UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

FERNANDO TONDIN BORGES

Filtro Notch digital para 60 Hz : projeto, simulação e montagem

Guaratinguetá
2018

Fernando Tondin Borges

Filtro Notch digital para 60 Hz : projeto, simulação e montagem

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica .

Orientador: Prof^{o} Dr. Samuel Euzedice de Lucena

Guaratinguetá
2018

Borges, Fernando Tondin
B732f Filtro Notch digital para 60 Hz: projeto, simulação e montagem / Fernando Tondin Borges – Guaratinguetá, 2018.
35 f : il.
Bibliografia: f. 31

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2018.
Orientador: Prof. Dr. Samuel Euzédice de Lucena

1. Filtros elétricos digitais. 2. Sistemas eletrônicos. 3. Métodos de simulação. I.Título.

CDU 621.372.54

Luciana Máximo
Bibliotecária CRB-8/3595

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

FERNANDO TONDIN BORGES

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUANDO EM ENGENHARIA ELÉTRICA "**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Profª Dra. PALOMA MARIA SILVA ROCHA RIZOL
Coordenador

BANCA EXAMINADORA:

Profº Dr. Samuel Euzédice de Lucena
Orientador/UNESP-FEG

Profº Dr. Daniel Julien Barros da Silva Sampaio
UNESP-FEG

Profº Dr. Fernando Ribeiro Filadelfo
Membro Externo

Dezembro , 2018

DADOS CURRICULARES

FERNANDO TONDIN BORGES

NASCIMENTO 19/08/1995 - Santos / SP

FILIAÇÃO Elmar de Paiva Borges
Izilda Fátima Aparecida Tondin de Paiva Borges

Dedico este trabalho aos meus pais, que sempre me incentivaram a estudar e muito me apoiaram durante todo o meu curso de engenharia.

AGRADECIMENTOS

Agradeço a todos os meus professores, que me transmitiram muitos conhecimentos durante este curso. E à minha família, que colaborou com a tranquilidade necessária para a elaboração deste trabalho.

RESUMO

Este Trabalho de Graduação tem como objetivo o projeto, simulação e montagem de um filtro Notch de 60 Hz, para eliminar sinais induzidos pela rede que possam poluir a saída de qualquer dispositivo eletrônico e serão apresentados os passos utilizados para projetar, simular e montar esse filtro, bem como uma análise dos resultados obtidos e também algumas formas alternativas para sua montagem.

PALAVRAS-CHAVE: Filtro notch. Filtro digital. 60Hz.

ABSTRACT

This graduation work' s objective is the project, simulation and mounting a notch filter for 60Hz, in order to eliminate grid inducted signals that may pollutee the output of any electronic devices and then will be presented the steps to project, simulate and assemble such filter, followed by an analysis of the results and also some alternative methods to assemble it.

KEYWORDS: Notch filter. Digital filter. 60Hz.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Um filtro, no sentido mais comum da palavra, é algo utilizado para separar partículas de um meio líquido ou gasoso, por exemplo: separar areia da água, sendo assim utilizado para limpar ou filtrar a água.

No contexto da engenharia elétrica, sabe-se que qualquer sinal periódico, independente de seu formato, pode ser descrito através de uma série de Fourier, ou seja, pode ser descrito como um somatório de senos e cossenos e um valor constante, esses senos e cossenos cujas frequências são diferentes da fundamental são chamados de componentes harmônicas. Mesmo quando um sinal não é periódico, ele pode possuir componentes em outras frequências, desde que seu valor não seja constante.

Um dispositivo capaz de eliminar certas componentes indesejáveis de um sinal de entrada é chamado de filtro, pois, assim como um filtro de água, ele limpa o sinal de entrada e o disponibiliza em seu terminal de saída. Seus usos variam desde remover ruídos, que normalmente possuem alta frequência, até a separação de sinais de alta, média e baixa frequência, para serem tratados individualmente e depois somados, como ocorre em equalizadores de áudio.

Uma forma de classificar os tipos de filtros é pelo seu comportamento para diferentes frequências. Existem os passa baixas, que removem todas as componentes com frequência superior a uma frequência de corte, passa alta, que elimina as componentes de frequência inferior, os passa faixa, que elimina as componentes com frequência acima ou abaixo de uma frequência de corte superior e inferior, respectivamente, existe o rejeita faixa, que elimina todas as componentes com frequência entre duas frequências de corte, uma inferior e outra superior, existe também o filtro passa tudo, que não elimina nenhuma componente, mas é utilizado para mudar a fase do sinal de entrada. Por último existe o filtro notch, que elimina somente a componente com frequência igual à frequência de corte, seu funcionamento seria semelhante a um filtro rejeita faixa em que as frequências de corte superior e inferior são iguais.

Os filtros também podem ser classificados como filtros passivos, que não possuem alimentação externa e filtros ativos, que possuem alimentação externa, o último normalmente feito utilizando um amplificador operacional. Existem também filtros que utilizam capacitores chaveados, esses capacitores substituem a resistência em um filtro RC com amplificador operacional, aumentando a precisão dos filtros em circuitos integrados.

Um filtro passivo normalmente é um indutor ou capacitor em série ou paralelo com a carga, uma vez que para sinais de alta frequência um indutor é equivalente a um circuito aberto e um capacitor é equivalente a um curto circuito, esse comportamento é invertido em baixa frequência, ou seja, o capacitor seria um circuito aberto e o indutor um curto circuito. Para eliminar uma frequência é necessário um circuito aberto em série com a carga ou um curto circuito em paralelo com a carga, desta forma um indutor em série ou um capacitor em paralelo com a carga fazem a função de um filtro passa baixa, porém, é necessário lembrar que qualquer componente em série com a carga causará uma queda de tensão na carga, por menor que seja.

Filtros ativos utilizam o mesmo princípio dos filtros passivos, porém recebem esse nome porque dependem de uma fonte de alimentação externa, pois utilizam um amplificador operacional para obter um ganho na saída. Esses filtros são mais sofisticados que os filtros passivos, justamente devido ao

amplificador operacional.

Os filtros citados anteriormente são chamados de filtros analógicos, pois suas respostas são no domínio do tempo contínuo, mas existem também filtros digitais, que utilizam amostras do sinal de entrada para determinar a saída, esses filtros não possuem elementos reativos, como capacitores ou indutores, pois eles são microprocessadores que executam uma equação que utiliza o valor de n amostras anteriores para determinar a saída. Essas amostras podem ser somente da entrada, nesse caso seria um filtro FIR, ou podem ser n amostras da entrada e n amostras da saída, nesse caso tem-se um filtro IIR.

Linguagem C é uma linguagem de programação de uso genérico padronizada pela norma ISO 9899 e é também comumente usada como porta de entrada para cursos de programação, bem como base para várias outras linguagens, tanto de programação de software como para hardware, alguns exemplos são C++, que é uma linguagem de programação orientada a objetos, vários microprocessadores podem ser programados utilizando uma variação da linguagem C e até mesmo a linguagem utilizada pelas placas de arduino são baseadas em C.

Como um filtro digital é implementado com microprocessadores, a única semelhança entre ele e um filtro analógico é o seu comportamento, pois filtros digitais usam coeficientes para caracterizar suas frequências de cortes e são esses coeficientes que ocupam metade do espaço da memória utilizada pelo filtro, sendo que o filtro ideal é aquele com a melhor resposta e que utiliza menos memória, uma vez que esta, dependendo de onde for implementado o filtro, pode ser o fator limitante para o filtro.

Dito isso, a quantização desses coeficientes e, por consequência, das amostras pode ser feita utilizando números inteiros ou números reais, cada um com suas vantagens e desvantagens. Um número inteiro é representado como int em linguagem C, que pode ser usada para programar microprocessadores, pode ocupar dois ou quatro bytes e, portanto pode possuir valores com módulo até 32.768 ou 2.147.483.647. Já um número real pode ser representado como float, um número de ponto flutuante, que ocupa quatro bytes de memória, mas diferente do int, pode assumir valores até dez elevado à trigésima oitava potência e possui uma precisão de 6 casas decimais.

Como as linguagens de programação possuem princípios equivalentes, tais como: vetores; funções condicionais que realizam um determinado conjunto de operações, dependendo se uma condição dada é verdadeira ou não; e funções de laço que executam um determinado conjunto de operações enquanto uma condição fornecida é verdadeira. Pode-se testar o programa de um filtro digital utilizando o computador, simulando as amostras que o filtro teria, rodando o equacionamento e salvando o resultado obtido em um arquivo para ser plotado num gráfico, dessa forma é possível analisar o comportamento de um filtro sem ter que montá-lo e, caso esteja diferente do esperado, sabe-se que o problema se encontra na programação e não na montagem do circuito.

