.

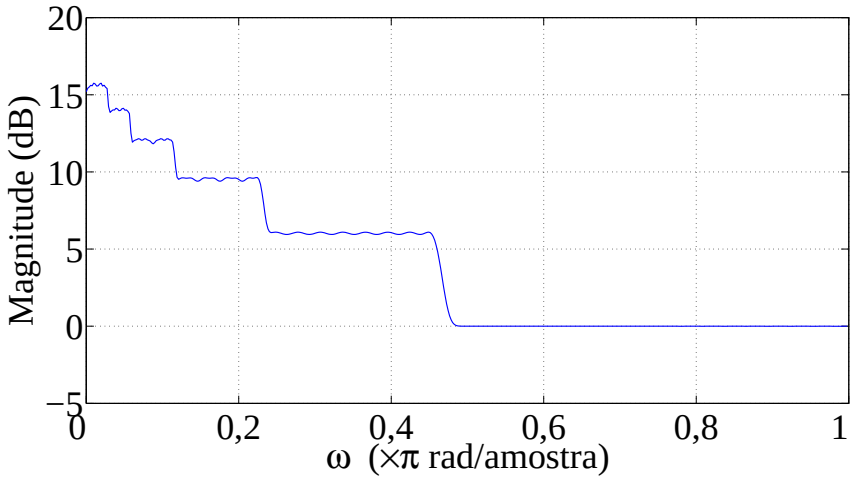

Figura 3.6: Ganho unitário em todos os canais do equalizador completo utilizando a FFB.

3.2 Alterações Sobre os Filtros *Antialiasing* e Antiimagem

Essa seção irá abordar a modificação da estrutura completa a fim de se obter uma resposta em frequência mais plana no equalizador. A Seção 3.2.1 irá descrever o problema da complementaridade dos canais adjacentes de forma detalhada e a Seção 3.2.2 trará a solução adotada para correção do problema.

3.2.1 Problema da Complementaridade de Canais Adjacentes

Conforme descrito na Seção 2.3, há a necessidade de se zerar alguns canais do equalizador para dar ao usuário um controle mais eficaz das regiões de frequência de passagem e/ou rejeição desejadas. Quando isso é feito, algumas imperfeições estruturais surgem na resposta em frequência final do equalizador, que idealmente deveria ser plana. Para melhor exemplificação e detalhamento do problema será feito o estudo de caso para um equalizador com apenas dois ramos. Para cada ramo da estrutura de teste, os ganhos dos equalizadores lineares seguem a configuração descrita na Tabela 3.3.

Ramo / Canal	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	0	0	1	1	1	1	1	1
2	1	1	1	1	1	1	0	0	0

Tabela 3.3: Configuração dos ganhos de um equalizador com apenas 2 ramos.

Como em cada ramo (com exceção do Ramo 1) os últimos canais não são utilizados, os filtros *antialiasing* e antiimagem não influenciam na sobreposição dos canais adjacentes. No Ramo 1, o filtro responsável pelo corte naquela região é um filtro do FFB. Já no Ramo 2, o filtro responsável pelo corte também é do banco de filtros, porém numa taxa de processamento mais baixa. Consequentemente, realizando-se a soma destes canais adjacentes que operam em taxas diferentes, há o surgimento de uma "corcova" com pico de aproximadamente 6 dB. Este problema ocorre em

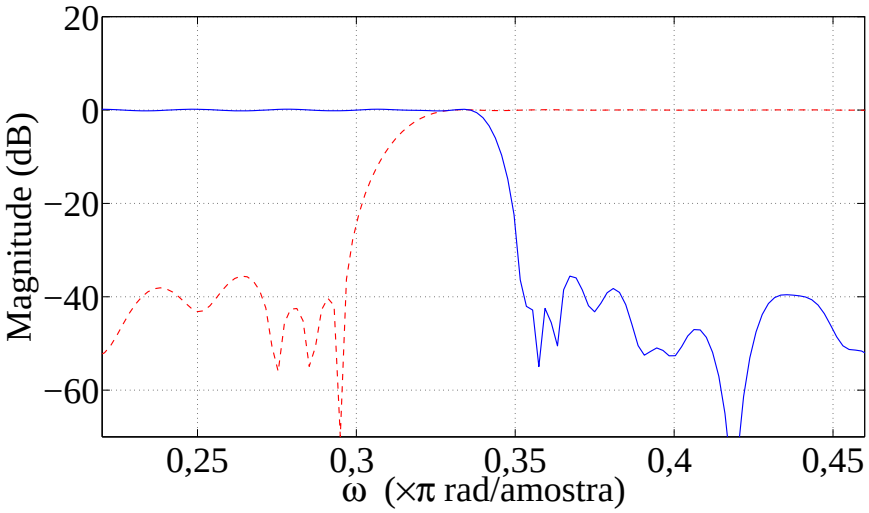

Figura 3.7: Resposta na frequência dos filtros limítrofes de um equalizador com 2 ramos: a curva em linha contínua se refere ao canal de região de baixas frequências, e a curva em linha tracejada se refere ao canal de altas frequências.

todas as instâncias de canais adjacentes operando em taxas diferentes na estrutura completa, ou seja, no equalizador com 6 ramos ocorrem 5 problemas deste tipo (5 "corcovas"). A Figura 3.7 mostra em linha contínua a resposta do Ramo 2 e em linha tracejada a resposta do Ramo 1 para um equalizador composto por 2 ramos. A Figura 3.8 mostra a soma dos Ramos 1 e 2. Percebe-se claramente a região (indicada pela seta) onde a complementaridade entre os filtros deixa de ser atendida por um erro bastante elevado.

3.2.2 Solução Proposta

Diante do problema descrito na subseção anterior, buscou-se uma solução para tentar reduzir as variações indesejadas de ganho nas sobreposições de canais, aproximando-se da melhor maneira uma resposta plana.

A primeira tentativa para se tentar solucionar o problema descrito foi o rearranjo de canais, que consistia ou na eliminação de um canal em determinado ramo seguido da adição de dois canais no ramo imediatamente inferior ou na adição de um canal em determinado ramo seguida da eliminação de dois canais no ramo imediatamente

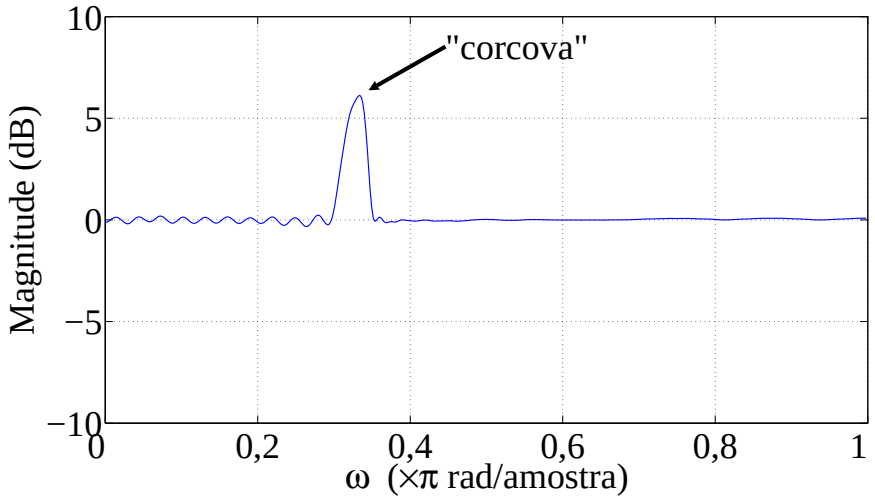

Figura 3.8: Soma da saída dos dois ramos descritos pela Figura 3.7.

inferior. No entanto, o problema persistiu devido à sua natureza estrutural.

