
**UNIVERSIDADE FEDERAL DE CAMPINA GRANDE**
**CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA**
**UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA**
**LABORATÓRIO DE AUTOMAÇÃO E PROCESSAMENTO DE SINAIS**

**RELATÓRIO DE ESTÁGIO SUPERVISIONADO**

# IMPLEMENTAÇÃO DE FILTROS DIGITAIS RECURSIVOS USANDO O PROCESSADOR DIGITAL DE SINAIS TMS320C6711

Aluno
Johannes Dantas de Medeiros Júnior
johannesee@gmail.com

Orientador
Prof. Dr. Bruno Barbosa Albert
albert@dee.ufcg.edu.br

Campina Grande, Dezembro de 2008

JOHANNES DANTAS DE MEDEIROS JÚNIOR

# IMPLEMENTAÇÃO DE FILTROS DIGITAIS RECURSIVOS USANDO O PROCESSADOR DIGITAL DE SINAIS TMS320C6711

**Relatório de Estágio Supervisionado apresentado à Universidade Federal de Campina Grande, como requisito parcial para a obtenção do título de Engenheiro Eletricista.**

**Orientador: Prof. Dr. Bruno Barbosa Albert**

Campina Grande, Dezembro de 2008


Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

# Dedicatória

Aos meus pais, Johannes e Lúcia, à minha irmã, Palas Atenéia e ao meu tio, Giovanni Bosco.

# Lista de Figuras

# Sumário

# 1 – Introdução

Processamento de sinais é uma tecnologia presente há décadas em diversas áreas como medicina, comunicações e exploração espacial. Até o início da década de 1960 esse processamento era basicamente analógico, no entanto nessa época, com a evolução dos computadores digitais e microprocessadores surgiu um novo meio de realizar esse processamento, o processamento digital de sinais (OPPENHEIM et al, 1999).

A teoria de processamento digital de sinais também é aplicada em diversas áreas como processamento de fala (reconhecimento de fala, conversão texto-fala), sistemas biomédicos (raio X, ultra-som), processamento de imagens (restauração de imagens, processamento de dados geofísicos e sísmicos) e eletrônica de consumo (aparelhos de CD e DVD, copiadoras) (SHENOI, 2006).

Muitas vezes não é possível utilizar um computador para realizar o processamento dos sinais, com isso foi necessário o desenvolvimento de um dispositivo específico para esse fim, que é o Processador Digital de Sinais (DSP – *Digital Signal Processor*).

O custo dos DSPs caiu muito nos últimos anos com o avanço dos circuitos integrados e hoje são utilizados em diversos aparelhos. Em geral os DSPs são usados para processamento em tempo real e têm várias vantagens sobre os sistemas analógicos, como facilidade de reprogramação e são menos afetados pelas condições ambientais (CHASSAING, 2002).

Devido a todos esses fatores surgiu o interesse em utilizar o DSP para implementar filtros digitais recursivos, isso foi realizado como estágio supervisionado no Laboratório de Automação e Processamento de Sinais da Universidade Federal de Campina Grande.

Durante o estágio, além dos experimentos de filtragem de sinais, foram realizados experimentos de geração e amostragem de sinais usando o DSP TMS320C6711 da Texas Instruments™ (TI). Nesse relatório esses experimentos são descritos e seus resultados são apresentados e discutidos.

# 2 – Material Utilizado

## 2.1 – DSP TMS320C6711

O TMS320C6711 (figura 1) é um DSP de ponto flutuante baseado na arquitetura de palavras de instrução muito longas (VLIW – *Very Long Instruction Word*), o que é apropriado para algoritmos que realizam muito cálculos, essas instruções são de 32 bits e é possível executar oito delas por ciclo. Com esse DSP é possível acessar dois bancos de memória ao mesmo tempo, isso significa que é possível executar instruções em paralelo, por exemplo, duas somas e multiplicações por ciclo (CHASSAING, 2002).


**Figura 1**: DSP TMS320C6711. Fonte: http://vm.outs.pl/~jacek/speech

## 2.2 – DSP Starter Kit (DSK)

O *kit* de inicialização do DSP (DSK) é composto do *Code Composer Studio* (apresentado no item seguinte), uma placa que contém o DSP, um cabo paralelo (DB25) e uma fonte de alimentação.

A placa contém além do DSP, um codec AD535 que permite conversão de sinais digital para analógico e de analógico para digital com taxa de amostragem fixa de 8 kHz; conectores P2 para entrada e saída de som; 16 MB de memória SDRAM e 128 KB de memória *flash ROM*, também é possível utilizar placas de expansão (CHASSAING, 2002).

Na figura 2 é mostrada uma dessas placas, onde foram destacados os conectores de entrada e saída de som, o conector para o cabo paralelo, o conector da fonte de alimentação e o DSP.

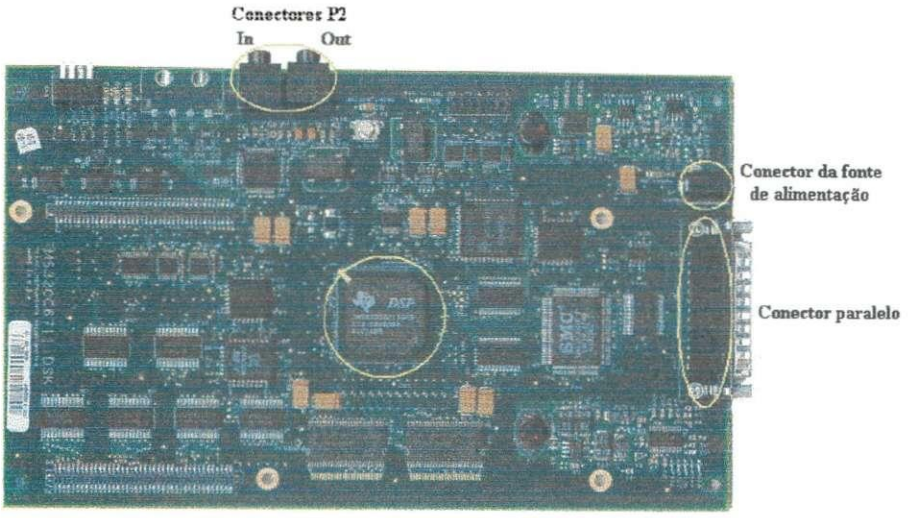



**Figura 2**: Placa do DSK. Fonte: http://vm.outs.pl/~jacek/speech

## 2.3 – Programas Utilizados

Foram usados vários programas durante o estágio para realizar os experimentos, esses programas são listados a seguir.

- **Code Composer Studio (CCS)**

CCS é um ambiente de desenvolvimento desenvolvido pela TI que integra ferramentas como compilador, *linker* e *assembler*. Usando o CCS é possível realizar todo o fluxo de projeto, desde a criação do código fonte em C, até a geração de um executável de extensão *.out* que posteriormente é carregado e executado no DSP (CHASSAING, 2002).