2 DESENVOLVIMENTO

Para coleta de dados na montagem do circuito é necessário Filtros para 60 Hz, uma vez que a rede elétrica fornece energia utilizando um valor de tensão que varia de acordo com uma senóide de 60 Hz, esse sinal pode gerar ruídos através de interferência eletromagnética. Aparelhos que utilizam sinais elétricos em que a frequência de trabalho inclua 60 Hz podem ter seu desempenho comprometido por causa deste ruído, uma vez que ele irá mascarar a saída. Esse ruído é inadmissível em equipamentos utilizados na medicina, uma vez que ele pode dificultar um diagnóstico ou até mesmo provocar um diagnóstico errado.

O filtro notch possui o mesmo comportamento que um filtro rejeita faixa, porém com uma única frequência de corte, ou seja, ele remove somente as componentes com uma determinada frequência de sua entrada. Esse comportamento o torna ideal para a tarefa de eliminar ruídos causados pela rede, uma vez que esses ruídos terão a mesma frequência do sinal que os gerou, ou seja, a mesma frequência da rede, que é 60 Hz. Caso não seja possível utilizar um filtro notch, um filtro rejeita faixa, com frequências de corte próximas de 60 Hz, é um substituto razoável.

Para a montagem desse filtro, podem ser utilizados filtros analógicos, compostos por amplificadores operacionais e elementos reativos, como indutores e capacitores. Também é possível utilizar filtros digitais, que são microprocessadores que utilizam uma equação discreta para remover o ruído das amostras. Uma das vantagens dos filtros digitais em relação aos filtros analógicos é que, por serem compostos por semicondutores programados para aplicar uma função de transferência, o comportamento do filtro não será alterado no decorrer do tempo, enquanto que, nos filtros analógicos, qualquer variação em seus componentes causa uma mudança na resposta em frequência.

Uma vantagem dos filtros digitais inclui a facilidade de alterar parâmetros do filtro, uma vez que para isso é necessário somente a alteração de uma seção do programa armazenado, em vez de trocar componentes físicos do circuito, como é o caso dos filtros analógicos. Outra vantagem dos filtros digitais é a precisão, que é dependente somente do erro de arredondamento do programa, que pode ser tão pequeno quanto necessário.

2.1 TRANSFORMADA Z

A transformada Z transforma uma sequência de valores em uma operação aritmética, assim como a transformada de Laplace transforma integrais e derivadas em multiplicações e divisões.

Sendo uma sequência de valores (x(0), x(1), x(2), x(3), x(4)..., x(k)), a chamada transformada Z desta sequência será:

$$X(z) = x(0) + x(1)z^{-1} + x(2)z^{-2} + x(3)z^{-3} + x(4)z^{-4} + ... + x(k)z^{-k} \tag{1}$$

Que pode ser reduzida para:

$$X(z) = \sum_{n=0}^{\infty} x(nT)z^{-n} \tag{2}$$

Ou seja, dada qualquer sequência de números como, por exemplo, $(1, \frac{1}{2}, \frac{1}{4})$, a sua transformada Z pode ser escrita da seguinte forma:

$$H(z) = 1 + \frac{1}{2}z^{-1} + \frac{1}{4}z^{-2} \tag{3}$$

A transformada Z é importante em filtros digitais, pois ela descreve como será o processo de amostragem do sistema, bem como possui um papel no domínio digital semelhante à transformada de Laplace em filtros analógicos, ou seja, a transformada Z também descreve a função de transferência do sistema.

Isso ocorre devido ao princípio de superposição da transformada Z, ou seja,

$$Z(f_k + g_k) = Z(f_k) + Z(g_k) \tag{4}$$

e

$$Z(af_k) = aZ(f_k) \tag{5}$$

Outra característica importante da transformada Z é a transformada do impulso unitário, chamado de delta de Dirac (δ), que é a sequência: δ=(1,0,0,0,0,0...)

Desta forma, percebe-se que a transformada Z do impulso unitário é um. Podemos também avaliar a transformada Z da função impulso com atraso, por exemplo, (0, 1, 0,0,0,0,0...), claramente percebe-se que a transformada Z desta nova sequência é z^{-1}, portanto pode-se dizer que z^{-1} é o atraso unitário e matematicamente o atraso de duas amostras seria $z^{-1}z^{-1}$, que é z^{-2}.

2.2 FUNÇÃO DE TRANSFERÊNCIA

A função de transferência de um sistema é a relação entre os sinais de entrada e saída. Para um amplificador a função de transferência é chamada de ganho, pois a saída deve possuir a mesma forma de onda da entrada, caso o amplificador esteja funcionando corretamente.

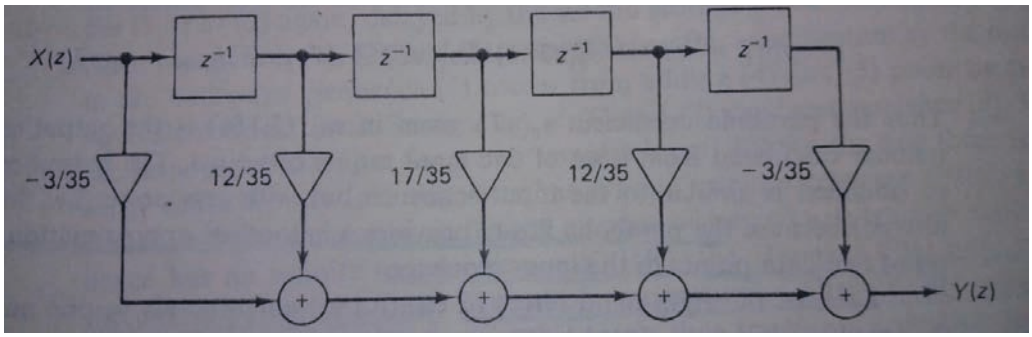
Obviamente essa característica da função de transferência se mantém, mesmo para um sistema digital que trabalha com amostras, porém assim como em sistemas analógicos em que a função de transferência é dada no domínio s, para simplificar a representação de saídas que são integrais ou derivadas da entrada, em sistemas digitais, utiliza-se a função de transferência no domínio z, da transformada z.

Portanto a função de transferência de um sistema digital será:

$$H(z) = \frac{Y(z)}{X(z)} \tag{6}$$

Chamamos de zeros de uma função de transferência, os valores de z em que essa mesma função possui valor zero. Já os polos da função de transferência são os valores de z em que a função possui valor tendendo ao infinito. Esses conceitos são importantes, pois para calcular um filtro digital simples, é necessário posicionar os zeros da função de transferência na posição das frequências de cortes desejadas.

Figura 1 – Diagrama de Blocos



fonte: Tompkins W. J.; Webster, J. G. (1981)

2.2.1 Pólos e zeros do plano Z

Matematicamente, a transformada z é baseada na definição $z = e^{sT}$

O s na equação acima se refere ao mesmo s da transformada de Laplace, que pode ser substituído por $s = \sigma + j\omega$, portanto $s = \sigma + j\omega$

Por definição a amplitude de z é dada por $|z| = e^{\sigma T}$ e a fase de z é dada por $fase(z) = \omega T$

Utilizando $\sigma = 0$, que é o eixo do plano s que separa funções de transferências estáveis e instáveis, temos que

$$z = e^{j\omega T} = cos(\omega T) + jsen(\omega T) \tag{7}$$

A equação acima é a equação de um circulo unitário que delimita a posição dos polos para manter a função de transferência estável. Esse critério é análogo ao fato de que polos no semipleno positivo de s tornam o sistema instável. Isso ocorre, pois o plano z é um mapeamento matemático direto do plano s.

2.3 FILTROS DIGITAIS

Microcomputadores são utilizados como filtros digitais em tempo real, pois para montar um filtro digital, são necessárias somente as operações de armazenamento, multiplicação por uma constante e adição, que são operações universais em microcomputadores e linguagens de programação em geral.

Existem dois tipos de filtros digitais: os filtros não recursivos, chamados também de FIR ou resposta finita ao impulso; e os filtros recursivos, também chamados de IIR ou resposta infinita ao impulso.

Filtros não recursivos possuem esse nome, pois sua saída é determinada somente pelo valor atual e valores anteriores da entrada, porém não são todos os valores anteriores que interferem na saída; o número exato de valores anteriores da entrada, usados em um filtro não recursivo, é determinado pelo chamado tap do filtro. Esses filtros também são chamados de FIR (finite impulse response), pois ao colocar um impulso na entrada, chegará um momento em que a saída retornará a zero, portanto a resposta ao impulso é finita.

Um diagrama de blocos genérico para este tipo de filtro é mostrado em Figura 1

Por não haver realimentação neste tipo de filtro, sua função de transferência não possui polos e, portanto, filtros FIR serão sempre estáveis, porém é necessário um filtro com mais taps para obter uma

atenuação semelhante à atenuação de um filtro recursivo, que ocupa menos espaço de armazenamento.

