UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELÉTRICA
CURSO DE ENGENHARIA ELÉTRICA

BRUNA MACHADO MULINARI

FILTROS DIGITAIS EM ÁUDIO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2017

BRUNA MACHADO MULINARI

FILTROS DIGITAIS EM ÁUDIO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Engenharia Elétrica do Departamento Acadêmico de Elétrica – DAELE – da Universidade Tecnológica Federal do Paraná – UTFPR, Câmpus Pato Branco, como requisito parcial para obtenção do título de Engenheira Eletricista.

Orientador: Prof. Dr. Fábio Luiz Bertotti

PATO BRANCO

2017

TERMO DE APROVAÇÃO

O trabalho de Conclusão de Curso intitulado **FILTROS DIGITAIS EM ÁUDIO,** da aluna **Bruna Machado Mulinari** foi considerado **APROVADO** de acordo com a ata da banca examinadora **N° 164** de **2017**.

Fizeram parte da banca os professores:

Prof. Dr. Fábio Luiz Bertotti

Prof. Dr. Giovanni Alfredo Guarneri

Prof. Dr. Cesar Rafael Claure Torrico

DEDICATÓRIA

Dedico este trabalho a minha mãe, Adiagre, a qual tornou meu sonho de formação o seu. Sem os seus esforços e incentivos essa realização não seria possível.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me proteger e dar forças durante esta jornada.

A minha família por caminharem sempre ao meu lado. Aos presentes que sempre me encorajaram e aos já ausentes que de alguma forma me iluminaram durante essa etapa.

Aos meus amigos agradeço por toda colaboração e apoio principalmente durante os momentos de insegurança. Em especial a meu colega Julio Cesar Leme por toda paciência e colaboração na parte de implementação do trabalho.

A todos os professores e servidores que de alguma forma contribuíram para minha formação. Em especial ao meu professor orientador, Fábio Luiz Bertotti, por toda atenção e dedicação prestadas durante o desenvolvimento desse trabalho.

EPÍGRAFE

“Não há demanda por mulheres na engenharia, assim como existe para mulheres na medicina; mas há sempre uma demanda para qualquer um que pode fazer um bom trabalho”

(Edith Clarke)

RESUMO

MULINARI, Bruna Machado. Filtros Digitais em Áudio. 2017. 129 f. Monografia (Trabalho de Conclusão de Curso) – Curso de Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

O seguinte trabalho apresenta os procedimentos realizados para implementação de filtros digitais em um dispositivo microcontrolado com objetivo de realizar a filtragem de um sinal de áudio. Para isso são abordados os principais conceitos a respeito das ondas sonoras e os principais problemas encontrados em sistemas de comunicação devido a sinais indesejáveis. Além disso, o trabalho aborda o conceito da aplicação de filtragem na área musical por meio da equalização de som. Também, são apresentados os conceitos de processamento digital de sinais e o processo de filtragem digital de sinais que sustentam o trabalho. Com base nos conceitos apresentados foram projetados filtros digitais do tipo FIR e IIR para simulações no MatLab e implementação no microcontrolador STM32F407VG. As simulações apresentaram as respostas da filtragem de dois sinais de entrada diferentes, um sinal de áudio e um sinal multisenoidal, para cada um dos filtros projetados, permitindo verificar a atuação e a seletividade de cada tipologia. Com os resultados das simulações foi possível realizar comparações quanto a ordem utilizada e a seletividade obtida, para os filtros FIR e IIR. Embora obteve-se somente os resultados práticos da implementação dos filtros FIR, foi possível evidenciar que as respostas destes filtros estão condizentes com as simulações.

Palavras-chave: Ondas Sonoras. Equalização. Processamento Digital de Sinais. Filtros Digitais.

ABSTRACT

MULINARI, Bruna Machado. Digital Filters in Audio. 2017. 129 f. Monografia (Trabalho de Conclusão de Curso) – Curso de Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

The following work presents the performed procedures to the project and implementation of digital filters in one microcontroller device with the objective of filtering an audio signal. The main concepts about sound waves and the main problems encountered in communication systems due to undesirable signals are discussed. In addition, the paper approaches the concept of the application of filtering in the musical area through the equalization of sound. Also presenting the concepts of digital signal processing and the process of digital filtering of signals that sustain the work. Based on the concepts, the FIR and IIR digital filters were designed for simulations in Matlab and implemented in the STM32F407VG microcontroller. The simulations presented the filtering responses of two different input signals, an audio signal and a multisenoidal signal, for each of the projected filters, allowing to verify the performance and selectivity of each typology. In addition, with the results of the simulations it was possible to make comparisons, as to the order used and the selectivity obtained, of the FIR and IIR filters. Although only the practical results of the implementation of the FIR filters were obtained, it was possible to show that the responses of these filters are consistent with the simulations.

Keywords: Sound Waves. Equalization. Digital Signal Processing. Digital Filters.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

SNR	*Signal-to-Noise Ratio* (Relação Sinal-Ruído)
ROC	*Region of Convergence* (Região de Convergência
ADC	A*nalog-to-Digital Converter* (Conversor Analógico para Digital)
DAC	*Digital-to-Analog Converter* (Conversor Digital para Analógico)
FIR	*Finite Impulse Response* (Resposta Finita ao Impulso)
IIR	*Infinite Impulse Response* (Reposta Infinita ao Impulso)
JTAG	*Joint Test Access Group*
WIC	*Wake-up Interrupt Controller (*Controlador de Interrupção de Despertar)
FPU	*Float Point Unit* (Unidade de Ponto Flutuante)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
RTOS	*Real Time Operating System* (Sistema Operacional de Tempo Real)
SPI	*Serial Peripheral Interface* (Interface Periférica Serial)
DMA	*Direct Memory Access* (Acesso Direto à Memória)
I2S	*Inter-IC Sound*
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
FAT	*File Allocation Table* (Tabela de Alocação de Arquivos)

LISTA DE SÍMBOLOS

ω	Frequência angular
c_k	Coeficientes da série de Fourier
I	Intensidade da onda sonora
P	Potência da onda sonora
S	Área
β	Nível de intensidade
T_s	Período de amostragem
ω_s	Frequência angular de amostragem
f_s	Frequência de amostragem
f_{max}	Frequência máxima de um sinal
C	Capacidade máxima de transmissão
W	Banda passante
n_b	Número de *bits*
V_{max}	Tensão máxima
V_{min}	Tensão mínima
N	Níveis do quantizador
L	Comprimento do filtro
ω_c	Frequência angular de corte
d	Coeficientes do filtro FIR
a,**b**	Coeficientes do filtro IIR

SUMÁRIO

1 INTRODUÇÃO

Embora não de forma tão intuitiva, os sinais estão presentes em várias aplicações do cotidiano do ser humano, seja em forma de tensão, corrente, vídeo, áudio, entre outros. De acordo com Roberts (2009), uma das definições para um sinal é qualquer fenômeno físico variante no tempo capaz de transmitir informações.

Sendo o objetivo a transmissão de informações, é imprescindível que o sinal chegue ao receptor da forma mais fiel possível. Dessa maneira, a forma como se processa o sinal é fundamental para que não ocorra perda ou deterioração das informações. Assim, o processamento de sinais tem como objetivo representar, transformar e manipular as informações contidas em um sinal sem corrompê-las (OPPENHEIM; SCHAFER, 1998).

A comunicação entre seres humanos por exemplo, é baseada em sinais de áudio. Nesse tipo de sinal, a forma como o mesmo é processado pode implicar em problemas de ruídos e interferências. Tanto o ruído quanto a interferência são sinais indesejáveis capazes de degradar a clareza da informação no sinal original, em que dependendo das intensidades, pode até torná-lo incompreensível. Quanto maior a relação sinal ruído ou SNR, do inglês *Signal-to-Noise Ratio*, maior é a relação entre a potência do sinal e a potência do ruído, permitindo definir a qualidade do sinal recebido (ROBERTS, 2009).

O processamento de sinais pode ser aplicado em sons, imagens, séries temporais, sinais de telecomunicações e entre outros para, por exemplo, eliminação de sinais indesejáveis. Além disso, pode ser aplicado para seleção de uma faixa de frequência, maximizando ou atenuando componentes de frequência do sinal, processo esse nomeado como equalização do som. A equalização do som é amplamente utilizada no meio musical, atenuando ou maximizando as frequências graves, médias e agudas do sinal de áudio, a fim de tornar o som mais agradável (MARQUES, 2014).

A solução para problemas com ruídos e interferências em processamento de sinais, além da implementação de equalizadores de som, baseia-se na utilização de técnicas de filtragem de sinais. A filtragem de sinais é realizada por um sistema denominado filtro. Um filtro de forma geral, apresenta a capacidade de selecionar uma

parte especifica de frequência e descartar o restante, permitindo separar, recuperar, atenuar e maximizar sinais (MARQUES, 2014).

No processamento de sinais, os filtros podem ser implementados de forma analógica ou de forma digital. Graças ao avanço tecnológico dos conversores analógicos-digitais (A/D), que realizam a amostragem e quantização de sinais, e do desenvolvimento dos processadores, os filtros digitais tornaram-se ótimos aliados na área de processamento de sinais e apresentam algumas vantagens consideráveis perante aos filtros analógicos, principalmente nos quesitos flexibilidade, precisão e confiabilidade (PROAKIS; MANOLAKIS, 2007). Os filtros digitais podem ser representados conforme sua resposta a uma entrada do tipo impulso e classificados de duas formas: quando a resposta do filtro é finita, ou seja, apresenta uma resposta nula após um tempo finito, denomina-se filtros FIR (do inglês *Finite Impulse Response*) e quando a resposta é infinita, isto é, mesmo quando o sinal de entrada cessar o filtro pode apresentar uma resposta não-nula, denomina-se filtros IIR (do inglês *Infinite Impulse Response*) (NALON, 2009).

A decisão de implementar um filtro FIR ou IIR depende da sua aplicação, em que as características de cada tipo devem ser levadas em consideração. Como o filtro FIR apresenta memória finita, sempre será estável e qualquer transitório tem duração limitada. Entretanto, esse tipo de filtro necessita de vários coeficientes, exigindo mais memória, maior número de operações e, consequentemente, mais tempo de processamento por um processador. Por outro lado, os filtros IIR apresentam característica de corte de frequência acentuada com um filtro de ordem relativamente baixa, o que reduz a complexidade e o tempo de processamento (KUO; GAN, 2005).

Visto a importância da etapa de filtragem no processamento digital de sinais este trabalho apresenta o projeto e implementação, em um *kit* de desenvolvimento microcontrolado, de filtros digitais FIR e IIR nas configurações passa-baixa, passa-alta, passa-banda e rejeita-banda.

Inicialmente é apresentado a fundamentação teórica de suporte ao projeto de filtros digitais. Na sequência apresenta-se a descrição do projeto de filtros digitais FIR e dos filtros IIR para cada tipo de configuração. Além do projeto, utilizou-se o *software* Matlab® para realizar as simulações da filtragem de um sinal de áudio e de um sinal multisenoidal utilizando os diferentes tipos de filtros. Apesar das inúmeras aplicações possíveis de filtros digitais, a escolha do sinal de áudio deve-se ao fato

desse tipo de sinal, como tratado anteriormente, ser um dos mais perceptíveis para os humanos. Utilizando o mesmo sinal multisenoidal das simulações, são implementados os filtros no *kit* de desenvolvimento microcontrolado, para que com os resultados obtidos da filtragem seja possível realizar a comparação dos dados ideais, apresentados na fundamentação teórica, simulados, obtidos utilizando o *software* Matlab® e práticos adquiridos. Vale ressaltar que este trabalho apresenta uma das possíveis aplicações para o protótipo. Contudo, adequando as características desejadas dos filtros, é possível projetar sistemas que realizam a filtragem de outros tipos de sinais ou sinais da mesma natureza, porém que apresentam outras especificações de aplicação.

1.1 OBJETIVO GERAL

Realizar um estudo sobre filtros digitais, assim como projetar, simular e implementar esses filtros, utilizando-os em aplicações com sinais de áudio a partir de uma ferramenta de simulação e de um *kit* de desenvolvimento de sistemas microcontrolados.

1.2 OBJETIVOS ESPECÍFICOS

Para atender o objetivo geral, foram estipulados o seguintes objetivos específicos:

- Revisão teórica sobre filtros digitais: realizada em livros, dissertações e teses, para aprofundar o conhecimento teórico no assunto;
- Estudo sobre o microcontrolador escolhido: destacando as principais características e descrevendo sobre seus componentes e periféricos;
- Projeto para implementação de filtros digitais: estabelecendo os parâmetros e coeficientes para cada tipo de filtro;
- Simulação dos filtros digitais: em que se possa avaliar a efetividade dos filtros projetados;
- Implementação de filtros digitais no microcontrolador: a partir dos filtros projetados e simulados;

- Validação dos resultados obtidos: comparando os dados obtidos na simulação e na implementação com os resultados ideais apresentados na teoria.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho encontra-se estruturado de modo que no Capítulo 2 é abordada a fundamentação teórica necessária para o desenvolvimento do projeto. Esse capítulo apresenta conceitos relativos a ondas sonoras e audição, ruídos, interferências, seleção de frequência e equalização de som, processamento digital de sinais e filtros digitais. No Capítulo 3 são abordados os procedimentos para desenvolvimento do trabalho, como projeto dos filtros, métodos utilizados no *software* e no *hardware* do protótipo, além de apresentar os critérios dos materiais escolhidos. No Capítulo 4 são apresentados os resultados obtidos a partir da simulação dos filtros e o protótipo implementado em *hardware*. Por fim, o Capítulo 5 aborda as conclusões a respeito do trabalho desenvolvido, expondo também possíveis melhorias e possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresenta a fundamentação teórica utilizada para desenvolvimento do trabalho. Primeiramente é apresentado o estudo de sinais de áudio e de algumas modificações que podem ocorrer nesses tipos de sinais. Na sequência é apresentado a teoria de processamento digital de sinais, abordando a respeito de cada etapa do processo. Para finalizar o capítulo aborda-se os filtros digitais, apresentando as definições e modo de projetá-los.

2.1 ONDAS SONORAS E AUDIÇÃO

Muitas espécies de animais utilizam a audição para se localizar e diferenciar sons, seja para reconhecer espécies ou identificar o perigo. Além disso, para alguns animais a audição é utilizada para aprender e reproduzir sons, como os seres humanos que utilizam o reconhecimento sonoro como aprendizado e o reproduzem para se comunicar entre si (KANDEL, *et al.*, 2014).

O som pode ser definido como a sensação percebida pelo cérebro das ondas sonoras que chegam ao aparelho auditivo. Essa definição se deve aos animais apresentarem capacidades auditivas diferentes, ou seja, dependendo das características da onda sonora ela pode ser identificada ou não por algum animal. Dessa forma, considera-se como som as ondas sonoras que encontram-se dentro da capacidade auditiva de cada espécie (GARCIA, 2002) (SILVERTHORN, 2010).

As ondas sonoras são produzidas pelas vibrações das moléculas do meio material de propagação sincronizadas com as vibrações de um corpo imerso nesse meio, como a vibração das cordas de um instrumento ou a vibração das cordas vocais (GARCIA, 2002) (HEWITT, 2015). As vibrações das moléculas geram variações de pressão e densidade do meio que são transmitidas. Essas variações são geradas quando as moléculas do meio são forçadas a sair de suas posições de equilíbrio tendendo a pressionar as moléculas próximas, formando uma região de alta pressão chamada de *região de compressão*. As posições de onde as moléculas partiram encontram-se em baixa pressão, pois existe uma distância maior entre as partículas, sendo chamada de *região de rarefação* (RAPOSO, 2016) (FILHO, 2003).

Para a propagação das ondas sonoras é necessário um meio material deformável ou elástico, de modo que as moléculas possam sair e voltar para as posições de equilíbrio na mesma direção que a vibração do corpo foi produzida. Isso gera uma série de compressões e rarefações no meio (RUNSTEIN; HUBER, 2011). Essas perturbações também são chamadas de *movimento oscilatório da partícula* e podem ser representadas graficamente por uma onda periódica que apresenta características específicas, como comprimento de onda, período, frequência e amplitude. Tendo conhecimento das características e da velocidade da onda sonora é possível distingui-la de outras ondas (HEWITT, 2015). Desta forma, as ondas sonoras podem ser caracterizadas como ondas mecânicas e longitudinais (RAY D. KENT, 2015) (BERTOLDO, 2013).

A velocidade de propagação da onda sonora depende da natureza, da temperatura e da pressão do meio. Desse modo, a velocidade reflete as características mecânicas do meio (GARCIA, 2002). A maioria dos sons audíveis são ondas sonoras propagadas pelo ar, porém, se a mesma onda sonora se propagasse em um meio líquido ou sólido a velocidade dessa onda seria diferente (HEWITT, 2015).

Para os humanos, o sentido de ouvir um som depende da capacidade do ouvido em realizar múltiplas transduções. Esse processo de transdução ocorre pois vibrações são geradas no meio ao redor da orelha, devido à energia da onda sonora, que são reconhecidas por receptores sensoriais de audição e transformadas em sinais elétricos para que possam ser transmitidas pelo sistema nervoso (COSTANZO, 2007) (SILVERTHORN, 2010). A interpretação que o cérebro faz das características e qualidades da onda sonora, que chegam até ele por meio de sinais elétricos, permite que os humanos diferenciem e classifiquem cada som (SILVERTHORN, 2010).

A interpretação do som pode ser diferente do som produzido, pois em sua propagação a onda interage com as partículas do meio e/ou com outras ondas sonoras e pode sofrer algumas mudanças de suas características. Essas mudanças ocorrem devido a fenômenos conhecidos como: reflexão, refração, ressonância e interferência sonora (GARCIA, 2002). Além disso, a interpretação pode ser distorcida ou totalmente prejudicada devido a interação da onda sonora com sinais aleatórios indesejáveis, denominados ruídos (ROBERTS, 2010).

2.1.1 Características da onda sonora

As partículas do meio material em que uma onda sonora foi produzida realizam movimentos oscilatórios. Esses movimentos oscilatórios das partículas podem ser representados por uma onda senoidal como a apresentada na Figura 1 (RAPOSO, 2016). A onda senoidal pura é uma onda periódica, na qual a partícula apresenta a mesma velocidade após intervalos de tempos iguais, também chamada de movimento harmônico simples (GARCIA, 2002).

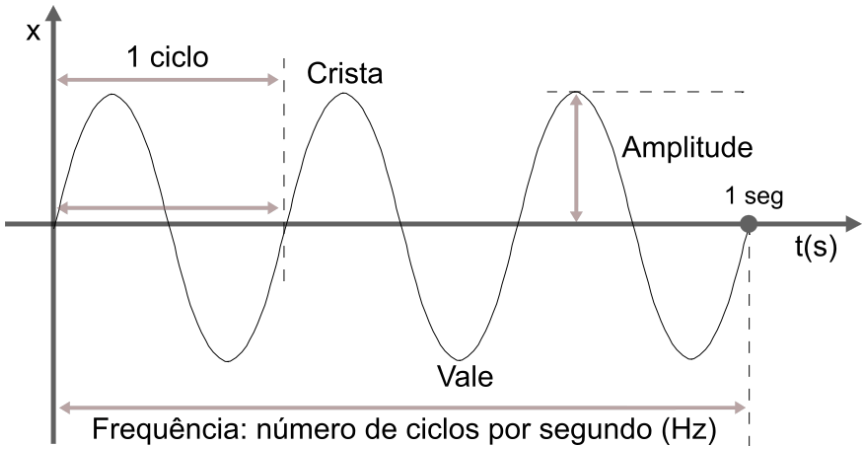


Figura 1 - Representação de uma onda senoidal.
Fonte: Adaptado de Raposo (2016).

Uma onda senoidal periódica apresenta algumas características importantes que permitem distinguir uma forma de onda das outras, como velocidade da onda, comprimento de onda, período, frequência e amplitude (RUNSTEIN; HUBER, 2011).

O comprimento de uma onda é dado pela distância entre dois pontos consecutivos com a mesma fase (RAPOSO, 2016). O período de uma onda é o intervalo de tempo que uma partícula leva para realizar um ciclo, ou seja, o movimento completo de ida e o de volta a posição inicial. A frequência da onda, expressa em Hertz (Hz), é a quantidade de ciclos realizados dentro de um segundo (GARCIA, 2002). A amplitude, no caso de uma onda sonora, é o valor do maior deslocamento das partículas em relação a posição de equilíbrio. Essa característica da onda relaciona o valor com a amplitude de variação da pressão do meio de propagação, ou

intensidade sonora, sendo expressa em decibéis (dB) (COSTANZO, 2007) (RAPOSO, 2016).

2.1.2 Espectro de frequências dos sons

Um sinal representado no domínio do tempo pode ser representado no domínio da frequência e vice-versa por intermédio das transformadas de Fourier. O teorema de Fourier estabelece que todo sinal no domínio do tempo é composto por uma soma de senos e cossenos. Por esse motivo, o sinal pode ser decomposto e representado no domínio da frequência pelas componentes de frequência de cada senóide e suas amplitudes (NALON, 2009) (PROAKIS; MANOLAKIS, 2007).

A Figura 2 apresenta uma exemplificação de um sinal conforme o teorema de Fourier. Segundo o teorema, a onda da Figura 2(a) pode ser decomposta e representada pelas ondas mostradas na Figura 2(b), sendo representada no domínio da frequência pelo gráfico da Figura 2(c).

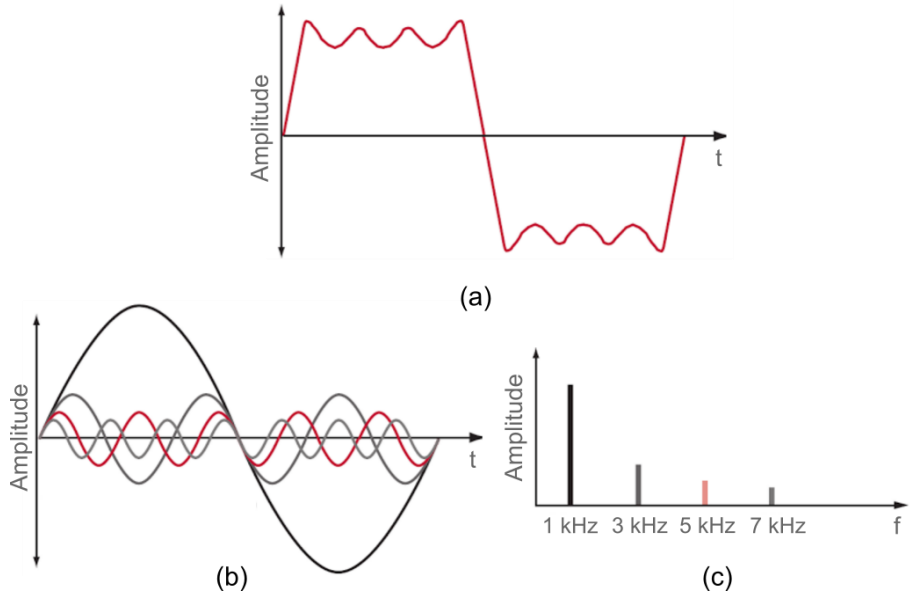


Figura 2 - Representação da teoria de Fourier: (a) onda original; (b) onda decomposta e (c) representação no domínio da frequência.
Fonte: Adaptado de Schuler (2013).

A maioria dos sons presentes na natureza não são senóides puras e sim um resultado da sobreposição de sons senoidais, chamadas de parciais. Essa sobreposição de senóides pode ser verificada por meio da análise espectral de uma

onda sonora. Desta forma, pode-se classificar o espectro de um som em dois tipos: espectro harmônico e espectro inarmônico (FILHO, 2003). Quando as frequências dos parciais são múltiplos inteiros da frequência fundamental, tipicamente a frequência mais grave do sinal, denomina-se como espectro harmônico. Nesse tipo de espectro, também chamado de som tônico ou som composto, há a percepção de altura definida devido a vibrações periódicas ou quase-periódicas (FILHO, 2003). O espectro inarmônico é dado quando as frequências dos parciais são múltiplos fracionários da frequência mais grave. Nesse tipo de espectro, também denominado como som complexo, a percepção de altura é indefinida devido a vibrações aperiódicas (ZUBEN, 2004).

O som harmônico que vibra de maneira periódica contém o espectro harmônico discreto, pois apresenta energia em uma banda de frequências em determinadas frequências. Enquanto que o som inarmônico contém um espectro contínuo, pois possui energia em toda gama de frequências em uma determinada banda (FILHO, 2003).

2.1.3 Qualidades fisiológicas do som

A psicoacústica é o ramo de estudo que analisa a percepção do som, ou seja, os impactos psicológicos e fisiológicos das ondas sonoras no sistema nervoso. Essa área acredita que há algumas características da onda sonora, que processadas pelo cérebro, ajudam a distinguir as informações recebidas (FONSECA, 2012). De acordo com Garcia (2002), as qualidades fisiológicas do som que ajudam a distinguir um som de outro e são dadas pela altura, intensidade e timbre.

2.1.3.1 Altura

A altura é a qualidade que permite classificar os sons conforme a sua frequência audível. Em média, a audição humana percebe sons com frequências na faixa de 20 Hz a 20 kHz, apesar de variar dependo do indivíduo e/ou sua idade (GARCIA, 2002).

Embora não haja uma classificação rígida, de acordo com determinadas frequências os sons podem ser classificados em sons graves, médios e agudos. Em

média, os sons graves possuem componentes de frequências entre 20 Hz a 300 Hz, enquanto que os sons médios compreendem sinais na faixa de 300 Hz a 2 kHz e os sons agudos abrangem sinais com frequências de 2 kHz até 20 kHz (SENAI. SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL, 2014).

Os sons ainda podem ser classificados a partir dos limites do espectro sonoro audível ao ser humano (20 Hz a 20 kHz) em infrassom e ultrassom. Os infrassons são classificados como os sons que apresentam frequências inferiores a 20 Hz. Sons com frequências maiores que 20 kHz são denominadas de ultrassons (LEVITOV; DALLAS; SLONIM, 2013).

2.1.3.2 Intensidade

A qualidade denominada como intensidade permite classificar os sons em fracos e fortes. Essa característica relaciona-se com a capacidade de transmissão de energia da onda sonora, pois determina a quantidade de potência fornecida a uma unidade de superfície (LEVITOV; DALLAS; SLONIM, 2013). De acordo com Garcia (2002), a intensidade pode ser calculada por:

$$I = \frac{P}{S}, \tag{1}$$

em que a intensidade da onda (I) é dada pela potência (P) transmitida em uma área (S) perpendicular à direção do fluxo sonoro, sendo medida em Watts/m². O som mais intenso tolerado ao ouvido humano e o mais fraco são de 1 W/m² e de 10^{-12} W/m², respectivamente (GARCIA, 2002).

A intensidade também pode ser expressa na escala de decibéis, que são a medida relativa em uma escala logarítmica. Dessa maneira, pode-se definir o nível de intensidade β, ou apenas intensidade, comparando a intensidade I do som com a intensidade padrão I_o (FILHO, 2003):

$$\beta = 10\log_{10}\left(\frac{I}{I_o}\right) \tag{2}$$

Utiliza-se como referência o limiar da audição como 0 dB. Na escala logarítmica, 20 dB correspondem a um aumento de dez vezes a intensidade de referência. Intensidades maiores que 100 dB podem causar danos a audição, dependendo também do tempo de exposição (COSTANZO, 2007).

2.1.3.3 Timbre

A qualidade que permite distinguir fontes sonoras, mesmo que apresentem altura e intensidade semelhantes, é o timbre. O que permite diferenciar uma fonte sonora de outra é que cada fonte emite uma onda sonora com frequência fundamental e com múltiplos dessa frequência, denominado de sons harmônicos (SENAI. SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL, 2014). A intensidade do som dos harmônicos é diferente e, além disso, apresentam frequências com o dobro, o triplo ou o quadruplo da frequência fundamental. A superposição da frequência fundamental com o conjunto de harmônicos permite que se possa identificar cada fonte (SIQUEIRA, 2008).

2.1.4 Modificações das ondas sonoras

Uma onda sonora que se propaga por um meio pode sofrer algumas mudanças de características, como alteração da intensidade ou do timbre. Essas mudanças se devem a fenômenos denominados como reflexão, refração, ressonância e interferência sonora (GARCIA, 2002).

2.1.4.1 Reflexão

Quando uma onda sonora que se propaga por um meio atinge uma superfície ou um objeto, parte da energia da onda é refletida e a outra parte da energia continua a ser transmitida. Esse fenômeno é chamado de reflexão e ocorre pois a onda sonora se depara com uma interface entre dois meios com propriedades físicas distintas (RAPOSO, 2016). A parte da onda sonora refletida é mais fraca comparada a onda direta, uma vez que parte da energia sonora é absorvida pela interface que a reflete (FILHO, 2003). A proporção de energia da onda que será refletida e que será transmitida é dada pela diferença das impedâncias acústicas entre os meios e o ângulo de incidência da onda sonora inicial. Quanto maior for a diferença de impedâncias, maior será a reflexão da interface (LEVITOV; DALLAS; SLONIM, 2013).

O fenômeno da reflexão de uma onda sonora pode gerar dois efeitos sonoros conhecidos: o eco e a reverberação. O eco é causado pela própria reflexão

da onda sonora, enquanto que a reverberação ocorre quando a onda sonora sofre múltiplas reflexões e persiste mesmo após a fonte parar de vibrar (HEWITT, 2015).

2.1.4.2 Refração

Quando uma onda sonora atinge uma interface com diferentes velocidades de propagação sonora, ocorre uma alteração de direção, ou curva, da onda transmitida. Esse fenômeno é chamado de refração e pode ocorrer em meios sólidos, líquidos e gasosos (HEWITT, 2015).

Um exemplo do fenômeno de refração ocorre quando a onda sonora se propaga pelo ar. As camadas mais próximas da terra apresentam uma temperatura maior que as camadas mais distantes. A velocidade de propagação da onda sonora é menor em uma camada fria comparada a velocidade de propagação em uma camada quente. Quando a onda sonora atinge a interface entre as duas camadas acontece a refração da onda (FILHO, 2003).

2.1.4.3 Ressonância

A ressonância com o som ocorre quando uma onda sonora atinge um objeto elástico e esse começa a vibrar espontaneamente (HEWITT, 2015). Esse fenômeno se deve ao fato de a onda sonora apresentar uma frequência igual a frequência natural de vibração do objeto (FILHO, 2003). Quando ocorre a ressonância em um objeto, ele passa a vibrar com uma amplitude relativamente maior. Esse objeto passa a se chamar ressonador, uma vez que também passa a emitir som (GARCIA, 2002).

Como a ressonância implica em um aumento da amplitude, isso provoca alguma perda de energia visto a necessidade de transformação da energia de vibração em outra forma de energia. A perda de energia depende das características constituintes do sistema ressoante e influenciam na intensidade e qualidade do som emitido (FILHO, 2003).

2.1.4.4 Interferência

Em um mesmo ambiente, diversas ondas sonoras podem ser produzidas, por diferentes fontes sonoras, e serem propagadas no meio. Quando essas ondas se encontram, ocorre uma sobreposição delas no espaço, fenômeno conhecido como interferência sonora (BAUER; WESTFALL; DIAS, 2013).

O fenômeno ondulatório conhecido como interferência pode ser classificado em relação a fase das ondas em dois tipos: interferência construtiva e interferência destrutiva (LEVITOV; DALLAS; SLONIM, 2013).

A interferência construtiva acontece quando as cristas de onda dos sinais coincidem em um dado ponto no espaço, ou seja, as ondas se encontram em fase umas com as outras. Nesse ponto, as ondas sonoras se combinam e resultam em um aumento da amplitude (FILHO, 2003).

Quando as ondas sonoras se encontram em um dado ponto no espaço em oposição, fora de fase, ocorre o fenômeno chamado interferência destrutiva, provocando uma diminuição da amplitude (LEVITOV; DALLAS; SLONIM, 2013). Se as ondas sonoras forem ondas senoidais puras e apresentarem a mesma frequência e amplitude, na ocorrência de uma interferência destrutiva nenhum som seria captado por qualquer dispositivo auditivo no determinado local. Entretanto, na realidade, as ondas sonoras são mais complexas que as ondas senoidais puras e a interferência destrutiva não ocorre de maneira tão drástica a ponto de anular totalmente o som (FILHO, 2003). Ainda assim, a interferência destrutiva é muito utilizada para a criação de equipamentos antirruído. Um exemplo de equipamento antirruído que utiliza a interferência destrutiva são os fones utilizados por pilotos de avião, onde um microfone capta o som exterior e o envia a um *microchip*. Nesse *microchip* é gerado um padrão ondulatório que é a imagem especular das ondas sonoras originais. Esse padrão ondulatório é enviado aos fones de ouvido e ocorre a interferência destrutiva, atuando como proteção auricular ao usuário (HEWITT, 2015).

2.1.4.5 Ruídos

Alguns sons produzidos por fontes naturais ou artificiais provocam sensações auditivas desagradáveis, como chiado ou rangido, e quando interagem

com uma onda sonora podem ocasionar degeneração da informação contida no sinal (HAYKIN; MOHER, 2008). Esses sons, denominados ruídos, são sinais indesejáveis que não apresentam um padrão de frequências (ROBERTS, 2010).

Em aplicações de áudio, pode-se denominar diferentes tipos de ruídos, sendo os mais conhecidos o ruído branco e o ruído rosa (BRAGA, 2014). Denomina-se ruído branco aquele que apresenta uma gama grande de frequências com mesma intensidade, isto é, apresenta densidade de energia espectral constante (FILHO, 2003). O ruído rosa, quando comparado ao ruído branco, apresenta uma diminuição gradual da amplitude de cada componente da faixa de frequência relativas ao som grave e ao agudo, isto é, a intensidade diminui com o aumento da frequência (BRAGA, 2014).

Em um sistema de comunicação, a medida da qualidade de um sinal recebido é dado pela SNR. A SNR é obtido pela relação entre a potência do sinal e a potência do ruído. Quanto menor for o valor dessa relação maior é a degradação que o ruído causa na clareza da informação do sinal. A Figura 3 apresenta um sinal com diferentes níveis de ruído e seus respectivos valores de SNR (ROBERTS, 2010).

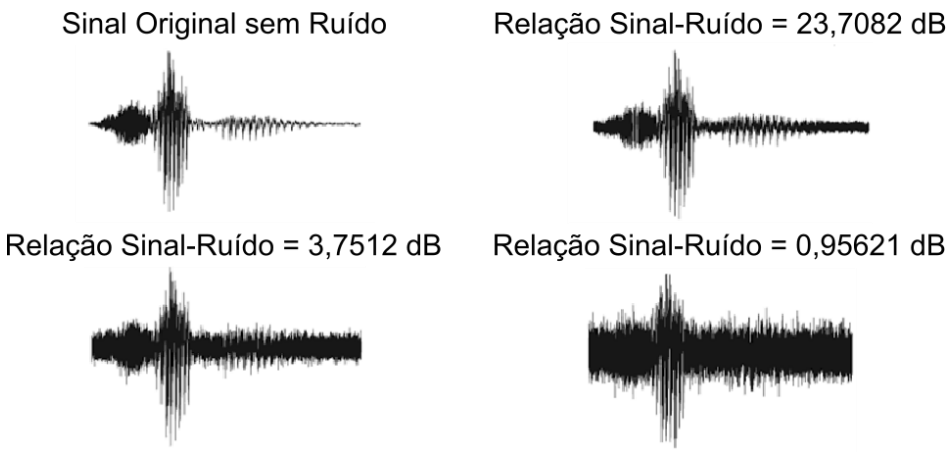


Figura 3 - Sinal com diferentes níveis de ruídos.
Fonte: Adaptado de Roberts (2010).