- **Matlab**

Durante o estágio o Matlab[1] foi utilizado para projetar os filtros que foram implementados usando o DSP. Para projetar os filtros foi utilizada a ferramenta *fdatool* (figura 3) do Matlab, ela foi acessada a partir da ferramenta *sptool* (figura 4) do Matlab,

1 – Matlab é marca registrada da Mathworks™

escolhendo-se a opção *New* em *filters*. Para acessar a ferramenta *sptool* basta digitar *sptool* na linha de comando do Matlab, a ferramenta *fdatool* também pode ser acessada desse modo, no entanto preferiu-se não fazê-lo, posteriormente será explicado porque foi feita essa escolha.

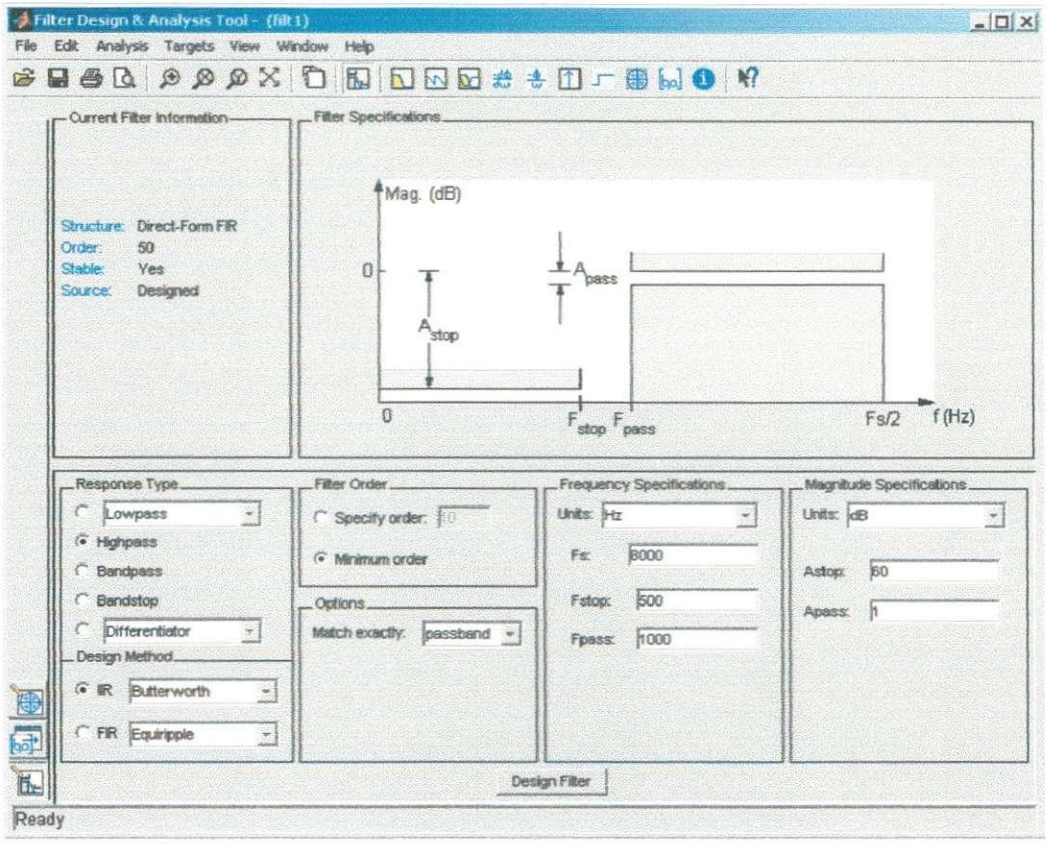



**Figura 3**: Ferramenta *fdatool* do Matlab

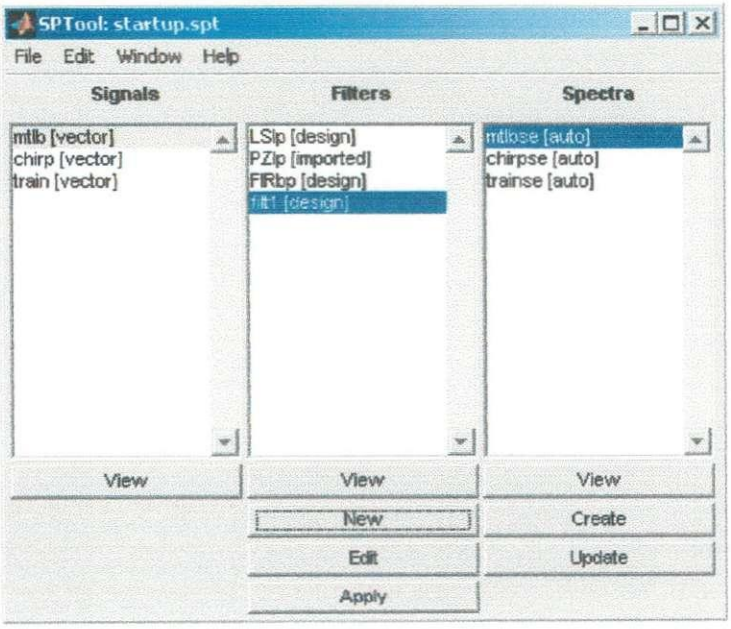



**Figura 4**: Ferramenta *sptool* do Matlab

- **Scope**

Scope é um programa *freeeware* desenvolvido por Christian Zeitnitz (zeitnitz.de/Christian/Scope/Scope_en.html) que funciona como osciloscópio utilizando a placa de som do computador, com ele é possível visualizar sinais tanto no tempo como na freqüência. Na figura 5 é mostrada a tela principal desse programa.