Filtros recursivos possuem esse nome, pois o valor disponível na saída é em função dos valores anteriores da saída, além dos valores atuais e anteriores da entrada. Esses filtros também são chamados de IIR (infinite impulse response), pois caso ocorra um impulso na entrada, a saída sempre possuirá um valor em resposta ao impulso, devido à realimentação. Para contornar esse problema, o equacionamento da saída possui coeficientes que multiplicam os valores anteriores da entrada e da saída, dessa forma apesar de que teoricamente a saída sempre será influenciada por todos os valores da entrada, chega um momento em que o efeito causado pelos valores anteriores da entrada, que estão presentes na saída, se torna desprezível. A ordem de um filtro IIR é determinada pela quantidade de valores anteriores da entrada e da saída, que estão presentes no equacionamento da saída.

Uma equação genérica da saída Y(z) de um filtro IIR de segunda ordem, que possui entrada X(z), está representada abaixo.

$$Y(z) = aX(z) + b_1X(z)z^{-1} + b_2X(z)z^{-2} + c_1Y(z)z^{-1} + c_2Y(z)z^{-2} \tag{8}$$

Sendo a, b1, b2, c1 e c2 os coeficientes da equação. Essa equação pode ser expandida infinitamente, pois:

$$Y(z)z^{-1} = (aX(z) + b_1X(z)z^{-1} + b_2X(z)z^{-2} + c_1Y(z)z^{-1} + c_2Y(z)z^{-2})z^{-1} \tag{9}$$

$$Y(z)z^{-1} = aX(z)z^{-1} + b_1X(z)z^{-2} + b_2X(z)z^{-3} + c_1Y(z)z^{-2} + c_2Y(z)z^{-3} \tag{10}$$

Através da equação acima se percebe que cada termo da saída pode ser expandido em mais uma equação de valores anteriores da entrada e da saída, infinitamente, deste fato surge o nome de resposta infinita ao impulso.

2.3.1 Funcionamento de Filtros Digitais

Filtros digitais possuem um comportamento semelhante aos filtros analógicos, porém a forma com que um desses filtros chega nesse comportamento é completamente diferente.

Em filtros analógicos as frequências de corte são determinadas pelos elementos reativos, ou seja, capacitores e indutores. Já em filtros digitais as frequências de corte são determinadas pelas posições dos zeros no círculo unitário.

Considerando que a frequência de amostragem do filtro seja o 2π radianos, um zero que fique em $\frac{\pi}{4}$ radiano significará que o filtro possui uma frequência de corte igual a um quarto da frequência de amostragem.

Portanto para projetar um filtro digital, basta modelar a função de transferência para que os polos e zeros estejam na posição desejada. Uma vez que qualquer posição que não seja z=1 ou z=-1 o valor de z será um número complexo, o pólo ou zero correspondente possuirá um par conjugado, que ficará na parte inferior do circulo unitário. No entanto, esses pólos e zeros representarão frequências de cortes com valor superior à metade da frequência de amostragem e, como o limite superior da frequência de um sinal que será usado em qualquer sistema digital é metade da frequência de amostragem, essas

frequências de corte não alterarão o comportamento do filtro, pois estarão acima do valor de frequência que será processado pelo sistema digital, se o filtro for projetado corretamente.

Para facilitar o projeto de um filtro digital, existe um software gratuito chamado winfilter, de autoria de Andrian Kundert, disponível no site www.winfilter.20m.com , que calcula um filtro digital tendo como dados as especificações fornecidas pelo usuário, como: frequências de corte, tipo do filtro, ordem do filtro, frequência de amostragem e outros parâmetros. O software também gera uma imagem com o diagrama de polos e zeros, as respostas de fase e em frequência, a resposta ao impulso e ao degrau e o atraso de grupo do filtro calculado. Esse software também gera um código em C para esse filtro. A versão mais atual consegue gerar um código em VHDL para filtros FIR.

Tendo um software que calcula o filtro, é possível realocar o tempo que seria usado para esse cálculo, para a definição dos parâmetros do filtro, para que ele possa atender aos requisitos da aplicação desejada. Por exemplo: qual tipo de filtro será usado, se é melhor utilizar um FIR ou IIR, qual tipo de filtro será melhor, se Bessel, Chebychev ou Butterworth. Havendo mais tempo para essas escolhas, existem mais garantias de que o filtro será melhor otimizado.

2.3.2 Winfilter

A interface deste software é intuitiva e simples de usar, conforme mostrado abaixo. Para projetar um filtro, o primeiro passo é escolher se o filtro será realimentado ou não. O segundo passo é escolher o tipo do filtro, ou seja, se o filtro desejado é passa baixa, passa alta, passa faixa ou rejeita faixa. Em seguida escolhe-se o modelo do filtro, os modelos disponíveis são Butterworth, Chebychev, Bessel, retangular ou cosseno levantado, os dois últimos somente são implementados em filtros FIR. Os únicos dados restantes são as especificações do filtro, entre elas estão a taxa de amostragem, as frequências de corte e a ordem do filtro. O último elemento escolhido para calcular o filtro é a quantização dos coeficientes, que é o único fator que não influencia o comportamento do filtro, somente a utilização de memória pelo filtro. Após esses passos basta clicar em calcular filtro para que o programa realize todos os cálculos necessários. Caso o filtro seja instável o próprio software avisará.

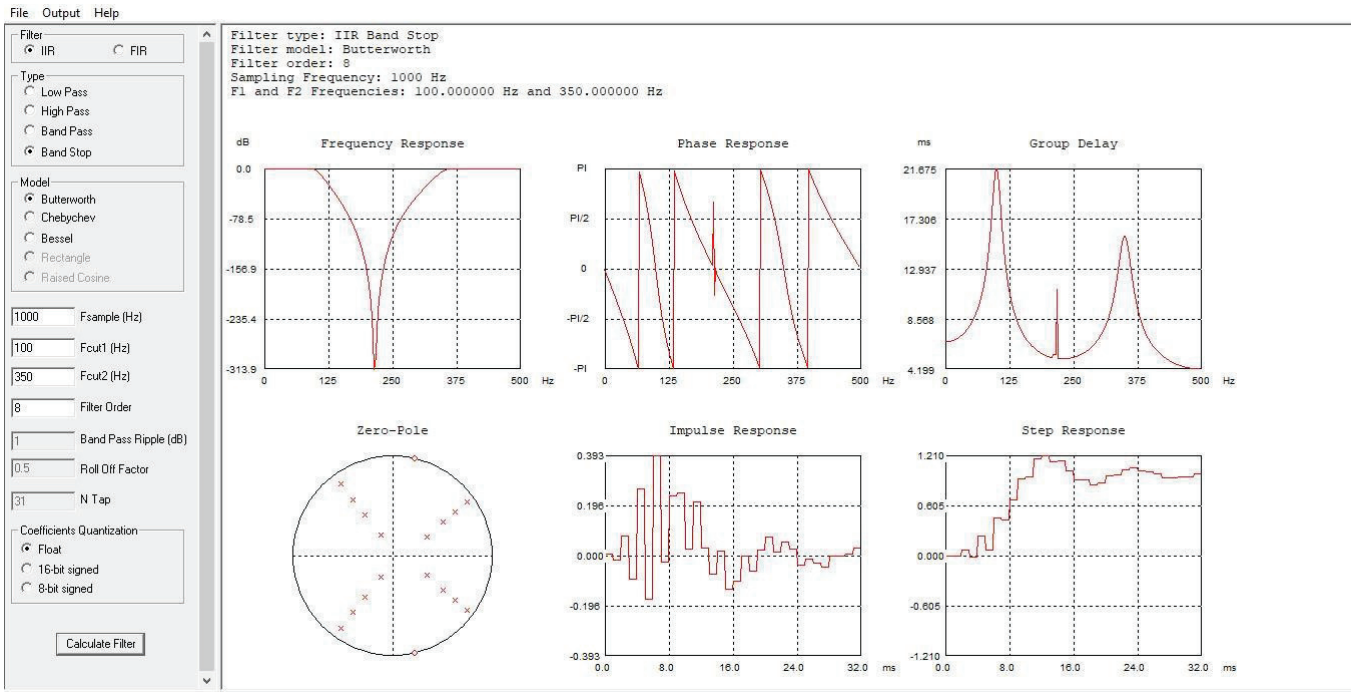
Tendo calculado o filtro, existe o menu de output que exporta um programa em C, VHDL, ou os gráficos em formato gif.