Após essas análises, chegou-se a uma solução satisfatória para o problema. Pela inspeção da Figura 3.7, pode-se notar que se as inclinações do final da banda passante do Ramo 2 e do início da banda passante do Ramo 1 forem modificadas, pode-se obter uma resposta mais plana do equalizador. Para que estas exigências fossem cumpridas, adotou-se a solução de ajustar a resposta dos próprios filtros *antialiasing* e antiimagem no eixo da frequência, fazendo com que cada um deles possuísse uma frequência de corte inferior a $0{,}5\pi$ rad/amostra e um decaimento na resposta similar ao de um canal do banco de filtros na taxa imediatamente superior, mantendo-se a mesma ordem (140) dos antigos filtros *antialiasing* e antiimagem. Essa alteração não prejudica as funções originais dos filtros que evitam *aliasing* e eliminam réplicas indesejadas, já que pelo arranjo inicial da estrutura em multitaxa, é feita a eliminação de canais do equalizador linear (conforme foi descrito nas Seções 2.3 e 3.2.1). Com isso, assegura-se que os filtros *antialiasing* e antiimagem exercerão seus papéis e não causarão distúrbios nas frequências abaixo de $0{,}5\pi$ rad/amostra. Para o projeto dos novos filtros, utilizou-se a função `firpm` do Matlab®. A função permite o controle sobre a faixa de passagem, a faixa de rejeição e a importância relativa de ambas (como das duas regiões devem ser priorizadas durante o processo de otimização)

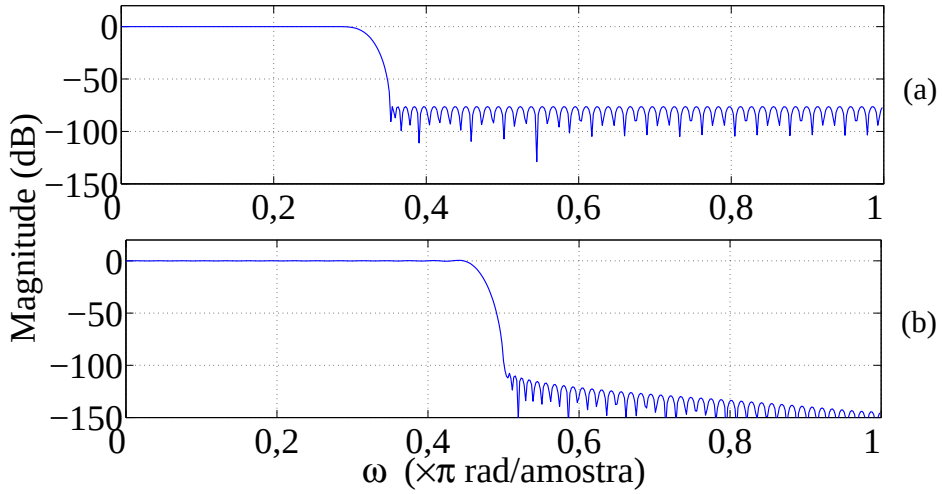

Figura 3.9: Comparação entre as respostas em magnitude dos novos filtros *antialiasing* e antiimagem com os originais – (a) novos filtros, (b) filtros originais.

num projeto *equiripple*. A variação desse parâmetro de "importância" sobre a faixa de passagem e de rejeição é que permite controlar a inclinação do corte do filtro. Os novos filtros foram obtidos através do seguinte procedimento:

1. ajustar as faixas de passagem e rejeição do filtro;
2. visualizar a resposta em frequência;
3. ajustar os parâmetros de prioridade das faixas de passagem e rejeição;
4. visualizar a resposta em frequência do filtro e a resposta em frequência dos ramos somados;
5. repetir até que o *ripple* máximo fique abaixo de 0,5 dB na resposta final.

Chegou-se então a um resultado satisfatório, em que restam somente microondulações na resposta com picos de, no máximo, 0,46 dB. A Figura 3.9 mostra a resposta dos novos filtros *antialiasing* e antiimagem. Maior detalhamento sobre o projeto deste filtro pode ser encontrado na Seção A.4

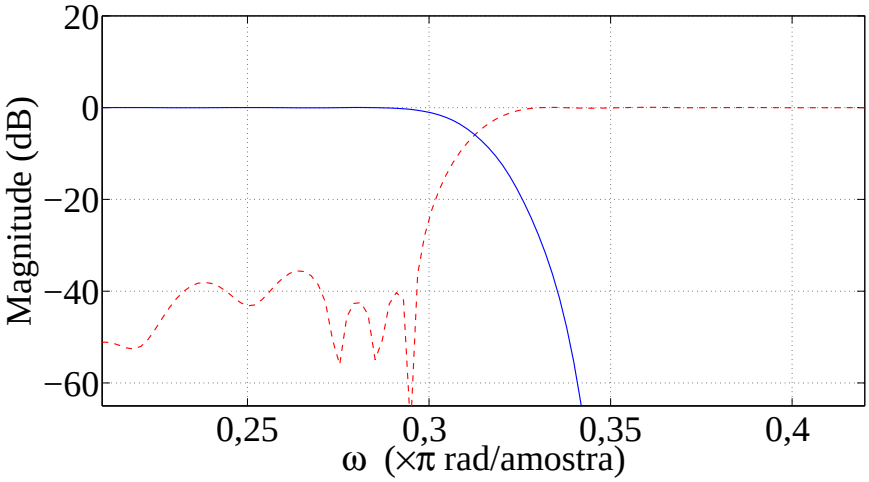

Figura 3.10: Resposta em frequência dos ramos da nova versão de um equalizador com 2 níveis com os novos filtros *antialiasing* e antiimagem.

Para visualização do que acontece nos canais adjacentes com a utilização dos novos filtros *antialiasing* e antiimagem, a Figura 3.10 mostra separadamente a resposta de um equalizador composto por apenas dois ramos.

A estrutura completa do equalizador com os 6 ramos ficou com a resposta em frequência mais próxima de plana do que as primeiras versões. As únicas imperfeições na resposta do equalizador se devem ao *ripple* introduzido pelos filtros utilizados no sistema junto com as "corcovas" reduzidas. A Figura 3.11 mostra a resposta em frequência desta versão modificada (que deve ser comparada com a da Figura 2.10).

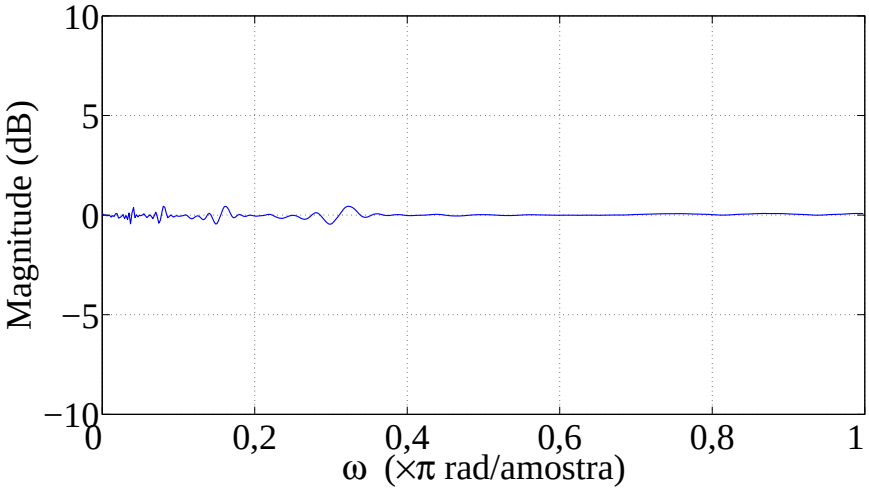

Figura 3.11: Resposta em frequência da nova versão do equalizador.

Capítulo 4

Implementação

Este capítulo irá abordar a implementação de toda a estrutura do FlawQ, com uma descrição das classes criadas para cada bloco da Figura 2.1 e do *Plugin* VST.

Primeiramente, a versão protótipo do sistema foi projetada e implementada em Matlab®. Numa segunda fase, o equalizador foi transportado para a linguagem de programação C++ [12], tendo como base o protótipo da primeira fase. Por fim, a versão do equalizador em C++ foi utilizada para gerar um *Plugin* VST [13] com interface gráfica; esta fornece ao usuário um conjunto de controles de ganho por faixa de frequência e apresenta visualmente a resposta de frequência do filtro resultante. O sistema final permite equalizar um sinal de áudio em tempo real[1]. A implementação do equalizador multitaxa apresentado nos Capítulos 2 e 3 foi feita criando-se um conjunto de classes que realizasse a filtragem sequencial do sinal no tempo, aproveitando ao máximo a eficiência computacional da estrutura. Essas classes foram utilizadas para a criação do *Plugin* VST. A Figura 4.1 mostra um diagrama de relação simplificado das classes utilizadas na implementação do *Plugin*.

