Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

NUSTENIL SEGUNDO DE MORAES LIMA MARINUS

PROCESSAMENTO DIGITAL DE SINAIS APLICADOS À EFEITOS DE ÁUDIO

Campina Grande, Paraíba
Fevereiro de 2011

NUSTENIL SEGUNDO DE MORAES LIMA MARINUS

PROCESSAMENTO DIGITAL DE SINAIS APLICADOS A EFEITOS DE ÁUDIO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Orientador:
Professor Edmar Candeia Gurjão, D. Sc.

Campina Grande, Paraíba
Fevereiro de 2011

NUSTENIL SEGUNDO DE MORAES LIMA MARINUS

PROCESSAMENTO DIGITAL DE SINAIS APLICADOS A EFEITOS DE ÁUDIO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Edmar Candeia Gurjão, D. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho aos meus pais, que sempre me apoiaram nas horas difíceis e sempre me proporcionaram tudo de melhor que estava no alcance deles.

AGRADECIMENTOS

Agradeço à minha mãe e ao meu pai, Maria de Moraes e Antonio Nustenil, pelo esforço e dedicação na minha criação, me proporcionando educação de qualidade e ensinamentos que levarei por toda minha vida.

Agradeço também os meus irmãos João Vilian e Marinus Lima por sempre estarem comigo me ajudando e por serem exemplos de vida para mim.

Agradeço à minha namorada Germana Rafaela pela paciência e cuidados comigo.

Um agradecimento a Rubem Medeiros e ao prof. Edmar Candeia pela ajuda e orientação para a conclusão desse trabalho.

Agradeço a Pierre Camilo por ter me ajudado na parte final desse trabalho, aos meus companheiros do grupo PET-Elétrica, meus amigos que ingressaram junto comigo no curso de Engenharia Elétrica e a todos que de alguma forma, passaram pela minha vida e contribuíram para a construção de quem sou hoje.

"Agradeço todas as dificuldades que enfrentei; Se não fossem por elas, não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas nos auxiliam muito"

Chico Xavier.

RESUMO

Este trabalho trata do desenvolvimento de algoritmos eficientes de processamento digital de sinais para o uso na geração de efeitos de áudio. Foram implementados diversos tipos de filtros digitais com o uso do algoritmo CORDIC para melhorar a eficiência dos filtros. Após os filtros estarem prontos, foi desenvolvido um software dividido em duas partes, sendo a primeira para a aplicação dos filtros tipo função – janela FIR (passa-baixas, passa-altas, rejeita-faixa e passa-faixa) em arquivos de áudio no formato WAVE, possibilitando a visualização da resposta no domínio da frequência, do filtro desejado, do sinal no tempo, do sinal filtrado no tempo e na frequência. A segunda parte do software é um equalizador digital gráfico que é aplicado no arquivo de áudio. Todas as funções feitas foram desenvolvidas em ANSI C e otimizadas com o uso de algoritmos eficientes para facilitar o seu uso em outros *hardwares*, como DSP's e FPGA's.

Palavras-chave: Áudio, Filtros Digitais, CORDIC, Equalizador Gráfico.

ABSTRACT

This work deals with the development of efficient algorithms for digital signal processing for use in audio effects. Various types of digital filters were implemented using the CORDIC algorithm to improve the efficiency and speed of the filters. After the filters are ready, a program was developed in two parts, where the first part is an application of type function filter - FIR window (low-pass, high pass, band-reject and band-pass) to audio files in WAVE format, enabling visualization of the signal in the frequency domain, of the desired filter to apply in the signal, the signal in time, the filtered signal and filtered signal in frequency domain. The second part of the program is a graphic digital equalizer that is applied to the audio file. All functions performed were developed in ANSI C and optimized with the use of efficient algorithms to facilitate its use on other hardware such as DSP's and FPGA's.

Keywords: Digital Signals, Digital Filters, CORDIC, Graphic Equalizer.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Devido à grande utilização nos dias atuais de músicas, vídeos e imagens em formato digital, o processamento digital de sinais vem ganhando cada vez mais importância e espaço no mundo moderno.

Um sinal é uma grandeza que varia no tempo e/ou espaço (de Carvalho, 2002). O som, por exemplo, é a pressão do ar em função do tempo em determinado ponto do espaço (Pierce, 1989).

Para extrair características importantes e desejáveis de um sinal digital, usam-se técnicas de processamento digital de sinais. Para sinais de áudio, a forma mais comum de processamento é a filtragem em determinadas freqüências a fim de tornar um sinal (uma música, por exemplo) mais agradável ao ouvinte.

Um conjunto de filtros para processar sinais de áudio recebe o nome de equalizador, que é responsável pelo ajuste de graves, médios e agudos, no contexto de espectro de freqüência de um sinal de áudio (da Silva, 2007).

Neste trabalho foi realizado a implementação em linguagem C de vários algoritmos eficientes de processamento digital de sinais, a fim de se obter efeitos desejáveis em arquivos de áudio digital.

Este trabalho é dividido em introdução teórica, descrição doa algoritmos desenvolvidos e resultados da aplicação feita com tais algoritmos.

1.1 OBJETIVOS

Desenvolvimento de filtros digitais eficientes em linguagem C para o uso em áudio.

Criação de uma plataforma de testes para os filtros implementados e também o desenvolvimento de uma aplicação de processamento digital de sinais aplicados a áudios digitais.

2 EMBASAMENTO TEÓRICO

Neste capítulo, alguns conceitos matemáticos e computacionais de processamento digital de sinais são introduzidos.

2.1 SINAIS E SISTEMAS

Um sinal pode ser entendido como uma função de uma ou mais variáveis independentes, cuja variação representa o comportamento de algum fenômeno natural (Diniz, da Silva, & Lima Neto, 2004). Na grande área da Engenharia Elétrica, um sinal é uma grandeza elétrica (tensão, por exemplo) cuja sua variação no tempo pode ser usada para representar algum fenômeno (de Carvalho, 2002).

Dispositivos conhecidos como transdutores são usados para converter fenômenos físicos (como exemplo, a voz humana, o som de um instrumento musical, uma paisagem, a temperatura de um ambiente ou processo químico, etc.) em grandeza elétrica, como tensão ou corrente.

Um sistema é um dispositivo que produz alterações em sinais, como pode ser visto na Figura 1. Um sistema é caracterizado por sua relação entrada/saída, ou seja, pela transformação que opera em um sinal à sua entrada, para produzir outro sinal em sua saída (de Carvalho, 2002).

Figura 1: Visão geral de um sistema.

Com o grande e rápido desenvolvimento da eletrônica digital e sistemas computacionais, o conjunto de técnicas e métodos usados para processamento de sinais foi dividido em dois grandes ramos, o Processamento Analógico de Sinais, que é responsável pelo tratamento de sinais que variam continuamente no tempo e o

Processamento Digital de Sinais, que trata de sinais digitais no tempo, ou seja, sinais para os quais a variável independente, tempo, assume uma forma discreta.

Nas Figuras 2 e 3 pode-se observar a representação gráfica de um sinal no tempo contínuo e um sinal no tempo discreto, respectivamente (de Carvalho, 2002).

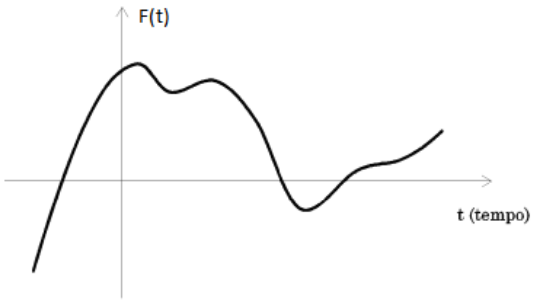

Figura 2: Representação gráfica de um sinal no tempo contínuo.

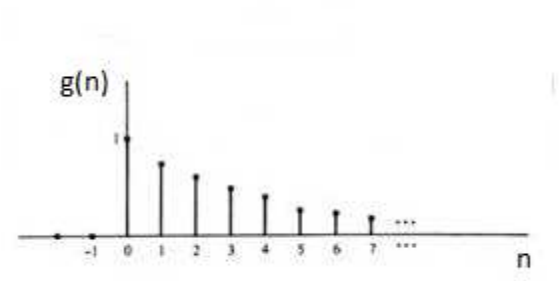

Figura 3: Representação gráfica de um sinal no tempo discreto.

Como exemplo de sinais mais comuns no uso de processamento de sinais pode-se citar a função seno e a exponencial, dadas pelas Equações (1) e (2), respectivamente.

$$x(t) = Asen(\omega t + \varphi). \quad (1)$$

Na Equação (1), tem-se a variável independente *t* (pode ser contínua ou discreta) e as constantes do sinal que são a freqüência angular ω, a amplitude do sinal A e a defasagem φ. Na Figura 4 pode ser observado um exemplo da função seno no tempo continuo.

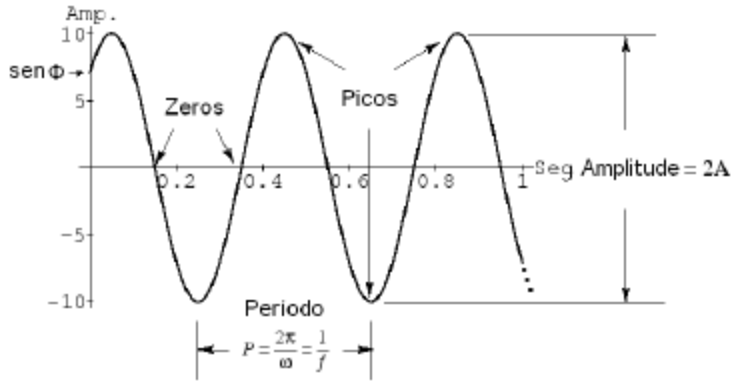

Figura 4: Exemplo de uma senóide contínua (da Silva, 2007).

$$x(t) = Ae^{-t/\tau}. \tag{2}$$

Na Equação (2) tem-se a variável independente t e as constantes do sinal que são a amplitude do sinal A e a constante de tempo τ. Na Figura 5 pode-se observar um exemplo de função exponencial.

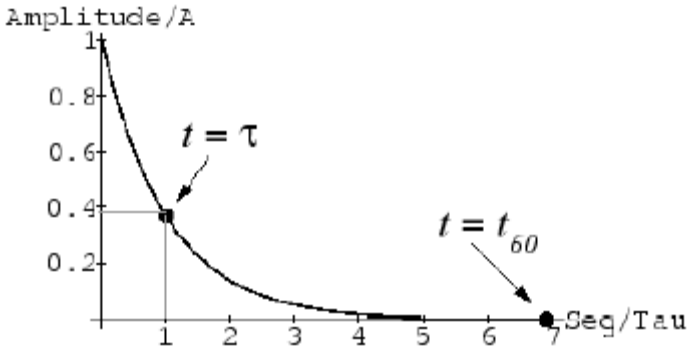

Figura 5: Exemplo de uma função exponencial contínua (da Silva, 2007).

2.2 CONVOLUÇÃO

Uma operação bastante comum em processamento digital de sinais é chamada de convolução que é dada, no domínio do tempo, pelas Equações (3) e (4), no tempo continuo e discreto, respectivamente (Oppenheim, Willsky, & Young, 1983).

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau. \quad (3)$$

Na Equação (3) tem-se que a função *x(t)* é o sinal de entrada, a função *h(t)* é a resposta ao impulso do sistema e *y(t)* é a saída do sistema.

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k]. \quad (4)$$

Na Equação (4) tem-se que a função *x[n]* é o sinal de entrada, a função *h[n]* é a resposta ao impulso do sistema e *y[n]* é a saída do sistema.

A operação de convolução pode ser representada pelo operador *,tanto para tempo continuo como discreto, conforme mostrado nas Equações (5) e (6).

$$y(t) = x(t) * h(t). \quad (5)$$

$$y[n] = x[n] * h[n]. \quad (6)$$

A convolução é uma operação associativa, comutativa e distributiva, como pode ser visto na Tabela 1 (Oppenheim, Willsky, & Young, 1983).

Tabela 1: Propriedades da convolução (Oppenheim, Willsky, & Young, 1983).

Propriedade	Contínuo	Discreto
Comutativa	$x(t) * h(t) = h(t) * x(t)$	$x[n]*h[n] = h[n] * x[n]$
Associativa	$x(t)*[h_1(t)*h_2(t)]=[x(t)*[h_1(t)]*h_2(t)$	$X[n]*(h_1[n]*h_2[n]) = (x[n]*[h_1[n])*h_2[n]$
Distributiva	$x(t)*[h_1(t)+h_2(t)] = [x(t)*h_1(t)]+[x(t)*h_2(t)]$	$X[n]*(h_1[n]+h_2[n]) = (x[n]*h_1[n])+(x[n]*h_2[n])$

2.3 TRANSFORMA DE FOURIER

Os sinais podem ser analisados de diferentes formas, sendo que as mais comuns são a análise no domínio do tempo e no domínio da freqüência. Um dos métodos para realizar a transformação de do domínio do tempo para a freqüência é através da Transformada de Fourier.

Uma das formas da transformada contínua de Fourier é definida pela Equação (7) (Oppenheim, Willsky, & Young, 1983).

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}dt. \quad (7)$$

As funções *F(ω)* e *f(t)* são funções contínuas no domínio da frequência e tempo, respectivamente. O termo $e^{-i\omega t}$ representa uma senóide complexa, que é definida pela Equação (8).

$$Ae^{-i(\omega t+\varphi)} = Acos(\omega t + \varphi) + iAsen(\omega t + \varphi). \quad (8)$$

A transformada de Fourier expressa o produto interno da função no domínio do tempo pela função da senóide complexa, ou seja, a transformada é uma operação de produto interno que calcula os coeficientes de projeção da função *f(t)* sobre a senóide complexa. *F(ω)* é a medida da amplitude e da fase da senóide complexa que representa o sinal de entrada na freqüência ω.

Uma condição necessária para que se possa realizar a transformada de Fourier de uma função *f(t)* qualquer é dada pela Equação (9), ou seja, a função deve ser integrável e finita (Lathi, 2004).

$$\int_{-\infty}^{+\infty} |f(t)|dt < \infty. \quad (9)$$

Pode-se ainda fazer a transformação inversa, domínio da freqüência para o domínio do tempo, usando a transformada inversa contínua de Fourier, definida pela Equação 10 (Mitra, 1998.)

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)e^{i\omega t}dt. \quad (10)$$

Na Tabela 2 podem-se observar algumas propriedades da transformada de Fourier.

Tabela 2: Algumas propriedades da Transformada de Fourier (Mitra, 1998.).

Propriedade	Domínio no tempo	Domínio na freqüência
Linearidade	$\alpha f(t) + \beta f(t)$	$\alpha F(\omega) + \beta F(\omega)$
Atraso no tempo	$f(t-\tau)$	$e^{-i\omega\tau}F(\omega)$
Atraso na freqüência	$e^{i\omega_0 t}f(t)$	$F(\omega-\omega_0)$
Convolução	$f(t)*g(t)$	$F(\omega)G(\omega)$

2.4 SINAIS NO TEMPO DISCRETO

Um sinal no tempo discreto é aquele que pode ser representado por uma seqüência de números, como pode ser observado na Equação (11) (Diniz, da Silva, & Lima Neto, 2004).

$$\{x(n),\ n \in \mathbb{Z}\}. \tag{11}$$

O intervalo entre dois valores de n é chamado de tempo de amostragem e o sinal pode ser caracterizado pelo seu tempo e não pela amostragem, como pode ser visto na Equação (12).

$$\{x(nT),\ n \in \mathbb{Z}\}. \tag{12}$$

Um sinal no tempo contínuo pode ser discretizado com um determinado período de amostragem. De acordo com critério de Nyquist, a freqüência mínima para a amostragem de um sinal é o dobro da máxima freqüência do sinal no tempo contínuo. Como exemplo, uma senóide com freqüência de 60 Hz só pode ser amostrada com

freqüência de amostragem igual ou superior a 120 Hz, caso contrário, não será possível a recuperação do sinal contínuo original (Lathi, 2004).

