Marcelo Batista Carmo

Ricardo Cagnacci Orozco

Filtro Digital com Crossover para Caixa Acústica Omni-Direcional

São Paulo

2009

Universidade de São Paulo Escola Politécnica

Departamento de Engenharia Elétrica

PSI 2594

Projeto de Formatura II

Filtro Digital com Crossover para Caixa Acústica Omni-Direcional

Relatório Final

Grupo:

- Ricardo Cagnacci Orozco N°USP: 5435769
- Marcelo Batista Carmo N°USP: 5436072

Email:

- orozco797@gmail.com
- marcelo.b.carmo@gmail.com

Orientador:

- Professor Doutor Flavio A.M. Cipparrone

Email:

- flavio@lps.usp.br
- Sala: D2-21 Ramal: 5132

Departamento: PSI – Sistemas Integráveis

São Paulo

2009

RESUMO

O trabalho de formatura que será desenvolvido consiste em projetar e implementar um filtro digital que irá separar as freqüências, vindas de uma fonte de áudio, em altas, médias e baixas e enviar cada um desses sinais obtidos a alto-falantes dedicados para tais freqüências.

O filtro digital desejado será dedicado a uma caixa acústica omni-direcional que está sendo desenvolvida para obter uma melhor resposta que as caixas seladas convencionais.

Pretende-se ao final do prazo determinado ter-se desenvolvido um programa em linguagem computacional adequada que implemente esse filtro digital, bem como realizar estudos para a melhor performance da caixa mencionada e utilizar como ferramenta um hardware adequado para lidar com o sinal de uma fonte conveniente.

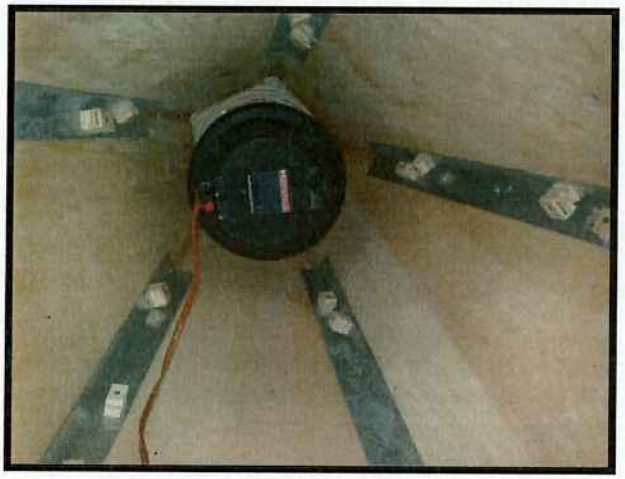


Foto do driver de médios

Foto do falante de Graves

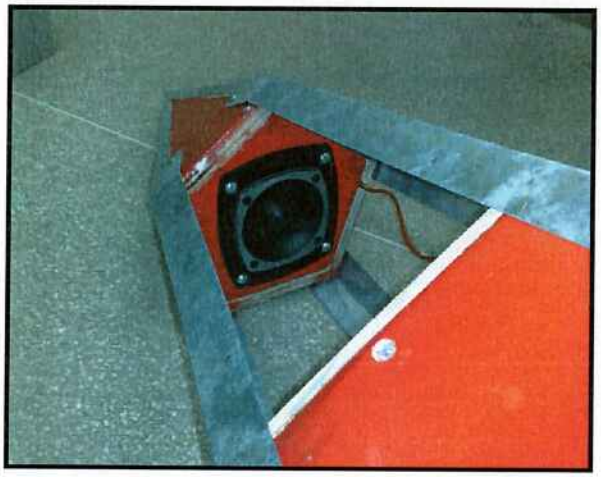


Foto do falante de agudos (tweeter)

Foto da caixa acústica Omni-direcional

SUMÁRIO

1 INTRODUÇÃO -1

2 JUSTIFICATIVA- -4

3 OBJETIVOS DO PROJETO- -5
3.1 Objetivos gerais- -5
3.2 Objetivos Específicos- 5

4 METODOLOGIA- -6

5 METAS- 7

6 CRONOGRAMA TF 1- 8

7 CRONOGRAMA TF 2- 8

8 RECURSOS NECESSÁRIOS- 9

9 EXPOSIÇÃO TEÓRICA -14
9.1 Filtros IIR e FIR -- 14
9.2 Síntese de filtros Digitais -15

10 DESENVOLVIMENTO -17
10.1 O projeto do filtro no Matlab -- 17
10. 2 Definição do filtro digital utilizado - - - - - - - - - - - - - - - - - - 17

10.3 A ferramenta Design Filter do MatLab - - - - - - - - - - - - - - 18
10.4 Ensaio dos Filtros no Matlab - 23
10.5 Medições dos parâmetros da caixa e seu aprimoramento - - 27
10.6 A programação no DSP -35
10.7 A implementação do filtro peaking e as medições finais - - -38

11 CONCLUSÃO - 41

12 REFERÊNCIAS - 43

13 BIBLIOGRAFIA -44

ANEXOS -45

1 INTRODUÇÃO

Na eletrônica, ciências computacionais e na matemática, um filtro digital é um sistema que realiza operações matemáticas em uma amostra de sinal de tempo discreto para reduzir ou melhorar certo aspecto desse sinal. (Wikipédia, digital filters).

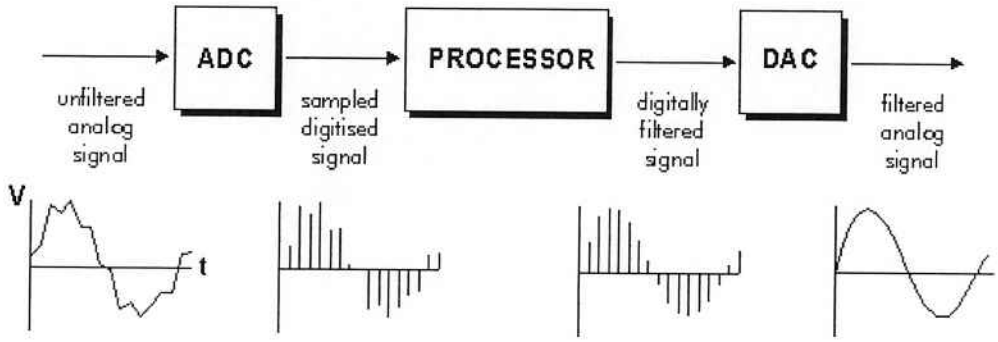


Figura1:Filtro Digital Aplicado

Atualmente vemos, no mundo do áudio, uma grande influência da eletrônica. Ela está imersa dentro de muitos aparelhos e produções de modo que muito do que é feito hoje não seria possível sem seu auxílio. Nem sempre foi assim, no começo do século XX as gravações ainda eram feitas usando gramofones, com uma qualidade pouco fiel e sofrível, sem contar que a música e o fascínio por áudio era, na época, luxo, possível apenas para poucas pessoas. Fazer uma gravação ou escutar uma música era muito caro e, assim, muitos artistas e compositores, por mais competentes e talentosos que fossem, ficavam à margem do mundo musical por falta de dinheiro. Com o passar dos anos e com o desenvolvimento da eletrônica a situação melhorou substancialmente. O próprio disco de vinil tão antigo conseguiu uma qualidade muito superior de modo que até hoje muitos o preferem ao CD.

A eletrônica está também hoje, mais do que nunca, imersa em aparelhos. Amplificadores por exemplo são um belíssimo uso da eletrônica. Os melhores mantêm até hoje o uso de válvulas ao invés de transistores para as fases de pré e pós-amplificação conseguindo assim uma qualidade sonora superior.

Outro aspecto que melhorou muito foi à acessibilidade do mundo musical e do áudio. Para se gravar uma música hoje basta possuir um computador, alguns

microfones, uma placa de som de boa qualidade e alguns periféricos. Claro que o conhecimento sobre o assunto e a qualidade dos aparelhos usados fará ainda uma grande diferença, mas comparado ao que era há 80 anos hoje se grava um CD de forma muito mais rápida e barata e com uma qualidade muito superior do que as gravações do inicio do século. O desenvolvimento da internet e do MP3 foi também um grande passo para a melhoria da acessibilidade. Hoje se consegue escutar um CD na mesma semana em que ele é lançado, artistas disponibilizam suas músicas abertamente em sites fazendo assim com que esse mundo da música esteja aberto a um número irrestrito de pessoas.

O uso de filtros permeia esse mundo, sendo usado extensamente em gravações, equalizações, mesas de som, televisores, sons automotivos, entre muitas outras aplicações. Seu uso é indispensável e torna a qualidade sonora muito superior, pois faz com que se possa adequar o som da maneira que se deseja.

Um filtro digital pode ser resumido em três blocos principais: Um bloco de conversão analógico-digital, onde se transforma o sinal de forma a poder trabalhar com ele em forma digital; Tal sinal passa a ser de tempo discreto, ou seja, múltiplas amostras no tempo.

O segundo bloco seria o do processador propriamente dito, onde os cálculos são realizados e a matemática do processo acontece. É comum esse processador ser dedicado para a finalidade que foi projetada o filtro.

O último bloco é um espelho do primeiro e é designado por digital-analógico. Depois de ter o sinal desejado o sistema precisa "devolver" a resposta calculada. A transformação do meio digital de volta para o analógico acontece nesse momento.

Filtros digitais são mais comuns do que se pensa, e estão em diferentes sistemas eletrônicos do nosso cotidiano, como celulares e aparelhos de reprodução sonora.

Com vantagens como flexibilidade, robustez em relação ao tempo e a temperatura, facilidade de implementação entre outros aspectos, os filtros digitais vêm tomando o lugar dos analógicos nos últimos tempos, e acompanharmos essa mudança ajudará a nos manter dentro das inovações tecnológicas na área da eletrônica.

Diante desse contexto criou-se a idéia do filtro digital para uma caixa acústica especial. A falta de uma tecnologia em larga escala na área, junto com a possibilidade de por em prática o que aprendemos em disciplinas anteriores foram fatores decisivos para dar inicio a esse projeto, em que com conceitos elementares de eletrônica será proposta uma idéia inovadora de projeto na área de filtros digitais.

2 JUSTIFICATIVA

A idéia do projeto começou pelo interesse da dupla no setor de áudio e a vontade de desenvolver o trabalho de formatura dentro dessa área, que está em crescente desenvolvimento tecnológico. A constante melhoria na qualidade sonora foi um fator decisivo para criar um projeto na área musical, objetivando a criação de um sistema dedicado, a fidelidade sonora e por fim a criação de trabalho inovador, pois atualmente a parte de filtros para sinais de áudio são dominados por componentes analógicos.

Alem disso, em um âmbito mais geral, acredita-se que com esse trabalho pode ser concluído o curso de engenharia pondo em prática grande parte das disciplinas estudadas, como eletrônica, programação, filtros digitais, sistemas de sinais, entre outras.

Assim, visando os anseios e expectativas da dupla foi escolhido o professor Dr. Flávio Cipparrone para a orientação do trabalho, que sugeriu um projeto de interesse de ambos os lados.

3 OBJETIVOS DO PROJETO

3.1 Objetivos Gerais:

O objetivo geral do projeto é o desenvolvimento do filtro digital com crossover que fará a separação das freqüências em altas, médias e baixas com o auxílio do software MatLab e a implementação dele através do uso de um processador digital de sinais (DSP), que será programado por um software familiarizado com linguagem C. Além disso, enviar essas freqüências a auto-falantes dedicados que estarão montados na caixa acústica omni-direcional, obtendo assim uma qualidade sonora superior.

3.2 Objetivos Específicos:

-Estudo dos filtros digitais para aplicações específicas de sinais sonoros.

-Levantamento das características físicas da caixa omni-direcional e seus alto-falantes.

-Projeto do filtro no Matlab:

Crossover é o termo que significa a faixa de freqüências em que se tem a intersecção entre o término de um filtro e o começo de outro. Neste projeto teremos dois crossovers, um para a intersecção entre o término do filtro passa-baixas e início do filtro passa-faixa e outro para a intersecção entre o término do filtro passa-faixa e início do filtro passa-altas. Um objetivo específico do projeto é conseguir, através do projeto no Matlab, uma faixa de crossover bem curta, atingindo assim uma perda de ganho muito baixa em uma faixa bem curta de freqüências, deixando assim a resposta sonora mais uniforme, ou seja, sem diferenças de ganho entre as diferentes faixas de freqüências.

-Aquisição do DSP para programação do filtro.

-Instalação do DSP.

-Estudo da linguagem de programação específica do DSP.

-Estudo do funcionamento do DSP.

-Programação do Filtro no DSP.

-Testes e Implementação na Caixa Acústica.

-Respostas e curvas de Impedância e Fase Mínima

-Ajustes e melhorias dos resultados.

4 METODOLOGIA

Como uma primeira parte foi feita à leitura de livros para o aprendizado do assunto.

Em seguida, dando continuidade ao cronograma apresentado, o foco de estudo principal foi o software e o hardware utilizados na programação do filtro digital. O programa final foi compilado no DSP. O aparelho consiste de um hardware que executa o que é implementado por um software. Em outras palavras, depois de projetado o filtro no Matlab, utilizou se o compilador que já vem no dispositivo. O mesmo trabalha em linguagem C e através do algoritmo fez-se assim a separação de freqüências do sinal digitalizado.

O estudo foi a fim de garantir o entendimento do funcionamento da programação do DSP. Esse DSP (fornecido pelo professor Coelho que também auxiliou no projeto) fabricado pela Analog Devices modelo Blackfin, não era a princípio a placa a ser utilizada, pois aparentemente a mesma não é a mais indicada para o trabalho. Isso se deve pois ela utiliza aritmética de ponto fixo para fazer as convoluções necessárias para se implementar o filtro digital, metodologia essa mais imprecisa e mais difícil de ser programada .Cogitou-se o uso de outro DSP similar, porém que utilizasse como aritmética o ponto flutuante, mais preciso e mais fácil de ser programada. Tal DSP (também produzido pela Analog Devices, modelo Sharc) possui uma interface mais simples e arquitetura voltada a projetos de filtros.

Inicialmente uma tentativa de uso foi feita na placa já obtida. A leitura de manuais além de pesquisas em internet sobre o assunto deram um auxílio no entendimento de seu funcionamento, e ficou evidente a possibilidade de se projetar o filtro com o mesmo. No entanto a complexidade de se trabalhar com o modelo é maior.

A decisão de utilizar o BlackFin foi concretizada após dificuldades burocráticas e orçamentárias vindas dos fornecedores.

Definido tais parâmetros, o projeto desenvolveu-se semanalmente com a leitura de manuais do MatLab, consultas a livros sobre o assunto, manuais do DSP Blackfin, projetos de filtros no_MatLab, estudos sobre filtros FIR e IIR, levantamento dos parâmetros da caixa, implementação, testes e pesquisa e leitura de material obtido na internet.

5 METAS

Foram definidas as seguintes metas ao longo do projeto:

- Estudo de filtros digitais e suas aplicações nos aparelhos de hoje.
- Comparação e definição de qual DSP utilizar, bem como suas vantagens e desvantagens.
- Obtenção do melhor filtro digital possível, com maior ganho e menores distorções nas áreas de transição de freqüência, assim como nas extremidades.
- Garantir o melhor desempenho da caixa acústica omni-direcional.
- Obtenção dos parâmetros de resposta da caixa para possibilitar o projeto do filtro.
- Obter com a maior precisão possível a separação das freqüências sonoras.
- Programar em linguagem adequada o filtro digital para gravação no DSP.
- Construção de todos os módulos envolvidos no projeto, isto é, garantir que todas as partes funcionem não só separadas como também em conjunto.
- Fazer a correta amplificação dos sinais já separados, de forma a obter uniformidade sonora.

