Problema 4: Implementação de Filtro em Dispositivo Embarcado

1st Allan Capistrano de Santana Santos
Universidade Estadual de Feira de Santana
Feira de Santana - BA, Brasil
asantos@ecomp.uefs.br

2nd Daniel Fernandes Campos
Universidade Estadual de Feira de Santana
Feira de Santana - BA, Brasil
dfc152@gmail.com

3rd João Erick Barbosa Teixeira da Silva
Universidade Estadual de Feira de Santana
Feira de Santana - BA, Brasil
jsilva@ecomp.uefs.br

4th João Pedro Rios Carvalho
Universidade Estadual de Feira de Santana
Feira de Santana - BA, Brasil
jprcarvalho1@gmail.com

***Resumo*—Este relatório descreve o processo de implementação de filtros digitais implementados em um dispositivo embarcado programável, para que os sinais de áudio processados em tempo real, sejam transmitidos sem interferências que possam comprometer a qualidade e a clareza da informação.**

***Index Terms*—filtros digitais, sinais de áudio, dispositivo embarcado**

I. Introdução

A filtragem do áudio desempenha um papel crucial nas telecomunicações, sendo essencial para garantir a qualidade e a clareza das comunicações. Possibilita uma melhor compreensão das mensagens transmitidas, reduzindo a interferência e os distúrbios que podem prejudicar a comunicação.

O estudo realizado junto aos institutos de pesquisa revelou que o filtro FIR desenvolvido pela *Sigma Delta Inc.* apresentou os melhores resultados em termos de custo-benefício para análises desse tipo. Em decorrência disto, a equipe foi designada a implementar uma nova solução de filtros digitais, levando em consideração as limitações de implementação em dispositivos embarcados, que possuem menor precisão em comparação aos computadores convencionais.

Este projeto tem como objetivo, implementar na prática filtros digitais capazes de eliminar, ou ainda, minimizar componentes de frequências que possam interferir na qualidade do áudio que será transmitido pelo dispositivo embarcado programável *MPLAB Starter Kit for dsPIC DM3330011*. A partir disso, será possível avaliar os efeitos da implementação desses filtros em dispositivos de menor precisão, bem como realizar uma análise da escolha da quantidade de coeficientes dos filtros e na largura de banda adequada para obter os melhores resultados.

Este relatório descreverá detalhadamente as etapas e os procedimentos adotados para a implementação dos filtros digitais, bem como os resultados obtidos a partir da aplicação da filtragem no dispositivo em questão. Espera-se fornecer uma visão geral sobre a solução proposta, destacando benefícios, limitações e impactos esperados na área das telecomunicações.

II. Desenvolvimento

Esta seção tem como objetivo apresentar todos os conceitos necessários para o desenvolvimento do projeto, abrangendo desde as ferramentas utilizadas, até os fundamentos matemáticos relacionados ao processo de filtragem de um sinal de áudio.

A. Dispositivo embarcado e softwares utilizados

O desenvolvimento e implementação dos filtros contaram com o uso do *software MatLab* versão 2019a. Mais especificamente, foi empregada a ferramenta *FDAtool*, que proporciona uma interface gráfica para a modelagem de diversos tipos de filtros, abrangendo desde os filtros IIR (*Infinite Impulse Response*) até os filtros FIR (*Finite Impulse Response*).

O filtro será implementado no dispositivo embarcado *MPLAB Starter Kit for dsPIC DM3330011*. Esse dispositivo possui uma memória *flash* de 256kB, entrada para microfone, saídas para fone de ouvido/alto-falante, interruptores reconfiguráveis, potenciômetros, um sensor de temperatura e uma *EEPROM* serial de 4MB, destinada ao armazenamento de dados, como amostras de áudio [3].

O código a ser implementado no dispositivo foi desenvolvido em linguagem de programação *C*, sendo recomendado utilizar a *IDE* (Ambiente de desenvolvimento integrado - *Integrated Development Environment*) fornecida pelo fabricante. Neste caso, utilizou-se a *MPLAB® X IDE* versão 6.05. Para compilar o código, empregou-se o compilador *MPLAB XC16 Compiler* versão 2.10, também disponibilizado pelo fabricante.

