Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Digitale Signalverarbeitung
Informations- & Elektrotechnik
Semestergruppe E6

Protokoll zum 3. Praktikum

FIR-Filter

Versuchstag
Di, 28. November, 2012

Teilnehmer:
Michael Langen, Markus Wacker

Professor:
Prof. Dr. Jürgen Vollmer

Inhaltsverzeichnis

1 Vorbereitung

1.1 Tiefpassfilter Abschätzung der Ordnung

Zur Vorbereitung auf das Praktikum sollte zunächst ein Tiefpassfilter entworfen werden. Die Eckdaten des FIR-Tiefpassfilters lauten:

Eckfrequenz Durchlassbereich 1800 Hz
Eckfrequenz Sperrbereich 2600 Hz
Maximaler Ripple im Durchlassbereich 0.5 dB
Minimale Sperrdämpfung 40 dB
Abtastfrequenz 8 kHz

Mit dem in der Aufgabenbeschreibung gegebenen MATLAB-Skript wurden nun die Ordnung des Filters abgeschätzt. Damit ergibt sich laut Kaiser_Order_01.m folgender MATLAB Konsolenausdruck:

```
*** Filter Order ***
Estimated filter order is 23
```

Die Ordnung des Filters beträgt damit nach Abschätzung mit der Kaiser-Formel N = 23.

1.2 Tiefpassfilter Entwurf mit fir1

Das gegebene MATLAB-Skript wurden nun wie folgt modifiziert. Durch nicht Angabe einer Fenster-Funktion verwendet MATLAB standardmäßig das Hamming Window. Weiterhin wurde die Ordnungszahl laut der vorherigen Formel angepasst und die Grenzfrequenz auf den Wert der Aufgabenstellung gesetzt. Um die Filterkoeffizienten zu skalieren wurden diese mit dem Wert von 2^{15} multipliziert um die maximale Auflösung zu erhalten.

```
%% FIR Lowpass
% authors: Markus Wacker, Michael Langen
% date: 01.12.2012

clear all;
clc;

Fs=8e3;          % Specify Sampling Frequency
Ts=1/Fs;         % Sampling period.
Ns=512;          %No of time samples to be plotted.
t=[0:Ts:Ts*(Ns-1)]; %Make time array that contains Ns elements

%create sampled sinusoids at different frequencies
f1=500; f2=1800; f3=2000; f4=3200;
x1=sin(2*pi*f1*t); x2=sin(2*pi*f2*t); x3=sin(2*pi*f3*t); x4=sin(2*pi*f4*t);
x=x1+x2+x3+x4; %Calculate samples for a 4-tone input signal

N=22; %FIR1 requires filter order (N) to be EVEN when gain = 1 at Fs/2.
W=[0.45]; %Specify lowpass filter
B=fir1(N,W); %Design FIR Filter using default Hamming window.

%create header file fir_coef.h (FIR filter coefficients)
filnam = fopen('LP_coeff.h', 'w'); % generate include-file
fprintf(filnam,'#define N %d\n', N+1);
fprintf(filnam,'short h[N]={\n');
j = 0;
for i= 1:N+1;
fprintf(filnam,' %6.0f,', B(i)*2^15);
j = j + 1;
if j >7
fprintf(filnam, '\n');
j = 0;
end
end
fprintf(filnam,'};\n');
fclose(filnam);
A=1; %FIR filters have no poles, only zeros.
grid on;
figure(1)
freqz(B,A); %Plot frequency response - both amp and phase response.

figure(2); %Create a new figure window, so previous one isn't lost.
subplot(2,1,1); %Two subplots will go on this figure window.
```

```
Npts=200;
plot(t(1:Npts),x(1:Npts)) %Plot first Npts of this 4-tone input signal
title('Time Plots of Input and Output');
xlabel('time (s)'); ylabel('Input Sig');

%Now apply this filter to the 4-tone test sequence
y = filter(B,A,x);
subplot(2,1,2); %Now go to bottom subplot.
plot(t(1:Npts),y(1:Npts)); %Plot first Npts of filtered signal.
xlabel('time (s)'); ylabel('Filtered Sig');
figure(3); %Create a new figure window, so previous one isn't lost.
subplot(2,1,1);
xfftmag=(abs(fft(x,Ns))); %Compute spectrum of input signal.
xfftmagh=xfftmag(1:length(xfftmag)/2);

%Plot only the first half of FFT, since second half is mirror imag
f=[1:1:length(xfftmagh)]*Fs/Ns; %Make freq array from 0 Hz to Fs/2 Hz.
plot(f,xfftmagh); %Plot frequency spectrum of input signal
title('Input and Output Spectra');
xlabel('freq (Hz)'); ylabel('Input Spectrum');
subplot(2,1,2);
yfftmag=(abs(fft(y,Ns)));
yfftmagh=yfftmag(1:length(yfftmag)/2);

%Plot only the first half of FFT, since second half is mirror image
plot(f,yfftmagh); %Plot frequency spectrum of input signal
xlabel('freq (Hz)'); ylabel('Filt Sig Spectrum');
```

Somit ergibt sich die in Abbildung **??** gezeigte Filtercharakteristik.