6 CRONOGRAMA TF 1

	Fevereiro	Março	Abril	Maio	Junho	Julho
Definição do projeto	X					
Estudo de viabilidade inicial	X	X				
Definição do DSP			X	X		
Estudo teórico de filtros digitais	X	X	X	X		
Estudo de viabilidade final				X	X	X

7 CRONOGRAMA TF 2

	Agosto	Setembro	Outubro	Novembro	Dezembro
Levantamento dos parâmetros da caixa			X		
Estudo da placa de DSP		X	X	X	
Estudo da interface do DSP				X	
Projeto do filtro digital teórico (MatLab)	X	X	X		
Casamento do filtro teórico com os parâmetros da caixa			X	X	
Programação do DSP			X	X	
Implementação (Caixa + Filtro digital)				X	X
Testes e Otimização				X	X

8 RECURSOS NECESSÁRIOS

Foram necessários para a realização deste projeto de formatura alguns equipamentos, sendo que boa parte deles estavam à disposição na escola. Para o correto projeto do filtro levou-se em conta as características dos alto-falantes e da caixa-acústica. Para isso utilizou-se softwares e microfones específicos, que mediram as respostas em freqüências de cada um dos alto-falantes dedicados. Inicialmente acreditava-se que essas medições seriam feitas algumas na própria sala do orientador, onde se encontra o computador com o software, e outra em salas anecóicas, que são salas onde não há reflexão sonora, para se conseguir assim uma medição mais precisa dos alto-falantes.

Na impossibilidade de se usar a sala anecóica utilizou-se o estacionamento da engenharia elétrica em um dia sem movimento e sem ruídos, minimizando assim a interferência sonora. Indiretamente mediu-se também os parâmetros Thiele/Small que são parâmetros que definem o comportamento de um alto-falante. Com eles pode-se projetar muito mais facilmente as caixas acústicas. Antigamente usavam-se métodos mecânicos para se obter tais parâmetros, porém eram mais imprecisos e difíceis de serem aplicados. Esses parâmetros são de extrema importância para o projeto do filtro, pois são eles que definem alguns de seus coeficientes através da resposta da caixa acústica.

Com as características da caixa e dos falantes definidas, foram necessários computadores, softwares e o DSP para se projetar e implementar o filtro digital. Os computadores ou estavam disponíveis na escola ou utilizou-se os próprios computadores dos integrantes do grupo.

Utilizou-se o Matlab para o projeto e desenvolvimento do filtro e o software específico do DSP. O Matlab estava à disposição na escola e na internet. Já o software específico do DSP vem em conjunto com o hardware, e como o DSP foi disponibilizado pelo professor Luiz Coelho, não houve problemas nesse sentido.

Anteriormente houve uma discussão no projeto de como seria feita a amplificação dos sinais. Duas opções foram analisadas: Podia-se usar o filtro digital para fazer uma separação simples entre freqüências baixas e freqüências médias e altas e posteriormente usar apenas dois amplificadores, um para freqüências baixas e

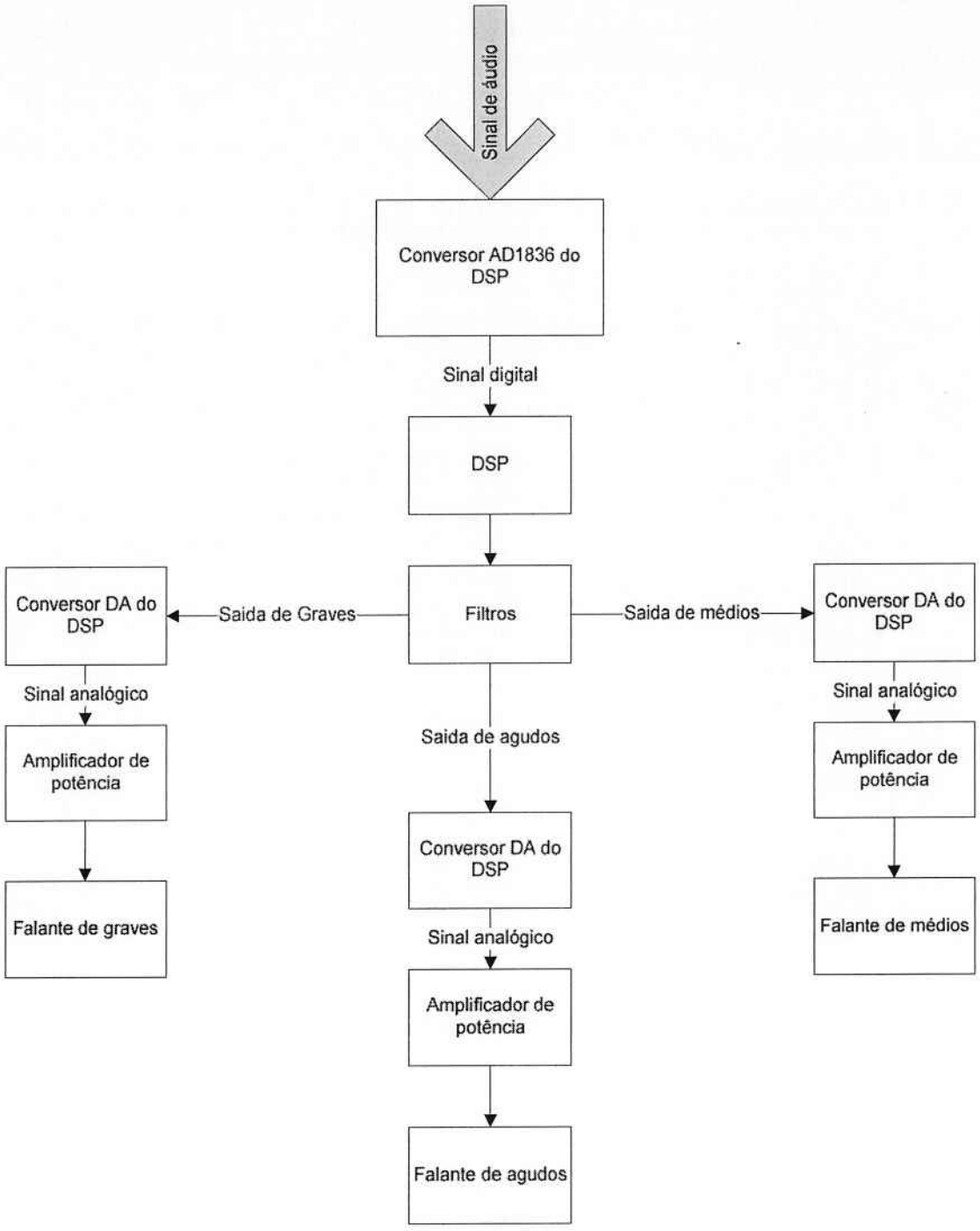
um para as outras duas e finalmente usar um filtro passivo para separar as médias das altas.

Outra opção seria usar três amplificadores e apenas o filtro digital separando as três faixas de freqüências, e depois amplificar cada uma separadamente e enviar aos alto-falantes.

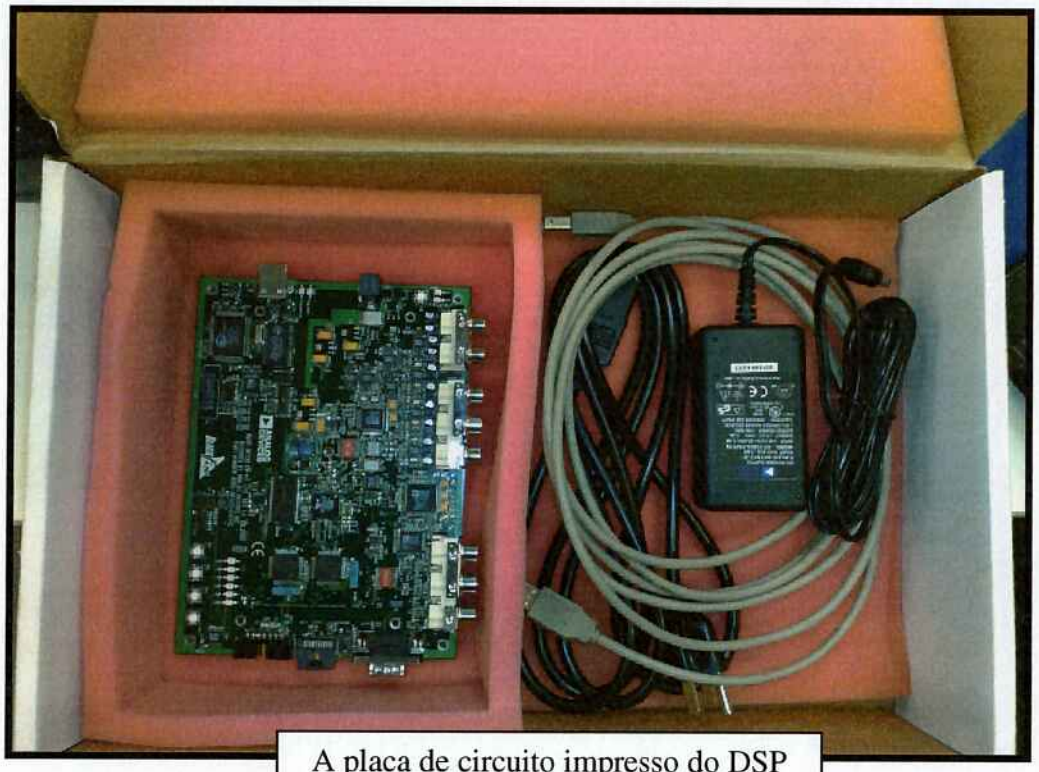
O professor Flavio Cipparrone possui dois amplificadores, logo a primeira opção poderia ter sido aplicada desde o inicio, porém havendo a possibilidade de aplicar a segunda também seria interessante por se fazer uso apenas do filtro digital. Olhando por outro lado, ao se desenvolver a primeira opção seria posto em prática o projeto de filtros analógicos que também foi estudado na escola e assim ter-se ia uma visão geral dos dois mundos de filtros, digital e analógico.

As duas propostas foram estudadas e levando em consideração os parâmetros acima exemplificados foi decidido usar três amplificadores.

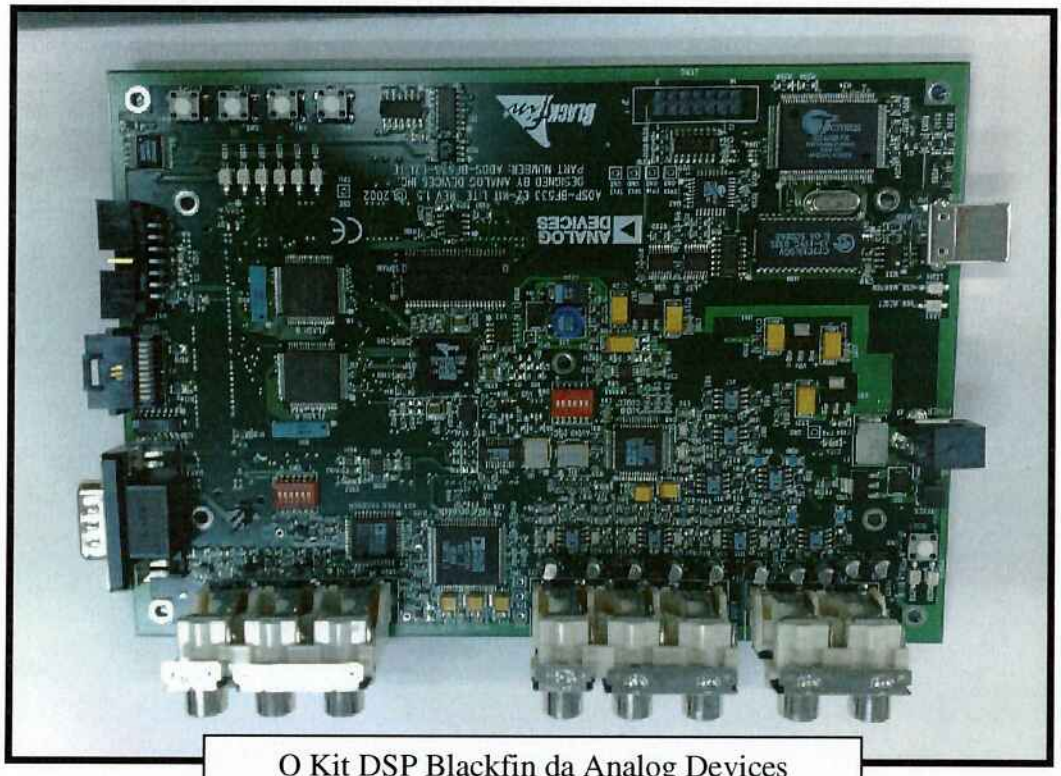
Outro tópico que foi discutido foi à opção de se usar outro DSP. A placa de hardware Blackfin, fornecida pelo professor Luiz Coelho, a princípio não foi a mais indicada para o assunto, conseqüentemente poderia se perder algum tempo com assuntos que não eram do escopo principal do projeto. Por isso foi feito um orçamento de outro DSP, o Sharc da Analog Devices, que poderia facilitar a construção do filtro. Dado a demora do orçamento, descrita anteriormente na metodologia, além do tempo de importação inviável e a verba destinada à disciplina limitada, foi decidido usar o DSP Blackfin que apesar de um pouco mais complexo conseguiu cumprir suficientemente os objetivos do projeto. Abaixo está um fluxograma da idéia geral do projeto seguido de algumas fotos do DSP.



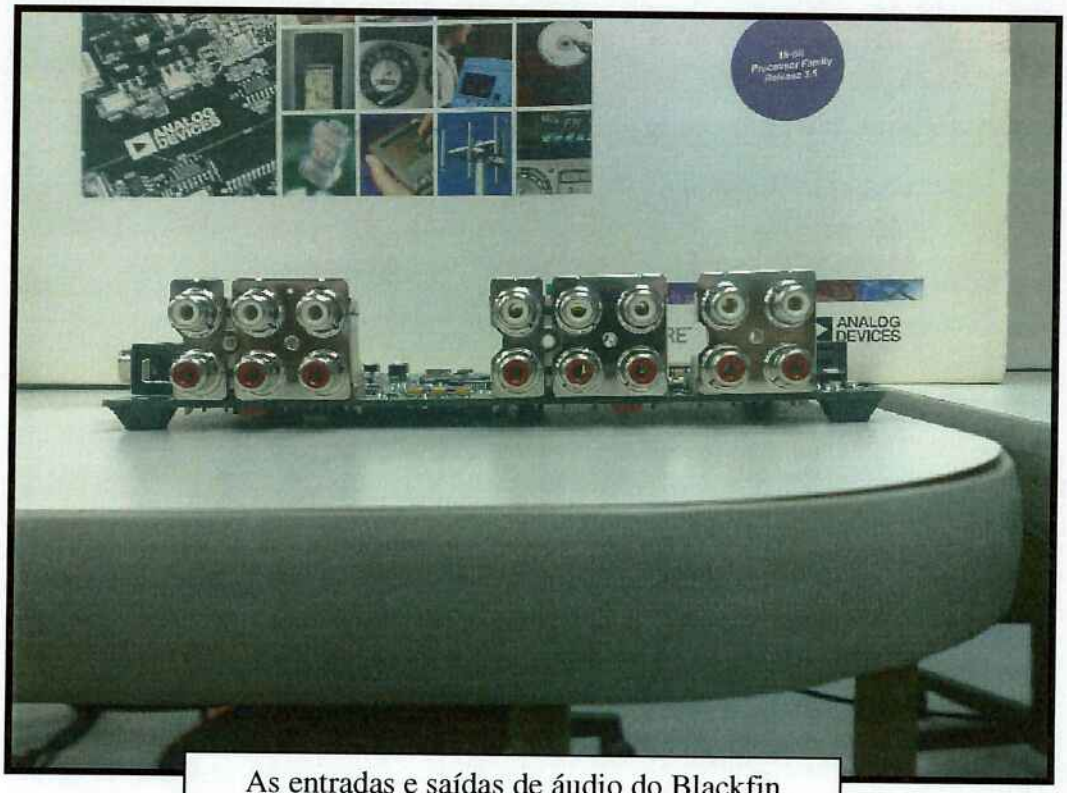
Caminho percorrido pelo sinal, desde a entrada no DSP até a reprodução nas caixas acústicas



A placa de circuito impresso do DSP



O Kit DSP Blackfin da Analog Devices



As entradas e saídas de áudio do Blackfin

9 EXPOSIÇÃO TEÓRICA

9.1 Filtros IIR e FIR

Como dito anteriormente, um filtro digital nada mais é do que um manipulador de amostras de um sinal no tempo discreto. São exemplos de filtro digitais: De ganho unitário, de ganho simples, de atraso, de diferença de dois termos, filtro de média ponderada, e de media aritmética.