2.1.5 Seleção de Frequência e Equalização de Som

Certas aplicações exigem que um sinal passe por uma seleção de frequências, como em sistemas de comunicação nos quais selecionam o sinal pretendido dentre vários outros e minimizam os problemas com ruídos e interferências

(SADIKU; ALEXANDER; MUSA, 2014). Dessa maneira, o sinal desejado apresenta uma largura de frequência limitada, além de uma qualidade melhor do que antes da seleção (LATHI, 2006).

Além da seleção de frequência, algumas aplicações requerem maximizar ou atenuar componentes de frequência do sinal. Esse processo denominado como equalização de som é muito utilizado em produções musicais para tornar o som mais agradável em cada ambiente, realizando a atenuação ou maximização de frequências graves, médias e agudas do sinal de áudio (MARQUES, 2014).

A seleção de frequências de um sinal é feita por meio do processo de filtragem, no qual ocorre a rejeição ou passagem de faixas de frequências, também chamadas de bandas (NALON, 2009). Ainda, por meio dos filtros é possível, alterando seus ganhos, atenuar ou adicionar energia numa determinada faixa de frequência para realização da equalização do sinal (MARQUES, 2014).

2.2 TRANSFORMADA DE FOURIER

Jean Baptiste Fourier desenvolveu um teorema no qual todo sinal contínuo no tempo pode ser decomposto em um determinado número de senóides puras. Conforme comentado anteriormente, esse teorema é uma ferramenta utilizada para realizar a transformação de sinais no domínio do tempo para o domínio da frequência, permitindo uma análise diferenciada das informações e a realização de operações com maior facilidade (SCHULER, 2013).

O processo de obtenção de cada senóide, ou componente de frequência, é chamado de análise. Para sinais não-periódicos ilimitados no tempo, o método de análise utilizado é a transformada de Fourier. Sinais periódicos ou sinais limitados no tempo podem ser analisados por meio das séries de Fourier (NALON, 2009).

Processadores digitais utilizam sinais discretizados obtidos a partir de sinais contínuos no tempo. Dessa maneira, para implementar um analisador de espectro é necessário que ele se encontre no mesmo domínio. Por esse motivo, a transformada e a série de Fourier podem ser representadas em tempo discreto (SCHULER, 2013).

A transformada discreta de Fourier $X(\omega)$ de um sinal discreto $x[n]$ é definida como (PROAKIS; MANOLAKIS, 2007):

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n].e^{-j\omega n}, \quad (3)$$

em que $e^{-j\omega n}$ = cos ω_n – j.sen ω_n e ω é a frequência analisada. Em alguns casos, X(ω) é definido como o espectro do sinal x[n] (NALON, 2009).

Para representar sinais discretos periódicos utiliza-se a série discreta de Fourier, que é definida como (PROAKIS; MANOLAKIS, 2007):

$$x[n] = \sum_{k=0}^{N-1} c_k.e^{\frac{j2\pi kn}{N}}, \quad (4)$$

em que N representa o período do sinal discreto x[n] e c_k são os coeficientes da série de Fourier e representam a amplitude e a fase de cada uma das exponenciais complexas (NALON, 2009). Esses coeficientes são obtidos por meio de (PROAKIS; MANOLAKIS, 2007):

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n].e^{\frac{-j2\pi kn}{N}} \quad (5)$$

2.2.1 Fenômeno de Gibbs

Ao realizar a aproximação por séries trigonométricas de Fourier truncadas de um sinal periódico que apresente descontinuidades é possível observar ondulações e um sobre-sinal nas proximidades da descontinuidade no pico mais próximo da oscilação. Esse comportamento pode ser observado na Figura 3, em que mostra uma onda quadrada x(t) e sua aproximação por séries de Fourier (LATHI, 2006).O comportamento apresentado na Figura 3 é denominado como fenômeno de Gibbs, em homenagem ao matemático chamado Josiah Gibbs que foi o primeiro a descrever este comportamento matematicamente (ROBERTS, 2010).

O fenômeno de Gibbs ocorre devido a um truncamento abrupto da série quando utilizam-se apenas os primeiros N termos da série de Fourier para sintetizar a função descontínua, atribuindo peso unitário para as primeiras N harmônicas e peso zero para as harmônicas restantes após N (LATHI, 2006).

Segundo o fenômeno de Gibbs, a ultrapassagem vertical máxima próximo a uma descontinuidade não reduz seu valor independente se N tender a infinito. Porém, é possível notar pela Figura 4, que com o aumento do valor de N, as

ondulações diminuem sua largura e tendem a ficar confinadas próximas a descontinuidade. A medida que N tende ao infinito, a altura desse sobre-sinal torna-se constante e sua largura tende a zero (ROBERTS, 2010).

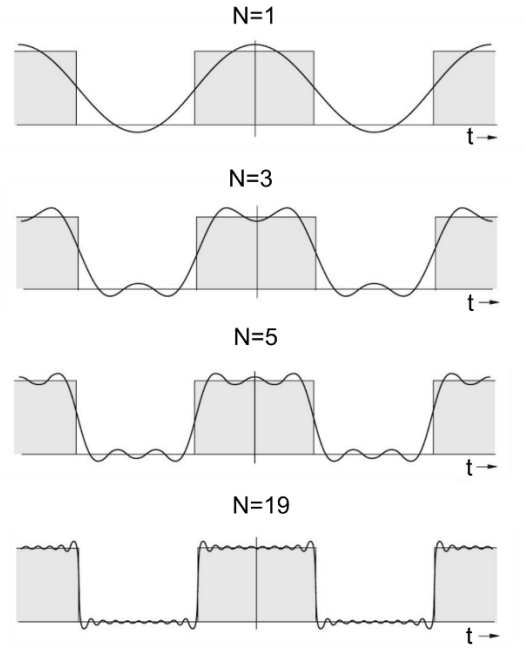


Figura 4 – Fenômeno de Gibbs.
Fonte: Adaptado de Lathi (2006).

2.2.2 Transformada Z

Uma das vantagens de representar uma resposta do sistema LTI no domínio da frequência é que as interpretações de seu comportamento são mais acessíveis. Uma função pode ser convertida do domínio do tempo para o domínio da frequência através de transformadas e da mesma forma pode-se transformar um sinal representado no domínio da frequência para o domínio do tempo (OPPENHEIM; SCHAFER, 1998) (PROAKIS; MANOLAKIS, 2007).

A transformada Z para sinais e sistemas de tempo discreto desempenha a mesma função que a da transformada de Laplace para sinais de tempo contínuo, sendo capaz de caracterizar o sistema LTI pelo seus locais de pólos e zeros. A transformada Z bilateral, infinita para ambos os lados, de uma sequência x[n] é definida como (PROAKIS; MANOLAKIS, 2007):

$$X(z) = \sum_{n=-\infty}^{\infty} x[n].z^{-n} \quad (6)$$

Essa equação é uma série de potência infinita que considera z uma variável complexa, transforma um sinal do domínio do tempo x[n] em sua representação no plano complexo X(z). A transformada unilateral é dada quando considera-se x[n]=0 para todo n<0, denotada como (PROAKIS; MANOLAKIS, 2007):

$$X(z) = \sum_{n=0}^{\infty} x[n].z^{-n} \quad (7)$$

Por conveniência, considera-se a equação como um operador que transforma uma sequência de x[n] para uma função X(z), da seguinte forma (PROAKIS; MANOLAKIS, 2007):

$$Z\{x[n]\} = X(z) \quad (8)$$

A variável complexa z é expressa na forma polar como $z=e^{j\omega}$, sendo r = |z|, isto é, sua representação gráfica representa um círculo no plano complexo.

A transformada Z não converge para todas as sequências ou para todos os valores de z. Para dada sequência, o conjunto de valores de z para o qual a transformada converge é chamada de região de convergência - ROC (do inglês *Region of Convergence*) (OPPENHEIM; SCHAFER, 1998). Essa região pode ser definida a partir de:

$$\sum_{n=-\infty}^{\infty} |x[n]||z|^{-n} < \infty \quad (9)$$

É evidente que existe uma relação estreita entre a transformada de Fourier e transformada z. Se a variável complexa z for substituída pelo complexo $e^{j\omega}$ e restringindo |z|=1 e r=1, a transformada Z reduz para a transformada de Fourier. No plano z, o contorno correspondente a |z| = 1 é um círculo de raio unitário, chamado de círculo unitário, é o conjunto dos pontos $z=e^{j\omega}$, para o intervalo de $0 \leq \omega < 2\pi$. Dessa maneira, se a ROC inclui o círculo unitário, a convergência da transformada Z para |z|=1, ou a transformada de Fourier da sequência converge. Analogamente, se a ROC não inclui o círculo unitário, a transformada de Fourier não converge absolutamente (OPPENHEIM; SCHAFER, 1998).

A transformada Z é mais adequada quando a soma infinita pode ser expressa de forma fechada, ou seja, somada e expressa como uma fórmula matemática simples. Entre as transformadas z mais importantes estão aquelas em que X(z) é igual a uma função racional dentro da ROC, onde P(z) e Q(z) são polinômios em z, da forma (OPPENHEIM; SCHAFER, 1998):

$$X(z) = \frac{P(z)}{Q(z)} \tag{10}$$

Os valores de z para o qual X(z) = 0, são os chamados zeros da função X(z), e os valores para z no qual X (z) é infinito, são os chamados pólos da função X(z) (PROAKIS; MANOLAKIS, 2007).

Quaisquer sequências que podem ser representadas como uma soma de exponenciais podem, equivalentemente, ser representadas por uma transformada Z racional, determinada por seus zeros e pólos (PROAKIS; MANOLAKIS, 2007).

2.3 PROCESSAMENTO DE SINAIS

Em nosso cotidiano, os sinais encontram-se de diversas formas, como sinais de áudio, vídeo e imagem. O processamento de sinais consiste em representar, transformar e manipular os sinais e as suas informações utilizando processadores, de modo que algumas características do sinal sejam alteradas ou a informação contida seja extraída com fidelidade sem qualquer perda ou distorção (OPPENHEIM; SCHAFER, 1998).

Com a evolução da tecnologia digital de computadores e microprocessadores, a partir da década de 60, surge o processamento digital de sinais. Esse tipo de processamento realiza o processamento de sequência de amostras do sinal analógico por meio de algoritmos computacionais (PROAKIS; MANOLAKIS, 2007). As etapas do processamento digital de sinais são apresentadas na Figura 5.

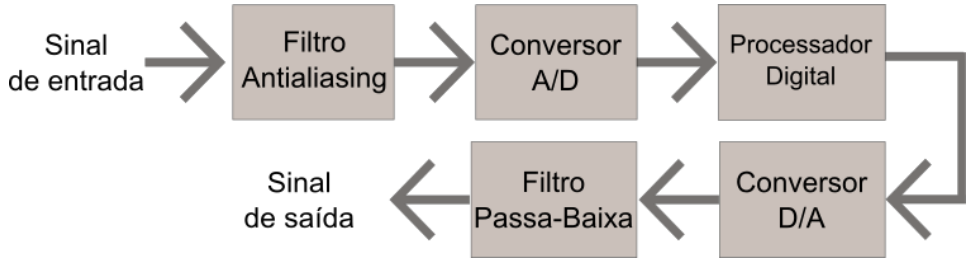


Figura 5 - Etapas do processamento digital.
Fonte: Autoria própria.

O processamento digital de sinais pode apresentar algumas vantagens comparado ao processamento analógico graças a sua flexibilidade de reconfiguração, precisão, redução de custos e facilidade de implementação. A flexibilidade do processamento digital deve-se ao fato do *software* ou *hardware* programável permitir operações programáveis de modo que pode ser modificado para atender ou melhorar o desempenho de um sistema (JESZENSKY, 2004).

A precisão de um sistema digital em relação a um sistema analógico é dada pelo fato dos dispositivos digitais não apresentam tolerâncias de fabricação além de não sofrem tanto com as variações do ambiente quanto os dispositivos analógicos. Entretanto, para que a precisão seja satisfatória é necessário que o conversor utilizado para transformar o sinal analógico em digital apresente uma alta velocidade de conversão e resolução no processo de quantização (OPPENHEIM; SCHAFER, 1998).

Em alguns casos, o processamento digital é mais barato que o analógico, pois não necessita de circuitos analógicos de precisão, além de apresentar uma quantidade menor de circuitos integrados e a reconfiguração ou manutenção pode ser feita de maneira mais flexível (JESZENSKY, 2004).

A facilidade de implementação deve-se ao fato de algumas operações matemáticas serem de difícil implementação analógica quando comparando com as mesmas operações implementadas em um processador digital (PROAKIS; MANOLAKIS, 2007). Graças a essa facilidade e as vantagens citadas anteriormente, as técnicas de processamento digital de sinais têm sido muito utilizadas em aplicações de telecomunicações, informática, instrumentação e diversas outras áreas (OPPENHEIM; SCHAFER, 1998).

2.3.1 Sinais e Sistemas

Os sinais estão presentes em diversas situações do cotidiano dos humanos, como o sinal de voz que permite a comunicação pessoalmente ou por um canal de transmissão. Um sinal pode ser definido como uma função no qual contém informações sobre o estado ou comportamento de um fenômeno físico denotado no domínio adequado (HAYKIN; VEEN, 2001).

Considerando o domínio do tempo, os sinais podem ser classificados em: sinais contínuos e sinais discretos. Os sinais contínuos apresentam um valor definido para qualquer instante de tempo. Enquanto um sinal discreto apresenta valor definido apenas para alguns instantes de tempo (HAYKIN; VEEN, 2001).

Quando um sinal, além de apresentar valores para qualquer instante de tempo, também apresentar uma amplitude de qualquer valor dentro de um intervalo contínuo é chamado de sinal analógico. Da mesma maneira, quando um sinal apresenta valores discretos de tempo e além disso apresenta um valor de amplitude discreto que assume apenas um valor dentro de um intervalo finito, é chamado de sinal digital. Por essa razão o sinal digital é constantemente representado matematicamente por uma sequência de números reais ou complexos (PROAKIS; MANOLAKIS, 2007). A Figura 6 apresenta um sinal analógico x(t) em função do tempo t e um sinal digital x[n] em função do número de amostras n.

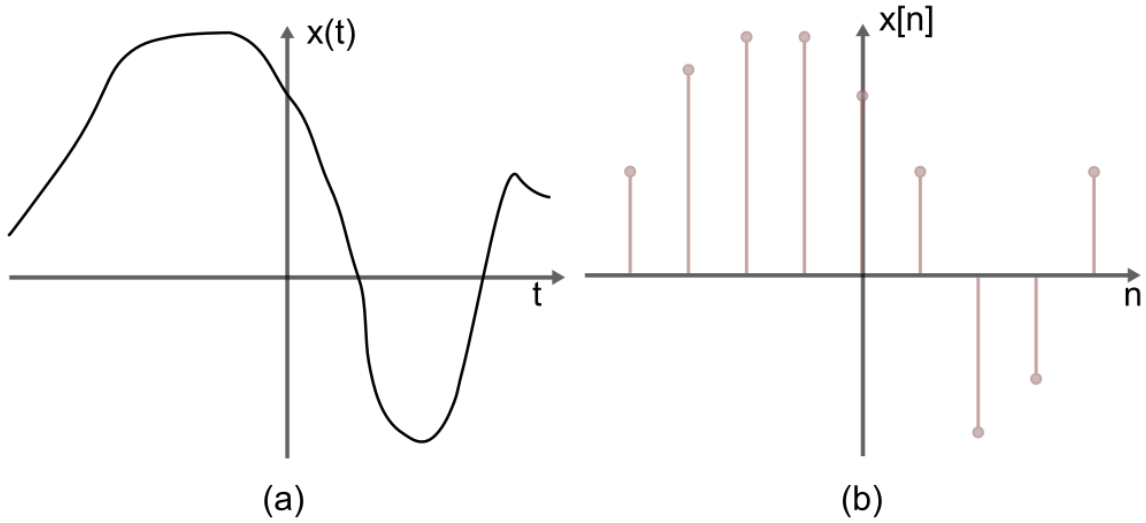


Figura 6 - Representação de um: (a) sinal analógico x(t) e (b) sinal digital x[n].
Fonte: Autoria própria.

Além da classificação conforme o tempo, os sinais podem ser classificados em periódicos e aperiódicos. Os sinais periódicos apresentam a reincidência de seus valores de amplitude após intervalores regulares de tempo, ou seja, apresentam um período fundamental T que satisfaz a seguinte condição (HAYKIN; VEEN, 2001):

$$x(t) = x(t+T) \tag{11}$$

Já os sinais aperiódicos não apresentam repetição de seus valores ao longo do tempo de modo que não apresentam um período que satisfaça (11) (HAYKIN; VEEN, 2001).

De acordo com HAYKIN e VEEN (2001), um sistema é uma entidade que manipula sinais para realizar determinadas funções a fim de resultar em novos sinais. Os sistemas também podem ser classificados como sistemas contínuos e sistemas discretos. Enquanto os sistemas contínuos realizam operação com os sinais e resultam em um novo sinal contínuo, os sistemas discretos realizam operações sobre sinais discretos e obtêm uma sequência de saída discreta correspondente.

Os sinais encontrados na natureza são analógicos, como a onda sonora e a onda luminosa que são transformados em sinais elétricos por transdutores (HAYKIN; VEEN, 2001). Porém, esse tipo de sinal não é adequado para ser utilizado por um processador digital devido a infinidade de valores que a função deste tipo de sinal pode assumir. Para utilização desse sinal é necessário que ele seja convertido em um sinal digital por um conversor Analógico-Digital (A/D) (PROAKIS; MANOLAKIS, 2007).

Em algumas aplicações é necessário que o sinal de saída do processamento seja analógico. Para que isso seja possível é necessário que o sinal digital seja convertido em um sinal analógico. Esse processo de conversão pode ser realizada por um dispositivo conhecido como conversor digital-analógico (D/A) (OPPENHEIM; SCHAFER, 1998).

2.3.2 Conversão A/D

A conversão A/D é caracterizada por três estágios: amostragem, quantização e codificação. O estágio de amostragem consiste em adquirir amostras a cada tempo de amostragem T_S (HAYES, 2006). Na etapa de quantização, são estipulados valores discretos a cada amostra, adquirida na etapa de amostragem, dentro de uma escala definida. Na codificação, cada valor discretizado passa a ser

representado por uma sequência de bits. Desta forma, o sinal discreto torna-se um sinal digital (PROAKIS; MANOLAKIS, 2007). A Figura 7 apresenta as etapas de um conversor A/D.

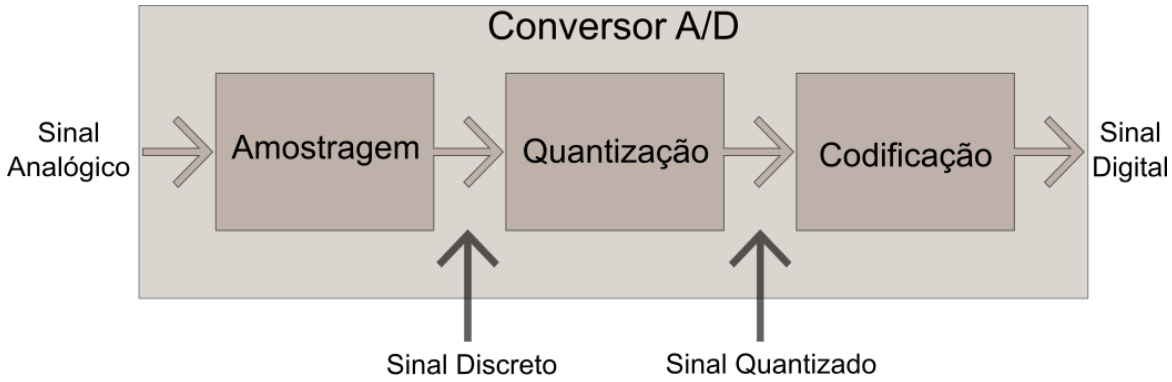


Figura 7 - Etapas do conversor A/D.
Fonte: Adaptado de Proakis e Monolakis (2007).

2.3.2.1 Amostragem de sinais e fenômeno de a*liasing*

A etapa de amostragem de um conversor A/D consiste em obter uma amostra do sinal contínuo em intervalos regulares de tempo, denominado período de amostragem T_a. Dessa maneira, obtêm-se o sinal discreto $x_d[n]$ por meio da relação (PROAKIS; MANOLAKIS, 2007):

$$x_d[n] = x(nT_a), \tag{12}$$

em que n representa o número da amostra, podendo assumir valores de -∞ a +∞.

Matematicamente, o sinal amostrado $x_\delta(t)$ pode ser representado como o produto do sinal de tempo contínuo original x(t) por um trem de impulsos, podendo ser expresso como (HAYKIN; VEEN, 2001):

$$x_\delta(t) = x(t).\sum_{n=-\infty}^{\infty} \delta(t - nT_a) \tag{13}$$

A Figura 8 apresenta um exemplo da representação de (13), também chamada de amostragem por impulsos (HAYKIN; VEEN, 2001).

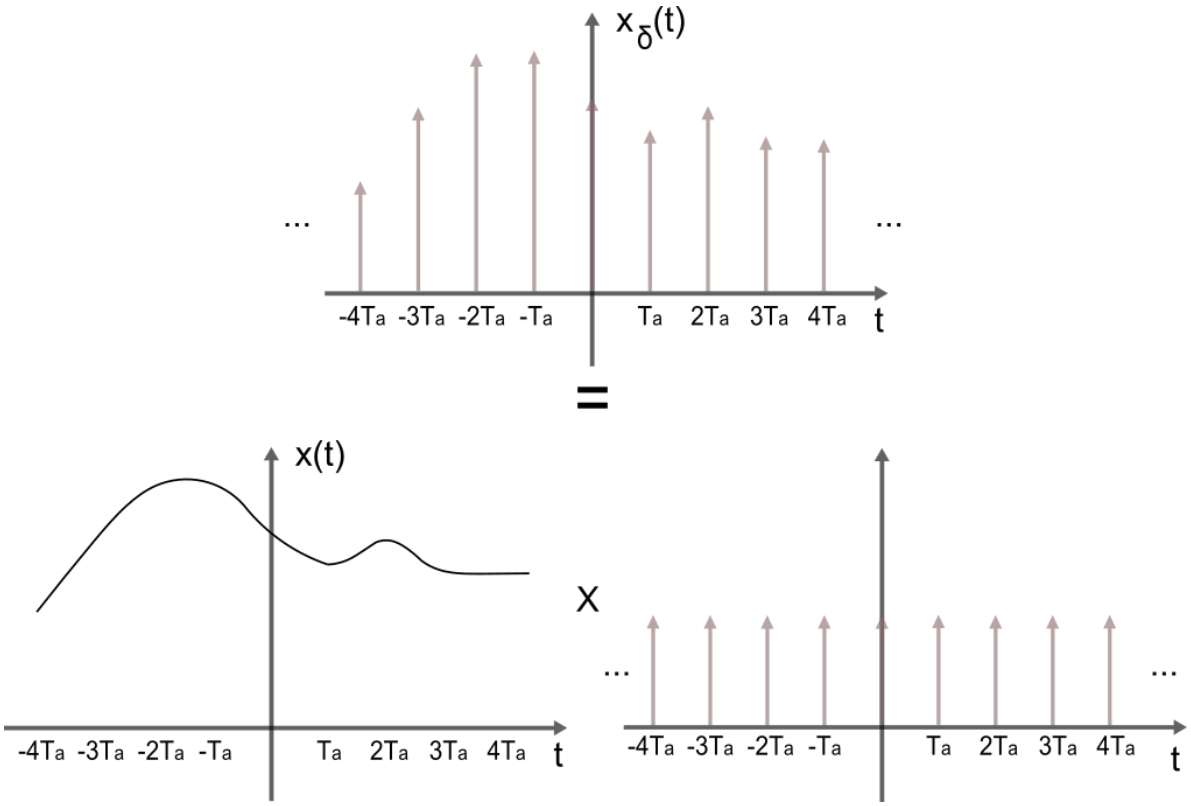


Figura 8 - Representação matemática de um sinal amostrado.
Fonte: Adaptado de Haykin e Veen (2001).

Aplicando a transformada de Fourier na equação (13) tem-se que (HAYKIN; MOHER, 2008):

$$X_{\delta}(t) = \frac{1}{T_a} \sum_{k=-\infty}^{\infty} X(j(\omega - k\omega_s)) \tag{14}$$

A partir de (14) é possível perceber que o sinal amostrado é a soma infinita de versões deslocadas do sinal original, espaçadas por múltiplos inteiros da frequência angular de amostragem ω_s. Se a frequência de amostragem não for suficientemente grande comparada com a extensão de frequência do sinal original, ocorre a sobreposição das versões deslocadas, cujo fenômeno é denominado *aliasing* (HAYKIN; VEEN, 2001). O fenômeno de *aliasing* resulta na perda das componentes de frequência com valores acima da metade da frequência de amostragem. Além disso, essas frequências mais altas assumem a identidade das frequências mais baixas (LATHI, 2006).

Com a finalidade de se evitar qualquer distorção, em 1927, Harry Nyquist propôs um teorema conhecido como Teorema da Amostragem (ROCHOL, 2012).

Considerando um sinal composto por componentes de diversas frequências e limitado em banda, Nyquist determina que a frequência de amostragem f_s deve ser pelo menos o dobro da frequência mais elevada contida no sinal f_{max}. Desse modo (FOROUZAN, 2008):

$$f_s \geq 2 f_{\max} \tag{15}$$

Um sinal amostrado de maneira que atenda a condição do teorema de amostragem pode ser reconstruído por meio de uma boa aproximação, sem sobreposições de frequências, contendo toda a informação necessária presente no sinal (PROAKIS; MANOLAKIS, 2007).

De acordo com o teorema de amostragem, Nyquist estabeleceu que a capacidade máxima de transmissão C de um canal, considerando somente a banda passante do meio e o tipo de codificação, é dada por (CARISSIMI; ROCHOL; GRANVILLE, 2009):

$$C = 2Wn_b, \tag{16}$$

em que a capacidade C é dada em bits/s, a banda passante W dada em Hertz (Hz) e n_b corresponde ao número de bits.

Com base em (16), a capacidade do canal é ilimitada, pois, teoricamente, n_b pode se estender ao infinito, dependente somente da banda passante e da codificação. Com base nisso, em 1948, o americano Claude Shannon provou que um canal apresenta limitação na capacidade de transmissão por meio de (COMER, 2016):

$$C = W \log_2(1 + S / N), \tag{17}$$

em que a capacidade do canal C (bit/s) depende da banda de passagem W (Hz) e uma relação sinal-ruído S/N (dB), ou seja, a capacidade depende somente de condições do meio, sem levar em consideração a capacidade do codificador do sistema (CARISSIMI; ROCHOL; GRANVILLE, 2009) .

2.3.2.2 Problemas de Quantização

Um quantizador é sistema irreversível que transforma uma sequência discreta de entrada, com intervalo contínuo de amplitude, em uma sequência discreta em que cada amplitude assume um valor dentro de um intervalo, ou seja, a sequência

de amostras é convertida a um sinal digital em que os valores de amplitudes são discretizados e apresentam somente valores dentro de um intervalo estipulado (HAYES, 2006).

Seja um sinal analógico com valores entre V_{max} e V_{min}, a resolução do quantizador ou tamanho do degrau é dado por (FOROUZAN, 2008):

$$\Delta = (V_{\max} - V_{\min}) / (N - 1), \tag{18}$$

em que $N=2^B$ é chamado de níveis do quantizador e B é a quantidade de bits do conversor. Na quantização atribui-se os valores quantizados entre 0 e N-1 para o ponto médio de cada nível. Assim, para cada amplitude amostrada é aproximado um valor quantizado de um dos níveis (FOROUZAN, 2008).

A etapa de quantização do conversor A/D gera erros no sinal processado pois cada amostra de sinal analógico é aproximado de seu equivalente digital mais próximo. A quantidade de erro depende da precisão do conversor A/D que é dado pela medida do número de *bits*. Quanto maior for o número de *bits* menor será o erro, pois um número maior de *bits* implica em uma resolução menor capaz de tornar a aproximação mais precisa (SCHULER, 2013). O erro de quantização, também denominado de ruído de quantização, muda a relação sinal-ruído do sinal. A contribuição desse erro depende da quantidade de bits por amostra é dada em decibéis pela equação (FOROUZAN, 2008):

$$SNR_{dB} = 6,02B + 1,76dB \tag{19}$$

A Figura 9 apresenta uma quantização realizada com maior e menor número de bits.

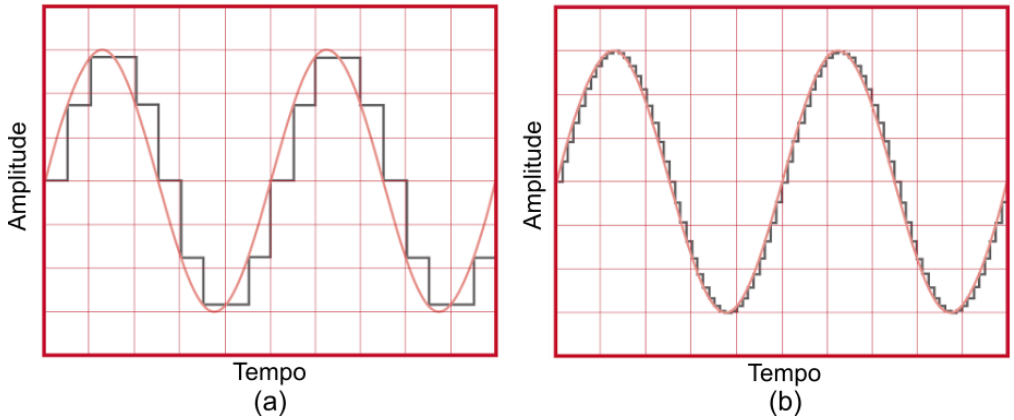


Figura 9 - Erros de quantização: (a) com número menor de *bits* e (b) com número maior de *bits*. Fonte: Adaptado de Schuler (2013).

2.3.3 Reconstrução de sinais

O processo de reconstrução consiste em converter o sinal digital x[n] em um trem de impulsos modulados $x_s(t)$. Dessa maneira, cada amostra é convertida em um impulso unitário de tempo contínuo espaçado do impulso seguinte por um período de amostragem T_a. Portanto, $x_s(t)$ é dado por (NALON, 2009):

$$x_s(t) = \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT_a) \tag{20}$$

Para obter o sinal reconstruído $x_c(t)$ é necessário que o sinal $x_s(t)$ passe por um filtro passa baixa, eliminando as componentes de alta frequência que podem gerar o efeito de *aliasing* (HAYKIN; VEEN, 2001). A resposta em frequência do filtro é dada por (NALON, 2009):

$$H(j\Omega) = \begin{cases} T_a & |\Omega| \leq \dfrac{\pi}{T_a} \\ 0 & |\Omega| > \dfrac{\pi}{T_a} \end{cases} \tag{21}$$

A Figura 10 mostra as etapas da conversão ideal de um sinal digital x[n] para um sinal analógico $x_c(t)$.

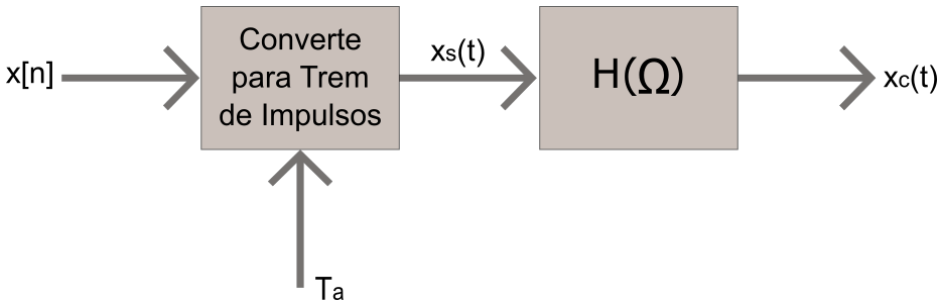


Figura 10 - Etapas de reconstrução de um sinal.
Fonte: Adaptado de Nalon (2009).

Na prática, a geração dos impulsos não ocorre facilmente devido aos transitórios abruptos. Por isso, na reconstrução de sinais são utilizados retentores ou interpoladores. Devido a sua simplicidade, utiliza-se o retentor de ordem zero que mantém ou retém o valor de x[n] pelo período T_a. Esse processo acarreta em uma

aproximação em degraus de escada para o sinal contínuo, como pode-se observar na Figura 11 (HAYKIN; MOHER, 2008).

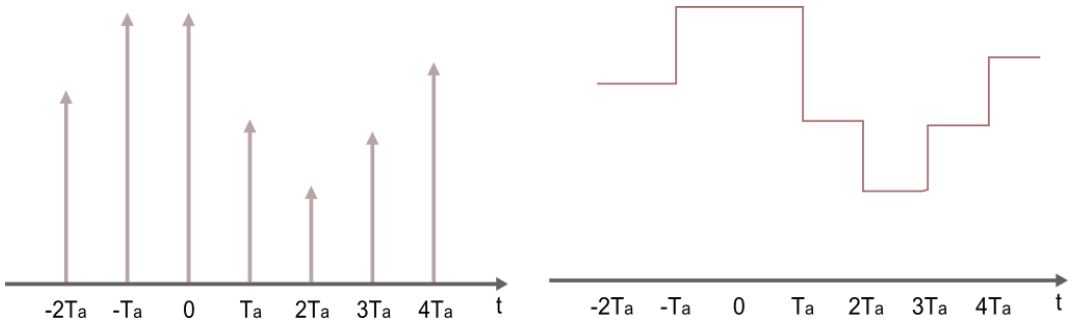


Figura 11 - Efeito da saída de um retentor de ordem zero.
Fonte: Adaptado de Hayes (2006).

Embora o retentor de ordem zero altere a resposta de frequência, ainda é necessário a utilização do filtro *antialiasing*, para eliminar as componentes de frequências fora da faixa original do sinal. Portanto, na prática, a reconstrução do sinal digital é dada pela combinação da conversão do sinal digital em trem de pulsos, o retentor de amostra e de um filtro passa baixa (PROAKIS; MANOLAKIS, 2007). A Figura 12 apresenta essas etapas de reconstrução.

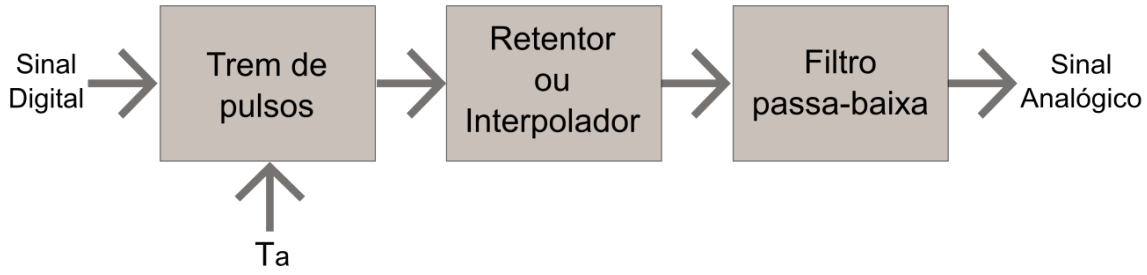


Figura 12 - Etapas da conversão D/A na prática.
Fonte: Adaptado de Proakis e Monolakis (2007).