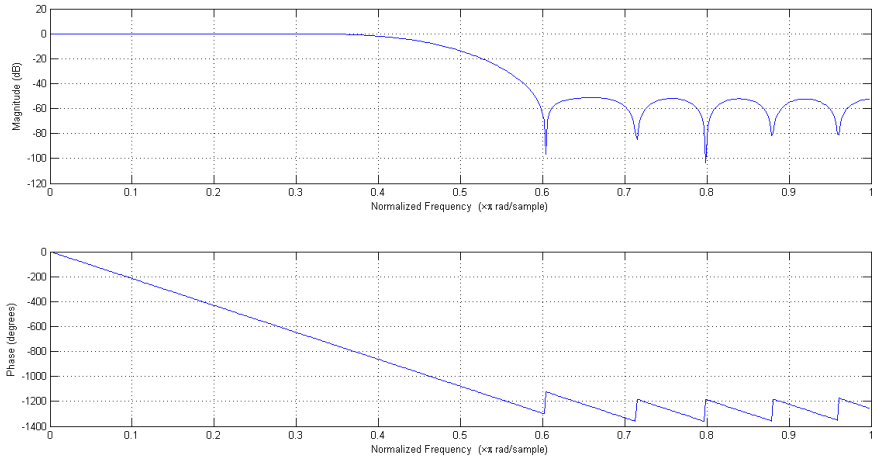

Abbildung 1: Frequenzgang Tiefpassfilter mit fir1()

Erkennbar ist ein typisches Tiefpassverhalten. Auch ist der Abbildung zu entnehmen, das die Sperrdämpfung von mindestens 40 dB eingehalten wird und keinen sichtbaren Ripple im Durchlassbereich besitzt. Der Filter erfüllt damit alle geforderten Spezifikationen.

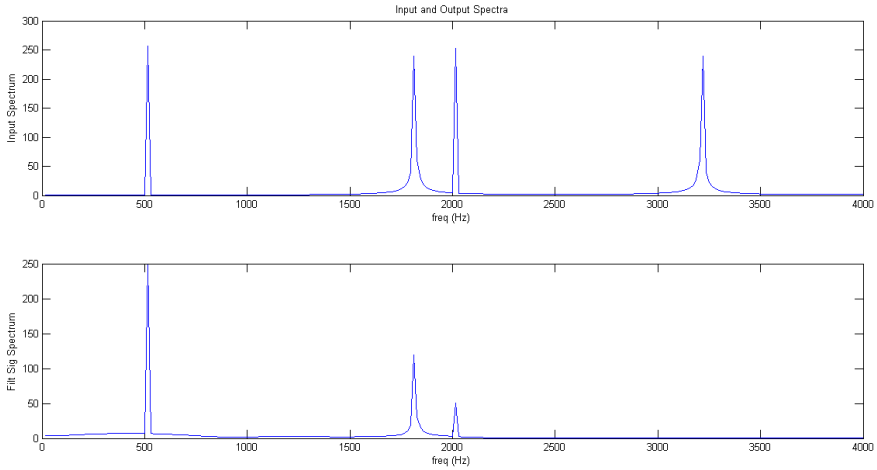

Abbildung 2: Spektrukm Tiefpassfilter mit fir1()

In Abbildung **??** ist das Verhalten des Filters bei Sinussignalen mit verschiedenen Frequenzen erkennbar. Dabei ist ersichtlich, dass Signalanteile nach der Grenzfrequenz stark bedämpft werden. Allerdings werden auch die Signalanteile bei der Eckfrequenz von 1800 Hz schon recht stark bedämpft. Was auch bereits in Abbildung **??** erkennbar ist, denn hier hat der Amplitudengang bei der eigentlichen Eckfrequenz bereits eine recht hohe Dämpfung.

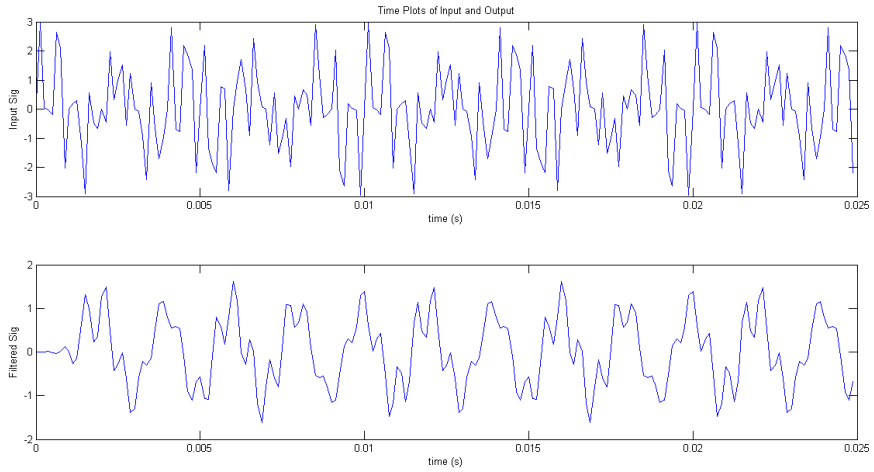

Abbildung 3: Zeitsignal Tiefpassfilter mit fir1()

Das Verhalten im Frequenzbereich (filtern der hohen Frequenzen) zeigt sich natürlich auch im Zeitsignal. In Abbildung **??** ist erkennbar, dass das Signal deutlich weniger hohe Frequenzanteile besitzt-.
Damit ergeben sich mit dem MATLAB-Skript folgenden Koeffizienten.

```
#define N 23
short h[N]={
    12,    103,     28,   -295,   -235,    666,    891,  -1118,
 -2600,   1491,  10087,  14712,  10087,   1491,  -2600,  -1118,
   891,    666,   -235,   -295,     28,    103,     12,};
```