Nos filtros digitais há duas formas básicas de construção, os filtros IIR e os filtros FIR.

Um filtro que depende apenas de entradas no tempo presente, não levando em conta saídas passadas do mesmo, é chamado de filtro FIR. Outra forma de traduzir o significado do filtro FIR é exatamente sua definição, a resposta ao impulso finita. Como características principais pode-se dizer que tais filtros são estáveis, por outro lado são de ordem muito maior se comparados com os de tipo IIR para a mesma resposta desejada.

Abaixo temos a definição matemática para filtros FIR, bem como sua transformada z:

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k] \Leftrightarrow T(z) = \sum_{k=0}^{M} a_k \cdot z^{-k}$$

Onde a saída do filtro y(n) só depende da somatória das multiplicações dos coeficientes do filtro Bk pelas entradas do mesmo x(n).

Ainda em relação à resposta ao impulso, um filtro que tenha como característica principal à realimentação, ou seja, se a saída do filtro depende em maior ou menor grau dos valores de suas amostras passadas, esse tipo de filtro é denominado IIR.

Abaixo temos a definição matemática para filtros IIR, bem como sua transformada z:

$$y[n] = \sum_{k=0}^{M} a_k \cdot x[n-k] + \sum_{k=1}^{N} b_k \cdot y[n-k] \Leftrightarrow T(z) = \frac{\sum_{k=0}^{M} a_k \cdot z^{-k}}{1 - \sum_{k=1}^{N} b_k \cdot z^{-k}}$$

Repare que agora além das amostras de entrada do filtro a saída é determinada também pelas saídas (y(n)) anteriores do mesmo.

Filtros IIR são normalmente preferidos em relação aos filtros FIR quando se diz respeito a analise de sons, pois em filtros recursivos há uma faixa de transição na resposta em freqüência muito mais estreita, quando comparados filtros de mesma ordem. No entanto há outros parâmetros a serem levados em conta quando se quer definir o tipo de filtro a ser utilizado, como poder de processamento e memória do dispositivo sobre o qual esta se trabalhando.

Uma definição citada anteriormente em relação aos filtros digitais é a sua ordem. A ordem de um filtro digital é dada pelo número de entradas anteriores (guardadas na memória do processador) usadas para calcular a saída atual. No caso de filtros recursivos, essa ordem pode ser definida como o maior número de entradas anteriores (tenham sido elas saídas ou entradas do filtro) necessárias para se obter à saída atual.

Os coeficientes do filtro digital são os fatores multiplicativos das entradas do filtro, sejam elas saídas anteriores ou novas entradas. São eles que dão a forma desejada para o filtro digital que será projetado.

Para o caso do projeto, os coeficientes do filtro digital serviram para modificar o filtro de forma a ficar o mais parecido possível com a resposta da caixa acústica medida.

9.2 Síntese de filtros digitais

A síntese de filtros digitais faz-se usando vários métodos, sendo estes diferentes para filtros IIR e FIR.

Os coeficientes dos filtros IIR podem ser obtidos através das características dos filtros analógicos correspondentes segundo vários métodos, como por exemplo, a conservação da resposta impulsiva, a transformação bilinear e a localização de pólos e zeros da função de transferência. Porém, embora simples, este último método não é indicado para filtros mais complexos, devido aos efeitos dos erros numéricos na localização de pólos e zeros. O método mais usual é o método da transformação bilinear.

Já os filtros FIR não podem ser obtidos pelas aproximações utilizadas em sistemas contínuos, dado que não existem sistemas analógicos com resposta impulsiva finita. A resposta impulsiva dos filtros FIR foi definida anteriormente. Para a síntese de filtros FIR usam-se, geralmente, o método dos mínimos quadrados ou o método das janelas, para truncatura da resposta impulsiva.

10 DESENVOLVIMENTO

10.1 O projeto do Filtro no MatLab:

Para se projetar o filtro corretamente no MatLab, a primeira discussão que deve ser feita é qual filtro usar e porque. O principal é definir o método de design e dentro desse método específico discutir as características do mesmo. Por exemplo, um método de design muito usado é o método do janelamento e dentro desse método existem alguns tipos de janelas, as quais podem ser aplicadas dependendo do que se quer, ou seja, define-se o filtro e suas características com base no que se deseja.

Para o caso de um filtro de áudio não é diferente, podem-se usar muitos métodos para se projetar o filtro como se queira. No caso do projeto se quer uma resposta em freqüência plana e linear, com uma queda bem acentuada próximo das freqüências de corte para se ter assim um crossover que não prejudique nas freqüências de transição dos filtros, reduzindo assim seus ganhos.

10.2 Definição do filtro digital utilizado:

Para o projeto foi usado um filtro do tipo IIR (Infinite Impulse Response), o qual apesar de poder vir a ter instabilidade e uma resposta menos linear em termos de fase comparado com o outro tipo de filtro FIR (Finite Impulse Response), foi preferido. Foi usada uma quantidade de coeficientes adequada, a qual controla a instabilidade não desejada, e quanto à fase não ser linear não será um problema, pois o ouvido humano não percebe tal comportamento. Outros parâmetros devem ser levados em conta:

- A limitação quanto ao processamento do DSP a ser utilizado. Será discutido a seguir as características de desempenho do BlackFin; no entanto como adiantamento para justificativa vale ressaltar que o mesmo trabalha em freqüência de 27kHz, fator determinante para o número de coeficientes e que tipo de operações matemáticas o algoritmo do filtro digital consegue realizar.
- Precisão dos coeficientes. A placa da Analog Devices escolhida trabalha com ponto fixo, fato que limita a precisão dos números que estão sendo utilizados. Apesar de tal inconveniente atrapalhar nos dois

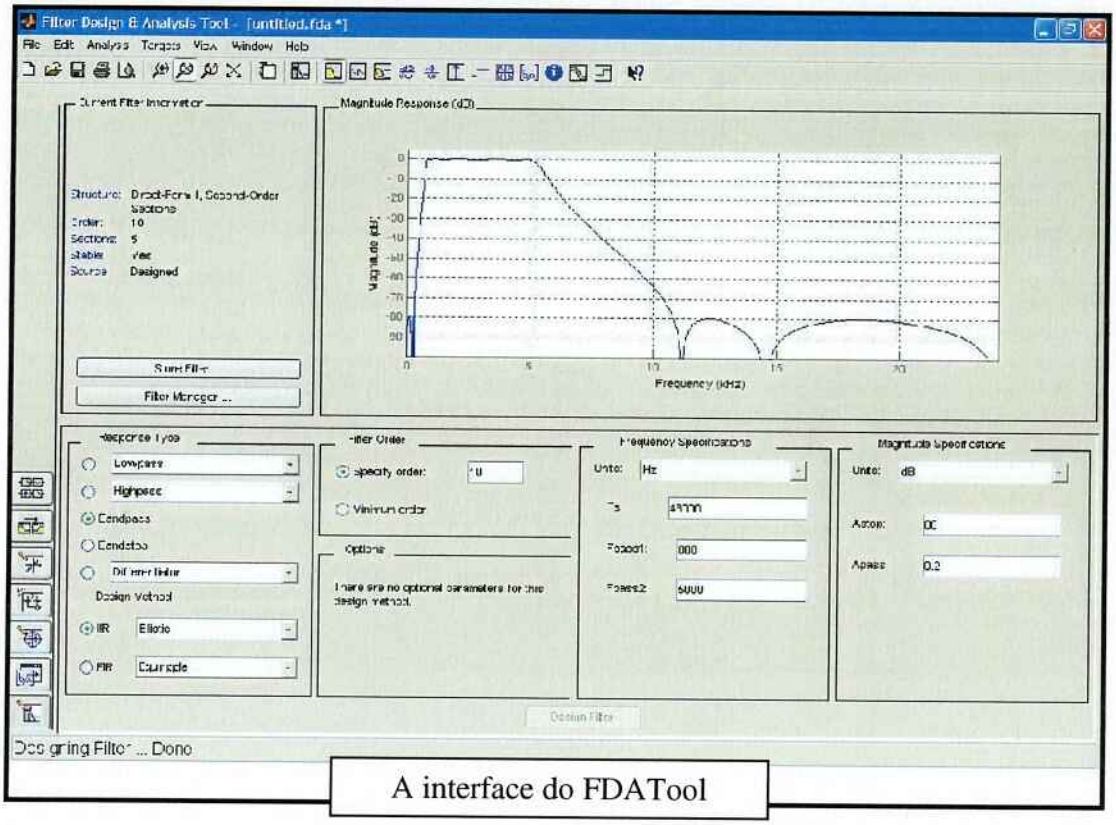
modelos, o fato de que para um filtro IIR ser necessário poucos coeficientes foi um fator decisivo para a escolha do mesmo.

Vale ressaltar que o cálculo dos coeficientes do filtro foi feito por software especifico, portanto o projeto matemático do mesmo, independente de qual fosse escolhido, não foi parâmetro de decisão, por ser feito automaticamente.

10.3 A ferramenta Design Filter do MatLab:

Para a construção teórica do filtro digital utilizado foi usado a ferramenta especifica de construção de filtros digitais do MatLab, o Design Filter Tool Box.

A seguir como exemplo interface do software:



A interface do FDATool

A ferramenta Design Filter do Matlab permite que o projetista determine diversas características desejadas.

Podem ser citadas como mais importantes para o projeto, e que foram levadas em consideração para a criação do filtro, as seguintes características:

- Tipo de filtro: IIR ou FIR. Para o projeto, como justificado anteriormente, foi utilizado o filtro IIR.

- Tipo de janelamento: Para a construção dos filtros desejados o Matlab utiliza o método do janelamento, para ambos os casos. Como exemplo são opções para o projetista a janela de Butterworth, Chebyshev tipos I e II, Elíptico (usado para o projeto), entre outros.
- Tipo de filtro. Para o projeto será usado o filtro passa baixa, passa altas e passa banda.
- Freqüência de amostragem. A ferramenta de desenvolvimento de filtros digitais do Matlab é genérica, ou seja, pode ser utilizada em qualquer modelo. Para o nosso caso a aplicação foi feita para o DSP BlackFin, que trabalha na freqüência de 48kHz.
- Freqüências de corte do filtro. Uma vez definido o tipo de filtro que será utilizado definem-se as características de resposta em freqüência do mesmo. No caso do filtro passa banda o software automaticamente mostra os campos de freqüência de corte inferior e superior para o projetista.
- Oscilações na banda de passagem e na banda de rejeição. Pode-se definir em dB tais parâmetros. Apesar de matematicamente o Matlab desenvolver quase que qualquer tipo de característica nesse sentido, e, além disso, fica claro que freqüências abaixo de -70 dB são desejadas para o projeto para a banda de rejeição, os parâmetros definidos nesse trecho da ferramenta podem afetar significativamente a aplicação real do filtro, tornado viável ou não seu processamento.

Esses são os parâmetros principais de criação do filtro do projeto.

A ferramenta Design Filter, além disso, cria o filtro desejado através de diferentes métodos. Para o projeto foi posto em questão duas formas, a forma direta I e II.

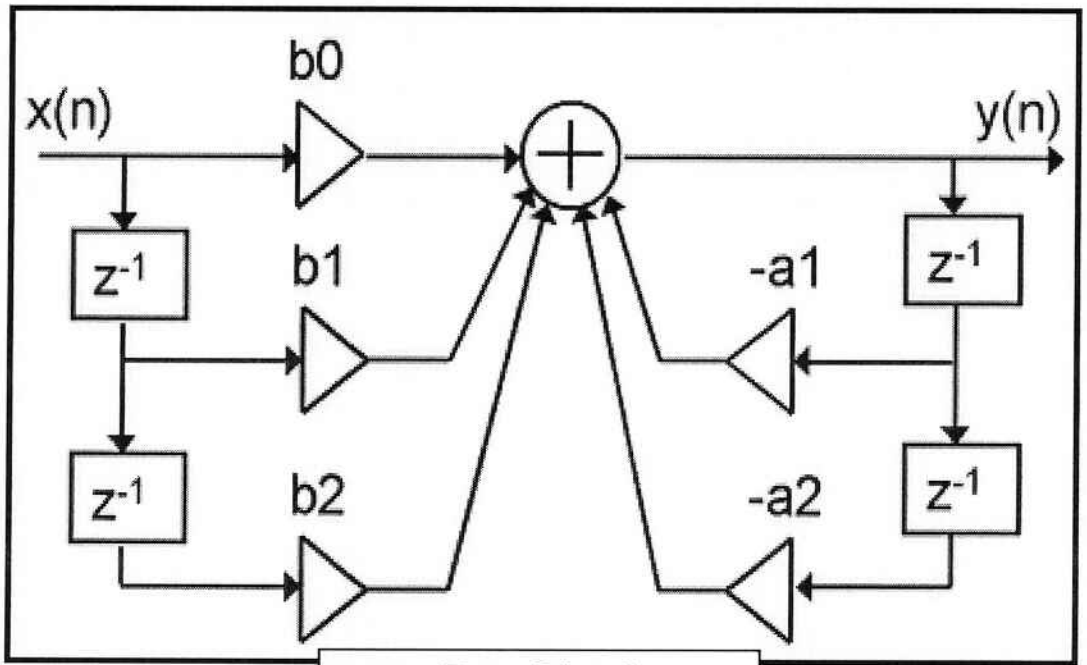
Dentro do Design Filter as duas formas citadas funcionam de forma muito parecida. Como padrão o software define o filtro projetado em blocos de segunda ordem, independentemente do número de coeficientes desejados. Assim, para um filtro de ordem superior a dois coloca-se vários blocos menores em cascata para obter a resposta desejada.

$$H(z) = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 + (A_1 z^{-1}) + (A_2 z^{-2})}$$

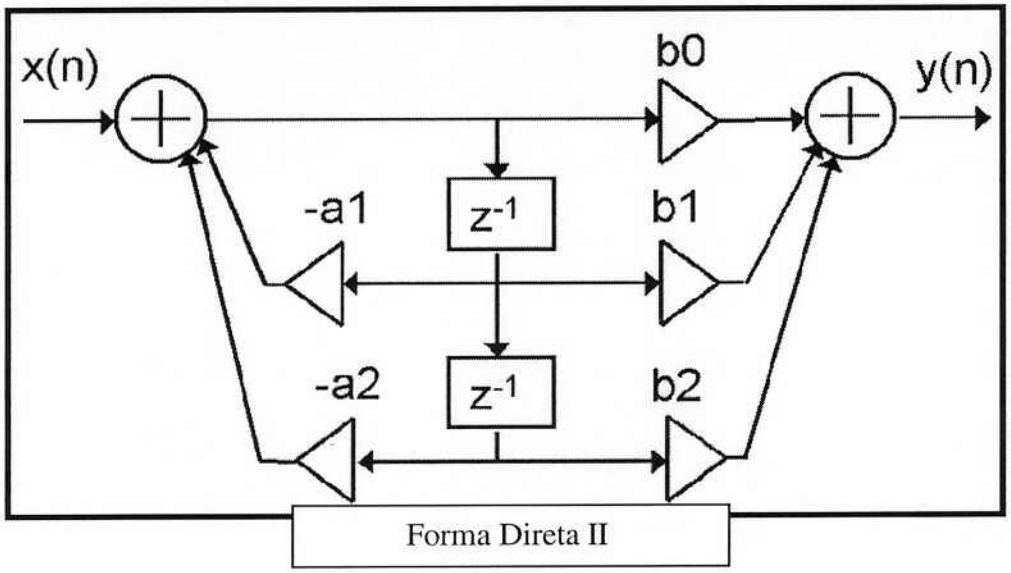
Bloco de segunda ordem

Voltando para o tipo de formas de se implementar o filtro, a diferença entre as formas diretas I e II esta no número de atrasadores e na quantidade de operações aritméticas realizadas. Por um lado, na DFI usa-se um número de atrasadores que é o dobro da DFII, dada a mesma quantidade de coeficientes do filtro. Por outro lado, na DFII a quantidade de operações aritméticas é maior. Assim, fez-se uma análise para ver qual dos dois métodos seria mais interessante.