A conversão D/A baseia-se em produzir um sinal analógico a partir de um sinal digital, cujo processo também é conhecido como interpolação. Na prática, o sinal resultante dos conversores D/A é equivalente a saída do retentor ou interpolador. Então, a reconstrução pode ser feita a partir de um conversor D/A e um filtro passa-baixa.

2.3.4 Filtragem de sinais e filtros

Como descrito nos capítulos anteriores, a filtragem de sinais é utilizada para diversas aplicações tecnológicas e, por esse motivo, tem sido foco de estudo em processamento de sinais afim de realizar melhorias e otimização desse processo.

A filtragem de sinais é realizada por sistemas, denominados filtros, que são utilizados para selecionar faixas de frequências de modo a deixar passar determinada banda de frequência e rejeitar as indesejadas de um sinal (SADIKU; ALEXANDER; MUSA, 2014). A banda em que apresenta as frequências que passam pelo filtro é denominada banda de passagem, e a banda que apresenta as frequências rejeitadas pelo filtro é denominada de banda de rejeição (NALON, 2009).

Os filtros podem ser classificados como filtro: passa-baixa, passa-alta, passa-faixa e rejeita-faixa (SADIKU; ALEXANDER; MUSA, 2014). A Figura 13 apresenta a resposta em frequência de um filtro ideal de cada tipo.

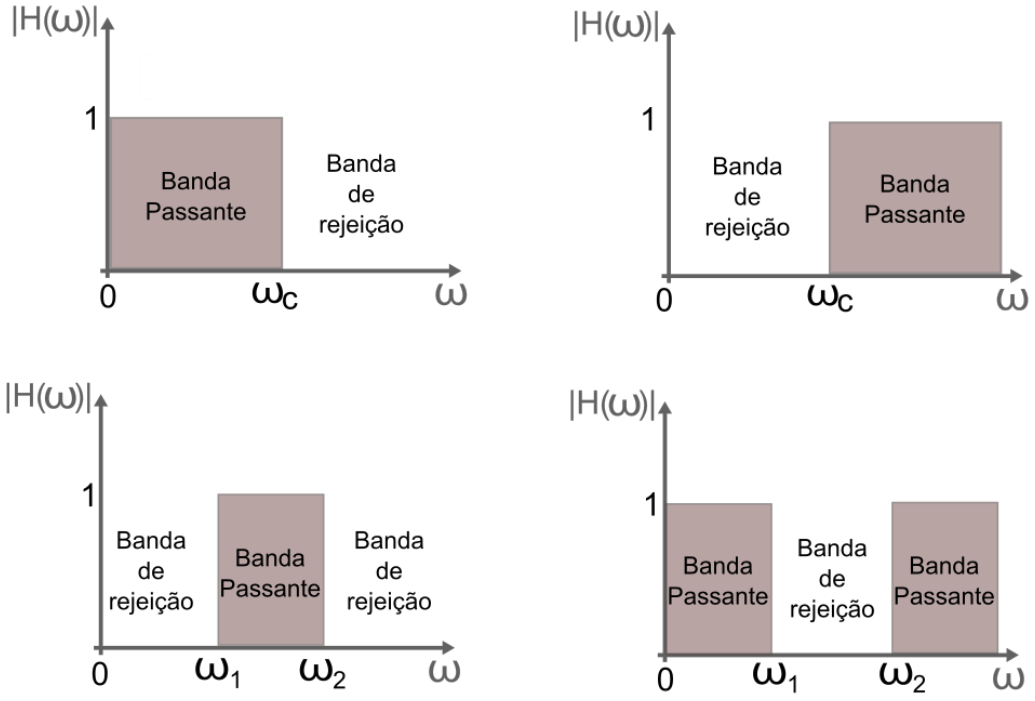


Figura 13 - Respostas ideais em frequência para filtro: (a) passa-baixa, (b) passa-alta, (c) passa-faixa e (d) rejeita banda.
Fonte: Adaptado de Sadiku, Alexander e Musa (2014).

Um filtro passa-baixa permite a passagem de baixas frequências do sinal de entrada e rejeita as componentes de alta frequência do sinal. De modo contrário,

um filtro passa-alta permite a passagem de altas frequência e rejeita as baixas frequências de um sinal. O filtro passa-faixa permite a passagem de uma banda intermediária, enquanto rejeita as componentes de baixa e alta frequência do sinal fora dessa banda. Já o filtro rejeita-faixa rejeita as componentes de frequência dentro de uma determinada banda intermediária e permite a passagem das frequências fora dessa banda (NALON, 2009).

As frequências que separam a banda de passagem da banda de rejeição de um filtro são denominadas de frequências de corte. Define-se que na frequência de corte há a perda da metade da potência do sinal, equivalente a -3 dB na amplitude (NALON, 2009). Por exemplo, na Figura 13, as frequências de corte são representadas por ω_c, ω_1 e ω_2. Quando o ganho em amplitude do filtro $|H(\omega)|$ for menor que um, o sinal de saída é atenuado, ou seja, a amplitude do sinal de saída é menor que a amplitude do sinal de entrada. No entanto, se o ganho do filtro for maior que um, o sinal será amplificado (NALON, 2009).

Em filtros ideais, considera-se a passagem abrupta entre a banda de passagem e a banda de rejeição. Porém, na prática, os filtros apresentam uma banda de transição entre as bandas de passagem e rejeição (SADIKU; ALEXANDER; MUSA, 2014). A Figura 14 apresenta a resposta em frequência de um filtro passa-baixa ideal e real para comparação.

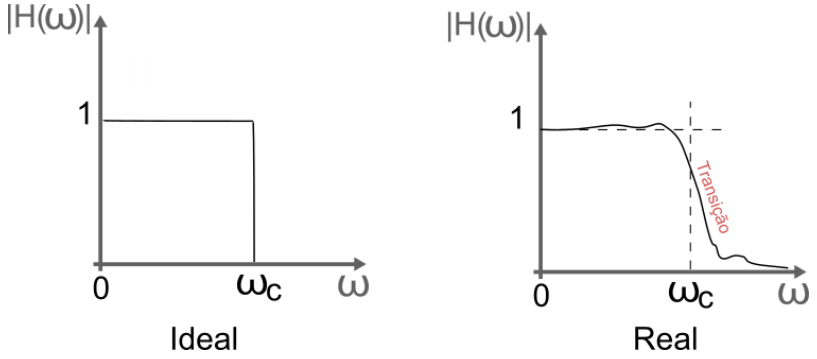


Figura 14 - Comparação da resposta em frequência de um filtro passa-baixa ideal e real.
Fonte: Adaptado de Schuler (2013).

Os filtros reais se diferenciam ainda mais dos filtros ideais, pois podem apresentar ondulações, denominadas *ripples*, na banda passante e na banda de rejeição. Também, apresentam a possibilidade de perda na banda passante e a atenuação na banda de corte não é infinita. A ordem do filtro gera influência sobre

esses aspectos de modo que quanto maior a ordem melhor é a resposta em frequência do filtro real (SCHULER, 2013).

Além da classificação quanto a função executada por um filtro, ainda pode ser classificado quanto a sua função-resposta e tecnologia empregada (JUNIOR, 2003). As funções-resposta mais conhecidas são as de Butterworth, Chebyshev, Cauer e Bessel. Essas funções apresentam uma função matemática capaz de aproximar a curva de resposta para cada tipo de filtro (FRENZEL, 2013). A Figura 15 apresenta as curvas-respostas dessas aproximações.

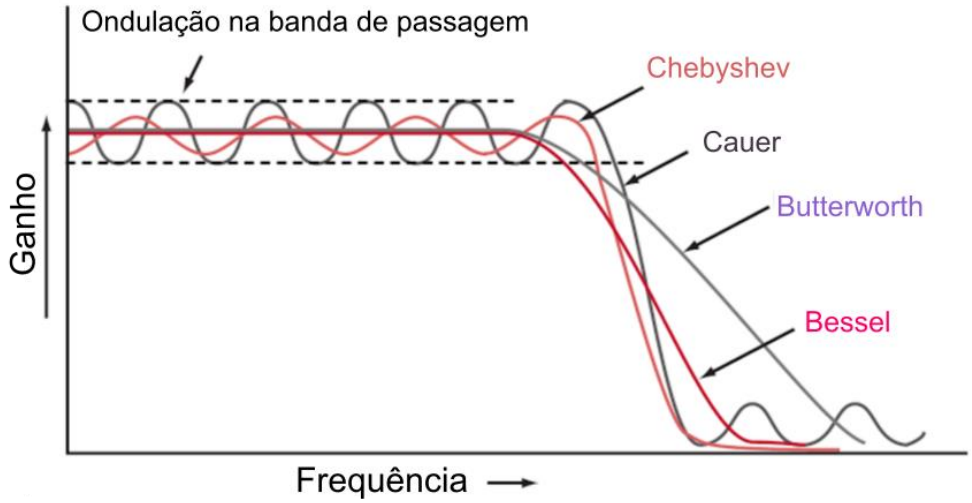


Figura 15 - Curvas de respostas de filtros Chebyshev, Cauer, Butterworth e Bessel.
Fonte: Adaptado de Frenzel (2013).

O filtro com aproximação de Butterworth apresenta uma curva plana na banda de passagem e uma atenuação uniforme com a frequência, porém a atenuação fora da banda de passagem não é tão acentuada. O filtro Cauer apresenta um decaimento acentuado e uma atenuação acentuada fora da banda de passagem, contudo apresenta uma oscilação na banda de passagem e na banda de corte. Já o filtro Chebyshev apresenta um decaimento acentuado, mas inferior ao filtro Cauer, e sua atenuação da banda de corte é alta, entretanto, também possui ondulações na banda de passagem. Por fim, o filtro Bessel não apresenta ondulações na banda de passagem, mas apresenta uma atenuação menor na banda de corte (FRENZEL, 2013).

Os filtros podem ser classificação quanto a tecnologia empregada em sua implementação, sendo divididos em filtros analógicos e filtros digitais. Essas duas classificações estão relacionadas ao domínio em quem os sinais são tratados. Os

filtros analógicos manipulam sinais analógicos e filtros digitais atuam em sinais digitais (NALON, 2009).

Na forma analógica, os filtros podem ser implementados de duas maneiras: utilizando componentes passivos, como os resistores, capacitores e indutores, ou utilizando componentes ativos, como os amplificadores operacionais, amplificadores operacionais de transcondutância e transistores.

Os filtros digitais utilizam processadores digitais para realizar as funções de filtragem no sinal digital. Para realizar a filtragem digital de um sinal analógico, este precisa ser convertido em sinal digital, por meio de um conversor A/D, e, posteriormente, reconstruído a partir de um conversor D/A e um filtro passa-baixa (JUNIOR, 2003).

2.4 FILTROS DIGITAIS

Na área de processamento de sinais os filtros digitais representam soluções proeminentes, pois apresentam algumas vantagens consideráveis perante aos filtros analógicos, principalmente nos quesitos flexibilidade, precisão e confiabilidade (PROAKIS; MANOLAKIS, 2007).

Os filtros digitais podem ser representados conforme sua resposta à função impulso e classificados de duas formas. Quando a resposta ao impulso de um filtro é finita, denomina-se de filtro FIR (do inglês *Finite Impulse Response*) e quando a resposta é infinita, denomina-se de filtros IIR (do inglês *Infinite Impulse Response*) (NALON, 2009).

2.4.1 Filtros FIR

O filtro FIR apresenta uma estrutura bastante regular, no qual pode ser implementado por meio de equações de diferenças que não contenham termos recursivos, ou seja, a saída não é realimentada. Sendo um filtro FIR de comprimento L e considerando que a resposta a uma entrada impulso torna-se zero após um número finito de amostras L de saída, tem-se que a saída desse tipo de filtro é dada por (KUO; GAN, 2005):

$$y(n) = \sum_{i=0}^{L-1} d_i x(n-i), \tag{22}$$

em que d_i são os coeficientes desse filtro. Aplicando uma entrada do tipo impulso unitário em um filtro FIR, a saída será exatamente os valores dos coeficientes desse filtro. A quantidade de coeficientes desse tipo de filtro, ou comprimento L, é denominada *taps*, enquanto que a ordem do filtro apresenta diferença de menos um dessa quantidade de *taps* (WEEKS, 2012). Os filtros FIR podem implementar diferentes funções apenas alterando os valores desses coeficientes (PROAKIS; MANOLAKIS, 2007). A Figura 16 apresenta uma forma geral do filtro FIR para k+1 coeficientes.

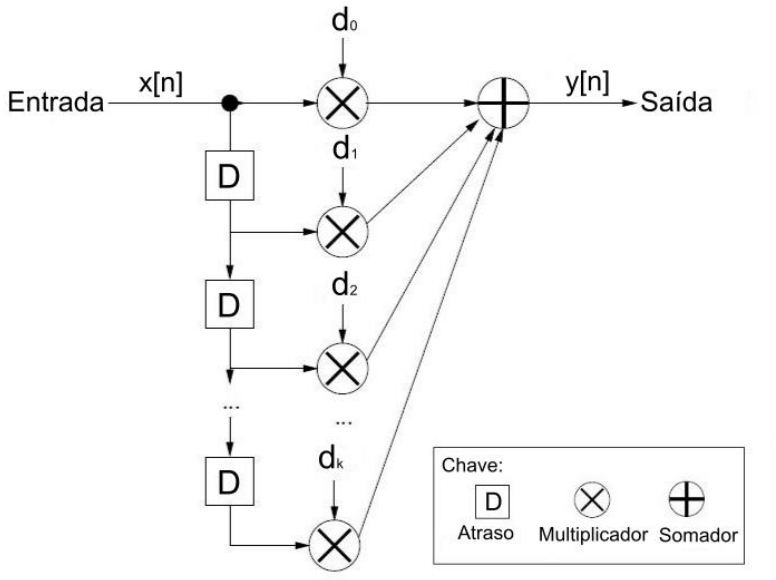


Figura 16 – Forma geral de um filtro FIR.
Fonte: Adaptado de Weeks (2012).

Aplicando a transformada Z em ambos os lados de (22), obtém-se a função de transferência do filtro FIR como (KUO; GAN, 2005):

$$H(z) = \sum_{i=0}^{L-1} d_i z^{-i} \tag{23}$$

Um filtro FIR com fase linear é utilizado principalmente para aplicações que desejam eliminar distorções em transmissões. Um filtro FIR de fase linear tem correspondência com a simetria da resposta ao impulso do filtro, pois a simetria garante que a resposta em frequência do filtro não altera a fase do sistema, somente

a magnitude (PROAKIS; MANOLAKIS, 2007). Em relação a simetria, os filtros FIR podem ser divididos em quatro tipos: tipo I, tipo II, tipo III e tipo IV. As características desses tipos estão presentes na Tabela 1.

Tabela 1 – Comparativo entre simetrias.

Tipo	*Resposta ao impulso*	Ordem (M)
I	h[n] = h [M-n]	Par
II	h[n] = h [M-n]	Ímpar
III	h[n] = - h [M-n]	Par
IV	h[n] = - h [M-n]	Ímpar

Fonte: Adaptado de Nalon (2009).

2.4.2 Filtros IIR

O filtro IIR é um tipo de filtro que apresenta realimentação da saída, sendo conhecido também como filtro recursivo. Por isso, sua resposta a uma entrada impulso não é finita. A saída desse filtro é expressa por (KUO; GAN, 2005):

$$y(n) = \sum_{i=0}^{L-1} b_i x(n-i) - \sum_{m=1}^{M} a_m y(n-m), \tag{24}$$

em que os coeficientes b_i e a_m determinam a resposta do filtro. A expressão (24) também pode ser reorganizada e, aplicando a transformada Z, resultar na função de transferência do filtro IIR, dada por (KUO; GAN, 2005):

$$H(z) = \frac{\sum_{i=0}^{L-1} b_i z^{-i}}{1 + \sum_{m=1}^{M} a_m z^{-m}} \tag{25}$$

Embora, na teoria, um filtro IIR nunca chegar a uma saída nula, ou seja, apresenta uma resposta infinita, na prática um filtro digital possui uma precisão finita. Entretanto, um filtro IIR pode apresentar saídas indesejadas, como saídas que permanecem constantes ou que continuem crescendo em magnitude com o mesmo valor de entrada. Um filtro IIR é denominado estável quando a saída se aproxima de zero em consequência da entrada se aproximando de zero. Quando a saída oscila a uma taxa constante, o filtro é denominado de condicionalmente estável. Se a saída

cresce em magnitude ou oscila rumo ao infinito, denomina-se o filtro IIR como instável (WEEKS, 2012). A Figura 17 – Forma geral de um filtro IIR.Figura 17 apresenta a forma geral de um filtro IIR, também referenciada como forma direta I.

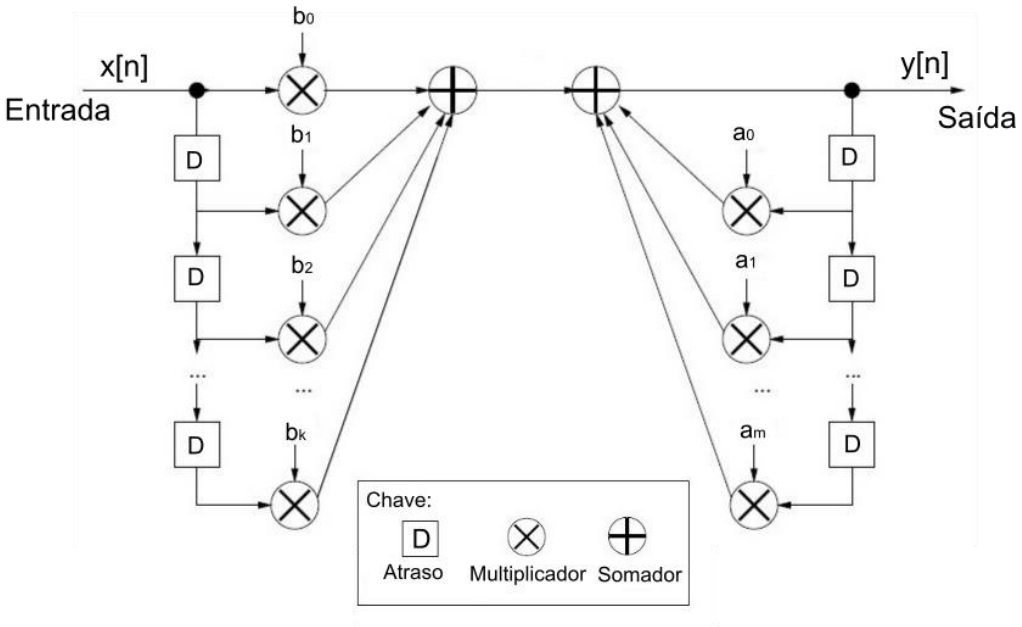


Figura 17 – Forma geral de um filtro IIR.
Fonte: Adaptado de Weeks (2012).

Por meio de (22) e (24), é possível observar que os filtros digitais podem apresentar três propriedades consideráveis: causalidade, linearidade e invariância no tempo. Um sistema é denominado causal quando sua saída depende apenas das entradas correntes e/ou passadas. Dessa forma, um sinal é causal quando possui valor zero para valores de índice menores que zero. Um sistema é considerado linear quando sua saída é expressa por meio de uma soma de entradas multiplicadas por constantes. Todo sistema linear possui propriedade de escalonamento e de aditividade. O escalonamento é garantido quando obtém-se o mesmo resultado multiplicando a entrada por uma constante ou a saída pela mesma constante. Enquanto que a propriedade de aditividade significa que deve-se obter o mesmo resultado passando dois ou mais sinais por um sistema e os somando posteriormente, do somatório dos dois ou mais sinais passando por esse mesmo sistema. Já a invariância no tempo significa que a saída de um sistema reflete a invariância do sinal de entrada, ou seja, se a entrada for defasada em k unidades a saída deve sofre a mesma variação k (WEEKS, 2012).

A decisão de implementar um filtro FIR ou IIR depende da aplicação, mas as características de cada um devem ser levadas em consideração. Como o filtro FIR apresenta memória finita, sempre será estável e qualquer transitório tem duração limitada. Além disso, com algumas restrições, um filtro sem distorção de fase pode ser implementado facilmente. Entretanto, esse tipo de filtro necessita de vários coeficientes, exigindo mais memória, maior número de operações e, consequentemente, mais tempo de processamento por um processador, enquanto os filtros IIR apresentam característica de corte de frequência acentuada com um filtro de ordem relativamente baixa, o que reduz a complexidade e o tempo de processamento (KUO; GAN, 2005).

2.5 PROJETO DE FILTROS DIGITAIS

O projeto e implementação de um filtro digital requer que sejam seguidas algumas etapas, tais como (NALON, 2009):

1 – determinar as características desejadas (conforme a aplicação do filtro);

2 – a partir das especificações, determinar os coeficientes da função transferência do filtro;

3 – selecionar a estrutura adequada que representará o filtro;

4 – implementar em *software* ou *hardware.*

2.5.1 Projeto de Filtros FIR

Os métodos de projeto de um filtro FIR podem ser a partir de janelamento, amostragem em frequência e de médias móveis. Nesse trabalho optou-se pela utilização do método do janelamento, pois é uma forma bastante direta de obtenção de um filtro FIR (NALON, 2009). Além disso, o *software* Matlab® apresenta algumas funções baseadas nesse método.

O método de projeto por janelamento consiste em obter um filtro FIR causal por meio do truncamento da resposta a uma entrada impulso de um filtro ideal $h_D[n]$, com uma função finita w[n], de M+1 amostras, denominada janela (PROAKIS; MANOLAKIS, 2007).

Quando aplicada uma entrada impulso em um filtro ideal obtém-se uma resposta infinita. A Figura 18 apresenta a resposta ao impulso de um filtro passa-baixas em (a) domínio da frequência e (b) no domínio do tempo (NALON, 2009). Em (a) é possível observar que não existe faixa de transição entre a faixa passante e a faixa de transição. Em (b) é possível observar que o filtro é infinito no tempo.

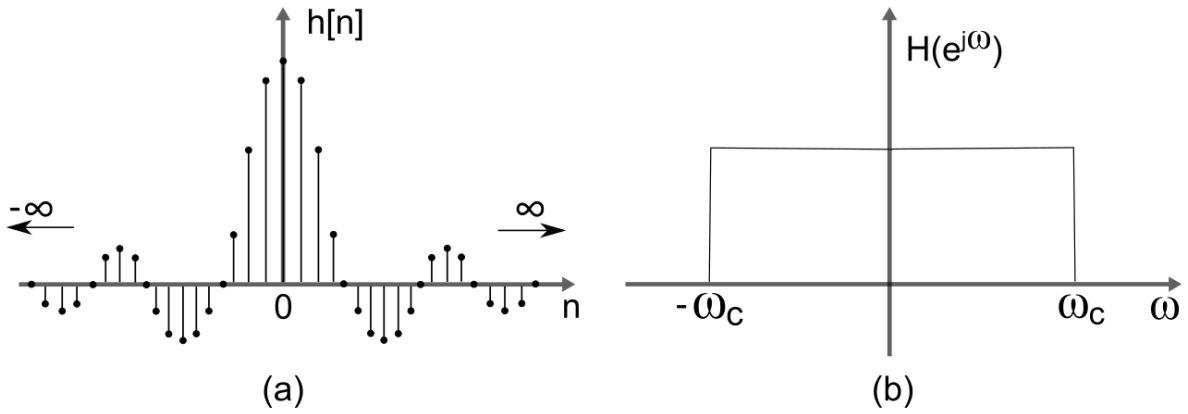


Figura 18 – Resposta de um filtro passa-baixa ideal a uma entrada impulso: (a) no domínio do tempo e (b) no domínio da frequência.
Fonte: Adaptado de Corinthios (2009).

A resposta ao impulso do filtro ideal passa-baixa dada na Figura 18 (b) pode ser expressa por:

$$h[n] = \begin{cases} \dfrac{\Omega_c}{\pi} & para \quad n = 0 \\ \dfrac{sen(\Omega_c n)}{\pi n} & para \quad n \neq 0 \end{cases} \tag{26}$$

Uma janela w[n] é utilizada para limitar a resposta do filtro e assim torná-lo realizável. A aplicação desse método resulta em uma transição mais suave da banda de passagem para banda de rejeição do filtro, além de adicionar uma ondulação a banda passante. A escolha da janela utilizada influencia no filtro que será obtido (NALON, 2009). A Figura 19 apresenta a resposta da janela retangular no domínio da frequência em (a) e no domínio do tempo em (b).

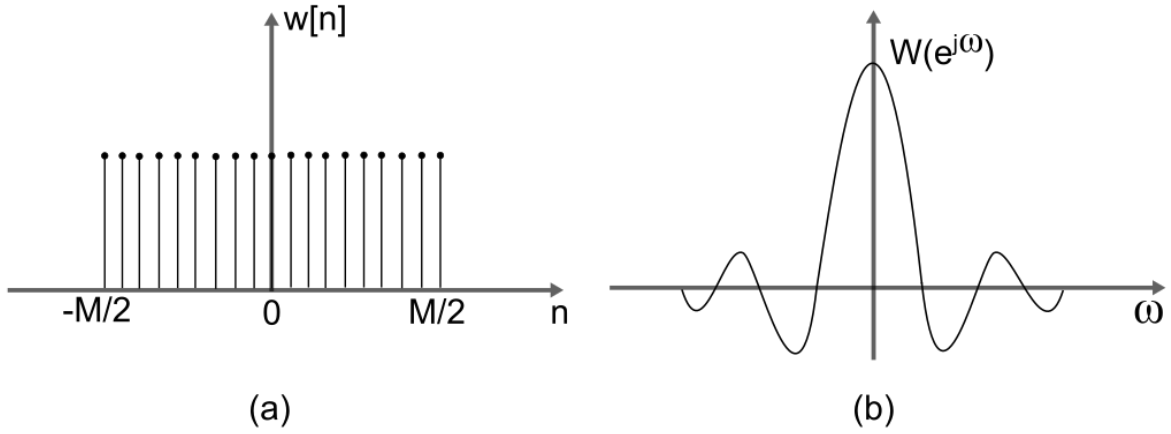


Figura 19 – Janela rentangular no (a) domínio do tempo e (b) no domínio da frequência.
Fonte: Adaptado de Corinthios (2009).

Na Figura 19(b) é possível notar que a janela apresenta M+1 amostras não-nulas, as amostras ou também chamados coeficientes que não aparecem na referida figura apresentam valor zero (PROAKIS; MANOLAKIS, 2007).

Ao truncar a janela w[n] com o filtro ideal $h_D[n]$, as amostras que se localizarem fora da janela serão anuladas, isto é, ao realizar a multiplicação entre o sistema mostrado na Figura 18(b) com a Figura 19(b) obtém-se um filtro finito com apenas M+1 amostras não-nulas (PROAKIS; MANOLAKIS, 2007).

Na Figura 20 é possível observar a resposta do filtro finito resultante no domínio da frequência em (a) e no domínio do tempo em (b). A multiplicação ou truncamento entre sinais no domínio do tempo é equivalente ao processo convolução no domínio da frequência.

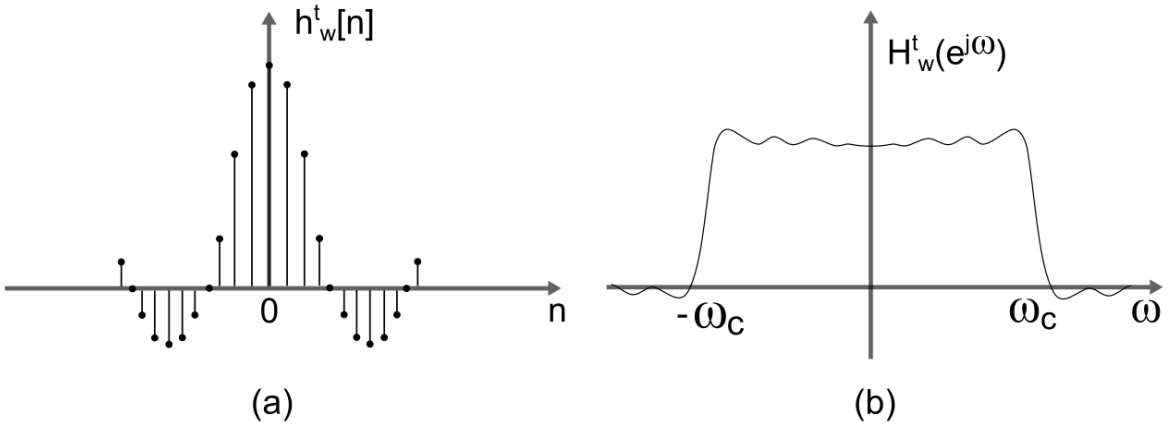


Figura 20 – Resposta do filtro FIR no (a) domínio do tempo e (b) no domínio da frequência.
Fonte: Fonte: Adaptado de Corinthios (2009).

Observando a forma da janela, no domínio do tempo e no da frequência, na Figura 19 e a forma do filtro resultante na Figura 20, é possível analisar algumas dependências. No domínio da frequência é possível observar que o filtro resultante apresenta algumas ondulações similares, tanto na banda de passagem quanto na banda de corte. Essas ondulações são explicadas pelo fenômeno de Gibbs e dependem do tipo de janela escolhida. As mesmas ondulações presentes na forma da janela aparecerão na resposta em frequência do filtro resultante (PROAKIS; MANOLAKIS, 2007).

Além da ordem do filtro, a largura do lóbulo principal da janela influencia na largura da faixa de transição do filtro resultante. Uma janela ideal apresentaria um lóbulo principal muito estreito, isto é, seria um impulso, em que multiplicado pela resposta ideal se obteria um filtro sem transição entre banda passante e banda de corte. Consequentemente, isso implicaria em uma janela com infinitas amostras não-nulas, que por sua vez não é possível aplicar na prática (NALON, 2009).

O filtro FIR obtido na Figura 20 é um filtro não-causal, pois apresenta valores não-nulos antes de zero. Para tornar o filtro causal é necessário realizar um deslocamento no tempo. Esse deslocamento no tempo implica em uma fase linear no domínio da frequência e sua inclinação é dependente do número de amostras (PROAKIS; MANOLAKIS, 2007).

A Tabela 2 apresenta um comparativo entre algumas funções de janelas disponíveis $w[n]$ para $0 \leq n < N$. Na Figura 21 é possível observar os espectros dessas janelas e verificar algumas das características apresentadas na tabela.

Tabela 2 – Comparativo entre algumas janelas.

Janela	$w[n]$	Lóbulo principal	Faixa de transição ($\Delta\omega$)	Pico do lóbulo lateral (dB)
Retangular	1	$2\pi/N$	$0{,}91\pi/N$	-21 dB
Hanning	$0{,}5 - 0{,}5 \cos(2\pi n/N-1)$	$4\pi/N$	$2{,}51\pi/N$	-44 dB
Hamming	$0{,}54 - 0{,}46 \cos(2\pi n/N-1)$	$4\pi/N$	$3{,}14\pi/N$	-53 dB
Blackman	$0{,}42 - 0{,}5 \cos(2\pi n/N-1) + 0{,}08 \cos(4\pi n/N-1)$	$6\pi/N$	$4{,}60\pi/N$	-74 dB

Fonte: Adaptado de Nalon (2009).

Como é possível observar na Figura 21, a janela que apresenta o lóbulo central mais estreito é a janela retangular. Porém, o lóbulo secundário encontra-se em -13 dB e seu lóbulos sucessores permanecem em -21 dB. Essa característica da

janela retangular pode não ser interessante para algumas aplicações que exigem uma atenuação maior (NALON, 2009).

As janelas de Hamming e de Hanning apresentam lóbulo principal maior quando comparado a da janela retangular. Entretanto, a diferença entre o lóbulo principal e o secundários dessas janelas são de -43 dB e -32 dB, respectivamente. Ainda, os lóbulos sucessores na janela de Hanning tem um decaimento maior com a frequência. A janela de Blackman é a que apresenta o maior lóbulo principal, porém também é a que apresenta a maior diferença entre os lóbulos com -57 dB (PROAKIS; MANOLAKIS, 2007).

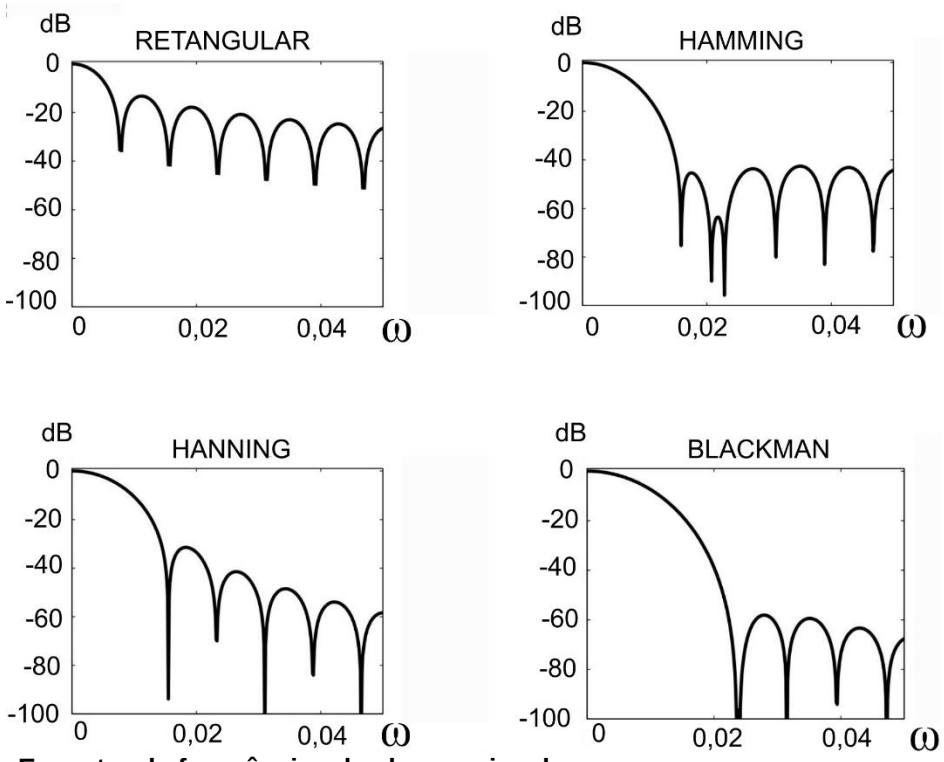


Figura 21 – Espectro de frequências de algumas janelas.
Fonte: Adaptado de Higuti.

Como tratado na seção anterior, os filtros FIR apresentam os valores de seus coeficientes iguais ao valores não-nulos da resposta ao impulso do filtro. Portanto, para exemplificação da obtenção dos coeficientes, utilizando a resposta ideal do filtro no tempo e a janela retangular e considerando o filtro com 3 *taps* (ordem igual a 2) e frequência de corte π/5 rad, tem-se que:

- A janela retangular é definida como $w[n] = 1$, se $-N/2 \leq n \leq N/2$.
- As amostras do filtro FIR ideal obtido podem ser calculados por meio de (21), para as 3 amostras.