B. Filtro FIR

O filtro FIR opera em sinais digitais e é caracterizado por ter uma resposta ao impulso com duração finita. Com isso, ele é próprio para sistemas de memória limitada, BIBO (*Bounded Input - Bounded Output*) estáveis e com resposta desejada de fase linear. Sua função de transferência é definida por:

$$H(z) = \sum_{n=0}^{M} b_n \cdot z^{-n}$$

C. Filtragem

A filtragem consiste em processo que envolve a aplicação de uma operação sobre um sinal de entrada, com o objetivo de remover elementos indesejados, como ruídos, deixando apenas a informação de interesse. Dessa forma, é possível obter um sinal mais limpo e preciso.

Como pode ser observado na Fig. 1, ao passar o sinal de entrada $x[n]$ pelo sistema $h[n]$, obtém-se o sinal filtrado $y[n]$. Essa operação pode ser representada no domínio de n como uma convolução [1]:

$$y[n] = x[n] * h[n]$$
$$y[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot x[n-k]$$

Aplicando a Transformada de Tempo Discreto de Fourier, obtém-se:

$$Y(e^{j\Omega}) = X(e^{j\Omega}) \cdot H(e^{j\Omega})$$

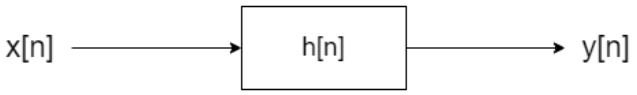

Figura 1. Diagrama de filtragem.

1) Projeto de Filtro FIR por janelamento: O método de janelamento é o mais simples para o projeto de filtros do tipo FIR. Esse método utiliza a resposta em frequência de um filtro ideal [1]. Essa resposta pode ser representada como:

$$D(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} d[n] \cdot e^{-j\Omega n}$$

E o filtro no domínio de tempo discreto é obtido pela Transformada Inversa de Fourier de Tempo Discreto, como:

$$d[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(e^{j\Omega}) e^{j\Omega n}\, d\Omega$$

$D(e^{j\Omega})$ é um filtro passa-baixa retangular com amplitude igual a 1 e limitado pela frequência de corte Ω_c, como pode ser visto abaixo:

$$D(e^{j\Omega}) = \begin{cases} 1, & |\Omega| \leq \Omega_c \\ 0, & \Omega_c \leq |\Omega| \leq \pi \end{cases}$$

Então, $d[n]$ será:

$$d[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} D(e^{j\Omega}) e^{j\Omega n}\, d\Omega$$
$$d[n] = \frac{1}{2\pi} \int_{-\Omega_c}^{\Omega_c} 1 \cdot e^{j\Omega n}\, d\Omega$$
$$d[n] = \frac{1}{2\pi jn} [e^{j\Omega_c n} - e^{-j\Omega_c n}]$$
$$d[n] = \frac{sen(\Omega_c n)}{\pi n}, -\infty \leq n \leq \infty$$

Para efeito de implementação em *hardware*, é necessário truncar a função $d[n]$, dado que a mesma apresenta duração infinita e funções dessa natureza não podem ser implementadas em meio digital [2]. O truncamento resultará em um filtro aproximado, com características distintas do filtro original.

Além disso, como filtros FIR são filtros causais, o que significa que só levam em consideração um número finito de amostras anteriores do sinal de entrada para calcular o valor do sinal de saída atual, é necessário realizar um deslocamento para que $d[n]$ tenha valores apenas a partir de $n = 0$, por conta disso:

$$h_d[n] = d[n - \tfrac{M}{2}]$$
$$h_d[n] = \frac{sen((n-\frac{M}{2}) \cdot \Omega_{cN})}{\pi(n-\frac{M}{2})}$$

Em que M é a quantidade de coeficientes do filtro e Ω_{cN} é a frequência de corte normalizada, o cálculo de ambos serão demonstrados no próximo tópico.

