MI PDS - Problema 4 - Implementação de um Filtro em Dispositivos Embarcados

Alyson Dantas, Gabriel Sá, Lucas Cardoso, Marcelo Mota
Universidade Estadual de Feira de Santana

***Resumo*—This report describes the implementation of a filter digital to eliminate high frequencies in a programmable embedded device (MPLAB Starter Kit for dsPIC DM3330011), filtering implementation analysis in this type of device and choice in the number of coefficients used as input signal to be filtered the combination of signals entered by a MIC.**

***Index Terms*—Janelamento, Retangular, Bartlett, Hann, Hamming, Blackman, sinc, FIR**

I. Introdução

O diretor da ANATEL contatou o diretor de P&D, Sr. Gibbs, para uma reunião com a finalidade de solicitar a construção de uma solução visando utilizar filtros digitais embarcados, de forma a eliminar ruídos de altas frequências presentes em ligações telefônicas, tendo em mente que dispositivos embarcados possuem menor precisão, para melhor distribuição e aplicação.

Sendo assim, a equipe de engenheiros da Sigma Delta Inc. deveria implementar um filtro digital para eliminar altas frequências em um dispositivo embarcado programável (*MPLAB Starter Kit for dsPIC DM3330011*), utilizando como sinal a ser filtrado um sinal de áudio ambiente e um ruído de alta frequência gerado por um aplicativo gerador de sinal de áudio. A combinação dos sinais de teste seria realizado por um microfone externo conectado ao dispositivo. O projeto deverá ser capaz de avaliar os efeitos da implementação da filtragem realizada em dispositivos de menor precisão, e a análise sobre a escolha da quantidade de coeficientes do filtro.

II. Metodologia

Devido ao requisito da filtragem de frequências superiores a uma determinada faixa limite e deixar passar todas as demais inferiores, definiu-se o uso do filtro Passa-Baixas (*Low-Pass*).

Em seguida, com a escolha do filtro, foi preciso verificar quais ferramentas seriam utilizadas. Por se tratar de um kit desenvolvido pela empresa Microchip, optou-se por utilizar o MPLAB v3.65, pois é uma *Integrated Development Environment* (IDE) dedicada a programação de dispositivos embarcados dessa natureza disponibilizada pela própria Microchip.

O kit de desenvolvimento utilizado, conforme mostrado na Figura 1, foi o *dsPIC33FJ256GP506*. Trata-se de um dispositivo com recursos limitados, possuindo apenas 256Kb de memória flash, 16Kb de memória RAM, 48KHz de frequência máxima de amostragem e um conversor analógico digital configurável de 10 ou 12 bits. Também, todos os sinais de entrada inseridos no dispositivo são limitados em banda na faixa de 3.3KHz. Essa limitação é realizada pelo próprio dsPIC. Na tabela I, temos a especificação da enumeração apresentada na Figura 1.