Na Figura 6 é retratado um diagrama de blocos genérico e simplificado da maioria dos processos de tratamento digital de sinais, onde o sinal original contínuo é discretizado e em seguida é realizado o processamento do sinal discreto com algoritmos a fim de satisfazer alguma necessidade. Por fim, o sinal é convertido para contínuo.

Figura 6: Esquema básico do processo de tratamento de sinais digitais.

2.5 TRANSFORMADA Z E A TRANSFORMADA DISCRETA DE FOURIER

A transformada Z é uma sequência x(n) definida pela Equação (13).

$$X(z) = \mathrm{Z}\{x(n)\} = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{13}$$

Na equação 13, *z* é uma variável complexa. Nota-se que *X(z)* só é definida para as regiões do plano complexo em que o somatório à direita converge.

A transformada inversa é dada pela Equação (14) (Diniz, da Silva, & Lima Neto, 2004).

$$x(n) = \frac{1}{2\pi \mathrm{i}} \oint \mathrm{X(z)z^{n-1}dz} \tag{14}$$

Como exemplo, a transformada Z do sinal $y[n] = a_1x[n] + a_2x[n-1] + a_3x[n-2]$ pode ser esquematizada de acordo com a Figura 7.

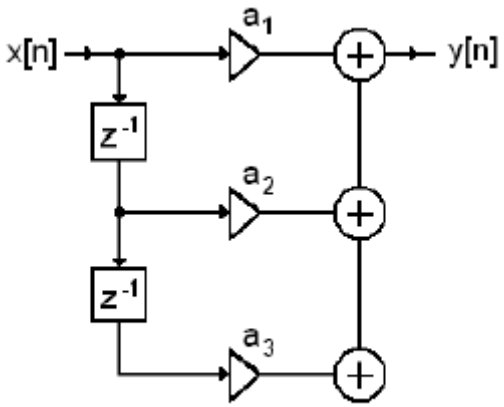

Figura 7: Sistema no tempo discreto representado por um diagrama com transformada Z (Mitra, 1998.).

Para realizar a análise na freqüência, ou seja, realizar a transformada de Fourier discreta de um sinal discreto, basta usar a transforma Z para o caso em que $z = e^{j\omega}$.

Assim, a transformada discreta de Fourier é dada pela Equação (15) e sua transformada inversa pela Equação (16).

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{15}$$

$$x(n) = \frac{1}{2\pi \mathrm{i}} \oint \mathrm{X}(e^{j\omega})e^{j\omega n}\mathrm{d}\omega \tag{16}$$

Para sequências definidas em k = 0,1,2,...,N-1, tem-se as Equações (17) e (18) para a transformada de Fourier discreta e transformada inversa.

$$X(\omega_k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \tag{17}$$

$$x(n) = \frac{1}{2\pi \mathrm{j}} \sum_{n=0}^{N-1} X(\omega_k)e^{j2\pi nk/N} \tag{18}$$

Importante ressaltar que se o sinal tem L < N amostras, o sinal deve ser preenchido com zero até atingir o comprimento N, adaptando o comprimento da

sequência para o cálculo de sua Transformada Discreta de Fourier (Diniz, da Silva, & Lima Neto, 2004).

2.6 TRANSFORMADA RÁPIDA DE FOURIER (FAST FOURIER TRANSFORM – FFT)

Para uma sequência de N amostras, o uso das Equações (15) e (16) para o cálculo da transformada discreta de Fourier e a transformada inversa, precisa-se realizar em torno de N^2 multiplicações complexas, isto é, a complexidade da transformada discreta de Fourier cresce com o quadrado do comprimento do sinal. Isso limita severamente seu uso prático para sinais longos. Em 1965, os pesquisadores Cooley e Turkey propuseram um algoritmo eficiente para o uso das Equações (15) e (16), o qual requer um número de multiplicações complexas da ordem de $N\log_2 N$. Este algoritmo é conhecido como FFT (Diniz, da Silva, & Lima Neto, 2004).

Na Figura 8 é ilustrado como essa redução é bastante significativa. Pode-se observar que para sinais com tamanho de apenas N = 128, que é um sinal considerado pequeno para muitas aplicações, a diferença já é bastante grande.

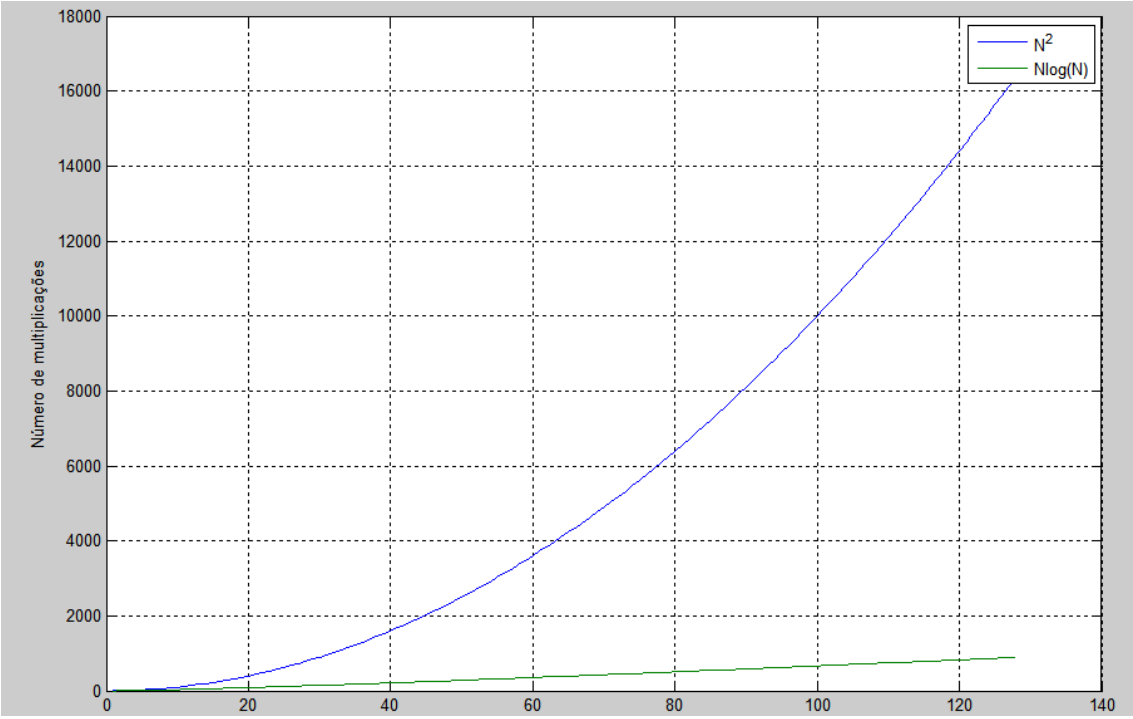

Figura 8: Comparação de complexidade da FFT e da Transformada discreta de Fourier.

2.7 FILTROS DIGITAIS

Para o processamento digital de sinais, existem dois tipos principais de filtros, que são os de resposta ao impulso finita (*Finite Impulse Response* - FIR), e resposta ao impulso infinita (*Infinite Impulse Response* – IIR).

Devido à grande dificuldade e complexidade de desenvolver funções eficientes em C para calcular parâmetros de filtros IIR, pois necessitam de muitos cálculos com variáveis complexas, foram desenvolvidos apenas filtros FIR.

A forma característica dos filtros FIR é dada pela Equação (19).

$$y(n) = \sum_{l=0}^{M} b_l x(n-l). \tag{19}$$

Na Equação (19) tem-se que os coeficientes b_l se relacionam diretamente com a resposta ao impulso do sistema, isto é, $b_l = h(l)$.

A transformada Z de um filtro FIR é dada pela Equação (20).

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{l=0}^{M} h(l) z^{-n}. \tag{20}$$

Os módulos dos filtros ideais mais comuns, passa-baixas, passa-altas, passa-faixa e rejeita-faixa são mostrados na Figura 9.

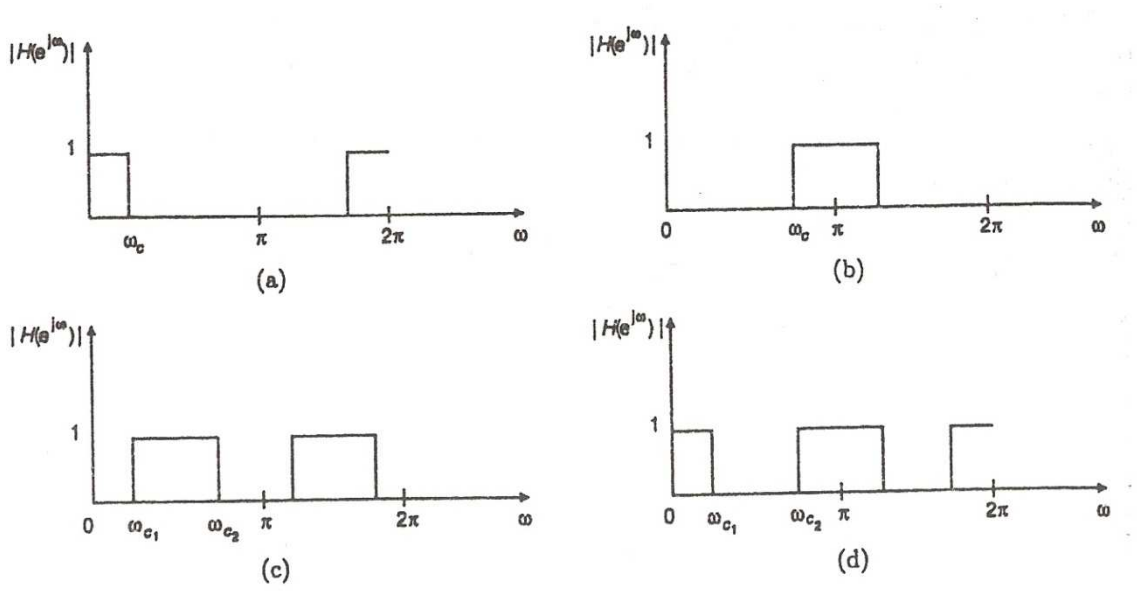

Figura 9: Respostas de módulo ideais: filtros (a) passa-baixas; (b) passa-altas; (c) passa-faixa; (d) rejeita-faixa (Diniz, da Silva, & Lima Neto, 2004).

Na Equação (21), temos a equação da resposta ao impulso do filtro passa-baixas e na Equação (22), módulo da função de transferência deste filtro.

$$h(n) = \begin{cases} \dfrac{w_c}{\pi}, & para\ n = 0 \\ \dfrac{1}{\pi n} sen(w_c n)\,, & para\ n \neq 0 \end{cases} \tag{21}$$

$$\left|H(e^{jw})\right| = \begin{cases} 1, & para\ 0 \leq |w| \leq w_c \\ 0, & para\ w_c < |w| \leq \pi \end{cases} \tag{22}$$

Na Equação 23 temos a equação da resposta ao impulso do filtro passa-altas e na Equação 24, módulo da função de transferência deste filtro.

$$h(n) = \begin{cases} 1 - \dfrac{w_c}{\pi}, & para\ n = 0 \\ -\dfrac{1}{\pi n} sen(w_c n)\,, & para\ n \neq 0 \end{cases} \tag{23}$$

$$\left|H(e^{jw})\right| = \begin{cases} 0\,, para\ 0 \leq |w| \leq w_c \\ 1, para\ w_c < |w| \leq \pi \end{cases} \tag{24}$$

Na Equação 25 temos a equação da resposta ao impulso do filtro passa-faixa e na Equação 26, módulo da função de transferência deste filtro.

$$h(n) = \begin{cases} \dfrac{(w_{c2} - w_{c1})}{\pi}, & para\ n = 0 \\ \dfrac{1}{\pi n}[sen(w_{c2} n) - sen(w_{c1} n)], para\ n \neq 0 \end{cases} \tag{25}$$

$$\left|H(e^{jw})\right| = \begin{cases} 0\,, & para\ 0 \leq |w| \leq w_{c1} \\ 1\,, para\ w_{c1} \leq |w| \leq w_{c2} \\ 0\,, & para\ w_{c2} \leq |w| \leq \pi \end{cases} \tag{26}$$

Na Equação 27 temos a equação da resposta ao impulso do filtro rejeita-faixa e na Equação 28, módulo da função de transferência deste filtro.

$$h(n) = \begin{cases} 1 - \dfrac{(w_{c2} - w_{c1})}{\pi}, & para\ n = 0 \\ \dfrac{1}{\pi n}\left[sen(w_{c1}n) - sen(w_{c2}n)\right], & para\ n \neq 0 \end{cases} \quad (27)$$

$$\left|H(e^{jw})\right| = \begin{cases} 1\ , & para\ 0 \leq |w| \leq w_{c1} \\ 0\ , & para\ w_{c1} \leq |w| \leq w_{c2} \\ 1\ , & para\ w_{c2} \leq |w| \leq \pi \end{cases} \quad (28)$$

Analisando as Equações 21, 22, 23, 24, 25, 26, 27 e 28, nota-se que as respostas ao impulso correspondentes a todos esses filtros ideais não são realizáveis, pois elas tem duração infinita e são não causais. Uma maneira contornar essa situação é definir uma sequência auxiliar *h'(n)* de comprimento finito que resulte num filtro de ordem M. Um método de aproximação do filtro ideal é a feito por meio da multiplicação da amplitude da resposta ao impulso *h(n)* por uma função-janela $w_r(n)$. A janela $w_r(n)$ deve ser projetada de forma a minimizar o desvio em relação a resposta ideal do filtro desejado. Desta forma, os coeficientes dos filtros ficam da forma da Equação 29.

$$h'(n) = h(n)w_r(n) \quad (29)$$

Dentre os tipos de filtros FIR, foram desenvolvidas as funções janela do tipo Retangular, Triangular, Hamming, Hanning, Blackman e de Kaiser.

A janela de Kaiser difere das outras, de forma que esta pode realizar o que o filtro IIR é capaz de fazer (Diniz, da Silva, & Lima Neto, 2004), por isso a não obrigatoriedade de fazer o filtro IIR.

Filtros FIR do tipo janela são feitos obtendo a função de transferência ideal do filtro desejado, passa-altas, passa-baixas, passa-faixas e rejeita-faixas e, após obter o filtro ideal, é feito a multiplicação no domínio do tempo (ou pela convolução no domínio da freqüência), por uma função de acordo com o tipo e ordem da janela desejada de forma a maximiza a aproximação do filtro realizado para o filtro ideal (Diniz, da Silva, & Lima Neto, 2004).

A Equação 30 mostra a janela do tipo retangular de ordem M.

$$w_r(n) = \begin{cases} 1, & para\ |n| \leq \dfrac{M}{2} \\ 0\ , & para\ |n| > \dfrac{M}{2} \end{cases} \tag{30}$$

Na Figura 10 é mostrado o gráfico de um exemplo do filtro tipo Janela Retangular.

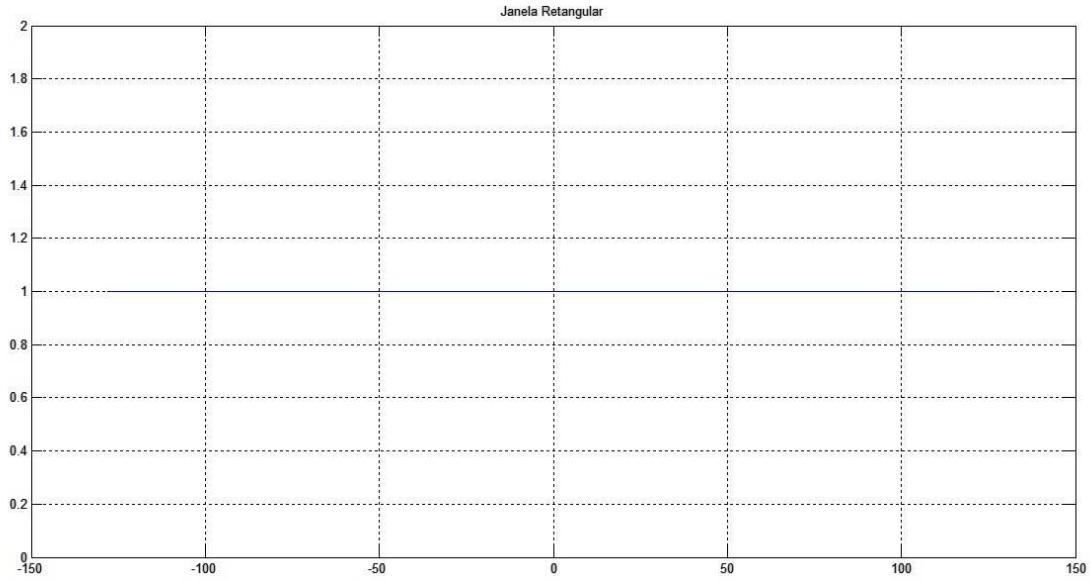

Figura 10: Filtro janela Retangular de ordem 256.