Dessa forma, para resolver o problema de truncamento, a função $h_d[n]$ é limitada a partir de uma janela $w[n]$. Esse processo pode ser expresso analiticamente como:

$$h[n] = h_d[n] \cdot w[n]$$

2) Janela de Kaiser: A janela de *Kaiser* é definida como [1]:

$$w[n] = \begin{cases} \frac{I_0[\beta(1-[(n-\alpha)/\alpha]^2)^{\frac{1}{2}}]}{I_0(\beta)}, & 0 \leq n \leq M \\ 0, & c.c. \end{cases}$$

sendo $\alpha = M/2$ e $I_0(.)$ representa a função de *Bessel* modificada de ordem zero.

$$I_0(x) = 1 + \sum_{k=1}^{\infty} [\frac{(x/2)^k}{k!}]^2$$

Definido $A = -20\log_{10}(\delta)$, *Kaiser* determinou empiricamente que o valor de β necessário para alcançar um valor especificado de *A* é dado por:

$$\beta = \begin{cases} 0,1102(A-8,7), & A > 50, \\ 0,5842(A-21)^{0,4} + 0,07886(A-21), & 21 \leq A \leq 50, \\ 0, & A < 21 \end{cases}$$

Além disso, *Kaiser* descobriu que, para alcançar os valores prescritos de A e Δw, M precisa satisfazer:

$$M = \frac{A-8}{2,285\Delta w}$$

onde $\Delta w = w_p - w_s$, isto é, frequência de passagem menos frequência de rejeição.

Vale ressaltar que para esse projeto, como foi utilizada a ferramenta *FDAtool*, não foram necessários calcular analiticamente os parâmetros para a janela de *Kaiser*.

D. Reamostragem do sinal de entrada

O sinal de áudio que contém ruído possui uma taxa de amostragem de 44100Hz. No entanto, ao observar as Figuras 2 e 3, pode-se notar que a partir de 3600Hz, há apenas ruído presente no áudio, e, como visto em [1], as faixas de frequência que incorporam a voz humana estão presentes de até 3kHz a 4kHz. Então, de acordo com o teorema de Nyquist, que estabelece que a taxa de amostragem deve ser pelo menos duas vezes maior do que a maior frequência do sinal, é possível realizar uma reamostragem nesse sinal de entrada para 8kHz sem perda de informações.

Ao reduzir a taxa de amostragem, o desempenho do processo de filtragem melhora, pois há menos amostras para processar. Isso é especialmente importante ao utilizar um dispositivo embarcado com recursos limitados. O espectro do sinal reamostrado pode ser visualizado na Figura 4.

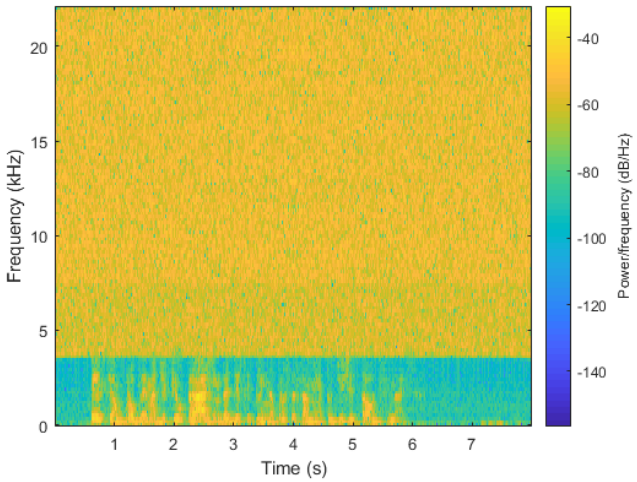

Figura 2. Espectrograma do sinal de entrada.

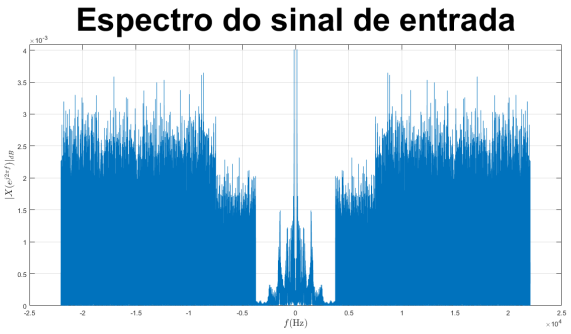

Figura 3. Espectro do sinal de áudio ruidoso.