- O filtro FIR é obtido por meio do truncamento h[n] = h_D[n]w[n].

Para o exemplo, em que h[n] = h_D[n], então:

$$h[-1] = \frac{sen((\pi/5).(-1))}{\pi(-1)} = 0{,}1871$$

$$h[0] = \frac{\pi/5}{\pi} = 0{,}2$$

$$h[1] = \frac{sen((\pi/5).(1))}{\pi(1)} = 0{,}1871$$

Estes coeficientes obtidos são para um filtro FIR com fase não linear. Para um filtro de fase linear é necessário realizar um deslocamento no tempo de modo que os coeficientes di do filtro sejam iguais a h[n – N/2]. Isso fará com a primeira amostra não-nula ocorra em n=0 (NALON, 2009). Para o exemplo, tem-se:

$$d_0 = h[0-1] = 0{,}1871$$

$$d_1 = h[1-1] = 0{,}2$$

$$d_2 = h[2-1] = 0{,}1871$$

2.5.2 Projeto de Filtros IIR

Para o projeto de filtros IIR, a abordagem mais comum envolve a adaptação ou transformação de um filtro de tempo contínuo para um filtro de tempo discreto. Esse tipo de abordagem de projeto de filtro IIR é razoável visto que as técnicas utilizadas para projetar filtros analógicos são avançadas e, quando bem dimensionadas, apresentam resultados desejáveis. Além disso, dimensionar um filtro analógico é relativamente simples quando comparado ao projeto discreto desse filtro (OPPENHEIM; SCHAFER, 1998).

A transformação dos domínios pode ser realizada por meio do mapeamento de polos do plano complexo *s* (contínuo) para o plano complexo *z* (discreto). A Figura 22 apresenta o mapeamento de polos. Esse mapeamento pode ser realizado por diversas técnicas, destacando-se a técnica de invariância do impulso e a transformação bilinear (NALON, 2009).

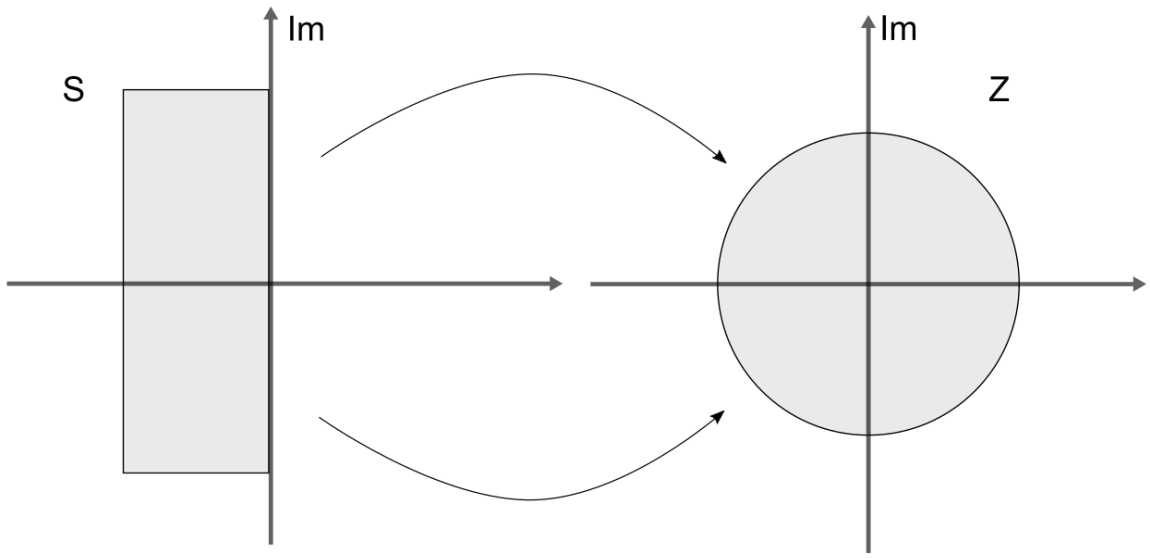


Figura 22 – Mapeamento de polos do plano S para o plano Z.
Fonte: Adaptado de Nalon (2009).

A técnica de invariância do impulso é uma técnica simples, que consiste em fazer o mapeamento por meio de $z= e^{sTa}$, porém, acarreta no efeito de *aliasing,* resultado da operação de amostragem (OPPENHEIM; SCHAFER, 1998).

A técnica de transformação bilinear elimina o *aliasing*, entretanto exige uma complexidade maior, pois consiste no mapeamento algébrico entre o plano *s* e o plano *z*, no qual todo eixo imaginário de *s* é mapeado para o círculo de raio unitário por meio de:

$$s = \frac{2}{T_a}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \tag{27}$$

A decisão de qual a melhor técnica de mapeamento depende do requisito do problema ao qual está sendo projetado (NALON, 2009).

2.6 CONSIDERAÇÕES FINAIS

O Capítulo 2 apresentou a fundamentação teórica para o desenvolvimento desse trabalho. Iniciou-se com uma breve introdução a respeito das ondas sonoras, expondo sobre suas características e qualidades fisiológicas, nas quais nos permitem distinguir uma onda das outras. Além disso, foram apresentados alguns fenômenos capazes de modificar esse tipo de sinal, como sob influência dos sinais indesejáveis

e o fenômeno da equalização de som, que permite selecionar determinadas frequências desse sinal. Também, buscou-se abordar a decomposição de sinais complexos, como o sinal de áudio, em um somatório de senóides, resultando no espectro de frequências.

Esse capítulo também aborda cada etapa do processamento digital de sinais. Iniciando-se na apresentação de algumas definições importantes, como sinais e sistemas, amostragem de sinais e *Aliasing*, seguido com a explicação do processo de conversão de um sinal analógico para um sinal digital. Para finalizar a explanação sobre as etapas, apresentou-se a respeito da reconstrução de sinais, explicando cada estágio desse processo.

Ainda, apresentou-se sobre os sistemas, denominados filtros, capazes de permitir a passagem de determinadas faixas de frequência e rejeitar outras. Classificou-se esses sistemas quanto a tecnologia empregada, analógica ou digital, e explanou-se sobre os filtros que intitulam esse trabalho, os filtros digitais. Dentre os filtros digitais apresentou-se dois tipos classificados conforme sua resposta a uma entrada do tipo impulso. O filtro denominado FIR apresenta uma resposta finita ao impulso e por esse motivo é um sistema sempre estável. Enquanto os filtros denominados IIR apresentam uma resposta infinita ao impulso, que embora não sejam sistemas sempre estáveis, podem apresentar uma seletividade de frequência mais acentuada comparada aos filtros FIR.

Por fim, apresentou-se a teoria dos projetos de filtros digitais, abordando sobre o método do janelamento para projeto dos filtros FIR e a transformação bilinear para o projeto dos filtros IIR.

3 MATERIAIS E MÉTODOS

Nesse capítulo apresenta-se o projeto dos filtros utilizando funções do Matlab, baseadas na teoria apresentada no Capítulo 2. Além disso, esse capítulo apresenta os componentes utilizados para implementação dos filtros no *kit* de desenvolvimento microcontrolado, como também a metodologia utilizada para tal.

3.1 PROJETO E SIMULAÇÃO DOS FILTROS

Utilizando o *software* matemático Matlab®, desenvolvido pela corporação privada MathWorks, projetou-se os filtros FIR e IIR utilizados nesse trabalho. As funções utilizadas são descritas a seguir e são baseadas na teoria apresentada anteriormente.

3.1.1 Filtros FIR

Para o projeto de filtros FIR utilizou-se a função `fir1` do Matlab®. Essa função implementa o método de janelamento para o desenvolvimento de um filtro FIR com fase linear, retornando os valores dos coeficientes d_i do filtro. Nela é possível desenvolver filtros com as configurações padrões do tipo passa-baixas, passa-altas, passa-faixa e rejeita-faixa.

A função `fir1` aceita como parâmetros de entrada a ordem do filtro, a frequência de corte normalizada, o tipo de configuração e o tipo de janela a ser utilizada. Caso os parâmetros do tipo de configuração e tipo de janela não forem determinados, como padrão o Matlab® retorna os coeficientes de um filtro do tipo passa-baixas e aplica a janela de Hamming para o projeto. Ainda, vale ressaltar que por convenção da MathWorks, a frequência de corte corresponde a -6 dB e não em -3 dB como outros métodos (MATHWORKS, 2017).

Levando em consideração o espectro do sinal de áudio escolhido para simulação, de modo que ficasse evidente as características de corte e passagem dos filtros, optou-se por fixar os parâmetros dados abaixo para os filtros FIR:

- Filtro passa-baixa com f_c = 1500 Hz e N=15;
- Filtro passa-alta com f_c = 6000 Hz e N=16;
- Filtro passa-banda com f_{c1} =4000 Hz, f_{c2} = 7000 Hz e N=20;
- Filtro rejeita-banda com f_{c1} =1500 Hz, f_{c2} = 6000 Hz e N=20.

Todos os filtros foram projetados aplicando a janela de Hamming. Como apresentada no Capítulo 2.5.1 essa janela apresenta algumas características relevantes para filtragem de sinais quando comparada a janela retangular, principalmente quanto a atenuação na faixa de rejeição. Além disso, essa janela é utilizada como padrão pelo Matlab®.

Para o projeto do filtro FIR passa-alta utilizou-se como base o código do Apêndice A, porém realizando as modificações no trecho de código presente no Apêndice B. Para o projeto do filtro FIR passa-banda também utilizou-se como base o código do Apêndice A, porém realizando as alterações no trecho de código presente no Apêndice C. Finalmente, para o projeto do filtro FIR rejeita-banda utilizou-se como base o código do Apêndice A, porém realizando as modificações no trecho de código presente no Apêndice D.

3.1.2 Filtros IIR

No *software* Matlab® para o projeto dos filtros IIR utilizou-se a função `buttord` que retorna a menor ordem de um filtro digital do tipo Butterworth, além da frequência de corte. Essa função opera no domínio analógico e, por esse motivo, primeiramente ela converte os parâmetros de entrada para o domínio s, estima a ordem e a frequência e na sequência converte novamente para o domínio z (MATHWORKS, 2017).

A função `buttord` aceita como parâmetros de entrada a frequência de corte normalizada Wp, frequência da banda de rejeição normalizada Ws, o valor do *ripple* ap, na banda de passagem em dB e o valor da atenuação as, em dB na banda de rejeição. Os parâmetros de entrada da função para um filtro passa-baixa podem ser visto na Figura 23.

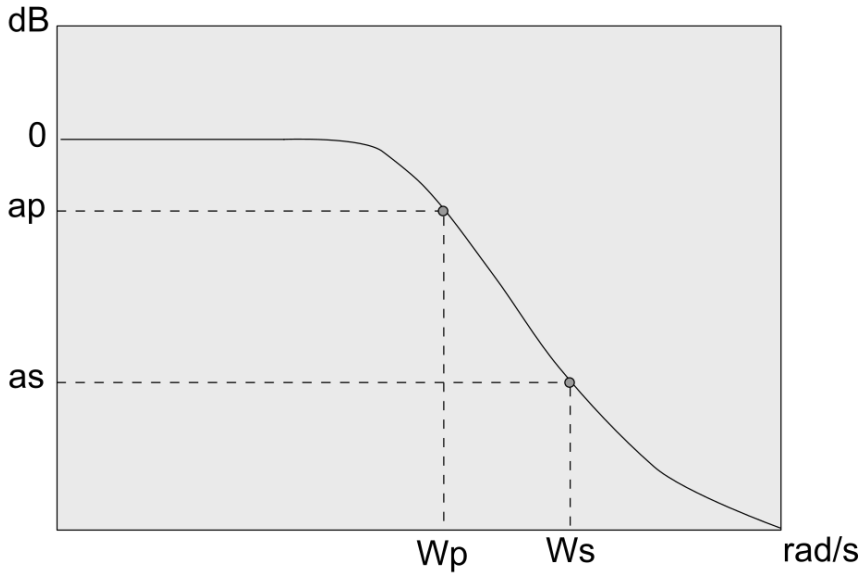


Figura 23 - Parâmetros de entrada para função buttord.
Fonte: Autoria própria.

Com a ordem e a frequência obtidas pela função `buttord`, utilizou-se a função `butter` que retorna os coeficientes a_i e b_i de um filtro Butterworth digital. Essa função utiliza um algoritmo de cinco passos. Primeiramente ela encontra os zeros, polos e ganhos do protótipo passa-baixa analógico, converte essas informações para espaço de estados, se necessário usando espaço de estados transforma o filtro passa-baixa no tipo determinado (passa-alta, passa-banda, rejeita-banda). Na sequência, utilizando a transformação bilinear com pré-controle de frequência, transforma o filtro analógico em digital. Finalmente, transforma o filtro de espaço de estado para sua função transferência (MATHWORKS, 2017). Como o ajuste da frequência é feita de maneira cuidadosa, a função `butter` permite que os filtros analógicos e os filtros digitais apresentem a mesma magnitude de resposta em frequência.

Também de modo a evidenciar as características de corte e passagem dos filtros IIR optou-se por fixar as frequências de passagem e parada dadas a seguir.

- Filtro passa-baixa com f_p = 1500 Hz e f_s= 3000 Hz;
- Filtro passa-alta com f_c = 6000 Hz e f_s= 3500 Hz;
- Filtro passa-banda com f_{c1} =4000 Hz, f_{c2} = 7000 Hz, f_{s1}= 1500 Hz e f_{s1}= 9500 Hz;
- Filtro rejeita-banda com f_{c1} =1500 Hz, f_{c2} = 6000 Hz, f_{s1}= 50 Hz e f_{s1}= 8000 Hz;

Determinando como atenuação de – 3 dB na banda passante e – 40 dB na frequência de parada para todos os filtros.

O código utilizado para o projeto do filtro IIR passa-baixa encontra-se no Apêndice E, o qual foi utilizado como base para o projeto dos demais filtros. O Apêndice F apresenta o código utilizado no projeto do filtro IIR passa-alta. Para projeto do filtro IIR passa-banda também utilizou-se como base o código do Apêndice E, entretanto para esse tipo de filtro foi necessário especificar as duas frequências de corte e as duas frequências de parada. As modificações realizadas no trecho de código base encontram-se presente no Apêndice G. Por fim, o Apêndice H contém o código para o projeto do filtro IIR rejeita-banda.

Para simulações realizadas dos filtros IIR utilizou-se os mesmos sinais de entrada, áudio e multisenoidal, que para os filtros FIR.

3.2 IMPLEMENTAÇÃO DOS FILTROS EM *HARDWARE*

A implementação dos filtros projetados nesse trabalho utiliza o *kit* de desenvolvimento microcontrolado STM32F4 – Discovery que apresenta algumas características interessantes para aplicação de processamento de sinais. Esse capítulo busca descrever os materiais e métodos empregados para implementar os filtros digitais projetados.

3.2.1 Microcontrolador STM32F407VG

O desempenho do microcontrolador STM32F407VG deve-se ao seu núcleo de processamento ARM Cortex-M4F de 32 *bits* podendo operar a uma frequência de 168 MHz. Os núcleos de processamento Cortex-M4 permitem o controle e o processamento de sinais de forma fácil e eficiente, pois apresentam uma combinação de funcionalidades de processamento com alta eficiência e baixa potência. O núcleo de um processador M4 é mostrado na Figura 24, nela é possível observar algumas características.

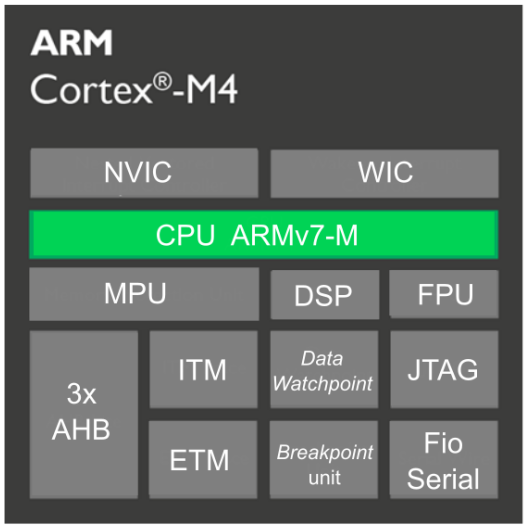


Figura 24 – Diagrama de blocos núcleo ARM Cortex-M4.
Fonte: Adaptado de ARM Limited (2017).

A unidade central de processamento (CPU do inglês *Central Processing Unit*) que executada as instruções do programa, apresenta arquitetura ARMv7-M. Essa é uma arquitetura do tipo *Harvard* de dados, isso é, possui duas memórias distintas, uma destinada aos dados e outra destinada as instruções, além disso, é caracterizada por dois barramentos internos, um de dados e outro de instruções. Como apresenta os barramentos distintos, executa simultaneamente a busca de dados e instruções, acessando paralelamente as memórias, assegurando assim sua principal característica, a rapidez de execução de uma instrução comparada a arquitetura de *von Neumann* (STALLINGS, 2002).

A unidade de ponto flutuante (FPU do inglês *Float Point Unit*) pode acelerar operações de ponto flutuante em relação a operações com ponto fixo. A unidade de proteção de memória (MPU do inglês *Memory Protect Unit*) permite o acesso apenas para algumas áreas específicas da memória, garantindo maior confiabilidade do *software*, já que evita acessos que podem corromper dados críticos.

O controlador de interrupção com vetor integrado (NVIC do inglês *Nested Vectored Interrupt Controller*) gerencia as interrupções do processador, capaz de suportar até 240 interrupções com até 256 níveis de prioridade. Na entrada de uma interrupção o *hardware,* salva o estado do processador e ao sair o restaura. Além disso, o NVIC suporta o encadeamento de interrupções (ARM LIMITED, 2010).

O WIC (do inglês *Wake-up Interrupt Controller*) é utilizado para detectar uma interrupção e despertar o processador do modo de hibernação profunda, quando

a CPU e a memória RAM são desativadas enquanto um evento importante não ocorre, para economia de energia. As instruções de processamento digital de sinais são representadas pelo bloco DSP (do inglês *Digital Signal Processing*), na qual são operações matemáticas capazes de manipular e processar os sinais (ARM LIMITED, 2010).

Os blocos chamados ITM *trace* e ETM *trace* são recursos disponibilizados para depuração do código de modo a auxiliar o desenvolvedor. O ITM (do inglês *Instrumentation Trace Macrocell)* permite criar um canal de depuração onde o microcontrolador pode fornecer mensagens de eventos de rastreamento (ARM LIMITED, 2010). Enquanto que o ETM (do inglês *Embedded Trace Macrocell*) permite que o *hardware* possa transmitir o passo a passo de como ele percorre o fluxo das instruções (PLS DEVELOPMENT TOOLS, 2017). Outro recurso para depuração é o *Data Watchpoint* que permite interromper a simulação para uma localização direcionada, e o *Breakpoint unit* é registro do ponto de interrupção (ARM LIMITED, 2010).

O JTAG (do inglês *Joint Test Access Group*) é uma interface que permite programar e testar circuitos digitais. Com esse recurso é possível executar o programa passo a passo, visualizar as variáveis utilizadas e valores na memória e ainda gravar esse programa no dispositivo microcontrolado (ARM LIMITED, 2017). Enquanto, o *Serial Wire* fornece rastreamento de dados, rastreamento de eventos e informação de rastreamento de instrumentação em um formato de saída de fio serial.

O bloco 3xAHB representa os três barramentos de alto desempenho do microcontrolador que permite a conexão entre os seus blocos funcionais. Esses três barramentos e suas conexões podem ser vistos no Anexo A.

Com todos os recursos disponibilizados pelo núcleo Cortex-M4 em conjunto com a capacidade das memórias *Flash* e SRAM, o STM32F407 apresenta condições para implementação dos filtros digitais projetados nesse trabalho.

3.2.2 Descrição do sistema

A implementação do trabalho busca demostrar a eficiência dos filtros na prática e por meio dos resultados realizar a comparação com os resultados da simulação e dos filtros ideais apresentados no capítulo 2.

Para implementar os filtros digitais projetados no *kit* utilizado é necessário compreender as etapas de funcionamento dados na Figura 25. Na figura é possível observar que o sistema implementado é composto por três módulos básicos: a aquisição de dados, o processamento que realizará a filtragem do sinal e finalmente a reconstrução do sinal.

Figura 25 – Etapas de funcionamento da implementação.
Fonte: Autoria própria.

Cada etapa da figura acima implementa diferentes processos e operações. A Figura 26 apresenta um diagrama de como os recursos de hardware estão relacionados na aplicação proposta do sistema descrito. Como aquisição de dados, tem-se que um arquivo de áudio é lido a partir de um cartão de memória SD (do inglês Secure Digital), o processamento do sinal adquirido ocorre na CPU do microcontrolador e por fim o sinal processado é reproduzido a partir de um CODEC.

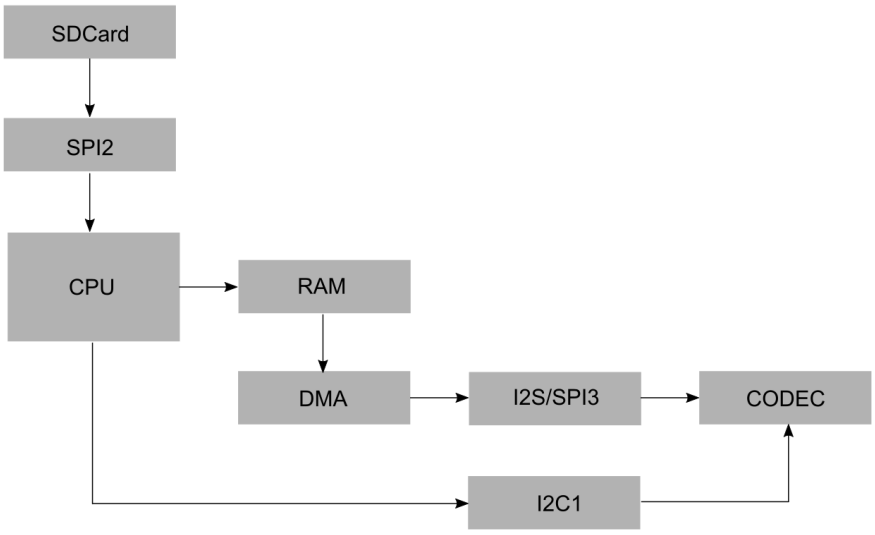


Figura 26 – Diagrama dos recursos de *hardware* utilizados.
Fonte: Autoria própria.

Para o gerenciamento das tarefas assíncronas da aplicação utilizou-se a FreeRTOS. Um sistema de tempo real - RTOS (do inglês Real Time Operating System) é utilizado para gerenciar os recursos do sistema, como processador e memória, disponibilizando ou não para as tarefas. Além disso, o RTOS apresenta a

capacidade de atender restrições de tempo das atividades, garantindo que elas comecem e terminem no prazo (ALMEIDA, 2015).

A camada de *software* responsável pelo gerenciamento e coordenação da execução das tarefas é chamado de *kernel*. Esse kernel utiliza critérios para realizar esse gerenciamento, como criticidade de uma tarefa, tempo de execução pré-definido, prioridades, entre outros (ALMEIDA, 2015). Nesse trabalho utilizou-se como critério o conceito de prioridade. Quando a tarefa com maior prioridade estiver pronta para ser executada o *kernel* disponibiliza os recursos para ela.

Uma tarefa é um trecho de código executável pelo sistema operacional embarcado (ALMEIDA, 2015). Na Figura 27 é possível observar três das quatro tarefas criadas no trabalho. A tarefa *SDCard_Task* é a tarefa onde são executadas as configurações do periférico SPI2, as configurações das entradas e saídas – GPIO (do inglês *General Purpose Input/Output*) referentes a esse periférico, além do *link* entre o *driver* desenvolvido e o FATFs, que serão tratados posteriormente. Essa tarefa foi configurada com a maior prioridade do trabalho.

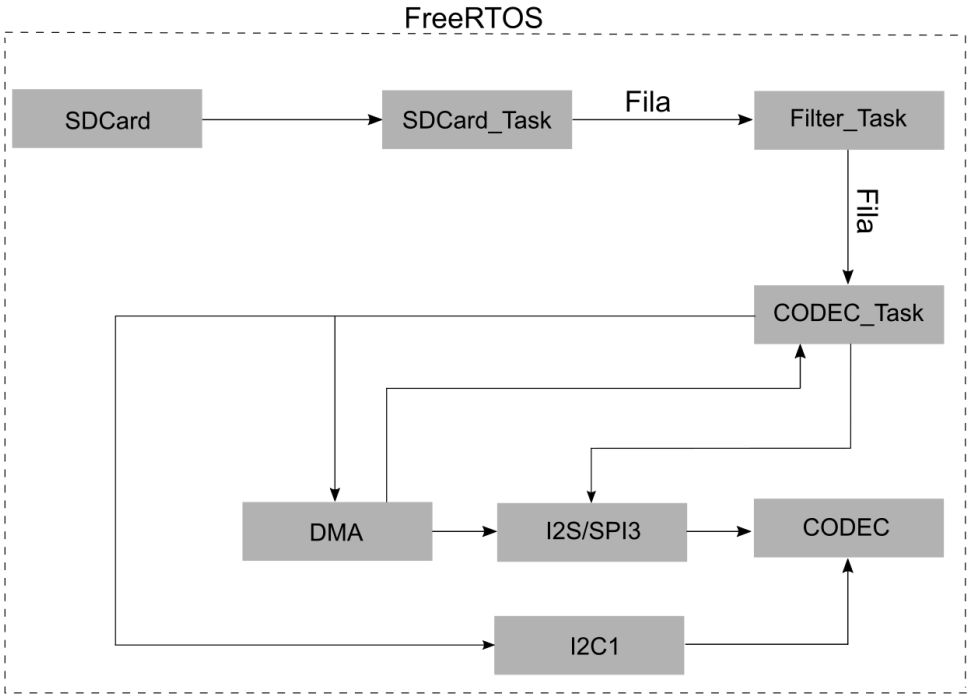


Figura 27 - Diagrama de execução das tarefas do sistema.
Fonte: Autoria própria.

Na tarefa *Filter_task* ocorre o processamento dos dados obtidos, ou seja, onde ocorre a filtragem dos sinais de entrada, essa tarefa foi definida como a segunda

maior prioridade do sistema. Já a tarefa *CODEC_task* realiza as configurações dos periféricos DMA, I2S e I2C, além das configurações do CODEC, para reprodução dos dados processados na tarefa de filtragem. Essa tarefa foi definida como a terceira maior prioridade do sistema.

Por fim, também implementou-se uma tarefa para piscar um *led* do kit, denominada *Task1* que não se encontra no diagrama acima pois foi implementada somente com fim de verificação do funcionamento da FreeRTOS. Essa tarefa foi configurada para apresentar a menor prioridade do sistema.

3.2.3 Descrição dos componentes

O *kit* de desenvolvimento microcontrolado STM32F4 – Discovery, que integra o microcontrolador STM32F407VG, foi desenvolvido para aplicações em que são necessários altos níveis de integração e desempenho, grande quantidade de periféricos e memória embutida.

Nesse trabalho os recursos de *hardware* do *kit* utilizados podem ser visualizados na Figura 26.

Para aquisição do sinal utiliza-se o periférico SPI2, que realiza a comunicação do microcontrolador com o cartão de memória por meio do protocolo SPI (do inglês Serial Peripheral Interface). Os dados obtidos da aquisição são processados na CPU do microcontrolador e encontram-se salvos na memória RAM do dispositivo.

Os dados armazenados na memória RAM são acessados pelo periférico *Direct Memory Access* – DMA. O DMA permite que as transferências dos dados sejam transferidas de modo constante, da CPU para o periférico SPI3. Em um ciclo de *clock* a DMA acessa a memória de origem do dado e em outro ciclo envia o dado para o destinatário.

O periférico SPI3 utiliza o protocolo de comunicação I^2S (do inglês *Inter-IC Sound*) para interconectar dispositivos de áudio digital. Esse periférico serve de ponte de conexão entre o DMA e o CODEC do *kit* de desenvolvimento.

O CODEC é o circuito lógico CS43L22 apresentado no Anexo B. Esse circuito é um DAC estéreo altamente integrado e de baixa potência com

amplificadores de alto falante para fone de ouvido e do tipo Classe D. O controle de operação do CS43L22 é realizada pelo periférico I2C1 do *kit*.

Para o acesso aos periféricos do microcontrolador descritos acima, utilizou-se a biblioteca CMSIS fornecida pela ARM®. Além disso, utilizou-se uma biblioteca para realizar a configuração e uso do CODEC.

A implementação em *software* também incluiu o uso do RTOS, para gerenciar mais facilmente as tarefas assíncronas existentes. O RTOS utilizado nesse trabalho é a FreeRTOS que apresenta código aberto e é muito utilizado no mundo.

A IDE (*Integrated Development Environment*) EmBitz® foi empregada para o desenvolvimento de *firmware* para o microcontrolador STM32F407. Essa ferramenta permite, também, gravar e depurar o *firmware* no microcontrolador.

3.2.4 Descrição da metodologia

A etapa inicial de leitura de dados é realizada por meio da comunicação do microcontrolador com um cartão de memória SD. O cartão contém um arquivo padrão de áudio no formato WAVE, o mesmo utilizado nas simulações, para que seja possível verificar as respostas obtidas na prática. Um arquivo WAVE contém um sinal de áudio digital, amostrado a uma frequência de 44100 Hz e apresenta 16 *bits* por amostra.

Para o STM32F407 se comunicar com o cartão é necessário implementar as camadas de *software* mostradas na Figura 28. Uma aplicação que necessita armazenar ou receber dados de um cartão de memória SD precisa utilizar comandos a nível de arquivo, como abrir, ler e gravar, para acessar arquivos específicos dentro do sistema de arquivo do cartão SD.

O sistema de alocação de arquivos FAT (do inglês *File Allocation Table*) é um protocolo que estabelece uma organização e uma forma de armazenamento de arquivos em um dispositivo de armazenamento, unidade de disco ou um dispositivo de memória (STMICROELECTRONICS , 2014).

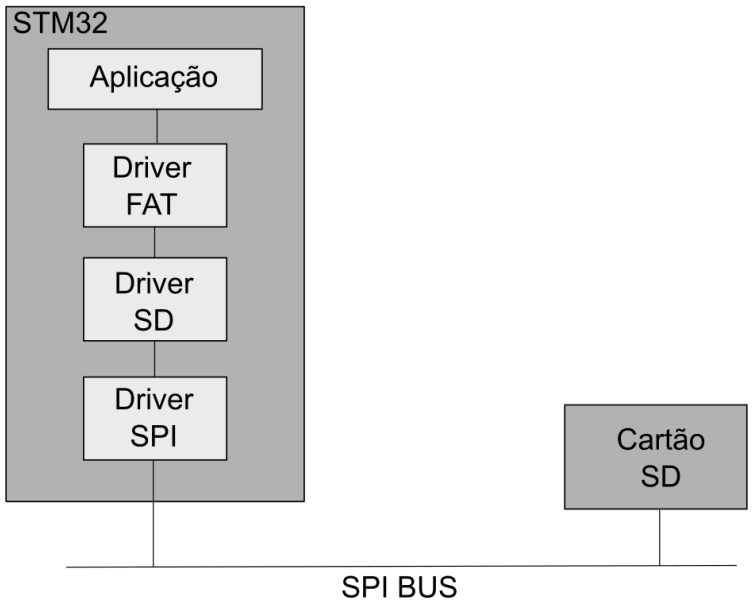


Figura 28 – Etapas de acesso ao cartão SD.
Fonte: Adaptado de Brown (2016).

Para acessar um sistema de arquivo FAT por meio de comandos é necessário um *driver*. O FatFs é um módulo genérico que permite acessar os volumes FAT, independente da arquitetura de *hardware*. Esse módulo serve como mediador, ou *middleware*, em que são oferecidas funções como abrir, fechar, ler e escrever (STMICROELECTRONICS , 2014).

O sistema de arquivo lê e escreve emitindo comandos de blocos, porém sem qualquer conhecimento como esses comandos são implementados. Quem implementa esses comandos é um *driver* separado, o *SD driver* mostrado na Figura 28. O *SD driver* implementa cinco funções para suportar o FATFs (inicialização, *status*, leitura, escrita e controle) e utiliza o driver da interface SPI2 para se comunicar com o cartão SD. As funções do *SD driver* podem ser vistas no Apêndice J.

A tarefa *SDCard_Task*, apresentada no Apêndice K, cria o link entre a FATFs e o *driver* desenvolvido, configura a SPI2 e realiza, após a montagem do cartão, a leitura dos dados do cartão.

A configuração do periférico SPI2 e das GPIO correspondentes a ele podem ser visualizadas no Apêndice L. Na estrutura dos pinos são especificados quais deles são utilizados, modo de operação, velocidade, configuração do tipo de saída e do resistor utilizado. Na estrutura do SPI2 são determinados a direção dos dados, modo de operação, tamanho dos dados, polaridade e fase do *clock*, frequência

de operação, determinação de transferência do dado, pelo *bit* mais significativo ou menos significativo Ainda nas configurações, o periférico é inicializado com as especificações dada e o clock para o periférico e para o barramento é habilitado.

O protocolo de comunicação com o cartão SD é baseado em transações que começam com um código de comando, opcionalmente seguido por parâmetros e para finalizar uma transferência de dados, leitura ou escrita. Cada transação que começa com um comando do microcontrolador para o cartão é seguido de uma resposta do cartão para o microcontrolador.

Na Tabela 3 são apresentados alguns comandos para acesso no cartão SD, como iniciar e reiniciar, parâmetros de leitura/gravação e blocos de dados de leitura/gravação (BROWN, 2016). Vale ressaltar que para cada versão de cartão SD existe uma sequência de inicialização própria, sendo que pode haver incompatibilidade nos comandos aceitos por determinadas versões.

Entre as operações realizadas com o cartão SD são acionados atrasos em função dos tempos específicos para que ele realize determinadas tarefas, como: *reset*, entrada e saída de modo de espera, leitura e escrita de blocos, entre outras. Após a inicialização correta do cartão e o envio correto dos comandos para leitura de dados, a aquisição da informação de áudio no cartão pode ser realizada pela CPU.

Tabela 3 – Alguns comandos do cartão SD.

Comando	Descrição
CMD0	Reinicia o cartão SD
CMD1	Inicializa o cartão
CMD8	Verifica nível de tensão de operação, apenas para SDC V2
CMD12	Status de erro
CMD16	Configura o tamanho do bloco de transferência
CMD17	Lê um bloco
CMD18	Lê múltiplos blocos
CMD58	Lê o registrador OCR
ACMD41	Inicializa o cartão, apenas SDC

Fonte: Adaptado de Brown (2016).

Os dados são armazenados em uma *quee*, que é uma fila utilizada para comunicação entre tarefas em uma FreeRTOS. Quando a fila encontra-se cheia, os recursos são liberados para a tarefa de filtragem, essa tarefa é apresentada no Apêndice M.