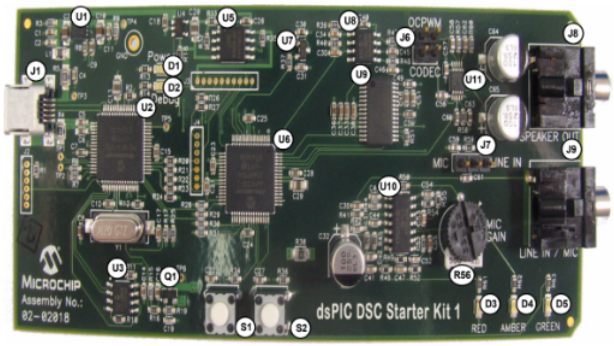

Figura 1. MPLAB STARTER KIT Modelo dsPIC33FJ256GP506

Tabela I
Tabela de componentes

Item	Componente
D1	Mini-B USB Connector (J1)
D2	MCP1727 (U1)
D3	STATUS LED - Debug (D2)
D4	STATUS LED - System Power (D1)
D5	PIC18F67J50 MCU (U2)
D6	Low V_{CE} Saturation PNP Transistor Switch (Q1)
D7	25LC010A Serial EEPROM (U3)
A1	Flash Memory (U5)
A2	Digital Signal Control (U6)
A3	Temperature Sensor (U7)
A4	PWM Low-Pass Filter (U8:A,B)
A5	Output Select Jumper (J6)
A6	Codec (U9)
A7	Headphone Amplifier (U11)
A8	Line/Microphone Input Select Jumper (J7)
A9	Headphone Stereo Output Jack (J8)
A10	Line/Microphone Input Phone Jack (J9)
A11	Use LEDs (D3, D4 e D5)
A12	Microphone Gain Control (R56)
A13	Line/Microphone Pre-Amplifier (U10:A)
A14	Anti-Aliasing Low-Pass Filter (U10:B,C,D)
A15	Use Switches (S2 e S1)

A Figura 2 apresenta um diagrama de blocos do dsPic, demonstrando o seu processamento inicial de um sinal de entrada, desde a captura do áudio, amplificação do sinal de entrada, limitação em frequência por meio de um filtro *anti-aliasing*, até a reconstrução do sinal processado.

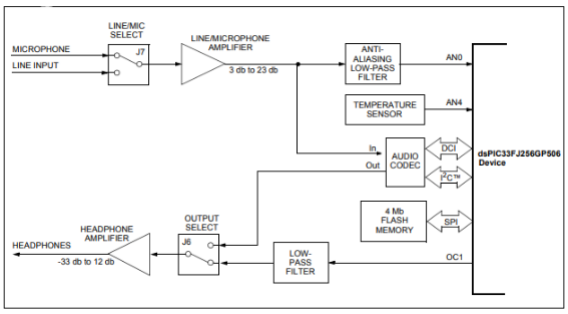

Figura 2. STARTER KIT Block Diagram

Com o entendimento inicial do funcionamento do dsPic partiu-se para as configurações mais específicas do filtro e da função de janelamento utilizada como referência para desenvolver o projeto, além dos testes de simulação antes de realizar a implementação no dispositivo.

A. *Parâmetros do Filtro Passa-Baixa*

Com a definição do tipo de filtragem a ser implementado no projeto, utilizou-se de um *software* desenvolvido pela equipe para realizar o projeto do filtro *low pass*. A Figura 3 ilustra o *software* e os parâmetros utilizados.

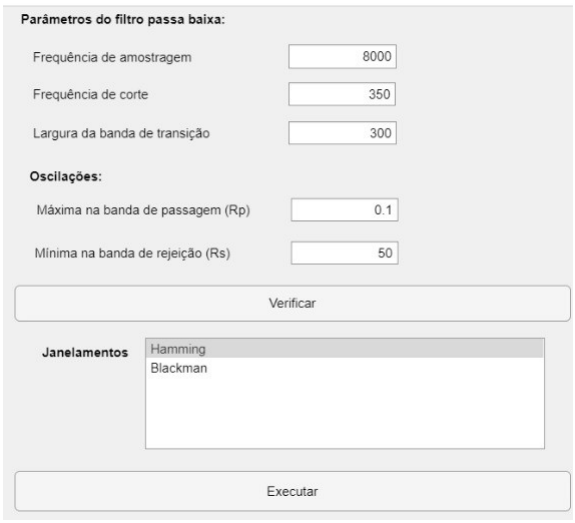

Figura 3. Parâmetros para o filtro Passa-Baixa

Escolheu-se a frequência de amostragem em 8KHz, frequência de corte em 350Hz, largura da banda de transição em 300Hz, uma atenuação mínima na banda de rejeição com valor de 50dB, e janela de Hamming para aproximação da resposta ideal. A equação de janelamento por Hamming é exibida na equação (1).

$$w[n] = \begin{cases} 0,54 - 0,46cos(\frac{2n\pi}{M}), & 0 \leq n \leq M \\ 0, & \text{caso contrário} \end{cases} \quad (1)$$

De acordo aos parâmetros especificados na Figura 3, o *software* desenvolvido realiza a análise de quais funções de Janelamento se encaixam no projeto. Logo, as funções disponíveis foram Hamming e Blackman. Dentre as duas, escolhemos a *Hamming*. A escolha de qual seria utilizada se deu pela verificação de qual fornecia um menor número de coeficientes, além de manter a atenuação mínima na banda de rejeição conforme desejado. A utilização de menos coeficientes se torna essencial devido a limitação do *hardware* utilizado, obrigando assim a modelagem de um filtro composto por poucas amostras, e que ao mesmo tempo, atenda aos requisitos de projeto. A Figura 4 mostra a janela do filtro projetado, pode-se observar que para os parâmetros informados será necessário utilizar 88 coeficientes. A Figura 5 mostra o comportamento da magnitude do filtro projetado.