Este capítulo irá abordar a implementação de toda a estrutura descrita nos Capítulos 2 e 3 com uma descrição das classes criadas e do *Plugin* VST.

A implementação do sistema foi feita seguindo a mesma estrutura do diagrama de blocos da Figura 2.1. Inicialmente, foram implementadas a classe que realiza

[1]A expressão tempo real é empregada aqui no sentido de que o tempo necessário para se processar uma amostra do sinal de entrada é menor que o seu período de amostragem.

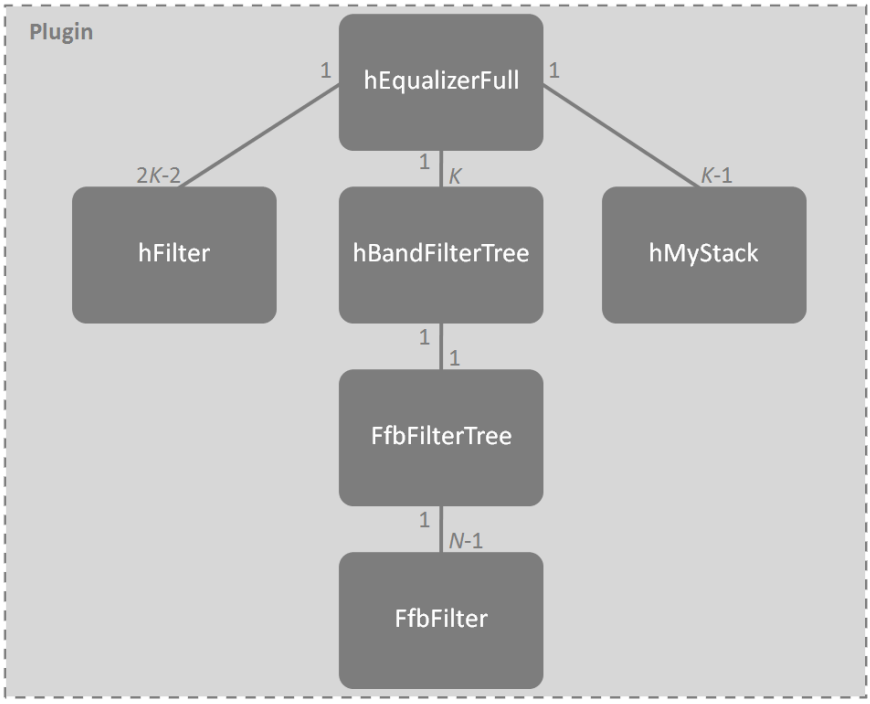

Figura 4.1: Diagrama de relação simplificado das classes que compõem o *Plugin*. K representa o número de ramos e N representa o número de canais complexos do FFB. Os valores indicados representam a relação entre o número de elementos de uma classe e o número de elementos da classe relacionada.

as filtragens *antialiasing* e antiimagem (classe `hFilter`), e a classe que implementa os filtros que compõem o equalizador linear da Seção 2.2 (classe `hBandFilter`); em seguida, vieram a classe que compõe o bloco dos atrasos (classe `hMyStack`) e a classe que compõe a estrutura em árvore do equalizador linear (classe `hBandFilterTree`), tanto na versão original quanto na versão formada pelo FFB (classe `ffb` descrita em [8]); finalmente, passou-se à classe (classe `hEqualizerFull`), que forma o equalizador em multitaxa, e à adaptação das classes para a construção do *Plugin* VST. A implementação de cada classe será descrita nas seções seguintes. No Apêndice B pode ser encontrada uma descrição mais detalhada dos atributos e dos métodos de cada classe.

4.1 Classe `hBandFilter` - Filtragem

A primeira classe criada implementa os filtros de meia-banda interpolados utilizados no equalizador linear da Seção 2.2. Esses filtros possuem uma estrutura muito particular, com a maior parte dos coeficientes iguais a zero. O projeto aqui realizado é uma adaptação da classe criada em [8].

Os filtros foram implementados na forma direta, multiplicando-se a saída da memória pelo seu respectivo coeficiente e somando-se os resultados, apenas para os coeficientes não-nulos e não-unitários. Para tal, foi necessária uma estrutura de dados que levasse em conta o posicionamento dos zeros, de modo a acessar a memória diretamente (sem precisar percorrer toda a estrutura), além de poder "deslocar a memória" alterando apenas um elemento.

Em [8] foi criada uma lista encadeada circular modificada, esquematizada na Figura 4.2, de modo a atender essas especificações. Cada elemento da lista contém um ponteiro para o seu antecessor, e mais quatro ponteiros para os elementos situados a 2^{L-i} amostras e a 2^{L-i-1} amostras, tanto à sua esquerda quanto à sua direita, onde $L = \sqrt{N}$ e i é o nível da árvore. Essas distâncias correspondem aos elementos não-nulos, sendo que para os coeficientes $h_B(1)$ e $h_B(-1)$ a distância é a metade, onde $h_B(\cdot)$ representa os coeficientes do filtro-protótipo, organizados de forma não-causal. Um ponteiro sempre é mantido no elemento da memória correspondente

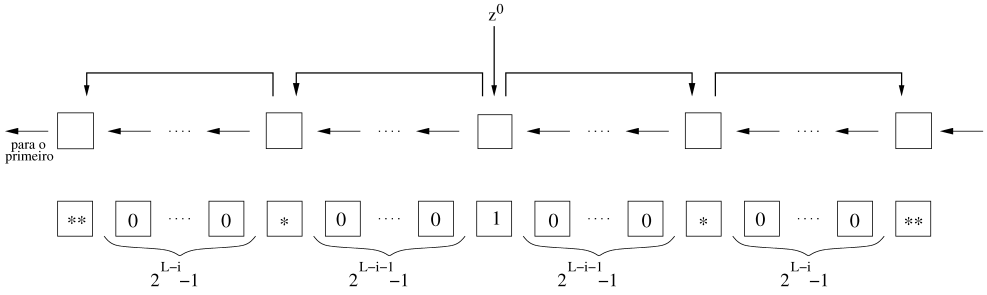

Figura 4.2: Diagrama da organização da memória de um sub-filtro do nível i, mostrando sua correspondência com os coeficientes do filtro (abaixo). As casas marcadas com asterisco indicam os coeficientes não-nulos. As setas indicam os ponteiros. Figura retirada de [8].

ao coeficiente em z^0 e outro no elemento correspondente à amostra mais recente. Dessa maneira, a lista pode ser deslocada com apenas uma troca de ponteiros, e os elementos não-nulos podem ser acessados diretamente.

A classe `hBandFilter` utiliza essa lista encadeada para implementar a memória do filtro. Os coeficientes não-nulos e não-unitários são armazenados num vetor estático, membro da classe.

4.2 Classe `hFilter` - Filtragem *antialiasing* e antiimagem

Também foi criada uma classe auxiliar para os filtros *antialiasing* e antiimagem, chamada `hFilter`. Esta classe implementa filtros FIR de ordem par simétricos na forma direta de forma eficiente.

4.3 Classe `hMyStack` - Atrasos

Uma classe auxiliar chamada `hMyStack` foi criada para implementar os atrasos. Esta classe possui uma pilha do tipo `vector` da biblioteca padrão [12] de C++ como membro. Esta pilha possui comprimento igual ao comprimento do atraso e possui

como propriedade o tempo constante de inserção de um elemento no topo da pilha e de leitura de um elemento do final da pilha. A classe possui um método para cada uma dessas duas ações, e é utilizada para implementação dos blocos de atraso utilizados no sistema.

4.4 Classe `hBandFilterTree` - Equalizador linear

A classe `hBandFilterTree` é a classe que compõe o bloco do Equalizador Linear. A Seção 4.4.1 descreve a implementação da classe que representa o equalizador linear formado pelo banco de filtros da Seção 2.2, e a Seção 4.4.2 descreve a adaptação feita na classe para a utilização do FFB.