A Equação 31 mostra a janela do tipo triangular de ordem M.

$$w_r(n) = \begin{cases} -\dfrac{2|n|}{M+2} + 1, para\ |n| \leq \dfrac{M}{2} \\ 0\ , \quad para\ |n| > \dfrac{M}{2} \end{cases} \tag{31}$$

Na Figura 11 é mostrado o gráfico de um exemplo do filtro tipo Janela Triangular.

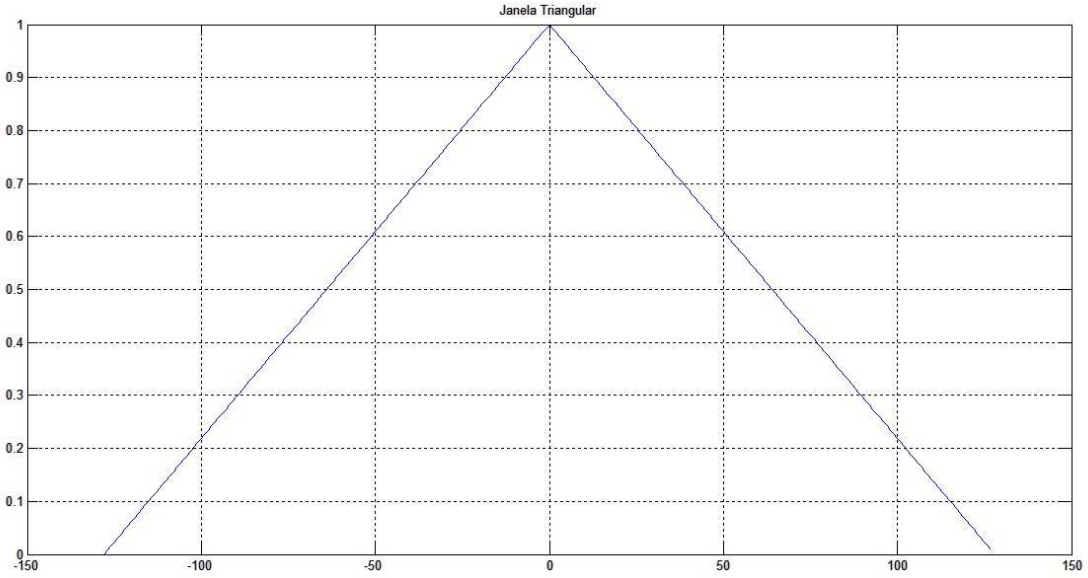

Figura 11: Filtro Janela Triangular de ordem 256.

A equação 32 mostra a janela do tipo Hamming de ordem M.

$$w_r(n) = \begin{cases} 0{,}54 + 0{,}46\cos\left(\frac{2\pi n}{M}\right), & para\ |n| \leq \frac{M}{2} \\ 0\ , & para\ |n| > \frac{M}{2} \end{cases} \quad (32)$$

Na Figura 12 é mostrado o gráfico de um exemplo do filtro tipo Janela de Hamming

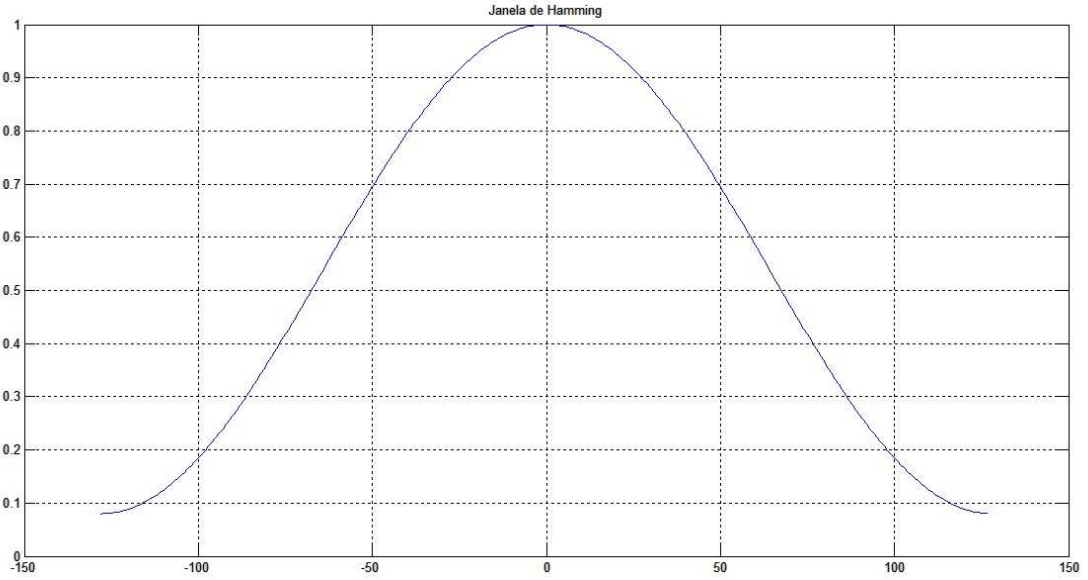

Figura 12: Filtro Janela de Hamming de ordem 256.

A Equação 33 mostra a janela do tipo Hanning de ordem M.

$$w_r(n) = \begin{cases} 0{,}5 + 0{,}5\cos\left(\frac{2\pi n}{M}\right), para\ |n| \leq \frac{M}{2} \\ 0\ , \qquad\qquad\qquad para\ |n| > \frac{M}{2} \end{cases} \tag{33}$$

Na Figura 13 é mostrado o gráfico de um exemplo do filtro tipo Janela de Hanning.

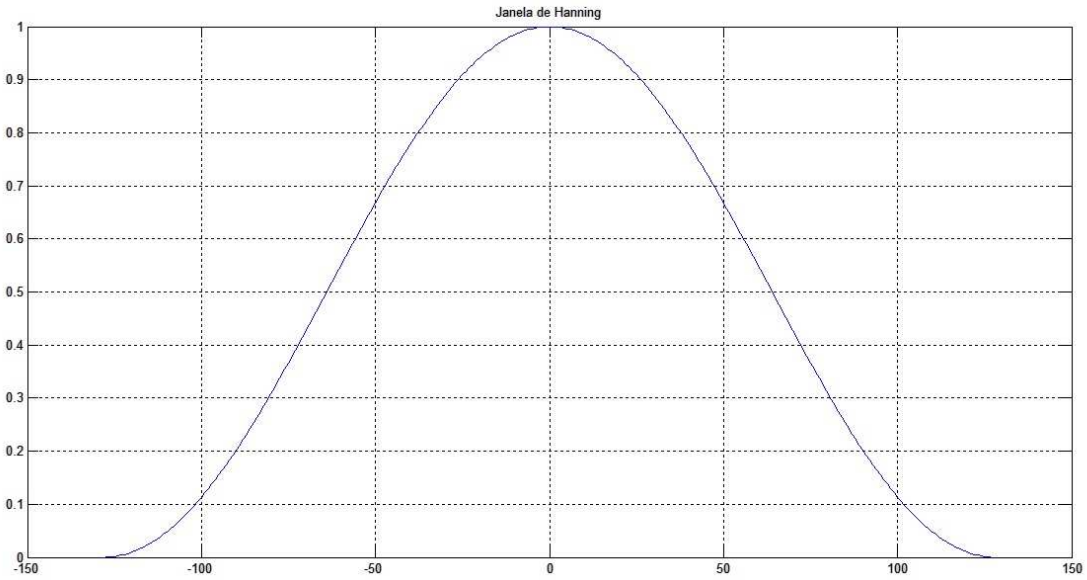

Figura 13: Filtro Janela de Hanning de ordem 256.

A Equação 34 mostra a janela do tipo Blackman de ordem M.

$$w_r(n) = \begin{cases} 0{,}42 + 0{,}5\cos\left(\frac{2\pi n}{M}\right) + 0{,}08\cos\left(\frac{4\pi n}{M}\right), para\ |n| \leq \frac{M}{2} \\ 0\ , \qquad\qquad\qquad\qquad\qquad\qquad para\ |n| > \frac{M}{2} \end{cases} \tag{34}$$

Na Figura 14 é mostrado o gráfico de um exemplo do filtro tipo Janela de Blackman.

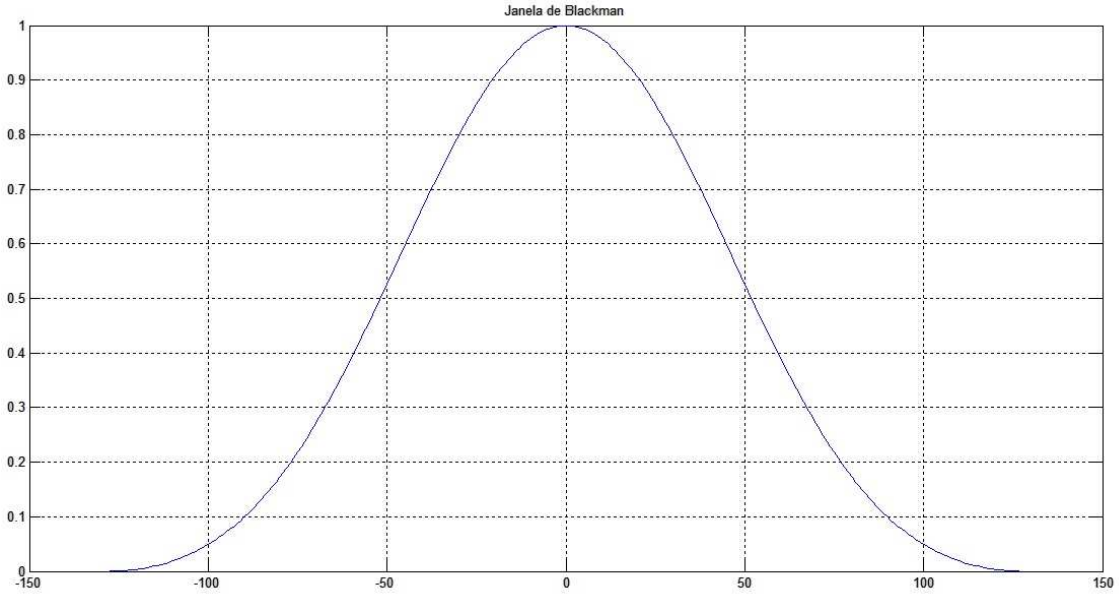

Figura 14: Filtro Janela de Blackman de ordem 256.

O desempenho e eficiência dos filtros FIR dos tipos janela Retangular, Triangular, Hamming, Hanning e Blackman só dependem da ordem, sendo mais precisos e próximos do filtro ideal de acordo com a ordem escolhida para os filtros. Quanto maior, mais próximo do filtro ideal desejado.

Já para o filtro FIR do tipo janela de Kaiser, a especificação do filtro é dada informando valores da função de transferência do filtro, como ondulação máxima da faixa de passagem e de rejeição, freqüência da banda de passagem e freqüência da banda de rejeição. Com essas especificações, o algoritmo calcula a ordem do filtro e a função a ser aplicada no filtro ideal desejado.

Na Figura 15 é mostrado um gráfico de uma função de transferência de um filtro passa-faixa contendo as especificações de um filtro tipo Janela de Kaiser.

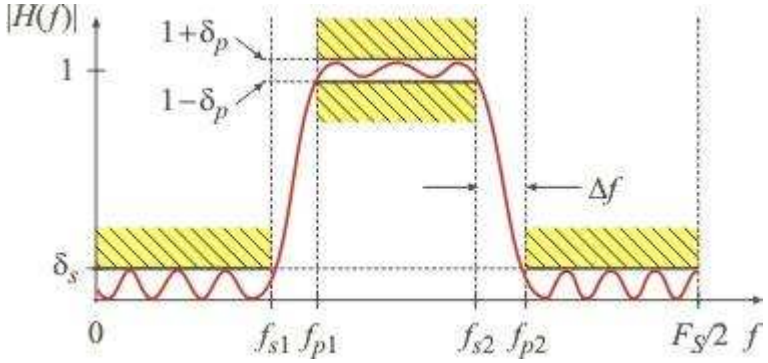

Figura 15: Filtro passa-faixa.

Na Figura 16 é mostrado o gráfico de um exemplo do filtro tipo Janela de Kaiser.

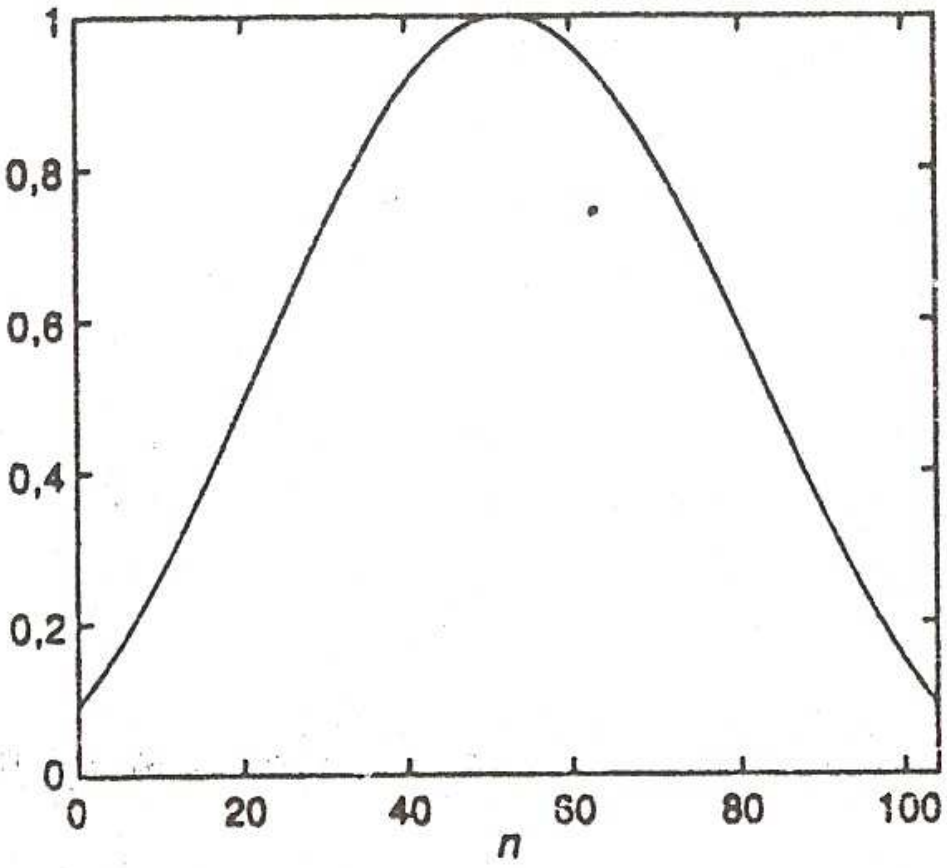

Figura 16: Filtro janela de Kaiser (Diniz, da Silva, & Lima Neto, 2004).

2.8 SINAIS DE ÁUDIO

O som é um efeito audível produzido por movimentos de corpos vibratórios, chamados de vibrações sonoras. Estas vibrações podem ser convertidas em sinais elétricos por meio de transdutores, como os contidos nos microfones e também os sinais elétricos podem ser convertidos em vibrações sonoras por transdutores, como os contidos nos alto-falantes (Pierce, 1989).

As vibrações sonoras apresentam uma potência que é proporcional ao quadrado da amplitude das vibrações, e possui como forma mais simples uma onda senoidal.