Na tarefa de filtragem inicialmente ocorre a inicialização do filtro. Na sequência, os dado retirados da fila passam pela filtragem, os dados referentes ao lado direito são filtrados e armazenados na fila que será transferida para o CODEC, enquanto que os dados referentes ao lado esquerdo passam por essa tarefa sem filtragem, para no final, os dados processados e não processados sejam comparados. Na função de filtragem, para os filtros FIR são utilizados os coeficientes declarados em `filter_taps`, enquanto que para os filtros IIR os coeficientes são declarados como `input_filter_taps` e `output_filter_taps`, para declarar os coeficientes bi e ai, respectivamente. As funções de inicialização, filtragem FIR e filtragem IIR é apresentada no Apêndice N.

Na tarefa *CODEC_task* são realizadas as configurações dos periféricos para o funcionamento da reprodução dos dados do sistema, como pode-se observar no Apêndice O. Além disso, nessa tarefa é inicialmente preenchido uma fila com dois blocos de dados para ser enviados para a DMA e sequencialmente a I^2S e ao CODEC, dando início a transmissão de dados nessas estruturas. Quando o I^2S terminar de transmitir os dois blocos iniciais para o CODEC, ele solicita mais dados ao DMA. O DMA gera uma interrupção na CPU e os novos blocos são preenchidos sendo obtidos diretamente da fila, enviada pela tarefa de filtragem, no tratamento da sua interrupção. O tratamento da interrupção do DMA pode ser observado no Apêndice P.

O DMA permite que as transferências dos dados de áudio sejam transferidas de modo constante, da CPU para o I^2S. Em um ciclo de *clock* a DMA acessa a memória de origem do dado e em outro ciclo envia o dado para o destinatário. Por sua vez, o I^2S (periférico SPI3) transfere os dados diretamente para o CODEC. O controle de operação do CS43L22 é realizada pela I2C1 e esse funciona como um dispositivo escravo, que receberá os dados da I^2S.

A configuração do periférico SPI3 (protocolo I^2S) e das GPIO utilizadas por ele pode ser visto no apêndice Q. Na estrutura dos pinos são especificados quais deles são utilizados, modo de operação, velocidade, configuração do tipo de saída e do resistor utilizado. Na estrutura do I^2S são determinados o modo de operação, padrão utilizado, formato dos dados, saída de clock, frequência de operação e polaridade. Ainda nas configurações, o periférico é inicializado com as especificações dada, o clock para o periférico e para o barramento é habilitado, além da habilitação da interface entre os periféricos DMA e I^2S.

A configuração do periférico I^2C1 e das GPIO utilizadas por ele, pode ser visto no Apêndice R. Na estrutura dos pino são especificados quais deles são utilizados, modo de operação, velocidade, configuração do tipo de saída e do resistor utilizado. Enquanto, na estrutura da I^2C a frequência e modo de operação, modo de *duty cicle*, o endereço, habilitação do *acknowledged* (usado para confirmar se um dado foi recebido pelo dispositivo escravo) e de seu tamanho. Além disso, na função de configuração ocorre a inicialização do periférico com a estrutura especificada e a habilitação do clock no barramento.

No Apêndice S é apresentada a configuração do DMA. Na estrutura é determinado o canal a ser utilizado, o endereço do periférico base e da memória inicial, o tamanho do *buffer*, a largura dos dados da memória e do periférico, a direção dos dados, se o registrador de memória incrementa ou não, modo de operação, quantidade de dados a ser transmitida e se o modo primeiro a entrar, primeiro a sair - FIFO (do inglês *Firt in Firt out*) é habilitado ou não. Além disso, na configuração do DMA é realizada a configuração e inicialização da interrupção pelo DMA. Na estrutura de interrupção são especificados quais dos canais do DMA estão habilitados para interrupção, prioridade e subprioridade. Também, ocorre a limpeza de todas as *flags* de interrupção e a habilitação do *clock* do periférico e da interrupção.

No Apêndice T é apresentado a configuração do CODEC. Essa função de configuração foi extraída de códigos abertos na *internet*. Na configuração do CODEC, incialmente ocorre uma sequência de para inicializada, além disso, seus registradores são ajustados para determinar volume do áudio, fonte de clock, tipo de saída de som (fones de ouvido ou caixa amplificadora), habilitação de reprodução mono ou estéreo, controle de *playback*, entre outros. Todos os controles a serem ajustados nessa configuração são enviados para o CODEC e recebidos pelo microcontrolador por meio do periférico I^2C.

3.3 CONSIDERAÇÕES FINAIS

Esse capítulo apresentou os materiais e métodos utilizados para projeto e implementação dos filtros digitais.

Na descrição do projeto apresentou-se as funções utilizadas, do *software* Matlab®, para obtenção dos coeficientes dos filtros especificados. Para o projeto dos

filtros FIR utilizou-se a função `fir1` que utiliza o método do janelamento para obtenção dos coeficientes. Enquanto no projeto dos filtros IIR, utilizou-se a função `buttord` para obtenção da menor ordem conforme as especificações determinadas. Ainda, para obtenção dos coeficientes do filtro, utilizou-se a função `butter` que emprega o método da transformação bilinear, de um filtro com topologia Butterworth.

Na seção que descreve a implementação dos filtros, primeiramente aborda-se a respeito da escolha do *kit* de desenvolvimento, destacando os recursos interessantes para a aplicação em questão. Na sequência da seção, são descritos os recursos de *hardware* e *software* utilizados para implementação. Por fim, apresentou-se os métodos utilizados para integração dos recursos e implementação dos filtros no microcontrolador.

4 RESULTADOS

A seguir são apresentados os resultados em dois tópicos, o primeiro abordando os resultados do projeto e simulação dos filtros e o segundo os resultados obtidos na prática.

4.1 RESULTADOS DA SIMULAÇÃO

As simulações dos filtros projetados foram realizados por meio do *software* Matlab®. Utilizou-se dois sinais de entrada diferentes, uma música e um sinal multisenoidal, possibilitando visualizar a eficiência de cada filtro para sinais diferentes.

4.1.1 Filtros FIR

O código utilizado para o projeto do filtro FIR passa-baixa encontra-se no Apêndice A. Esse primeiro código utiliza como sinal de entrada uma música. A Figura 29 apresenta a resposta em frequência do filtro FIR passa-baixa. Nessa figura é possível observar que na frequência de corte ($f_{normalizada}$=0,0680 π/rad) determinada a atenuação do filtro é de -1,71 dB e a fase em -92, 29º.

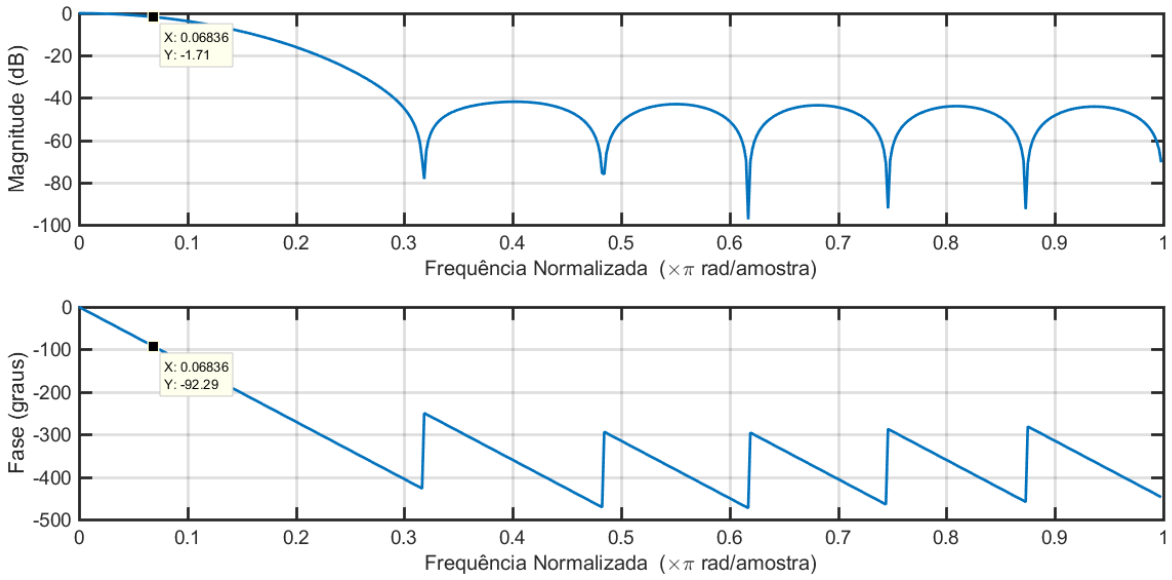


Figura 29 – Resposta em frequência do filtro FIR passa-baixa utilizando janela de Hamming.
Fonte: Autoria própria

A Figura 30 apresenta os espectros do sinal de áudio antes e após a filtragem. As setas indicam a posição da frequência de corte do filtro. Observando a figura é possível notar a atenuação já na frequência de corte e nas frequências acima dela, enquanto que as frequências abaixo de 1500 Hz se encontram sem atenuação.

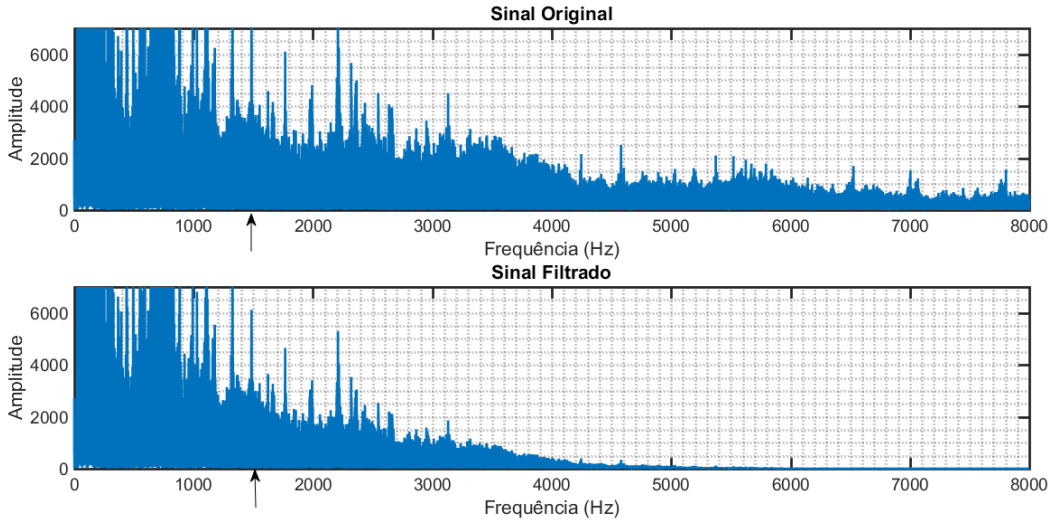


Figura 30 – Passa-baixa: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Mantendo o mesmo filtro obtido na Figura 29, porém alterando o sinal de entrada para um sinal multisenoidal, dado no Apêndice B, obtém-se os espectros dados na Figura 31. Com o sinal multisenoidal fica ainda mais claro a eficiência do filtro passa-baixa projetado.

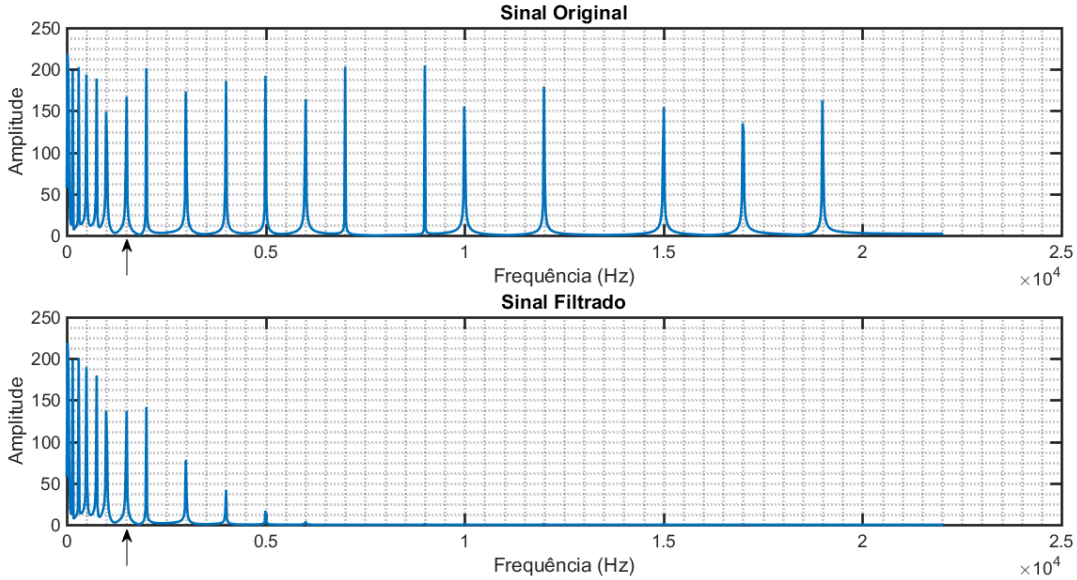


Figura 31 – Passa-baixa: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

Para o projeto do filtro passa-alta, substitui-se no código do Apêndice A as linhas dadas no Apêndice C. O diagrama de frequência do filtro FIR passa-alta é apresentado na Figura 32. Nessa figura é possível observar que na frequência de corte ($f_{normalizada}$=0,2721 π/rad) determinada a atenuação do filtro é de -5,917 dB e a fase em -33, 75º.

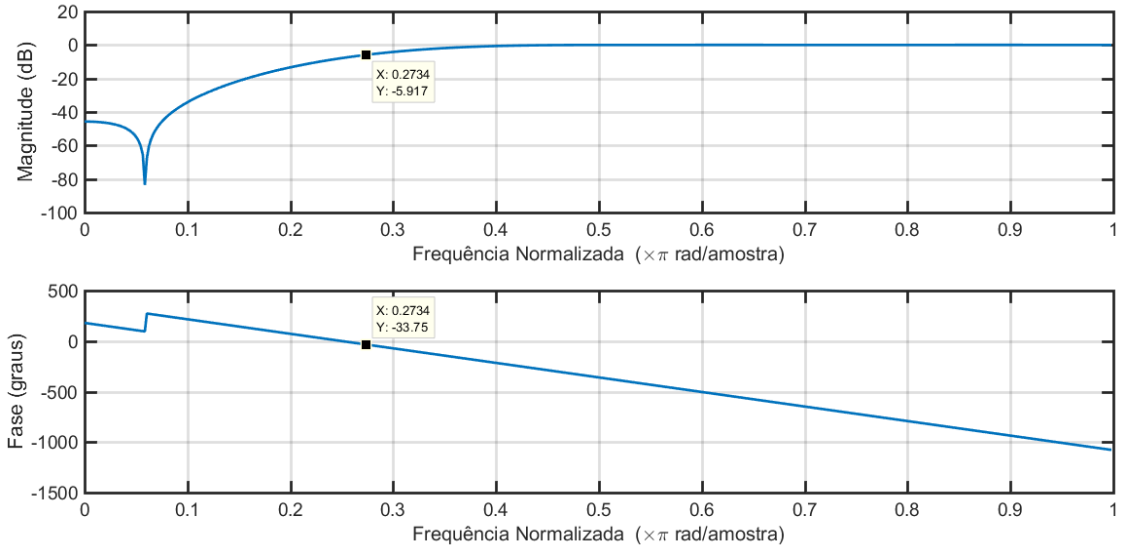


Figura 32 – Resposta em frequência do filtro FIR passa-alta utilizando janela de Hamming.
Fonte: Autoria própria.

Na Figura 33 são apresentados os espectros do sinal de áudio antes e após a filtragem utilizando o filtro passa-alta projetado. Nessa figura, as setas indicam a posição da frequência de corte 6000 Hz.

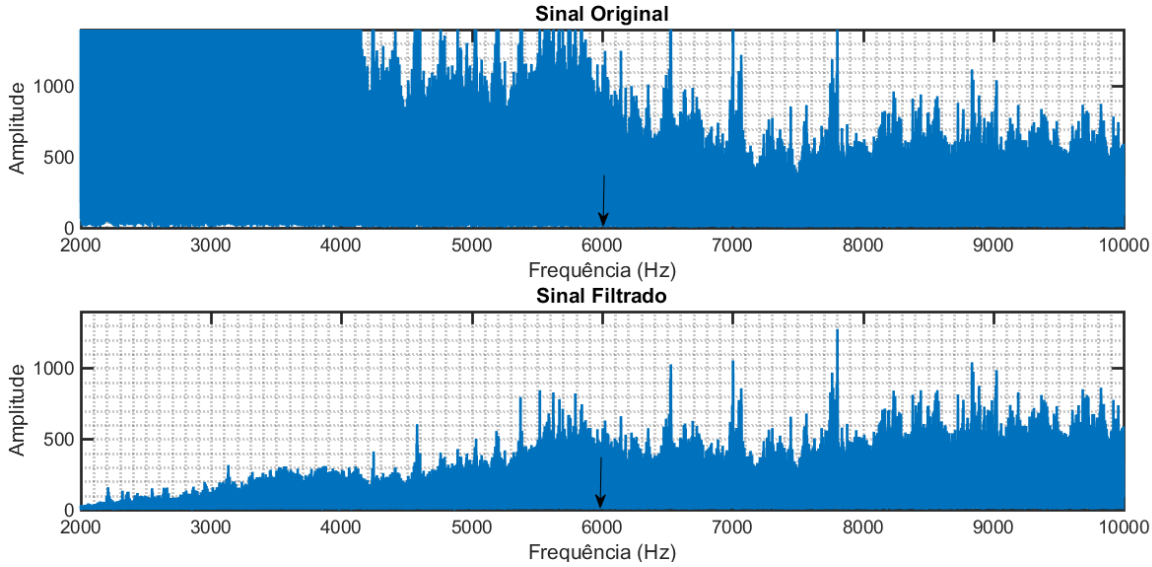


Figura 33 – Passa-alta: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Na Figura 33 é possível observar que na frequência de corte de 6000 Hz o sinal teve uma atenuação, devido aos -5,917 dB. Com o aumento da frequência, os sinais não são mais atenuados devido à seletividade do filtro.

Utilizando o código do sinal multisenoidal do Apêndice B e aplicando o filtro passa-alta dado no Apêndice C, obtêm-se os espectros de frequência dado na Figura 34. Na figura utiliza-se setas para indicação da frequência de corte de 6000 Hz.

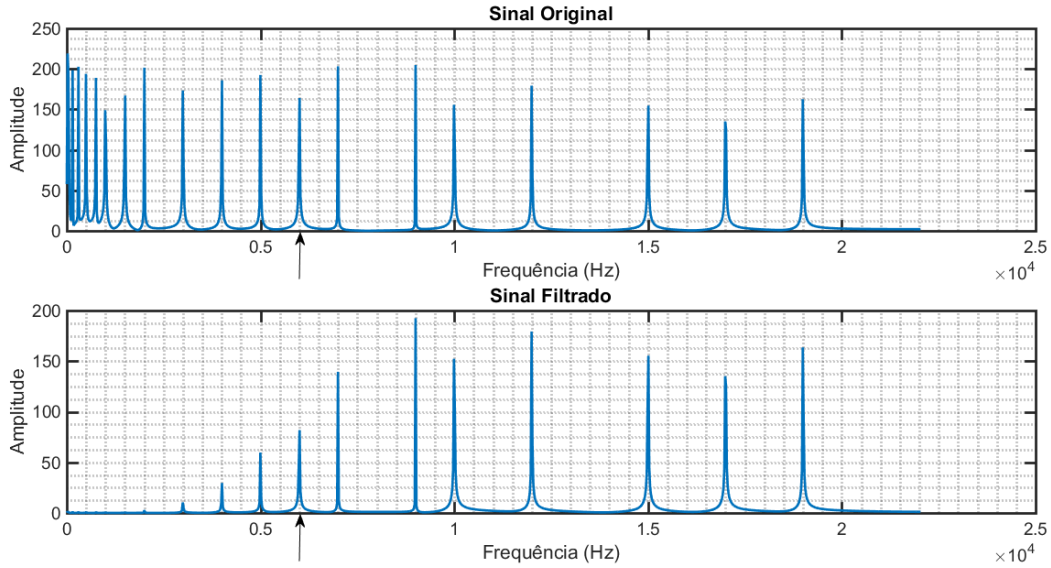


Figura 34 – Passa-alta: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

A Figura 34 confirma o que foi tratado anteriormente. O filtro passa-alta projetado atenua as frequências abaixo de 6000 Hz e permite a passagem das frequências maiores. Ainda assim, na frequência de corte ocorre uma pequena atenuação de -5,917 dB.

Para o projeto do filtro FIR passa-banda, modificou-se o código no Apêndice A pelo código presente no Apêndice D. A Figura 35 apresenta a resposta em frequência do filtro projetado. Como mostra a figura, o filtro apresenta uma atenuação de aproximadamente –2,60 dB nas frequências de corte normalizadas (0,1814 π/rad e 0,3175 π/rad).

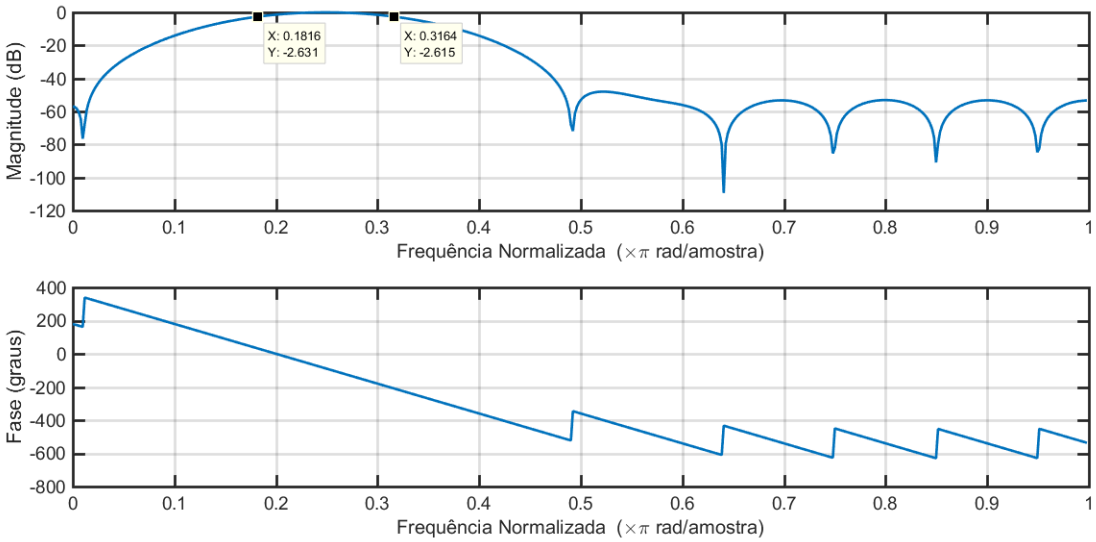


Figura 35 – Resposta em frequência do filtro FIR passa-banda utilizando janela de Hamming.
Fonte: Autoria própria.

A Figura 36 apresenta os espectros do sinal de áudio antes e após a filtragem utilizando o filtro FIR passa-banda projetado. Na figura as setas indicam as frequências de corte (4000 Hz – 7000 Hz) do filtro. Observa-se que as componentes de frequências são atenuadas fora da banda de passagem.

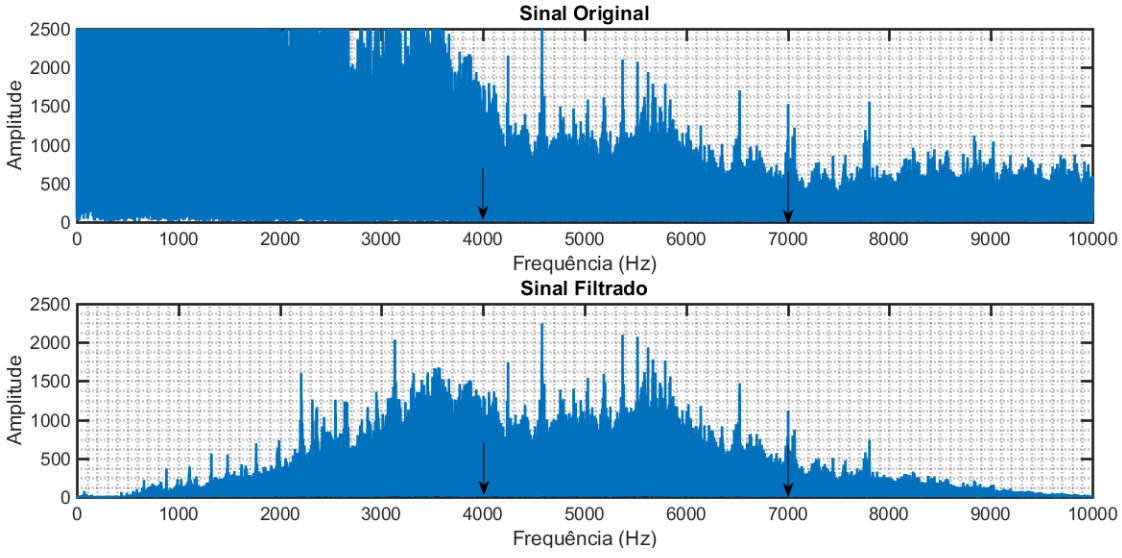


Figura 36 – Passa-Banda: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Utilizando o código do sinal multisenoidal do Apêndice B e aplicando o filtro FIR passa-banda, obtêm-se os espectros de frequência dado na Figura 37. As setas

indicam as frequências de corte do filtro. De acordo com a figura, o filtro passa-banda atenua consideravelmente as frequências fora da banda de passagem (4000 Hz – 7000 Hz). Porém, há uma pequena atenuação na banda de passagem desse filtro, isso ocorre porque ele não é um filtro ideal e apresenta atenuações próximas das frequências de corte.

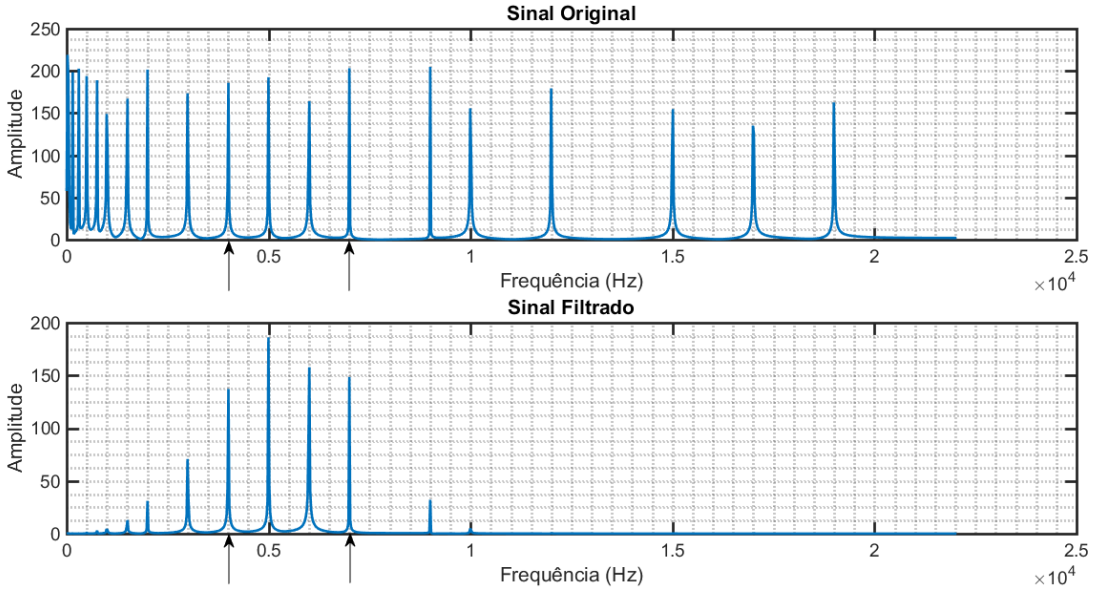


Figura 37 – Passa-Banda: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

O filtro FIR rejeita-banda projetado utilizou o código dado no Apêndice E, oriundo da alteração do código do Apêndice A. A resposta em frequência desse filtro é dado na Figura 38. Como mostra a figura o filtro apresenta uma atenuação de aproximadamente –2,54 dB nas frequências de corte normalizadas (0,0680 π/rad e 0,2721 π/rad). Além disso, nesse filtro é possível observar que as frequências maiores que 0,3 π/rad apresentam um ganho de aproximadamente 3,6 dB.

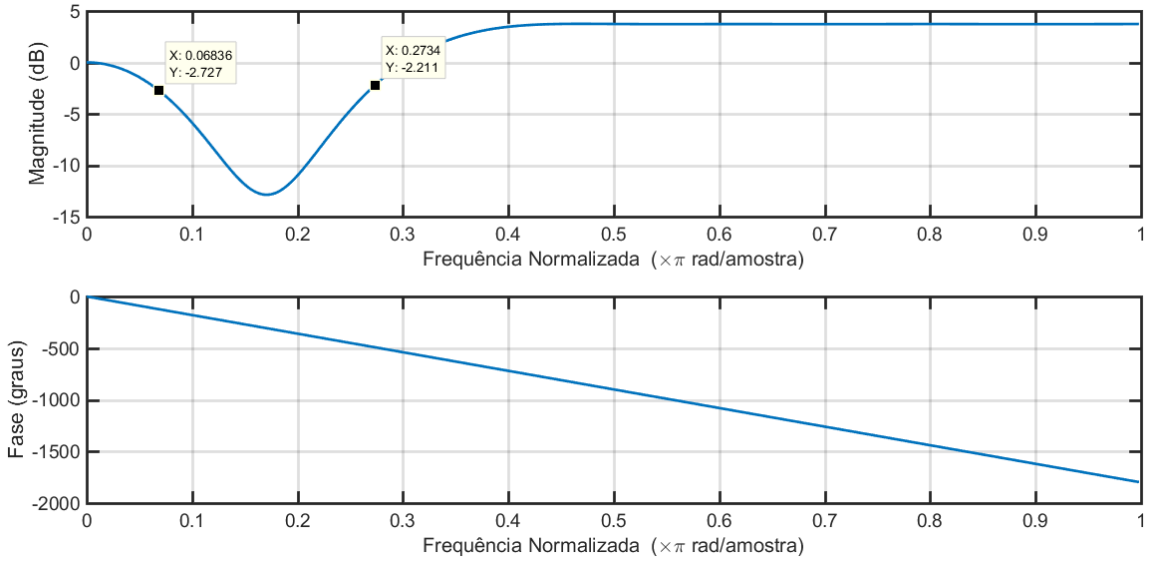


Figura 38 – Resposta em frequência do filtro FIR rejeita-banda utilizando janela de Hamming.
Fonte: Autoria própria.

A Figura 36Figura 39 apresenta os espectros do sinal de áudio antes e após a filtragem pelo filtro FIR rejeita-banda projetado. Nessa figura, as setas indicam as frequências de corte (1500 Hz – 6000 Hz) do filtro. Observa-se que as frequência são atenuadas fora da banda de passagem, entre 1500 Hz e 6000 Hz. No entanto, o filtro presenta uma pequena atenuação nas frequências de corte e um ganho nas frequências acima de 6615 Hz.

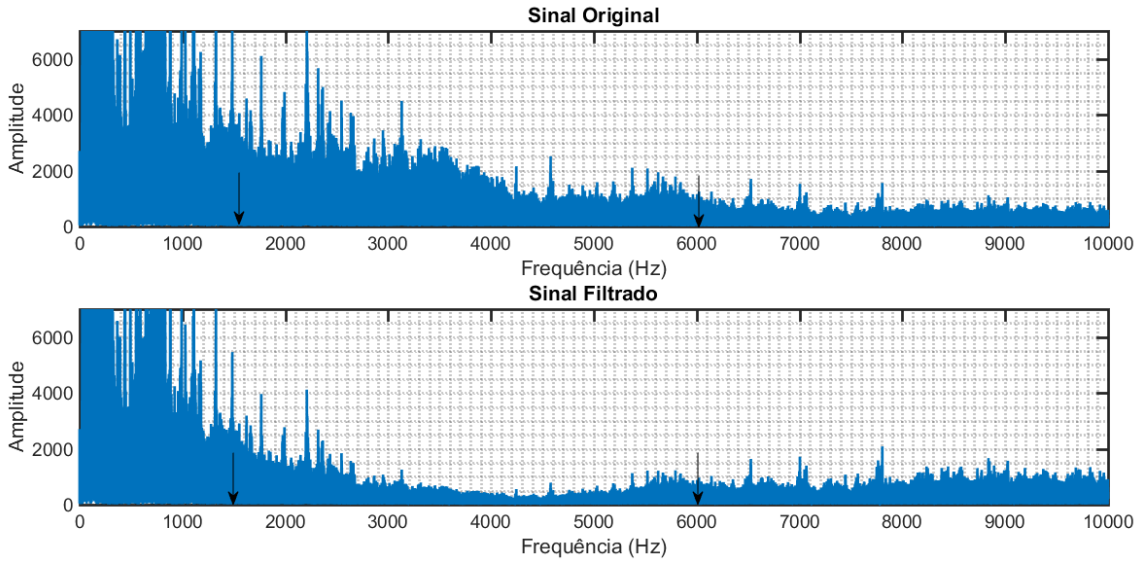


Figura 39 – Rejeita-Banda: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Utilizando o código do sinal multisenoidal do Apêndice B e aplicando o filtro FIR rejeita-banda, obtêm-se os espectros de frequência mostrados na Figura 40. As setas indicam as frequências de corte do filtro. Percebe-se que o filtro rejeita-banda atenua as frequências fora da banda de passagem (1500 Hz – 6000 Hz). Porém há uma pequena atenuação próximo das frequência de corte desse filtro, além de um ganho nas frequências acima de 6615 Hz, isso pode ocorrer porque esse filtro não é ideal, podendo apresentar *ripples* nas bandas passantes e apresentar uma banda de transição considerável.

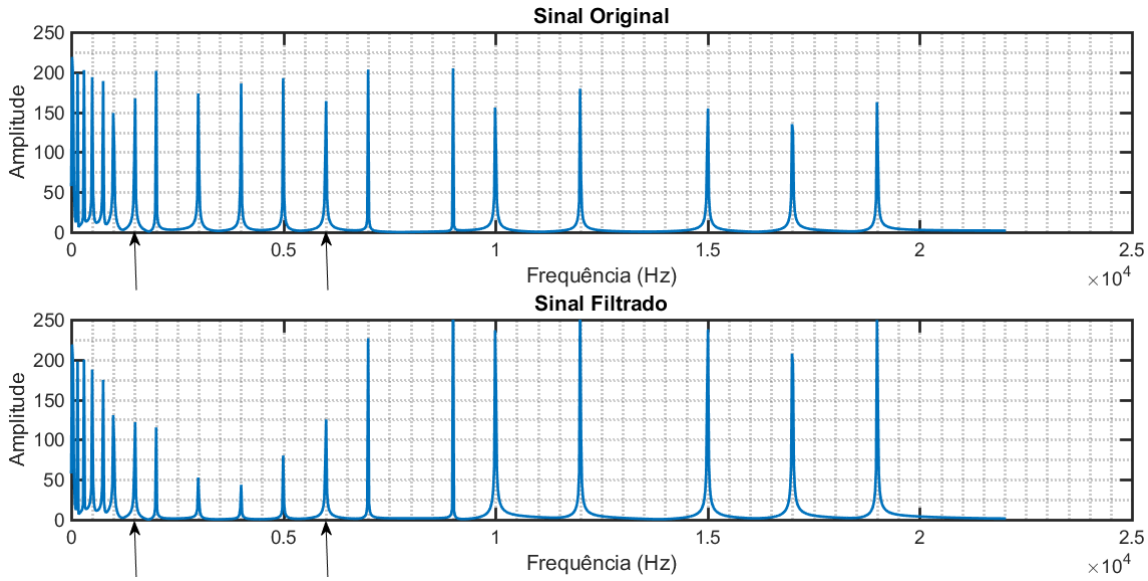


Figura 40 – Rejeita-Banda: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

Com as especificações consideradas dos filtros FIR e com os resultados obtidos na simulação são apresentados os dados na Tabela 4. A tabela mostra as frequências de corte e ordem escolhidas para cada tipo de filtro, a largura do lóbulo principal, a largura da faixa de transição e os coeficientes obtidos. A largura do lóbulo principal e da faixa de transição foram calculadas com as equações dadas na Tabela 2.