1.3 Tiefpassfilter Entwurf mit firpm

Das vorher genutzt MATLAB-Skript wurde nun dahingehend verändert, sodass nun mit der Funktion firpm() und firpmord() der Tiefpassfilter neu dimensioniert wurde.

```
%% FIR Lowpass
% authors: Markus Wacker, Michael Langen
% date: 01.12.2012

clear all;
clc;

fs = 8000;          % Sampling frequency
f = [1800 2600];      % Cutoff frequencies
a = [1 0];          % Desired ampl
rp = 0.5;             % Passband ripple
rs = 40;            % Stopband ripple
Ts=1/fs; %Sampling period.
ns=512; %no of time samples to be plotted.
t=[0:Ts:Ts*(ns-1)]; %Make time array that contains ns elements

%create sampled sinusoids at different frequencies
f1=500; f2=1800; f3=2000; f4=3200;
x1=sin(2*pi*f1*t); x2=sin(2*pi*f2*t); x3=sin(2*pi*f3*t); x4=sin(2*pi*f4*t);
x=x1+x2+x3+x4; %Calculate samples for a 4-tone input signal

dev = [(10^(rp/20)-1)/(10^(rp/20)+1)  10^(-rs/20)]; % Compute deviations
[n,fo,ao,w] = firpmord(f,a,dev,fs);
n = n+2;
B = firpm(n,fo,ao,w);

%create header file fir_coef.h (FIR filter coefficients)
filnam = fopen('LP_coeff_firpm.h', 'w'); % generate include-file
fprintf(filnam,'#define N %d\n', n+1);
fprintf(filnam,'short h[N]={\n');
j = 0;
for i= 1:n+1;
fprintf(filnam,' %6.0f,',B(i)*2^15 );
j = j + 1;
if j >7
fprintf(filnam, '\n');
j = 0;
end
end
fprintf(filnam,'};\n');
fclose(filnam);
A=1; %FIR filters have no poles, only zeros.
grid on;
figure(1)
freqz(B,A); %Plot frequency response - both amp and phase response.

%; %User must hit any key on PC keyboard to continue.
figure(2); %Create a new figure window, so previous one isn't lost.
subplot(2,1,1); %Two subplots will go on this figure window.
npts=200;
plot(t(1:npts),x(1:npts)) %Plot first npts of this 4-tone input signal
title('Time Plots of Input and Output');
xlabel('time (s)'); ylabel('Input Sig');

%now apply this filter to the 4-tone test sequence
y = filter(B,A,x);
subplot(2,1,2); %now go to bottom subplot.
plot(t(1:npts),y(1:npts)); %Plot first npts of filtered signal.
xlabel('time (s)'); ylabel('Filtered Sig');
figure(3); %Create a new figure window, so previous one isn't lost.
subplot(2,1,1);
xfftmag=(abs(fft(x,ns))); %Compute spectrum of input signal.
xfftmagh=xfftmag(1:length(xfftmag)/2);

%Plot only the first half of FFT, since second half is mirror imag
f=[1:1:length(xfftmagh)]*fs/ns; %Make freq array from 0 Hz to fs/2 Hz.
plot(f,xfftmagh); %Plot frequency spectrum of input signal
title('Input and Output Spectra');
xlabel('freq (Hz)'); ylabel('Input Spectrum');
subplot(2,1,2);
yfftmag=(abs(fft(y,ns)));
yfftmagh=yfftmag(1:length(yfftmag)/2);

%Plot only the first half of FFT, since second half is mirror image
plot(f,yfftmagh); %Plot frequency spectrum of input signal
xlabel('freq (Hz)'); ylabel('Filt Sig Spectrum');
```

Mit diesem MATLAB-Skript ergibt sich nun folgende Filtercharakteristik (Abbildung **??**).

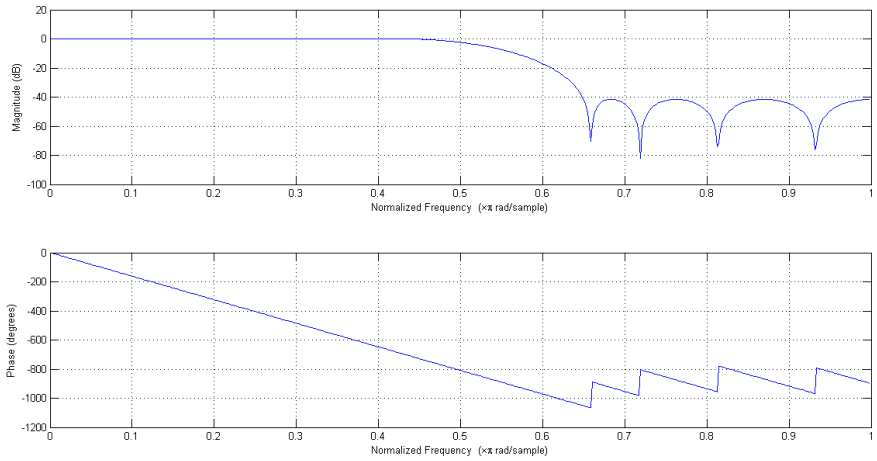

Abbildung 4: Frequenzgang Tiefpassfilter mit firpm()

In Abbildung **??** ist erkennbar, dass der Amplitudengang insgesamt eine höhere Eckfrequenz besitzt als der Amplitudengang in Abbildung **??**. Trotzdem wird die nötige Sperrdämpfung von mindestens 40 dB erreicht und der Ripple im Durchlassbereich überschreitet nicht 0.5 db.

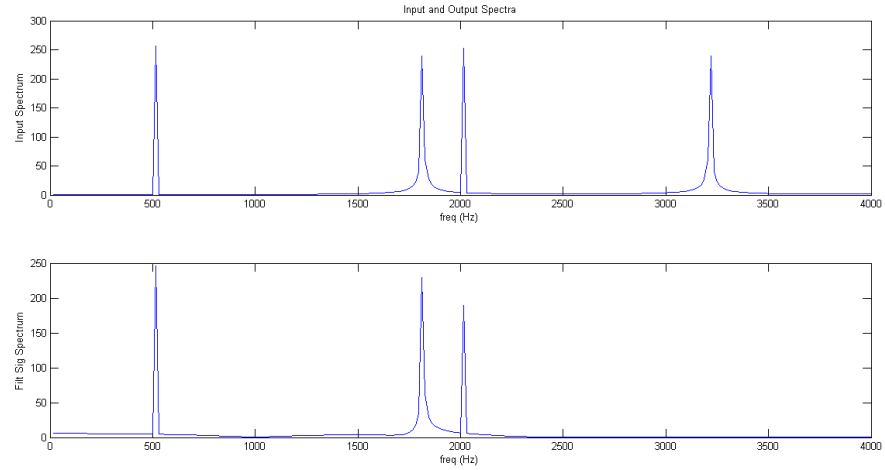

Abbildung 5: Spektrukm Tiefpassfilter mit firpm()

Durch die höhere Eckfrequenz des Filters sind nun die Signalanteile, welche sehr nah an der Grenzfrequenz liegen, weniger stark bedämpft (Abbildung **??**) als in Abbildung **??** dargestellt. Trotzdem wird der Signalanteil bei ca. 3200 Hz komplett unterdrückt. Allerdings ist der Signalanteil im Übergangsbereich bei ca. 2 kHz hier noch stärker vorhanden als bei dem vorherigen Entwurf. Der Filterentwurf stellt also oft ein Kompromiss dar.