Formas complexas de vibrações sonoras podem ser sintetizadas como combinações de ondas senoidais em diferentes freqüências.

O som apresenta alguns parâmetros perceptíveis como a intensidade e altura.

A intensidade é a percepção da amplitude e da energia do sinal sonoro. Ela apresenta a propriedade do som ser fraco ou forte. Na Figura 17 é mostrado um sinal sonoro simples com amplitudes diferentes, ou seja, com potências diferentes. A potência sonora é geralmente medida em decibéis. Na Tabela 3, pode-se observar algumas faixas de intensidade sonora.

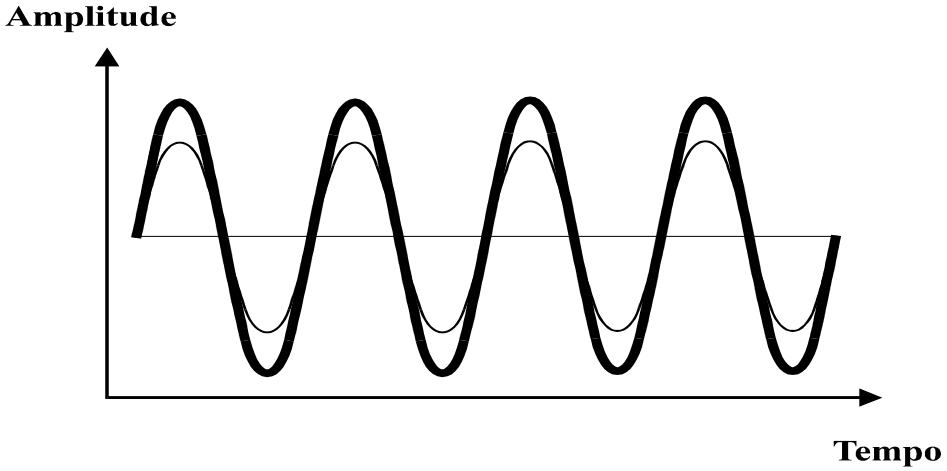

Figura 17: Dois sinais sonoros de amplitudes diferentes (Fechine, 2006)

Tabela 3: Faixas de intensidade sonora (Fechine, 2006).

Decibéis	Intensidade sonora	Exemplo
0 – 20	Muito-Baixo	Farfalhar das folhas
20 – 40	Baixo	Conversa silenciosa
40 – 60	Moderado	Conversa normal
60 – 80	Alto	Ruído médio de fábrica
80 – 100	Muito-alto	Apito de guarda
100 – 200	Ensurdecedor	Avião decolando

A altura é a percepção da freqüência fundamental do sinal sonoro. Apresenta a propriedade de o som ser grave ou agudo. Podem ser observados na Figura 18 dois sinais sonoros simples com freqüências diferentes.

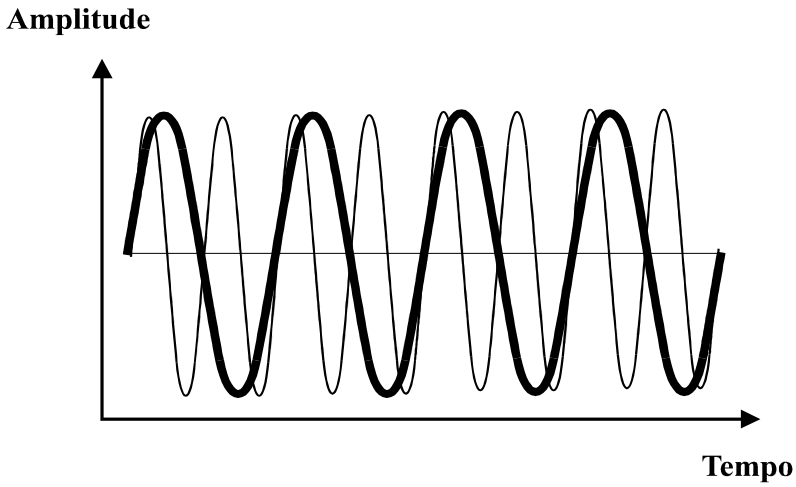

Figura 18: Dois sinais sonoros de freqüência diferentes (Fechine, 2006).

Como exemplos de alturas diferentes de som, a voz do homem tem em média 120 Hz enquanto que a da mulher e de uma criança tem 220 e 300 Hz, respectivamente. A percepção auditiva do ser humano é cerca de 20 Hz a 20 kHz (Fechine, 2006).

Os sinais de áudio atualmente são geralmente processados e tratados de forma digital. O sinal sonoro digital segue a regra da Figura 19, sendo processados nos dias atuais em computadores, telefones, leitores de CD's, etc. (da Silva, 2007). O processo de digitalização do som passa por uma filtragem, para limitar a sua faixa de freqüência, por uma amostragem no sinal, que é a conversão do sinal analógico em digital (sequência de pulsos), pela quantização que é a conversão dos pulsos em números binários, e por fim, na sua gravação em arquivos de áudio. Este processo simplificado é mostrado na Figura 19.

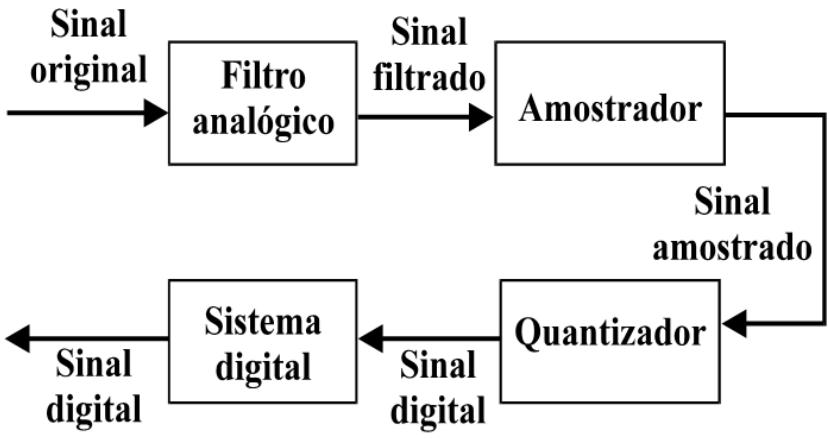

Figura 19: Processo simplificado de digitalização do sinal sonoro (Fechine, 2006).

Para atender o critério de Nyquist, os dados nos arquivos de áudio devem ter sido obtidos com um período de amostragem superior a 40 kHz, pois a capacidade auditiva humana é de 20 kHz. Como exemplo, os arquivos armazenados em CD's são de 44,1 kHz.e com 16 bits de quantização.

Um exemplo de um sinal digital de áudio é mostrado na Figura 20.

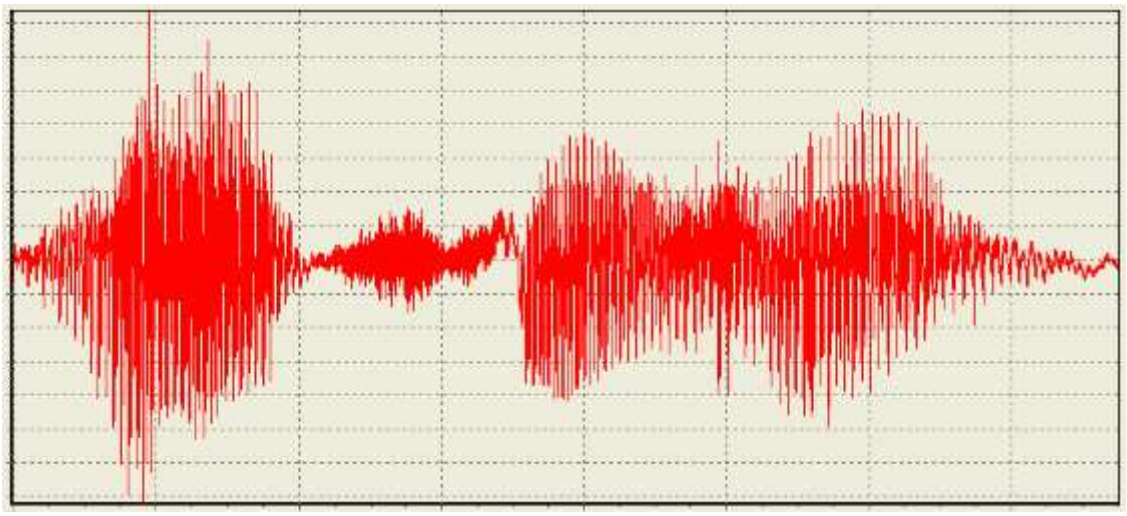

Figura 20: Áudio digital.

2.9 ARQUIVOS *WAVE AUDIO FORMAT*

O formato *WAVE audio format* (formato de áudio WAVE), ou simplesmente WAVE foi criado pela IBM em 1995 para o sistema operacional Windows. Estes arquivos possuem extensão .wav.

WAVE é uma forma de armazenamento de dados de áudio não comprimidos utilizando modulação por codificação de pulso, ou PCM (*Pulse Code Modulation*). Consiste em um formato com alta qualidade de áudio e de fácil manipulação a partir de softwares.

Um arquivo WAVE é dividido em blocos, chamados de *chunks*, como pode ser visto na Figura 21.

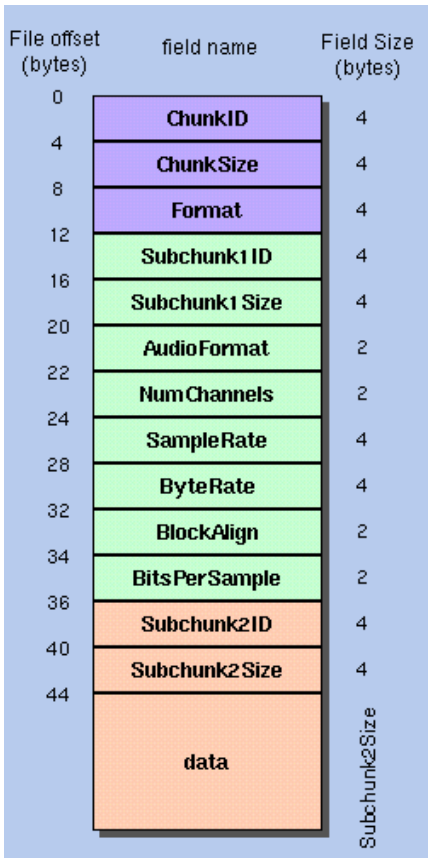

Figura 21: Estrutura dos arquivos no formato WAVE.

O primeiro bloco, em roxo, identifica o arquivo como sendo RIFF e informa o tamanho do arquivo menos os 8 bytes já utilizados. Em ChunkID está contida a palavra RIFF em ASCII e em ChunkSize está contida a quantidade de bytes do arquivos menos 8. Em Format estão contidas as letras WAVE para indicar o formato do arquivo.

O segundo bloco, em verde, descreve o formato dos dados de áudio. Em *Subchunk1ID* está contido o texto “fmt”, para sinalizar o início do bloco. Em *Subchunk1Size* está a indicação do tamanho deste bloco menos 8 bytes já utilizados, sendo o valor igual a 16 para PCM. *AudioFormat* especifica o tipo de compressão utilizada, onde ‘1’ indica PCM. *NumChannels* indica o número de canais (1 para Mono, 2 para dois canais (estéreo), etc...). *SampleRate* nos dá a infomação de taxa de amostragem do arquivo de áudio; *ByteRate* indica quantos bytes de amostras devem ser enfileirados por segundo no conversor digital/analógico para que o arquivo seja executados (SampleRate * NumChannels * BitsPerSample/8); *BlockAlign* é o número de bytes por amostra, incluindo-se todos os canais (NumChannels * BitsPerSample/8); *BitsPerSample* é a quantidade de bit por amostra.

No bloco em laranja estão contidas de fato as informações sobre o áudio digitalizado. *Subchunk2ID* indica o início do bloco dom as letras “data”; *Subchunk1Size* contém o tamanho, em bytes dos dados e *data* que contém as amostras de áudio digitalizado.

Na Figura 22 mostra um exemplo de arquivo WAVE indicando os significa cada dado contido no arquivo.

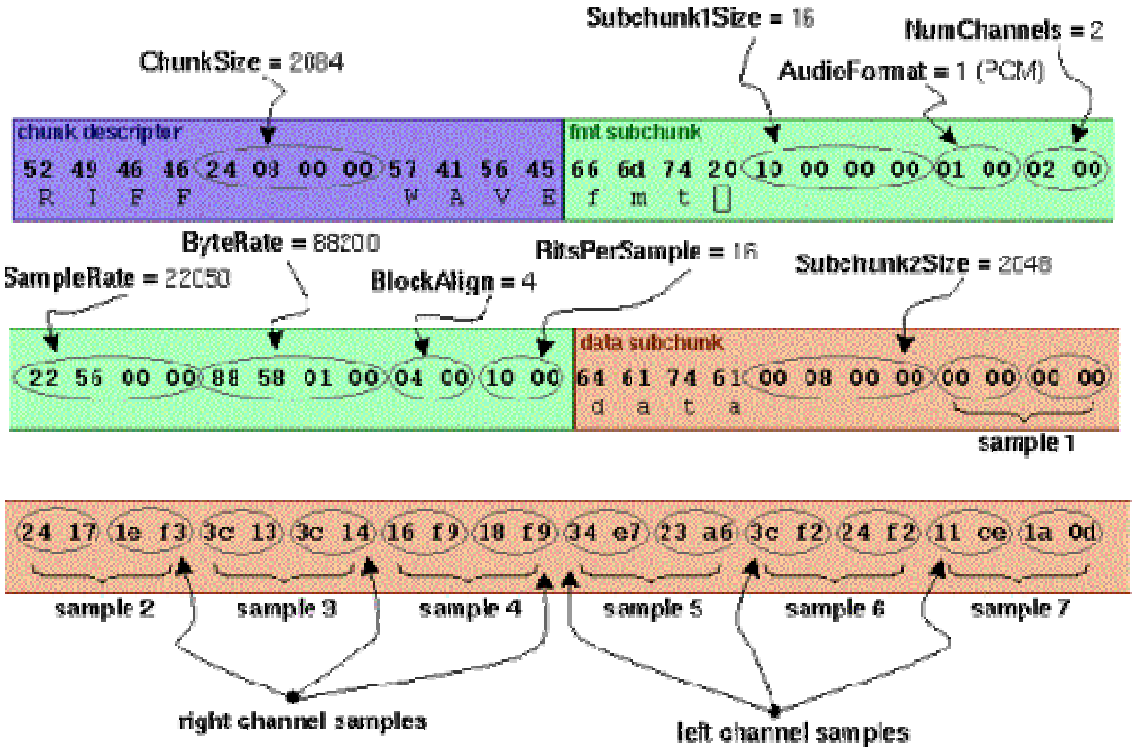

Figura 22: Exemplo de arquivo .wav (Fechine, 2006).

2.10 CORDIC

Com o objetivo de melhorar o desempenho de determinados cálculos complexos realizados pelo computador, foi criado, em 1959, os algoritmos baseados em rotação de vetores, *Coordinate Rotation Digital Computer* (CORDIC). Este algoritmo possibilita realizar cálculos de funções trigonométricas e funções hiperbólicas usando apenas soma, subtração e a função deslocamento binário por um método iterativo.

Quanto maior o número de interações realizadas, maior é a precisão do algoritmo.

O CORDIC é um conjunto de rotações de ângulos fixos calculados por uma sequência de deslocamento, com direção de rotação determinada por um esquema de

controle que garante a convergência do algoritmo. Os ângulos de rotação vão diminuindo a cada iteração, e dependendo do esquema de controle usado, podem-se ter diferentes modos de operações possibilitando a implementação de diferentes funções possibilitando implementações mais eficientes de arquiteturas e algoritmos DSP (*Digital Signal Processing*) (Costa, 2006).

A explicação do algoritmo CORDIC para se encontrar no Anexo A.

3 DESENVOLVIMENTOS E RESULTADOS

O desenvolvimento dos filtros foi realizado de forma padronizada, ou seja, as entradas e saídas das funções são da mesma forma para facilitar a utilização das mesmas.