Tabela 4 – Dados dos filtros FIR projetados.

	Passa-baixa	Passa-alta	Passa-banda	Rejeita-banda
f_C (Hz)	1500	6000	4000 – 7000	1500 – 6000
Ordem (N)	15	16	20	20
Lóbulo principal (dB)	0,8377	0,7853	0,6283	0,6283
Faixa de transição (Δω)	0,2093 π	0,1962 π	0,1570 π	0,1570 π
Coeficientes d_i obtidos	0,0065 0,0111 0,0239 0,0445 0,0702 0,0963 0,1177 0,1297 0,1297 0,1177 0,0963 0,0702 0,0445 0,0239 0,0111 0,0065	-0,0017 0,0015 0,0104 0,0210 0,0118 -0,0415 -0,1367 -0,2323 0,7297 -0,2323 -0,1367 -0,0415 0,0118 0,0210 0,0104 0,0015 -0,0017	0,0001 0,0075 0,0203 0,0262 -0,0007 -0,0659 -0,1255 -0,1108 0,0007 0,1432 0,2084 0,1432 0,0007 -0,1108 -0,1255 -0,0659 -0,0007 0,0262 0,0203 0,0075 0,0001	0,0003 -0,0003 0,0047 0,0243 0,0607 0,0940 0,0857 0,0069 -0,1283 -0,2591 1,2218 -0,2591 -0,1283 0,0069 0,0857 0,0940 0,0607 0,0243 0,0047 -0,0003 0,0003

Fonte: Autoria própria.

4.1.2 Filtros IIR

Para o projeto do filtro IIR passa-baixa utilizou-se o código no Apêndice A com as modificações presentes no código no Apêndice F. A Figura 41 apresenta a resposta em frequência do filtro IIR passa-baixa. Nessa resposta é possível observar que na frequência de corte ($f_{normalizada}$=0,0680 π/rad) a atenuação do filtro é de -1,92 dB e a fase em -296,6º.

A Figura 42 apresenta os espectros do sinal de áudio antes e depois da filtragem utilizando o filtro IIR passa-baixa. As setas indicam a posição da frequência de corte do filtro.

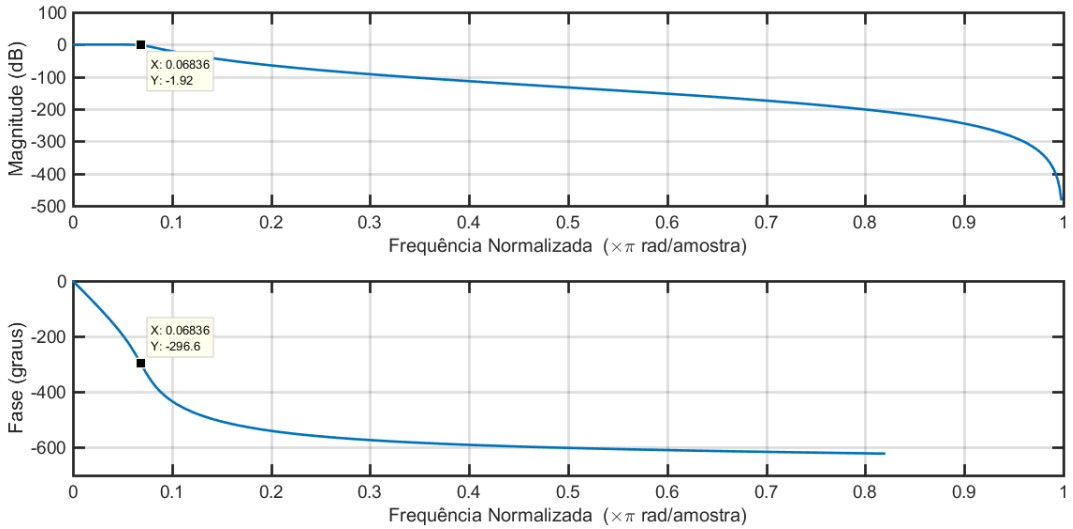


Figura 41 – Resposta em frequência do filtro IIR passa-baixa.

Fonte: Autoria própria.

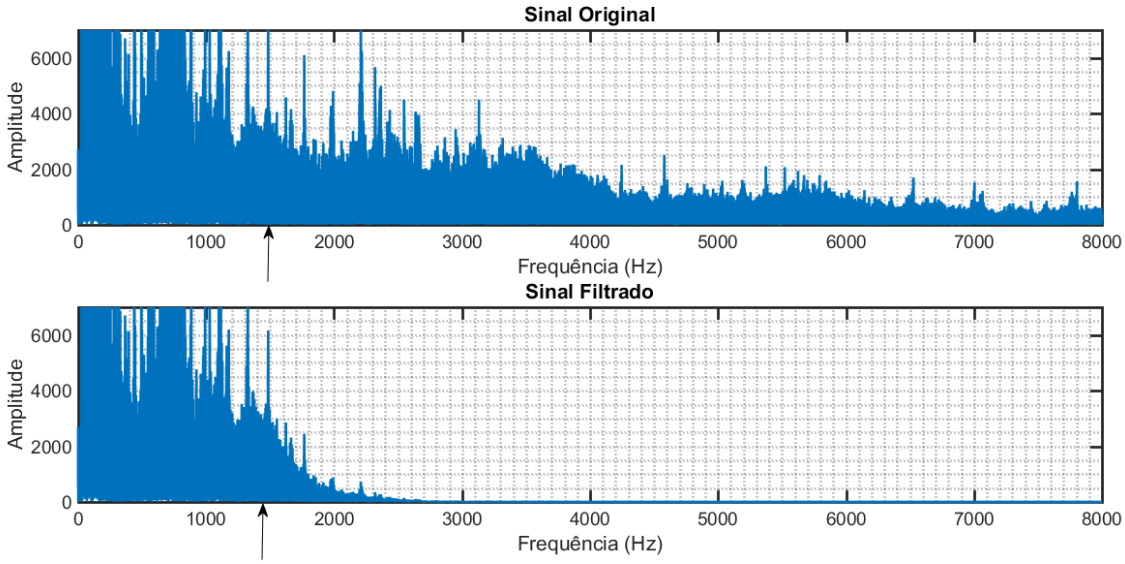


Figura 42 – Passa-baixa: espectro do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Na Figura 42 é possível observar que para frequências maiores que 1500 Hz o sinal é consideravelmente atenuado. Ainda, na frequência de corte, 1500 Hz, há uma atenuação de -1,92 dB.

Utilizando o filtro IIR passa-baixa projetado com o sinal multisenoidal, dado no Apêndice B. É possível obter os espectros de frequência do sinal antes e depois a filtragem, sendo apresentados na Figura 43.

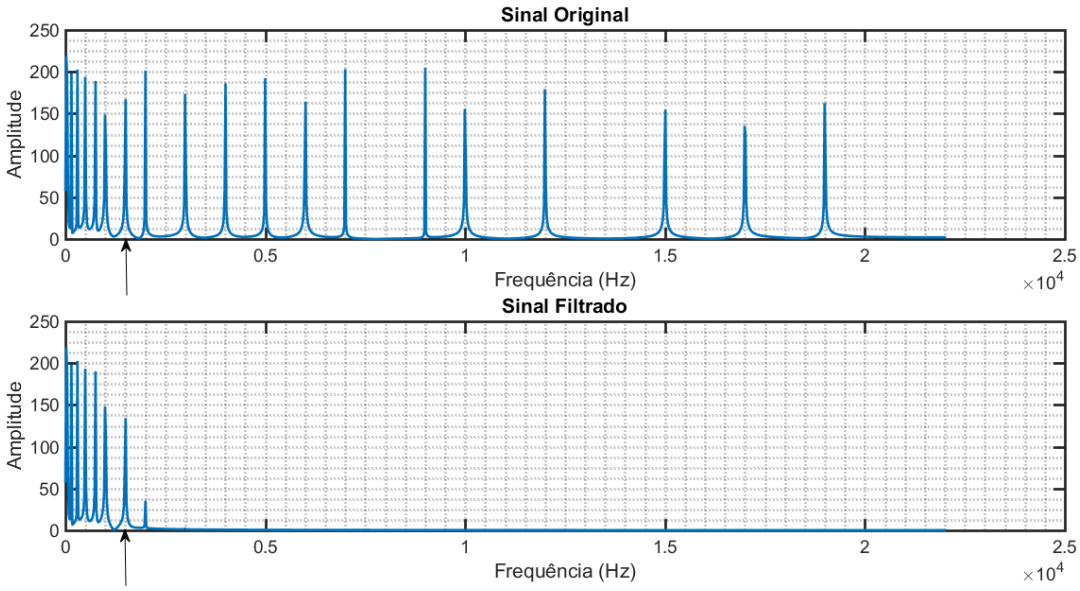


Figura 43 – Passa-baixa: espectro do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

Como tratado anteriormente e analisando a Figura 43, o filtro IIR projetado realizou uma atenuação considerável das frequências maiores que 1500 Hz. Também é possível observar que apesar desse tipo de filtro apresentar uma ordem menor, quando comparado a ordem do filtro FIR do mesmo tipo, esse filtro apresentou uma atenuação muito maior das frequências na banda de rejeição comparada a atenuação obtida com o filtro FIR (Figura 30 e Figura 31).

Para o projeto do filtro IIR passa-alta também utilizou-se o código no Apêndice A com as modificações presentes no código no Apêndice G. A Figura 44 apresenta a resposta em frequência do filtro IIR passa-alta. Nessa figura é possível observar que, na frequência de corte determinada ($f_{normalizada}$=0,2721 π/rad), a atenuação do filtro é de -2,895 dB.

A Figura 45 apresenta os espectros do sinal de áudio antes e depois da filtragem utilizando o filtro IIR passa-alta. As setas indicam a posição da frequência de corte do filtro de 6000 Hz.

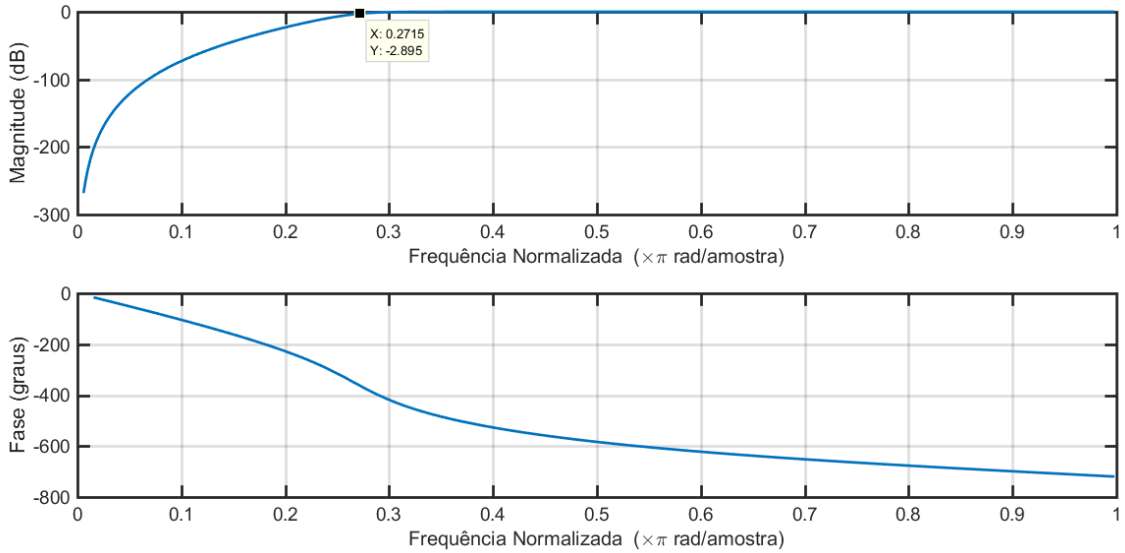


Figura 44 – Resposta em frequência do filtro IIR passa-alta.
Fonte: Autoria própria.

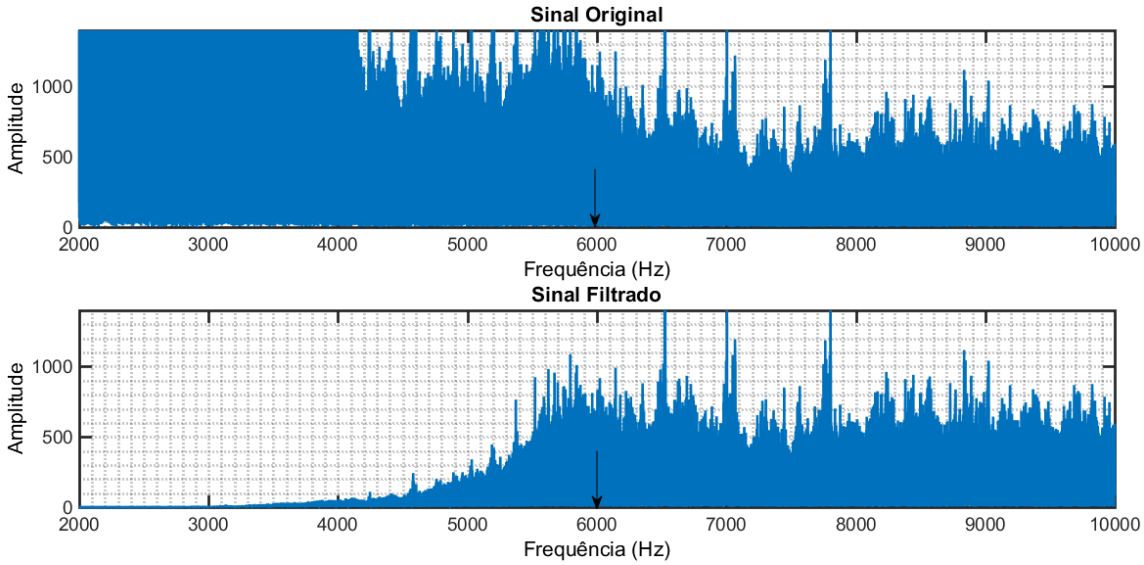


Figura 45 – Passa-alta: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Na Figura 45 é possível observar que para frequências menores que 6000 Hz o sinal é consideravelmente atenuado. Na frequência de corte, de 6000 Hz, o filtro apresenta uma atenuação de -2,895 dB.

Utilizando o filtro IIR passa-alta projetado com o sinal multisenoidal, dado no Apêndice B. A Figura 46 apresenta os espectros de frequência do sinal antes e depois a filtragem. Nessa figura, a frequência de corte de 6000 Hz é indica por uma seta.

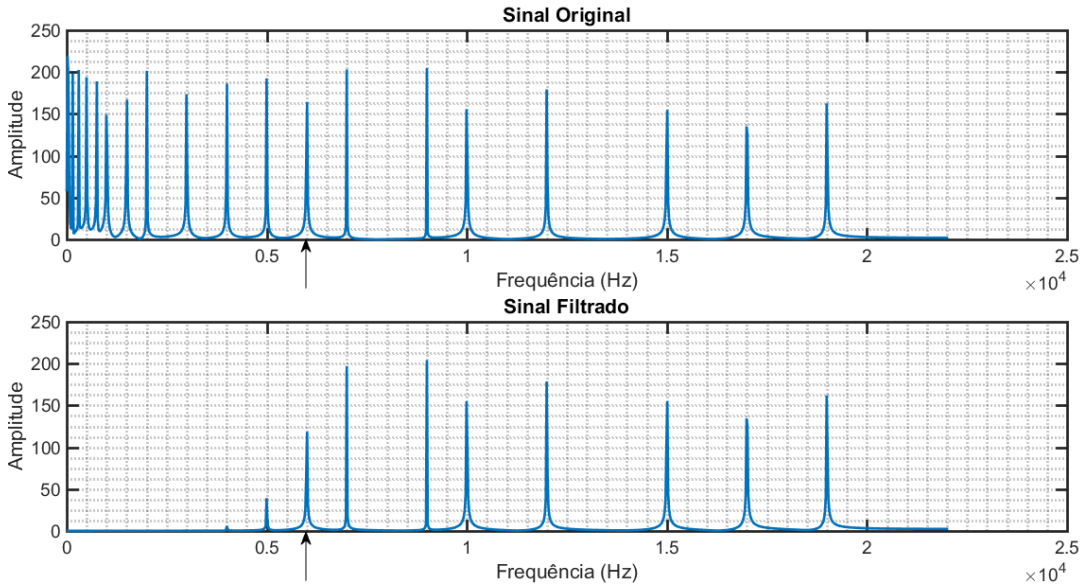


Figura 46 – Passa-alta: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

Para o projeto do filtro IIR passa-banda utiliza-se o código base no Apêndice A e realiza-se as alterações necessárias, resultando no código presente no Apêndice H. A Figura 47 apresenta a resposta em frequência do filtro projetado. Ainda, nessa figura, é possível observar que nas frequências de corte normalizada, 0,1814 π/rad e 0,3175 π/rad, ocorre uma atenuação de aproximadamente -0,778 dB.

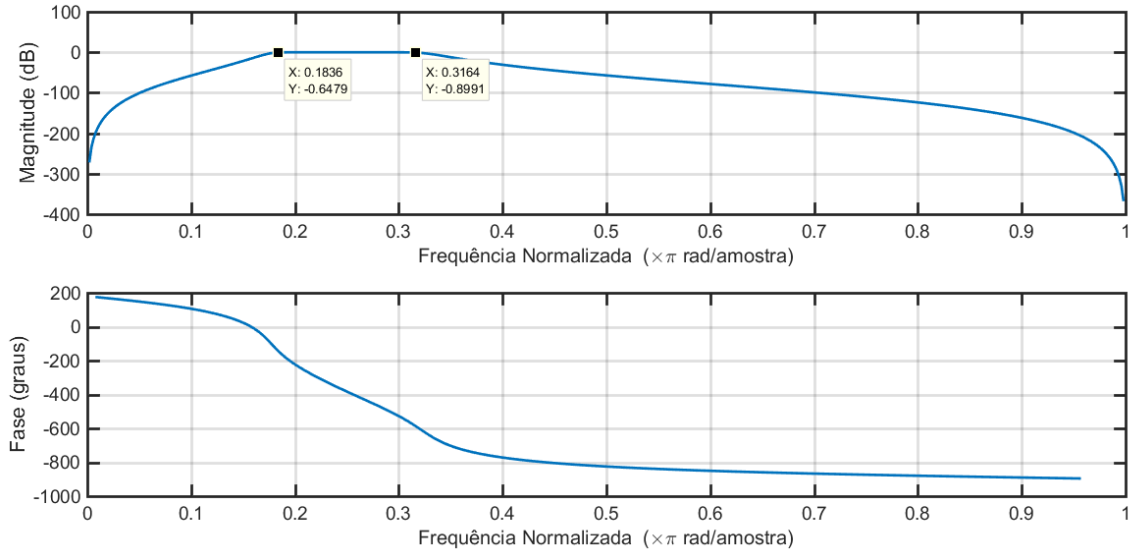


Figura 47 - Resposta em frequência do filtro IIR passa-banda.
Fonte: Autoria própria.

A Figura 48 apresenta os espectros do sinal de áudio antes e depois da filtragem utilizando o filtro IIR passa-banda projetado. As setas indicam as posições das frequências de corte do filtro de 4000 Hz e 6000 Hz.

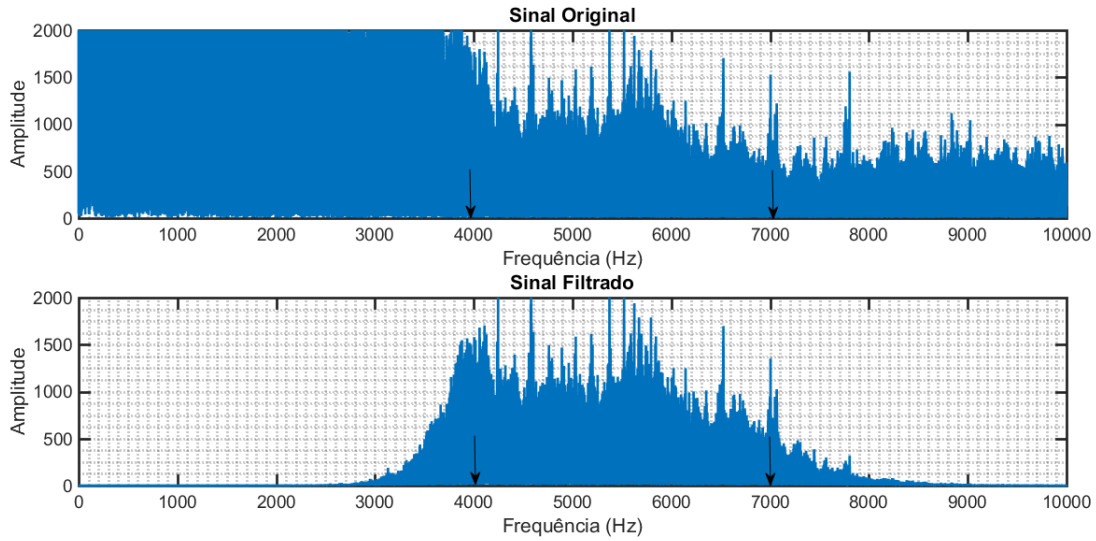


Figura 48 – Passa-Banda: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Utilizando o sinal multisenoidal, dado no Apêndice B, com o filtro IIR passa-banda projetado obtêm-se os espectros de frequência do sinal antes e depois a filtragem dados na Figura 49.

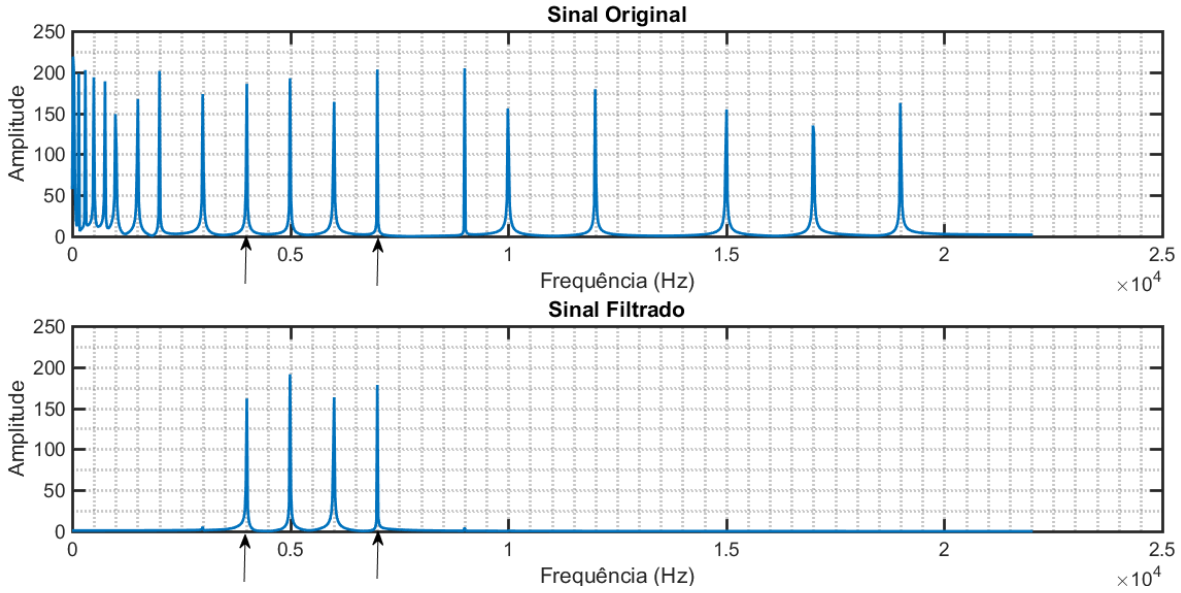


Figura 49 - Passa-Banda: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

Analisando as Figuras 48 e 49 é possível visualizar que o filtro IIR passa-banda projetado atende a resposta de seu tipo, ou seja, atenuando as frequências menores que 4000 Hz e maiores que 7000 Hz e permitindo a passagem, sem atenuações consideráveis, das frequências intermediárias. Nas frequências de corte determinadas, o filtro apresenta uma pequena atenuação, devido ao fato do filtro projetado não ser ideal.

Para o projeto do filtro IIR rejeita-banda realizou-se as modificações do código base, presente no Apêndice A, resultando no código do Apêndice I. A Figura 50 apresenta a resposta em frequência do filtro IIR rejeita-banda. Nessa figura é possível observar que nas frequências de corte normalizada, 0,068 π/rad e 0,2721 π/rad, ocorre uma atenuação de aproximadamente -4,43 dB.

Já na Figura 51 são apresentados os espectros do sinal de áudio antes e depois da filtragem utilizando o filtro IIR rejeita-banda. As setas indicam as posições das frequências de corte do filtro de 1500 Hz e 6000 Hz.

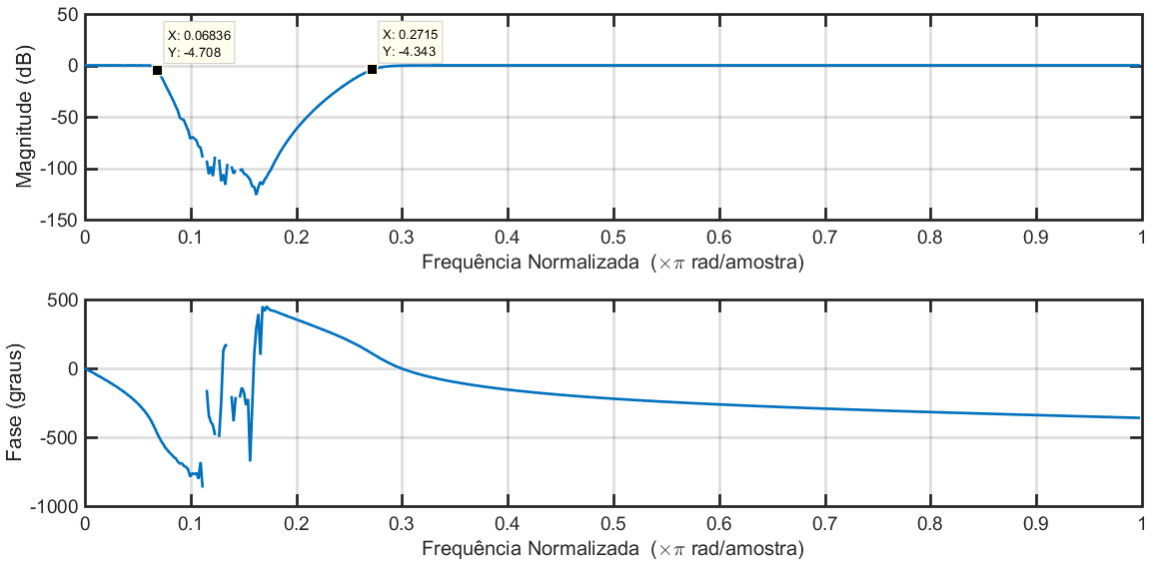


Figura 50 – Resposta em frequência do filtro IIR rejeita-banda.
Fonte: Autoria própria.

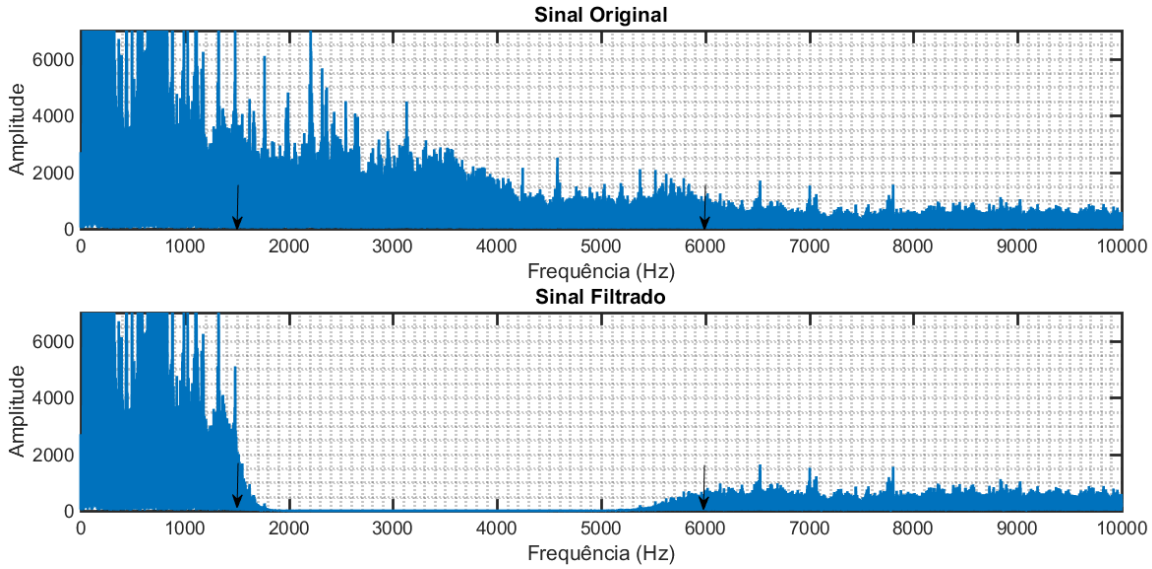


Figura 51 – Rejeita-Banda: espectros do sinal de áudio antes e depois da filtragem.
Fonte: Autoria própria.

Utilizando o filtro IIR rejeita-banda projetado no código com o sinal multisenoidal, dado no Apêndice B, obtém-se os espectros de frequência, antes e depois a filtragem, dados na Figura 52. Analisando as Figuras 51 e 52 é possível concluir que o filtro rejeita-banda projetado apresenta uma resposta de saída conforme o esperado. As frequências entre 1500 Hz e 6000 Hz foram atenuadas, enquanto as frequências fora dessa faixa não apresentem atenuações consideráveis.

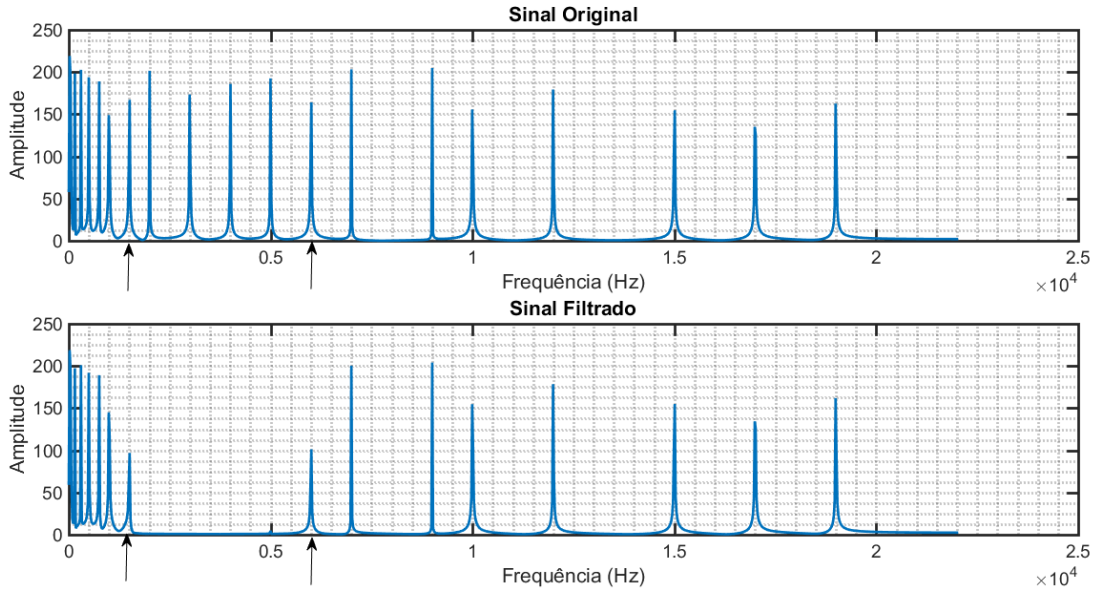


Figura 52 – Rejeita-Banda: espectros do sinal multisenoidal antes e depois da filtragem.
Fonte: Autoria própria.

Com as especificações consideradas dos filtros IIR e com os resultados obtidos na simulação são apresentados os dados na Tabela 5.

Tabela 5 – Dados dos filtros IIR projetados.

		Passa-baixa	**Passa-alta**	**Passa-banda**	**Rejeita-banda**
f_P (Hz)		1500	6000	4000 – 7000	1500 – 6000
f_S (Hz)		3000	3500	1500 – 9000	50 – 8000
Ordem (N)		7	8	13	20
Coeficientes obtidos	a_i	1,0000 -5,9942 15,4620 -22,2423 19,2659 -10,0460 2,9193 -0,3646	1,0000 -3,6531 6,4965 -7,0420 5,0175 -2,3818 0,7310 -0,1319 0,0107	1,0000 -7,3702 26,9093 -63,4868 107,1502 -135,7829 132,2209 -99,6148 57,6616 -25,0550 7,7872 -1,5645 0,1560	0,0001. 10^4 -0,0014.10^4 0,0099.10^4 -0,0436.10^4 0,1377.10^4 -0,3315.10^4 0,6315.10^4 -0,9746.10^4 1,2375.10^4 -1,3056.10^4 1,1509.10^4 -0,8491.10^4 0,5234.10^4 -0,2681.10^4 0,1130.10^4 -0,0386.10^4 0,0104.10^4 -0,0022.10^4 0,0003.10^4 0,0000 0,0000
	b_i	0,0137.10^{-5} 0,0959.10^{-5} 0,2876.10^{-5} 0,4793 .10^{-5} 0,4793 .10^{-5} 0,2876.10^{-5} 0,0959.10^{-5} 0,0137.10^{-5}	0,1034 -0,8270 2,8945 -5,7891 7,2364 -5,7891 2,8945 -0,8270 0,1034	0,0001 0,0000 -0,0005 0,0000 0,0012 0,0000 -0,0016 0,0000 0,0012 0,0000 -0,0005 0,0000 0,0001	0,0000 -0,0002.10^4 0,0019.10^4 -0,0103.10^4 0,0411.10^4 -0,1245.10^4 0,2978.10^4 -0,5754.10^4 0,9125.10^4 -1,1990.10^4 1,3124.10^4 -1,1990.10^4 0,9125.10^4 -0,5754.10^4 0,2978.10^4 -0,1245.10^4 0,0411.10^4 -0,0103.10^4 0,0019 .10^4 -0,0002 .10^4 0,0000
Ripple na banda de passagem (dB)		3			
Atenuação na frequência de parada (dB)		- 40			

Fonte: Autoria própria.