Analisado as opções do programa, foi decidido utilizar o winfilter para calcular um filtro IIR rejeita faixa, com frequências de corte iguais a 55 e 65 Hz, para que o 60 Hz esteja entre as duas frequências de corte e uma taxa de amostragem igual a 1000 Hz, para que o filtro receba uma amostra a cada 1 ms, pois números mais simples implicam em uma programação mais simples e precisa.

Para comparação foram calculados filtros Bessel, Butteworth e Chebychev de ordem 2, 3 e 4. O que diferencia esses filtros são os coeficientes de cada filtro, como é mostrado em Figura 3,Figura 4 e Figura 5, respectivamente.

Os programas gerados pelos winfilter possuem um cabeçalho com algumas informações importantes, como o contato do criador do software, as especificações e os polos e zeros do filtro. Esses pólos e zeros dos filtros calculados são mostrados em Figura 6,Figura 7 e Figura 8.

Figura 2 – Interface do Winfilter



fonte: Produção do próprio autor

Figura 3 – Coeficientes de um filtro Bessel

```
#define NCoef 4
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.89982121639333945000,
        -3.34818256000143500000,
        4.91424071377380580000,
        -3.34818256000143500000,
        0.89982121639333945000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -3.63833436332478930000,
        5.22213020800852590000,
        -3.48014936946933150000,
        0.91497583477697930000
    };
```

```
#define NCoef 6
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.88435681612444139000,
        -4.93596063521563800000,
        11.83628207674771300000,
        -15.56695425008728000000,
        11.83628207674771300000,
        -4.93596063521563800000,
        0.88435681612444139000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -5.46453532681942190000,
        12.83024204101259100000,
        -16.52352610386716600000,
        12.30376062619022500000,
        -5.02528342025197720000,
        0.88189313056979091000
    };
```

```
#define NCoef 8
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.87836332020001207000,
        -6.53667794548438330000,
        21.75540293037059300000,
        -42.23577939730523700000,
        52.27771509846542100000,
        -42.23577939730523700000,
        21.75540293037059300000,
        -6.53667794548438330000,
        0.87836332020001207000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -7.28916309716733850000,
        23.76308694449119900000,
        -45.19156417713475800000,
        54.79779795865511700000,
        -43.37366199068716100000,
        21.88974506656207100000,
        -6.44444756512906650000,
        0.84855599924387892000
    };
```

fonte: Produção do próprio autor.

Figura 4 – Coeficientes de um filtro Butterworth

```
#define NCoef 4
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.98744380887960537000,
        -3.67422114486670330000,
        5.39277856506869120000,
        -3.67422114486670330000,
        0.98744380887960537000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -3.66131366045129350000,
        5.28931921562450570000,
        -3.54851541727678080000,
        0.93937113609272649000
    };
```

```
#define NCoef 6
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.93652090408901201000,
        -5.22710995421270570000,
        12.53444920586051300000,
        -16.48517634781546100000,
        12.53444920586051300000,
        -5.22710995421270570000,
        0.93652090408901201000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -5.43795016437390280000,
        12.70298470001579400000,
        -16.27278238389892100000,
        12.04964964956488900000,
        -4.89275086089054500000,
        0.85336130203737992000
    };
```

```
#define NCoef 8
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.89966064737190243000,
        -6.69517018397011650000,
        22.28289755965326800000,
        -43.25985359473344300000,
        53.54527212941858500000,
        -43.25985359473344300000,
        22.28289755965326800000,
        -6.69517018397011650000,
        0.89966064737190243000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -7.27900354491116630000,
        23.70154807392541200000,
        -45.03015555594424100000,
        54.56071104433449600000,
        -43.16360315037061500000,
        21.77793720891499900000,
        -6.41150107399267280000,
        0.84444172205922730000
    };
```

fonte: Produção do próprio autor.

Figura 5 – Coeficientes de um filtro Chebychev

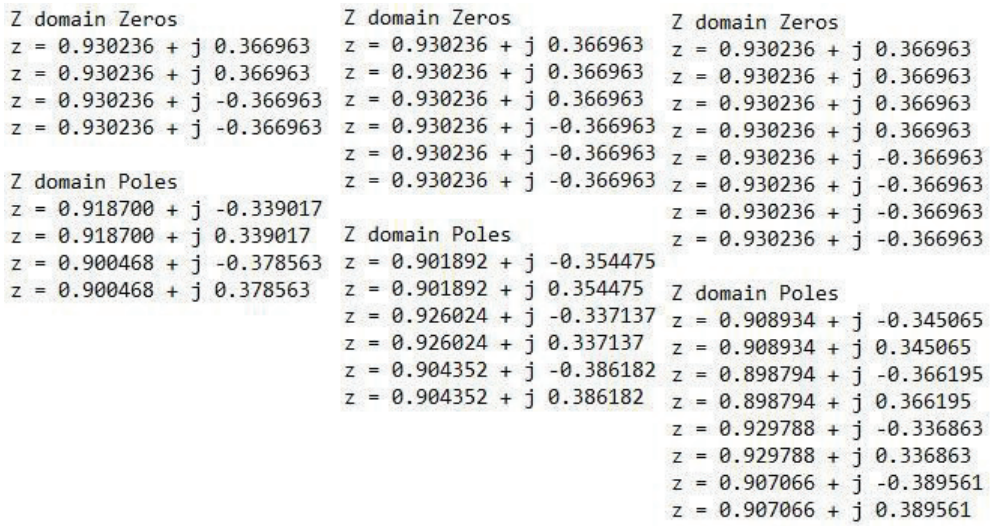
```
#define NCoef 4
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.89829341840551769000,
        -3.34249771229520580000,
        4.90589687064431330000,
        -3.34249771229520580000,
        0.89829341840551769000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -3.62150875260450710000,
        5.17270822659219040000,
        -3.42960600302133310000,
        0.89685259840643450000
    };
```

```
#define NCoef 6
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.88442469968145287000,
        -4.93633952138361560000,
        11.83719063414729400000,
        -15.56814917526574000000,
        11.83719063414729400000,
        -4.93633952138361560000,
        0.88442469968145287000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -5.43945358389172020000,
        12.71122792231176200000,
        -16.29134039529324000000,
        12.07104498977030500000,
        -4.90537220985098620000,
        0.85640860573618971000
    };
```

```
#define NCoef 8
float iir(float NewSample)
{
    float ACoef[NCoef+1] =
    {
        0.87909371229058098000,
        -6.54211343870248640000,
        21.77349336500288500000,
        -42.27090003417766400000,
        52.32118598203127400000,
        -42.27090003417766400000,
        21.77349336500288500000,
        -6.54211343870248640000,
        0.87909371229058098000
    };

    float BCoef[NCoef+1] =
    {
        1.00000000000000000000,
        -7.25741294851198140000,
        23.55471405504359700000,
        -44.59346539495167100000,
        53.82507282978843900000,
        -42.40572039881765700000,
        21.30026670650920900000,
        -6.24086370630433860000,
        0.81775184811252399000
    };
```

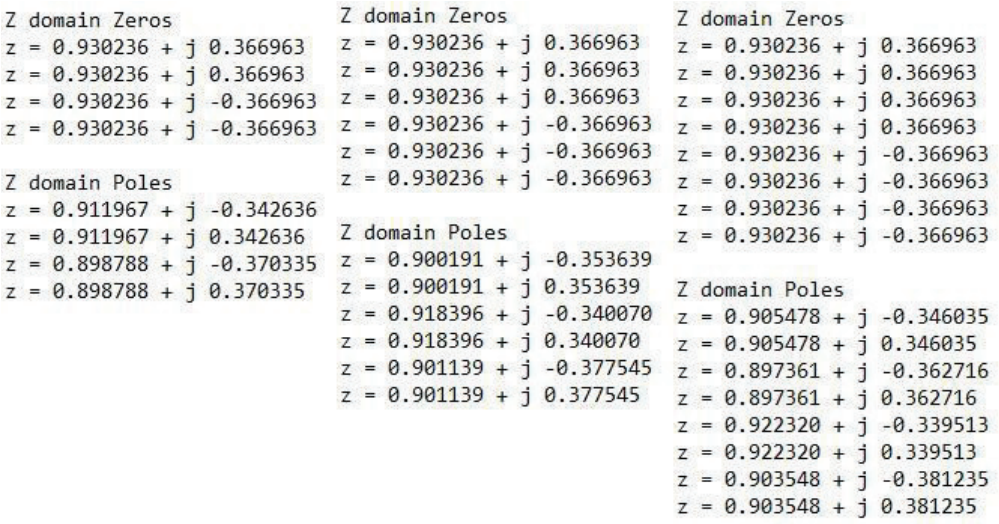
fonte: Produção do próprio autor.

Figura 6 – pólos e Zeros de um filtro Bessel



fonte: Produção do próprio autor.