Os filtros desenvolvidos foram os FIR com janelas Retangular, Triangular, Hamming, Hanning, Blackman e Kaiser. Para todos os filtros, foram desenvolvidas funções auxiliares para a realização do filtro, como as funções da FFT, IFFT, filtros ideais passa-baixas, passa-altas, rejeita-faixa, passa-faixa e a função CORDIC para seno e cosseno.

3.1 FAST FOURIER TRANSFORM (FFT)

A função FFT discreta encontra-se no arquivo *FFT.c* e seu cabeçalho em *FFT.h*. Têm como parâmetros de entrada: as amostras, número de amostras e um vetor no qual será retornado a transformada.

*FFT(double *arg1, int arg2, double *arg3)*

arg1 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg2 -> Variável contendo o número de amostras do vetor de entrada. Este valor deve ser potência exata de 2 para o uso correto da função. Tipo INTEIRO.

arg3 -> Vetor vazio para armazenar a saída da FFT.

Esse último vetor terá o dobro de tamanho do vetor de entrada, e será preenchido da seguinte forma:

- As primeiras N posições do vetor (0, 1,...,N-2,N-1) serão preenchidas com a parte real do sinal de saída no domínio da freqüência.
- As outras N posições (N, N+1,...,2*N-2,2*N-1) será preenchida com a parte imaginária do sinal de saída no domínio da freqüência.

3.2 INVERSE FAST FOURIER TRANSFORM (IFFT)

A função IFFT discreta encontra-se no arquivo IFFT.c e seu cabeçalho em IFFT.h. Têm como parâmetros de entrada: as amostras no domínio da freqüência, número de amostras no domínio da freqüência e um vetor no qual será retornada a transformada inversa.

*IFFT(double *arg1, int arg2, double *arg3)*

arg1 -> Vetor de entrada no domínio da freqüência. Este vetor possui o seguinte formato:

- Tipo DOUBLE.
- Tamanho 2*N, sendo N o número de amostras do sinal de entrada.
- As primeiras N posições (0, 1,...,N-2,N-1) contém a parte real do sinal de entrada no domínio da freqüência.
- As outras N posições (N, N+1,...,2*N-2,2*N-1) contém a parte imaginária do sinal de entrada no domínio da freqüência.

arg2 -> Variável contendo a metade do número de amostras do vetor de entrada. Este valor deve ser potência de 2 para o uso correto da função IFFT. Tipo INTEIRO.

arg3 -> Vetor vazio para armazenar a saída da IFFT. Tipo DOUBLE.

3.3 PASSA-BAIXAS

Calcula os coeficientes no domínio do tempo de um filtro passa-baixas ideal. A função se encontra no arquivo *filtroPBFIR.c* e seu cabeçalho em *filtroPBFIR.h*.

Têm como parâmetros de entrada: As amostras de entrada no domínio do tempo, número de amostras, tipo de filtro a ser utilizado, freqüência de amostragem do sinal, freqüência de corte desejada e o vetor vazio para conter a saída (sinal filtrado).

*filtroPBFIR(double *arg1,int arg2, int arg3, double arg4, double arg5, double arg6);*

arg1 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg2 -> Variável contendo o número de amostras do vetor de entrada. Este valor deve ser potência de 2 para posterior uso da função FFT e seu valor também corresponde a ordem do filtro a ser utilizado. Tipo INTEIRO.

arg3 -> Variável informando o tipo de filtro FIR a ser utilizado. Tipo INTEIRO.

Seu valor é definido da seguinte maneira:

arg3 = 1 -> Filtro FIR Janela Retangular.

arg3 = 2 -> Filtro FIR Janela Triangular.

arg3 = 3 -> Filtro FIR Janela de Hamming.

arg3 = 4 -> Filtro FIR Janela de Hanning.

arg3 = 5 -> Filtro FIR Janela de Blackman.

Se arg3 for algum outro valor, a função denunciará.

arg4 -> Variável contendo a freqüência de amostragem do sinal de entrada. Tipo DOUBLE.

arg5 -> Variável contendo a freqüência de corte desejada pelo filtro Passa-Baixas. Tipo DOUBLE.

arg6 -> Vetor vazio para armazenar a saída da filtro no domínio do tempo. Tipo DOUBLE.

3.4 PASSA-ALTAS

Calcula os coeficientes no domínio do tempo de um filtro passa-altas ideal. A função se encontra no arquivo filtroPAFIR.c e seu cabeçalho em filtroPAFIR.h.

Têm como parâmetros de entrada: As amostras de entrada no domínio do tempo, número de amostras, tipo de filtro a ser utilizado, freqüência de amostragem, freqüência de corte desejada e o vetor vazio para conter a saída (sinal filtrado).

*filtroPAFIR(double *arg1,int arg2, int arg3, double arg4, double arg5, double *arg6);*

arg1 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg2 -> Variável contendo o número de amostras do vetor de entrada. Este valor deve ser potência de 2 para posterior uso da função FFT e seu valor também corresponde a ordem do filtro a ser utilizado. Tipo INTEIRO.

arg3 -> Variável informando o tipo de filtro FIR a ser utilizado. Tipo INTEIRO.

Seu valor é definido da seguinte maneira:

arg3 = 1 -> Filtro FIR Janela Retangular.

arg3 = 2 -> Filtro FIR Janela Triangular.

arg3 = 3 -> Filtro FIR Janela de Hamming.

arg3 = 4 -> Filtro FIR Janela de Hanning.

arg3 = 5 -> Filtro FIR Janela de Blackman.

Se arg3 for algum outro valor, a função denunciará.

arg4 -> Variável contendo a freqüência de amostragem do sinal de entrada. Tipo DOUBLE.

arg5 -> Variável contendo a freqüência de corte desejada pelo filtro Passa-Altas. Tipo DOUBLE.

arg6 -> Vetor vazio para armazenar a saída da filtro no domínio do tempo. Tipo DOUBLE.

3.5 REJEITA-FAIXA

Calcula os coeficientes no domínio do tempo de um filtro rejeita-faixa ideal. A função se encontra no arquivo *filtroRFFIR.c* e seu cabeçalho em *filtroRFFIR.h*.

Têm como parâmetros de entrada: As amostras de entrada no domínio do tempo, número de amostras, tipo de filtro a ser utilizado, freqüência de amostragem, freqüência de corte e o vetor vazio para conter a saída (sinal filtrado).

*filtroRFFIR(double *arg1,int arg2, int arg3, double arg4, double arg5, double arg6, double *arg7);*

arg1 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg2 -> Variável contendo o número de amostras do vetor de entrada. Este valor deve ser potência de 2 para posterior uso da função FFT e seu valor também corresponde a ordem do filtro a ser utilizado. Tipo INTEIRO.

arg3 -> Variável informando o tipo de filtro FIR a ser utilizado. Tipo INTEIRO.

Seu valor é definido da seguinte maneira:

arg3 = 1 -> Filtro FIR Janela Retangular.

arg3 = 2 -> Filtro FIR Janela Triangular.

arg3 = 3 -> Filtro FIR Janela de Hamming.

arg3 = 4 -> Filtro FIR Janela de Hanning.

arg3 = 5 -> Filtro FIR Janela de Blackman.

Se arg3 for algum outro valor, a função denunciará.

arg4 -> Variável contendo a freqüência de amostragem do sinal de entrada. Tipo DOUBLE.

arg5 -> Variável contendo a primeira freqüência de corte desejada pelo filtro rejeita-faixas. Tipo DOUBLE.

arg6 -> Variável contendo a segunda freqüência de corte desejada pelo filtro rejeita-faixas. Tipo DOUBLE.

arg7 -> Vetor vazio para armazenar a saída da filtro no domínio do tempo. Tipo DOUBLE.

3.6 PASSA-FAIXA

Calcula os coeficientes no domínio do tempo de um filtro passa-faixa ideal. A função se encontra no arquivo *filtroPFFIR.c* e seu cabeçalho em *filtroPFFIR.h*.

Têm como parâmetros de entrada: As amostras de entrada no domínio do tempo, número de amostras, tipo de filtro a ser utilizado, freqüência de amostragem, freqüência de corte e o vetor vazio para conter a saída (sinal filtrado).

*filtroPFFIR(double *arg1,int arg2, int arg3, double arg4, double arg5, double arg6, double *arg7);*

arg1 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg2 -> Variável contendo o número de amostras do vetor de entrada. Este valor deve ser potência exata de 2 para posterior uso da função FFT e seu valor também corresponde a ordem do filtro a ser utilizado. Tipo INTEIRO.

arg3 -> Variável informando o tipo de filtro FIR a ser utilizado. Tipo INTEIRO.

Seu valor é definido da seguinte maneira:

arg3 = 1 -> Filtro FIR Janela Retangular.

arg3 = 2 -> Filtro FIR Janela Triangular.

arg3 = 3 -> Filtro FIR Janela de Hamming.

arg3 = 4 -> Filtro FIR Janela de Hanning.

arg3 = 5 -> Filtro FIR Janela de Blackman.

Se arg3 for algum outro valor, a função denunciará.

arg4 -> Variável contendo a freqüência de amostragem do sinal de entrada. Tipo DOUBLE.

arg5 -> Variável contendo a primeira freqüência de corte desejada pelo filtro Passa-Faixas. Tipo DOUBLE.

arg6 -> Variável contendo a segunda freqüência de corte desejada pelo filtro Passa-Faixas. Tipo DOUBLE.

arg7 -> Vetor vazio para armazenar a saída da filtro no domínio do tempo. Tipo DOUBLE.

3.7 CORDIC

Para que as funções dos filtros implementados funcionem de forma mais rápida para posterior utilização em hardware como FPGA, as funções de grande processamento, como seno e cosseno, da biblioteca math.h foram substituídas por funções do algoritmo CORDIC, não havendo a necessidade de incluir a biblioteca math.h, poupando memória e processamento. Este algoritmo armazena algumas constantes e calcula valores de algumas funções usando apenas deslocamentos, somas e subtrações.

O algoritmo CORDIC da função cosseno e seno está no arquivo cordic.c e seu cabeçalho em cordic.h.

Para o uso da função SENO -> cordic_seno(double arg1, int arg2)

arg1 -> valor do ângulo em radianos

arg2 -> número de iterações. Quanto maior o número de iterações, mais lento será o processamento, mas será maior a precisão do valor calculado.

Retorna o valor do seno desejado.

Para o uso da função COSSENO -> cordic_cosseno(double arg1, int arg2)

arg1 -> valor do angulo em radianos

arg2 -> número de iterações. Quanto maior o número de iterações, mais devagar será o processamento, contudo será maior a precisão do valor calculado.

Retorna o valor do cosseno desejado.

3.8 FUNÇÕES – JANELAS RETANGULAR, TRIANGULAR, HAMMING, HANNING E BLACKMAN

As funções que calculam os coeficientes do filtro FIR do tipo janela Retangular, Triangular, Hamming, Hanning e Blackman são filtroJretangular.c, filtroJtriangular.c, filtroJhamming.c, filtroJhanning.c e filtroJblackman.c, respectivamente. Seus cabeçalhos encontram-se nos arquivos filtroJretangular.h, filtroJtriangular.h, filtroJhamming.h, filtroJhanning.h e filtroJblackman.h.

Todas essas funções têm como parâmetros de entrada: os coeficientes do filtro ideal no tempo, as amostras de entrada no domínio do tempo, número de amostras e o vetor vazio para conter a saída (sinal filtrado). Como exemplo, mostra-se a função da janela de blackman, sendo similar o uso das demais janelas.

*filtroJblackman(double *arg1, double *arg2, int arg3, double *arg4)*

arg1 -> Vetor contendo os coeficientes do filtro ideal desejado. Tipo DOUBLE

arg2 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg3 -> Variável informando o número de amostras da entrada. Tipo INTEIRO.

arg4 -> Vetor vazio para armazenar a saída da filtro no domínio do tempo. Tipo DOUBLE.

3.9 JANELA DE KAISER

O arquivo onde está a função que calcula os coeficientes do filtro Janela de Kaiser se encontra em filtroJkaiser.c.

Como o filtro janela de Kaiser recebe parâmetros para calcular a ordem e depois que se calculam os coeficientes, foi feito outro arquivo com o objetivo de calcular somente a ordem do filtro de Kaiser. Esta função encontra-se no arquivo ordemJkaiser.c.

A função para o cálculo da ordem tem como parâmetros: a primeira freqüência da banda de passagem, a primeira freqüência da banda de rejeição, a segunda freqüência da banda de passagem, a segunda freqüência da banda de rejeição, o ganho da banda de passagem, o ganho da banda de rejeição, a freqüência de amostragem do sinal e o tipo de filtro desejado.

ordemJkaiser(double arg1, double arg2, double arg3, double arg4, double arg5, double arg6, double arg7, int arg8)

arg1 -> Valor da primeira freqüência da banda de passagem. Tipo DOUBLE
arg2 ->Valor da primeira freqüência da banda de rejeição. Tipo DOUBLE.
arg3 -> Valor da segunda freqüência da banda de passagem. Tipo DOUBLE
*arg*4->Valor da segunda freqüência da banda de rejeição. Tipo DOUBLE.
arg5 -> Ganho (ondulação) da banda de passagem. Tipo DOUBLE
arg6 -> Ganho (ondulação) da banda de rejeição. Tipo DOUBLE.
arg7 -> Freqüência de amostragem do sinal. Tipo DOUBLE
arg8 -> Tipo de filtro desejado. Tipo DOUBLE.
*arg*8= 1 -> Filtro passa-baixas.
*arg*8= 2 -> Filtro passa-altas.
*arg*8= 3 -> Filtro rejeita-faixa.
*arg*8= 4 -> Filtro passa-faixa.

A função retorna a ordem do filtro janela de Kaiser de acordo com as especificações solicitadas.

A função para o cálculo dos coeficientes do filtro janela de Kaiser tem como parâmetros: a primeira freqüência da banda de passagem, a primeira freqüência da

banda de rejeição, ganho da banda de passagem, o ganho da banda de rejeição, a freqüência de amostragem do sinal, o vetor contendo o sinal de entrada no tempo, o vetor para armazenar a saída (sinal filtrado), tipo do filtro desejado, a segunda freqüência da banda de passagem, a segunda freqüência da banda de rejeição e a ordem do filtro.

*filtroJkaiser(double arg1, double arg2, double arg3, double arg4, double arg5, double *arg6, double *arg7, int arg8, double arg9, double arg10, int arg11)*

arg1 -> Valor da primeira freqüência da banda de passagem. Tipo DOUBLE

arg2 ->Valor da primeira freqüência da banda de rejeição. Tipo DOUBLE.

arg3 -> Ganho (ondulação) da banda de passagem. Tipo DOUBLE

*arg*4-> Ganho (ondulação) da banda de rejeição. Tipo DOUBLE.

arg5 -> Freqüência de amostragem do sinal. Tipo DOUBLE

arg6 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg7 -> Vetor vazio para armazenar a saída da filtro no domínio do tempo. Tipo DOUBLE.

arg8 -> Tipo de filtro desejado. Tipo DOUBLE.

*arg*8= 1 -> Filtro passa-baixas.

*arg*8= 2 -> Filtro passa-altas.

*arg*8= 3 -> Filtro rejeita-faixa.

*arg*8= 4 -> Filtro passa-faixa.

arg9 -> Valor da segunda freqüência da banda de passagem. Tipo DOUBLE

*arg*10->Valor da segunda freqüência da banda de rejeição. Tipo DOUBLE.

*arg*11->Ordem do filtro. Tipo INTEIRO.

3.10 FIR

A função que realiza a filtragem do tipo FIR se encontra no arquivo FIR.c e seu cabeçalho no arquivo FIR.h.

Têm como parâmetros de entrada: os coeficientes do filtro FIR, o sinal de entrada, um vetor para ser a saída do programa e a quantidade de amostras do sinal.

*FIR(double *arg1, double *arg2, double *arg3, int arg4)*

arg1 -> Vetor contendo os coeficientes do filtro FIR. Tipo DOUBLE.

arg2 -> Vetor de entrada no domínio do tempo. Tipo DOUBLE.

arg3 -> Vetor vazio para armazenar o sinal filtrado. Tipo DOUBLE.

arg4 -> Variável contendo o número de amostras do vetor de entrada. Tipo INTEIRO.