4.4.1 Banco de Filtros

Esta classe possui um vetor de 8 objetos da classe `hBandFilter` que implementam os 8 filtros de meia-banda da estrutura. O outro membro desta classe é um vetor de 7 objetos da classe `hMyStack` que representam os 7 atrasos da estrutura. Todos os objetos são inicializados no construtor da classe, sendo os coeficientes do filtro-protótipo lidos em tempo de compilação através de um vetor estático num arquivo de cabeçalho. Os atrasos inseridos em cada canal são calculados em função do número de coeficientes do filtro-protótipo durante a execução do construtor da classe. Um método da classe retorna o atraso total do equalizador linear e o método principal da classe recebe como entrada a amostra a ser filtrada e retorna uma amostra filtrada (e devidamente atrasada).

4.4.2 Adaptação para utilização com o FFB

A classe `hBandFilterTree` implementa o equalizador linear com o FFB de cada ramo. Esta classe constrói a árvore que compõe o equalizador linear. Possui um objeto da classe `FfbFilterTree` (melhor descrita na próxima seção) que constrói o banco de filtros do FFB propriamente dito. O método `filter` recebe como entrada a amostra a ser filtrada e retorna uma amostra filtrada pela árvore do FFB. Esta classe também realiza o mapeamento dos canais do FFB (ordenados segundo seu número com os bits revertidos), e o método chamado `getDelay` retorna o atraso

total do equalizador linear, que será utilizado para compor os blocos de atraso da estrutura em multitaxa.

4.5 Classe `ffb` - *Fast Filter Bank*

Esta seção realiza uma breve descrição da classe que implementa o FFB.

O FFB foi implementado criando-se duas classes, a `FfbFilter` e a `FfbFilterTree`. A primeira descreve um único filtro dentro da estrutura em árvore, enquanto que a outra descreve a própria árvore.

Como já explicado na Subseção 3.1.1 os filtros utilizados pelo FFB possuem uma estrutura muito particular que permite reduzir o número de operações. Os filtros foram implementados na forma direta não-causal, multiplicando-se a saída da memória pelo seu respectivo coeficiente e somando os resultados, apenas para os coeficientes não-nulos e não-unitários, exatamente da mesma forma que a estrutura implementada pela classe `hBandFilter` da Seção 4.1. Os dois principais métodos da `FfbFilter` são o `setparam`, no qual são passados a posição do filtro dentro da árvore (i e j) e os seus coeficientes; e o `filter`, que recebe um valor complexo correspondente à entrada e retorna a amostra filtrada por ele e pelo seu complementar.

A classe `FfbFilterTree` possui um vetor contendo $N-1$ objetos do tipo `FfbFilter`, onde os filtros estão ordenados externamente por i e internamente por j, ou seja, o primeiro elemento desse vetor correspondente ao par (i,j) é (0,0), o segundo é (1,0), o terceiro é (1,1), e assim por diante.

O construtor da `FfbFilterTree` lê os coeficientes de cada filtro a partir de um vetor estático. Este vetor contém o valor de metade dos coeficientes não-nulos e não-unitários de cada filtro (devido à simetria dos filtros), suficientes para o cálculo.

O método que realiza a filtragem nessa classe é denominado `filter`; recebe um valor em ponto flutuante como entrada e retorna um vetor complexo contendo as saídas de todos os canais. A saída de cada filtro é armazenada no próprio vetor de saída (*inplace*), da mesma maneira que na FFT [4].

A função responsável pelo processamento do sinal recebe um bloco de amostras e retorna um bloco de mesmo comprimento. Para cada amostra do bloco, ela utiliza o método `filter` da `FfbFilterTree` para obter a saída de todos os canais. Cada saída é multiplicada pelo ganho correspondente ao seu canal e esses produtos são somados, gerando a amostra de saída atual.

Em [14] é mostrada uma simplificação adicional da estrutura do banco de filtros para o caso de sinais de entrada reais, utilizando sua simetria no domínio da frequência. Com isso, apenas metade dos filtros é utilizada, reduzindo-se o número necessário de operações. A ordenação dos canais na saída do filtro, originalmente em *bit-reversal*, é perdida e deve ser ajustada seguindo o algoritmo descrito em [8], conforme foi citado na Subseção 4.4.2.

4.6 Classe `hEqualizerFull` - Equalizador em multitaxa

A estrutura multitaxa foi implementada na classe `hEqualizerFull`. Esta classe possui um vetor contendo objetos da classe `hBandFilterTree` representando os equalizadores lineares dos ramos, um vetor contendo os atrasos (através da classe `hMyStack`) em cada ramo e dois vetores contendo os filtros antiimagem e *antialiasing* em cada ramo. Todos os objetos são inicializados no construtor da classe, e os coeficientes dos filtros *antialiasing* e antiimagem são lidos de um vetor estático. Os atrasos de cada ramo são calculados em função do número de coeficientes dos filtros *antialiasing* e antiimagem e do atraso do equalizador linear (obtido através do método `getDelay` da classe `hBandFilterTree`). O número de ramos da estrutura é escolhido em tempo de compilação.

O método `filter` da classe `hEqualizerFull` implementa a filtragem, recebendo uma amostra do sinal de entrada e retornando uma amostra filtrada. A implementação dos decimadores e interpoladores foi realizada observando-se a estrutura multitaxa. O aninhamento dos decimadores provoca uma periodicidade na chamada das funções `filter` dos equalizadores de cada ramo. O k-ésimo ramo, por exemplo,

é executado com um período igual a 2^{k-1} chamadas de função. Considerando-se todos os K ramos, a estrutura possui um período total igual a 2^{K-1}. Esse período foi utilizado no método `filter` da classe `hEqualizerFull` para se saber quais ramos estão ativos para uma determinada chamada do método. É criado um vetor de comprimento 2^{K-1}, do qual cada elemento indica quais ramos estão ativos. Por exemplo, para $K = 3$ seria gerado o vetor $[3, 1, 2, 1]$, indicando que na primeira chamada da função os 3 ramos estão ativos, na segunda chamada apenas o primeiro ramo está ativo, e assim por diante. A utilização deste vetor se dá através de uma variável auxiliar que aponta para a posição atual dentro do vetor. Esta variável é incrementada (módulo 2^{K-1}) toda vez que o método `filter` é chamado, fazendo com que o número de ramos ativos seja modificado. Dessa maneira a interpolação e a decimação são feitas de maneira eficiente, sem a necessidade de se armazenar o estado de cada ramo. Além disso, o vetor que indica quais ramos estão ativos é calculado durante o construtor da classe; as únicas operações necessárias durante a execução do método `filter` são uma soma e uma leitura da memória, além das operações realizadas pela filtragem em si.

4.7 Descrição do *Plugin*

A grande maioria dos programas comerciais para áudio digital são vendidos hoje na forma de *plugins*. Para usá-los, o usuário precisa dispor de um programa *host*, normalmente chamado de DAW (*Digital Audio Workstation*). As DAWs oferecem ferramentas básicas de edição de áudio (multitrilhas, visualização da forma de onda, alteração de ganhos etc.), enquanto que os *plugins* se encarregam dos efeitos mais elaborados (reverberação, equalização, sintetizadores etc.). A comunicação entre *plugin* e *host* se dá através de algum protocolo previamente estabelecido, que basicamente regula procedimentos como a troca de amostras de áudio e a passagem dos valores dos parâmetros.

Dos padrões existentes hoje no mercado, o VST se mostra mais atraente por dois motivos:

1. Seu SDK (do inglês *Software Development Kit*, conjunto de rotinas necessárias

para a implementação do *plugin*) pode ser obtido gratuitamente no site do fabricante [13];

2. O padrão pode ser usado nas plataformas Mac OSX e Windows.

Este projeto foi implementado na forma de *plugin* VST, com uma interface gráfica que recebe os ganhos do usuário e apresenta a resposta em frequência resultante do filtro. O processamento é feito *online*, ou seja, a equalização é feita enquanto a DAW toca as amostras de áudio.

Uma das grandes vantagens da programação em VST é a possibilidade de utilização de uma arquitetura em camadas. Desta maneira, foi possível desacoplar a implementação do *plugin* da implementação do equalizador.

A função principal de um *plugin* VST chama-se `processReplacing`: é ela quem recebe as amostras e as devolve ao *host*. No caso deste projeto, ela passa as amostras a um objeto da classe `hEqualizerFull` junto com os ganhos recebidos pela interface gráfica. O objeto, por sua vez, devolve-lhe as amostras processadas, para que possam então ser enviadas de volta ao *host*. Aí reside o atraso inerente ao sistema, provocado pelo filtro: enquanto o *host* fornece a amostra atual, recebe aquela que estava na saída do filtro, atrasada pelo processamento.