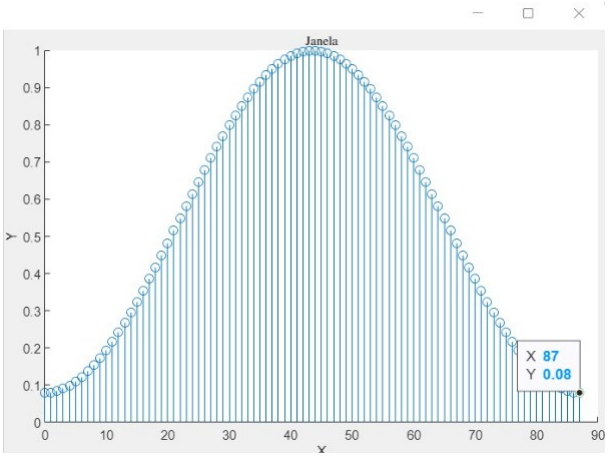

Figura 4. Janela do filtro projetado

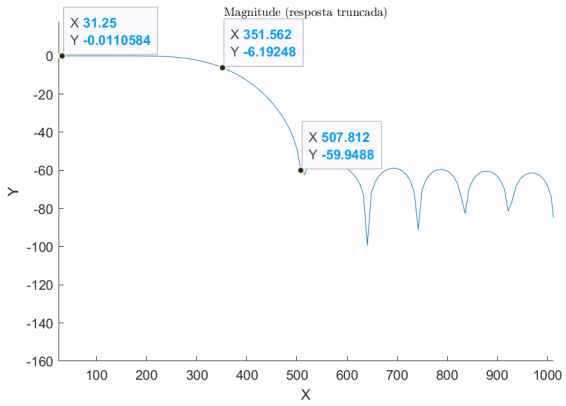

Figura 5. Magnitude do filtro projetado

A partir do *software* e dos parâmetros informados, obteve-se, além da informação da quantidade de coeficientes, os valores que cada coeficiente assume. Estes valores serão indispensáveis para a etapa de implementação do filtro no dsPIC.

B. *Implementação no dsPIC*

Conforme citado anteriormente, utilizou-se o *software* de desenvolvimento específico do fabricante da placa: MPLAB

X na sua versão 3.65. Dentre as linguagens de programação disponíveis optou-se pelo uso da linguagem C, pois, além da placa possuir suporte, é uma linguagem na qual os integrantes da equipe possuem domínio prévio. Além disso, já existem bibliotecas fornecidas pelo próprio fabricante para intermediar a integração entre o *software* e o *hardware*, reduzindo o tempo de desenvolvimento.

1) Configuração do ADC: A fim de realizar a correta captura do áudio se fez necessário a configuração do *ADC* presente no kit de desenvolvimento, para isso foram definidos alguns parâmetros:

- Clock do *ADC* em 40MHz;
- Frequência de amostragem em 8KHz;
- 12 bits de operação;
- *Buffer* de 128 amostras