Abaixo está a ilustração dos dois métodos:



Forma Direta I



Diante desse escopo foi determinada a forma direta II no algoritmo desenvolvido. A grande desvantagem da forma direta I é que se torna impraticável a construção desse tipo de filtro com um número de coeficientes grandes (filtros de grande porte).

Um problema da DFII é que, por possuir um número maior de operações matemáticas, corre-se o risco de overflow. No entanto para a placa utilizada, depois de alguns testes sonoros, constatou-se que não haveria problemas.

Os diferentes métodos de design:

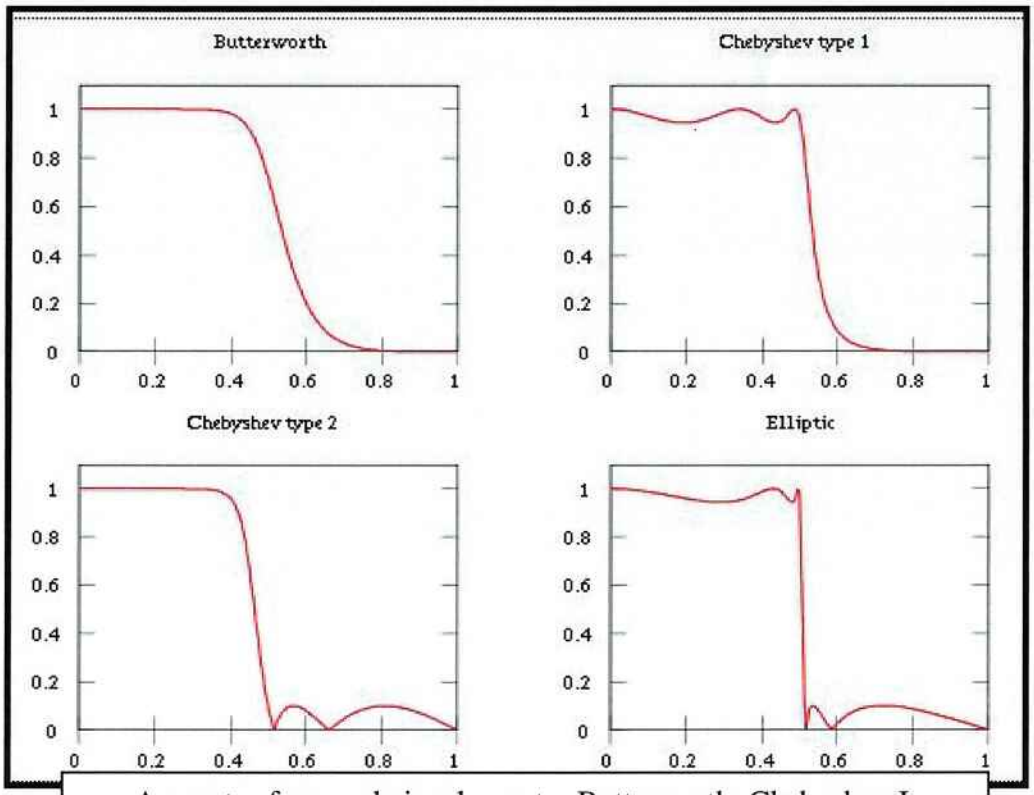
Para efeito de projeto, foram quatro as opções de design que poderiam implementar o filtro, Chebyshev I e II, Butterworth e Elíptico. Cada um deles possui uma característica que é favorável ou não dependendo da aplicação:

Butterworth – Como característica principal, o método de Butterworth possui a resposta mais plana dos quatro tipos de design. Em compensação é o que tem a faixa de transição entre a banda passante e de rejeição mais longa.

Elíptico – Já no tipo de design elíptico a faixa de transição é a mais curta dos quatros modelos. Como conseqüência há uma pequena oscilação na faixa próxima a transição de bandas.

Chebyshev tipo I e II. – Seria a mistura dos dois modelos. Os filtros Chebyshev possuem faixa de transição menor que a do Butterwoth e oscilação menor que os filtros elípticos na banda de passagem. Como observação, a diferença entre os dois tipos (I e II) está na região onde há uma pequena oscilação nas bandas do filtro. No caso do tipo I, a oscilação está na banda de passagem. Já no caso da II, inversamente, a oscilação está na banda de rejeição.

A seguir uma ilustração das quatro formas de janelamento estudadas:



As quatro formas de janelamento: Butterworth, Chebyshev I, Chebyshev II, Elíptico.

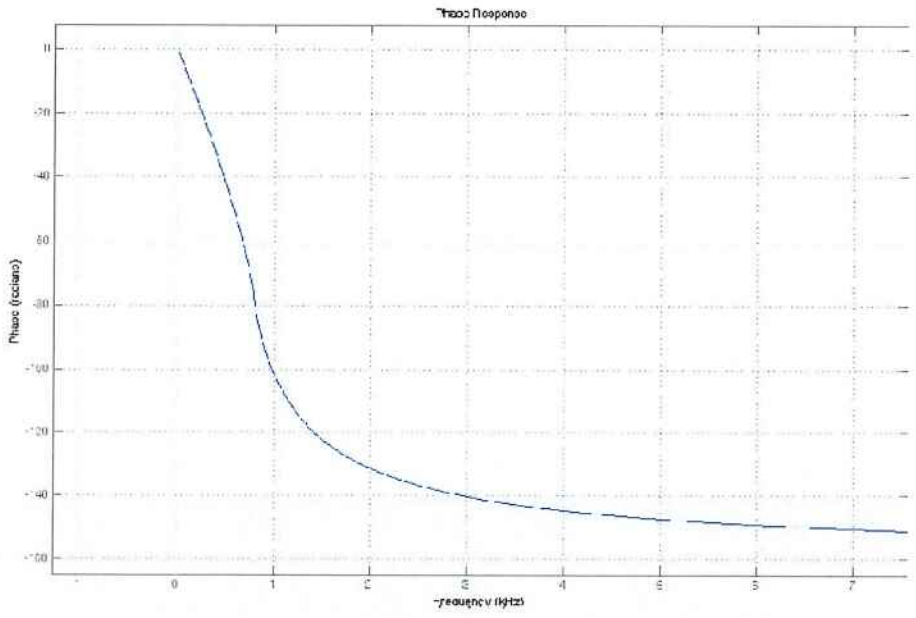
Diante desses fatos, e lembrando os objetivos do projeto, optou-se pelo método de design elíptico. Dado que o principal parâmetro a ser melhorado no projeto era a faixa de transição dos filtros, e que a oscilação na banda de passagem

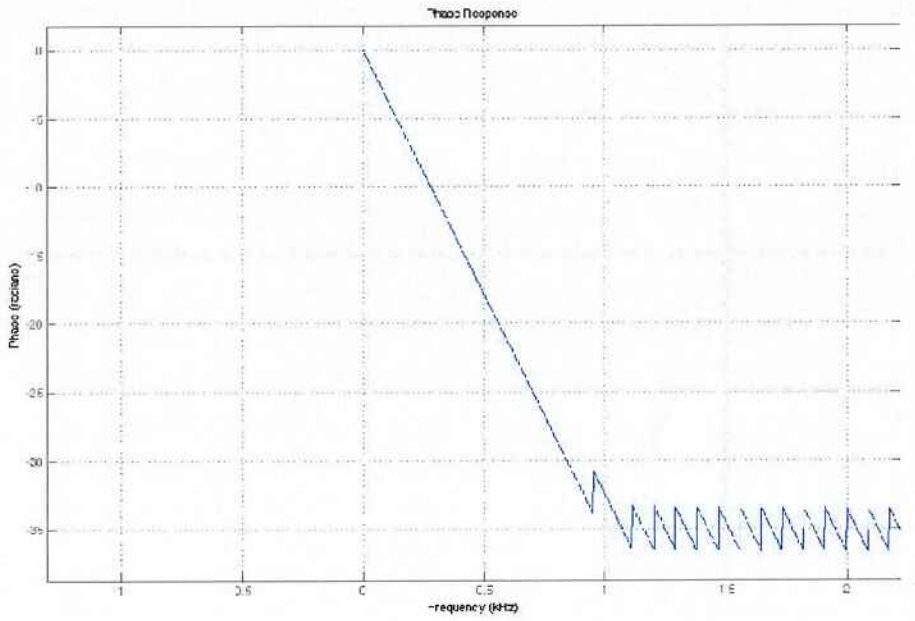
consegue ser contornada pelo ToolBox do Matlab com certa facilidade através dos coeficientes, foi escolhido esse modelo para o projeto do filtro digital.

10.4 Ensaio dos Filtros no Matlab

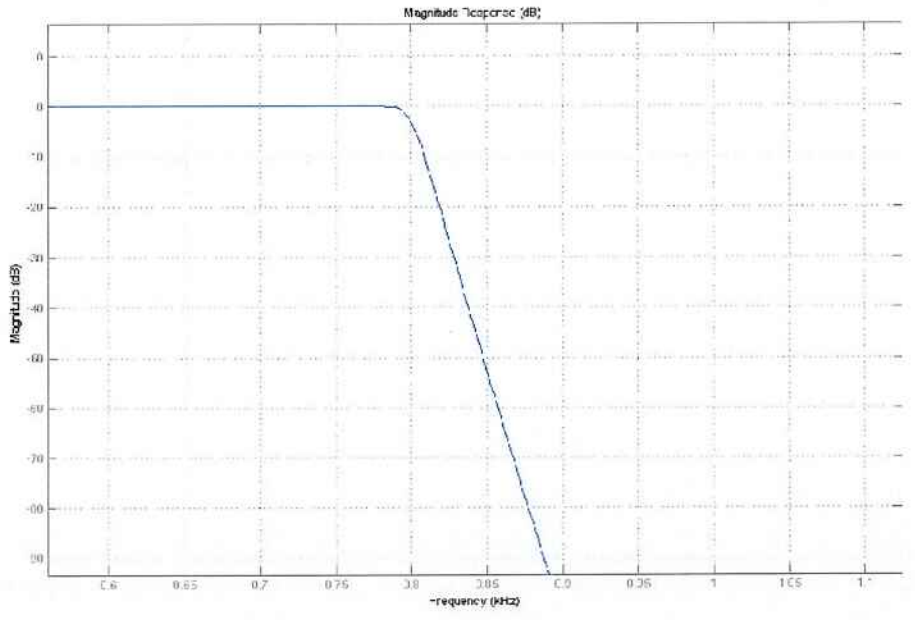
Abaixo se pode ver o principal motivo pelo qual não foram usados filtros FIR. Vemos que a sua resposta comparada com a reposta de um filtro IIR é muito inferior e utiliza muitos coeficientes a mais, o que traria problemas na fase da computação e das convoluções que o computador precisaria fazer, pois ele teria que realizar todas as convoluções dentro do intervalo de tempo determinado pela freqüência de amostragem, que no projeto será de 48000 Hz, e com muitos coeficientes isso se tornaria mais complexo. Para os projetos abaixo foi usado um filtro FIR com janelamento de Hamming com 500 coeficientes e um filtro IIR com método de Butterworth e 100 coeficientes, com freqüências de corte em 800 Hz e 5 kHz que foram as freqüências de corte definidas após o estudo da caixa e seus alto-falantes, como será visto a seguir.

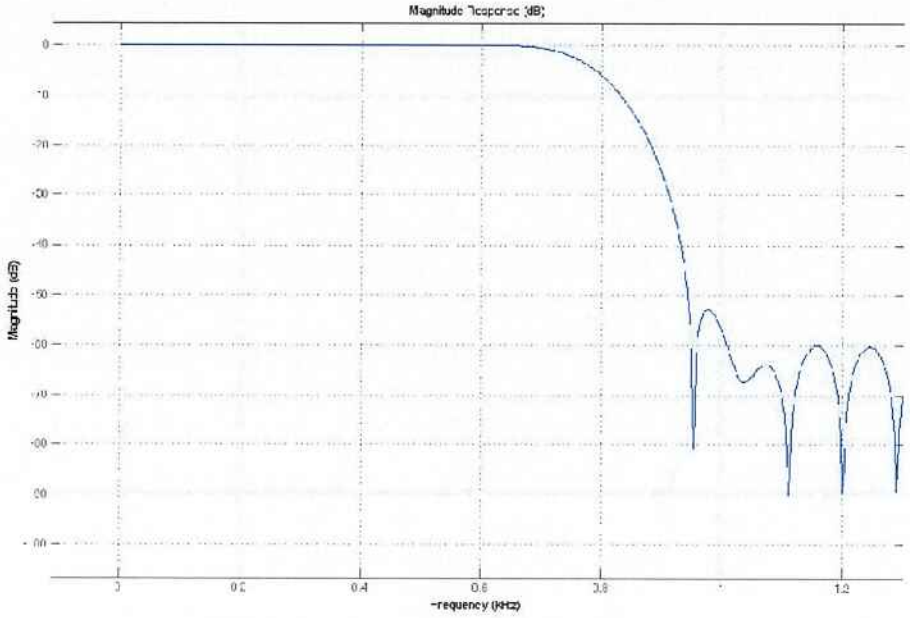
Respostas de fase para o passa-baixas dos filtros IIR e FIR respectivamente, mostrando a não linearidade na fase do filtro IIR



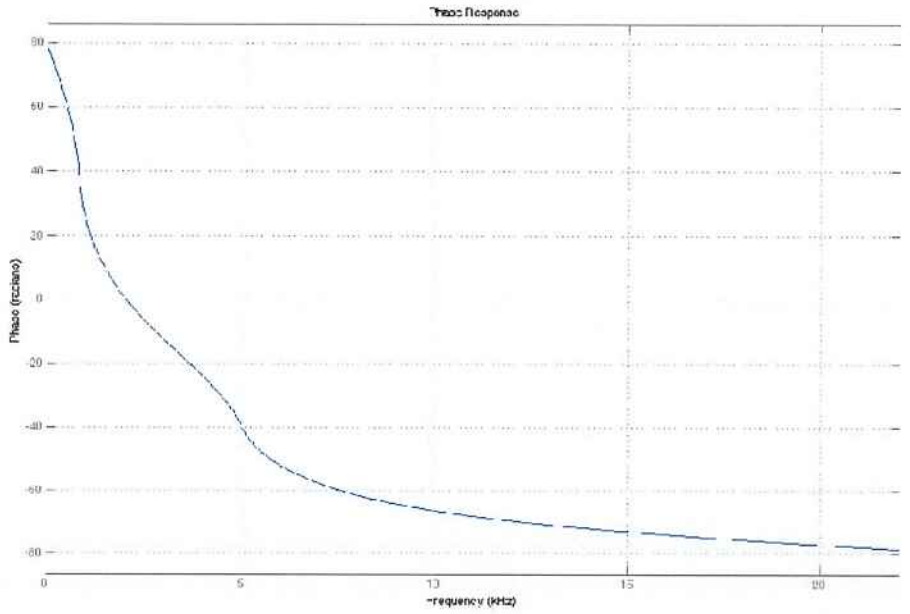


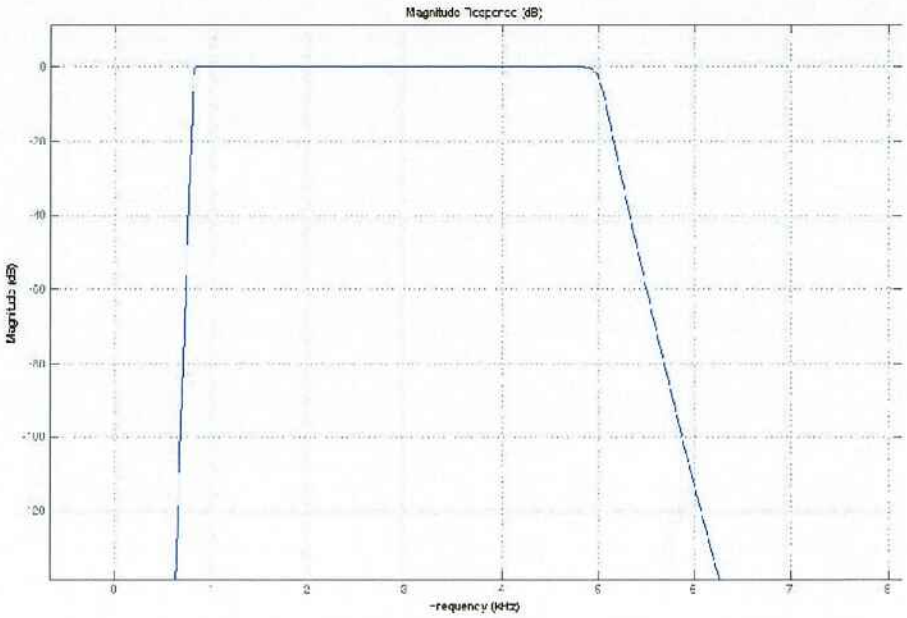
Respostas de magnitude dos filtros IIR e FIR respectivamente, mostrando como o filtro IIR possui uma queda bem mais acentuada com relação ao FIR.