Um dos desafios na implementação do *plugin* foi a interface com o usuário. Na configuração original, mantendo as interseções entre diferentes canais do equalizador, seria necessário apresentar o controle para os 54 ganhos da estrutura (9 ganhos para cada um dos 6 ramos), o que por si só já era uma dificuldade. Quando se decidiu evitar a sobreposição entre bandas, o número de *faders* a apresentar se reduziu a 24; todos puderam ser dispostos na mesma tela, de maneira bem intuitiva.

A resposta de magnitude do filtro é apresentada de forma realística para o usuário. A cada mudança no valor dos ganhos, o *plugin* instancia um objeto da classe `hEqualizerFull`, fornecendo um impulso e a configuração de ganhos. Com a resposta ao impulso em mãos, calcula-se a resposta na frequência por uma FFT de

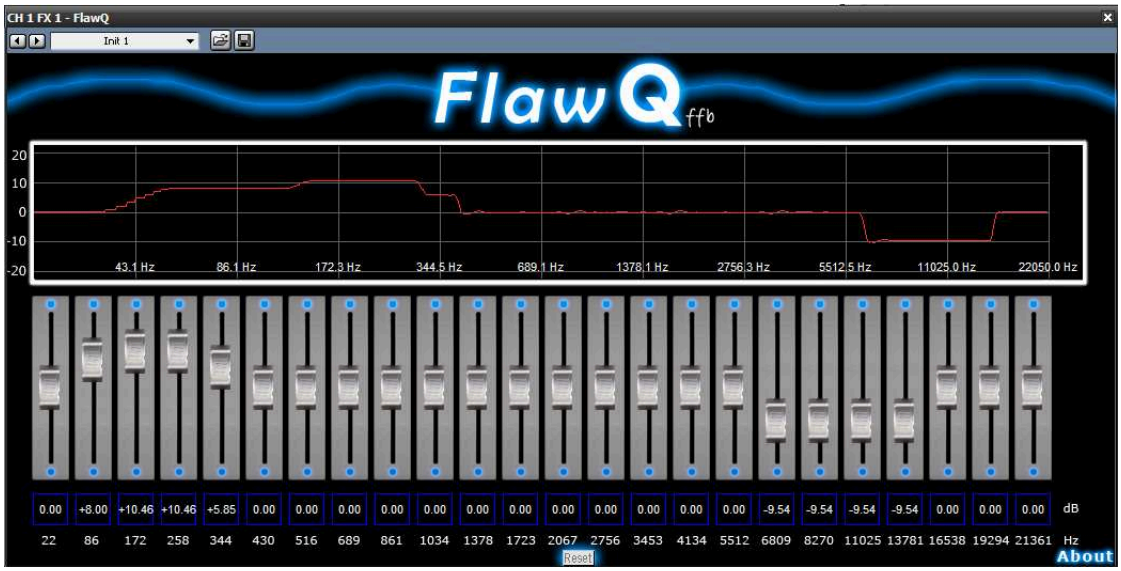

Figura 4.3: Interface gráfica do FlawQ com configuração de reforço nos graves e atenuação nos agudos.

16384 pontos. O cálculo é feito através da biblioteca FFTW [15]. A interface gerada pode ser vista na Figura 4.3.

4.7.1 Características do *Plugin*

O *Plugin* gerado opera sobre sinais mono com taxa de amostragem igual a 44,1 kHz. Nesta taxa de amostragem, o atraso do equalizador é equivalente a, aproximadamente, 17 ms. A distribuição dos canais é mostrada na Tabela 4.1. A faixa dinâmica de operação é de 40 dB, ou seja, o usuário pode aplicar um ganho ou uma atenuação de até 20 dB em cada um dos canais.

O desempenho do plugin atendeu às expectativas, realizando as operações *online*. A sensibilidade dos *faders* não é instantânea, pois a cada mudança de valor nos ganhos é necessário calcular novamente a resposta do filtro. Contudo, a visualização da resposta em frequência real do equalizador foi apontada como um fator positivo em relação aos equalizadores normalmente disponíveis.

A interface gráfica do equalizador é composta por alguns elementos principais. São eles: visualização da resposta em frequência real do equalizador, 24 *faders* para

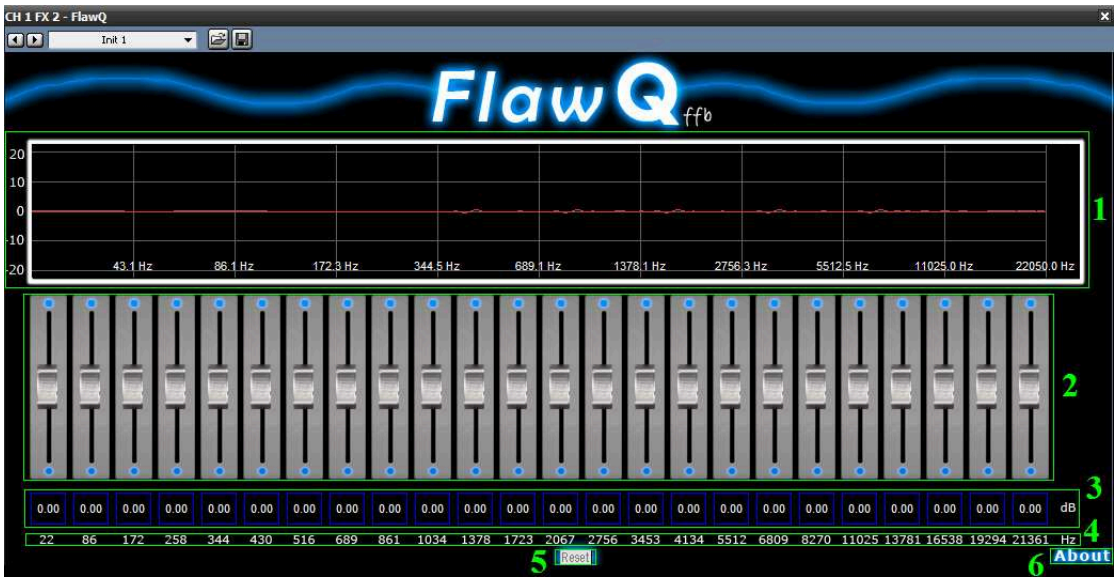

Figura 4.4: Interface gráfica do FlawQ detalhada – (1) resposta real em frequência do equalizador, (2) *faders* para controle de ganhos, (3) valores reais dos ganhos ajustados, (4) frequência central de cada canal, (5) botão *Reset* e (6) botão *About*.

Canal	Centro (Hz)	Canal	Centro (Hz)
1	22	13	2067
2	86	14	2756
3	172	15	3453
4	258	16	4134
5	344	17	5512
6	430	18	6809
7	516	19	8270
8	689	20	11025
9	861	21	13781
10	1034	22	16538
11	1378	23	19294
12	1723	24	21361

Tabela 4.1: Localização dos centros dos canais.

controle de ganho sobre os canais, 24 campos mostrando o valor do ganho, em dB, ajustado no *fader* correspondente e 24 campos que mostram a frequência central de cada canal correspondente a cada *fader*. Há também o botão *Reset*, cuja função é de voltar todos *faders* para a posição de ganho 0 dB, e o botão *About*, que contém informações sobre o *plugin*. A Figura 4.4 mostra a descrição detalhada da interface gráfica do *plugin*.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho apresentou o projeto de um equalizador gráfico digital. Um equalizador multitaxa com estrutura em árvore da literatura foi modificado e então implementado em C++ na forma de *plugin* VST, com uma interface gráfica que permite a visualização da resposta em frequência real do filtro. Eliminaram-se as sobreposições entre canais da arquitetura original pelo bem da usabilidade, ao custo da perda da complementaridade entre filtros adjacentes de oitavas distintas, que foi posteriormente corrigida, possibilitando uma resposta final aproximadamente plana. Foi feita a troca do banco de filtros do equalizador linear para o FFB, o que possibilitou uma melhora da seletividade dos canais do FlawQ com um incremento pequeno na complexidade do sistema. A estrutura dos filtros em árvore e algumas características peculiares da estrutura permitiram uma implementação rápida que garantiu o funcionamento *online* do *plugin*.