Com os parâmetros definidos, configurou-se o *ADC* por meio do arquivo ADCChanelDRV.h. Com o auxilio do datasheet [5] foram determinados os valores de cada constante para configuração do ADC, representado na Figura 6.

```
#ifndef __ADCCHANNELDRV_H__
#define __ADCCHANNELDRV_H__

#define ADC_CHANNEL_FCY          40000000
#define ADC_FSAMP                8000            /* Sampling Frequency   */
#define ADC_BUFFER_SIZE          128             /* This is the size of each buffer  */
#define ADC_CHANNEL_DMA_BUFSIZE (ADC_BUFFER_SIZE*2)

#include<p33Fxxxx.h>

typedef struct sADCChannelHandle {
    int * buffer1;
    int * buffer2;
    volatile int bufferIndicator;
    volatile int isReadBusy;

}ADCChannelHandle;

void ADCChannelInit (ADCChannelHandle * pHandle,int * pBufferInDMA);
void ADCChannelStart    (ADCChannelHandle * pHandle);
void ADCChannelRead (ADCChannelHandle * pHandle,int *buffer,int size);
int ADCChannelIsBusy    (ADCChannelHandle * pHandle);
void ADCChannelStop (ADCChannelHandle * pHandle);

#define ADCON1VAL       0x0744      /* 12 bit ADC with signed fractional format
                                       * Triggered by Timer 3 and auto start
                                    * sampling after conversion complete. */
#define ADCON2VAL       0x0000      /* AVdd and AVss voltage reference,
                                    * use channel 0 with no scan   */
#define ADCON3VAL       0x0010      /* Tad is 16 Tcy                */
#define ADCH5VAL            0x0000      /* AN0 input on channel 0     */
#define ADPCFGVAL       0xFFFE      /* AN0 input is Analog          */
#define ADCSSLVAL       0x0000      /* No channel scanning          */

#endif
```

Figura 6. Arquivo ADCChanelDRV.h de configuração para inicializar o ADC.

Os parâmetros da frequência de amostragem e quantidade de amostras, como já citado, foram provenientes do *software* de projeto de filtros utilizado anteriormente, os demais parâmetros são referentes à especificidades prévias da placa, como clock de operação do ADC, quantidade de bits de precisão a serem utilizados, tamanho do *buffer*, etc.

O valor de 128 amostras foi assumido após testes realizados com a placa, pois verificou-se que com valores menores o áudio resultante na saída apresentava distorções (voz "robotizada") ou ruídos, assim, optou-se esse valor suportado pelo *hardware* do dsPIC para apresentar uma saída de áudio sem muitas distorções.

2) Configuração dos valores dos coeficientes: Como já citado, o *software* de projeto de filtros desenvolvido anteriormente pela equipe e utilizado nesse projeto, permitiu além do cálculo da quantidade de coeficientes necessários, encontrar também seus respectivos valores. Sendo assim, para maior facilidade de implementação desses coeficientes no dispositivo, criamos um *script* no MATLAB visando a captura dos coeficientes e sua formatação, de forma a facilitar a inserção no *software* do dsPIC através do MPLAB X IDE. A Figura 7 mostra o código criado para a formatação dessas informações.

```
app.hn = app.hd.*app.w;

v_hn = int32(app.hn'*10000);

for i = 1:length(app.hn)

    vetorhnString = ['h[',num2str((i-1)),'] = ', num2str(v_hn(i)), ';'];

    disp(vetorhnString)
end
```

Figura 7. Código para formatação dos valores dos coeficientes

Em seguida, antes da realização das simulações, foi necessário compreender o conceito de soma de convolução.

C. Soma de Convolução

A resposta de um sistema linear invariante no tempo(*LTI*) é a soma ponderada das respostas ao impulso deslocadas no tempo [1]. Sua expressão é dada por

$$y[n] = x[n] * h[n] \tag{2}$$

ou de forma equivalente,

$$\sum_{k=-\infty}^{\infty} x[k].h[n-k] \tag{3}$$

onde é conhecida como soma de convolução para o tempo discreto. Sua aplicação se dá na geração de um terceiro sinal a partir de outras duas entradas em um sistema. Nesse projeto, é importante esse conceito porque é através dessa operação que será gerado o novo sinal com menos ruído a partir de um sinal de entrada e do filtro projetado.

Assim, com esses conceitos em mãos, partiu-se para a realização de simulações no *MATLAB* para validação.