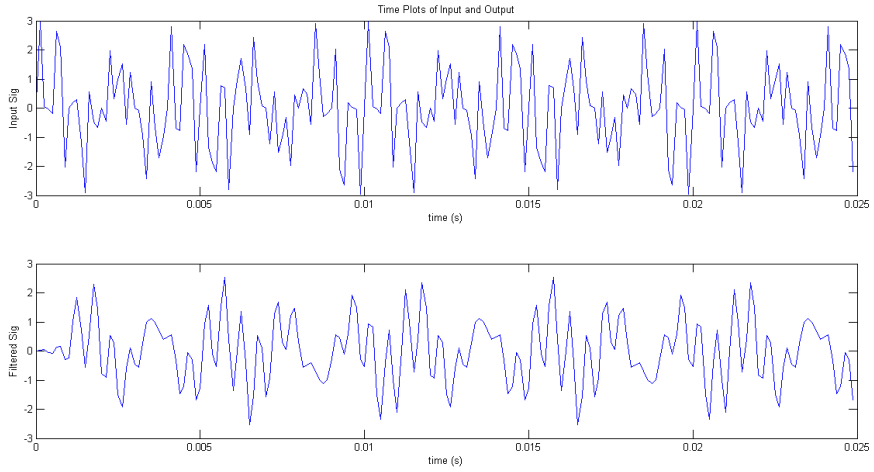

Abbildung 6: Zeitsignal Tiefpassfilter mit firpm()

In Abbildung **??** ist das Zeitsignal dargestellt, welches wie erwartet nur noch niedrige Frequenzanteile enthält.
Die skalierten Koeffizienten laut MATLAB sind damit:

```
#define N 19
short h[N]={
    292,    539,   -523,   -723,   1220,   1031,  -2887,  -1247,
  10218,  17713,  10218,  -1247,  -2887,   1031,   1220,   -723,
   -523,    539,    292,};
```

Insgesamt ist erkennbar, dass die Filtercharakteristik mit der MATLAB-Funktion firpm besser angenähert wurde. Es wurden weniger Koeffizienten verwendet (19 im Vergleich zu 23) und die Spezifikation wurden trotzdem eingehalten. Das Verhalten bis zur Eckfrequenz ist sogar deutlich besser, denn es werden vorher keine Signalanteile stark unterdrückt.

1.4 Bandpassfilter Entwurf mit firpm

Nach dem Tiefpassfilter aus der vorherigen Aufgabe sollte nun ein Bandpassfilter mit folgenden Werten entworfen werden:

Durchlassbereich liegt zwischen 800 Hz und 2400 Hz
Übergangsbereich soll 600 Hz betragen
Maximaler Ripple im Durchlassbereich 0.4 dB
Minimale Sperrdämpfung 40 dB
Abtastfrequenz 8 kHz

Auch diese Dimensionierung wurde mit der MATLAB-Funktion firpm() durchgeführt. Dafür wurde das MATLAB-Skript wie folgt geändert(hier nur verkürzte Form gezeigt, weil die Plot-Funktion identisch mit der Vorgänger Version):

```
%% FIR Lowpass
% authors: Markus Wacker, Michael Langen
% date: 01.12.2012

clear all;
clc;

Fs=8e3; %Specify Sampling Frequency
Ts=1/Fs; %Sampling period.
Ns=512; %No of time samples to be plotted.
t=[0:Ts:Ts*(Ns-1)]; %Make time array that contains Ns elements

f1=500; f2=1800; f3=2000; f4=3200;
```

```
x1=sin(2*pi*f1*t); x2=sin(2*pi*f2*t); x3=sin(2*pi*f3*t); x4=sin(2*pi*f4*t);
x=x1+x2+x3+x4; %Calculate samples for a 4-tone input signal

rp = 0.4;              % Passband ripple
rs = 0.4;             % Stopband ripple

f = [500 800 2400 2700];     % Cutoff frequencies
a = [0 1 0];           % Desired amplitudes
dev = [10^(-rs/20),(10^(rp/20)-1)/(10^(rp/20)+1),10^(-rs/20)];
[N,fo,mo,w] = firpmord( f, a, dev, Fs );
N = N+70;
B = firpm(N,fo,mo,w);

%and so on
```

Allerdings konnten die Anforderungen nur durch eine stark erhöhte Anzahl an Koeffizienten eingehalten werden. In unserem Fall mussten es mind. 86 Koeffizienten sein, damit die Sperrdämpfung erreicht wird. Die abgeschätzte Ordnung von der MATALB Funktion firpmord() war viel zu gering. Damit ergibt sich folgender Frequenzgang (Abbildung **??**).