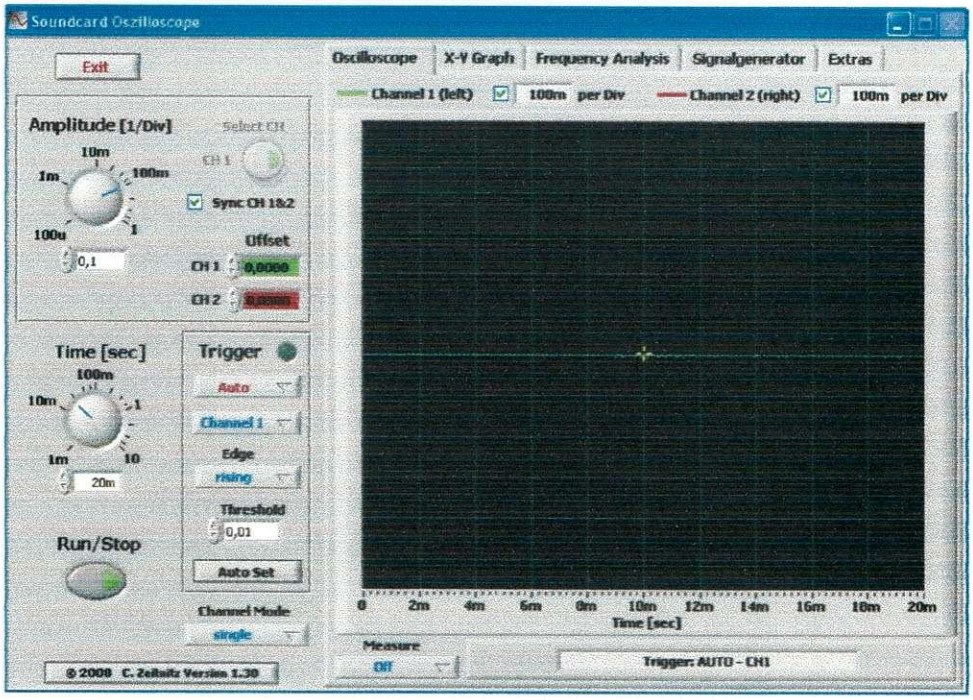



**Figura 5**: Tela do Scope

- **SigJenny**

SigJenny é um programa *freeware* desenvolvido pela empresa NaTCH Engineering (www.natch.co.uk) capaz de gerar diversos tipos de sinais, como senóides, ondas quadradas, triangulares e ruídos, o sinal gerado é enviado para a placa de som do computador. Na figura 6 é mostrada a tela do SigJenny.

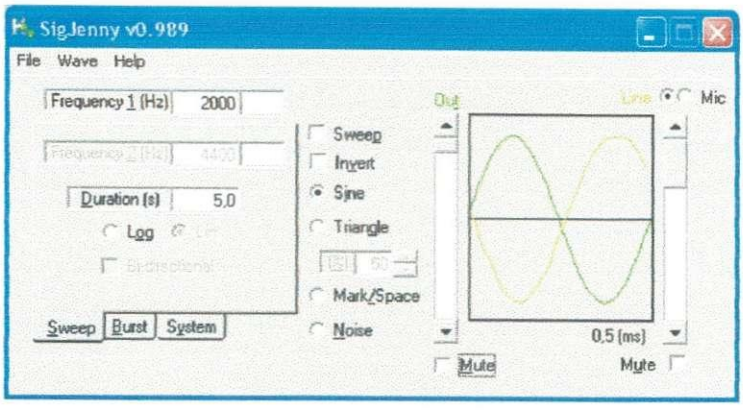



**Figura 6**: Tela do SigJenny

- **WinDSK**

WinDSK é uma ferramenta desenvolvida por Michael Morrow (morrow@ieee.org) que permite realizar diversas funções com DSPs da TI, como testar, gerar filtros e carregar programas, essa última foi a função utilizada durante o estágio. Na figura 7 é mostrada a tela do WinDSK.

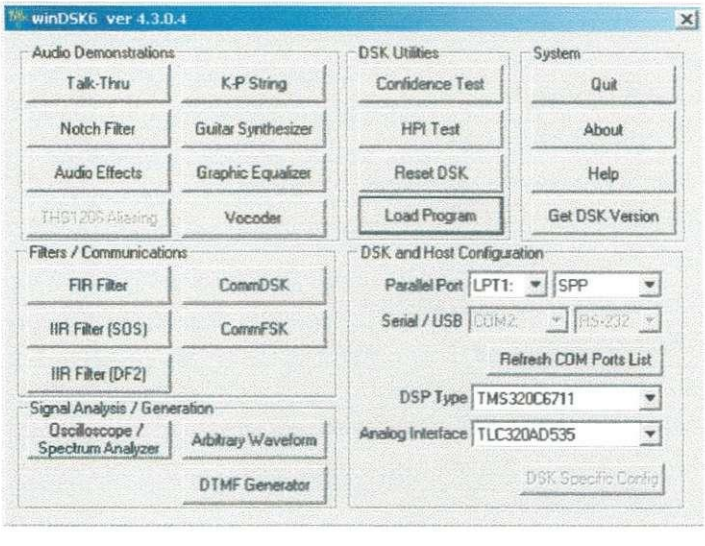



**Figura 7**: Tela do WinDSK

No próximo capítulo será abordado o tema de filtros digitais recursivos, sua definição e os tipos de aproximações existentes.

# 3 – Filtros Digitais Recursivos

Sistemas digitais recursivos são conhecidos também como sistemas com resposta ao impulso infinita (IIR – *Infinite Impulse Response*), o mesmo vale para os filtros, de certo modo todo sistema pode ser interpretado como sendo um filtro.

A equação que descreve um filtro recursivo é (SHENOI, 2006):

$$\sum_{k=0}^{N} a(k)y(n-k) = \sum_{k=0}^{M} b(k)x(n-k); \qquad a(0)=1 \tag{1}$$

Desse modo a função de transferência é:

$$H(z) = \frac{\sum_{k=0}^{M} b(k)z^{-k}}{\sum_{k=0}^{N} a(k)z^{-k}}; \qquad a(0)=1 \tag{2}$$

Os filtros digitais recursivos são divididos em Passa Alta, Passa Baixa, Passa Faixa e Rejeita Faixa e podem ser determinados a partir das aproximações de Butterworth, Chebyshev e Cauer (Elíptica), cada qual tem suas vantagens.

A aproximação de Butterworth apresenta curva de resposta plana na faixa de passagem, que é a faixa que se deseja preservar do sinal original, já na faixa de rejeição (faixa de freqüências que se deseja atenuar) ela cresce monotonicamente.

Com a aproximação de Chebyshev é possível obter valores de atenuação maiores para filtros da mesma ordem, isso é possível devido à oscilação presente na faixa de passagem.

Já a aproximação Elíptica permite que a faixa de transição entre a banda de passagem e a banda de rejeição seja bem estreita com um filtro de mesma ordem das outras aproximações, isso é possível porque há oscilação tanto na banda de passagem como na de rejeição (ANTONIOU, 1993).

No próximo capítulo são apresentados os experimentos realizados e os resultados obtidos nesses experimentos.

# 4 – Experimentos Realizados

Durante o estágio foram realizados alguns experimentos, utilizando como guia o livro de R. Chassaing citado nas referências bibliográficas. É possível programar o DSP em *assembly* ou em C, foi escolhida a última opção por ser mais simples.

Para obter um arquivo executável que pode ser carregado no DSP é necessário inicialmente criar um projeto usando o CCS e adicionar alguns arquivos necessários para o funcionamento correto do DSP, esses arquivos incluem rotinas para endereçamento de memória, rotinas para interrupção, entre outros. Para gerar o arquivo executável utiliza-se o comando *build*, após isso o arquivo *.out* já pode ser carregado no DSP e executado.