Figura 7 – pólos e Zeros de um filtro Butterworth



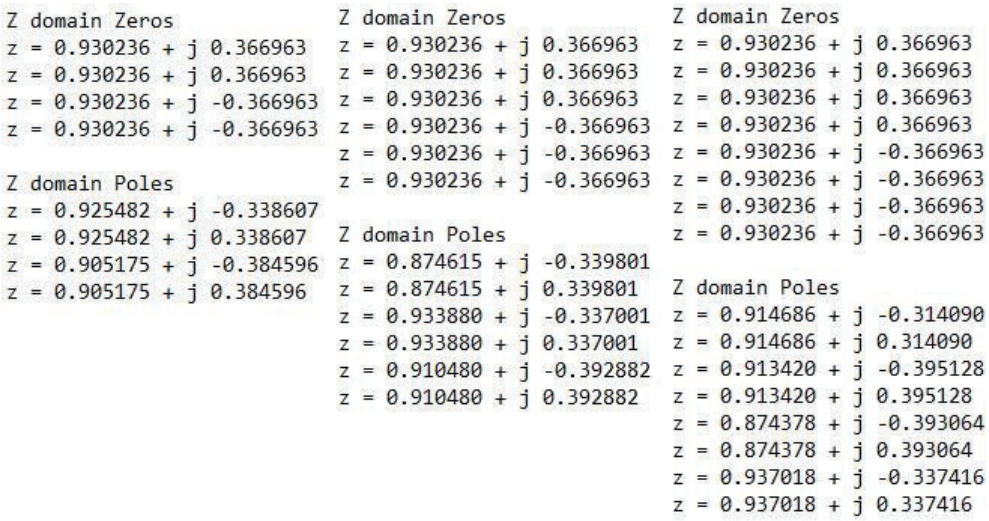
fonte: Produção do próprio autor.

2.4 SIMULAÇÃO

Uma vez que o filtro foi calculado usando um método desconhecido, as boas práticas de projeto e o bom senso nos obrigam a testar esse filtro. Uma vez que o filtro é uma listagem em C, um computador pode ser usado para testar o algoritmo, uma vez que todos os cálculos que serão realizados pelo filtro, um computador pessoal também consegue fazer.

Para isso a linguagem Python é uma ótima ferramenta, devido, principalmente, à sua facilidade para trabalhar com arquivos de texto. A biblioteca math possui a função sin(), que nos permite simular os valores do sinal senoidal que será a entrada do filtro. Esse valor passa pelo algoritmo no filtro e a saída gerada será semelhante àquela que seria gerada por um microprocessador, porém esta última será um sinal digital, em vez de bits na memória do computador. O código utilizado para testar o algoritmo está representado abaixo.

Figura 8 – pólos e Zeros de um filtro Chebychev



fonte: Produção do próprio autor.

Figura 9 – Criação das variáveis necessárias

```
import math

Acoef = []
Bcoef = []
Ncoef =  8
y=[]
x=[]

freq = 60
file1 = open("entrada"+str(freq)+"Hz.txt",'w')
file2 = open("saida"+str(freq)+"Hz.txt",'w')
```

fonte: Produção do próprio autor.

Figura 10 – Inicialização das listas com valores nulos

```
n=0
while n <= Ncoef:
    y.append(0)
    x.append(0)
    n = n+1
```

fonte: Produção do próprio autor.

A Figura 9 mostra o início do código, é aqui que os coeficientes do filtro são definidos, bem como são criadas duas listas vazias, que farão o papel de memória, pois armazenarão os mesmos valores que o filtro real irá armazenar. O campo freq define a frequência da senóide que será usada de entrada no filtro. Por último as duas linhas finais criarão dois arquivos onde serão armazenados todos os valores de entrada e saída para a frequência definida.

Este pedaço de código em Figura 10 preenche as listas com zeros. Isso é necessário para não haver problemas com índice inexistente quando os valores armazenados nas listas forem necessários.

Esta parte final do código, em Figura 11 calcula a fase do sinal de uma amostra usando a definição

Figura 11 – Algoritmo do filtro

```
amostra = 0
while amostra < 1000:
    angulo = (freq*2*math.pi*amostra/1000)
    entrada = math.sin(angulo)
    print("amostra"+str(amostra))
    n = Ncoef
    while n>0:
        x[n] = x[n-1]
        y[n] = y[n-1]
        n = n-1
    x[0] = entrada
    y[0] = Acoef[0] * x[0]
    n = 1
    while n <= Ncoef:
        y[0] = y[0] + Acoef[n] * x[n] - Bcoef[n] * y[n]
        n = n+1
    saida = y[0]
    amostra = amostra + 1
    file1.write(str(entrada)+"\n")
    file2.write(str(saida)+"\n")
file1.write("\n")
file2.write("\n")
file1.close()
file2.close()
```

fonte: Produção do próprio autor.

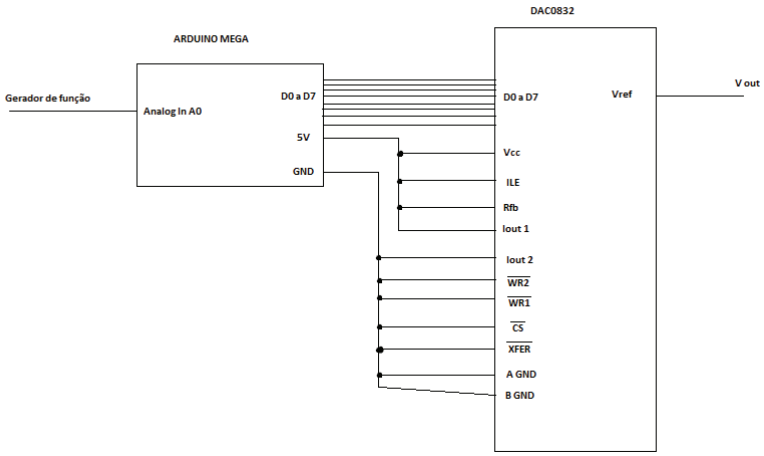
de que o ângulo de fase é igual a dois π, multiplicado pelo tempo, vezes a frequência. Neste caso o tempo é o número da amostra multiplicado por 1 ms. Com o valor da fase, o valor obtido na entrada é o valor do seno dessa fase. A função print logo abaixo mostra uma mensagem para informar em qual amostra o teste se encontra.

Após simular a entrada do filtro, o algoritmo do filtro pode ser testado. Esse algoritmo é descrito como primeiro, desloca-se os valores armazenados, indicando que o valor z da iteração passada é agora z^{-1}, tanto para entrada como para saída. Após isso o valor atual da entrada se torna o valor z para a entrada. Para determinar o valor da saída utiliza-se o equacionamento do filtro. Após a execução do algoritmo do filtro, o programa escreve nos arquivos que foram criados os valores da entrada e da saída. Ao fim da última amostra ser adicionada, esses arquivos serão salvos e fechados, permitindo a utilização deles por outros programas. O Excel é um excelente programa para visualizar as saídas e permite criar um gráfico mostrando os valores na entrada e na saída do filtro, simultaneamente, e também possui uma função para importar dados de arquivos de texto, como os dois que foram criados durante o teste.

2.5 MONTAGEM

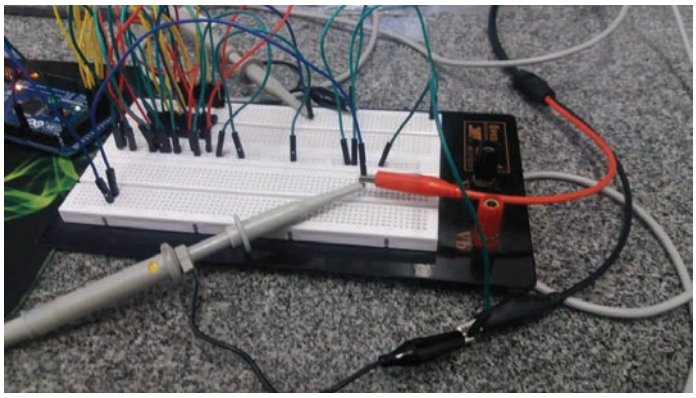
Para a montagem, foram utilizados um arduino MEGA 2560 e um conversor DA dac0832. O arduino foi responsável pela amostragem, realizada por seu conversor AD interno, bem como os cálculos, enquanto o conversor foi o responsável em transformar um numero de 8 bits em um valor analógico de tensão que representa a saída, sem nenhuma influência eletromagnética da rede, de um

Figura 12 – Circuito Montado



fonte:Produção do próprio autor.

Figura 13 – Montagem do Filtro



fonte:Produção do próprio autor.