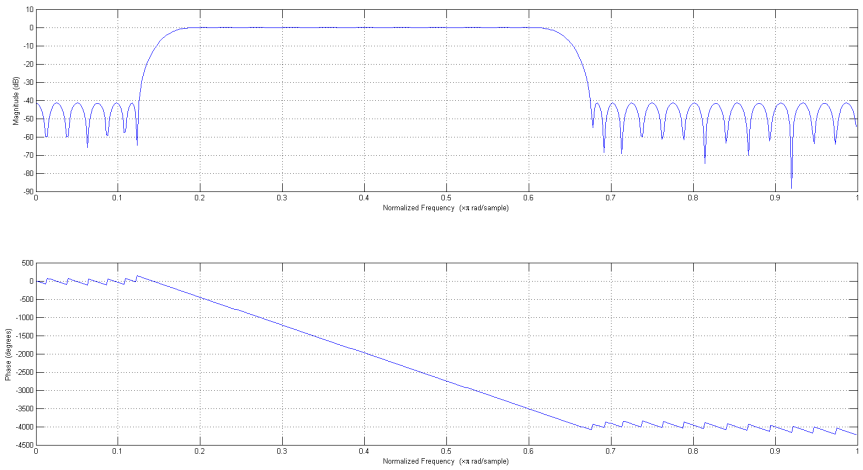

Abbildung 7: Frequenzgang Bandpassfilter

In den Spektren der Sinussignale sollten nun alle Signalanteile im Durchlassbereich nahezu ungedämpft durchgelassen werden und alle anderen Anteile komplett unterdrückt werden. Dieses erwartete Verhalten ist in Abbildung **??** gezeigt.

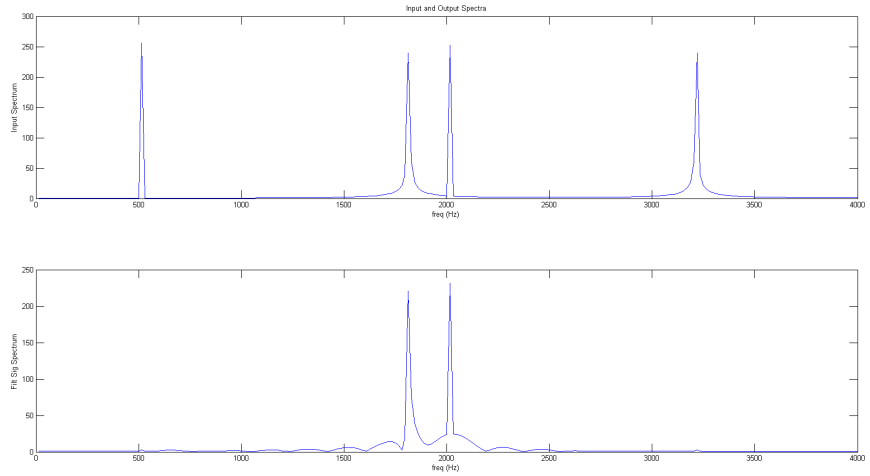

Abbildung 8: Spektrum Bandpassfilter

Die Veränderung im Spektrum führt zu einer geänderten Zeitfunktion, welche dann in Abbildung **??** dargestellt ist.

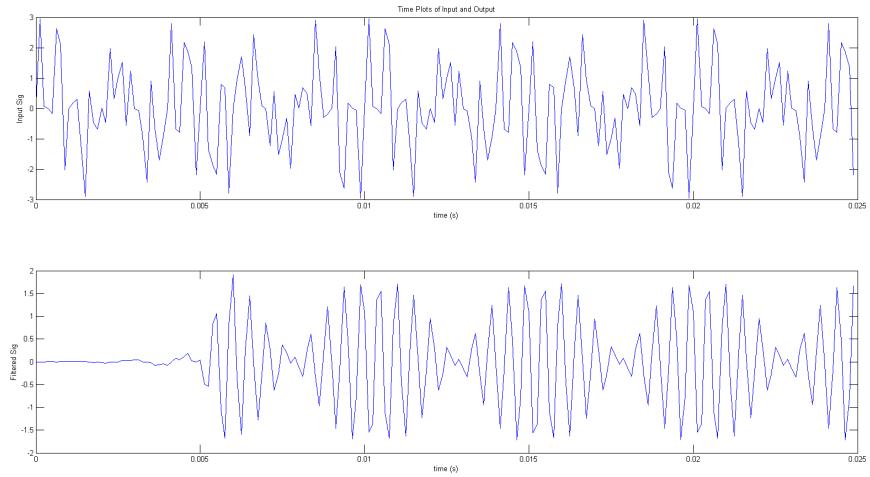

Abbildung 9: Zeitsignal Bandpassfilter

Die mit dem MATLAB Skript berechneten Koeffizienten lauten somit:

```
#define N 86
short h[N]={
    -3,    -3,    19,    22,    -6,   120,   -13,    74,
   105,   -34,   -12,    38,  -171,  -145,     6,  -195,
  -102,   227,    27,    91,   502,   134,   -56,   339,
  -231,  -666,   -44,  -477,  -896,   321,   285,  -257,
  1377,  1375,  -182,  1268,   904, -2633, -1163,  -786,
 -7772, -3920, 12670, 12670, -3920, -7772,  -786, -1163,
 -2633,   904,  1268,  -182,  1375,  1377,  -257,   285,
   321,  -896,  -477,   -44,  -666,  -231,   339,   -56,
   134,   502,    91,    27,   227,  -102,  -195,     6,
  -145,  -171,    38,   -12,   -34,   105,    74,   -13,
   120,    -6,    22,    19,    -3,    -3,};
```

An den Koeffizienten des Filters lässt sich auch sehr gut die Symmetrie der Impulsantwort erkennen.

2 Praktikumsdurchführung

2.1 Analoge Übertragungscharakteristik DSK Boards

Nun sollte im Praktikum bei einer eingestellten Abtastfrequenz von 8 kHz vom Board der Frequenzgang des DSK Boards aufgenommen werden. Hierfür wurden im Spektrumanalyzer das vorgefertigte Set-File sweep-up-to-8khz.set geladen. Der aufgenommene Frequenzgang ist in Abbildung **??** dargestellt.