III. Resultados e Discussões

A. Simulação no MATLAB

Para validar o funcionamento do filtro projetado com os parâmetros anteriormente citados, realizou-se a filtragem utilizando o *software MATLAB* de um arquivo de áudio gravado no ambiente de sessão tutorial, utilizou-se também os coeficientes encontrados durante o projeto do *low pass*, para assim, realizar uma convolução entre um sinal de entrada e o filtro projetado. A função de convolução utilizada foi a função *conv()* disponível pela própria plataforma.

$$y(k) = \sum_{n} h(n)x(k-n+1) \tag{4}$$

A Figura 8 mostra o resultado do teste no domínio do tempo, onde têm-se um sinal de entrada inicialmente e o mesmo sinal menos acentuado após a realização da convolução com o filtro. Observou-se que os coeficientes encontrados possuíam valores muito pequenos, assim, fez-se necessário a aplicação de um ganho de 34dB na saída da simulação no *MATLAB*.

A Figura 9 mostra o espectro de frequência do sinal original e do sinal já filtrado (com ganho). Pode-se perceber a filtragem em grande parte do sinal de entrada.

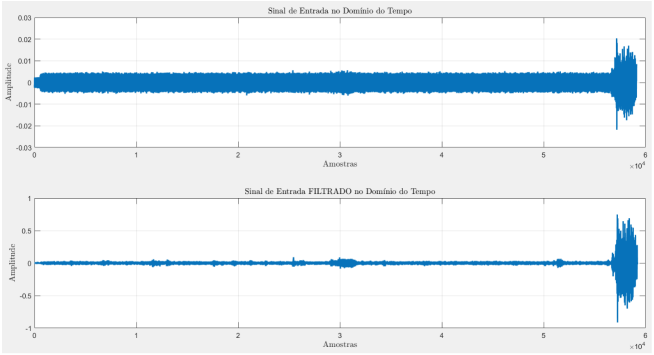

Figura 8. Teste de filtragem em ambiente *MATLAB*

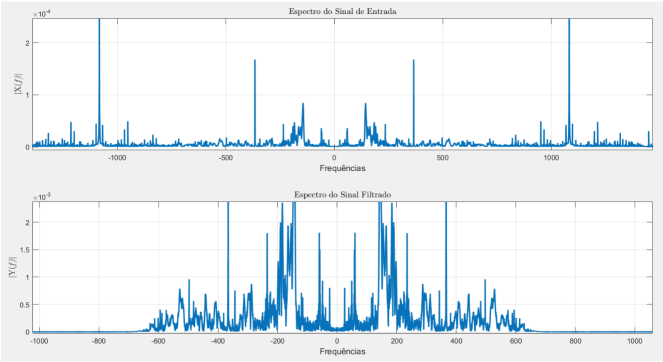

Figura 9. Espectros de frequência do teste de filtragem em ambiente *MATLAB*

Com a garantia de que o projeto do filtro funciona corretamente no ambiente de simulação, passou-se, para a etapa de implementação no dsPIC.

B. Normalização dos coeficientes do filtro

Observamos que nossas amostras no embarcado dsPic eram inteira e em contra partida os coeficientes obtidos através do *MATLAB* estavam em ponto flutuante, de inicio ocorreu o objetivo de normalizar as amostras multiplicando elas pela tensão de referência 3.3V dividida por dois elevado a doze, quantidade de bits configuradas anteriormente no ADC, o resultado da normalização foi o desejado, porem descobrimos que o dispositivo embarcado dsPic não opera de forma satisfatória com ponto flutuante.

Então optamos por normalizar os coeficientes do filtro, multiplicando cada um por um valor alto suficiente para que a parte fracionaria significativa se torne inteira é possível garantir que o filtro mantenha seu comportamento, porem é necessário realizar a divisão por esse mesmo valor após as operações, afim de manter a fidelidade da entrada. O valor definido após alguns teste foi de 10K, esse valor permitiu que coeficientes fracionários pequenos como por exemplo 0,00127 em inteiros 12, tornando capaz de operar com segurança no dsPic, apnas dividindo o resultado da operação de convolução por 10K.

