Desenvolvimento de um Filtro Passa-Faixa e Notch Digital

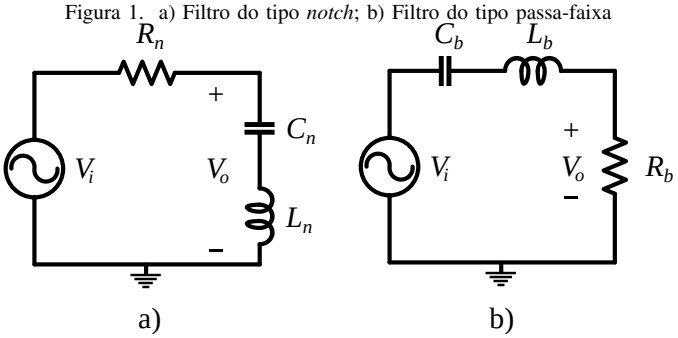
Cassius Rossi de Aguiar*, Gabriel Alexandre de Souza Braga†, Gianluca Hiss Garbim‡,
Guilherme Gabriel de Oliveira§ e Marcus Vinícius Silvério¶
*Coordenação de Engenharia de Computação
Universidade Tecnológica Federal do Paraná, Toledo, Paraná 85902-490
Email: *cassiusaguiar@utfpr.edu.br, †gabrielbraga@alunos.utfpr.edu.br, ‡gianluca@alunos.utfpr.edu.br,
§guioli@alunos.utfpr.edu.br, ¶marcussilverio@alunos.utfpr.edu.br

***Resumo*—Através dos conhecimentos adquiridos na matéria de Sensores e Atuadores, foram desenvolvidos dois filtros digitais: um passa-faixa e um *notch*. A plataforma de prototipagem Arduino Due foi utilizada para implementar e testar os filtros, com o código dos mesmos desenvolvido em C/C++. Foi possível, então, aprofundar o conhecimento na área de projeto de filtros digitais.**

I. Introdução

Além dos filtros que controlam a passagem de frequências acima ou abaixo de uma previamente especificada, temos também filtros que permitem ou negam a passagem de faixas específicas de frequências. Estes são os filtros passa-faixa (ou passa-banda), que permite apenas uma faixa específica de frequência, e os filtros *notch*, que impedem a passagem de frequências dentro desta banda especificada. É possível observar estas diferenças na figura 1.

Figura 1. a) Filtro do tipo *notch*; b) Filtro do tipo passa-faixa



Fonte: Autoria própria (2022)

Estes são descritos através de funções de transferência. Com base na análise dos circuitos apresentados, é possível definir

$$\frac{V_o}{V_i} = \frac{s^2 + \omega_0^2}{s^2 + \omega_c \cdot s + \omega_0^2} \quad (1)$$

$$\frac{V_o}{V_i} = \frac{\omega_c \cdot s}{s^2 + \omega_c \cdot s + \omega_0^2} \quad (2)$$

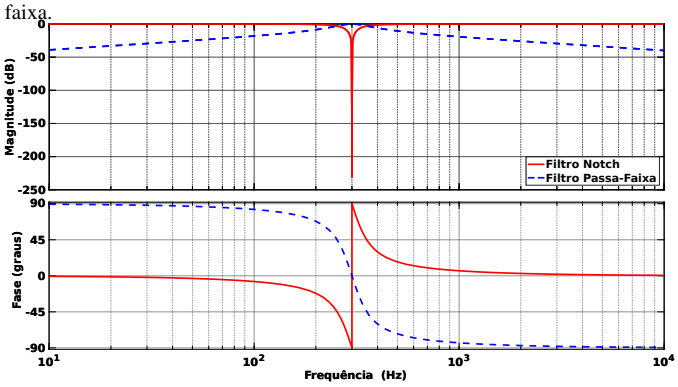
Nas quais a equação 1 descreve o comportamento do filtro *notch*, enquanto 2 descreve o comportamento do filtro passa-faixa.

ω_c representa o alcance da faixa de rejeição ou aceitação de frequências, ω_0 representa a frequência de ressonância.

II. Método

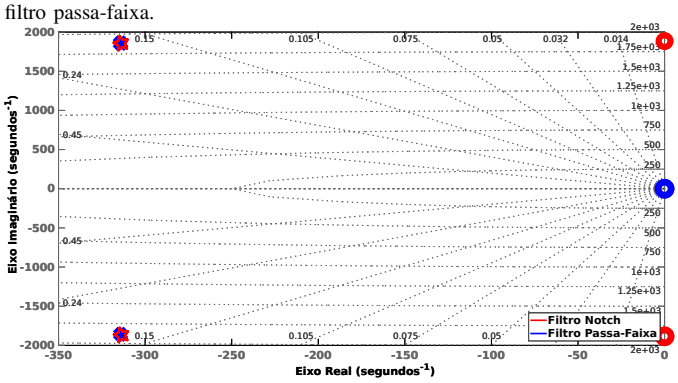
Definindo que $\omega_c = 200\pi\frac{rad}{s}$ e $\omega_0 = 600\pi\frac{rad}{s}$, e com base nas equações 1 e 2, foi possível traçar os diagramas de Bode e o lugar geométrico das raízes para ambos os filtros.