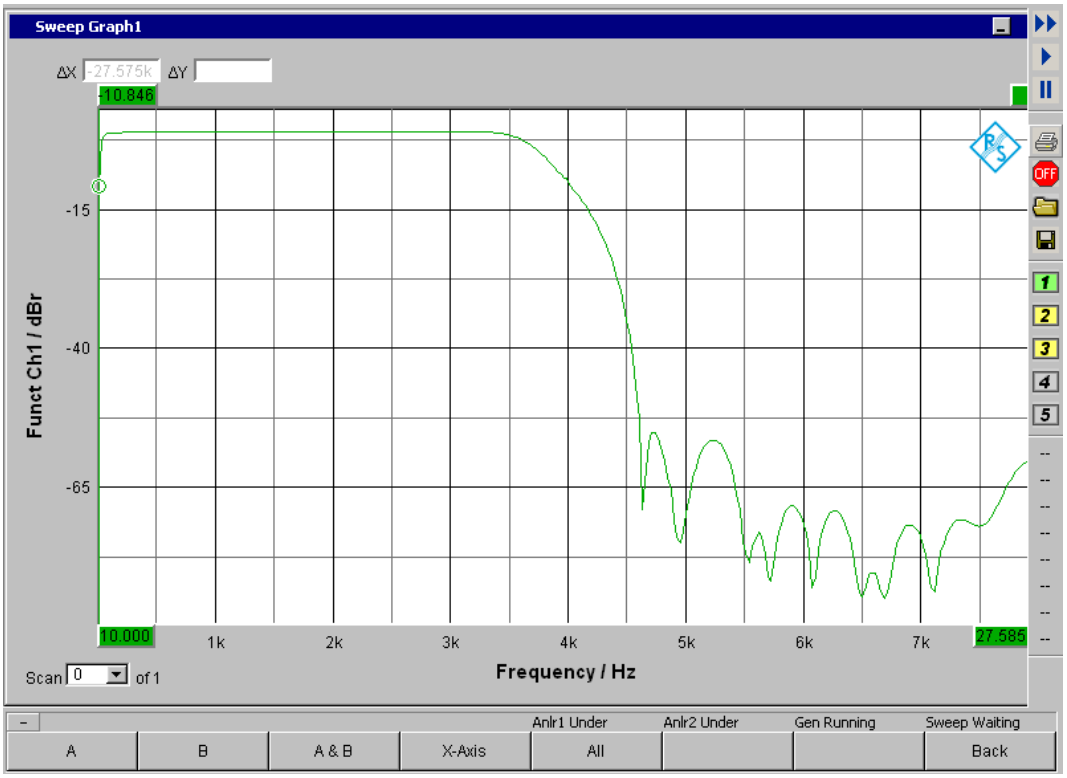

Abbildung 10: Frequenzgang DSK-Board

Insgesamt entspricht der Frequenzgang den Erwartungen, wenn eine Abtastfreqeunz von 8 kHz eingestellt ist. Ab der halben Abtastfrequenz ca. 4 kHz fällt der Frequenzgang mit Tiefpasscharakter ab, sodass nur Frequenzen bis $\frac{f_A}{2}$ verarbeitbar sind.

2.2 Implementierung des FIR-Filters

Zur Implementierung des Filters wurden folgendes C-Programm geschrieben. Alle von uns geänderten oder hinzugefügten Zeilen sind kommentiert, dass bedeutet hinter jedem Kommentar, wurde von uns etwas geändert und alle anderen Kommentare entfernt.

```
/* authors: Michael Langen
   date: 28-11-2012
   FIR Filter
   --> Lowpass
*/
#include "c6713dskinit.h"
#include "dsk6713.h"
#include <math.h>
#include "LP_coeff_firpm.h"

#define LEFT 1
#define RIGHT 0
#define BUFLEN N
```

```
//count variables for IRS
int i, j = 0;

//stores Result of FIR calculation
int FIR_accu32 = 0;

//Array of Input-Samples
short int delays[N];

static Uint32 CODECEventId;

//Sampling Frequency set 8kHz
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;

// two buffers for input and output samples with two counters
short int inBuf_L[BUFLEN];
short int inBuf_R[BUFLEN];
short int count_INT=0;

union {
        Uint32 both;
        short channel[2];
} AIC23_data;


//ISR for calculating the FIR-Filter
interrupt void intser_McBSP1()
{
        FIR_accu32 = 0;

//shift the old Data
        for(j = N-1; j > 0; j--){
                delays[j] = delays[j-1];
        }

//read from the Left channel
        AIC23_data.both = MCBSP_read(DSK6713_AIC23_DATAHANDLE);

//store newest always at position zero
        delays[0] = AIC23_data.channel[LEFT];

//calculate new output value
        for(i=0; i < N;i++){
                FIR_accu32 =  FIR_accu32 + h[i] * delays[i];
        }

//cast to short and store at left channel
        AIC23_data.channel[LEFT] = (short) (FIR_accu32 >> 15);

//write data to outout
        MCBSP_write(DSK6713_AIC23_DATAHANDLE, AIC23_data.both);

        return;
}

void main()
{
        IRQ_globalDisable();

        DSK6713_init();

        hAIC23_handle=DSK6713_AIC23_openCodec(0, &config);
        DSK6713_AIC23_setFreq(hAIC23_handle, fs);


        MCBSP_config(DSK6713_AIC23_DATAHANDLE,&AIC23CfgData);

        MCBSP_start(DSK6713_AIC23_DATAHANDLE, MCBSP_XMIT_START | MCBSP_RCV_START |
        MCBSP_SRGR_START | MCBSP_SRGR_FRAMESYNC, 220);

        CODECEventId= MCBSP_getXmtEventId(DSK6713_AIC23_DATAHANDLE);

//initilasing the Array with Zero
        for(i=0; i < N;i++){
        delays[i] = 0;
        }

        IRQ_map(CODECEventId, 5);
        IRQ_reset(CODECEventId);
        IRQ_globalEnable();
        IRQ_nmiEnable();
        IRQ_enable(CODECEventId);
        IRQ_set(CODECEventId);

    while(1);
}
```

Wird ein Interrupt ausgelöst, so läuft folgendes ab. Die alten eingelesenen Werte werden in einem Array um einen Wert nach rechts geschoben. Das Array hat die Länge von der Anzahl der Filterkoeffizienten. Der Wert am Ende des Array wird somit beim neueinspeichern immer verworfen. Danach wird der neue Wert vom AD-Wandler eingelesen und direkt an die 0-te Stelle im Array gespeichert. Dann wird in einer for-Schleife der neue Ausgabewert berechnet, indem die Summe über alle

Filterkoeffizienten mal den eingelesen Werten an den Stellen gebildet wird. Dieser Wert wird dann noch gecastet und ausgegeben.
Der Vorteil dieser Berechnung ist, dass in der eigentlich for-Schleife noch mit 32bit Integer Zahlen gerechnet wird (hohe Genauigkeit). Hierbei treten nur sehr geringe Rundungsfehler auf. Dieser Wert wird dann erst nach der Berechnung in der for-Schleife auf 16bit-short gecastet. Somit rechnet man möglichst lange mit hoher Auflösung und summiert so keine großen Rundungsfehler auf.