Com o CCS também é possível carregar o executável a partir do comando *load* e executá-lo usando o comando *run*, no entanto algumas vezes ocorrem problemas na conexão, por isso foi utilizado o programa WinDSK para gravar o programa, já que ele não apresenta esse tipo de problema.

## 4.1 – Geração de Sinais

O primeiro experimento realizado foi a geração de sinais, inicialmente foi usado o arquivo executável contido no CD que acompanha o livro de R. Chassaing, nessa parte a única tarefa feita foi carregar o arquivo e observar a saída, que era uma senóide.

Como segundo passo o código fonte do projeto foi alterado para gerar uma onda quadrada com freqüência de 1 kHz, esse código fonte é apresentado no apêndice (*quad.c*).

Para gerar a onda quadrada com essa freqüência foram utilizados oito valores de amplitude, quatro iguais a 400 e quatro iguais a 0, como o programa é executado uma vez a cada 125 us temos uma freqüência de 8/125 us = 1 kHz. Esses valores de amplitude citados são multiplicados por um fator de ganho, nas figuras de 8 a 10 é ilustrado o efeito desse valor na saída resultante. Como pode ser observado, com o valor de ganho igual a 10 há uma oscilação que diminui quando o ganho é aumentado para 20 e quase desaparece quando é igual a 30.

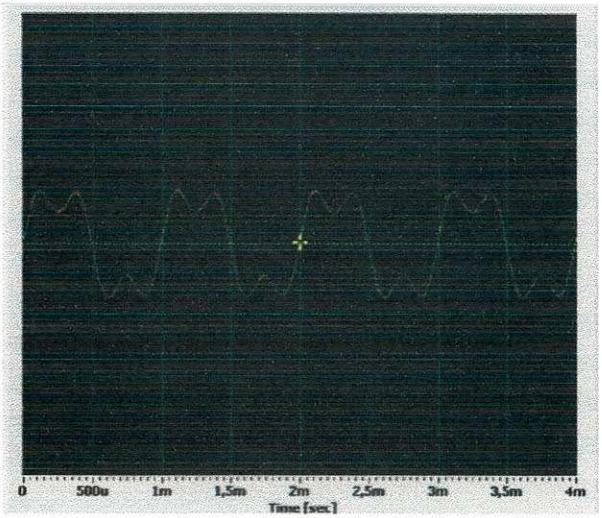



**Figura 8**: Onda quadrada com ganho igual a 10

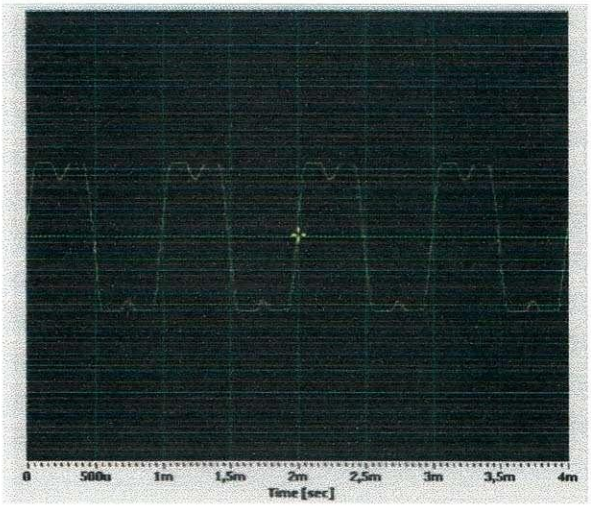



**Figura 9**: Onda quadrada com ganho igual a 20

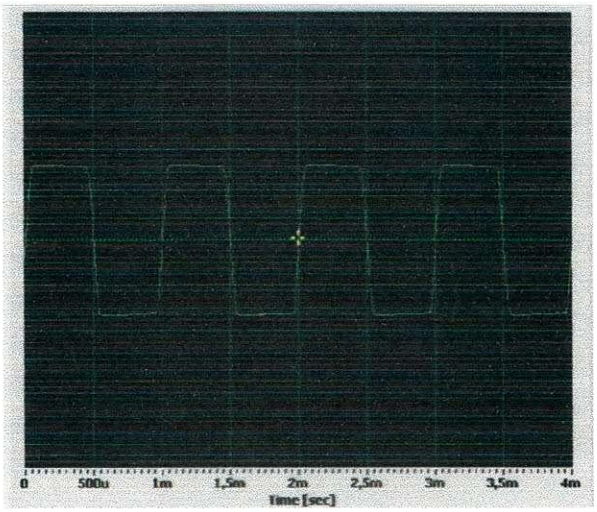



**Figura 10**: Onda quadrada com ganho igual a 30

## 4.2 – Amostragem de Sinais

O segundo experimento realizado foi de amostragem de sinais, esse experimento consistia em amostrar um sinal que estava na entrada do DSP e então enviá-lo de volta, depois essa saída seria visualizada utilizando o Scope, para gerar o sinal foi utilizado o SigJenny. A rotina desenvolvida por Chassaing (*loop_intr.c*) é mostrada no anexo. Na figura 11 é mostrada a saída visualizada quando o sinal de entrada é um seno com freqüência de 2 kHz e na figura 12 é mostrado o espectro da saída quando o sinal utilizado é uma onda quadrada de 1 kHz.