Ao realizar a reamostragem para uma taxa de amostragem menor, as informações relevantes do sinal são preservadas ao mesmo tempo que simplifica o processamento posterior do sinal, economizando recursos computacionais e permitindo um melhor aproveitamento dos dispositivos embarcados. O processo de reamostragem foi realizado no *MatLab* através da função *resample()*.

Figura 4. Espectro do sinal de áudio ruidoso reamostrado.

Esse processo de reamostragem foi realizado para todos os áudios utilizados no desenvolvimento deste projeto.

E. Filtros Projetados

Neste problema, foi solicitado o desenvolvimento de quatro tipos de filtros FIR: passa-baixa, passa-alta, passa-faixa e rejeita-faixa. Para atender a essa demanda, como citado anteriormente, utilizou-se a ferramenta *FDAtool* para a modelagem e obtenção dos coeficientes de cada filtro. Nas próximas seções, serão apresentadas as características individuais de cada filtro, destacando suas funcionalidades e aplicações.

1) Filtro passa-baixa: Um filtro passa-baixa é um tipo de filtro utilizado em processamento de sinais para atenuar ou eliminar componentes de alta frequência de um sinal. Ele permite a passagem de frequências mais baixas enquanto atenua ou bloqueia as frequências mais altas.

A Figura 5 ilustra a utilização da janela de *Kaiser* no projeto do filtro passa-baixa. Nessa configuração específica, foram definidos os seguintes parâmetros: uma frequência de amostragem (F_s) de 8kHz, uma frequência de passagem (F_{pass}) de 1kHz e uma frequência de rejeição (f_{stop}) de $2,6$kHz. Além disso, foram estabelecidas uma atenuação mínima (A_{pass}) de 1dB e uma atenuação máxima (A_{stop}) de 60dB.

Com base nessas configurações, o filtro resultante possui uma ordem (N) de 19, o que implica a utilização de 20 coeficientes (M) em sua implementação.

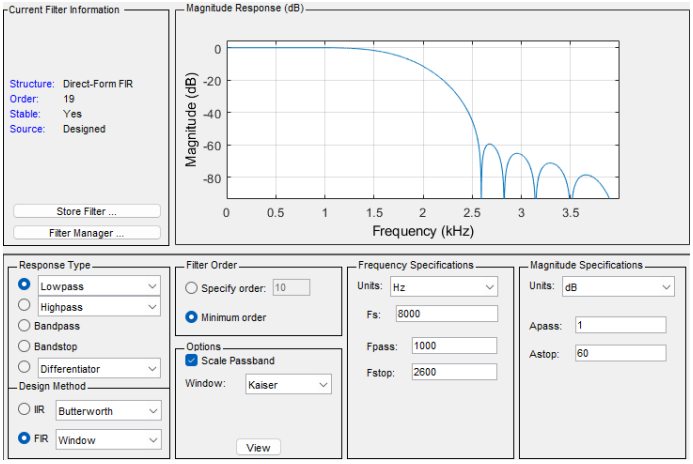

Figura 5. Filtro FIR passa-baixa.

2) Filtro passa-alta: Um filtro passa-alta é o oposto de um filtro passa-baixa, isto é, ele atenua frequências mais baixas e permite a passagem de frequências mais altas.

Na Figura 6, é possível observar a aplicação da janela de *Kaiser* no projeto do filtro passa-alta. Nessa configuração, foram utilizados os seguintes parâmetros: uma frequência de amostragem de 8kHz, uma frequência de passagem de 2kHz e uma frequência de rejeição de 3520Hz. Além disso, foram definidas uma uma atenuação máxima de 60dB e atenuação mínima de 1dB.

A partir dessas configurações, o filtro resultante possui uma ordem de 20, ou seja, 21 coeficientes em sua implementação.