Frequenzgänge des Filters

Nun wurde mit dem Spektrumanalyzer der Amplitudengang des implementierten Filters aufgenommen. Dieser ist in Abbildung **??** dargestellt.

Abbildung 11: Amplitudengang FIR-Filter

Insgesamt ist der Amplitudengang schon nahe an der geforderten Spezifikation laut der Aufgabenstellung, jedoch ist die Sperrdämpfung in diesem Fall noch nicht groß genug. Gefordert waren mindestens 40 dB. Diesen Wert erreicht der Filter jedoch erst mit der 3. Nebenkeule, was nicht den Anforderungen entspricht.
Laut Simulation sollte das Filter den Anforderungen entsprechen, in der Praxis reicht das allerdings nicht aus. Das reale System verhält sich nicht wie das simulierte Modell in MATLAB, denn dort wurde auch mit sehr hoher Genauigkeit gerechnet, was auf einem DSP nicht möglich ist. Um trotzdem noch die Anforderungen zu erfüllen erhöhten wir die Anzahl an Filterkoeffizienten und führten die Amplitudengangmessung erneut durch. Nun ergab sich der in Abbildung **??** dargestellte Amplitudengang. Diese Filtercharakteristik erfüllt mit den zusätzlichen Koeffizienten die Anforderung aus der Aufgabenstellung und hat im Sperrbereich eine Dämpfung von größer 40 dB. Die Koeffizienten wurden in diesem Fall von 17 auf 19 erhöht, um die Anforderungen zu erfüllen.

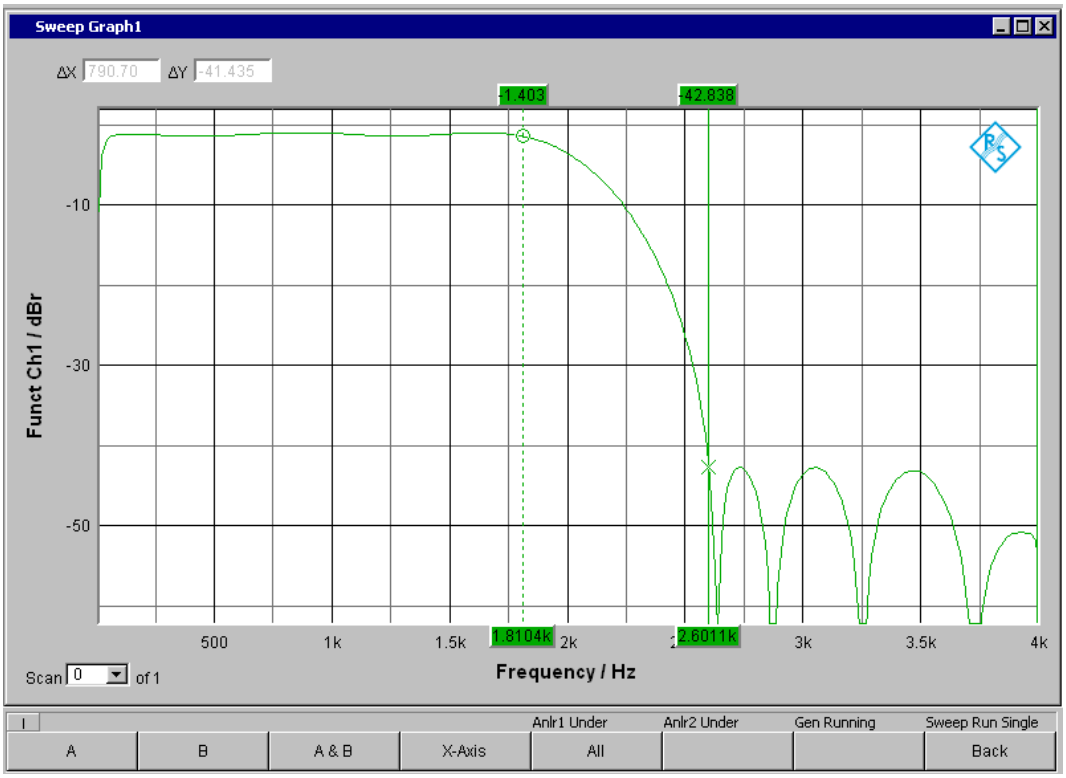

Abbildung 12: Amplitudengang FIR-Filter mit korrigierten Koeffizienten

Profiling der ISR

Nun sollte mittel Profiling der ISR ermittelt werden, wie viel Zeit der DSP in dieser ISR verbraucht und daraus dann die maximale Taktfrequenz berechnet werden. Das Profiling wurde laut der Anleitung auf der Laborseite durchgeführt.
Laut dem Code Composer Studio braucht die ISR etwa 1551 Taktzyklen. Zunächst wird die Zeitdauer eines Taktes berechnet.

$$T_T = \frac{1}{225\,\mathrm{MHz}} = 4.4\,\mathrm{ns}$$

Die gesamte in der ISR verbrauchte Zeit ist somit: Anzahl der Takte in der ISR mal Periodendauer eines Taktes

$$T_{ISR} = 4.4\,\mu\mathrm{s} \cdot 1151 = 6.89\,\mu\mathrm{s}$$

Daraus lässt sich nun die Abtastfrequenz bestimmen.

$$f_{maxISR} = \frac{1}{T_{ISR}} = \frac{1}{6.89\,\mu\mathrm{s}} = 145\,\mathrm{kHz}$$

Dieser Wert ist allerdings die maximale Taktfrequenz. Der Wert ist aber noch geringer zu wählen, um dem DSP auch noch etwas Zeit für andere Aufgaben zu geben. Bei dieser Berechnung wurden davon ausgegangen, dass wirklich nur die ISR mit dem FIR-Filter zu berechnen ist und keine andere Aufgabe noch bewältigt werden muss.