1) Implementação da Convolução: Conforme a fórmula matemática citada em 4, desenvolveu-se um algoritmo em C que representa de forma satisfatória esse comportamento no dispositivo. Esse algoritmo pode ser visto na Figura 10, é importante notar que o ganho na simulação foi dado em 50, porem o mesmo ganho na implementação do embarcado dsPic ficou muito alto, reduzimos para 10, então como ocorreu um ganho de 10K por conta da conversão dos coeficientes do filtro para inteiro, após a convolução só precisamos dividir o resultado por 1K.

```
for (i = 0; i < FRAME_SIZE + N -1; i++) {
    for (j = 0; j <N; j++) {
        if (i-j <0) {
            continue;
        }
        if (i - j < N-1) {
            k = i - j;
            currentSample = samplesBuffer[k];
        } else {
            k = i - j - N +1;
            currentSample = samples[k];
        }
        hn = h[j];
        hn = hn * currentSample; //multiplicação do hn por x[n]
        y_aux += hn;
    }

    if( i > N-2 ){
        amostra = (int) (y_aux/1000); // Desnormalizando e atribuindo ganho de 10x.
        samplesOut[i-N+1] = amostra;
    }
    y_aux = 0;
}

for(i = 0; i<N-1; i++){
    samplesBuffer[i] = samples[FRAME_SIZE - N +1 +i];
```

Figura 10. Algoritmo em C para cálculo da convolução

C. Testes realizados

Para testar e validar o filtro no dsPic, utilizamos 3 configurações de frequência de corte e largura da banda de transição, que geram 3 grupos de coeficientes diferentes para o filtro passa-baixa, os valores são representados na tabela II. Em todos os testes utilizamos janelamento de Hamming pois queríamos a atenuação miníma na banda de rejeição em 20dB, com a menor quantidade de coeficientes possíveis.

Tabela II
VALORES DE FREQUÊNCIA DE CORTE E LARGURA DE BANDA DE TRANSIÇÃO PARA TESTES.

Frequência de corte	Banda de transição	Coeficientes do filtro
350Hz	600Hz	44
2000Hz	400Hz	53
50Hz	300Hz	88

Assim com esses diferentes valores para frequência de corte e diferentes tamanhos para os grupos de coeficientes, é possível observar e validar o comportamento do filtro implementado em hardware embarcado, realizamos a gravação de áudio com e sem filtro, utilizando um notebook com a entrada de microfone conectado a saída de áudio do dsPic e o software *Audacity*, para visualizar o filtro operando.

Para o primeiro teste foi configurado uma frequência de corte em 350Hz e largura da banda de transição em 600Hz com um sinal de teste com um período de 5 segundos variando frequência de 1Hz até 2000Hz e depois retorna para 1Hz. O resultado obtido sem filtro pode ser visualizado na Figura 11, podemos observar que o microfone que captou o áudio possui um filtro embarcado o que limita um pouco o sinal próximo ao máximo, porem o filtro projetado deve ser capaz de filtrar ainda mais esse sinal.

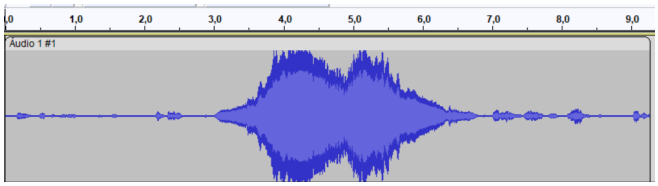

Figura 11. Sinal com período de 5s variando frequência de 1Hz até 2000Hz.

O filtro é representado na Figura 12 onde podemos observar que o filtro começa a atenuar a partir de uma frequência de aproximadamente 300Hz e em aproximadamente 630Hz ele já deve estar filtrando completamente.

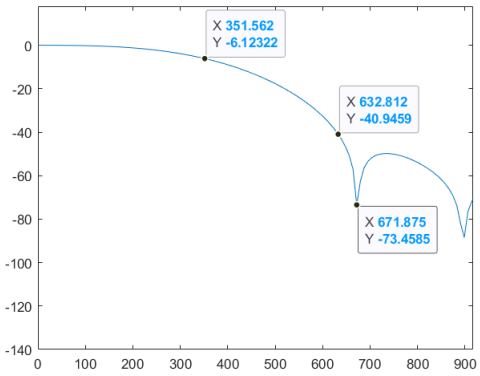

Figura 12. Filtro com frequência de corte em 350Hz e largura da banda de transição em 600Hz.