A principal meta deste trabalho foi realizar um estudo do emprego de uma arquitetura rápida no projeto de equalizadores digitais com interface amigável e alto desempenho. A sua continuação deve passar por uma modificação na implementação a fim de se eliminar a utilização dos filtros *antialising*, diminuindo ainda mais a complexidade computacional da estrutura, para torná-la mais rápida. A ideia dessa possível eliminação do filtro *antialiasing* consiste em operar diretamente sobre as amostras de determinados canais do equalizador da taxa imediatamente superior ao ramo em questão, possibilitando assim despensar os filtros *antialising*.

Referências Bibliográficas

[1] HERGUM, R., "A Low Complexity, Linear Phase Graphic Equalizer". In: *Presented at the 85th AES Convention*, Los Angeles, USA, November 1988. Preprint 4706.

[2] DINIZ, F. C. C. B., KOTHE, I., NETTO, S. L., *et al.*, "High-Selectivity Filter Banks for Spectral Analysis of Music Signals", *EURASIP Journal on Advances in Signal Processing*, v. 2007, pp. 1–12, March 2007.

[3] HAYKIN, S., VEEN, B. V., *Signals and Systems*. New York, USA, Wiley, 1996.

[4] DINIZ, P. S. R., DA SILVA, E. A. B., NETTO, S. L., *Digital Signal Processing: System Analysis and Design*. New York, USA, Cambridge, 2002.

[5] QUATIERI, T. F., MCAULAY, R. J., "Audio Signal Processing Based on Sinussoidal Analysis/Synthesis". In: Kahrs, M., Brandenburg, K. (eds.), *Applications of Digital Signal Processing to Audio and Acoustics*, New York, USA, Kluwer, 2002.

[6] MARTINS, F. C. V., NUNES, L. O., TYGEL, A. F., *et al.*, "FlawQ: Um Plug-in VST para Equalização Gráfica Digital". In: *Anais do VI Congresso de Engenharia de Áudio*, pp. 72–79, São Paulo, Brasil, Maio 2008.

[7] BOSI, M., GOLDBERG, R. E., *Introduction to Digital Audio Coding and Standards*. Norwell, USA, Kluwer, 2003.

[8] NUNES, L. O., TYGEL, A. F., DE JESUS, R. A., *et al.*, "Equalizador Gráfico Digital de Alta Seletividade em VST". In: *Anais do IV Congresso de Engenharia de Áudio*, pp. 47–52, São Paulo, Brasil, Maio 2006.

[9] LIM, Y. C., "Frequency-Response Masking Approach for the Synthesis of Sharp Linear Phase Digital Filters", *IEEE Transactions on Circuits and Systems*, v. 33, n. 4, pp. 357–364, April 1986.

[10] LIM, Y. C., FARHANG-BOROUJENY, B., "Analysis and Optimum Design of the FFB". In: *Proc. of the ISCAS´94 (IEEE International Symposium on Circuits and Systems)*, v. 2, pp. 509–512, London, UK, June 1994.

[11] FARHANG-BOROUJENY, B., LIM, Y. C., "A Comment on Computational Complexity of Sliding FFT", *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, v. 39, n. 12, pp. 875–876, December 1992.

[12] STROUSTRUP, B., *A Linguagem de Programação C++*. 3 ed. Porto Alegre, Brasil, Bookman, 2000.

[13] "Steinberg Media Technologies (VST)", Acessado em 26/01/2010.

[14] LEE, J. W., LIM, Y. C., "Efficient Implementation of Real Filter Banks Using Frequency Response Masking Techniques". In: *Proc. of the APCCAS'02 (Asia-Pacific Conference on Circuits and Systems)*, v. 1, pp. 69–72, Singapore, 2002.

[15] FRIGO, M., JOHNSON, S. G., "The Design and Implementation of FFTW3", *Proceedings of the IEEE*, v. 93, n. 2, pp. 216–231, February 2005.

Apêndice A

Filtros Utilizados

Este apêndice contém todos os coeficientes não-nulos e não-unitários dos filtros utilizados neste projeto. Como estes filtros são simétricos, somente metade dos coeficientes são apresentados. A Seção A.1 contém os coeficientes dos filtros *antialising* e antiimagem do projeto inicial, a Seção A.2 contém os coeficientes do filtro-protótipo do equalizador linear de [1], a Seção A.3 contém os coeficientes do FFB utilizado e, por fim, a Seção A.4 contém os coeficientes dos novos filtros *antialiasing* e antiimagem que corrigem o problema da complementaridade de canais adjacentes.

A.1 Filtros *Antialiasing* e Antiimagem

Filtros *antialiasing* e antiimagem utilizados na estrutura em multitaxa da primeira fase do projeto. São filtros passa-baixas, simétricos (metade de coeficientes não-nulos representados), idênticos, de ordem 140 e foram projetados por otimização *least-squares*. O seguinte comando do Matlab® foi utilizado para geração dos coeficientes destes filtros: `firls(140,[0 0.4535 0.5 1],[1 1 0 0],[1 1e8])`. A Tabela A.1 mostra os coeficientes desses filtros.

A.2 Filtro-Protótipo – Banco de Filtros de [1]

Filtro utilizado no equalizador linear da Seção 2.2. É um filtro passa-baixas *equi-ripple* de meia banda, simétrico (um quarto de coeficientes não-nulos e não-unitários representados) e de ordem 14 e com frequência de corte em $0{,}4453\pi$ rad/amostra.

$h(0)$	$4{,}712\times10^{-1}$	$h(18)$	$1{,}543\times10^{-2}$	$h(36)$	$8{,}323\times10^{-4}$	$h(54)$	$-1{,}473\times10^{-3}$
$h(1)$	$3{,}168\times10^{-1}$	$h(19)$	$2{,}198\times10^{-3}$	$h(37)$	$-4{,}623\times10^{-3}$	$h(55)$	$-6{,}865\times10^{-4}$
$h(2)$	$2{,}855\times10^{-2}$	$h(20)$	$-1{,}308\times10^{-2}$	$h(38)$	$-1{,}568\times10^{-3}$	$h(56)$	$1{,}150\times10^{-3}$
$h(3)$	$-1{,}018\times10^{-1}$	$h(21)$	$-4{,}164\times10^{-3}$	$h(39)$	$3{,}811\times10^{-3}$	$h(57)$	$8{,}211\times10^{-4}$
$h(4)$	$-2{,}796\times10^{-2}$	$h(22)$	$1{,}076\times10^{-2}$	$h(40)$	$2{,}077\times10^{-3}$	$h(58)$	$-8{,}453\times10^{-4}$
$h(5)$	$5{,}668\times10^{-2}$	$h(23)$	$5{,}519\times10^{-3}$	$h(41)$	$-2{,}993\times10^{-3}$	$h(59)$	$-8{,}881\times10^{-4}$
$h(6)$	$2{,}698\times10^{-2}$	$h(24)$	$-8{,}534\times10^{-3}$	$h(42)$	$-2{,}379\times10^{-3}$	$h(60)$	$5{,}784\times10^{-4}$
$h(7)$	$-3{,}593\times10^{-2}$	$h(25)$	$-6{,}352\times10^{-3}$	$h(43)$	$2{,}205\times10^{-3}$	$h(61)$	$9{,}129\times10^{-4}$
$h(8)$	$-2{,}565\times10^{-2}$	$h(26)$	$6{,}444\times10^{-3}$	$h(44)$	$2{,}502\times10^{-3}$	$h(62)$	$-3{,}473\times10^{-4}$
$h(9)$	$2{,}349\times10^{-2}$	$h(27)$	$6{,}746\times10^{-3}$	$h(45)$	$-1{,}479\times10^{-3}$	$h(63)$	$-9{,}527\times10^{-4}$
$h(10)$	$2{,}401\times10^{-2}$	$h(28)$	$-4{,}535\times10^{-3}$	$h(46)$	$-2{,}473\times10^{-3}$	$h(64)$	$1{,}310\times10^{-4}$
$h(11)$	$-1{,}498\times10^{-2}$	$h(29)$	$-6{,}778\times10^{-3}$	$h(47)$	$8{,}381\times10^{-4}$	$h(65)$	$1{,}076\times10^{-3}$
$h(12)$	$-2{,}211\times10^{-2}$	$h(30)$	$2{,}837\times10^{-3}$	$h(48)$	$2{,}324\times10^{-3}$	$h(66)$	$2{,}738\times10^{-4}$
$h(13)$	$8{,}767\times10^{-3}$	$h(31)$	$6{,}518\times10^{-3}$	$h(49)$	$-2{,}974\times10^{-4}$	$h(67)$	$-1{,}399\times10^{-3}$
$h(14)$	$2{,}001\times10^{-2}$	$h(32)$	$-1{,}369\times10^{-3}$	$h(50)$	$-2{,}088\times10^{-3}$	$h(68)$	$-2{,}023\times10^{-3}$
$h(15)$	$-4{,}069\times10^{-3}$	$h(33)$	$-6{,}032\times10^{-3}$	$h(51)$	$-1{,}367\times10^{-4}$	$h(69)$	$-1{,}341\times10^{-3}$
$h(16)$	$-1{,}776\times10^{-2}$	$h(34)$	$1{,}448\times10^{-4}$	$h(52)$	$1{,}794\times10^{-3}$	$h(70)$	$-3{,}793\times10^{-4}$
$h(17)$	$4{,}920\times10^{-4}$	$h(35)$	$5{,}380\times10^{-3}$	$h(53)$	$4{,}619\times10^{-4}$		