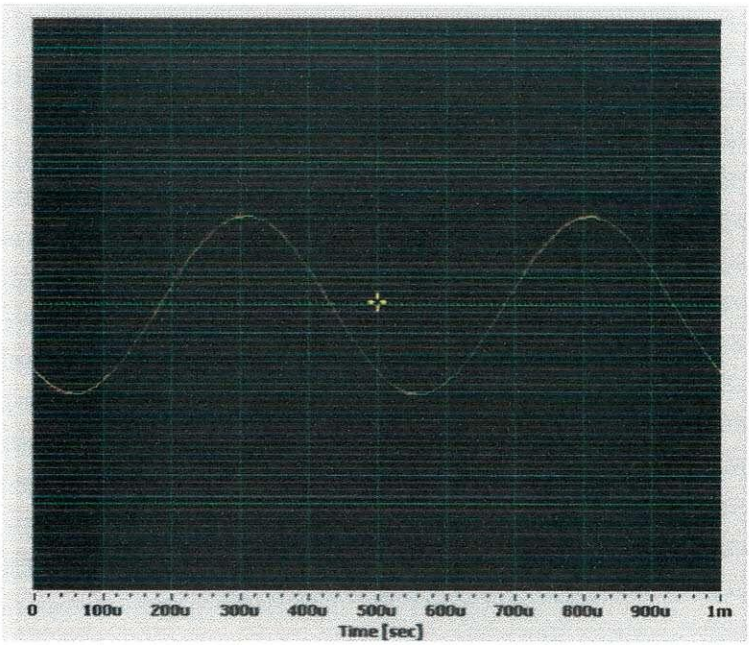



**Figura 11**: Saída com a entrada sendo um seno de 2 kHz

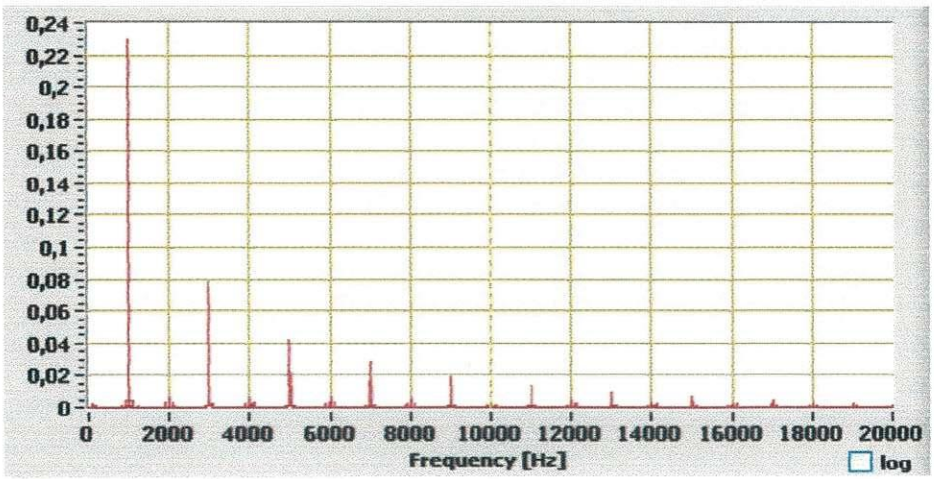



**Figura 12**: Espectro da saída com a entrada sendo uma onda quadrada de 1 kHz

## 4.3 – Filtragem de Sinais

Para realizar a filtragem de sinais foi utilizada a rotina desenvolvida por Chassaing mostrada no anexo, essa rotina utiliza os coeficientes de seções de segunda ordem para determinar a saída do filtro para uma determinada entrada, o que foi feito nesse experimento foram as especificações dos filtros e a determinação dos coeficientes das seções de segunda ordem dos filtros.

Para projetar os filtros foi usada a ferramenta *fdatool* do Matlab, ela é acessada através da ferramenta *sptool*, isso foi feito porque só é possível renomear o filtro desenvolvido a partir da janela da *sptool*.

Depois que os coeficientes do filtro são exportados para o *workspace* do Matlab usa-se a rotina *escreve_coeficiente.m* mostrada no apêndice para gerar o arquivo com os coeficientes das seções de segunda ordem do filtro, essa rotina utiliza os coeficientes do numerador e do denominador de um filtro nomeado como 'filtro', por isso a necessidade de renomear o filtro na janela do *sptool*.

Todos os filtros projetados foram de ordem 10 e com taxa de amostragem de 8 kHz, as outras especificações variaram conforme o tipo de filtro e a aproximação utilizada, já que para filtros com aproximação de Butterworth a ferramenta *fdatool* não permite que a atenuação na faixa de passagem seja diferente de 3 dB.

Foram implementados nove filtros, três para cada aproximação (Butterworth, Chebyshev e Elíptico) e para cada aproximação foi feito um filtro passa baixa, um passa alta e um passa faixa.

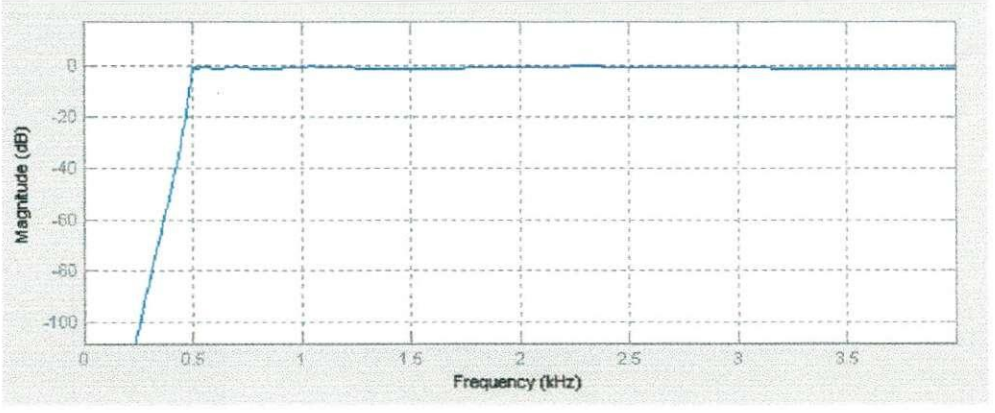
Os filtros passa altas tiveram freqüência de corte de 500 Hz, já nos passa faixa a banda de passagem foi de 300 Hz a 3400 Hz, essa é faixa do filtro de áudio utilizada em micro-filtros ADSL. Para os passa baixas a freqüência de corte foi de 3400 Hz, esse é o filtro utilizado em telefonia digital quando se utiliza modulação PCM (BELLAMY, 2000).

As atenuações utilizadas para o projeto dos filtros foram:

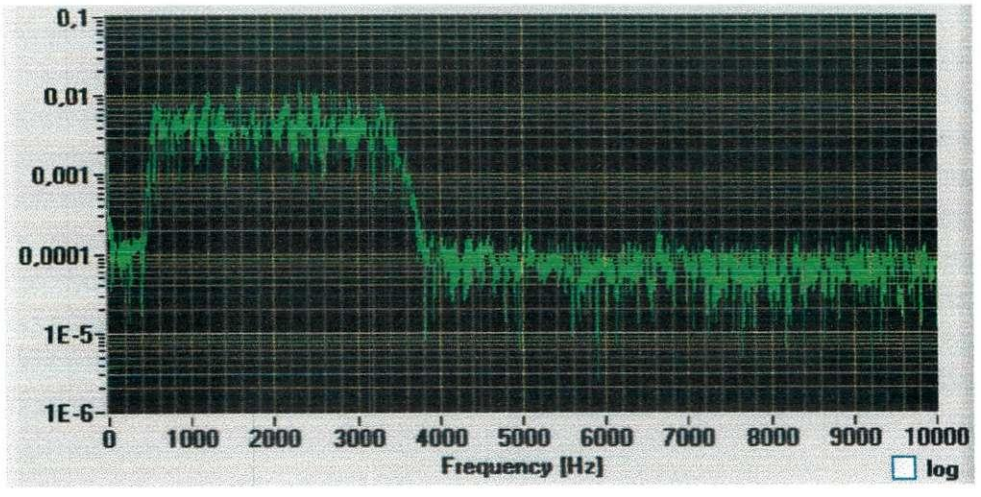
- Passa Alta Butterworth: $a_{passagem} = 3$ dB
- Passa Alta Chebyshev: $a_{passagem} = 1$ dB
- Passa Alta Elíptico: $a_{passagem} = 1$ dB, $a_{rejeição} = 60$ dB
- Passa Faixa Butterworth: $a_{passagem} = 3$ dB