4.2 RESULTADOS DA IMPLEMENTAÇÃO

A implementação desse trabalho consiste em aplicar filtros digitais em um microcontrolador. Como tratado nos capítulos anteriores, o *kit* de desenvolvimento STM23F4-Discovery foi escolhido por apresentar diversas características vantajosas para a aplicação.

O sistema desejado a ser aplicado no microcontrolador foi descrito na seção 3.2.2. No entanto, devido a problemas encontrados na aquisição dos dados no cartão de memória SD, foi necessário realizar algumas adaptações. O sistema desejado dado na Figura 27 necessitou ser substituído pelo sistema dado na Figura 53.

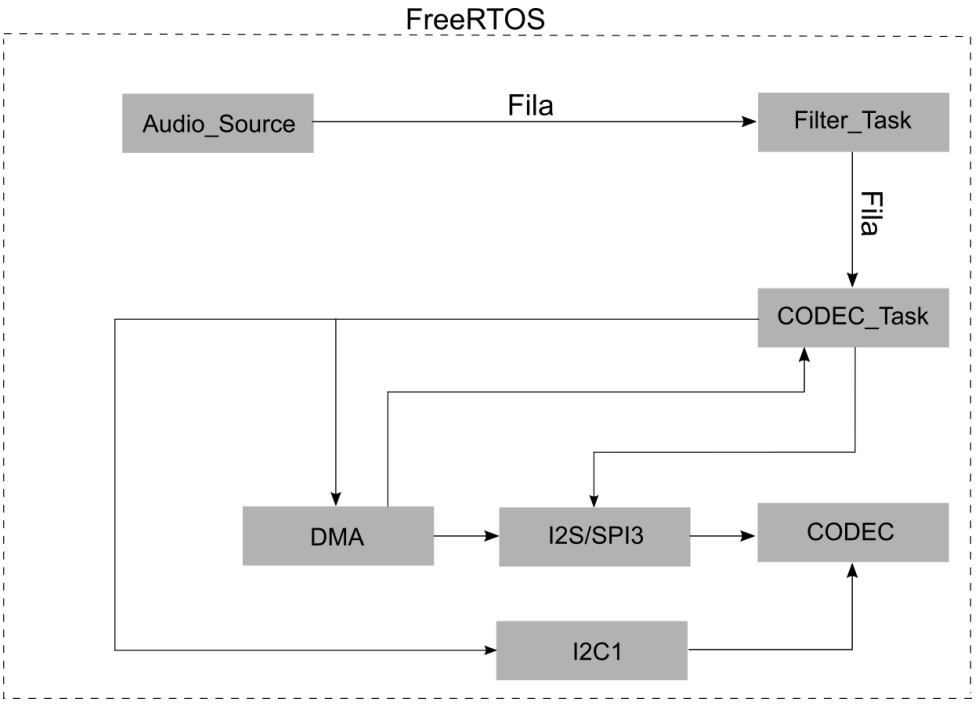


Figura 53- Novo diagrama de tarefas a serem executadas.
Fonte: Autoria própria.

A diferença entre o sistema desejado e o sistema implementado encontra-se na aquisição de dados. Desejava-se que os dados fossem adquiridos em um cartão de memória, por meio do *driver* descrito nos capítulos anteriores. Entretanto, devido a problemas nesse *driver,* que realizava a comunicação usando a FATFs, o sistema necessitou ser modificado de modo que os dados foram armazenados na memória interna do microcontrolador. Utilizou-se o mesmo sinal multisenoidal utilizados nas simulações para ser armazenado no dispositivo.

4.2.1 Filtros FIR

Implementando na memória do microcontrolador o sinal multisenoidal gerado no Matlab®, presente no código no Apêndice B, e fazendo com que esse sinal seja reproduzido sem passar por qualquer processo de filtragem, tem-se o espectro de frequência mostrado na Figura 54.

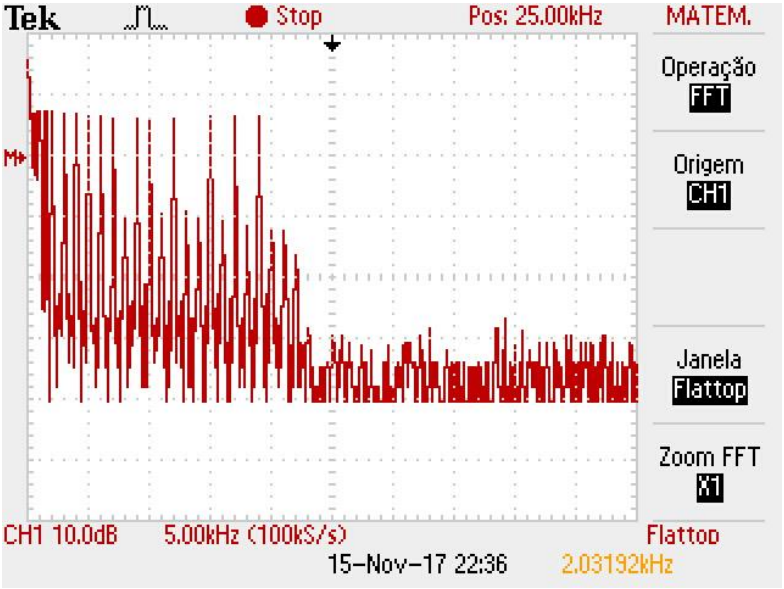


Figura 54 – Espectro de frequência do sinal multisenoidal sem qualquer filtragem.
Fonte: Autoria própria.

Empregando os coeficientes obtidos na simulação do filtro FIR passa-baixa no sistema descrito anteriormente, obtém-se o espectro de frequências dado na Figura 55. Nessa figura, a linha azul tracejada indica onde estavam os valores de amplitude do sinal antes da filtragem e a seta preta é utilizada para indicar a frequência de corte do filtro. Analisando essa resposta é possível observar que o filtro passa-baixa projetado realmente realizou a atenuação considerável das frequências acima de 1500 Hz. Ainda, como nas respostas obtidas na simulação, é possível observar que na frequência de corte ocorreu uma pequena atenuação visto que o filtro não é ideal.

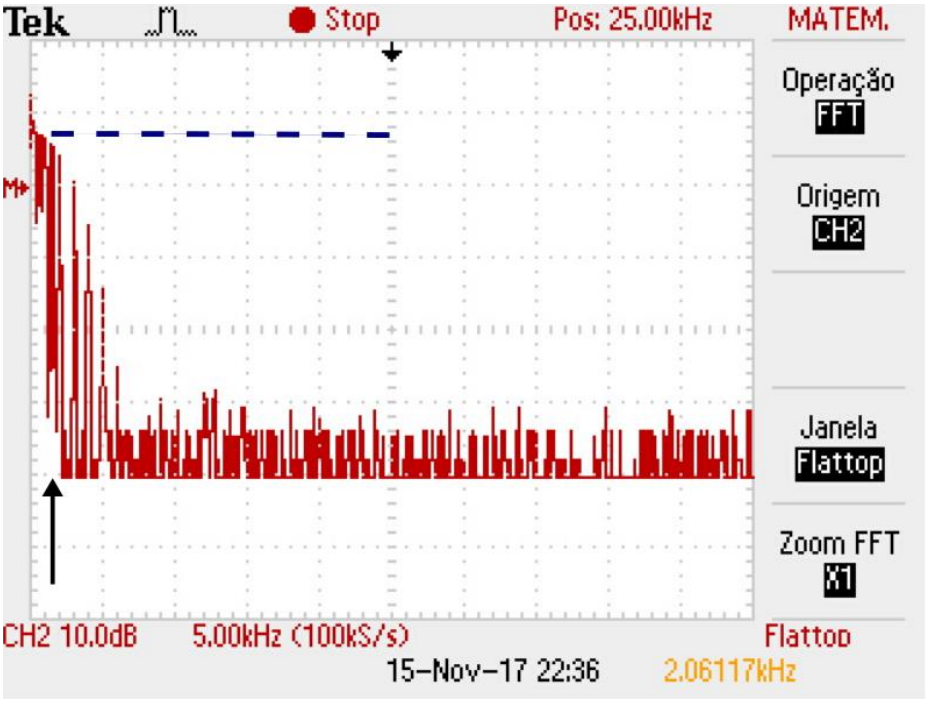


Figura 55 – Espectro de frequência do sinal multisenoidal após filtragem com o FIR passa-baixa.
Fonte: Autoria própria.

Empregando os coeficientes adquiridos na simulação do filtro FIR passa-alta no sistema descrito anteriormente, obtém-se o espectro de frequências dado na Figura 56. A linha azul tracejada indica onde estavam os valores de amplitude do sinal original e a seta preta indica a frequência de corte do filtro. Analisando essa resposta é possível observar que o filtro passa-alta projetado realizou a atenuação das frequências menores que 6000 Hz. Ainda, no espectro é possível observar que nas frequências próximo a de corte ocorre uma pequena atenuação, isso porque o filtro projetado não é ideal.

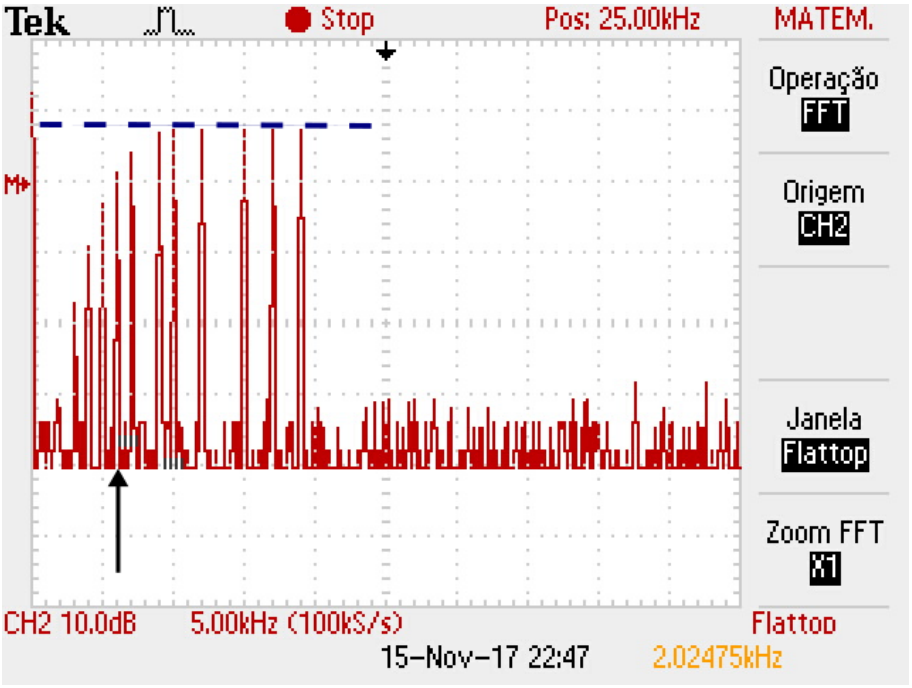


Figura 56 – Espectro de frequência do sinal multisenoidal após filtragem com o FIR passa-alta. Fonte: Autoria própria.

Utilizando os coeficientes obtidos na simulação do filtro FIR passa-banda no sistema descrito anteriormente, obtém-se o espectro de frequências dado na Figura 57. Nessa figura, a escala do osciloscópio mostra 2,5 kHz por divisão e são usadas uma linha azul tracejada para indicar onde estavam os valores de amplitude do sinal original e duas setas pretas para indicar as frequências de corte do filtro. Analisando essa resposta é possível observar que o filtro passa-banda projetado realizou a atenuação das frequências fora da banda de passagem de 4000 Hz a 7000 Hz. Como nos filtros anteriores, próximo as frequências de corte ocorre uma pequena atenuação das frequências devido a não idealidade do filtro.

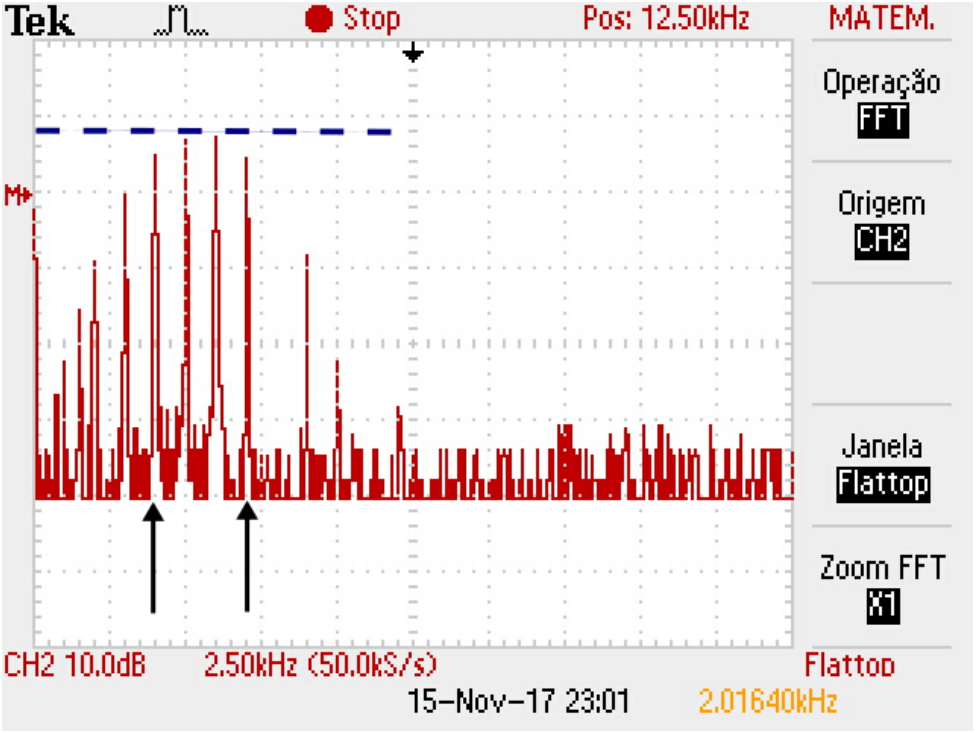


Figura 57 -Espectro de frequência do sinal multisenoidal após filtragem com o FIR passa-banda.
Fonte: Autoria própria.

Por fim, empregando os coeficientes adquiridos na simulação do filtro FIR rejeita-banda no sistema descrito anteriormente, obtém-se o espectro de frequências dado na Figura 58. Nessa figura, a escala do osciloscópio mostra 2,5 kHz por divisão e são usadas uma linha azul tracejada para indicar onde estavam os valores de amplitude do sinal original e duas setas pretas para indicar as frequências de corte do filtro. Analisando essa resposta é possível observar que o filtro rejeita-banda projetado realiza a atenuação das frequências entre 1500 Hz a 6000 Hz. As frequências fora dessa faixa, banda passante do filtro, sofrem atenuações somente quando próximas as frequências de corte. Também, é possível observar que nas frequências acima de 6615 Hz esse filtro apresenta um pequeno ganho nas componentes de frequência, como observado na sua resposta simulada na seção anterior.

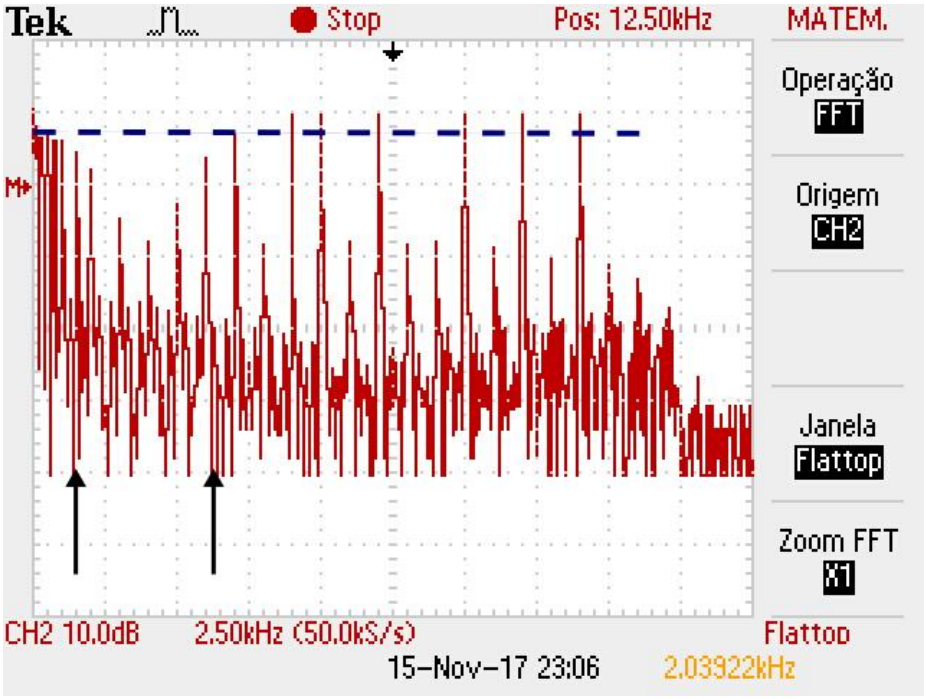


Figura 58 -Espectro de frequência do sinal multisenoidal após filtragem com o FIR rejeita-banda.
Fonte: Autoria própria.

4.2.2 Filtros IIR

Devido aos problemas encontrados com a aquisição dos dados, descrita anteriormente, gerou-se atraso no cronograma do trabalho e, por consequência, os filtros IIR não obtiveram resultados satisfatórios. Algumas tentativas foram realizadas, utilizando o código dado no Apêndice N, porém os filtros apresentaram resposta saturada. Esses problemas podem ser decorrentes dos coeficientes obtidos nas simulações, calculados com o Matlab®.

4.3 CONSIDERAÇÕES FINAIS

O Capítulo 4 apresentou os resultados das simulações dos filtros digitais projetados no Capítulo 3 e da implementação dos filtros FIR. Os resultados das simulações estão separados em filtros FIR e filtros IIR. Analisando as respostas obtidas é possível verificar a diferença entre cada configuração de filtro projetado (passa-baixa, passa-alta, passa-banda e rejeita-banda) quanto a sua seletividade de frequências. Pelos resultados é possível visualizar as bandas de passagem e as

bandas de rejeição de cada tipologia, visto que as bandas de rejeição apresentaram uma atenuação considerável.

Realizando a comparação entre as respostas obtidas dos filtros FIR e dos filtros IIR, é possível verificar algumas características de cada tipologia, tratadas na teoria do Capítulo 2, como a seletividade mais acentuada dos filtros IIR comparado aos filtros FIR, embora suas ordens sejam menores, e as fases lineares dos filtros FIR.

Na seção de implementação, apresentou-se os resultados obtidos com a utilização dos filtros FIR projetados. Embora somente a tipologia FIR tenha sido implementada, algumas considerações podem ser feitas a respeito das especificações e respostas obtidas. Todas as respostas obtidas dos filtros implementados aproximaram-se das respostas obtidas nas simulações. Assim, analisando cada resposta é possível classificar o tipo de filtro implementado em passa-baixa, passa-alta, passa-banda, rejeita-banda, pois é visível a atenuação das componentes de frequência nas bandas de rejeição. Também, é pertinente comentar a respeito da ordem dos filtros implementados, visto que apresentaram ordens elevadas, sendo a menor ordem a do filtro passa-baixa de tamanho 15. Essa especificação destaca uma das vantagens da utilização dos filtros digitais uma vez que a implementação de um filtro analógico, de mesma ordem, exigiria uma quantidade de componentes significativa. Outra vantagem dos filtros digitais, tratadas nos capítulos anteriores, visível na implementação desse trabalho, trata-se da flexibilidade de ajuste dos coeficientes para mudança de configuração.

5 CONCLUSÕES

O presente trabalho abordou o estudo, o projeto e a implementação de filtros digitais para aplicações em áudio. A partir de pesquisas bibliográficas foi possível entender os efeitos das ondas sonoras e as necessidades de processamento de sinais em áudio, a fim de atender as demandas existentes. Os parâmetros, os critérios e as metodologias de projeto foram apresentadas e discutidas. Com base nisso, filtros digitais do tipo FIR e IIR foram especificados e projetados. A ferramenta matemática Matlab® foi utilizada no projeto e na simulação de tais filtros. Os resultados da simulação demonstraram as respostas de cada filtro projetado. A implementação em *hardware* e *software* exigiu conhecimentos a respeito do microcontrolador utilizado, dos periféricos e do codec de áudio empregado, assim como do sistema de arquivos FAT e demais elementos de *software* utilizados.

Para todos os filtros projetados foram utilizados os mesmos sinais de entrada, a partir de um arquivo de áudio WAV e uma onda multisenoidal gerada no Matlab®. Assim, as respostas das simulações realizadas puderam ser comparadas. Os filtros FIR e IIR apresentaram diferenças consideráveis na filtragem dos sinais de entrada. Para uma mesma configuração, a ordem obtida para os filtros IIR foram menores que as dos filtros FIR.

Utilizando o sinal de áudio, a filtragem realizada ocasionou uma diferença audível do sinal original, devido a atenuação de determinadas frequências da música. Essa atenuação é capaz de modificar algumas qualidades fisiológicas do sinal original, como a intensidade e o timbre.

Por meio da análise dos resultados práticos obtidos com os filtros FIR é possível concluir que os filtros projetados obtiveram a seletividade das frequências esperada, realizando a atenuação das frequências fora de sua banda de passagem. Além disso, esses resultados confirmam algumas das vantagens dos filtros digitais quando comparados aos filtros analógicos, como a flexibilidade da mudança dos coeficientes no sistema, que por consequência podem alterar o tipo e/ou a ordem do filtro. Ainda, por meio da aplicação, é possível visualizar a facilidade de implementar um filtro com ordem elevada. A menor ordem utilizada nos filtros FIR nesse trabalho foi de ordem 15, sendo que, para um mesmo filtro analógico, pode-se demandar uma quantidade considerável de componentes.

Os imprevistos que ocasionaram a falha do gerenciamento de tempo de execução do trabalho e, por sua vez, a falta de alguns resultados práticos anexados a este, estão relacionados ao desenvolvimento do *driver* para a aquisição de dados do cartão SD. Embora os comandos presentes nesse *driver* obtivessem respostas do cartão SD, o acesso e a leitura dos dados não eram realizados de maneira eficiente e geravam atrasos consideráveis na obtenção dos dados, não atingindo a performance mínima necessária. Esse problema pode ser decorrente da sequência e da escolha dos comandos utilizados, bem como temporização e também a velocidade de acesso da FATFs.

As principais dificuldades encontradas estão relacionadas ao uso dos recursos do microcontrolador, na configuração do FreeRTOS, no desenvolvimento do *driver* para o cartão SD e no acoplamento deste ao FATFs, no gerenciamento das tarefas e na resolução de problemas. Ainda assim, os objetivos foram atingidos em grande parte, com a obtenção dos resultados da implementação dos filtros FIR, permitindo a análise da utilização de filtros digitais em sistemas microcontrolados.

Com a finalização desse trabalho algumas considerações podem ser tomadas em relação aos dados obtidos, os métodos utilizados, as dificuldades encontradas e, sobretudo, a respeito do aprendizado adquirido por meio da realização de suas etapas. A realização do trabalho permitiu desenvolver novas habilidades e aprimorar alguns conhecimentos já obtidos na graduação, principalmente na familiarização com novos *softwares* e microcontroladores.

REFERÊNCIAS

ARM LIMITED. Cortex-M series processors. **ARM Developer**, 2010. Disponível em: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337h/BABCHGFJ.html>. Acesso em: 02 Outubro 2017.

ARM LIMITED. Cortex - M4. **ARM Developer**, 2017. Disponível em: <https://developer.arm.com/products/processors/cortex-m/cortex-m4>. Acesso em: 02 Outubro 2017.

BAUER, Wolfgang; WESTFALL, Gary D.; DIAS, Helio. **Física para Universitários - Relatividade, Oscilações, Ondas e Calor**. [S.l.]: AMGH, 2013. 372 p.

BERTOLDO, Leandro. **Fenômenos Ondulatórios**. 1. ed. [S.l.]: [s.n.], 2013.

BRAGA, Newton C. **Fundamentos do Som e Acústica**. 1ª. ed. São Paulo: Instituto Newton C. Braga, v. 8, 2014.

BROWN, Geoffrey. **Discovering the STM32 Microcontroller**. [S.l.]: Indiana University, 2016.

CARISSIMI, Alexandre S.; ROCHOL, Juergen; GRANVILLE, Lisandro Z. **Redes de Computadores**. Porto Alegre : Bookman, v. 20, 2009.

COMER, Douglas E. **Redes de Computadores e Internet**. 6ª. ed. Porto Alegre: Bookman, 2016.

CORINTHIOS, Michael. **Signals, Systems, Transforms, and Digital Signal Processing with MATLAB**. [S.l.]: CRC Press, 2009.

COSTANZO, Linda S. **Fisiologia**. 2007. 3 - Elsevier. Rio de Janeiro, 2007.

DE LA VEJA, Alexandre S. **Apostila de Teoria para Processamento Digital de Sinais**. Disponível em: <http://www.telecom.uff.br/~delavega/public/DSP/apostila_teo_dsp.pdf>. Acesso em: Setembro 2016.

FILHO, Florivaldo M. **A Acústica Musical em Palavras e Sons**. Cotia: Ateliê Editorial, 2003.

FONSECA, Nuno. **Introdução a Engenharia do Som**. 6. ed. [S.l.]: FCA, 2012. 272 p.

FOROUZAN, Behrouz A. **Comunicação de Dados e Redes de Computadores**. Tradução de Ariovaldo Griesi. 4ª. ed. Porto Alegre: AMGH, 2008.

FRENZEL, Louis E. **Fundamentos da comunicação eletrônica**. 3ª. ed. Porto Alegre: AMGH, 2013.

GARCIA, Eduardo A. C. **Biofísica**. 1. ed. [S.l.]: Sarvier, 2002. 387 p.

HAYES, Monson H. **Processamento Digital de Sinais**. Tradução de Anatólio Laschuk. Porto Alegre: Bookman, 2006.

HAYKIN, Simon S.; VEEN, Barry V. **Sinais e Sistemas**. Tradução de José Carlos Barbosa do Santos. Porto Alegre: Bookman, 2001.

HAYKIN, Simon; MOHER, Michael. **Sistemas modernos de comunicações wireless**. [S.l.]: Bookman, 2008.

HEWITT, Paul G. **Física conceitual**. 12ª. ed. Porto Alegre: Bookman, 2015.

HIGUTI, Ricardo T. **Filtros Digitais Tipo FIR**. UNESP. São Paulo: [s.n.].

JESZENSKY, Paul J. E. **Sistemas Telefonicos**. Barueri: Manole, 2004.

JUNIOR, Antonio P. **Eletrônica analógica:** amplificadores operacionais e filtros ativos: teorias, projetos, aplicações e laboratório. Porto Alegre: Bookman, 2003.

KANDEL, Eric et al. **Princípios de Neurociências**. 5ª. ed. [S.l.]: AMGH, 2014.

KUO, Sen M.; GAN, Woon-Seng. **Digital signal processors:** architectures, implementations, and applications. Upper Saddle River: Pearson Prentice Hall, 2005.

LATHI, B. P. **Sinais e Sistemas Lineares**. 2ª. ed. Porto Alegre: Bookman, 2006.

LEVITOV, Alexander B.; DALLAS, Apostolos P.; SLONIM, Anthony D. **Ultrassonografia à Beira do Leito na Medicina Clínica**. Porto Alegre: Amgh, 2013.

MARQUES, Miguel P. **Sistemas e Técnicas de Produção Áudio**. 1ª. ed. [S.l.]: FCA, 2014.

MATHWORKS. Documentation. **MathWorks Corporation**, 2017. Disponível em: <http://www.mathworks.com/help/signal/ug/fir-filter-design.html>. Acesso em: 18 Setembro 2017.

NALON, José A. **Introdução ao Processamento Digital de Sinais**. Rio de Janeiro: LTC, 2009.

OPPENHEIM, Alan V.; SCHAFER, Ronald W. **Digital signal processing**. [S.l.]: Prentice-Hall, 1998.

PLS DEVELOPMENT TOOLS. Embedded Trace Macrocell (ETM) support. **PLS MC**, 2017. Disponível em: <https://www.pls-mc.com/embedded-trace-macrocell-etm-support/features-a-960.html?sh8=RVRN>. Acesso em: 02 Outubro 2017.

PROAKIS, John G.; MANOLAKIS, Dimitris G. **Digital signal processing:** principles, algorithms, and applications. 4ª. ed. Upper Saddle River: Prentice-Hall, 2007. 1084 p.

RAPOSO, João. **Neurossonologia vascular:** Ultrassonografia vascular em Neurologia. [S.l.]: João Raposo, 2016. 105 p.

RAY D. KENT, Charles R. **Análise Acústica da Fala**. Tradução de Alexsandro Rodrigues Meireles. São Paulo: Cortez, 2015.

ROBERTS, M. J. **Fundamentos de Sinais e Sistemas**. Tradução de Carlos Henrique Nogueira de Resende Barbosa. Porto Alegre: AMGH, 2010.

ROCHOL, Juergen. **Comunicação de Dados**. Porto Alegre: Bookman, v. 22, 2012.

RUNSTEIN, Robert E.; HUBER, David M. **Técnicas Modernas de Gravação de Áudio**. 7ª. ed. [S.l.]: Elsevier, 2011.

SADIKU, Matthew N. O.; ALEXANDER, Charles K.; MUSA, Sarhan. **Análise de Circuitos Elétricos com Aplicações**. Porto Alegre: AMGH, 2014.

SCHULER, Charles. **Eletrônica II**. Tradução de Fernando Lessa Tofoli. 7ª. ed. [S.l.]: Bookman, 2013.

SENAI. SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL. **Instalador de som e acessórios eletroeletrônicos automotivos**. São Paulo: SENAI-SP, 2014.

SILVERTHORN, Dee U. **Fisiologia Humana:** Uma Aborgadem Integrada. 2010. 5 - Artmed., 2010.

SIQUEIRA, Ethevaldo. **Para Compreender O Mundo Digital**. São Paulo: Globo, 2008.

STALLINGS, William. **Arquitetura e organização de computadores:** projeto para o desempenho. 5ª. ed. São Paulo: Prentice-Hall, 2002.

STMICROELECTRONICS. **Developing Applications on STM32Cube with FatFs**. 2014.[s.n.]., 2014. Disponível em: <http://www.st.com/content/ccc/resource/technical/document/user_manual/61/79/2b/96/c8/b4/48/19/DM00105259.pdf/files/DM00105259.pdf/jcr:content/translations/en.DM00105259.pdf>. Acesso em: 7 Outubro 2017.

STMICROELECTRONICS. STM32F407VG, 2017. Disponível em: <http://www.st.com/en/microcontrollers/stm32f407vg.html>. Acesso em: 02 Outubro 2017.

WEEKS, Michael. **Processamento digital de sinais utilizando MATLAB e Wavelets**. 2ª. ed. Rio de Janeiro: LTC, 2012.

ZUBEN, Paulo. **Música e tecnologia:** o som e seus novos instrumentos. São Paulo: Irmão Vitale, 2004.