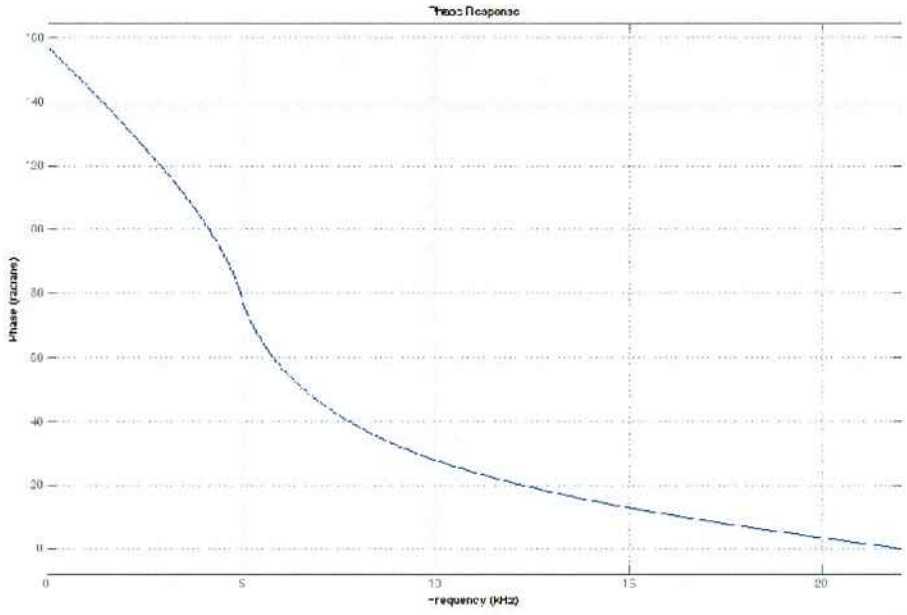


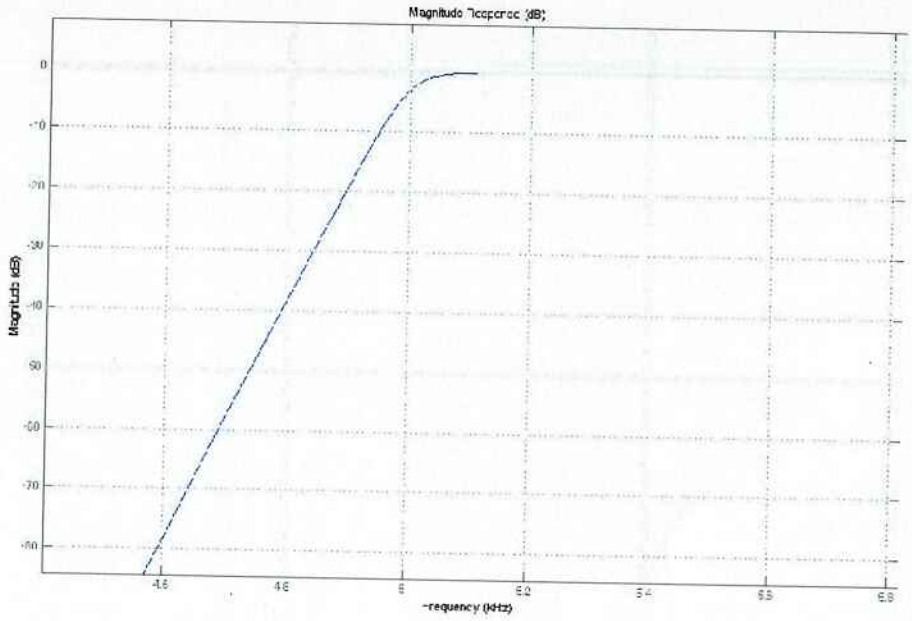
Respostas em Fase e em Magnitude respectivamente do filtro passa-faixa com filtro IIR e método Butterworth com 100 coeficientes.





Respostas em Fase e em Magnitude respectivamente do filtro passa-altas, com filtro IIR e método Butterworth com 100 coeficientes.





10.5 Medições dos parâmetros da caixa e seu aprimoramento

As medições de resposta em freqüência da caixa para os diferentes alto falantes são cruciais para o projeto. Sem essas medições não se levaria em conta à influência que a caixa tem nos alto-falantes e no projeto como um todo.

A caixa foi projetada de forma a se obter uma melhor resposta em freqüência dos alto-falantes na faixa em que eles devem trabalhar. A caixa, assim como o ambiente onde será ouvido o som, tem uma influência enorme na resposta em freqüência de alto-falantes, ela pode atenuar ou evidenciar algumas freqüências de acordo com as especificações dos alto-falantes e de cálculos de projeto da caixa. A caixa foi projetada de forma a evidenciar essas determinadas faixas de freqüências de cada alto-falante, fato conseguido fazendo dutos na base da caixa para reforçar os graves, uma corneta para os médios e uma cúpula para os agudos. Esses cálculos foram feitos com base na faixa de freqüência em que esses alto-falantes trabalham e nos seus respectivos comprimentos de onda, usando esses valores para se calcular medidas de base, altura e outros parâmetros da caixa.

As medições são na verdade a resposta em freqüência da caixa para os seus diferentes alto-falantes. A caixa foi levada para um ambiente onde se minimizaria os

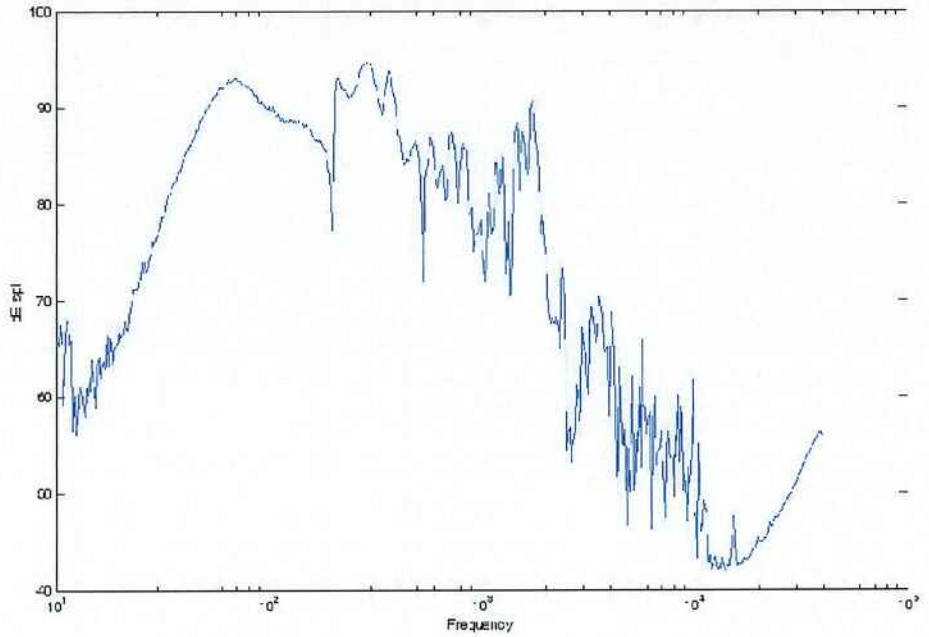
efeitos de reflexão que poderiam ser captados pelo microfone de medição, no caso o estacionamento da elétrica, e assim, com o auxilio do software LEAP se mediu sua resposta em freqüência. Essas reflexões poderiam, se o ensaio fosse realizado em uma sala pequena, influenciar na resposta em freqüência, pois poderia ocorrer cancelamento de fase e assim uma sensível redução no ganho em algumas freqüências. O ideal seria fazer o ensaio em uma sala anecóica, porém como uma sala desse tipo é caríssima e seria impossível de se realizar o experimento em uma no atual momento do projeto, optou-se pelo estacionamento, o qual, apesar de ter vazamento de ruído vindo de carros e outros ruídos do ambiente não possui o cancelamento e fase que uma sala pequena possui e os ruídos não foram tão prejudiciais para as medidas.

Primeiro se estabeleceu como potência fixa 1 W para ensaio. Essa potência foi constante durante as medidas para se ter certeza de que em cada alto-falante se usou a mesma quantidade de energia para se movimentar seus cones e assim obter uma comparação correta entre eles. Usou-se tensão de 2,8V para conseguir tal potência. Esta era distribuída pelo amplificador de potência aos alto-falantes. Estabelecidos esses parâmetros começou-se a se efetuar as medidas.

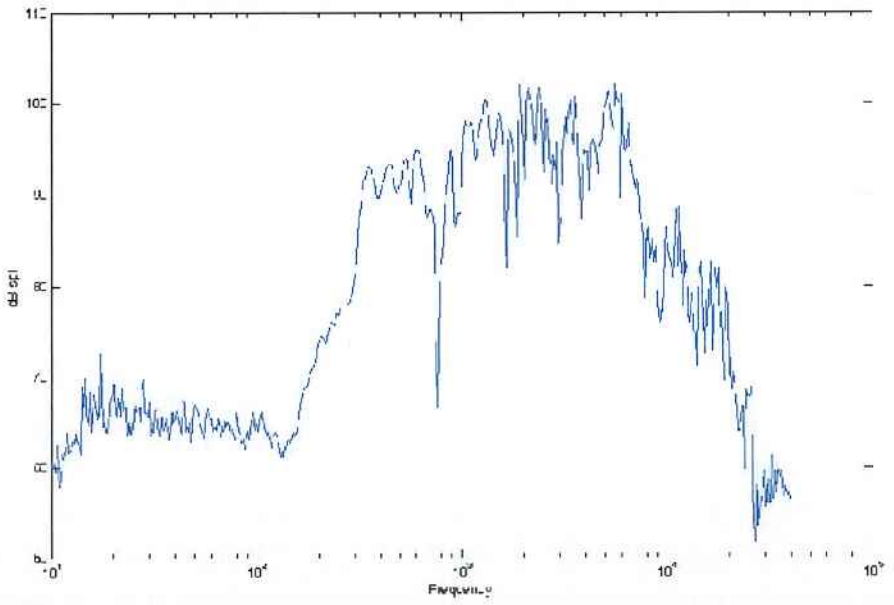
Primeiramente se mediu a resposta em freqüência do falante de graves com o microfone posicionado com uma altura de 1,25m a uma distância de 1m da caixa com ângulo de 0 grau com relação ao solo (posição horizontal). Obteve-se a resposta em freqüência abaixo. Nessa primeira medida observou-se que se teve uma queda acentuada de ganho na faixa de 200 Hz, isso foi devido provavelmente ao duto e a seus parâmetros, um estudo sobre isso poderia ser feito e no futuro uma possível correção seria feita de forma a minimizar essa queda de ganho. Por agora compensou-se a queda com o filtro.

O software fez uma varredura de 10 a 40 kHz e, mantendo a potência constante em 1 W, mediu-se a resposta em freqüência do conjunto caixa e alto-falante para 552 pontos entre 10 e 40KHz.

A resposta para o falante de graves está abaixo.

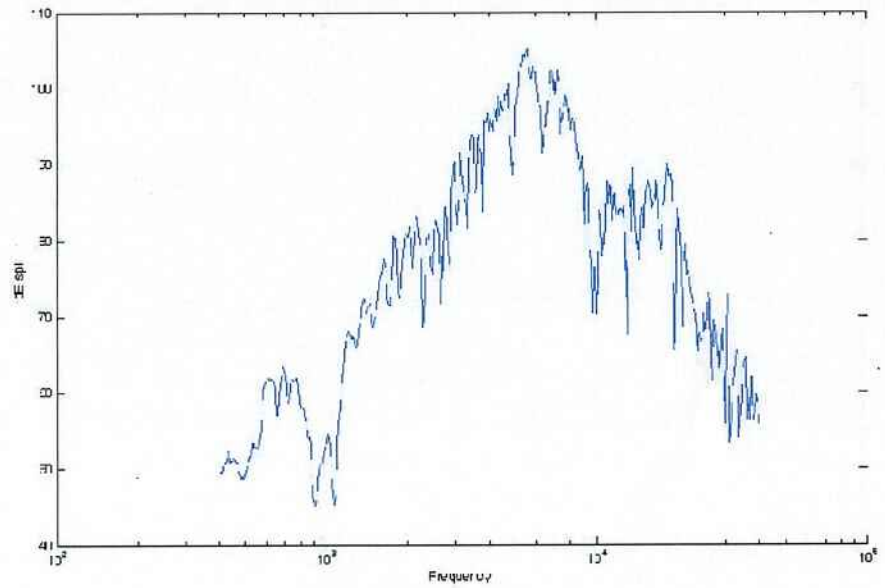


A próxima medida foi a medida do falante de médios. Para isso mantiveram-se as mesmas distancias e potência aplicadas para o falante de graves. Fez-se também uma varredura de 10 a 40 kHz. A resposta está abaixo.

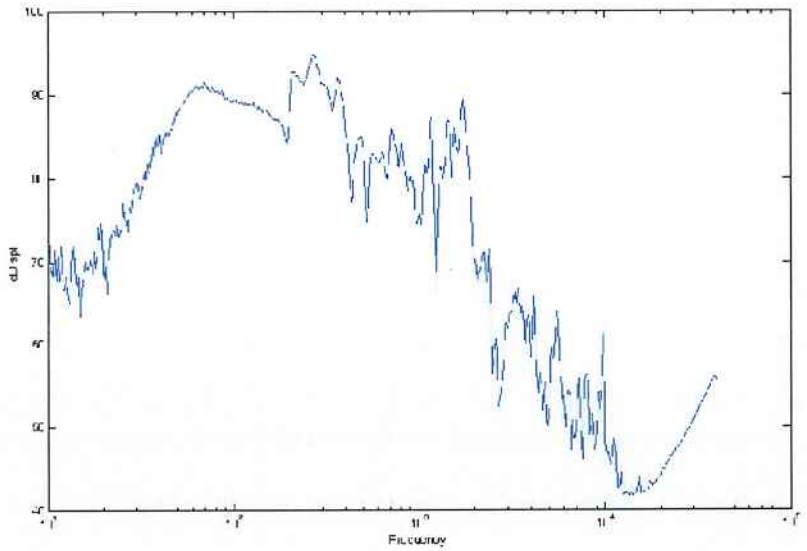


Depois se mediu o falante de agudos. Para isso mantiveram-se as mesmas distâncias e potência aplicadas para os falantes de graves e médios, porém se fez uma

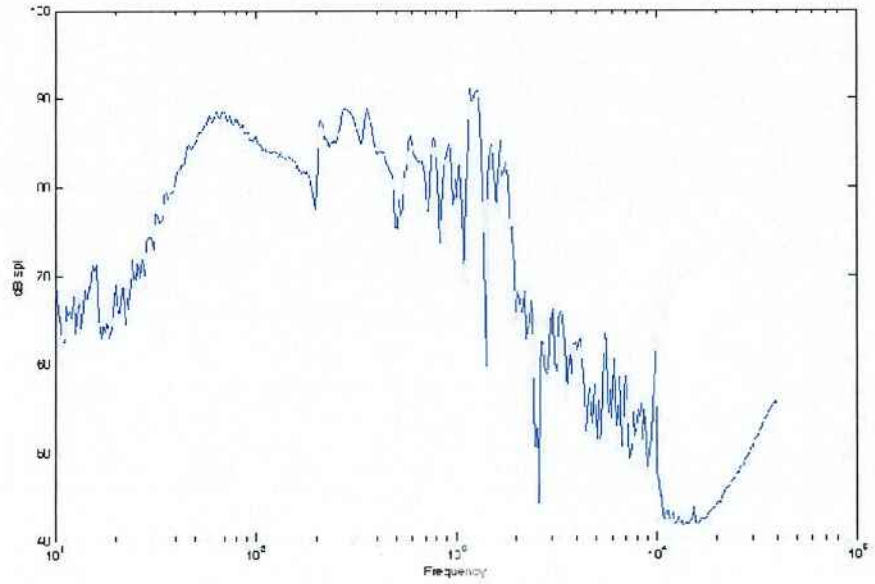
varredura de 400 a 40KHz para assim proteger o falante, que não esta preparado para trabalhar nessa faixa de baixas freqüências. A resposta esta abaixo.