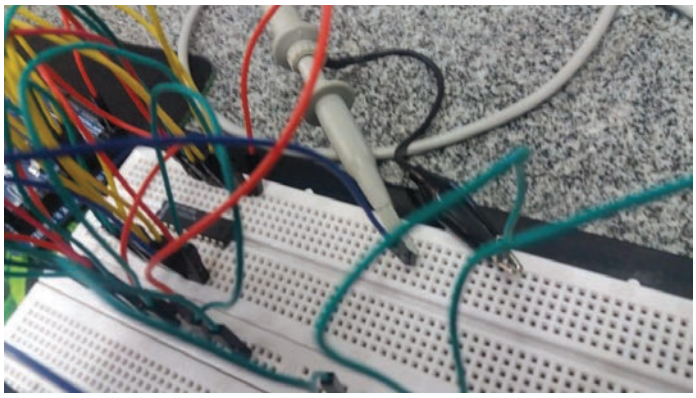
equipamento ao qual o filtro esteja ligado.

Como todos os componentes do filtro aceitam uma tensão de alimentação de 5 V, foi utilizado uma saída de 5 V do arduino para alimentar o conversor DA. Enquanto isso, o arduino foi alimentado pela porta USB de um computador, porém as placas de arduino podem ser alimentadas por uma fonte externa ou por uma fonte chaveada ligada na tomada. Na Figura 12 se encontra o esquemático do circuito e logo em seguida, em Figura 13, Figura 14 e Figura 15 é apresentada a montagem física.

Um programa arduino pode ser dividido em três partes, são elas: o cabeçalho, onde são declaradas as variáveis necessárias para o projeto; a função setup, onde o arduino configura suas entradas e saídas e a função loop, que nada mais é do que o laço infinito que executa a aplicação seja ela o piscar de um led ou até mesmo um filtro, porém também podem existir funções criadas pelo programador, ou seja, pedaços de código que serão executados várias vezes ou que, por motivo de uma maior facilidade de solução de problemas, estão fora da função loop, mas são chamadas dentro dela, ou seja, são pedaços de código externos à função loop, mas que são executados como se estivessem dentro dela.

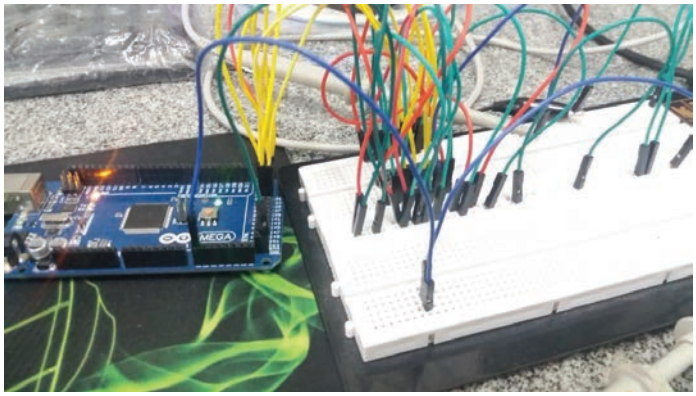
Devido à complexidade de um filtro, mesmo que um algoritmo já tenha sido simulado, ainda existem alguns passos necessários antes da implementação do filtro, são eles: o acerto da taxa de amostragem e a verificação do funcionamento do conversor DA, esses passos serão discutidos abaixo.

Figura 14 – Montagem do Filtro



fonte:Produção do próprio autor.

Figura 15 – Montagem do Filtro



fonte:Produção do próprio autor.

2.5.1 Taxa de Amostragem

Para ser obtido um código mais limpo, foi escolhido previamente uma taxa de amostragem de 1 kHz, ou seja, a cada 1 milissegundo a entrada do arduino será amostrada, para isso foi utilizada a função millis() do arduino, pois essa função retorna um integer no formato unsigned long com o número de milissegundos desde o começo da execução e, de acordo com a referência do arduino, esse valor só irá sofrer overflow, ou seja, voltar a zero, depois de aproximadamente cinquenta dias.

Para manter a taxa de amostragem no valor desejado, toda vez que o valor de millis() fosse alterado, significaria que a entrada deve ser amostrada, para tanto, toda vez que ocorre essa amostragem, o valor do millis() é salvo em uma variável e só haverá uma nova amostra, bem como um novo cálculo da saída, quando o valor da função for diferente do valor salvo e para ser atestado o tempo entre uma amostra e outra foi escrito o código do filtro e utilizou-se uma saída digital que é deixada em nível lógico alto no momento da amostra e é transformado em nível lógico baixo ao final dos cálculos, como é mostrado no código prsente na Figura 16. Em seguida , na Figura 17 é mostrado a forma de onda da saída digital, que apresentou um período de 1,025 milissegundo, que é, para todos os fins práticos, 1 milissegundo.

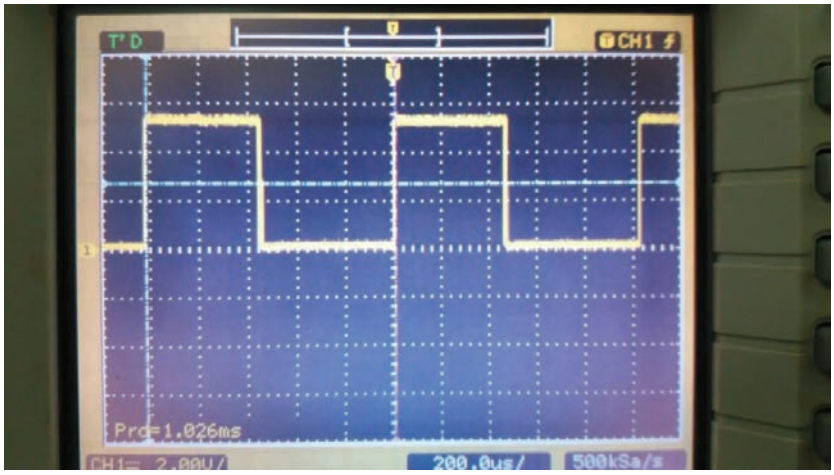
No pedaço de código na Figura 16 existem duas curiosidades, a primeira é a multiplicação realizada no valor da função analogRead() e a segunda é o laço for que verifica o valor de bitRead para executar

Figura 16 – Função Loop do arduino

```
void loop()
{
 if(millis() != tempo)
 {
    tempo = millis();
    digitalWrite(led,HIGH);

    amostra = analogRead(entrada)*256/1024;

    resultado = filtro(amostra);

    for(i = 0;i<=7;i++)
    {
      if(bitRead(resultado,i) == 1)
      {
        digitalWrite(saida[i],HIGH);
      }
      else
      {
        digitalWrite(saida[i],LOW);
      }
    }
    digitalWrite(led,LOW);
 }

}
```

fonte:Produção do próprio autor.

Figura 17 – Medição da Taxa de Amostragem



fonte:Produção do próprio autor.

Figura 18 – Função que calcula a rsposta do filtro

```
int filtro(int entrada)
{
    x[0] = entrada
    y[0]=ACoef[0]*x[0];

    for(n=1;n<=NCoef;n++)
      y[0] += ((ACoef[n]*x[n])-(BCoef[n]*y[n]));

    for(n=NCoef; n>0; n--)
    {
      x[n] = x[n-1];
      y[n] = y[n-1];
    }

    return int y[0];
}
```

fonte:Produção do próprio autor.

um digitalWrite, abaixo será discutido o motivo da existência desses códigos.

A multiplicação por 256 e logo em seguida divisão por 1024 é necessária, pois o conversor DA espera uma entrada de 8 bits, porém o arduino possui um conversor AD interno de 10 bits, portanto, para criar a compatibilidade entre esses dispositivos foi necessário utilizar esse método para converter os 10 bits do arduino em um valor de 8 bits, mas que mantenha o seu significado, dessa forma, garantiu-se que uma amostra de 5V, que seria convertida para 1023, que é o valor máximo de 10 bits, seria transformado em 255, que é o valor máximo de um número de 8 bits.

Em seguida, o laço foi necessário para que a variável integer resultado, que armazena o resultado do filtro, possa ter seu valor convertido em vários bits e esses bits possam ser distribuídos na saída do arduino. Para realizar isso se utilizou a função bitRead(x,n), que retorna o valor do enésimo bit de um número x, e dado o valor desse bit, uma das 8 saídas do arduino é colocada no respectivo nível lógico.

Nota-se na função loop acima que foi utilizada a função filtro para realizar os cálculos. Para criar uma função em arduino basta escrevê-la em qualquer parte do sketch, desde que ela tenha o tipo do valor a ser retornado, um nome e os parâmetros externos a serem passados, conforme mostrado na Figura 18.

Esta é a função responsável por pegar o valor da amostra e, com os coeficientes e amostras anteriores, bem como os valores anteriores da saída, calcular o valor da saída, após esses cálculos dois vetores com os valores de entrada e saída sofrem uma rotação, para permitir que na próxima iteração a nova saída seja calculada, usando o novo valor da entrada.