Na Figura 13 podemos observar o filtro operando, como esperado ele só permite a passagem de uma pequena parte do sinal, a parte que está antes do limite para o filtro começar a impedir, podemos observar a atenuação do filtro também.

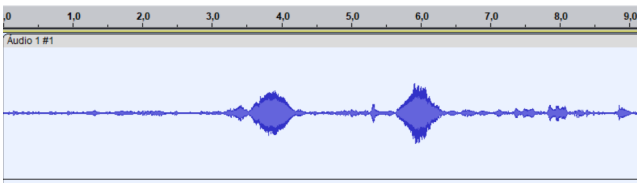

Figura 13. Frequência de corte em 350Hz e sinal com período de 5s variando frequência de 1Hz até 2000Hz com filtro.

No segundo teste foi configurado um filtro com frequência de corte em 2000Hz e um sinal de entrada com o período de 5 segundos variando frequência de 1Hz até 4000Hz e depois para 1Hz, o resultado obtido sem o filtro pode ser visualizado na Figura 14, também podemos observar o filtro embarcado no microfone que foi utilizado para captura atuando.

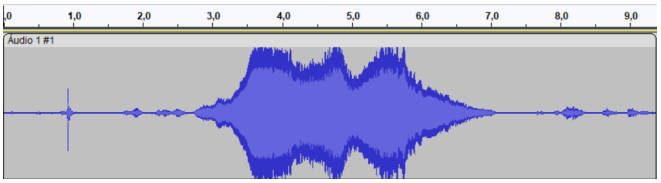

Figura 14. Frequência de corte em 2000Hz e sinal com período de 5s variando frequência de 1Hz até 4000Hz.

Na Figura 15 podemos visualizar o filtro referente a frequência de corte em 2000Hz e largura da banda d transição em 400Hz, podemos observar quem em aproximadamente 2200Hz até 2400Hz ele já deve estar filtrando completamente.

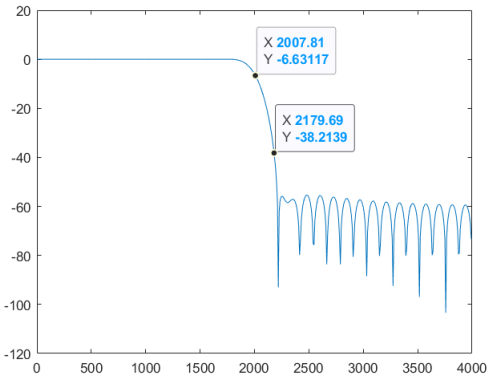

Figura 15. Filtro com frequência de corte em 2000Hz e largura da banda de transição em 400Hz.

O resultado da filtragem pode ser observado na Figura 16 podemos observar o comportamento da filtragem, operando como desejado. Existem alguns ruidos no inicio da gravação mas foram em decorrência do local de gravação, não do filtro ou do dsPic.

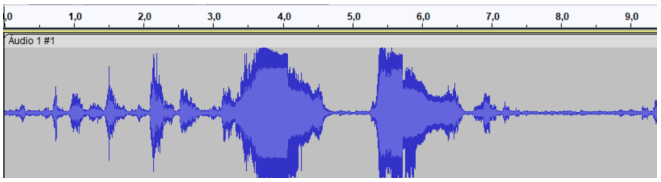

Figura 16. Frequência de corte em 2000Hz e sinal com período de 5s variando frequência de 1Hz até 4000Hz com filtro.

Por ultimo realizamos um teste com uma quantidade maior de coeficientes do filtro, para verificar um pouco mais o comportamento da dsPic e seus limites. Utilizamos a frequência

de corte mais baixa em 50Hz com banda de transição em 300Hz e obtemos 88 coeficientes. O sinal de teste foi em um período de 5 segundo e variando frequência de 1Hz até 2000Hz depois até 1Hz. O resultado obtido através do *Audacity* pode ser observado na Figura 17.