Tabela A.1: Coeficientes dos filtros *antialiasing* e antiimagem da primeira fase do projeto.

A Tabela A.2 mostra os coeficientes desses filtros.

$h(1)$	6,107 $\times 10^{-1}$	$h(5)$	4,113 $\times 10^{-2}$
$h(3)$	-1,446 $\times 10^{-1}$	$h(7)$	-7,471 $\times 10^{-3}$

Tabela A.2: Coeficientes do filtro protótipo do equalizador linear de [1].

A.3 Filtros-Protótipo – FFB

Os filtros do FFB são filtros de meia banda, simétricos (um quarto de coeficientes não-nulos e não-unitários representados) e com ordens que variam de acordo com a sua posição na árvore. As Tabelas A.3, A.4, A.5 e A.6 mostram os coeficientes não-nulos e não-unitários para cada filtro de cada nível da árvore do FFB.

	Real	**Imaginário**
$h(1)$	6,241 $\times 10^{-1}$	0
$h(3)$	-1,767 $\times 10^{-1}$	0
$h(5)$	7,477 $\times 10^{-2}$	0
$h(7)$	-3,225 $\times 10^{-2}$	0

Tabela A.3: Coeficientes do filtro protótipo do nível 1 do FFB.

	Real	**Imaginário**
$h(1)$	6,125 $\times 10^{-1}$	0
$h(3)$	-1,478 $\times 10^{-1}$	0
$h(5)$	4,322 $\times 10^{-2}$	0

Tabela A.4: Coeficientes do filtro protótipo do nível 2 do FFB.

A.4 Novos Filtros *Antialiasing* e Antiimagem

Filtros *antialiasing* e antiimagem utilizados na estrutura em multitaxa que corrigiram o problema da complementaridade de canais adjacentes. São filtros passa-baixas, simétricos (metade de coeficientes não-nulos representados), idênticos e de or-

	Real	**Imaginário**
$h(1)$	0	6,125 $\times 10^{-1}$
$h(3)$	0	1,478 $\times 10^{-1}$
$h(5)$	0	4,322 $\times 10^{-2}$

Tabela A.5: Coeficientes do filtro protótipo do nível 3 do FFB.

	Real	**Imaginário**
$h(1)$	5,758 $\times 10^{-1}$	0
$h(3)$	-7,796 $\times 10^{-2}$	0

Tabela A.6: Coeficientes do filtro protótipo do nível 4 do FFB.

dem 140. Para o projeto destes filtros, utilizou-se a função `firpm` do Matlab® com a seguinte configuração: `firpm(140,[0 0.28 0.353 1],[1 1 0 0],[1 0.09])`. Os coeficientes obtidos estão na Tabela A.7.

$h(0)$	$3{,}196\times10^{-1}$	$h(18)$	$-9{,}722\times10^{-3}$	$h(36)$	$-3{,}252\times10^{-3}$	$h(54)$	$-4{,}064\times10^{-4}$
$h(1)$	$2{,}683\times10^{-1}$	$h(19)$	$2{,}960\times10^{-3}$	$h(37)$	$-1{,}457\times10^{-3}$	$h(55)$	$-3{,}955\times10^{-4}$
$h(2)$	$1{,}437\times10^{-1}$	$h(20)$	$1{,}119\times10^{-2}$	$h(38)$	$1{,}302\times10^{-3}$	$h(56)$	$-5{,}180\times10^{-5}$
$h(3)$	$1{,}359\times10^{-2}$	$h(21)$	$8{,}565\times10^{-3}$	$h(39)$	$2{,}487\times10^{-3}$	$h(57)$	$2{,}530\times10^{-4}$
$h(4)$	$-6{,}036\times10^{-2}$	$h(22)$	$-1{,}067\times10^{-3}$	$h(40)$	$1{,}335\times10^{-3}$	$h(58)$	$2{,}680\times10^{-4}$
$h(5)$	$-5{,}957\times10^{-2}$	$h(23)$	$-8{,}375\times10^{-3}$	$h(41)$	$-7{,}688\times10^{-4}$	$h(59)$	$5{,}242\times10^{-5}$
$h(6)$	$-1{,}321\times10^{-2}$	$h(24)$	$-7{,}387\times10^{-3}$	$h(42)$	$-1{,}849\times10^{-3}$	$h(60)$	$-1{,}508\times10^{-4}$
$h(7)$	$2{,}977\times10^{-2}$	$h(25)$	$-1{,}600\times10^{-4}$	$h(43)$	$-1{,}152\times10^{-3}$	$h(61)$	$-1{,}714\times10^{-4}$
$h(8)$	$3{,}737\times10^{-2}$	$h(26)$	$6{,}157\times10^{-3}$	$h(44)$	$4{,}068\times10^{-4}$	$h(62)$	$-4{,}132\times10^{-5}$
$h(9)$	$1{,}260\times10^{-2}$	$h(27)$	$6{,}234\times10^{-3}$	$h(45)$	$1{,}333\times10^{-3}$	$h(63)$	$8{,}600\times10^{-5}$
$h(10)$	$-1{,}711\times10^{-2}$	$h(28)$	$9{,}103\times10^{-4}$	$h(46)$	$9{,}462\times10^{-4}$	$h(64)$	$1{,}025\times10^{-4}$
$h(11)$	$-2{,}646\times10^{-2}$	$h(29)$	$-4{,}419\times10^{-3}$	$h(47)$	$-1{,}762\times10^{-4}$	$h(65)$	$2{,}771\times10^{-5}$
$h(12)$	$-1{,}178\times10^{-2}$	$h(30)$	$-5{,}142\times10^{-3}$	$h(48)$	$-9{,}305\times10^{-4}$	$h(66)$	$-4{,}668\times10^{-5}$
$h(13)$	$1{,}015\times10^{-2}$	$h(31)$	$-1{,}314\times10^{-3}$	$h(49)$	$-7{,}412\times10^{-4}$	$h(67)$	$-6{,}046\times10^{-5}$
$h(14)$	$1{,}964\times10^{-2}$	$h(32)$	$3{,}074\times10^{-3}$	$h(50)$	$4{,}165\times10^{-5}$	$h(68)$	$-2{,}476\times10^{-5}$
$h(15)$	$1{,}081\times10^{-2}$	$h(33)$	$4{,}141\times10^{-3}$	$h(51)$	$6{,}268\times10^{-4}$	$h(69)$	$6{,}846\times10^{-5}$
$h(16)$	$-5{,}810\times10^{-3}$	$h(34)$	$1{,}470\times10^{-3}$	$h(52)$	$5{,}544\times10^{-4}$	$h(70)$	$-8{,}242\times10^{-6}$
$h(17)$	$-1{,}482\times10^{-2}$	$h(35)$	$-2{,}054\times10^{-3}$	$h(53)$	$2{,}646\times10^{-5}$		

Tabela A.7: Coeficientes dos novos filtros *antialiasing* e antiimagem que corrigiram os problemas dos canais adjacentes.