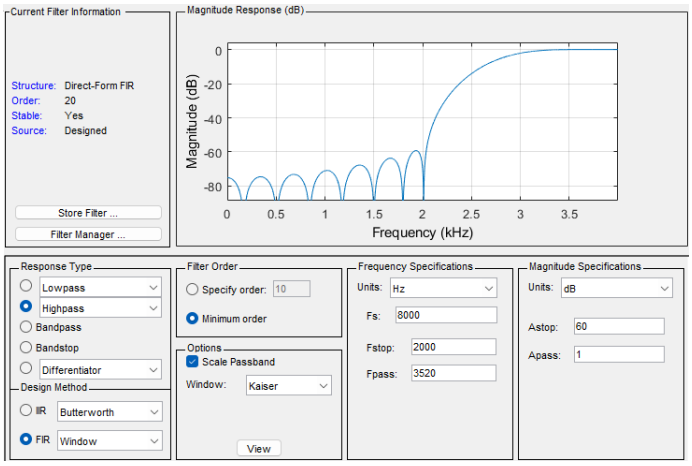

Figura 6. Filtro FIR passa-alta.

3) Filtro passa-faixa: Um filtro passa-faixa é um tipo de filtro utilizado em processamento de sinais para permitir a passagem de um intervalo específico de frequências, enquanto atenua as frequências fora desse intervalo. Ele é projetado para selecionar e realçar uma banda de frequências desejada, enquanto rejeita ou atenua as frequências mais altas e mais baixas.

Como pode ser observado na Fig 7, também foi utilizada a janela de *Kaiser*, com uma frequência de amostragem de 8kHz, frequências de passagem de 900Hz e 1100Hz, e frequências de rejeição de 400Hz e 1600Hz, além de atenuação mínima de 1dB e atenuações máximas de 40dB. O que resultou em um filtro com 37 coeficientes.

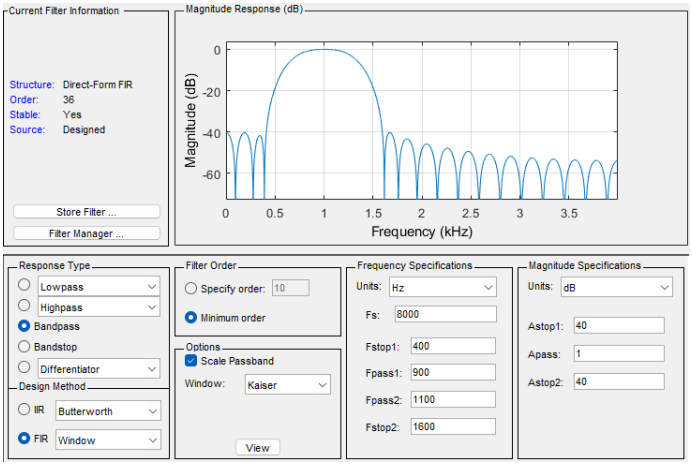

Figura 7. Filtro FIR passa-faixa.

4) Filtro rejeita-faixa: Um filtro rejeita-faixa, é um tipo de filtro utilizado para atenuar ou eliminar uma faixa específica de frequências, enquanto permite a passagem das frequências fora dessa faixa, ou seja, seu funcionamento é o contrário de um filtro passa-faixa.

Os parâmetros para esse filtro (Fig 8) foram os seguintes:

- F_s: 8kHz
- f_{pass1}: 400Hz
- f_{stop1}: 900Hz
- f_{stop2}: 1100Hz
- f_{pass2}: 1600Hz
- A_{pass1}: 1dB
- A_{stop}: 40dB
- A_{pass2}: 1dB
- N: 36
- M: 37

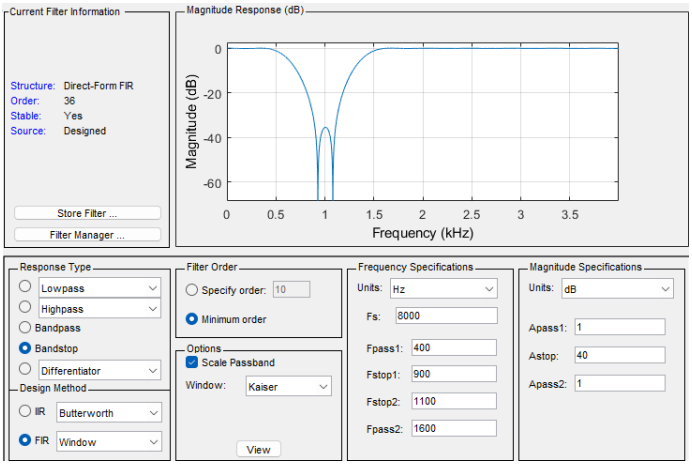

Figura 8. Filtro FIR rejeita-faixa.