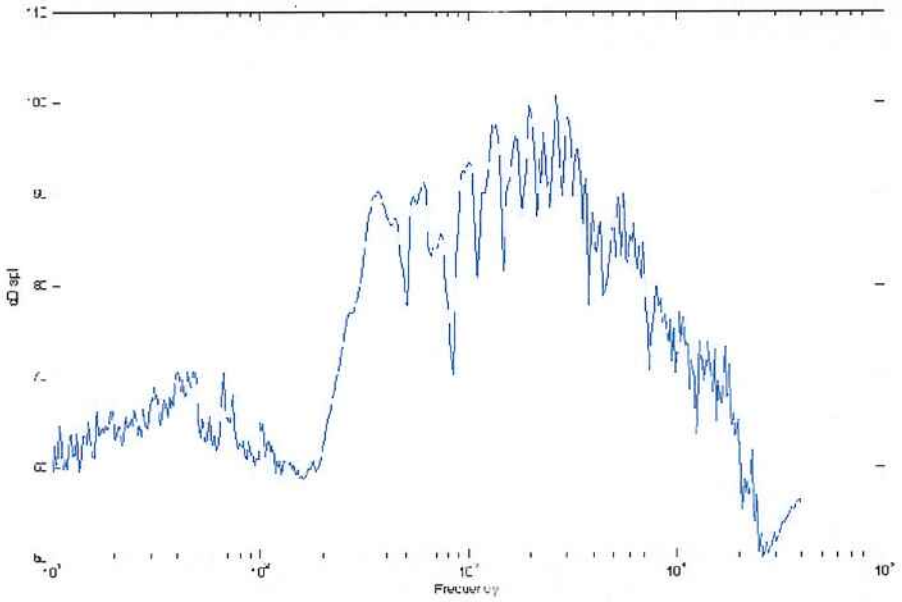


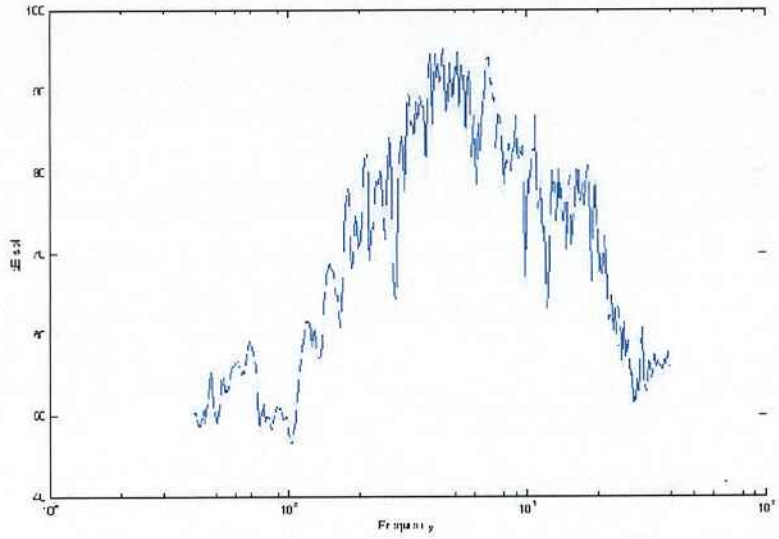
Após essas medições se tentou fazer uma nova medição para o falante de graves, porém agora com os dutos de ar tampados para assim se verificar a influência do duto na grande queda de ganho que se teve em 200Hz. A resposta está abaixo e pode-se ver claramente nela que se há um ganho de 4 dB aproximadamente com os dutos tampados o que indica algum erro de projeto, o qual poderia ser futuramente corrigido.



Depois se fez de novo as medidas, porém agora para a distância de 2m entre microfone e caixa, simulando assim uma pessoa a 2m da caixa. Os parâmetros foram os mesmos das medidas para 1m de distância. As respostas respectivamente para graves, médios e agudos estão abaixo.







A partir dessas respostas se pode finalmente definir as freqüências de corte do crossover. Baseados nas respostas obtidas optou-se por freqüências de corte em 800 Hz e em 5 kHz. Essas são as freqüências de corte do filtro passa-baixa e do filtro passa-alta, respectivamente, as quais também são as freqüências de corte do filtro passa-faixa.

Para se chegar a esses valores analisaram-se as curvas de módulo da resposta, e assim defini-las com base em seus ganhos em suas regiões de melhor resposta.

Outras evidências de desenvolvimento da caixa foram os novos e reforçados pés de apoio e os dutos, os quais, apesar de estarem atrapalhando na faixa de 200 Hz dão um aumento de ganho nas outras freqüências graves. As fotos estão abaixo juntamente com algumas fotos do processo de medição da caixa acústica.

A Caixa e seu equipamento de medição

A Caixa montada com seus novos pés de apoio

A Caixa com seus novos dutos

O Pedestal e o microfone em detalhes

O Computador e suas saídas e entradas para fazer a medição

O Amplificador de potência e seu input, aonde vai ligado o computador, e o output, aonde vai ligado o alto-falante

A Caixa sendo medida para 1m de distância

O Software LEAP

A Caixa sendo medida para 2m de distância

10.6 A programação no DSP

Feitas as medidas na caixa omni-direcional, as definições das freqüências de corte e conseqüentemente os filtros que seriam utilizados, passou-se a etapa de programação. Essa foi o maior desafio do projeto, devido à necessidade de se por em pratica tudo o que se tinha definido na teoria.

Primeiramente foi feito todo o processo de instalação e licenciamento do software e da placa. Fez-se contato com os engenheiros da Analog Devices, empresa que fabrica a placa, e em seguida foi instalado o software VisualDSP++ versão 5.0, versão mais atual do software. Não foi utilizada a versão do software que acompanha a placa pois esta já se encontrava obsoleta e não possuía uma documentação apropriada, documentação essa que foi de grande importância durante o desenvolvimento e programação do projeto.

Com a placa e o software instalados e funcionais, passou-se à parte do entendimento e aprendizado sobre como utilizar um DSP. Para isso seguiu-se os tutoriais sugeridos no help do VisualDSP++. Esses tutoriais deram uma boa visão sobre o funcionamento do DSP, como procedimentos para compilar, trabalhar com projetos, plotar dados, modificar funções, entre outras utilidades e funcionalidades.

Com um pouco mais de pesquisa, agora já voltada à parte prática se descobriu, no manual da placa, que se poderia usar um programa, parcialmente completo, que implementava a entrada e saída de áudio da placa. Esse programa, chamado de Talkthrough, recebe um sinal estéreo, através das entradas Left_0 e Right_0, em seguida amostra esse sinal, faz a alocação de memória e em seguida chama a função process_data, a qual nesse caso do talktrough apenas copia a entrada dos canais Left e Right e as joga nos canais de saída Left e Right. Após isso o sinal passa por um conversor DA e ouve-se o som original. Tudo isso é implementado com a ajuda do codec AD1836 que já vem com a placa, o qual implementa os conversores AD e DA.

Fundamentalmente trabalhou-se dentro do algoritmo na função process_data. Primeiramente definiu-se qual seria a freqüência de amostragem a ser usada. Para isso foi necessário alterar parcialmente o código do Talkthrough. Existe dois tipos de Talkthrough, o I2S e o TDM. No TDM, a freqüência de amostragem é 48 kHz, e podem-se usar todos os 3 canais de saída, já no I2S a freqüência de amostragem é 96KHz porém só se pode usar 2 canais de saída. Baseando-se na quantidade de processamento necessário e nas necessidades do projeto escolheu-se o método TDM. Com ele se pouparia processamento, o qual poderia depois ser um problema no projeto, pois são processadas menos amostras e mesmo assim consegui-se uma qualidade muito boa de som. Comparando-se com o áudio produzido por um CD essa qualidade é até superior, pois foi usado no projeto 48000 Hz ao invés dos 44100 Hz de um CD. A precisão das amostras é de 24 bits, também superior aos 16 bits do CD. O Talkthrough funcionou perfeitamente e com isso bastaria agora programar os três filtros dentro da função process_data.

Essa função é chamada toda vez que um frame de áudio e recebido pelo buffer de recepção. Um frame de áudio é uma coleção de amostras, as quais são agrupadas para se poupar processamento e acelerar o processo de transmissão do áudio. O filtro funciona da seguinte maneira: Cada vez que chega uma amostra e a função process_data é chamada, a amostra da entrada passa pelos três filtros e se sai já trabalhada por canais distintos. Em outras palavras, usa-se a mesma entrada em três filtros em paralelo e depois se responde à resposta específica de cada filtro em

canais específicos e distintos. Feito isso, a função process_data devolve as amostras nos canais e inicia-se novamente o processo.

Passou-se então para a próxima etapa que foi o projeto e implementação dos filtros dentro da função process_data. Primeiramente foram realizadas tentativas com as funções iir_fr16 e iirdr1_fr16. Ambas as funções vieram juntas com o programa VisualDSP++ e já estavam prontas, bastando-se tratar os coeficientes e passá-los como parâmetros na chamada da função, porém obtiveram-se muitos problemas com ambas as funções. A função iir_fr16 pede que os coeficientes A1 e A2 sejam ambos menores do que o A0, que é tomado como sempre 1, em módulo. Isso foi um obstáculo, pois o MatLab ao desenhar o filtro calcula os coeficientes de forma que nem sempre consegue-se cumprir essa exigência. Além disso, a função também necessita que todos os parâmetros passados a ela estejam em formato FRACT16. FRACT16 é a notação utilizada para DSPs que trabalham com aritmética de ponto fixo, que é o caso do DSP utilizado. Para se passar os coeficientes para essa notação pode-se usar uma biblioteca do VisualDSP++, chamada fract2float_conv.h. Essa biblioteca porém parte do pressuposto que os valores que são passados para as funções dela estejam em float entre -1 e 1. Para isso foi utilizado a opção do FDATool que quantizava os coeficientes do filtro; porém, ao quantizar o filtro e modificar os valores dos seus coeficientes, foi alterada a resposta em freqüência do filtro, opção essa não desejada. Por esse motivo desistiu-se de usar a função iir_fr16. Além disso, a parte de Help do VisualDSP++ não é clara, deixando de definir a forma sobre como esses coeficientes eram passados e se é necessário multiplicá-los pelos ganhos de seção dos filtros.

Quanto à outra função, iirdf1_fr16, algumas dessas exigências não eram necessárias, como A1 e A2 serem menores do que A0 e também não era necessário que os coeficientes estivessem em fract16, porém essa função não implementava um filtro de seções biquadráticas. Como o MatLab responde os coeficientes em seções biquadráticas seria inviável usar essa função, pois seria necessário implementar um filtro com no máximo ordem 2 ou agrupar as várias seções em uma só, compromentendo a resposta em freqüência do filtro. Além disso, essa função também não possuía uma boa descrição, e conseqüentemente possuía pontos que não eram claros em sua implementação.

Desistiu-se então de tentar utilizar as funções já prontas e passou-se então a programar as funções do inicio. Com o auxílio do professor Vitor Nascimento se conseguiu escrever, após algumas tentativas sem sucesso, o código que implementa um filtro IIR de seções biquadráticas. Esse código, que está na seção de anexos, implementa um filtro IIR através da declaração dos vetores de coeficientes de cada seção e dos blocos de atraso das entradas e saídas. Resumindo, o algoritmo recebe a entrada vinda do código Talkthrough que chama a função process_data, e passa-se a entrada pelas seções de cada filtro. Isso é feito através de um loop que passa a entrada por cada bloco do filtro. Finalmente calcula-se a saída do bloco todo e a reproduz-se no canal respectivo do filtro, sempre armazenando as entradas e saídas atuais para serem usadas em amostras posteriores. Mais comentários estão feitos no próprio código que está em anexo. Abaixo estão às principais características da placa DSP.

- Pacote Mini-BGA de 160 pinos
- 27 MHz de Clock
- Memória Flash de 2MB
- Codec AD1836 que possui Freqüência de amostragem de até 96 KHz
- 4 entradas mono de áudio e 6 saídas mono de áudio.
- 10 LEDs
- 5 Push Buttons sendo 1 reset e 4 flags programáveis

10.7 A implementação do filtro peaking e as medições finais

Como previsto no inicio do projeto, desejava-se ao fim do semestre não só desenvolver o filtro digital para separação das freqüências, mas também efetuar a correção dos parâmetros da caixa, verificadas no ensaio do estacionamento da escola.

Concluído o código para a separação das freqüências, dedicou-se o pouco tempo restante para a compensação via software da caixa acústica. Para isso utilizou-se o mesmo método desenvolvido para o restante do projeto.

Inicialmente projetou-se um filtro digital através do FDATool que compensasse a resposta obtida nos ensaios dos alto falantes. Ou seja, a partir da

analise visual da resposta da caixa acústica, o filtro de compensação corrigiu a curva somando ganhos de forma a se obter a resposta mais plana possível.

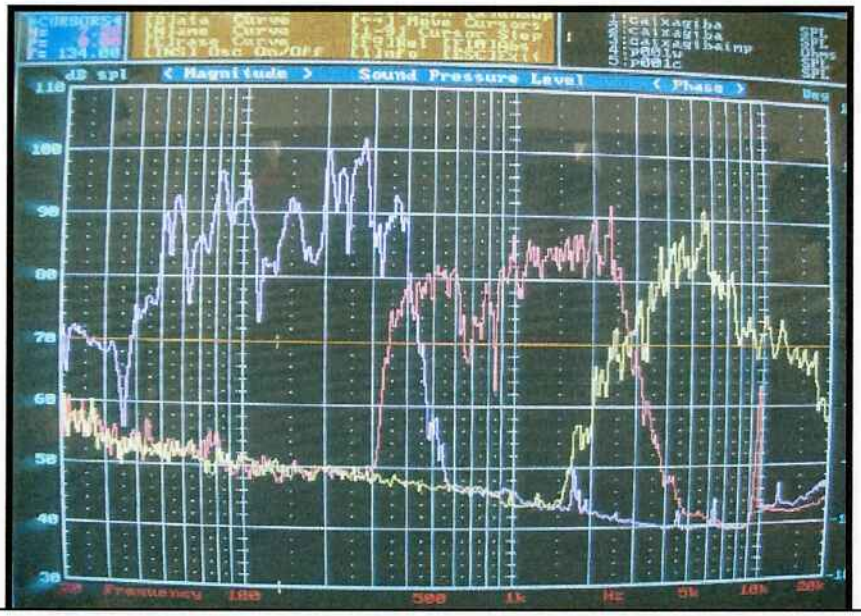
Já em relação ao DSP o algoritmo dessa compensação foi basicamente a adição de um ganho na resposta do filtro. Com a resposta do bloco do filtro digital, o algoritmo adicionava a resposta do filtro peaking de compensação a ele, buscando a resposta plana ideal.

Devido ao tempo curto a implementação da correção foi feita apenas para a parte de filtros passa-baixas. O detalhe completo do código pode ser verificado nos anexos ao fim do relatório.