Nas Figuras 23 e 24 pode-se observar um esquemático do uso das funções implementadas.

Figura 23: Esquema lógico para o uso dos filtros FIR tipo janela Retangular, Triangular, Hanning, Hamming e Blackman.

Figura 24: Esquema lógico para o uso da janela de Kaiser.

3.11 EQUALIZADOR GRÁFICO DIGITAL

Com todos os filtros desenvolvidos e testados, foi feito, em C++ Builder, um equalizador gráfico e um ambiente para usar os filtros. Este equalizador lê arquivos de áudio tipo WAV, mostra seu cabeçalho e as suas amostras em um gráfico. Depois há a opção de equalizar o arquivo ou apenas usar um filtro específico.

Na Figura 25 tem-se a tela inicial do programa feito em C++ Builder.

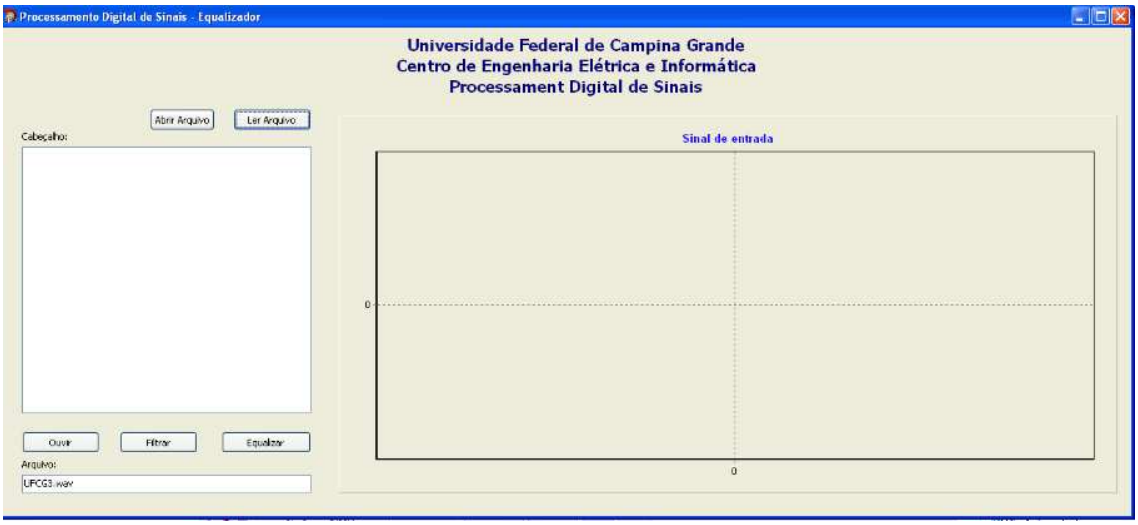

Figura 25: Tela inicial do programa

O programa possui um botão para abrir o arquivo, **Abrir Arquivo**, outro para ler o arquivo aberto, **Ler Arquivo**. Depois de ler o arquivo, pode-se visualizar o cabeçalho do arquivo WAV lido e visualizar suas amostras no gráfico do lado direito, **Sinal de entrada.** Também se pode ouvir o arquivo aberto pelo botão **Ouvir** e escolher entre apenas filtrar o sinal, **Filtrar**, escolhendo o tipo de filtro e as especificações ou usar o equalizador, **Equalizar**.

Na Figura 26 é mostrada uma visualização do programa após a leitura de um arquivo de áudio.

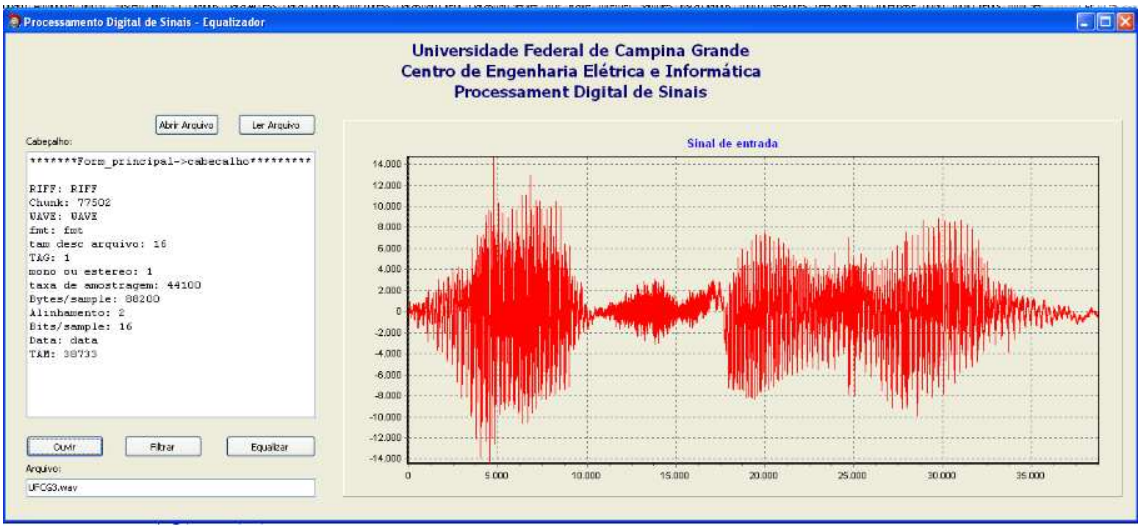

Figura 26: Programa depois da leitura de um arquivo de áudio.

3.11.1 FILTRAR

Na Figura 27 está apresentada a visualização do programa após a escolha da opção **Filtrar**. Nesta tela, tem-se a opção de escolher o tipo de filtragem que se deseja realizar em **Tipo de Filtro,** podendo escolher entre Passa-baixas, passa-altas, rejeita-faixa e passa-faixa. Também se pode escolher o algoritmo de filtragem em **Algoritmo de Filtragem,** onde nele tem os tipos de janelas desenvolvidas. Os parâmetros variam de acordo com o tipo de filtro e algoritmo escolhidos. Para o caso da Figura 20, têm-se escolhidos um filtro passa-baixas e o algoritmo Janela retangular. Pode-se perceber que aparecem os parâmetros que são a ordem desejada para o filtro e a freqüência de corte.

Após apertar o botão **Executar,** pode-se ouvir o sinal original em **Ouvir Original** e o sinal filtrado em **Ouvir Filtrado.** O volume do sinal filtrado pode ser alterado na barra **Volume.**

Os gráficos que aparecem são do sinal original **(Sinal de entrada)** o sinal original no domínio da freqüência **(Sinal de Entrada – FFT**) o sinal filtrado **(Sinal Filtrado)** o sinal filtrado no domínio da freqüência **(Sinal Filtrado – FFT)** e o gráfico da função de transferência do filtro, **(Filtro – Função de Transferência).** Pode-se, ainda, salvar o arquivo sinal filtrado apertando o botão **Salvar** e escolher o nome na caixa ao lado do botão.

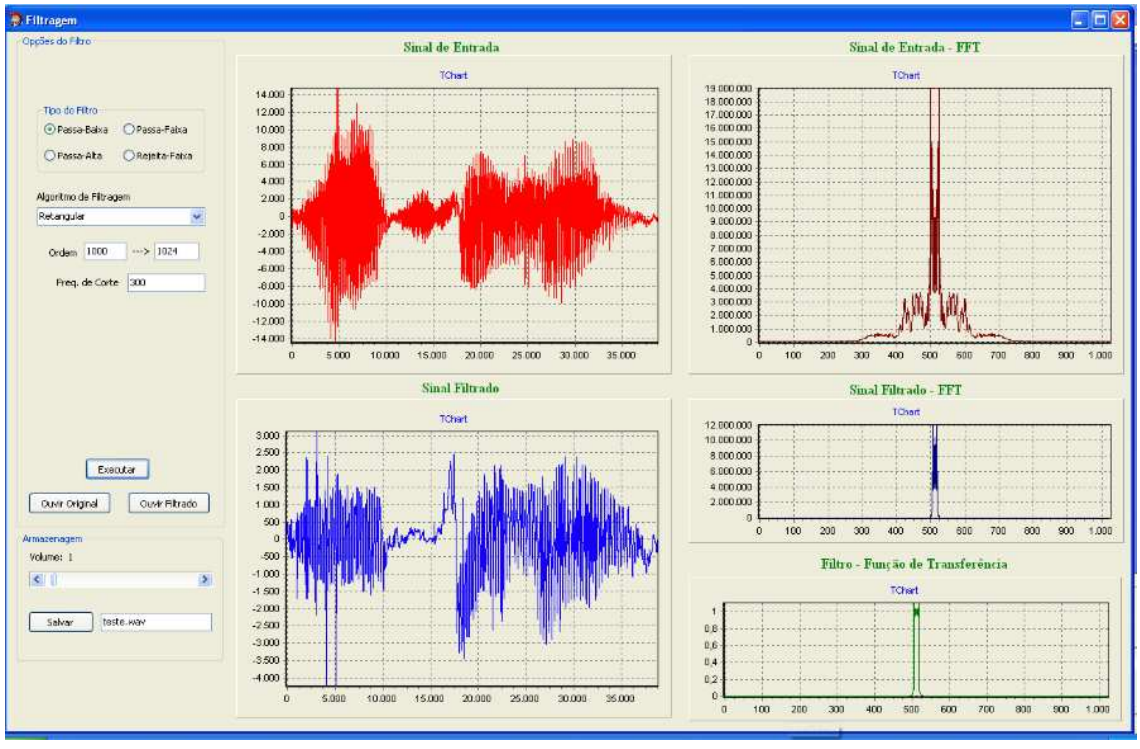

Figura 27: Sinal de áudio filtrado usando um filtro passa –baixas com freqüência de corte de 300 Hz do tipo janela Retangular de ordem 1024.

Nas figuras 27, 28, 29, 30, 31 e 32 são mostrados alguns exemplos de aplicações do software.

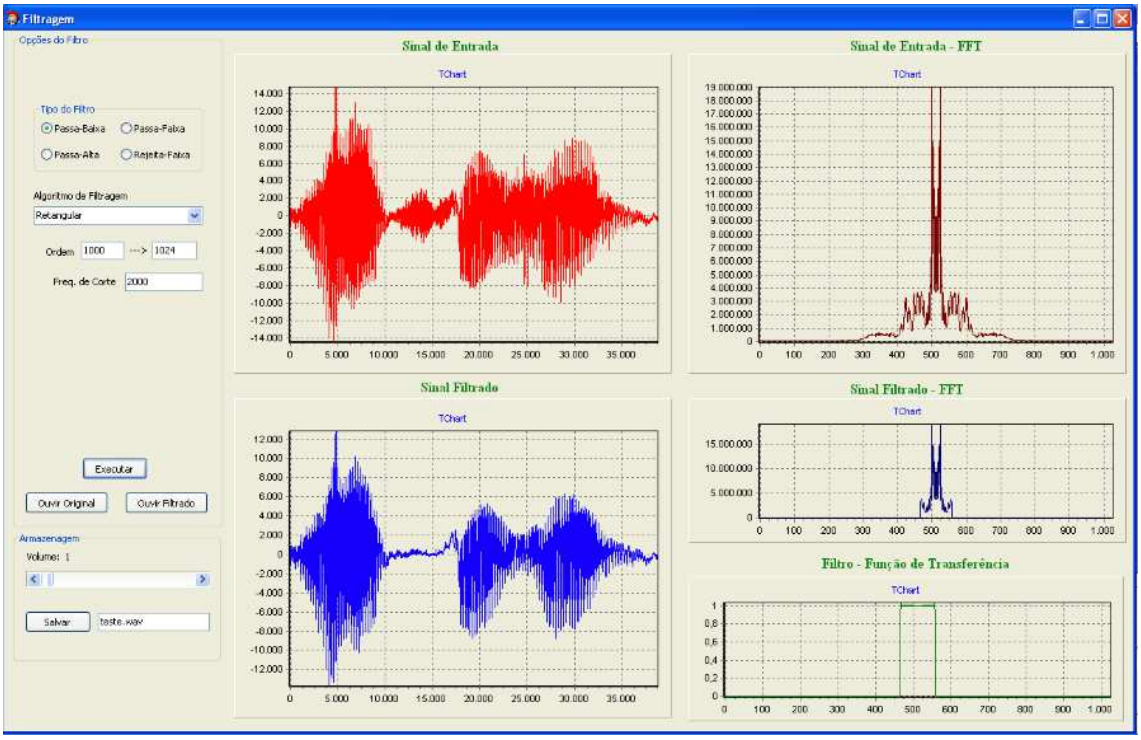

Figura 28: Sinal de áudio filtrado usando um filtro passa –baixas com freqüência de corte de 2000 Hz do tipo janela retangular de ordem 1024.

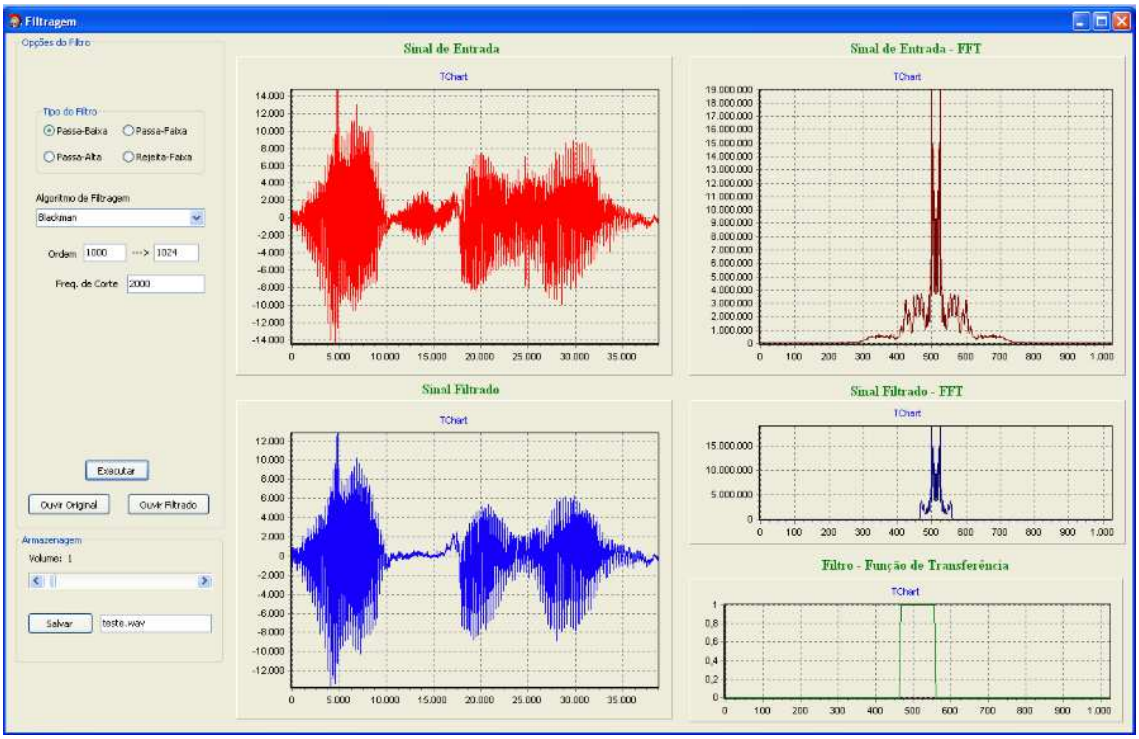

Figura 29: Sinal de áudio.filtrado usando um filtro passa –baixas com freqüência de corte de 2000 Hz do tipo janela de Blackman de ordem 1024.