Apêndice B

Descrição das Classes

Este apêndice contém a descrição das classes utilizadas neste projeto com seus atributos e métodos descritos sucintamente.

B.1 Classe `hBandFilter`

Esta classe implementa os filtros de meia-banda interpolados da Seção 2.2.

B.1.1 Atributos

- `unsigned inerpolation_factor`: variável membro – fator de interpolação do filtro;
- `unsigned m_filter_length`: variável membro – comprimento do filtro;
- `float output`: variável membro – sinal filtrado;
- `unsigned m_mem_length`: variável membro – tamanho do comprimento do filtro;
- `float *m_filter_coef`: variável membro – contém os coeficientes não-nulos e não-unitários do filtro;
- `struct MemType`: variável membro – essa estrutura modela a memória do filtro e permite um rápido cálculo da saída. Essa estrutura contém um elemento `float mem_item` que representa o valor guardado na maméoria do filtro e

contém 3 ponteiros (`MemType *previous`, `MemType *step_right` e `MemType *step_left`) que apontam para os elementos vizinhos não-nulos;

- `MemType *m_first`: variável membro – ponteiro para o primeiro elemento da memória do filtro;
- `MemType *m_middle_element`: variável membro – ponteiro para o elemento central da memória do filtro.

B.1.2 Métodos

- `void shiftMem(const float new_item)`: método privado – desloca o vetor para a direita inserindo o `new_item` no primeiro *slot*;
- `void loadCoef(const float *in_coef_vector)`: método privado – carrega os coeficientes no vetor `in_coef_vector`. `in_coef_vector` deve ter comprimento `m_filter_length`.
- `float filter(const float new_sample, float *comp_output)`: método público – adiciona a variável `new_sample` à memória do filtro e calcula a saída do filtro e de seu complementar.
- `void set_param(const long length,float i_factor,float *fil_coef)`: método público – define os parâmetros e cria a lista encadeada.

B.2 Classe `hFilter`

A classe `hFilter` contém os mesmos atributos e métodos da classe `hBandFilter`. A diferença está nos filtros utilizados, que, neste caso, são apenas simétricos. Os únicos métodos alterados foram o `void loadCoef(const float *in_coef_vector)` e o `void set_param(const long length,float i_factor,float *fil_coef)` que, agora, implementam filtros somente simétricos em que o coeficiente central não é unitário.

B.3 Classe `hMyStack`

A classe `hMyStack` implementa o bloco dos atrasos.

B.3.1 Atributos

- `std::deque<float> m_stack`: variável membro privada – pilha que implementa o atraso em si;
- `unsigned int m_size`: variável membro privada – tamanho da pilha / atraso;
- `float m_lastInFront`: variável membro privada – elemento no topo da pilha;
- `bool m_overflow`: variável membro privada – variável *booleana* que diz se a pilha está cheia.

B.3.2 Métodos

- `void push(const float stuff)`: método público – método que adiciona um elemento na pilha;
- `void setSize (const unsigned int size)`: método público – define o tamanho da pilha;
- `const unsigned int getSize() const`: método público – retorna o tamanho da pilha;
- `const bool isOverflow() const`: método público – verifica se a pilha está cheia;
- `const float overflowedElement() const`: método público – retorna o elemento "expulso" da pilha.

B.4 Classe `hBandFilterTree`

Esta classe implementa a árvore propriamente dita que compõe o equalizador linear da Seção 3.1.2.

B.4.1 Atributos

- `FfbFilterTree m_ffb_filter_tree`: variável membro – contém a árvore da FFB;
- `unsigned mTotalDelay`: variável membro – contém o atraso total do equalizador linear.

B.4.2 Métodos

- `unsigned getDelay()`: método público – retorna o atraso total do equalizador linear;
- `void filter(float input, float *output, float gain [N_GAINS])`: método público – função que retorna a amostra filtrada por toda a árvore do equalizador linear com seus ganhos em cada canal.

B.5 Classe `FfbFilter`

Esta classe implementa a filtragem de um único filtro dentro do FFB.

B.5.1 Atributos

- `unsigned m_filter_length`: variável membro – contém o tamanho do comprimento do filtro;
- `struct MyComplex`: variável membro – estrutura que implementa números complexos (parte real e imaginária) e suas operações (soma, subtração, multiplicação e divisão;
- `MyComplex m_phase_shift`: variável membro – contém um atraso de fase constante aplicado aos elementos do filtro;
- `MyComplex output`: variável membro – utilizada para obtenção da saída do filtro;
- `unsigned m_mem_length`: variável membro – contém o tamanho da memória do filtro;

- `MyComplex *m_filter_coef`: variável membro – ponteiro para os coeficientes não-nulos e não-unitários do filtro;
- `MemType *m_first`: variável membro – ponteiro para o primeiro elemento da memória do filtro;
- `MemType *m_middle_element`: variável membro – ponteiro para o elemento central da memória do filtro.

B.5.2 Métodos

- `void shift_mem(const MyComplex new_item)`: método privado – desloca o vetor para a direita inserindo o `new_item` no primeiro *slot*;
- `void shiftMem(const float new_item)`: método privado – desloca o vetor para a direita inserindo o `new_item` no primeiro *slot*;
- `void loadCoef(const float *in_coef_vector)`: método privado – carrega os coeficientes no vetor `in_coef_vector`. `in_coef_vector` deve ter comprimento `m_filter_length`.
- `MyComplex filter(const MyComplex new_sample, MyComplex *comp_output)`: método público – adiciona a variável `new_sample` à memória do filtro e calcula a saída do filtro e de seu complementar;
- `void set_param(const long length,const unsigned ii,const unsigned jj, MyComplex *fil_coef)`: método público – define os parâmetros do filtro e cria as listas encadeadas.

B.6 Clase `FfbFilterTree`

Esta classe implementa a árvore do FFB.

B.6.1 Atributos

- `FfbFilter m_ffb_filter_vec[M-1]`: variável membro – vetor composto pelos filtros que serão utilizados na árvore do FFB;

- `MyComplex m_output_vector[M]`: variável membro – vetor que contém a saída de cada canal do FFB;
- `unsigned m_delay`: variável membro pública – contém o atraso gerado pela árvore do FFB.

B.6.2 Métodos

- `unsigned read_line2(int &coef_index,double *coef)`: método privado – lê os coeficientes dos filtros de um arquivo de cabeçalho e carrega no vetor `coef`;
- `void filter(float input, MyComplex *output)`: método público – realiza a filtragem de uma amostra e retorna um vetor contendo a saída para cada um dos canais do FFB.

B.7 Classe `hEqualizerFull`

Esta classe implementa a estrutura completa em multitaxa.

B.7.1 Atributos

- `hBandFilterTree mEq[N_BRANCHES]`: variável membro – vetor contendo os equalizadores lineares, um para cada ramo;
- `hFilter mDecFilters[N_BRANCHES-1]`: variável membro – vetor contendo os filtros *antialiasing*;
- `hFilter mIterpFilters[N_BRANCHES-1]`: variável membro – vetor contendo os filtros antiimagem;
- `MyStack mBranchDelay[N_BRANCHES-1]`: variável membro – vetor contendo os blocos de atraso para cada ramo;
- `unsigned *mBranchPeriod`: variável membro – vetor que contém os equalizadores que estão ativos durante cada período de tempo;

- `unsigned mBranchPeriodSize`: variável membro – tamanho do vetor acima, $2^{(\texttt{N\_BRANCHES}-1)}$;

- `unsigned mActualBranch`: variável membro – posição atual no vetor `mBranchPeriod`;

- `unsigned mBranch`: variável membro – ramos ativos no período atual, `mBranchPeriod[mActualBranch]`.

B.7.2 Métodos

- `unsigned read_line2(double *coef)`: método privado – lê os coeficientes dos filtros *antialiasing* e antiimagem de um arquivo de cabeçalho e carrega no vetor `coef`;

- `void equalizer(float input, float *output, float gain[N_BRANCHES][N_GAINS])`: método público – realiza a filtragem completa pelo equalizador multitaxa.