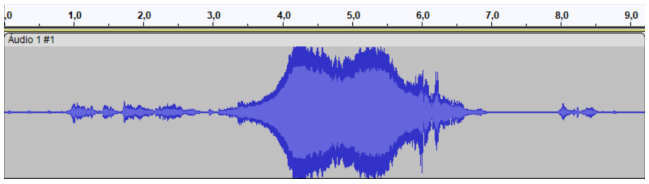

Figura 17. Sinal com período de 5s variando frequência de 1Hz até 2000Hz e depois para 1Hz.

Na Figura 18 é possível visualizar o filtro referente a frequência de corte em 50Hz e largura da banda de transição em 300Hz, ele começa a atenuar o sinal a partir do 50Hz e aproximadamente 200Hz até 220Hz o sinal já deve ser completamente filtrado.

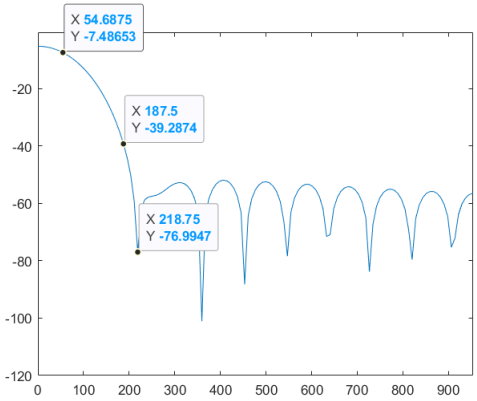

Figura 18. Filtro com frequência de corte em 50Hz e largura da banda de transição em 300Hz.

Porem no resultado revelado pela Figura 19 podemos observar que o filtro não obteve a mesma eficiência que nos outros testes, isso se deve as limitações de recursos que existem no hardware embarcado, então quanto mais coeficientes do filtro possui, mais incerto o hardware embarcado vai operar e com menos eficiência. O melhor resultado foi do primeiro teste com um filtro com menos coeficientes e com frequência de corte em 350Hz e largura da banda de transição em 600Hz.

IV. Conclusão

O projeto atende de forma satisfatória aos requisitos solicitados pela direção da Anatel. Entretanto, vale salientar, que por se tratar de um dispositivo de *hardware* embarcado com recursos limitados, o resultado final pode não ser tão satisfatório como o esperado se esse projeto tivesse sido aplicado em um *hardware* com mais abundância de recursos.

Um outro fator encontrado durante o desenvolvimento foi a escassez de materiais para auxiliar na compreensão do

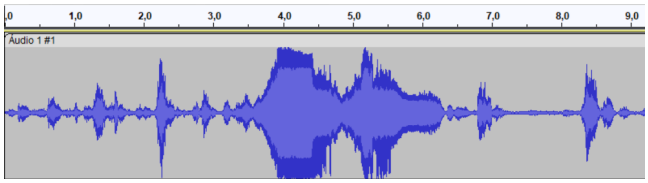

Figura 19. Frequência de corte em 50Hz e sinal com período de 5s variando frequência de 1Hz até 2000Hz com filtro.

dispositivo, limitando assim a recursos disponibilizados apenas pela Microchip. Esse fator, juntamente com a escassez de tempo, limitou algumas ideias iniciais, pois, houve a vontade da equipe em desenvolver a implementação de mais filtros de outras naturezas, propiciando no final um resultado muito mais robusto do que o solicitado.

Referências

[1] Oppenhem, A. V., Signal and Systems. Ed. Prentice Hall, Second Edition.
[2] Oppenhem, A. V. and SCHAFER R. W., ”Digital Signal Processing”Ed.Prentice Hall
[3] Haykin, S. and Veen, B. V. Sinais e Sistemas. Ed. Bookman. Published by Prentice Hall. 2007.
[4] A. O. e R. Schafer, Processamento em Tempo Discreto de Sinais. PearsonEducation do Brasil, 2012
[5] Datasheet dsPIC33FJ256GP506, MICROCHIP, Disponível em: https://www.microchip.com/en-us/product/dsPIC33FJ256GP506