- Passa Faixa Chebyshev: $a_{passagem}$ = 1 dB
- Passa Faixa Elíptico: $a_{passagem}$ = 1 dB, $a_{rejeição}$ = 20 dB
- Passa Baixa Butterworth: $a_{passagem}$ = 3 dB
- Passa Baixa Chebyshev: $a_{passagem}$ = 1 dB
- Passa Baixa Elíptico: $a_{passagem}$ = 3 dB

Nas figuras a seguir são mostradas as curvas de ganho dos filtros projetados e a saída do filtro quando a entrada é um ruído branco gaussiano, não são mostradas todas as curvas e saída já que os resultados são próximos. Na figura 13.a é mostrada a curva de ganho do filtro passa alta Chebyshev e na 13.b a resposta ao ruído; na 14.a é mostrada a curva de ganho do filtro passa faixa elíptico e na 14.b a resposta ao ruído; na 15.a é mostrada a curva de ganho do filtro passa baixa Butterworth e na 15.b a resposta ao ruído.

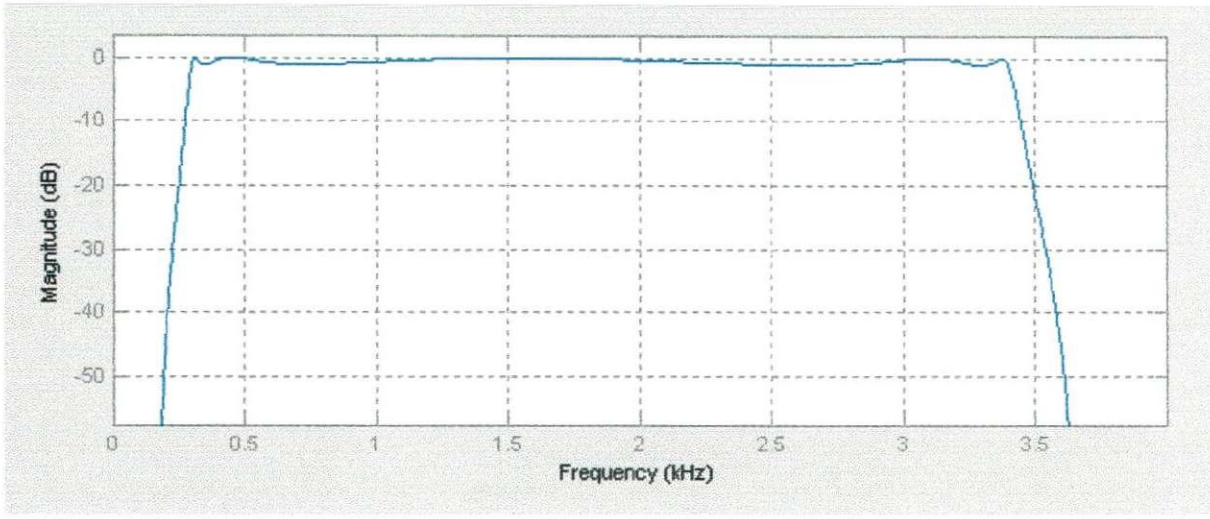



**(a)**

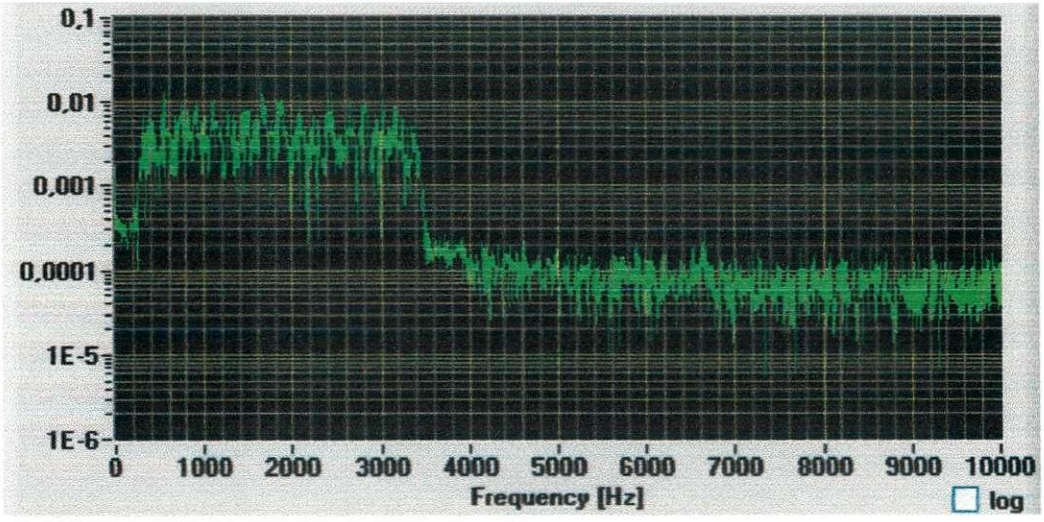