F. Filtragem utilizando o dispositivo embarcado

Após obter os coeficientes de cada filtro, o processo de filtragem de áudio na placa foi simples de implementar. Os coeficientes foram diretamente incorporados como variáveis no código e, em seguida, aplicou-se uma convolução no domínio do tempo entre o sinal de entrada e os coeficientes correspondentes aos filtros. Essa abordagem é ilustrada no diagrama apresentado na Figura 1.

Essa convolução permite combinar e processar adequadamente o sinal de entrada com as características de filtragem específicas de cada filtro. Dessa forma, o sinal resultante após a convolução representa a saída filtrada correspondente ao tipo de filtro selecionado.

G. Implementação

Nesta subseção, serão apresentados os detalhes da implementação realizada na placa. Para isso, foi feita uma modificação em uma das demonstrações disponibilizadas pelo fabricante. A demo escolhida foi a *"SASK Record Play Demo With Intro"*, na qual foram utilizadas as funções de leitura de áudio do dispositivo de entrada (*input*) e de escrita de áudio no dispositivo de saída (*output*).

A leitura do sinal de áudio de entrada é realizada por meio de dois vetores, nos quais uma *flag* indica em qual deles os resultados de saída do conversor analógico-digital (ADC) da placa estão sendo armazenados. O tamanho desses vetores é definido durante a inicialização do código, e neste projeto foi utilizado o valor de 128 amostras. Após a leitura das 128 amostras, a *flag* é invertida para indicar a escrita no outro vetor. Em seguida, é feita uma cópia desse vetor para evitar sobrescrita e essa cópia é utilizada na operação de convolução.

Após a operação de convolução, obtém-se um vetor resultado cujo tamanho é igual à soma do tamanho do vetor de entrada ($SAMPLE\_SIZE$) e da quantidade de coeficientes do filtro ($FILTER\_SIZE$), menos um. Essa soma define o tamanho do vetor resultado da convolução ($CONVOLUTION\_SIZE = SAMPLE\_SIZE + FILTER\_SIZE - 1$). Portanto, o tamanho do vetor resultado varia de acordo com a quantidade de coeficientes presentes no filtro.

Uma vez obtido o vetor resultado da convolução, ele é passado para a função de escrita da placa, que aplica a modulação por largura de pulso (PWM - *Pulse Width Modulation*) do sinal digital para criar uma saída analógica e, dessa forma, a modulação permite que o sinal seja reproduzido através de um alto-falante ou dispositivo de saída de áudio adequado.

Devido à implementação dos quatro filtros, cada um com um número diferente de coeficientes, foi estabelecido que o tamanho do vetor onde o resultado é copiado seja igual ao maior valor entre os tamanhos resultantes da convolução.

Conforme mencionado anteriormente, a filtragem é realizada por meio de uma operação de convolução, para a qual foi desenvolvido um código em linguagem *C*. Esse código itera sobre o vetor resultado da convolução, adicionando a cada posição as amostras multiplicadas pelo coeficiente do filtro correspondente, o que modifica a amostra para essa posição específica.

Quanto aos filtros, a fim de atender aos requisitos do projeto, foi implementado na placa um sistema de controle para selecionar qual filtro será utilizado, utilizando os botões presentes na placa. Para isso, foi criada uma variável de controle que determina os parâmetros a serem utilizados na operação de convolução. Após o envio do resultado da convolução para a saída de áudio, é verificado se algum botão foi pressionado. Caso seja identificado um botão pressionado, o sistema avança para o próximo filtro, utilizando um código que aciona os *LEDs* correspondentes na placa para indicar o funcionamento do filtro selecionado, conforme descrito na Tabela I.