Figura 2. Diagrama de Bode. Em vermelho filtro *Notch*. Em azul filtro passa-faixa.



Fonte: Autoria própria (2022).

Figura 3. Lugar Geométrico das Raízes. Em vermelho filtro *Notch*. Em azul filtro passa-faixa.



Fonte: Autoria própria (2022).

A geração dos gráficos do diagrama de Bode permitiu observar o funcionamento teórico adequado dos filtros projetados. Seguimos, então, para a discretização das funções de transferências (equações 1 e 2). Obtemos as discretizações para cada função: a equação 3 se refere a do filtro *notch* enquanto a equação 4 se refere a do filtro passa-faixa.

$$V_o[K] = 0.9698010481 \cdot (V_i[K] + V_i[K-2]) + 1.9054478861 \cdot (V_o[K-1] - V_i[K-1]) - 0.9396020961 \cdot V_o[K-2] \quad (3)$$

$$V_o[K] = 0.0301989519 \cdot (V_i[K] - V_i[K-2]) + 1.9054478861 \cdot V_o[K-1] - 0.9396020961 \cdot V_o[K-2] \quad (4)$$

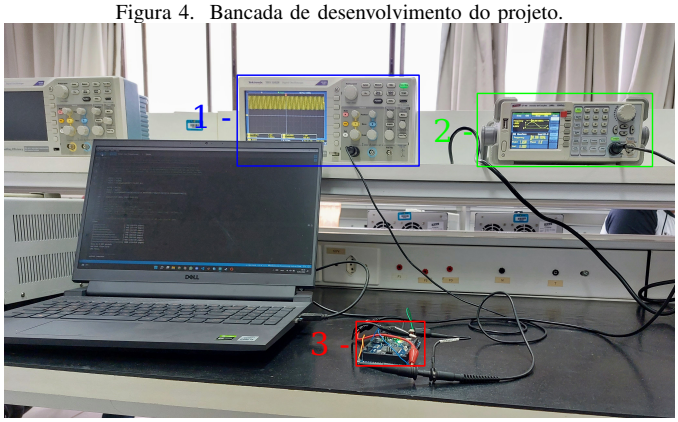
Com as discretizações é possível elaborar o código que será implementado no Arduino.

```
void PWM_Handler()
{
  volatile long dummy = PWM_INTERFACE->...
    PWM_ISR1; // clear interrupt flag
  dummy = PWM_INTERFACE->PWM_ISR2; // clear ...
    interrupt flag

  digitalWrite(13,HIGH);
  // Filtro Notch
  Vi[2] = Vi[1];
  Vi[1] = Vi[0];
  Vi[0] = ((analogRead(A0)*3.3)/4095.0) - ...
    (1.65/2.0);

  Vo[2] = Vo[1];
  Vo[1] = Vo[0];
  Vo[0] = 0.9698010481*(Vi[0]+Vi[2])...
    +1.9054478861*(Vo[1]-Vi[1])-0.9396020961*...
    Vo[2];

  analogWrite( DAC1, ((Vo[0]+(1.65/2.0))...
    *3071.0)/1.65 );
  /*
  // Filtro Passa-Faixa
  Vi[2] = Vi[1];
  Vi[1] = Vi[0];
  Vi[0] = ((analogRead(A0)*3.3)/4095.0) - ...
    (1.65/2.0);

  Vo[2] = Vo[1];
  Vo[1] = Vo[0];
  Vo[0] = 0.0301989519*(Vi[0]-Vi[2])...
    +1.9054478861*Vo[1]-0.9396020961*Vo[2];

  analogWrite( DAC1, ((Vo[0]+(1.65/2.0))...
    *3071.0)/1.65 );
  */
  // Atualizacao dos PWMs, 1050 representa um...
     Duty de 50% para freq de 10kHz
  PWMC_SetDutyCycle(PWM_INTERFACE, channel_1,...
     42000*saida);
  PWMC_SetDutyCycle(PWM_INTERFACE, channel_2,...
     42000*saida1);
  PWMC_SetDutyCycle(PWM_INTERFACE, channel_3,...
     42000*saida2);
  PWMC_SetDutyCycle(PWM_INTERFACE, channel_4,...
     20);
  digitalWrite(13,LOW);
  PWM_INTERFACE->PWM_IDR1 = 0x10; //enable ...
    interrupt on channel 4 - 00010000
}
```

Listing 1. Código introduzido no Arduino.