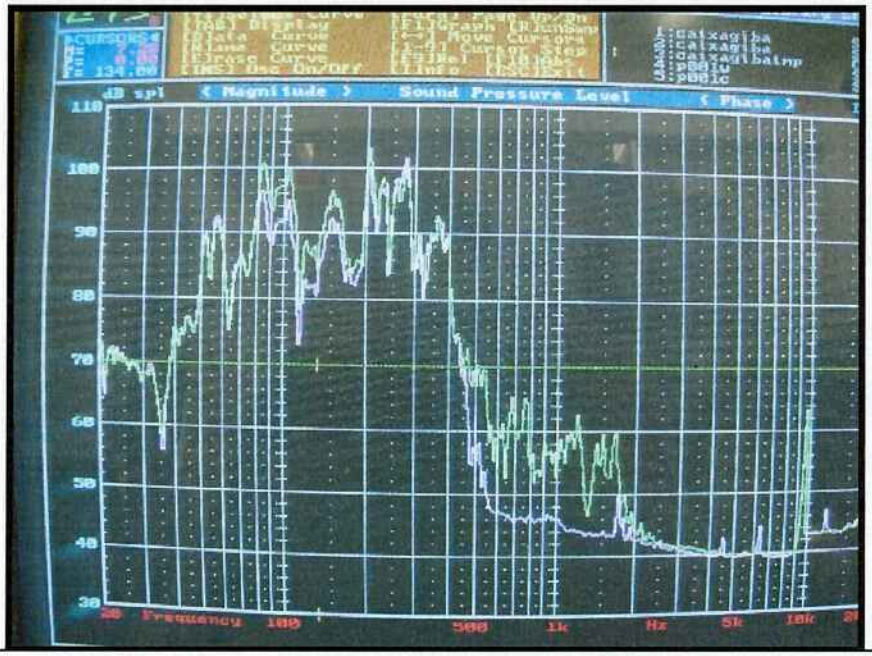
Depois de feitas as modificações foi feita a analise da resposta da caixa acústica, mas desta vez com o filtro digital e a compensação da etapa de passa baixas atuando. A resposta ideal, dado os parâmetros de construção do filtro, se resume a: Filtro passa baixas com corte em 800 Hz, filtro passa bandas com corte em 800 Hz e 2000 Hz e filtro passa altas com corte em 2000 Hz.

Novamente foi feita uma varredura através de uma senóide de freqüência variável através do software Leap. Usou-se a variação de 40 kHz a 10 Hz para os falantes de médios e graves, e variação de 40 kHz a 400 Hz para o de agudos, valores esses já justificados anteriormente nos ensaios iniciais.

A seguir a resposta de todos os falantes, bem como o filtro de compensação comparado a mesma resposta sem a correção.



Resposta da caixa para os três filtros, sem a compensação: Passa baixas (roxo), passa banda (vermelho) e passa altas (amarelo)



Filtro passa baixas com compensação (verde) e sem compensação (roxo)

Algumas considerações podem ser feitas a respeito das curvas obtidas:

As curvas dos três alto falantes se encontram desniveladas, uma em relação às demais. Isso se deve pela potência de cada amplificador ligado ao alto falante. Com o ajuste de potência (volume) poder-se-ia facilmente nivelar as curvas, e se obter a resposta plana desejada na reprodução final. No entanto, para análise termo a termo as curvas obtidas são suficientes.

Na figura de compensação do filtro passa baixas a correção foi feita por volta de 200 Hz. A curva inicial possui uma grande queda nessa região, já na curva corrigida essa curva fica muito mais atenuada.

Por fim, apesar do curto período de tempo, fica claro a facilidade de se projetar os outros filtros peaking de correção, e a melhoria do filtro passa baixas já realizada pode ser otimizada.

11 CONCLUSÃO

Ao final do projeto concluiu-se que foi possível se obter o melhor rendimento e resultado para as limitações que foram impostas ao longo do trabalho. Um estudo futuro com melhores equipamentos e com mais tempo poderá melhorar ainda mais a resposta do filtro projetado. Em uma visão geral pode-se dizer que se está próximo da resposta ideal, porém para isso algumas mudanças devem ser efetuadas.

No FDATool foram atingidas as metas pretendidas. A manipulação dos parâmetros do software tornou-se simples e de fácil entendimento do projeto após estudo inicial. Conseguiu-se projetar os filtros desejados e obter os coeficientes dos filtros. Foi possível também fazer uma interação com a placa DSP, porém apenas após ter-se mudado para o desenvolvimento na integra do filtro. Inicialmente, como o MatLab não trabalha com coeficientes em ponto fixo, obteve-se dificuldade de se implementar o filtro. Vale à pena ressaltar que o Toolbox Filter Design é uma ferramenta muito poderosa e que possui funções muito boas para o completo projeto e análise de filtros, porém, como já foi dito, o projeto foi limitado pela necessidade de se trabalhar em ponto fixo.

No DSP houveram dificuldades devidas a vários fatores e pode-se dizer que a placa foi o maior agravante no cumprimento das datas previstas no cronograma. Primeiro a implementação em ponto fixo que é mais imprecisa do que a implementação em ponto flutuante; Como se programou o filtro em ponto flutuante, era preciso converter os valores para ponto fixo, consumindo um processamento muito maior do que se estivesse trabalhando diretamente com ele. Para isso pode-se futuramente tomar duas providências:

Aquisição da placa Sharc da Analog Devices, que trabalha com ponto flutuante, ou alguma semelhante. Outra opção seria fazer a programação no VisualDSP++ em ponto fixo, porém essa é uma implementação ruim para filtros IIR uma vez que em ponto fixo acumula-se uma quantidade de erros maior na resposta do filtro por causa da imprecisão com que se está trabalhando. Como dito anteriormente, com isso são obtidas respostas erradas, e sem dúvida esse foi um dos fatores que agravaram o não cumprimento com êxito total do filtro. Como o filtro desenvolvido trabalha em ponto fixo e foi programado em ponto flutuante,

imprecisões foram introduzidas na conversão e cálculo, e conseqüentemente obteve-se uma resposta com distorção em alguns casos. Para contornar esse problema projetou-se filtros de ordem menores, obtendo-se assim uma resposta com menor ruído, distorção e menor processamento. Infelizmente a resposta obtida não foi tão boa quanto à desejada, porém ficou próxima ao desejado.

Após testes de respostas na caixa obtiveram-se bons resultados. O som como um todo ficou nítido e limpo, e a caixa respondeu como o esperado. O resultado só poderia ser melhor com o filtro mais próximo do desejado. Acredita-se que com mais tempo para o trabalho do filtro será possível se obter o resultado ideal.

Como um balanço geral, o projeto de formatura foi um sucesso na medida em que foi atingida quase a totalidade dos objetivos. Foi possível também, para alcançar tais objetivos, aprender muito, não só apenas sobre o projeto de filtros em si mas também sobre trabalho em grupo, perseverança, gestão de projetos, escrita de documentos, manuseio de DSPs, caixas acústicas, separação de tarefas, entre tantas outras qualidades que são imprescindíveis a um engenheiro.

Para escrever esse relatório foi usado o livro "Design Digital Filters"de Charles S. Williams, reuniões com o orientador e os professores Luiz Coelho, Vitor Nascimento e Guido Stolfi que também contribuiram com o projeto, leitura de manuais e o help do MatLab, help do Visual DSP++, manual do software Leap, leitura de manuais de diferentes DSPs além de extensivas pesquisas em internet e emails trocados com a Analog Devices.

12 REFERÊNCIAS

Para escrever esse relatório foi usado o livro "Design Digital Filters" de Charles S. Williams, reuniões com o orientador e os professores Luiz Coelho, Vitor Nascimento e Guido Stolfi que também contribuíram com o projeto, leitura de manuais e o help do MatLab, help do Visual DSP++, manual do software Leap, leitura de manuais de diferentes DSPs além de extensivas pesquisas em internet e emails trocados com a Analog Devices.

13 BIBLIOGRAFIA

- Designing Digital Filters (Charles S. Williams)
- Manual software LEAP
- Processamento Digital de Sinais. Disponível em: <http://www.cin.ufpe.br/~ctal/dsp/?Filtros_Eletr%F4nicos>
- DSP Guide. Disponível em: <http://www.dspguide.com/>
- Wikipedia. Disponível em: <http://pt.wikipedia.org/wiki/P%C3%A1gina_principal>
- Analog Devices. Disponível em: <http://www.analog.com.br/>
- Digital Filters: An introduction. Disponível em: <http://www.dsptutor.freeuk.com/dfilt1.htm>
- Introduction to digital filters with audio applications. Disponível em: <http://ccrma.stanford.edu/~jos/filters/>
- Embedded signal processing with the Micro Signal Architecture (Woon-Seng Gan, Sen-Maw Kuo)
- Benjamin Franklin Institute of Technology – course ECE 3551. Disponível em: <http://my.fit.edu/~vkepuska/ece3551/>
- Crossovers (*J.E. Mitchell).* Disponível em: <http://www.frazierspeakers.com/download/cross.pdf>
- Design of the Sound System. Disponível em: <http://www.silcom.com/~aludwig/Sysdes/Design_of_the_sound_system.htm>
- IIR Filter Structures. Disponível em: <http://cnx.org/content/m11919/latest/>
- Digital-Signal-Processing Filters (DSP) . Disponível em: <http://www.freqdev.com/guide/dspguide.html#digdesign>
- Signal Processing Toolbox 6.12. Disponível em: <http://www.mathworks.com/products/signal/description1.html>

ARQUIVO TALKTHROUGH.H

```
#ifndef __Talkthrough_DEFINED
        #define __Talkthrough_DEFINED

//------------------------------------------------------------------------//
// Header files
                                                                          //
//------------------------------------------------------------------------//
#include <sys\exception.h>
#include <cdefBF533.h>
#include <ccblkfn.h>
#include <sysreg.h>

//------------------------------------------------------------------------//
// Symbolic constants
                                                  //
//------------------------------------------------------------------------//
// addresses for Port B in Flash A
#define pFlashA_PortA_Dir       (volatile unsigned char *)0x20270006
#define pFlashA_PortA_Data      (volatile unsigned char *)0x20270004

// names for codec registers, used for iCodec1836TxRegs[]
#define DAC_CONTROL_1                   0x0000
#define DAC_CONTROL_2                   0x1000
#define DAC_VOLUME_0                    0x2000
#define DAC_VOLUME_1                    0x3000
#define DAC_VOLUME_2                    0x4000
#define DAC_VOLUME_3                    0x5000
#define DAC_VOLUME_4                    0x6000
#define DAC_VOLUME_5                    0x7000
#define ADC_0_PEAK_LEVEL    0x8000
#define ADC_1_PEAK_LEVEL    0x9000
#define ADC_2_PEAK_LEVEL    0xA000
#define ADC_3_PEAK_LEVEL    0xB000
#define ADC_CONTROL_1                   0xC000
#define ADC_CONTROL_2                   0xD000
#define ADC_CONTROL_3                   0xE000

// names for slots in ad1836 audio frame
#define INTERNAL_ADC_L0                 0
#define INTERNAL_ADC_L1                 1
#define INTERNAL_ADC_R0                 4
#define INTERNAL_ADC_R1                 5
#define INTERNAL_DAC_L0                 0
#define INTERNAL_DAC_L1                 1
#define INTERNAL_DAC_L2                 2
#define INTERNAL_DAC_R0                 4
#define INTERNAL_DAC_R1                 5
#define INTERNAL_DAC_R2                 6

// size of array iCodec1836TxRegs and iCodec1836RxRegs
#define CODEC_1836_REGS_LENGTH    11

// SPI transfer mode
#define TIMOD_DMA_TX 0x0003

// SPORT0 word length
```

```
#define SLEN_32         0x001f

// DMA flow mode
#define FLOW_1          0x1000


//-------------------------------------------------------------------------//
// Global variables
                                                                           //
//-------------------------------------------------------------------------//
extern int iChannel0LeftIn;
extern int iChannel0RightIn;
extern int iChannel0LeftOut;
extern int iChannel0RightOut;
extern int iChannel1LeftIn;
extern int iChannel1RightIn;
extern int iChannel1LeftOut;
extern int iChannel1RightOut;
extern volatile short sCodec1836TxRegs[];
extern volatile int iRxBuffer1[];
extern volatile int iTxBuffer1[];

//-------------------------------------------------------------------------//
// Prototypes
                                                                           //
//-------------------------------------------------------------------------//
// in file Initialisation.c
void Init_EBIU(void);
void Init_Flash(void);
void Init1836(void);
void Init_Sport0(void);
void Init_DMA(void);
void Init_Sport_Interrupts(void);
void Enable_DMA_Sport(void);
void Enable_DMA_Sport0(void);


// in file Process_data.c
void Process_Data(void);

// in file ISR.c
EX_INTERRUPT_HANDLER(Sport0_RX_ISR);

#endif //__Talkthrough_DEFINED
```

ARQUIVO MAIN.C

```
//-------------------------------------------------------------------------//
//
                                                                           //
//          Name:  Talkthrough for the ADSP-BF533 EZ-KIT Lite
                   //
//
                                                                           //
//-------------------------------------------------------------------------//
```

```
//                                                                          //
//      (C) Copyright 2006 - Analog Devices, Inc.  All rights reserved.     //
//                                                                          //
//      Project Name:   BF533 C Talkthrough TDM                             //
//                                                                          //
//      Date Modified:  04/03/03            Rev 1.0                         //
//                                                                          //
//      Software:               VisualDSP++4.5                              //
//                                                                          //
//      Hardware:               ADSP-BF533 EZ-KIT Board                     //
//                                                                          //
//      Connections:    Disconnect RSCLK0 and TSCLK0 (Turn SW9 pin 6 OFF)   //
//          Disconnect RFS0 and TFS0 (Turn SW9 pin 5 OFF)                   //
//          Connect an input source (such as a radio) to the Audio          //
//                              input jack and an output source (such as headphones) to //
//                              the Audio output jack                       //
//                                                                          //
//      Purpose:        This program sets up the SPI port on the ADSP-BF533 to //
//                              configure the AD1836 codec.  The SPI port is disabled //
//                              after initialization.  The data to/from the codec are //
//                              transfered over SPORT0 in TDM mode          //
//                                                                          //
//--------------------------------------------------------------------------//

#include "Talkthrough.h"
#include "sysreg.h"
#include "ccblkfn.h"

//--------------------------------------------------------------------------//
// Variables                                                                //
//                                                                          //
// Description:    The variables iChannelxLeftIn and iChannelxRightIn contain //
//                      the data coming from the codec AD1836.  The (processed) //
//                      playback data are written into the variables        //
//                      iChannelxLeftOut and iChannelxRightOut respectively, which //
//                      are then sent back to the codec in the SPORT0 ISR.  //
//                      The values in the array iCodec1836TxRegs can be modified to //
//                      set up the codec in different configurations according to //
```

```
//                                  the AD1885 data sheet.
                                                //
//-------------------------------------------------------------------------//
// left input data from ad1836
int iChannel0LeftIn, iChannel1LeftIn;
// right input data from ad1836
int iChannel0RightIn, iChannel1RightIn;
// left ouput data for ad1836
int iChannel0LeftOut, iChannel1LeftOut;
// right ouput data for ad1836
int iChannel0RightOut, iChannel1RightOut;
// array for registers to configure the ad1836
// names are defined in "Talkthrough.h"
volatile short sCodec1836TxRegs[CODEC_1836_REGS_LENGTH] =
{
                                        DAC_CONTROL_1     |0x000,
                                        DAC_CONTROL_2     |0x000,
                                        DAC_VOLUME_0      |0x3ff,
                                        DAC_VOLUME_1      |0x3ff,
                                        DAC_VOLUME_2      |0x3ff,
                                        DAC_VOLUME_3      |0x3ff,
                                        DAC_VOLUME_4      |0x3ff,
                                        DAC_VOLUME_5      |0x3ff,
                                        ADC_CONTROL_1     |0x000,
                                        ADC_CONTROL_2     |0x180,
                                        ADC_CONTROL_3     |0x000

};
// SPORT0 DMA transmit buffer
volatile int iTxBuffer1[8];
// SPORT0 DMA receive buffer
volatile int iRxBuffer1[8];


//-------------------------------------------------------------------------//
// Function:        main
                                                //
//
                                                                        //
// Description:     After calling a few initalization routines, main() just //
//                                  waits in a loop forever.  The code to process the incoming  //
//                                  data can be placed in the function Process_Data() in the        //
//                                  file "Process_Data.c".
                                                //
//-------------------------------------------------------------------------//
void main(void)
{
        sysreg_write(reg_SYSCFG, 0x32);         //Initialize System Configuration Register
        Init_EBIU();
        Init_Flash();
        Init1836();
        Init_Sport0();
        Init_DMA();
        Init_Sport_Interrupts();
        Enable_DMA_Sport0();
        while(1);
}
```