Tabela I
IDENTIFICAÇÃO DOS FILTROS PELOS *LEDs*.

Filtro	LED Âmbar	LED Verde
Passa-baixa	Desligado	Desligado
Passa-alta	Desligado	Ligado
Passa-faixa	Ligado	Desligado
Rejeita-faixa	Ligado	Ligado

O principal desafio enfrentado neste trabalho foi a conversão dos filtros gerados pela ferramenta *FDAtool* do *MatLab* em valores utilizáveis no DSP (*Digital Signal Processor*). Inicialmente, foram feitos testes multiplicando os coeficientes do filtro por 10.000 e arredondá-los para inteiros, e depois, após a convolução na placa, realizar uma divisão para normalizar os valores. No entanto, essa abordagem não obteve os resultados esperados.

Posteriormente, foram feitos novos testes, porém, com a conversão dos filtros por meio de multiplicação binária, isto é, os coeficientes foram submetidos a um deslocamento de *bit* para a esquerda (*left shift*) e convertidos para inteiros. Após a convolução na placa, esses valores seriam normalizados por meio de uma divisão binária (*right shift*). No entanto, essa abordagem também não produziu os resultados desejados.

Por fim, a solução adotada foi permitir que o próprio DSP fosse responsável pela conversão. Os filtros implementados foram transferidos para a placa com seus valores mantidos como ponto flutuante (*float*). Após a multiplicação com a amostra, o resultado é convertido para um valor inteiro antes de ser armazenado no vetor resultado em cada posição da convolução, através da operação de *casting*.

Desta forma, o algorítimo implementado consiste no seguinte:

1) Obter amostras, copiando do vetor onde o ADC escreve os resultados;
2) Realizar convolução com o filtro selecionado;
3) Enviar amostras para serem reproduzidas;
4) Verificar botão para troca de filtro.

III. DISCUSSÕES E RESULTADOS

Para visualizar os resultados obtidos no funcionamento dos quatro filtros implementados foram feitas gravações de cada uma das saídas de áudio do dispositivo com os respectivos sinais filtrados, possibilitando a plotagem do espectro de cada um deles, no qual foi possível visualizar e comprovar os seus funcionamentos. A Fig. 4 mostra o sinal de entrada utilizado para visualizar de forma mais proveitosa o funcionamento dos filtros passa-baixa e passa-alta, enquanto que a Fig. 9 foi o sinal de entrada para os filtros passa-faixa e rejeita-faixa. Alguns picos de magnitude no espectro das filtragens se dá por alguns problemas enfrentados, como de gravação e ruído intrínseco da caixa de som utilizada no projeto.

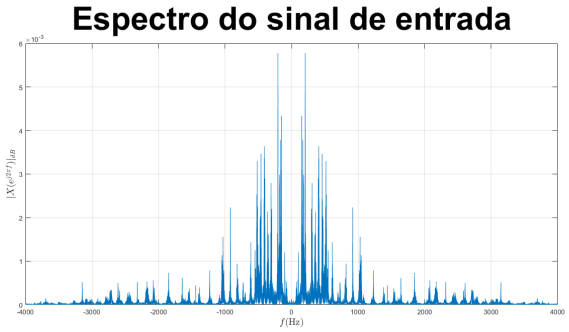

Figura 9. Sinal de entrada para filtros passa-faixa e rejeita-faixa.

Na Fig. 10, em comparação com a Fig. 4 é possível observar uma queda de amplitude em cerca de 1kHz até $2,6$kHz, e a partir desse valor não há mais presença de informações do sinal, como as componentes de 4kHz, conforme especificado no filtro passa-baixa projetado.

Já na Fig. 11, diferentemente da Fig. 10, tem-se a presença das altas frequências do sinal, como também valores bem atenuados por volta de 0Hz e a presença dos valores em 4kHz, especificado por um filtro passa-alta.