Na figura 4 é possível ver a bancada de teste, 1 representa o osciloscópio utilizado, 2 é o gerador de função e 3 é o Arduino DUE.

Figura 4. Bancada de desenvolvimento do projeto.



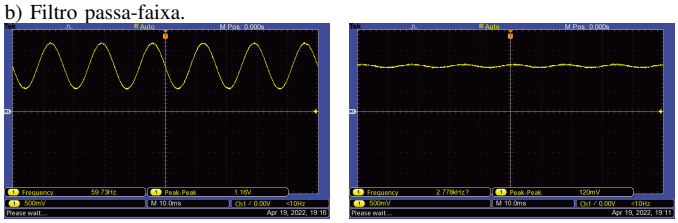
Fonte: Autoria própria (2022).

III. Resultados

Tendo em mãos as equações discretizadas e o código implementado, é possível realizar a implementação e testes subsequentes. Foram analisadas as saídas referentes a entradas de 60Hz, 300Hz e 600Hz para ambos os filtros.

A figura 5 demonstra o sinal de saída dos filtros *notch* e passa-faixa para 60 Hz, enquanto a figura 6 apresenta as saídas para 300Hz e a figura 7 para 600Hz.

Figura 5. Sinal de saída para 60 Hz, medido no osciloscópio. a) Filtro *notch*; b) Filtro passa-faixa.

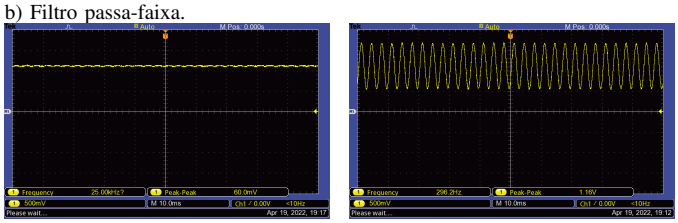


a) b)

Fonte: Autoria própria (2022).

É possível observar, na figura 5, o correto funcionamento de ambos os filtros. O filtro *notch* permite a passagem do sinal, enquanto o passa-faixa o atenua quase completamente.

Figura 6. Sinal de saída para 300 Hz, medido no osciloscópio. a) Filtro *notch*; b) Filtro passa-faixa.

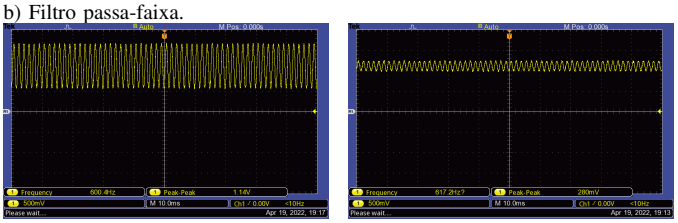


a) b)

Fonte: Autoria própria (2022).

A figura 6 também exibe o comportamento esperado do filtro, impedindo a passagem do sinal à frequência de rejeição no filtro *notch* e transmitindo-a completamente no passa-faixa.

Figura 7. Sinal de saída para 600 Hz, medido no osciloscópio. a) Filtro *notch*; b) Filtro passa-faixa.



a) b)

Fonte: Autoria própria (2022).

A saída de 600 Hz (figura 7) também demonstra funcionamento adequado, com passagem do sinal através do filtro *notch* e bloqueio do mesmo no filtro passa-faixa.

IV. Conclusão

Como é possível analisar nas figuras, os filtros se comportam da forma prevista nas demonstrações teóricas e simulações. As relações de entrada e saída não são perfeitas, porém, devido a natureza do Arduino e do funcionamento de filtros digitais.

Além das pequenas perdas previamente observadas, também é possível observar um efeito de defasagem, este é visto nas figuras 6 a) e 5 b). O efeito de defasagem ocorro por conta da atenuação da banda de rejeição, como é possível observar no diagrama de Bode da figura 2. Este não é um problema, dado o fato que ocorre apenas nas frequências de rejeição, que não são nossas frequências de interesse, dessa forma, é possível notar o corrento funcionamento dos filtros, deixando passar apenas as bandas de interesse, e não alterando seu comportamento.

O uso de filtros digitais permite uma implementação mais adaptável do que a construção de filtros analógicos. O fator custo-benefício também se mostra uma vantagem presente nestes tipos de filtro.

Dessa forma foi possível ampliar os conhecimentos obtidos na matéria, colocando-os em prática através do projeto de dois filtros digitais: *notch* e passa-faixa.