ARQUIVO ISR.C

```
#include "Talkthrough.h"

//------------------------------------------------------------------------//
// Function:        Sport0_RX_ISR                                         //
//                                                                        //
// Description: This ISR is executed after a complete frame of input data //
//                  has been received. The new samples are stored in      //
//                  iChannel0LeftIn, iChannel0RightIn, iChannel1LeftIn and //
//                  iChannel1RightIn respectively.  Then the function     //
//                  Process_Data() is called in which user code can be executed.//
//                  After that the processed values are copied from the   //
//                  variables iChannel0LeftOut, iChannel0RightOut,        //
//                  iChannel1LeftOut and iChannel1RightOut into the dma   //
//                  transmit buffer.                                      //
//------------------------------------------------------------------------//
EX_INTERRUPT_HANDLER(Sport0_RX_ISR)
{
        // confirm interrupt handling
        *pDMA1_IRQ_STATUS = 0x0001;

        // copy input data from dma input buffer into variables
        iChannel0LeftIn = iRxBuffer1[INTERNAL_ADC_L0];
        iChannel0RightIn = iRxBuffer1[INTERNAL_ADC_R0];
        iChannel1LeftIn = iRxBuffer1[INTERNAL_ADC_L1];
        iChannel1RightIn = iRxBuffer1[INTERNAL_ADC_R1];

        // call function that contains user code
        Process_Data();

        // copy processed data from variables into dma output buffer
        iTxBuffer1[INTERNAL_DAC_L0] = iChannel0LeftOut;
        iTxBuffer1[INTERNAL_DAC_R0] = iChannel0RightOut;
        iTxBuffer1[INTERNAL_DAC_L1] = iChannel1LeftOut;
        iTxBuffer1[INTERNAL_DAC_L2] = iChannel1LeftOut;
        iTxBuffer1[INTERNAL_DAC_R1] = iChannel1RightOut;
        iTxBuffer1[INTERNAL_DAC_R2] = iChannel1RightOut;
}
```

ARQUIVO INITIALIZE.C

```
#include "Talkthrough.h"

//-------------------------------------------------------------------------//
// Function:    Init_EBIU                                                  //
//                                                                         //
//                                                                         //
// Description: This function initializes and enables asynchronous memory  //
//              banks in External Bus Interface Unit so that Flash A can be //
//              accessed.                                                  //
//-------------------------------------------------------------------------//
void Init_EBIU(void)
{
        *pEBIU_AMBCTL0      = 0x7bb07bb0;
        *pEBIU_AMBCTL1      = 0x7bb07bb0;
        *pEBIU_AMGCTL       = 0x000f;
}


//-------------------------------------------------------------------------//
// Function:    Init_Flash                                                 //
//                                                                         //
//                                                                         //
// Description: This function initializes pin direction of Port A in Flash A//
//              to output.  The AD1836_RESET on the ADSP-BF533 EZ-KIT
board //
//              is connected to Port A.                                    //
//-------------------------------------------------------------------------//
void Init_Flash(void)
{
        *pFlashA_PortA_Dir = 0x1;
}


//-------------------------------------------------------------------------//
// Function:    Init1836()                                                 //
//                                                                         //
//                                                                         //
// Description: This function sets up the SPI port to configure the AD1836. //
//              The content of the array sCodec1836TxRegs is sent to the   //
//              codec.                                                     //
//-------------------------------------------------------------------------//
void Init1836(void)
{
```

```
    int i;
    int j;
    static unsigned char ucActive_LED = 0x01;

    // write to Port A to reset AD1836
    *pFlashA_PortA_Data = 0x00;

    // write to Port A to enable AD1836
    *pFlashA_PortA_Data = ucActive_LED;

    // wait to recover from reset
    for (i=0; i<0xf000; i++) asm("nop;");

    // Enable PF4
    *pSPI_FLG = FLS4;
    // Set baud rate SCK = HCLK/(2*SPIBAUD) SCK = 2MHz
    *pSPI_BAUD = 16;
    // configure spi port
    // SPI DMA write, 16-bit data, MSB first, SPI Master
    *pSPI_CTL = TIMOD_DMA_TX | SIZE | MSTR;

    // Set up DMA5 to transmit
    // Map DMA5 to SPI
    *pDMA5_PERIPHERAL_MAP    = 0x5000;

    // Configure DMA5
    // 16-bit transfers
    *pDMA5_CONFIG = WDSIZE_16;
    // Start address of data buffer
    *pDMA5_START_ADDR = (void *)sCodec1836TxRegs;
    // DMA inner loop count
    *pDMA5_X_COUNT = CODEC_1836_REGS_LENGTH;
    // Inner loop address increment
    *pDMA5_X_MODIFY = 2;

    // enable DMAs
    *pDMA5_CONFIG = (*pDMA5_CONFIG | DMAEN);
    // enable spi
    *pSPI_CTL = (*pSPI_CTL | SPE);

    // wait until dma transfers for spi are finished
    for (j=0; j<0xaff0; j++) asm("nop;");

    // disable spi
    *pSPI_CTL = 0x0000;
}


//-----------------------------------------------------------------------------//
// Function:        Init_Sport0                                                //
//                                                                             //
//                                                                             //
// Description:     Configure Sport0 for TDM mode, to transmit/receive data    //
//                  to/from the AD1836. Configure Sport for external clocks and //
//                  frame syncs.                                               //
//                                                                             //
//-----------------------------------------------------------------------------//
```

```
void Init_Sport0(void)
{
        // Sport0 receive configuration
        // External CLK, External Frame sync, MSB first
        // 32-bit data
        *pSPORT0_RCR1 = RFSR;
        *pSPORT0_RCR2 = SLEN_32;

        // Sport0 transmit configuration
        // External CLK, External Frame sync, MSB first
        // 24-bit data
        *pSPORT0_TCR1 = TFSR;
        *pSPORT0_TCR2 = SLEN_32;

        // Enable MCM 8 transmit & receive channels
        *pSPORT0_MTCS0 = 0x000000FF;
        *pSPORT0_MRCS0 = 0x000000FF;

        // Set MCM configuration register and enable MCM mode
        *pSPORT0_MCMC1 = 0x0000;
        *pSPORT0_MCMC2 = 0x101c;
}


//------------------------------------------------------------------------//
// Function:      Init_DMA
                                                //
//
                                                                        //
// Description:   Initialize DMA1 in autobuffer mode to receive and DMA2 in //
//                                 autobuffer mode to transmit
                                                //
//------------------------------------------------------------------------//
void Init_DMA(void)
{
        // Set up DMA1 to receive
        // Map DMA1 to Sport0 RX
        *pDMA1_PERIPHERAL_MAP = 0x1000;

        // Configure DMA1
        // 32-bit transfers, Interrupt on completion, Autobuffer mode
        *pDMA1_CONFIG = WNR | WDSIZE_32 | DI_EN | FLOW_1;
        // Start address of data buffer
        *pDMA1_START_ADDR = (void *)iRxBuffer1;
        // DMA inner loop count
        *pDMA1_X_COUNT = 8;
        // Inner loop address increment
        *pDMA1_X_MODIFY    = 4;


        // Set up DMA2 to transmit
        // Map DMA2 to Sport0 TX
        *pDMA2_PERIPHERAL_MAP = 0x2000;

        // Configure DMA2
        // 32-bit transfers, Autobuffer mode
        *pDMA2_CONFIG = WDSIZE_32 | FLOW_1;
        // Start address of data buffer
```

```
        *pDMA2_START_ADDR = (void *)iTxBuffer1;
        // DMA inner loop count
        *pDMA2_X_COUNT = 8;
        // Inner loop address increment
        *pDMA2_X_MODIFY    = 4;
}


//-------------------------------------------------------------------------//
// Function:        Init_Interrupts
                                        //
//
                                                                        //
// Description:    Initialize Interrupt for Sport0 RX
        //
//-------------------------------------------------------------------------//
void Init_Sport_Interrupts(void)
{
        // Set Sport0 RX (DMA1) interrupt priority to 2 = IVG9
        *pSIC_IAR0 = 0xffffffff;
        *pSIC_IAR1 = 0xffffff2f;
        *pSIC_IAR2 = 0xffffffff;

        // assign ISRs to interrupt vectors
        // Sport0 RX ISR -> IVG 9
        register_handler(ik_ivg9, Sport0_RX_ISR);

        // enable Sport0 RX interrupt
        *pSIC_IMASK = 0x00000200;
        ssync();
}


//-------------------------------------------------------------------------//
// Function:        Enable_DMA_Sport
                                        //
//
                                                                        //
// Description:    Enable DMA1, DMA2, Sport0 TX and Sport0 RX
        //
//-------------------------------------------------------------------------//
void Enable_DMA_Sport0(void)
{
        // enable DMAs
        *pDMA2_CONFIG        = (*pDMA2_CONFIG | DMAEN);
        *pDMA1_CONFIG        = (*pDMA1_CONFIG | DMAEN);

        // enable Sport0 TX and RX
        *pSPORT0_TCR1        = (*pSPORT0_TCR1 | TSPEN);
        *pSPORT0_RCR1        = (*pSPORT0_RCR1 | RSPEN);
}
```

ARQUIVO PROCESS_DATA.C

```
#include "Talkthrough.h"
//------------------------------------------------------------------------//
// Function:       Process_Data()                                         //
//                                                                        //
// Description: This function is called from inside the SPORT0 ISR every //
//                 time a complete audio frame has been received. The new //
//                 input samples can be found in the variables iChannel0LeftIn,//
//                 iChannel0RightIn, iChannel1LeftIn and iChannel1RightIn //
//                 respectively. The processed data should be stored in   //
//                 iChannel0LeftOut, iChannel0RightOut, iChannel1LeftOut, //
//                 iChannel1RightOut, iChannel2LeftOut and iChannel2RightOut //
//                 respectively.                                          //
//------------------------------------------------------------------------//

//Numero de Seções de cada filtro
#define LSTAGES 3
#define MSTAGES 5
#define HSTAGES 2

int aux;// coeficiente auxiliar para calcular
    // ganho de seção vezes seus coeficientes B

float cond=0;// condição para passar apenas uma vez pela
        //multiplicação de Bs pelos ganhos de seção

//Low Pass Filter IIR coeficients A and B arrays, IIR Delays and Output
float lpA0[LSTAGES] = {1.1.1};
float lpA1[LSTAGES] = {-1.91868515285543,  -1.94042089318505 , -1.97094205843429};
float lpA2[LSTAGES] = { 0.92127217668006 ,  0.94778553342034 ,  0.98255376593864};

float lpB0[LSTAGES] = {1,1,1};
float lpB1[LSTAGES] = {-1.48723344423739 , -1.91798229488401,  -1.95296403024632};
float lpB2[LSTAGES] = {1,1,1};

float lpSG[LSTAGES] =
{0.0049303839037747009,0.089793297982082784,0.24686867444554317};

float lpy2[LSTAGES]={0,0,0};
float lpy1[LSTAGES]={0,0,0};
float lpx1[LSTAGES]={0,0,0};
float lpx2[LSTAGES]={0,0,0};

float lpy=0;


//Mid Pass Filter IIR coeficients A and B arrays,IIR Delays

float mpA0[MSTAGES]={1,1,1,1,1};
```

```
float mpA1[MSTAGES]={-1.49289204132497 , -1.90249449344371 , -1.47142330298184 , -
1.97102806697544,-1.70029222506033};
float mpA2[MSTAGES]={0.72474118728400  , 0.91928060878535 ,  0.88807654187495 ,
0.98111331012834, 0.76188450091920};

float mpB2[MSTAGES]={1,1,1,1,1};
float mpB1[MSTAGES]={0.65773055297745,  -1.99936074611836,  -0.21714760263261 , -
1.99842629852928,0};
float mpB0[MSTAGES]={1,1,1,1,1};

float mpSG[MSTAGES] = {0.12611353423719771,
                       0.12611353423719771,
                       0.42246287916242276,
                       0.42246287916242276,
                       0.42410762444281641 };


float mpy2[MSTAGES]={0,0,0,0,0};
float mpy1[MSTAGES]={0,0,0,0,0};
float mpx1[MSTAGES]={0,0,0,0,0};
float mpx2[MSTAGES]={0,0,0,0,0};

float mpy=0;

//High Pass Filter IIR coeficients A and B arrays,IIR Delays and Output

float hpA0[HSTAGES]={1,1};
float hpA1[HSTAGES]={-0.80541353115778 , -1.48576085967000 };
float hpA2[HSTAGES]={ 0.25938711048946 ,  0.80383848744985};


float hpB2[HSTAGES]={1,1};
float hpB1[HSTAGES]={-1.99212622526700 , -1.95636248348562};
float hpB0[HSTAGES]={1,1};

float hpSG[HSTAGES] = {0.50544495013710933,
                       0.83147066550425475};


float hpy2[HSTAGES]={0,0};
float hpy1[HSTAGES]={0,0};
float hpx1[HSTAGES]={0,0};
float hpx2[HSTAGES]={0,0};

float hpy=0;



//inicialização do m que varre cada seção dos filtros
int m;


void Process_Data(void)
{
        if(cond==0){
        for(aux=0;aux<LSTAGES;aux++) {
        lpB0[aux]=lpB0[aux]*lpSG[aux];
        lpB1[aux]=lpB1[aux]*lpSG[aux];
```

```
 lpB2[aux]=lpB2[aux]*lpSG[aux];
 }
 for(aux=0;aux<MSTAGES;aux++) {  // blocos que calculam
        mpB0[aux]=mpB0[aux]*mpSG[aux];  // Bs vezes os ganhos de seção
        mpB1[aux]=mpB1[aux]*mpSG[aux];
 mpB2[aux]=mpB2[aux]*mpSG[aux];
 }
 for(aux=0;aux<HSTAGES;aux++) {
        hpB0[aux]=hpB0[aux]*hpSG[aux];
        hpB1[aux]=hpB1[aux]*hpSG[aux];
 hpB2[aux]=hpB2[aux]*hpSG[aux];
 }
 cond=1;
}

        /*******************LOW PASS**********************/
        float lpx=(float)(iChannel0LeftIn); // Frame de audio recebido pela funcao

                                // é passado para float e é processado pelo filtro
        for(m = 0; m < 3; m++)

        {
                        //Equação de saida do filtro Low Pass
    lpy=-lpA1[m]*lpy1[m]-lpA2[m]*lpy2[m]+lpB0[m]*lpx+lpB1[m]*lpx1[m]+lpB2[m]*lpx2[m];
    lpx2[m]=lpx1[m];
    lpx1[m]=lpx;
    lpx=lpy;        // Atualizacao dos vetores de dealy das entradas e saidas
    lpy2[m]=lpy1[m];
    lpy1[m]=lpy;

        }

        iChannel0LeftOut = ((int)lpy); // saida do filtro é armazenada no canal esquerdo 0



        /*************************MID PASS*****************************/
        float mpx =(float)(iChannel0LeftIn);// Frame de audio recebido pela funcao

                                // é passado para float e é processado pelo filtro


        for(m=0; m<5; m++)

        {
                //Equação de saida do filtro Mid Pass
    mpy=-mpA1[m]*mpy1[m]-
mpA2[m]*mpy2[m]+mpB0[m]*mpx+mpB1[m]*mpx1[m]+mpB2[m]*mpx2[m];
    mpx2[m]=mpx1[m];
    mpx1[m]=mpx;
    mpx=mpy;        // Atualizacao dos vetores de dealy das entradas e saidas
    mpy2[m]=mpy1[m];
    mpy1[m]=mpy;
        }

        iChannel0RightOut=((int)mpy);     // saida do filtro é armazenada no canal direito 0
```

```
                /*************************HIGH PASS*****************************/
        float hpx =(float)(iChannel0LeftIn);// Frame de audio recebido pela funcao

                                // é passado para float e é processado pelo filtro


        for(m=0; m<2; m++)

        {
                //Equação de saida do filtro High Pass
    hpy=-hpA1[m]*hpy1[m]-
hpA2[m]*hpy2[m]+hpB0[m]*hpx+hpB1[m]*hpx1[m]+hpB2[m]*hpx2[m];
    hpx2[m]=hpx1[m];
    hpx1[m]=hpx;
    hpx=hpy;      // Atualizacao dos vetores de dealy das entradas e saidas
    hpy2[m]=hpy1[m];
    hpy1[m]=hpy;
        }

        iChannel1LeftOut=((int)hpy);        // saida do filtro é armazenada no canal esquerdo 1

}
```