**(b)**

**Figura 13**: Filtro passa alta Chebyshev (a) Curva de ganho, (b) Resposta ao ruído

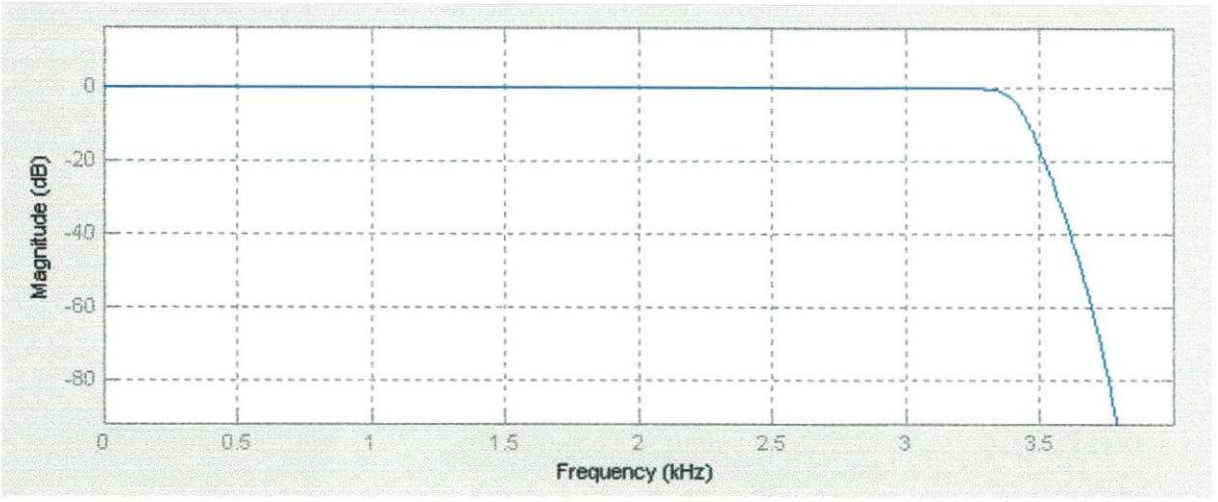



**(a)**

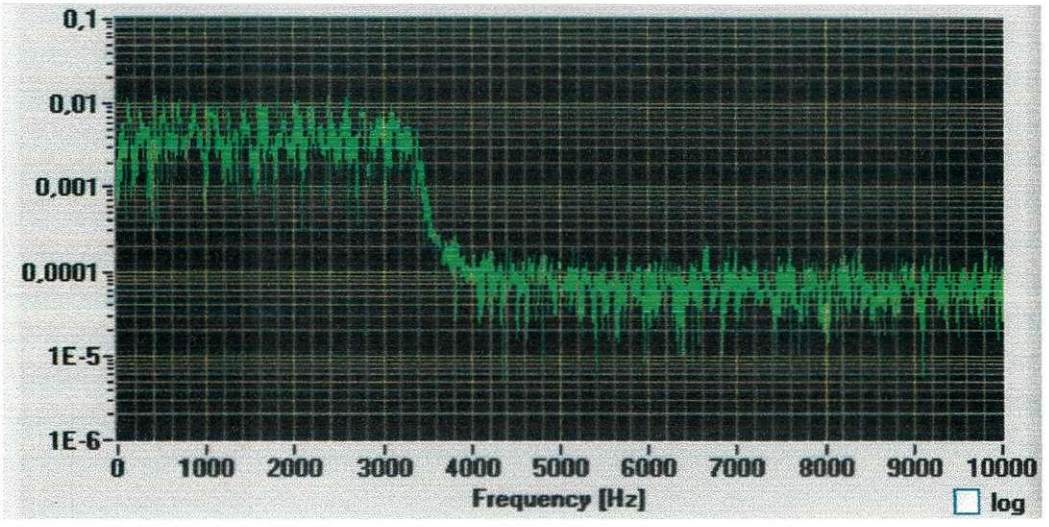



**(b)**

**Figura 14**: Filtro passa faixa Elíptico (a) Curva de ganho, (b) Resposta ao ruído



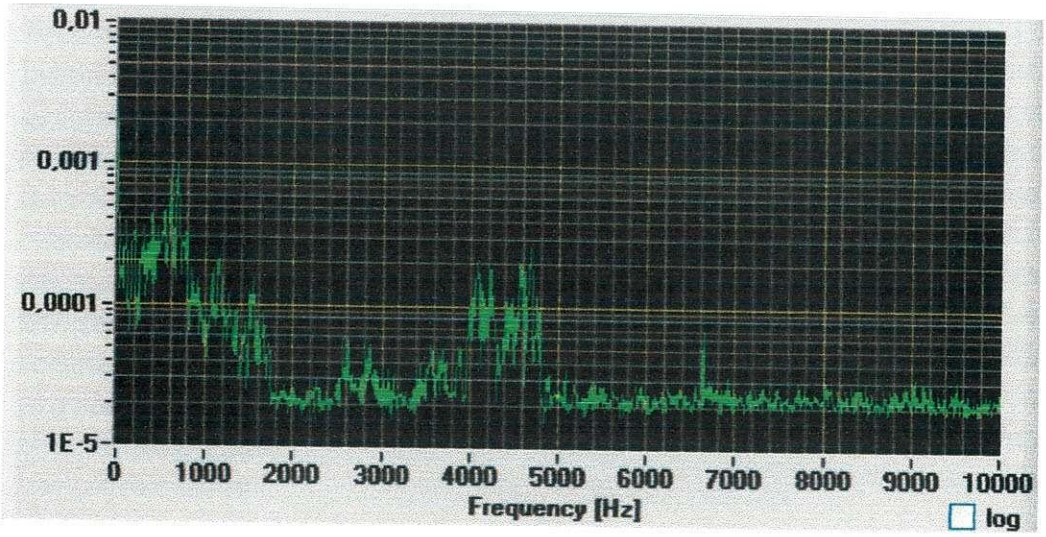

(a)



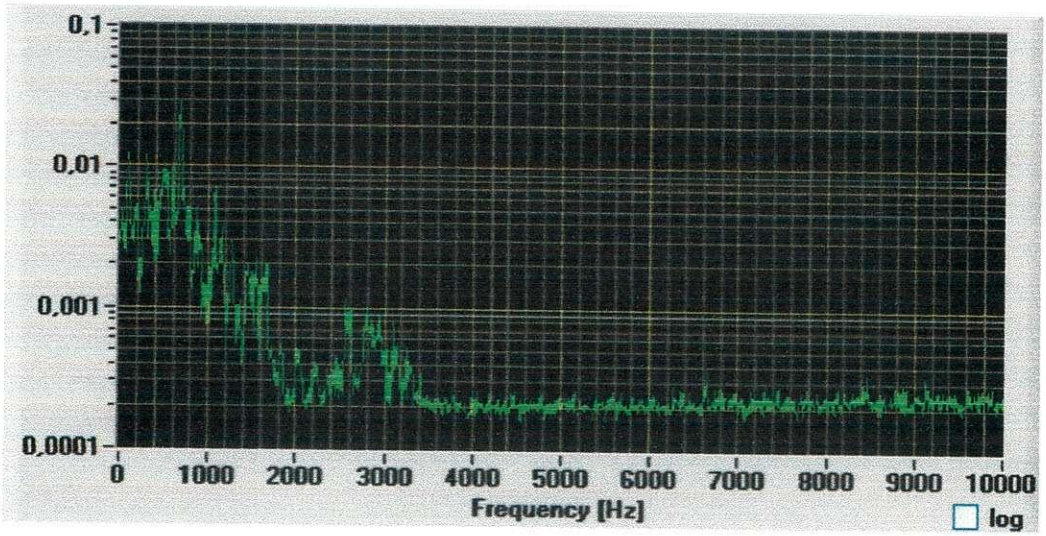

(b)

**Figura 15**: Filtro passa baixa Butterworth (a) Curva de ganho, (b) Resposta ao ruído

Esse filtro passa faixa também foi utilizado para filtrar um sinal de voz, na figura 16.a é mostrado o espectro desse sinal e na figura 16.b o sinal filtrado.




(a)




(b)

**Figura 16**: (a) Sinal de voz, (b) Sinal de voz filtrado

# 5 – Considerações Finais

Processamento digital de sinais é uma área extremamente importante atualmente e um dos meios de realizar esse tipo de processamento é utilizando os DSPs, que são dispositivos de *hardware* específicos para tal função.