2.5.2 Conversor DA

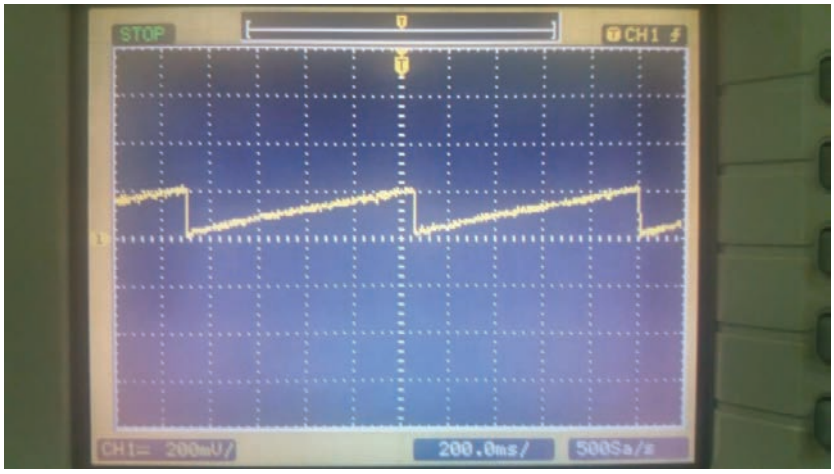
Para testar a saída do conversor o procedimento é mais simples, basta criar um código que gere uma rampa no arduino e colocar essa rampa no conversor, se o osciloscópio mostrar a forma de onda correta, o hardware está pronto para receber o filtro.

Figura 19 – Função que gera uma rampa no arduino

```
void loop()
{
  delay(1);
  resultado++;
  if(resultado == 256)
    resultado = 0;}
  Serial.print(resultado);
  Serial.print("\n");
  for(i = 0;i<=7;i++)
  {
    if(bitRead(resultado,i) == 1)
      digitalWrite(saida[i],HIGH);
    else
      digitalWrite(saida[i],LOW);
  }
}
```

fonte:Produção do próprio autor.

Figura 20 – Forma de onda na saída do Conversor DA



fonte:Produção do próprio autor.

Na Figura 19 se encontra o código utilizado para gerar uma rampa no arduino e, na Figura 20 é apresentada a saída de acordo com o osciloscópio.

Após isso se constatou que o hardware estava pronto e o único passo restante era carregar o filtro no arduino e verificar o seu funcionamento.

2.6 RESULTADOS

Conforme dito anteriormente, o filtro Butterworth apresenta a melhor resposta em frequência, uma vez que nos arredores das frequências de corte não há um ganho e ele ainda apresenta uma atenuação razoável. Também de acordo com as simulações, um filtro de quarta ordem seria estável, porém ao carregar o programa no arduino a saída do conversor apresentava um transitório e após isso caia para 0 V. Utilizando o monitor serial percebeu-se que o valor do resultado ficava crescendo em valores positivos e negativos alternados, ambos tendendo ao infinito, respectivamente positivo e negativo,

Figura 21 – Coeficientes de um filtro Butterworth de quarta ordem

```
const int NCoef = 8;
const float ACoef[9] = {
        0.87836332020001207000,
        -6.53667794548438330000,
        21.75540293037059300000,
        -42.23577939730523700000,
        52.27771509846542100000,
        -42.23577939730523700000,
        21.75540293037059300000,
        -6.53667794548438330000,
        0.87836332020001207000
    };
const float BCoef[9] = {
        1.00000000000000000000,
        -7.28916309716733850000,
        23.76308694449119900000,
        -45.19156417713475800000,
        54.79779795865511700000,
        -43.37366199068716100000,
        21.88974506656207100000,
        -6.44444756512906650000,
        0.84855599924387892000
    };
```

fonte:Produção do próprio autor.

portanto o filtro de quarta ordem era instável, mas ao trocar os parâmetros mostrados na Figura 21 para o de um filtro de terceira ordem, conforme mostrado na Figura 22, notou-se que a saída foi estabilizada.

Após corrigir a estabilidade do filtro foi possível verificar se o seu funcionamento estava correto ou não. Ao utilizar um gerador de função para simular uma entrada senoidal e variar sua frequência, notou-se uma queda da saída ao se aproximar da frequência de 60 Hz e após essa frequência a saída voltou ao seu valor normal, conforme mostrado em Figura 23, Figura 24 e Figura 25.

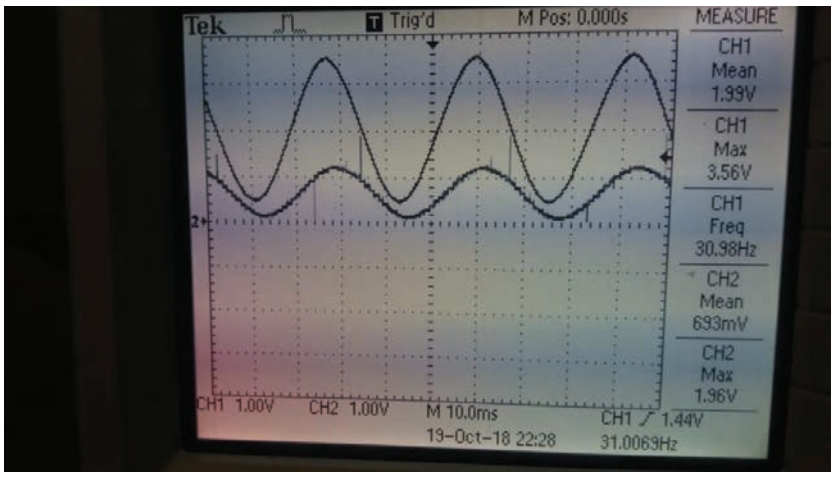
Ao analisar o funcionamento do filtro desde poucos hertz até mais ou menos cem hertz, pudemos coletar alguns dados e plotar a resposta em frequência apreesentada na Figura 26.

Figura 22 – Coeficientes de um filtro Butterworth de terceira ordem

```
const int NCoef = 6;
const float ACoef[7] = {
        0.88435681612444139000,
        -4.93596063521563800000,
        11.83628207674771300000,
        -15.56695425008728000000,
        11.83628207674771300000,
        -4.93596063521563800000,
        0.88435681612444139000
    };
const float BCoef[7] = {
        1.00000000000000000000,
        -5.46453532681942190000,
        12.83024204101259100000,
        -16.52352610386716600000,
        12.30376062619022500000,
        -5.02528342025197720000,
        0.88189313056979091000
    };
```

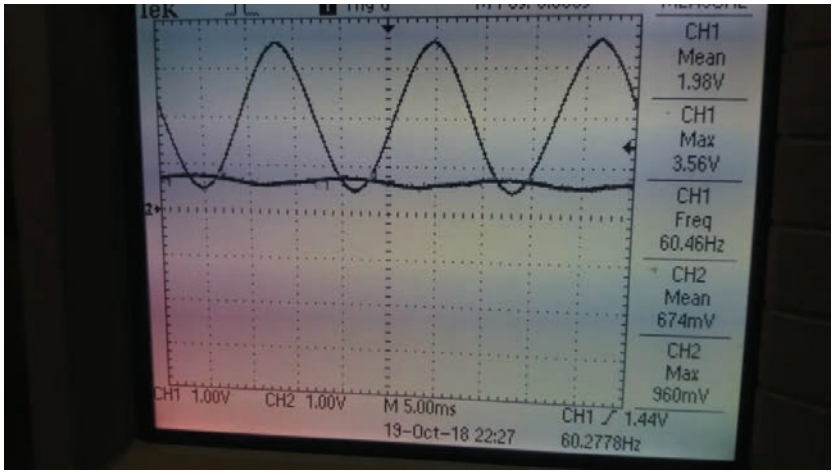
fonte:Produção do próprio autor.

Figura 23 – Saída do filtro para uma entrada com frequencia menor que 60 HZ



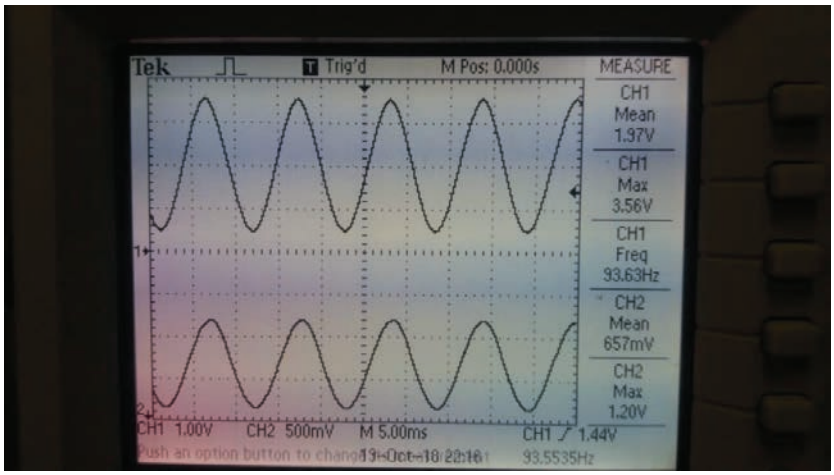
fonte:Produção do próprio autor.