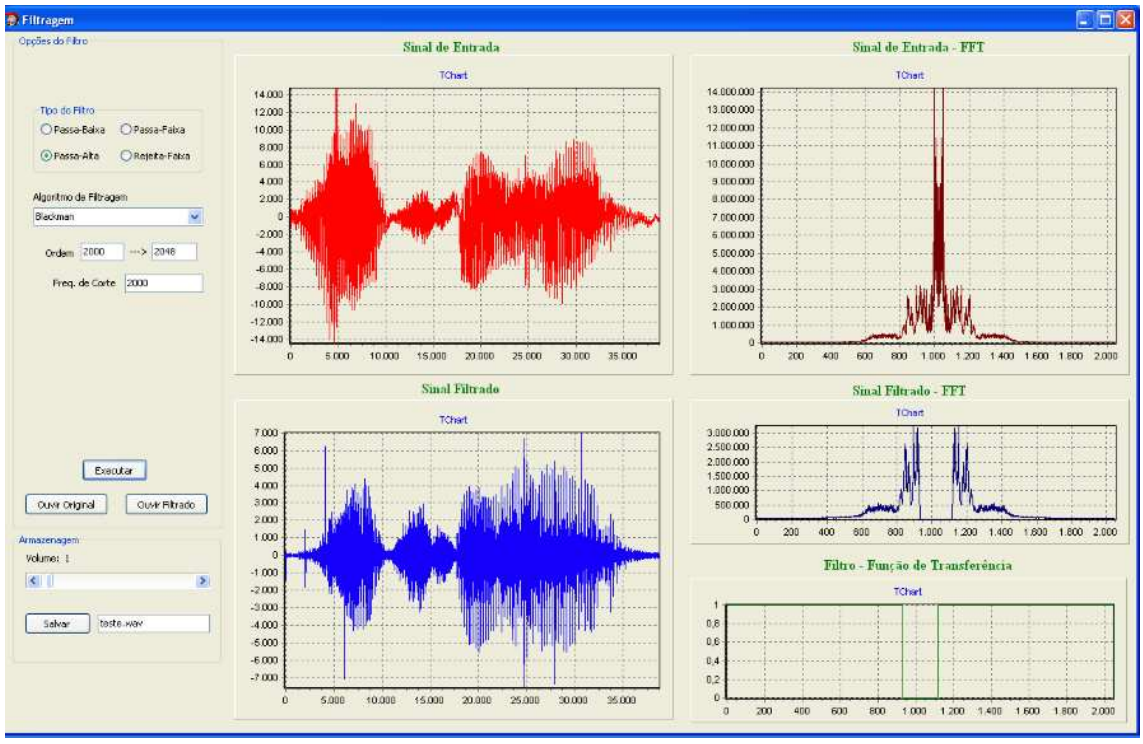

Figura 30: Sinal de áudio filtrado usando um filtro passa –altas com freqüência de corte de 2000 Hz do tipo janela de Blackman de ordem 2048.

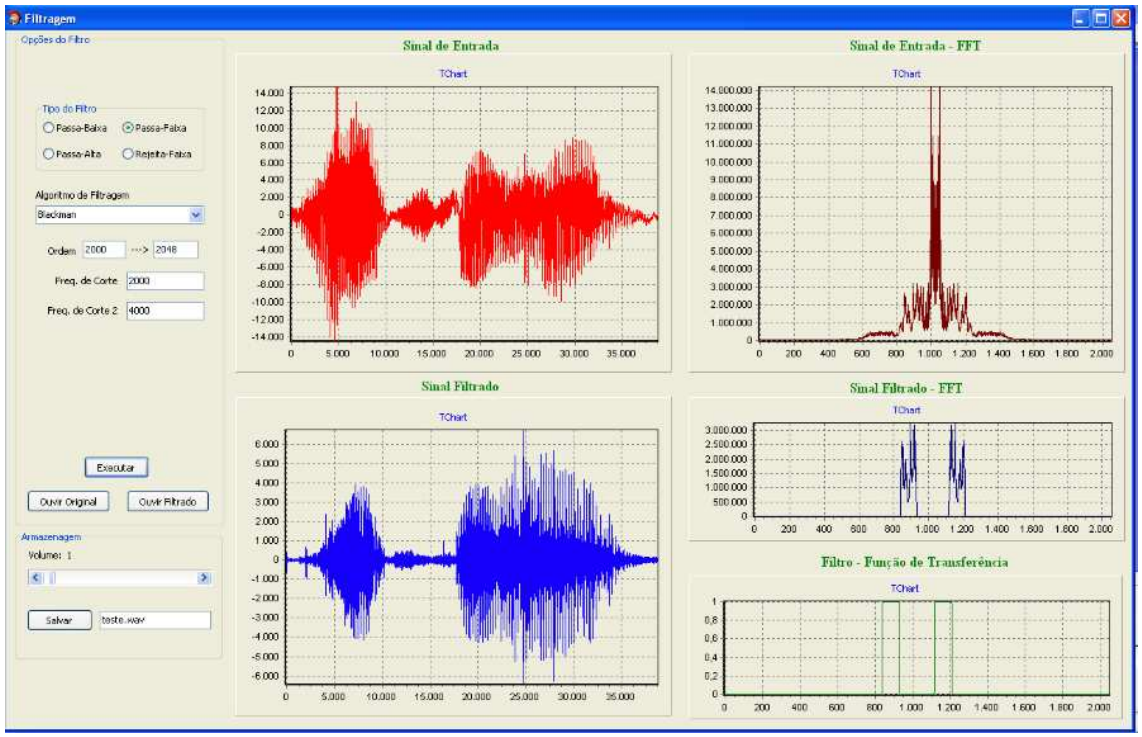

Figura 31: Sinal de áudio. filtrado usando um filtro passa –faixa com freqüências de corte de 2000 Hz e 4000 Kz do tipo janela de Blackman de ordem 2048.

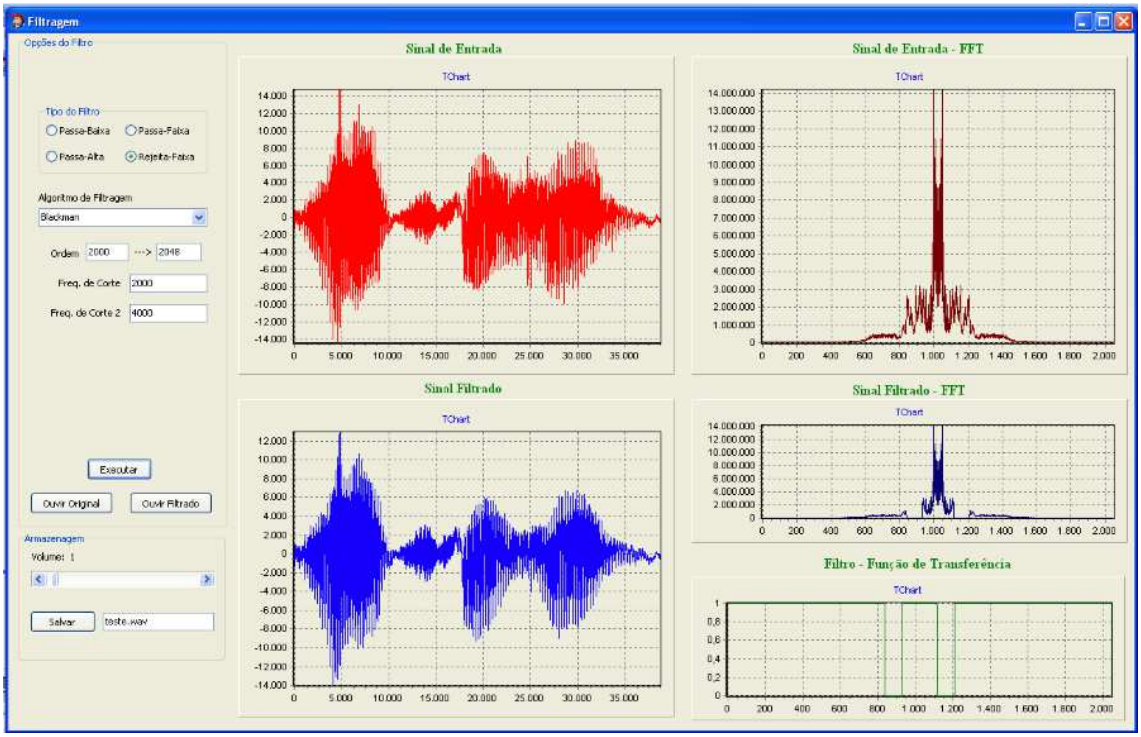

Figura 32: Sinal de áudio filtrado usando um filtro rejeita –faixa com freqüências de corte de 2000 Hz e 4000 Kz do tipo janela de Blackman de ordem 2048.

3.11.2 EQUALIZAR

Ao escolher a opção **Equalizar,** aparece a tela da Figura 33. Pode-se ver que aparece o gráfico do sinal original (**Sinal de Entrada)** o sinal original no domínio da freqüência (**FFT do Sinal de Entrada)** o sinal equalizado (**Sinal Equalizado**) o sinal equalizado no domínio da freqüência (**FFT do Sinal Equalizado)** e a função de transferência do filtro (**Função de Transferência – Banco de Filtros).**

Também tem a opção de ouvir o sinal original (**Ouvir Original)** de ouvir o sinal depois de equalizado (**Ouvir Equalizado)** de escolher o tipo de filtro que se deseja fazer a equalização, (**Tipo**) informar a ordem do filtro desejado (**Ordem)** e o botão para equalizar o sinal (**Equalizar)**.

Para equalizar também existem barras para atenuar ou dar ganho de até 10 vezes em determinadas freqüências centrais pré-determinadas.

As Figuras 34, 35, 36, 37 e 38 mostram o uso do equalizador.

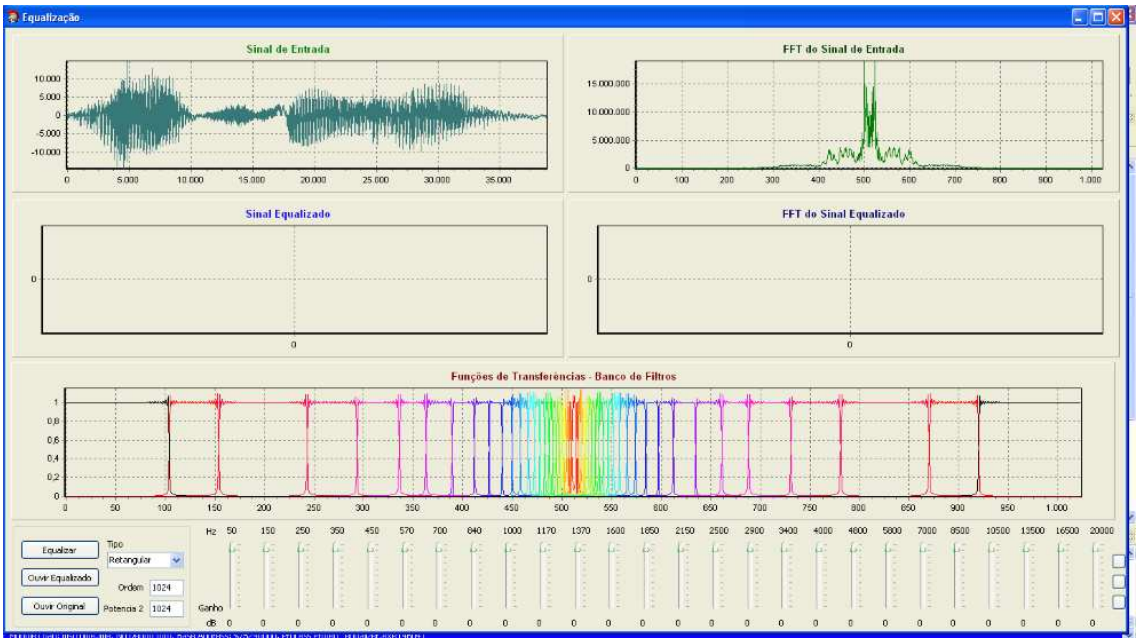

Figura 33: Equalizador digital. Função de transferência do filtro com ordem de 1024 usando janelas retangular.

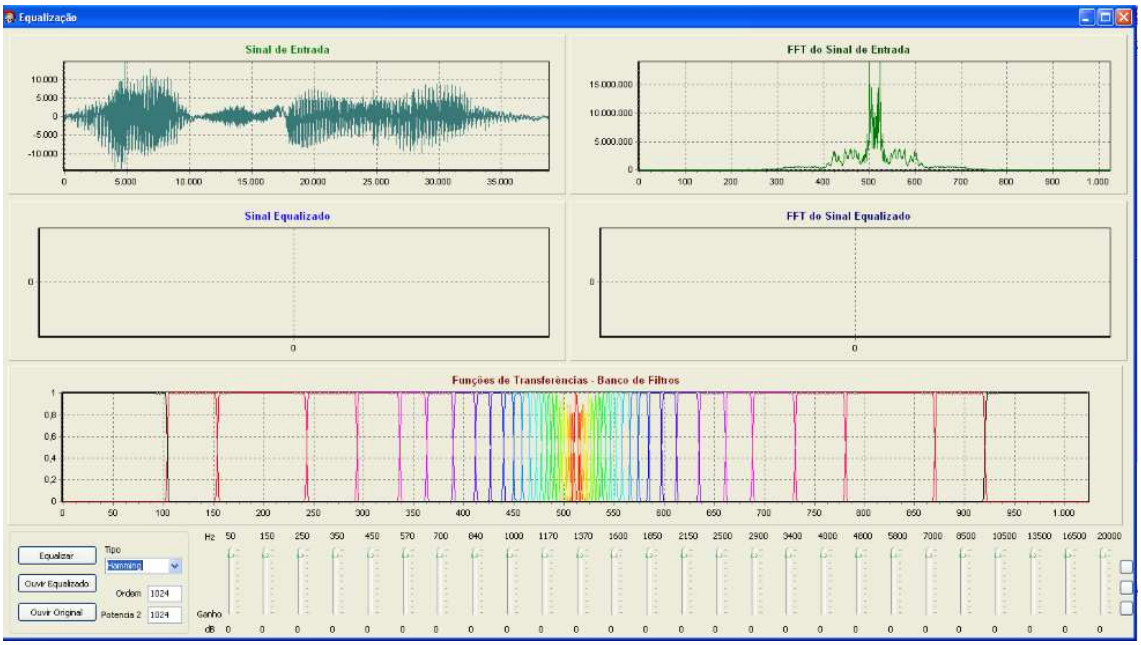

Figura 34: Equalizador digital. Função de transferência do filtro com ordem de 1024 usando janelas de Hamming.

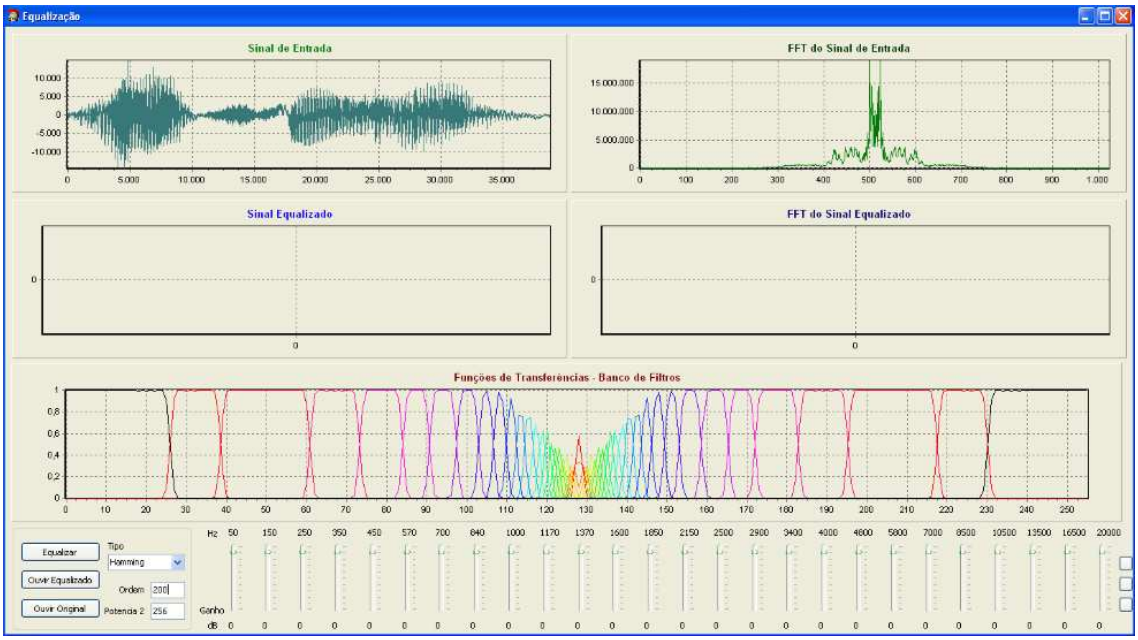

Figura 35: Equalizador digital. Função de transferência do filtro com ordem de 256 usando janelas de Hamming.