APÊNDICE A – Código implementado no Matlab para o projeto do filtro FIR passa-baixa.

```
% UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
% Trabalho de Conclusão de Curso
% Acadêmica: Bruna Machado Mulinari
%
% FILTRO FIR PASSA-BAIXA
%

clear all;
close all;
clc;

%LOCALIZAÇÃO DO ARQUIVO
infile = 'C:\Users\Bruna\Documents\UTFPR\9° Período\Trabalho de Conclusão
de Curso -TCC\FINAL\Filtragem de sinais\Música\audio.wav';

%LEITURA DO SINAL
[s, Fs]=audioread(infile);              % s - Indica canal mono ou canal
                                        % estereo, uma ou duas colunas;
                                        % Fs - Frequencia de amostragem;

%TEMPO DO SINAL
tempo=size(s,1)/Fs;                     % Tamanho do vetor s/Fs
amostras = length(s);                   % Quantidade de amostras


%ESPECIFICAÇÕES DO FILTRO
N = 15;                                 % Ordem do filtro
fc = 1500;                              % Frequências de corte
Wn = fc/(Fs/2);                         % Frequência normalizada

%ESPECIFICACAO DA JANELA
b = fir1(N,Wn,'low');                   % Janela de Hamming

%FILTRAGEM
s_filtrado = filter(b,1,s);

% %RESPOSTA EM FREQUÊNCIA DO FILTRO
freqz(b,1);                             % Apresenta módulo e fase

% FFT DOS SINAIS:
MTFSM = abs(fft(s));                    % Espectro do sinal original
MTFSMJH=abs(fft(s_filtrado));           % Espectro do sinal filtrado

% Plotagem da FFT
figure();
f=Fs*(0:((amostras/2)-1))/amostras;
subplot(2,1,1);
plot(f, MTFSM(1:(amostras/2)));
grid minor;
title('Sinal Original');
xlabel('Frequência (Hz)');
ylabel('Magnitude');
subplot(2,1,2);
plot(f, MTFSMJH(1:(amostras/2)));
grid minor;
```

```
title('Sinal Filtrado');
xlabel('Frequência (Hz)');
ylabel('Magnitude');

%sound(smult,Fs);
%sound(s_filtrado,Fs);
```

APÊNDICE B – Código implementado para geração de um sinal multisenoidal.

```
%
% UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
% Trabalho de Conclusão de Curso
% Acadêmica: Bruna Machado Mulinari
%
% PB C/ SINAL MULTISENOIDAL
%

close all;
clear all;
clc;

Fs = 44100;                     % Frequência de amostragem

% Frequências das senóides
f1=20;
f2=50;
f3=150;
f4=300;
f5=500;
f6=750;
f7=1000;
f8=1500;
f9=2000;
f10=3000;
f11=4000;
f12=5000;
f13=6000;
f14=7000;
f15=9000;
f16=10000;
f17=12000;
f18=15000;
f19=17000;
f20=19000;

amostras = 2048;
t=(1:amostras)/Fs;                      % Tempo

s1=0.2*sin(2*pi*f1*t);                  % Senóides individuais
s2=0.2*sin(2*pi*f2*t);
s3=0.2*sin(2*pi*f3*t);
s4=0.2*sin(2*pi*f4*t);
s5=0.2*sin(2*pi*f5*t);
s6=0.2*sin(2*pi*f6*t);
s7=0.2*sin(2*pi*f7*t);
s8=0.2*sin(2*pi*f8*t);
s9=0.2*sin(2*pi*f9*t);
s10=0.2*sin(2*pi*f10*t);
s11=0.2*sin(2*pi*f11*t);
s12=0.2*sin(2*pi*f12*t);
s13=0.2*sin(2*pi*f13*t);
s14=0.2*sin(2*pi*f14*t);
s15=0.2*sin(2*pi*f15*t);
```

```
s16=0.2*sin(2*pi*f16*t);
s17=0.2*sin(2*pi*f17*t);
s18=0.2*sin(2*pi*f18*t);
s19=0.2*sin(2*pi*f19*t);
s20=0.2*sin(2*pi*f20*t);
```

```
smult=s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12+s13+s14+s15+s16+s17+s18+s19+s2
0;
sc=(65535/5)*(smult);

%ESPECIFICAÇÕES DO FILTRO
N = 15;                                 % Ordem do filtro
fc = 1500;                              % Frequências de corte
Wn = fc/(Fs/2);                         % Frequência normalizada

%ESPECIFICACAO DA JANELA
b = fir1(N,Wn, 'low');                  % Janela de Hamming

%FILTRAGEM
s_filtrado = filter(b,1,smult);

% Plotagem do sinal no tempo
t=((1:amostras)/Fs)*1000;               % Delta T em milisegundos
figure()
subplot(2,1,1);
plot(t,smult);
grid on;
title('Sinal Multisenoidal - Resposta Temporal')
subplot(2,1,2);
plot(t,s_filtrado);
grid on;
title('Sinal Filtrado')

% FFT DOS SINAIS:
MTFSM = abs(fft(smult));                  % Espectro do sinal original
MTFSMJH=abs(fft(s_filtrado));             % Espectro do sinal filtrado

% Plotagem da FFT
figure();
f=Fs*(0:((amostras/2)-1))/amostras;
subplot(2,1,1);
plot(f, MTFSM(1:(amostras/2)));
grid minor;
title('Sinal Original');
xlabel('Frequência (Hz)');
ylabel('Amplitude');
subplot(2,1,2);
plot(f, MTFSMJH(1:(amostras/2)));
grid minor;
title('Sinal Filtrado');
xlabel('Frequência (Hz)');
ylabel('Amplitude');
```

APÊNDICE C – Alterações no código mostrado no Apêndice A para projeto do filtro FIR passa-alta.

```
%ESPECIFICAÇÕES DO FILTRO
N = 16;                                 % Ordem do filtro
fc = 6000;                              % Frequências de corte
Wn = fc/(Fs/2);                         % Frequência normalizada

%ESPECIFICACAO DA JANELA
b = fir1(N,Wn, 'high');                 % Janela de Hamming
```

APÊNDICE D – Alterações no código mostrado no Apêndice A para projeto do filtro FIR passa-banda.

```
%ESPECIFICAÇÕES DO FILTRO
N = 20;                              % Ordem do filtro
fc1 = 4000;                          % Frequências de corte
fc2 = 7000;
Wn1 = fc1/(Fs/2);                    % Frequências normalizadas
Wn2 = fc2/(Fs/2);

%ESPECIFICACAO DA JANELA
b = fir1(N, [Wn1 Wn2], 'bandpass');      % Janela de Hamming
```

APÊNDICE E – Alterações no código mostrado no Apêndice A para projeto do filtro FIR rejeita-faixa.

```
%ESPECIFICAÇÕES DO FILTRO
N = 20;                              % Ordem do filtro
fc1 = 1500;                          % Frequências de corte
fc2 = 6000;
Wn1 = fc1/(Fs/2);                    % Frequências normalizadas
Wn2 = fc2/(Fs/2);

%ESPECIFICACAO DA JANELA
b = fir1(N, [Wn1 Wn2], 'stop');      % Janela de Hamming
```

APÊNDICE F – Alterações no código mostrado no Apêndice A para projeto do filtro IIR passa-baixa

```
%ESPECIFICAÇÕES DO FILTRO

Rp= 3;                                  % Ripple na banda de passagem
Rs= 40;                                 % Atenuação na banda de rejeição
fp=1500;                                % Frequência de corte
fs=3000;                                % Frequência de rejeição
Wp=fp/(Fs/2);                           % Frequência normalizada
Ws=fs/(Fs/2);

[n,Wc]=buttord(Wp,Ws,Rp,Rs);            % Retorna ordem e frequência de corte

[b,a]= butter(n,Wc, 'low');             % Retorna os coeficientes a e b


%FILTRAGEM
s_filtrado = filter(b,a,canal_D);
```

APÊNDICE G – Alterações no código mostrado no Apêndice A para projeto do filtro IIR passa-alta.

```
%ESPECIFICAÇÕES DO FILTRO

Rp= 3;                                  % Ripple na banda de passagem
Rs= 40;                                 % Atenuação na banda de rejeição
fp=6000;                                % Frequência de corte
fs=3500;                                % Frequência de rejeição
Wp=fp/(Fs/2);                           % Frequência normalizada
Ws=fs/(Fs/2);

[n,Wc]=buttord(Wp,Ws,Rp,Rs);            % Retorna ordem e frequência de corte

[b,a]= butter(n,Wc, 'high');            % Retorna os coeficientes a e b

%FILTRAGEM
s_filtrado = filter(b,a,canal_D);
```

APÊNDICE H – Alterações no código mostrado no Apêndice E para projeto do filtro IIR passa-banda.

```
%ESPECIFICAÇÕES DO FILTRO

Rp= 3;                                  % Ripple na banda de passagem
Rs= 40;                                 % Atenuação na banda de rejeição
fp1=4000;                               % Frequência de corte
fp2=7000;
fs1=1500;                               % Frequência de rejeição
fs2=9500;
Wp1=fp1/(Fs/2);                         % Frequências normalizada
Wp2=fp2/(Fs/2);
Ws1=fs1/(Fs/2);
Ws2=fs2/(Fs/2);

[n,Wc]=buttord([Wp1 Wp2],[Ws1 Ws2],Rp,Rs); % Retorna ordem e frequência de
corte

[b,a]= butter(n,Wc, 'bandpass');        % Retorna os coeficientes a e b
```

APÊNDICE I - Alterações no código mostrado no Apêndice E para projeto do filtro IIR rejeita-banda.

```
%ESPECIFICAÇÕES DO FILTRO

Rp= 3;                                  % Ripple na banda de passagem
Rs= 40;                                 % Atenuação na banda de rejeição
fp1=1500;                               % Frequência de corte
fp2=6000;
fs1=50;                                 % Frequência de rejeição
fs2=8000;
Wp1=fp1/(Fs/2);                         % Frequências normalizada
Wp2=fp2/(Fs/2);
Ws1=fs1/(Fs/2);
Ws2=fs2/(Fs/2);

[n,Wc]=buttord([Wp1 Wp2],[Ws1 Ws2],Rp,Rs); % Retorna ordem e frequência de
corte

[b,a]= butter(n,Wc, 'stop');            % Retorna os coeficientes a e b
```

APÊNDICE J – Tarefa *SDCard_Task*.

```
void SDCard Task(void *args)
{
    FATFS_LinkDriverEx(&SD_Driver, (char *) pathName, 0);
    // Link entre SD driver e FAT

    SDCard Pin Configure();
    SPI_Configure(SPI_BaudRatePrescaler_16);

    f_mount(&FatFs, "", 0);


    do{
        fr = f_open(&AudioFile, "audio.wav", FA_READ);
    }while(fr != FR OK);

    ReadStatus = f_read(&AudioFile, (uint8_t *)&AudioWavHeader, 44, (UINT
    *)&nBytesLidos);

    if(ReadStatus == FR OK){
        AudioOffSet = 44 + AudioWavHeader.SubChunk2Size;
        f_lseek(&AudioFile, AudioOffSet);
        ReadStatus = f_read(&AudioFile, (uint8_t *)&AudioWavInfo, 8, (UINT
        *)&nBytesLidos);
    }

    while(1){
        ReadStatus = f_read(&AudioFile, (uint8_t *)&SDRawSamples.RawSamples[0],
        1024, (UINT *)&nBytesLidos);
        if(ReadStatus != FR_OK ){
        }
        Filter_Send_Raw_Samples_Block(&SDRawSamples);
    }
}
```

APÊNDICE K – Funções do *driver* SD.

```
void SDCard_Deselect(void)
{
    vTaskDelay(2);
    GPIO_SetBits(GPIOD, GPIO_Pin_0);
    vTaskDelay(2);
}

void SDCard_Select(void)
{
    vTaskDelay(2);
    GPIO_ResetBits(GPIOD, GPIO_Pin_0);
    vTaskDelay(2);
}

uint8_t SDCard_send_Command(uint8_t Command, uint32_t args)
{
    uint16_t i;
    uint8_t SDCCommand[6];
    uint8_t response;


    if (Command & 0x80) {
        Command &= 0x7F;
        response = SDCard_send_Command(CMD55, 0);
        if (response > 1){
            return response;
        }
    }

    SDCCommand[0] = 0x40 | Command;
    SDCCommand[1] = (uint8_t)(args >> 24);
    SDCCommand[2] = (uint8_t)(args >> 16);
    SDCCommand[3] = (uint8_t)(args >> 8);
    SDCCommand[4] = (uint8_t)args;

    SDCCommand[5] = 0xFF;

    if(Command == CMD0) SDCCommand[5] = 0x95;
    if(Command == CMD8) SDCCommand[5] = 0x87;

    spi_send_nbytes(SDCCommand, 6);

    for(i=0; i<SDCARD_MAX_TIMEOUT; i++){
        response = spi_send_byte(SDCARD_DUMMY_BYTE);
        if(response != 0xFF){
            break;
        }
        vTaskDelay(1);
    }
    return response;
}

volatile uint8_t dummy;

uint8_t read_block(uint8_t *buffer, uint32_t nData)
{
    uint32_t i;
    dummy = (uint8_t)SPI2->DR;
    for(i=0; i<nData; i++){
        SPI2->DR = (uint8_t)SDCARD_DUMMY_BYTE;
        while((SPI2->SR & SPI_SR_RXNE) != SPI_SR_RXNE);
        *(buffer++) = (uint8_t)SPI2->DR;
    }
    for(i=0; i<2; i++){
        SPI2->DR = (uint8_t)SDCARD_DUMMY_BYTE;
        while((SPI2->SR & SPI_SR_RXNE) != SPI_SR_RXNE);
```

```
            dummy = (uint8_t)SPI2->DR;
        }
        return 0;
    }

    uint8_t response2 = 0;
    uint32_t ate = 50000;

    uint8_t BSP_SD_ReadBlocks(uint32_t *buff, uint32_t sector, uint32_t nSectors,
    uint32_t timeOut)
    {
        uint16_t i, j;
        uint8_t status = FR_DISK_ERR;
        uint8_t response;


        if(nSectors == 1){

            if (!(SDCType & CT_BLOCK)) {
                sector *= 512;
            }
            if(SDCard_send_Command(CMD17,sector) == 0){
                for(i=0; i<timeOut; i++){
                    response = spi_send_byte(SDCARD_DUMMY_BYTE);
                    if(response != 0xFF){
                        break;
                    }
                    uint32_t i;
                    for(i=0; i<ate;i++);
                    //vTaskDelay(2);
                }
                if(response == 0xFE){
                    read_block((uint8_t *)buff, SDCARD_SECTOR_SIZE);
                    status = MSD_OK;

                }
            }
        }else{
            if (!(SDCType & CT_BLOCK)) {
                sector *= 512;
            }

            response2 = SDCard_send_Command(CMD18, sector);
            if(response2 == 0){
                for(i=0; i<nSectors; i++){
                    for(j=0; j<SD_DATATIMEOUT; j++){
                        response = spi_send_byte(SDCARD_DUMMY_BYTE);
                        if(response == 0xFE){
                            read_block((uint8_t *)(((uint32_t)buff)+(i*512)),
                            SDCARD_SECTOR_SIZE);
                            vTaskDelay(1);
                            break;
                        }
                        vTaskDelay(1);
                    }
                    if(j >= SD_DATATIMEOUT){
                        status = FR_DISK_ERR;
                        break;
                    }
                }

                if(i >= nSectors){
                    response = SDCard_send_Command(CMD12, sector);
                    while(spi_send_byte(SDCARD_DUMMY_BYTE) != 0xFF);
                    status = MSD_OK;
                }
            }
        }

    return status;
}

uint8_t BSP_SD_Init(void)
```

```
{
    uint16_t i;
    uint8_t OCR[4];
    uint8_t status = FR_DISK_ERR;
    uint8_t command = 0;
    uint8_t CommandResponse[20];
    uint8_t CommandStatus = 0;

    SDCard_Deselect();
    for(i=0; i<10; i++)
    {
        spi_send_byte(0xFF);
    }
    SDCard_Select();

    do{
        CommandStatus = SDCard_send_Command(CMD0, 0);
    }while(CommandStatus != 1);
    if (CommandStatus == 1){
        vTaskDelay(1000);
        // 1s
        if (SDCard_send_Command(CMD8, 0x1AA) == 0x01){
            for (i = 0; i < 4; i++) {
                OCR[i] = spi_send_byte(0xFF);
            if (OCR[2] == 0x01 && OCR[3] == 0x01) {
                for(i=0; i < SDCARD_MAX_TIMEOUT; i++){
                    CommandResponse[i] = SDCard_send_Command(ACMD41, 0x40000000);
                    if(CommandResponse[i] == 0){
                        break;
                    }
                    vTaskDelay(1);
                }
                if(i<SDCARD_MAX_TIMEOUT){
                    if(SDCard_send_Command(CMD58, 0) == 0) {
                        for (i = 0; i < 4; i++) {
                                OCR[i] = spi_send_byte(0xFF);
                          }
                          SDCType = (OCR[0] & 0x40) ? CT_SD2 | CT_BLOCK : CT_SD2;
                       }
                  }
              }
           }else{
              if(SDCard_send_Command(ACMD41, 0) <= 1){
                  SDCType = CT_SD1;
                  command = ACMD41;
              } else {
                  SDCType = CT_MMC;
                  command = CMD1;
              }
              for(i=0; i<SDCARD_MAX_TIMEOUT; i++){
    // Espera o fim da inicialização
                  if(SDCard_send_Command(command, 0) == 0){
                      break;
                  }
              }
              if(i < SDCARD_MAX_TIMEOUT){
                  if(SDCard_send_Command(CMD16, 512) != 0){
                      SDCType = 0;
                  }
              }
          }
    }

    if(SDCType){
        status = MSD_OK;
        SDCStatus = MSD_OK;
    }

    return status;
}
```

APÊNDICE L – Configurações das GPIO e do periférico I2S.

```
void SDCard_Pin_Configure(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    // Habilita o clock dos barramentos do periférico

    GPIO_InitTypeDef GPIOInit;
    //Inicializa estrutura
    GPIOInit.GPIO_Pin = GPIO_Pin_13| GPIO_Pin_15;
    // Pinos utilizados
    GPIOInit.GPIO_Mode = GPIO_Mode_AF;
    // Modo de operação
    GPIOInit.GPIO_Speed = GPIO_Speed_25MHz;
    // Velocidade
    GPIOInit.GPIO_OType = GPIO_OType_PP;
    // Conf. tipo de saída como push_pull
    GPIOInit.GPIO_PuPd = GPIO_PuPd_NOPULL;
    // Desabilita resistor de saída
    GPIO_Init(GPIOB, &GPIOInit);
    // Inicializa a estrutura

    GPIOInit.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOC, &GPIOInit);

    GPIOInit.GPIO_Pin = GPIO_Pin_0;
    GPIO_Init(GPIOD, &GPIOInit);

    //Funcao alternativa dos pinos
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource13, GPIO_AF_SPI2);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_SPI2);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_SPI2);

}
```

```
void SPI_Configure(uint16_t prescaler)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
    // Habilita o clock do barramento periférico


    SPI_Cmd(SPI2, DISABLE);
    // Desabilita a SPI
    SPI_InitTypeDef SPIOInit;
    // Inicializa a estrutura
    SPIOInit.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    // Separa o MOSI e MISO
    SPIOInit.SPI_Mode = SPI_Mode_Master;
    // Modo de operação MESTRE
    SPIOInit.SPI_DataSize = SPI_DataSize_8b;
    // Tamanho dados SPI e 8 bits
    SPIOInit.SPI_CPOL = SPI_CPOL_Low;
    // Polaridade
    SPIOInit.SPI_CPHA =  SPI_CPHA_1Edge;
    // Fase
    SPIOInit.SPI_NSS = SPI_NSS_Soft;
    // Sinal NSS gerenciado por software
    SPIOInit.SPI_BaudRatePrescaler = prescaler;
    // Frequência SPI e igual a frenquecia  APB2/prescaler
    SPIOInit.SPI_FirstBit = SPI_FirstBit_MSB;
    // Bit mais significativo primeiro
    SPIOInit.SPI_CRCPolynomial = 7;
    // Polinômio utilizado
    SPI_Init(SPI2, &SPIOInit);
    // Inicializa a SPI com as consigurações dadas
    SPI_Cmd(SPI2, ENABLE);
    // Habilita a SPI
}
```

APÊNDICE M – Tarefa *Filter_Task*.

```
void Filter_Task(void *args)
{
    uint32_t i,j;
    AudioBlockQueue = xQueueCreateStatic( AUDIO_BLOCK_QUEUE_LENGTH,
                                          AUDIO_BLOCK_QUEUE_SIZE,
                                          AudioBlockQueueStorageArea,
                                          &AudioBlockStaticQueue);
    Filter_Led_Pin_Configure();


    SampleFilter_init(&Filter);

    while(1){
        GPIO_ToggleBits(GPIOD, GPIO_Pin_14);
        xQueueReceive(AudioBlockQueue, &AudioBlockToProcess, portMAX_DELAY);

        j = 0;
        for(i=0; i<1024; i+=2){
            AudioBlockProcessed.Samples[i] = AudioBlockToProcess.Samples[j];
            if(FilterSelect == FIR){
                SampleFilter_put(&Filter, AudioBlockToProcess.Samples[j]);
                AudioBlockProcessed.Samples[i+1] = (int16_t)SampleFilter_get(&Filter);
            }else{
                AudioBlockProcessed.Samples[i+1] = IIRSampleFilter_run(&Filter,
                AudioBlockToProcess.Samples[j]);
            }
            j++;
        }
        Codec_Send_Audio_Block(&AudioBlockProcessed);

        for(i=0; i<1024; i+=2){
            AudioBlockProcessed.Samples[i] = AudioBlockToProcess.Samples[j];

       if(FilterSelect == FIR){
           SampleFilter_put(&Filter, AudioBlockToProcess.Samples[j]);
           AudioBlockProcessed.Samples[i+1] = (int16_t)SampleFilter_get(&Filter);
       }else{
           AudioBlockProcessed.Samples[i+1] = IIRSampleFilter_run(&Filter,


               AudioBlockToProcess.Samples[j]);
            }
            j++;
       }

       Codec_Send_Audio_Block(&AudioBlockProcessed);
    }

}
```

APÊNDICE N – Funções de inicialização e filtragem.

```
void SampleFilter_init(SampleFilter* f) {
    int i;
    for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
        f->history[i] = 0;
    for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
        f->OutputHistory[i] = 0;
    f->last_index = 0;
    f->OutputIndex = 0;
}

void SampleFilter_put(SampleFilter* f, float input) {
  f->history[f->last_index++] = input;
  if(f->last_index >= SAMPLEFILTER_TAP_NUM)
    f->last_index = 0;
}

// FILTRO FIR
float SampleFilter_get(SampleFilter *f) {
  float acc = 0;

  int index = f->last_index, i;
  for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i) {
    index = index != 0 ? index-1 : SAMPLEFILTER_TAP_NUM-1;
    acc += f->history[index] * filter_taps[i];
  };
  return acc;
}

// FILTRO IIR
float IIRSampleFilter_run(SampleFilter *f, float input) {
    float acc1 = 0, acc2 = 0;
    int index, i;

    f->history[f->last_index++] = input;
    if(f->last_index >= SAMPLEFILTER_TAP_NUM)
        f->last_index = 0;

    index = f->last_index;
    for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i) {
        index = index != 0 ? index-1 : SAMPLEFILTER_TAP_NUM-1;
        acc1 += f->history[index] * input_filter_taps[i];
    }
    index = f->OutputIndex;
    for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i) {
        index = index != 0 ? index-1 : SAMPLEFILTER_TAP_NUM-1;
        acc2 += f->OutputHistory[index] * output_filter_taps[i];
    }
    acc1 -= acc2;

    f->OutputHistory[f->OutputIndex++] = acc1;
    if(f->OutputIndex >= SAMPLEFILTER_TAP_NUM)
        f->OutputIndex = 0;

    return acc1;
}
```

APÊNDICE O – Tarefa *Codec_Task*.

```
void Codec_Task(void *args)
{
    uint32_t i = 0;

    CodecQueue = xQueueCreateStatic(CODEC_QUEUE_LENGTH,
                                    CODEC_QUEUE_SIZE,
                                    CodecQueueStorageArea,
                                    &CodecStaticQueue);

    // Chama as funcões de configuração
    Codec_Led_Pin_Configure();
    I2S_Pin_Configure();
    I2C_Pin_Configure();
    I2C_Configure();
    DMA_Configure();
    codec_ctrl_init();

    // Preenche dois blocos iniciais
    for(i=0; i < 2; i++){
        xQueueReceive(CodecQueue, &AudioBlockArray[i], portMAX_DELAY);
    }
    AudioBlockArrayIndex = 1;
    I2S_Configure();
    DMA_Launch((uint32_t)&AudioBlockArray[0]);
    SPI_I2S_SendData(SPI3, 0x0000);


    while(1){
        vTaskDelay(500);
    }
}
```

APÊNDICE P – Tratamento da interrupção do DMA.

```
void DMA1_Stream5_IRQHandler(void)
{
    GPIO_ToggleBits(GPIOD, GPIO_Pin_15);
    // Pisca LED
    static BaseType_t xHigherPriorityTaskWoken;
    DMA_ClearFlag(DMA1_Stream5, DMA_FLAG_TCIF5 | DMA_FLAG_HTIF5 | DMA_FLAG_TEIF5 |
    DMA_FLAG_DMEIF5);
    //Limpeza das flags de interrupção

    // DMA no canal 5
    DMA1_Stream5->M0AR = (uint32_t)&AudioBlockArray[AudioBlockArrayIndex].Samples[0];
    // Endereço 0 do bloco recebe o primeiro
    DMA1_Stream5->NDTR = 1024;
    // Quantidade de amostras por bloco
    DMA1_Stream5->CR |= DMA_SxCR_EN;
    // Registrador de configuração
    AudioBlockArrayIndex++;
    // Incremento de index do bloco
    if(AudioBlockArrayIndex >= 2) AudioBlockArrayIndex = 0;
    if(xQueueReceiveFromISR(CodecQueue, &AudioBlockArray[AudioBlockArrayIndex],
    &xHigherPriorityTaskWoken) == pdFALSE){
        if(AudioBlockArrayIndex >= 2){
            AudioBlockArrayIndex = 0;
            // Volta ao bloco 0
        }
    }
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
```

APÊNDICE Q – Configurações das GPIO e do periférico I2S.

```
void I2S_Pin_Configure(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    // Habilita o clock para o barramento do periférico


    GPIO_InitTypeDef PinInitStruct;
    // Inicializa a estrutura
    PinInitStruct.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_7 | GPIO_Pin_12;
    // Pinos utilizados
    PinInitStruct.GPIO_Mode = GPIO_Mode_AF;
    // Modo de operação aleternativa
    PinInitStruct.GPIO_OType = GPIO_OType_PP;
    // Conf. saída para push-pull
    PinInitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    // Desabilita o resistor pull-up
    PinInitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    // Velocidade das GPIO
    GPIO_Init(GPIOC, &PinInitStruct);
    // Inicializa a GPIO com as especificações

    PinInitStruct.GPIO_Pin = GPIO_Pin_4;
    GPIO_Init(GPIOA, &PinInitStruct);

   // Função alternativa dos pinos para I2S
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource10, GPIO_AF_SPI3);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_SPI3);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource12, GPIO_AF_SPI3);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource4, GPIO_AF_SPI3);
}

 void I2S_Configure(void)
 {
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI3, ENABLE);
     // Clock do barramento habilitado


     I2S_InitTypeDef I2S_InitType;
     // Inicializa a estrutura
     I2S_InitType.I2S_Mode = I2S_Mode_MasterTx;
     // Modo de operação mestre
     I2S_InitType.I2S_Standard = I2S_Standard_Phillips;
     // Padrão utilizado
     I2S_InitType.I2S_DataFormat = I2S_DataFormat_16b;
     // Formato dos dados
     I2S_InitType.I2S_MCLKOutput = I2S_MCLKOutput_Enable;
     // Saída do MCLK ativa
     I2S_InitType.I2S_AudioFreq = I2S_AudioFreq_44k;
     // Frequência de operação
     I2S_InitType.I2S_CPOL = I2S_CPOL_Low;
     // Polaridade
     I2S_Init(SPI3, &I2S_InitType);
     // Inicializa a SPI3 com a estrutuda acima


     SPI_I2S_DMACmd(SPI3, SPI_I2S_DMAReq_Tx, ENABLE);
     // Habilita a interface DMA e I2S
     I2S_Cmd(SPI3, ENABLE);
     // Habilita o clock para o I2S
 }
```

APÊNDICE R – Configurações das GPIO e do periférico I2C.

```
void I2C_Pin_Configure(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOB , ENABLE);
    // Habilita o clock para os periféricos

    GPIO_InitTypeDef PinInitStruct;
    // Inicializa a estrutura
    PinInitStruct.GPIO_Pin = GPIO_Pin_4;
    // Pinos utilizados
    PinInitStruct.GPIO_Mode = GPIO_Mode_OUT;
    // Modo de operação saída
    PinInitStruct.GPIO_OType = GPIO_OType_PP;
    // Config. saída como push-pull
    PinInitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    // Desabilita resistor de pull-up
    PinInitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    // Velocidade da GPIO
    GPIO_Init(GPIOD, &PinInitStruct);

    PinInitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_9;
    // Pinos utilizados
    PinInitStruct.GPIO_Mode = GPIO_Mode_AF;
    // Modo de operação alternativa
    PinInitStruct.GPIO_OType = GPIO_OType_OD;
    // Confi. saída para open-drain -linha é puxada para baixo
    PinInitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    // Desabilita resistor de pull-up
    PinInitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    // Velocidade das GPIO
    GPIO_Init(GPIOB, &PinInitStruct);


    // Função alternativa do pinos - P/I2C
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_I2C1);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource9, GPIO_AF_I2C1);
}

void I2C_Configure(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    // Habilita clock para o barramento do periférico

    I2C_InitTypeDef I2C_InitType;
    // Inicializa estrutura
    I2C_InitType.I2C_ClockSpeed = 100000;
    // Frequência
    I2C_InitType.I2C_Mode = I2C_Mode_I2C;
    // Modo de operação
    I2C_InitType.I2C_DutyCycle = I2C_DutyCycle_2;
    // Modo de duty cycle 50% -standard
    I2C_InitType.I2C_OwnAddress1 = 0xF0;
    // Endereço único (sem relevância em modo mestre)
    I2C_InitType.I2C_Ack = I2C_Ack_Disable;
    // Desabilita o acknowledged
    I2C_InitType.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    // Endereços de 7 bits
    I2C_Init(I2C1, &I2C_InitType);
    // Inicializa a I2C
}
```

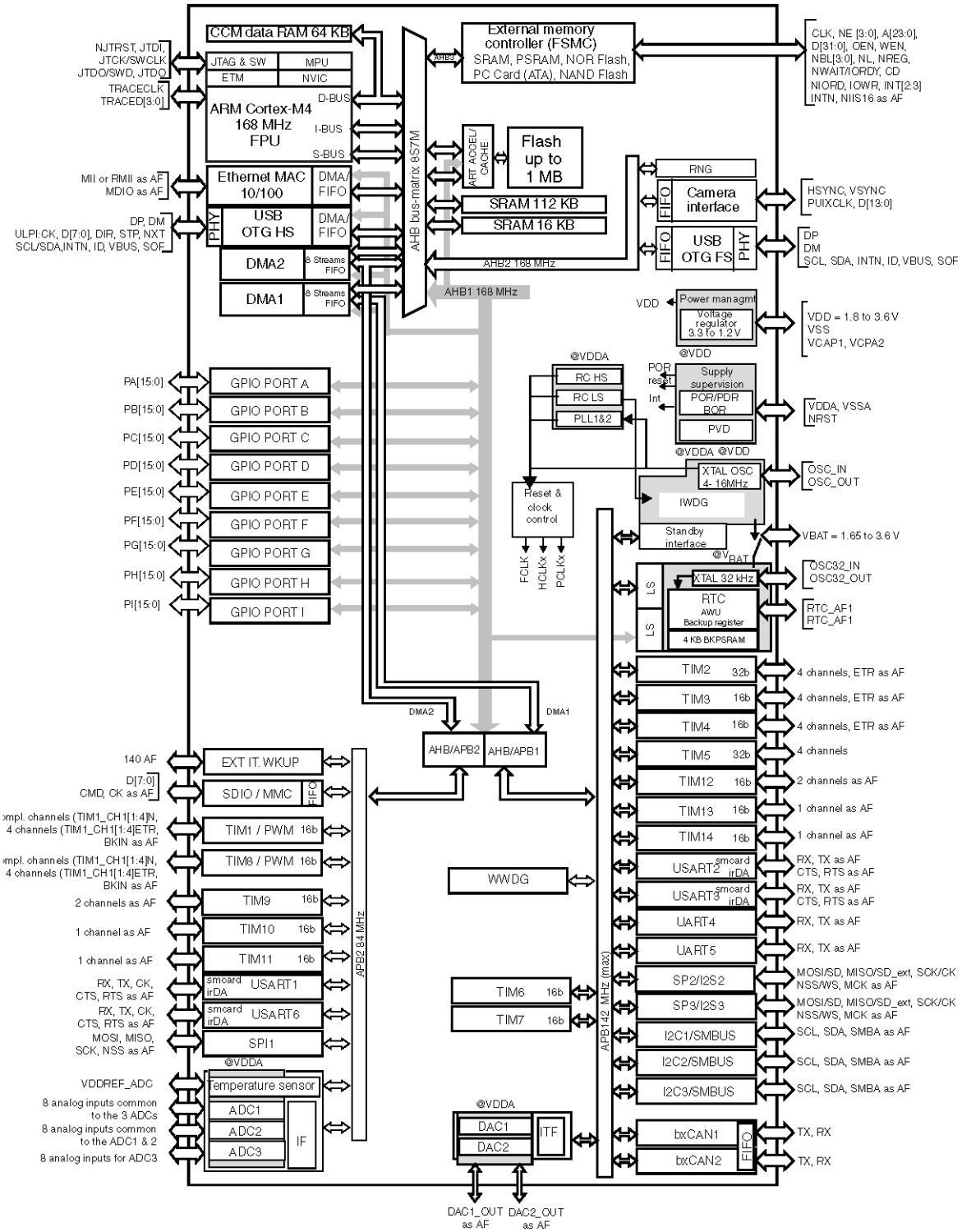
APÊNDICE S – Configurações do DMA.

```
void DMA_Configure(void){

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC | RCC_AHB1Periph_DMA1, ENABLE);
    // Habilita o clock no barramento do periférico

    DMA_InitStruct.DMA_Channel = DMA_Channel_0;
    // Especifica o canal
    DMA_InitStruct.DMA_Memory0BaseAddr = (uint32_t)0x00000000;
    // Especifica a memória zero
    DMA_InitStruct.DMA_DIR = DMA_DIR_MemoryToPeripheral;
    // Especifica a direção do dado (Memória paro o periférico)
    DMA_InitStruct.DMA_BufferSize = 1024;
    // Especifica o tamanho do buffer
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t) &SPI3->DR;
    // Especifica o endereço do periférico base
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    // Especifica se o registrador de endereço incrementa ou não
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
    // Especifica se o registrador de endereço de memória incrementa
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    // Espercifica a largura dos dados no periférico
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    // Espercifica a largura dos dados na memória
    DMA_InitStruct.DMA_Mode = DMA_Mode_Normal;
    // Modo de operação
    DMA_InitStruct.DMA_Priority =  DMA_Priority_VeryHigh;
    // Especifica a prioridade do software
    DMA_InitStruct.DMA_FIFOMode = DMA_FIFOMode_Disable;
    // Especifica se o modo FIFO pode ser usado
    DMA_InitStruct.DMA_FIFOThreshold = DMA_FIFOThreshold_1QuarterFull;
    // Especifica o limiar FIFO

    DMA_InitStruct.DMA_MemoryBurst = DMA_MemoryBurst_Single;
    // Especifica a quantidade de dados a serem transferidos em uma
    // transação
    DMA_InitStruct.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
    // Especifica a quantidade de dados a serem transferidos em uma
    // transação
    DMA_Init(DMA1_Stream5, &DMA_InitStruct);
    // Inicializa o DMA com as configurações

    NVIC_InitStruct.NVIC_IRQChannel = DMA1_Stream5_IRQn;
    // Especifica para qual canal a interrupção é habilitada
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    // Especifica se o canal está habilitado ou não
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority =
    configMAX_SYSCALL_INTERRUPT_PRIORITY+5;
    // Especifica prioridade
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    // Especifica subprioridade
    NVIC_Init(&NVIC_InitStruct);
    // Inicializa a interrupção para o DMA
    DMA_ClearFlag(DMA1_Stream5, DMA_FLAG_TCIF5 | DMA_FLAG_HTIF5 | DMA_FLAG_TEIF5 |
    DMA_FLAG_DMEIF5);
    // Limpa as flags de interrupção
    DMA_ITConfig(DMA1_Stream5, DMA_IT_TC, ENABLE);
    // Habilita as interrupção para o DMA
    DMA_Cmd(DMA1_Stream5, ENABLE);
    // Habilita o clock para o periférico
}
```

APÊNDICE T – Configurações do CODEC.

```
void codec_ctrl_init(void)
{
    uint8_t CodecCommandBuffer[5];

uint8_t regValue = 0xFF;

Codec_Set();
vTaskDelay(10);
//keep codec OFF
CodecCommandBuffer[0] = CODEC_MAP_PLAYBACK_CTRL1;
CodecCommandBuffer[1] = 0x01;
send_codec_ctrl(CodecCommandBuffer, 2);

//begin initialization sequence (p. 32)
CodecCommandBuffer[0] = 0x00;
CodecCommandBuffer[1] = 0x99;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = 0x47;
CodecCommandBuffer[1] = 0x80;
send_codec_ctrl(CodecCommandBuffer, 2);

regValue = read_codec_register(0x32);

CodecCommandBuffer[0] = 0x32;
CodecCommandBuffer[1] = regValue | 0x80;
send_codec_ctrl(CodecCommandBuffer, 2);

regValue = read_codec_register(0x32);

CodecCommandBuffer[0] = 0x32;
CodecCommandBuffer[1] = regValue & (~0x80);
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = 0x00;
CodecCommandBuffer[1] = 0x00;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = CODEC_MAP_PWR_CTRL2;
CodecCommandBuffer[1] = 0xAF;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = CODEC_MAP_PLAYBACK_CTRL1;
CodecCommandBuffer[1] = 0x70;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = CODEC_MAP_CLK_CTRL;
CodecCommandBuffer[1] = 0x81; //auto detect clock
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = CODEC_MAP_IF_CTRL1;
CodecCommandBuffer[1] = 0x07;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = 0x0A;
CodecCommandBuffer[1] = 0x00;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = 0x27;
CodecCommandBuffer[1] = 0x00;
send_codec_ctrl(CodecCommandBuffer, 2);

CodecCommandBuffer[0] = 0x1A | CODEC_MAPBYTE_INC;
CodecCommandBuffer[1] = 0x0A;
CodecCommandBuffer[2] = 0x0A;
send_codec_ctrl(CodecCommandBuffer, 3);

CodecCommandBuffer[0] = 0x1F;
CodecCommandBuffer[1] = 0x0F;
send_codec_ctrl(CodecCommandBuffer, 2);

   CodecCommandBuffer[0] = CODEC_MAP_PWR_CTRL1;
   CodecCommandBuffer[1] = 0x9E;
   send_codec_ctrl(CodecCommandBuffer, 2);
}
```

ANEXO A – Diagrama de blocos STM32F40x



ANEXO B – Diagrama de blocos CS43L22