Nesse estágio esses dispositivos foram utilizados para realizar geração, amostragem e filtragem de sinais, com os experimentos realizados foram adquiridos novos conhecimentos que podem ser úteis em diversas áreas.

No entanto os resultados obtidos teriam sido melhores se fossem disponíveis osciloscópios e geradores de sinais para realizar os experimentos, pois com isso não teríamos os erros inseridos pela placa de som.

A realização do estágio foi importante para que os conhecimentos teóricos adquiridos ao longo do curso fossem postos em prática e com isso perceber como esses conhecimentos são usados em aplicações do dia-a-dia.

# 6 – Referências

ANTONIOU, A. **Digital Filters: Analysis, Design and Applications**. 2ª ed. McGraw-Hill, 1993.

BELLAMY, J. C. **Digital Telephony**. 3ª ed. John Wiley & Sons, 2000.

CHASSAING, R. **DSP Applications Using C and the TMS320C6X DSK**. 1ª ed. John Wiley & Sons, 2002.

GILAT, A. **Matlab® com Aplicações em Engenharia**. 2ª Ed. Porto Alegre: Bookman, 2006.

SHENOI, B. A. **Introduction to Digital Signal Processing and Filter Design.** 1ª Ed. John Wiley & Sons. 2006.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. **Discrete-Time Signal Processing**. 2ª ed. Prentice Hall. 1999.

SMITH, S. W. **Digital Signal Processing: A Practical Guide for Engineers and Scientists.** 1ª ed. Newnes. 2003.

# 7 – Apêndice

## quad.c

```
// Rotina para gerar onda quadrada baseada na rotina para
// gerar um senóide disponível em Chassaing, 2002
short loop = 0;
// Valores de amplitude da onda quadrada
short quad_table[8] = {400, 400, 400, 400, 0, 0, 0, 0};

short cont=1;
short amplitude = 20; // Ganho

 interrupt void c_int11()  //interrupt service routine
{
 output_sample(quad_table[loop]*amplitude); // Onda quadrada
 if (loop < 7) ++loop;     //increment index loop
 else loop = 0;            //reinit index @ end of buffer
 return;                   //return from interrupt
}

void main()
{
  comm_intr();             //init DSK, codec, McBSP
  while(1);
}
```

## escreve_coeficiente.m

```
%*****************************************************************
% UFCG/CEEI/DEE/LAPS
% Rotina desenvolvida por Johannes Dantas
% e-mail: johannesee@gmail.com
%*****************************************************************
%
% Gera um arquivo com os coeficientes do filtro para ser incluído no
% projeto do filtro para o DSP, para isso utiliza os coeficientes do
% filtro que são encontrados usando a ferramenta do Matlab fdatool e
% exportados a partir da ferramenta sptool, o nome do filtro projetado
% usando a fdatool deve ser 'filtro' (sem as aspas)
%
% Os coeficientes 'B' do Matlab correspondem aos coeficientes 'a' do
% DSP e vice-versa
%
% Modelo de arquivo
%
% #define stages 5
%
% int a[stages][3]=       {     //numerator coefficients
% {27940, -10910, 27940},
% {32768, -11841, 32768},
% {32768, -13744, 32768},
```

```
% {32768, -11338, 32768},
% {32768, -14239, 32768}  };
%
% int b[stages][2]=       {     //denominator coefficients
% {-11417, 25710},
% {-9204, 31581},
% {-15860, 31605},
% {-10221, 32581},
% {-15258, 32584}         };

% Cálculo dos coeficientes
%
% Converte a função de transferência para pólos e zeros
[z,p,k] = tf2zp(filtro.tf.num, filtro.tf.den);

% Acha a matriz de estágios de segunda ordem
sec_ord_sec = zp2sos(z, p, k);

% Arredonda e escalona os coeficientes
sec_ord_sec = round(sec_ord_sec*2^15);

% Determina o número de estágios de 2ª ordem
tam = size(sec_ord_sec);
n_estagios = tam(1,1);

fid = fopen('filtro.cof', 'w'); % Cria o arquivo

% Preenche o arquivo
%
% Número de estágios de segunda ordem
fprintf(fid, '#define stages %d\n\n', n_estagios);

% Coeficientes do numerador
fprintf(fid, 'int a[stages][3]=       {     //numerator coefficients\n');

for ii = 1 : n_estagios - 1,
    fprintf(fid,'{%d, %d, %d}, \n', sec_ord_sec(ii, 1), sec_ord_sec(ii, 2),
sec_ord_sec(ii, 3));
end

% Na última linha não tem a vírgula depois do '}'
ii = ii + 1;
fprintf(fid,'{%d, %d, %d}  }; \n\n', sec_ord_sec(ii, 1), sec_ord_sec(ii,
2), sec_ord_sec(ii, 3));

% Coeficientes do denominador
fprintf(fid, 'int b[stages][2]=       {    //*denominator coefficients\n');

for ii = 1 : n_estagios - 1,
    fprintf(fid,'{%d, %d}, \n', sec_ord_sec(ii, 5), sec_ord_sec(ii, 6));
end

% Na última linha não tem a vírgula depois do '}'
ii = ii + 1;
fprintf(fid,'{%d, %d}  }; \n\n', sec_ord_sec(ii, 5), sec_ord_sec(ii, 6));

fclose(fid); % Fecha o arquivo
```

# 8 – Anexo

## loop_intr.c

```
//Loop_intr.c Loop program using interrupt, output=input
//Comm routines and support files included in C6xdskinit.c
interrupt void c_int11()         //interrupt service routine
{
   int sample_data;

   sample_data = input_sample(); //input data

   output_sample(sample_data*100);    //output data

   return;
}

void main()
{
  comm_intr();                    //init DSK, codec, McBSP
  while(1);                      //infinite loop
}
```

## IIR.c

```
//IIR.c  IIR filter using cascaded Direct Form II
//Coefficients a's and b's correspond to b's and a's from MATLAB
#include "filtro.cof"
short dly[stages][2] = {0};     //delay samples per stage

interrupt void c_int11()      //ISR
{
 int i, input;
 int un, yn;

 input = input_sample();      //input to 1st stage
 for (i = 0; i < stages; i++)  //repeat for each stage
  {
   un=input-((b[i][0]*dly[i][0])>>15) - ((b[i][1]*dly[i][1])>>15);
yn=((a[i][0]*un)>>15)+((a[i][1]*dly[i][0])>>15)+((a[i][2]*dly[i][1])>>15);
   dly[i][1] = dly[i][0];     //update delays
   dly[i][0] = un;              //update delays
   input = yn;                //intermediate output->input to next stage
  }
   output_sample(yn);           //output final result for time n
   return;                 //return from ISR
}

void main()
{
  comm_intr();                    //init DSK, codec, McBSP
  while(1);               //infinite loop
}
```