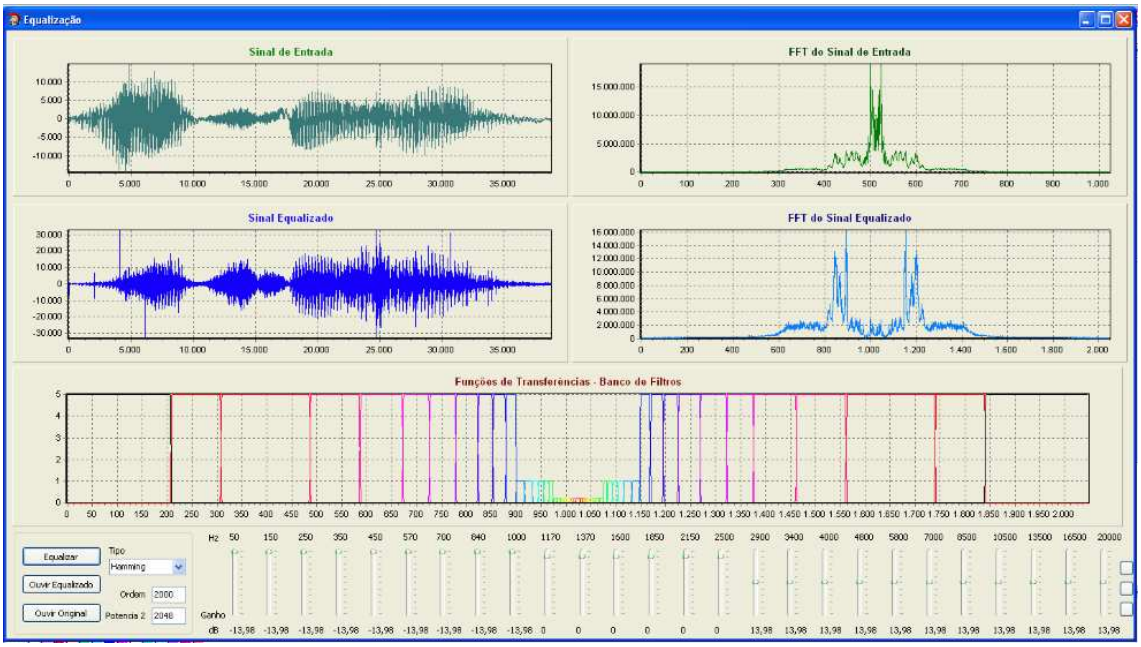

Figura 36: Equalizador digital. Execução do filtro usando janela de Hamming com ordem de 2048. Nesse exemplo, foi amplificado as altas freqüências e atenuadas as baixas, tornando o som mais grave.

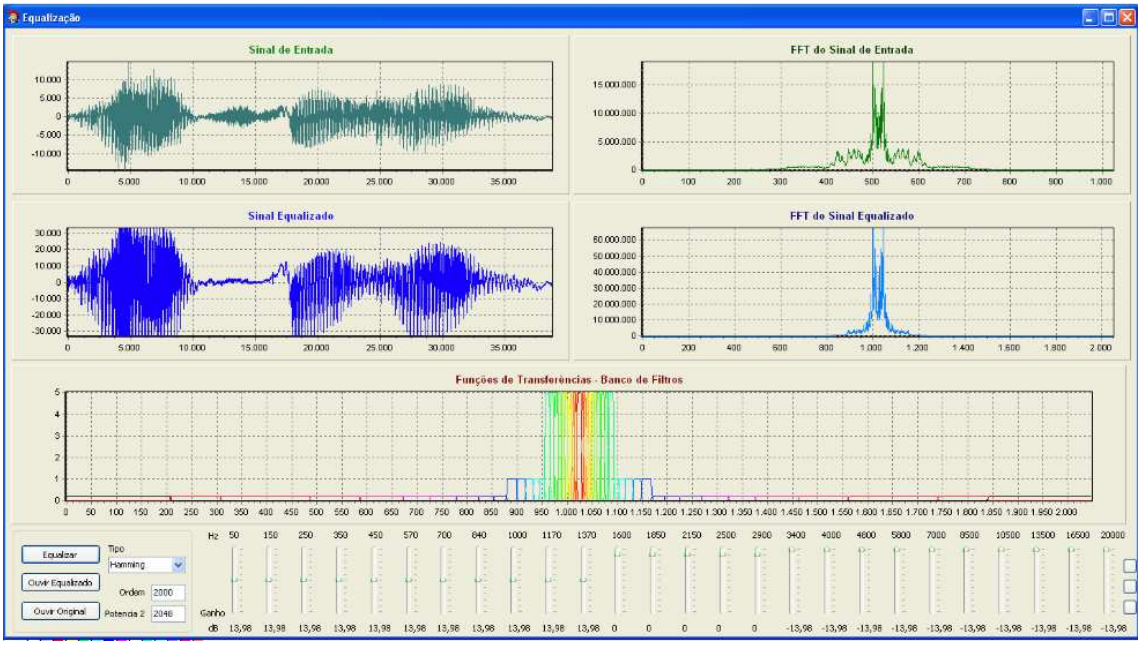

Figura 37: Equalizador digital. Execução do filtro usando janela de Hamming com ordem de 2048. Nesse exemplo, foi amplificado as baixas freqüências e atenuadas as altas, tornando o som mais agudo.

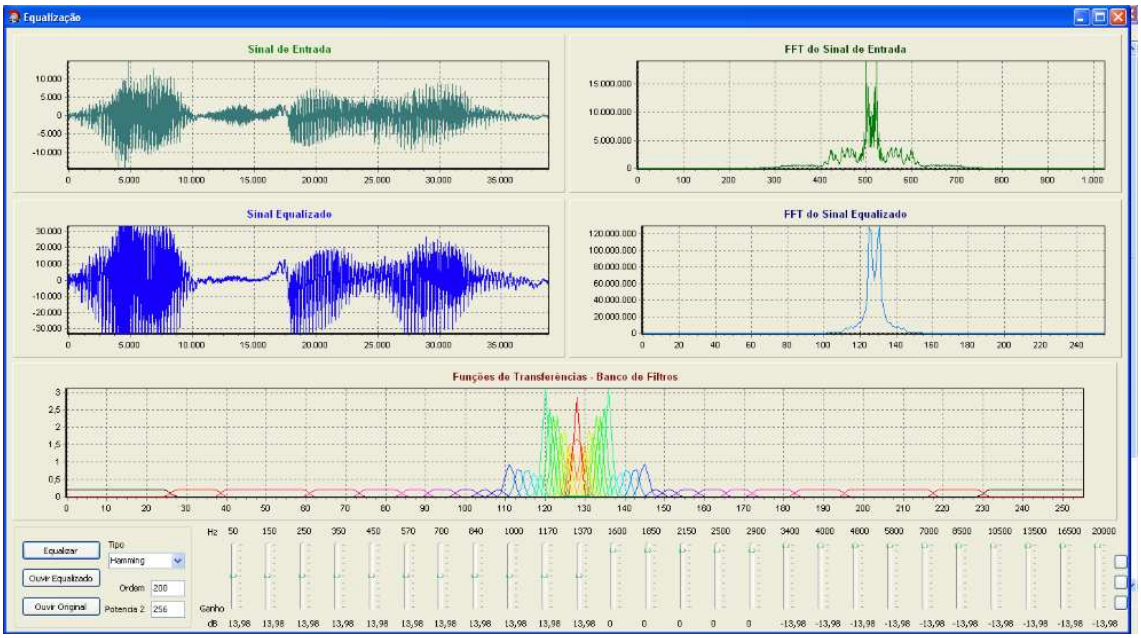

Figura 38: Equalizador digital. Execução do filtro usando janela de Hamming com ordem de 256. Nesse exemplo, foi amplificado as baixas freqüências e atenuadas as altas, tornando o som mais agudo.

4 CONCLUSÃO

O trabalho se mostrou bastante satisfatório porque vários filtros foram desenvolvidos e testados e, por fim, foi feito uma aplicação muito usada nos dias atuais que é o equalizador gráfico, comprovando a boa implementação e utilidade dos filtros.

Também se pode utilizar o software criado em C++ Builder para realizar análises de sinais no domínio da freqüência e aplicar diferentes tipos de filtros em sinais de áudio observando que filtro melhor se adéqua à determinadas aplicações. Ainda é possível usar o software como ferramenta ao ensino de processamento digital de sinais, visto que é de fácil utilização e análise de resultados no domínio do tempo e freqüência.

As funções foram desenvolvidas em ANSI C e otimizadas (uso do CORDIC) para facilitar o uso posterior em hardware dedicado à processamento de sinais e FPGA's.

BIBLIOGRAFIA

Andraka, R. (1998). A survey of CORDIC algorithms for FPGA based computer. *ACM* , 191-200.

Costa, B. C. (2006). *Implementação de Filtros de Wiener com posto de precisão reduzida.* Rio de Janeiro: Universidade Federal do Rio de Janeiro.

da Silva, M. A. (2007). *Filtros digitais aplicados em sinais de áudio.* Juiz de Fora: Universidade Federal de Juiz de Fora.

de Carvalho, J. M. (2002). *Texto didático complementar para o apoio ao ensino da disciplina Análise de Sinais e sistemas do Curso de Engenharia Elétrica da UFCG/DEE.* Campina Grande.

Diniz, P. S., da Silva, E. A., & Lima Neto, S. (2004). *Processamento Digital de Sinais - Projeto e Análise de Sistemas.* Bookman.

Fechine, J. M. (2006). *Notas de aula da disciplina de sistemas multimídia da UFCG.* Campina Grande.

Lathi, B. P. (2004). *Linear System and Signal.* Oxford University Press.

Mitra, S. K. (1998.). *Digital Signal Processing: A Computer Based Approach.* McGraw-Hill.

Oppenheim, A. V., Willsky, A. S., & Young, I. T. (1983). *Signals and Systems* (2ª ed.). New Jersey, Brasil: Prentice - Hall.

Pierce, A. D. (1989). Acoustics: An Introduction to Its Physical Principles and Applications. *Acoustical Society of Amer* .

ANEXO A – CORDIC

Volder desenvolveu o algoritmo CORDIC baseado na forma geral da matriz de rotação de Givens, que rotacional um vetor *V = (x; y)* por um ângulo no plano Euclidiano:

$$\begin{cases} x' = xcos(\alpha) - ysen(\alpha) \\ y' = ycos(\alpha) + xsen(\alpha) \end{cases} \quad (35)$$

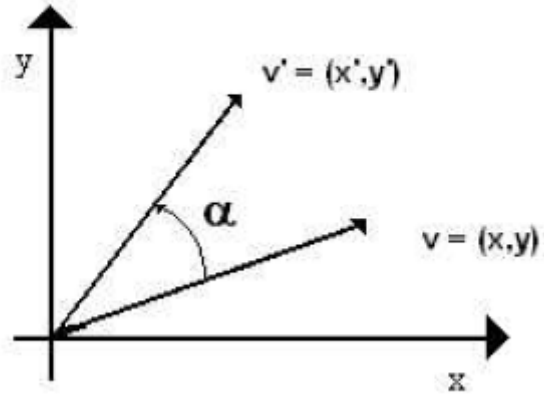

Figura 39: Rotação de um vetor v por um ângulo α.

Organizando a equação acima, temos:

$$\begin{cases} x' = cos(\alpha).[x - ytan(\alpha)] \\ y' = cos(\alpha)[y + xtan(\alpha)] \end{cases} \quad (36)$$

Se os ângulos de rotação forem limitados a $tan(\alpha) = \pm 2^{-i}$, a multiplicação do termo tangente é reduzida a uma operação simples de deslocamento (Andraka, 1998). A direção da rotação pode ser tanto no sentido horário quanto no sentido anti-horário, pois $cos(\alpha) = cos(-\alpha)$. A Equação (36) pode ser expressa como:

$$\begin{cases} x' = K_i.[x_i - y_i.\mu_i 2^{-i}] \\ y' = K_i.[y_i + x_i.\mu_i 2^{-i}] \end{cases} \quad (37)$$

em que:

$$K_i = \cos\left(tan^{-1}\left(2^{-i}\right)\right) = \frac{1}{\sqrt{1+2^{-2i}}} \quad (38)$$

$$\mu_i = \pm 1 \quad (39)$$

Se a variável K_i for removida das equações iterativas acima, pode-se rotacionar vetores somente com operações de soma e de deslocamento. A remoção desta variável, chamada de fator de correção de escala, pode ocorrer através de um produto realizado após cada iteração ou pode ocorrer ao final de todas as iterações, como um processo de ganho do algoritmo.

Os ângulos de rotação são determinados pelo valor de arcotangente, que variam de acordo com uma sequência não decrescente. A tabela abaixo exemplifica passo a passo os ângulos de rotação fixos para a sequência dos números naturais.

Tabela 4: Valores dos ângulos fixos para a sequência dos números naturais

$\tan(\alpha_i)$	α_i	$\cos(\alpha_i)$
$\tan(\alpha_1) = \frac{1}{1}$	$\alpha_1 = 45°$	$\cos(\alpha_1) = 0.7071$
$\tan(\alpha_2) = \frac{1}{2}$	$\alpha_2 = 26.5650°$	$\cos(\alpha_2) = 0.8944$
$\tan(\alpha_3) = \frac{1}{4}$	$\alpha_3 = 14.0362°$	$\cos(\alpha_3) = 0.9701$
$\tan(\alpha_4) = \frac{1}{8}$	$\alpha_4 = 7.1250°$	$\cos(\alpha_4) = 0.9922$
$\tan(\alpha_5) = \frac{1}{16}$	$\alpha_5 = 3.5763°$	$\cos(\alpha_5) = 0.9980$
$\tan(\alpha_6) = \frac{1}{32}$	$\alpha_6 = 1.7899°$	$\cos(\alpha_6) = 0.9995$
...	...	...

Uma terceira variável z é necessária para acumular os ângulos rotacionados, como exposto na equação a seguir:

$$z_{i+1} = z_i - \mu_i . tan^{-1}(2^{-i}) \quad (40)$$

Poe exemplo, se substituirmos os valores de tan(α_i) e cos(α_i) na Equação (36), e iniciarmos as variáveis x_0, y_0 e z_0 com os valores $\frac{1}{K_i}$, 0 e um ângulo qualquer θ respectivamente, temos ao final de i iterações o cosseno de θ armazenado na variável x_i. Se aplicarmos o método uma vez, temos a precisão de um bit, e assim por diante.

Juntando as Equações (35), (36) e (37), Volder descreve uma iteração CORDIC composta exclusivamente com operações de soma/subtração e deslocamento da seguinte maneira:

$$\begin{cases} x_{i+1} = x_i - m.\mu_i.y_i.\delta_{m,i} \\ y_{i+1} = y_i + \mu_i.x_i.\delta_{m,i} \\ z_{i+1} = z_i - \mu_i.\alpha_{m,i} \end{cases} \quad (40)$$

A cada iteração um vetor $v_i=(x_i,y_i)^T$ será transformado linearmente num vetor $v_{i+1}=(x_{i+1},y_{i+1})^T$, e a variável z assume o somatório dos ângulos de rotação $\alpha_{m,i}$. A Variável m pertencente ao conjunto {1,0,-1} determina o sistema de coordenadas, enquanto a variável μ_i, podendo assumir valores {1,-1}, irá determinar a direção da rotação, desempenhando um papel fundamental para a convergência do algoritmo. A variável $\delta_{m,i}$ é definida como $\delta_{m,i} = d^{-s_{m,i}}$, sendo d a base do sistema numérico e $s_{m,i}$ uma sequência de deslocamento de números inteiros, geralmente não decrescente (Costa, 2006).