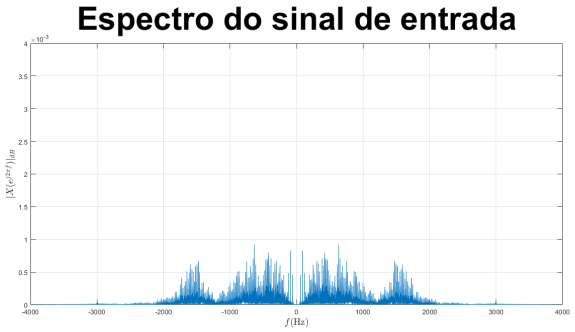

Figura 10. Espectro do filtro passa-baixa.

Figura 11. Espectro do filtro passa-alta.

Enquanto que na Fig. 12, em comparativo com a Fig. 9, o espectro observado diz respeito à um filtro passa-faixa com a presença de sinal entre as frequências de 500Hz à cerca de 2kHz.

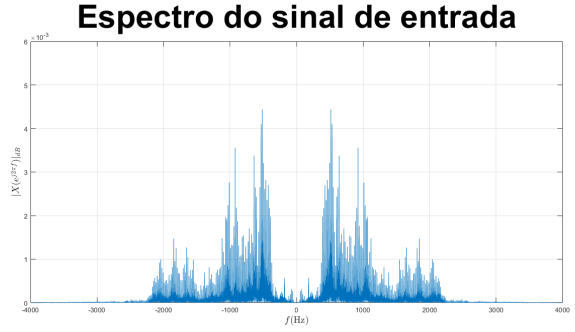

Figura 12. Espectro do filtro passa-faixa.

A Fig. 13 mostra o oposto de filtro passa-faixa, o filtro rejeita-faixa, atenuando valores próximos a 1kHz e mantendo os demais valores.

Como dito anteriormente, durante o processo de desenvolvimento do projeto, várias abordagens foram feitas para testar o funcionamento do código, e, como nenhuma delas foi possível alcançar o resultado esperado e mais próximo do obtido via *Matlab*, *software* no qual foram feitos e testados os filtros projetados. Assim, tem-se na saída das caixas um sinal filtrado,

Figura 13. Espectro do filtro rejeita-faixa.

mas com uma reprodução do áudio "robotizado", picotada e pouco audível.

IV. Conclusões

Apesar das expectativas iniciais, a implementação dos filtros digitais para eliminação de ruídos em um dispositivo embarcado programável em tempo real não proporcionou os resultados desejados. Durante o processo, enfrentou-se desafios devido à falta de documentação e exemplos disponíveis para auxiliar a implementação. Essa escassez de recursos tornou difícil a resolução de problemas e a obtenção de resultados satisfatórios.

Essa experiência permitiu compreender melhor as complexidades envolvidas na implementação de filtros em dispositivos embarcados e ressaltou a importância de considerar as limitações de armazenamento e processamento ao projetar soluções eficazes. Esses desafios são inerentes a ambientes de recursos limitados.

Para futuras melhorias, recomenda-se uma análise da aritmética da placa utilizada, a fim de investigar a causa dos problemas encontrados durante a implementação. Compreender o funcionamento interno da placa pode fornecer *insights* sobre possíveis limitações ou incompatibilidades que afetaram o desempenho dos filtros digitais.

Além disso, sugere-se a utilização de um osciloscópio para observar o espectro dos sinais de saída filtrados. Essa análise permitiria uma visualização mais detalhada dos resultados obtidos e auxiliaria na identificação de eventuais anomalias ou distorções que possam ter ocorrido durante o processo de filtragem.

Referências

[1] OPPENHEIM, Alan V.; SCHAFER, Ronald W. "Processamento em tempo discreto de sinais". Tradução Daniel Vieira. 3ª ed.-São Paulo: Pearson Education do Brasil, 2012.

[2] F. C. Comparsi d. C. (2001). *Processamento Digital de Sinais* [Online]. Available: https://www.fccdecastro.com.br/pdf/DSP%20A7.pdf. (accessed Mai 03, 2023).

[3] Microship (2018). *MPLAB STARTER KIT FOR DSPIC DSCS* [Online]. Available: https://www.microchip.com/en-us/development-tool/DM330011. (accessed Jun 11, 2023).