Figura 24 – Saída do filtro para uma entrada com frequencia igual 60 HZ



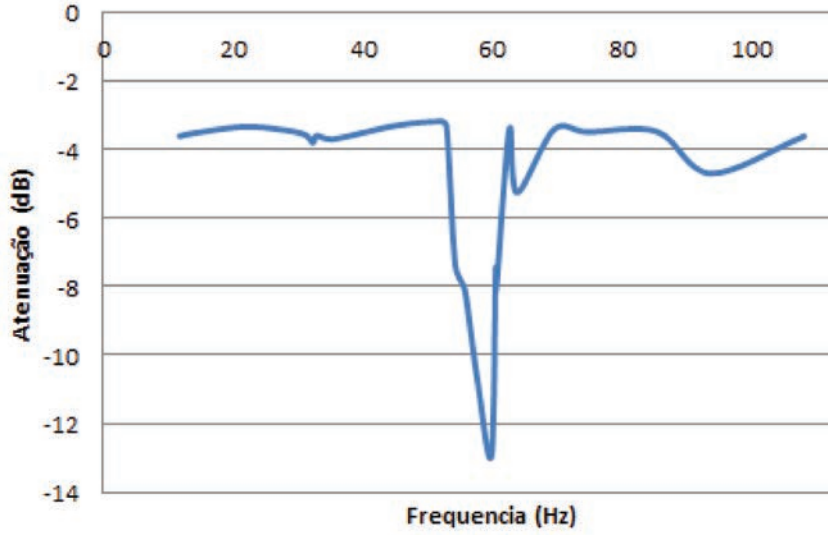
fonte:Produção do próprio autor.

Figura 25 – Saída do filtro para uma entrada com frequencia maior que 60 HZ



fonte:Produção do próprio autor.

Figura 26 – Resposta em frequência do filtro



fonte:Produção do próprio autor.

3 CONCLUSÃO

Apesar da atenuação teórica, de acordo com o winfilter, ser de mais de 30 decibéis, foi alcançada uma atenuação de 14 dB, que ao se corrigir a atenuação de 3 dB do filtro iria cair para mais ou menos 10 dB, porém, apesar de estar abaixo do esperado, significa que qualquer sinal de 60 Hz será 16 vezes menor na saída, o que, considerando que induções eletromagnéticas não iriam gerar sinais altos o suficiente para que o seu resíduo seja um problema, significa que este filtro consegue realizar o seu trabalho.

Outro fator que pode ser considerado são as diversas formas que um arduino pode ser usado para filtrar um sinal, por exemplo, pode-se utilizar o seu cabo USB para transmitir a entrada e saída para serem exibidos em um computador, ou ainda é possível utilizar o arduino somente para amostragem e processar todos os dados usando um computador, para isso existem várias linguagens de programação que permitem a comunicação serial com um arduino usando uma porta USB.

Com isso, é possível chegar a duas conclusões. A primeira é que não existe forma correta, nem errada de se criar um filtro digital, basta sempre ter em mente o propósito desse filtro e de sua montagem e que ele funcione corretamente.

A segunda conclusão é que o filtro digital discutido acima cumpre os seus dois propósitos, o de filtrar sinais de 60 Hz induzidos pela rede elétrica em equipamentos de medição e o segundo o de ser implementado utilizando dispositivos eletrônicos em vez de utilizar um computador, permitindo que aparelhos que possam ser beneficiados pela sua existência não precisem de um computador, apenas do arduino e do conversor DA.

REFERÊNCIAS

HOROWITZ, W. H. P. **The art of electronics.** New York: Cambridge University Press, 1989.

OPPENHEIM A. V.; SCHAFER, R. W. **Digital signal processing.** Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1975.

TOMPKINS W. J.; WEBSTER, J. G. **Design of microcomputer-based medical instrumentation.** Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1981.

APÊNDICE A – SCRIPT PYTHON QUE SIMULA UM FILTRO IIR

```
import math

Acoef = []
Bcoef = []
Ncoef = 8
y=[]
x=[]

freq = 60
file = open("4 ordem.txt",'w')
n=0
while n <= Ncoef:
   y.append(0)
   x.append(0)
   n = n+1

amostra = 0
while amostra < 3000:
   angulo = (freq*2*math.pi*amostra/1000)
   entrada = math.sin(angulo) + math.sin(2*math.pi*amostra/1000)
   if amostra > 1000:
      entrada = entrada +1 - math.sin(2*math.pi*amostra/1000)
   print("amostra"+str(amostra))
   n = Ncoef
   while n>0:
      x[n] = x[n-1]
      y[n] = y[n-1]
      n = n-1
   x[0] = entrada
   y[0] = Acoef[0] * x[0]
   n = 1
   while n <= Ncoef:
      y[0] = y[0] + Acoef[n] * x[n] - Bcoef[n] * y[n]
      n = n+1
   saida = y[0]
   amostra = amostra + 1
   file.write(str(entrada)+","+str(saida)+"\n")

file.close()
```

APÊNDICE B – SKETCH ARDUINO QUE GERA UMA RAMPA

```
int saida[8] = {30,31,32,33,34,35,36,37};
int resultado=0;
unsigned long tempo,tempomilli;
int i;

void setup()
{
  Serial.begin(9600);
  pinMode(saida[0],OUTPUT);
  pinMode(saida[1],OUTPUT);
  pinMode(saida[2],OUTPUT);
  pinMode(saida[3],OUTPUT);
  pinMode(saida[4],OUTPUT);
  pinMode(saida[5],OUTPUT);
  pinMode(saida[6],OUTPUT);
  pinMode(saida[7],OUTPUT);
}
void loop()
{
    delay(1);
    resultado++;
    if(resultado == 256)
    {
      resultado = 0;
    }
    Serial.print(resultado);
    Serial.print("\n");
    for(i = 0;i<=7;i++)
    {
      if(bitRead(resultado,i) == 1)
      {
        digitalWrite(saida[i],HIGH);
      }
      else
      {
        digitalWrite(saida[i],LOW);
      }
    }
}
```

APÊNDICE C – SKETCH ARDUINO DO FILTRO NOTCH PARA 60 HZ

```
const int entrada = A8;
const int saida[8] = {30,31,32,33,34,35,36,37};
const int NCoef = 6;
const int led = 13;
int n,i,amostra;
const float ACoef[7] = {
      0.88435681612444139000,
      -4.93596063521563800000,
      11.83628207674771300000,
      -15.56695425008728000000,
      11.83628207674771300000,
      -4.93596063521563800000,
      0.88435681612444139000
   };
const float BCoef[7] = {
      1.00000000000000000000,
      -5.46453532681942190000,
      12.83024204101259100000,
      -16.52352610386716600000,
      12.30376062619022500000,
      -5.02528342025197720000,
      0.88189313056979091000
   };
float x[NCoef + 1]={0,0,0,0,0,0,0};
float y[NCoef + 1]={0,0,0,0,0,0,0};
float incremento = 0;
int resultado = 255;
unsigned long tempo = 0;

void setup() {
   pinMode(entrada,INPUT);
   pinMode(saida[0],OUTPUT);
   pinMode(saida[1],OUTPUT);
   pinMode(saida[2],OUTPUT);
   pinMode(saida[3],OUTPUT);
   pinMode(saida[4],OUTPUT);
   pinMode(saida[5],OUTPUT);
   pinMode(saida[6],OUTPUT);
   pinMode(saida[7],OUTPUT);
```

```
}

void loop()
{
   if(millis() != tempo)
   {
      tempo = millis();
      digitalWrite(led,HIGH);
      amostra = analogRead(entrada)*256/1024;
      resultado = filtro(amostra);
      for(i = 0;i<=7;i++)
      {
         if(bitRead(resultado,i) == 1)
         {
            digitalWrite(saida[i],HIGH);
         }
         else
         {
            digitalWrite(saida[i],LOW);
         }
      }
      digitalWrite(led,LOW);
   }
}

int filtro(int entrada)
{
   x[0] = entrada;
   y[0]=ACoef[0]*x[0];

   for(n=1;n<=NCoef;n++)
   {
    y[0] += ((ACoef[n]*x[n])-(BCoef[n]*y[n]));
   }

   for(n=NCoef; n>0; n--)
   {
    x[n] = x[n-1];
    y[n] = y[n-1];
   }
   return int y[0];
}
```