2.3 Hoch- und Tiefpassfilter (Weichenfilter)

Aus dem gegebenem Tiefpassfilter sollte nun ein Hochpassfilter als Weichenfilter entwickelt werden. Die Grundidee lässt sich aus Abbildung **??** erklären.

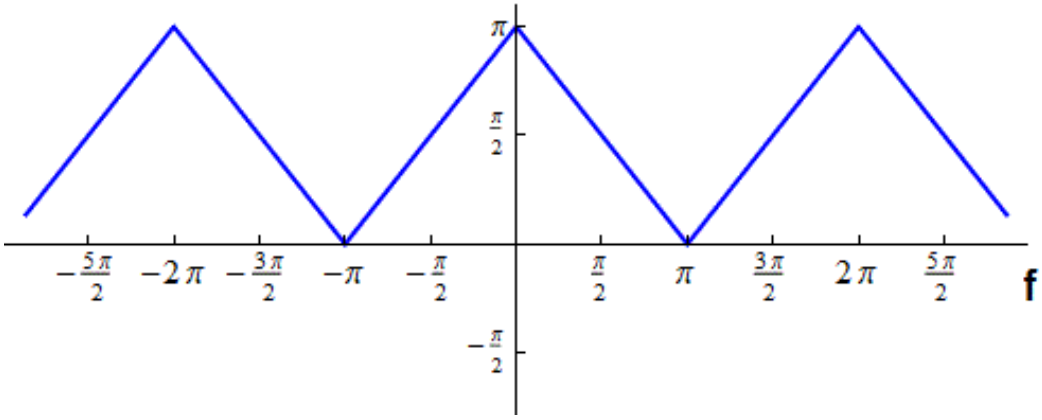

Abbildung 13: Spektrum Tiefpassfilter

Wir befinden uns nicht im zeitkontinuierlichen sonder im diskreten, damit sind alle Spektren periodisch. Die Abbildung **??** zeigt nun beispielhaft einen periodischen Amplitudengang eines Tiefpasses. $2 \cdot \pi$ entspricht hierbei der Abtastfrequenz und $\frac{\pi}{2}$ der halben Abtastfrequenz. Um aus diesem nun ein Hochpass zu machen, muss der Tiefpass-Amplitudengang um π verschoben werden. Das lässt sich mit dem Verschiebungssatz auch wie folgt angeben: $H_{HP}(f) = H_{LP}(f - \pi)$
Die Beziehung zwischen den beiden Filter ist gegeben durch $H(z) = H(-z)$. Mathematisch bedeutet das, dass ausgehend von der Tiefpass FIR-Filter Formel:

$$H_{LP}(z) = \sum_{k=0}^{N-1} b_k \cdot z^{-k} \quad \rightarrow \quad H_{TP}(-z) = \sum_{k=0}^{N-1} b_k \cdot -z^{-k} = \sum_{k=0}^{N-1} b_k \cdot (-1)^{-k} \cdot z^{-k}$$

Mit diesem Zusammenhang folgt dann:

$$b_{HP} = (-1)^k \cdot b_{LP}$$

Daraus folgt für die Hochpass-Samples das nur die Vorzeichen von allen ungeraden Samples geändert werden müssen.
Diese Funktion wurde nun auch in einem C-Programm realisiert. Die ISR wurden um eine Berechnung für den Hochpass erweitert. Wie im vorherigen Quellcode sind alle Änderung kommentiert.
Das Programm ist nahezu identisch mit dem der vorherigen Aufgabe. In der ISR wird bei der Berechnung für den Ausgabewert des Hochpasses überprüft, ob es hierbei um einen ungeraden Koeffizienten handelt. Ist dies der Fall, so wird dieser negativ multipliziert.

```
/* authors: Michael Langen
   date: 28-11-2012
   FIR Filter
  --> Lowpass and Highpass
*/

#include "c6713dskinit.h"
#include "dsk6713.h"
#include <math.h>
#include "LP_coeff_firpm.h"

#define LEFT 1
#define RIGHT 0
#define BUFLEN N

//count variables for IRS
int i, j = 0;

//stores Result of FIR HP and LP calculation
int FIR_accu32_LP, FIR_accu32_HP = 0;

//Array of Input-Samples
short int delays[N];

static Uint32 CODECEventId;
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;

// two buffers for input and output samples with two counters
short int inBuf_L[BUFLEN];
short int inBuf_R[BUFLEN];
short int count_INT=0;

union {
        Uint32 both;
        short channel[2];
} AIC23_data;

//ISR for calculating the FIR-Filter
interrupt void intser_McBSP1()
{
                FIR_accu32_LP = 0;
                FIR_accu32_HP = 0;

//shift the old Data
                for(j = N-1; j > 0; j--){
                        delays[j] = delays[j-1];
                }

//read from the Left and Right channel
                AIC23_data.both = MCBSP_read(DSK6713_AIC23_DATAHANDLE);

//store newest always at position zero
                delays[0] = AIC23_data.channel[LEFT];

//calculate new output value for LP and HP
                for(i=0; i < N;i++){
//Lowpass
                        FIR_accu32_LP =  FIR_accu32_LP + h[i] * delays[i];
//Highpass
                        if (i%2 != 0 )
                                FIR_accu32_HP =  FIR_accu32_HP  - h[i] * delays[i];
                        else
                                FIR_accu32_HP =  FIR_accu32_HP + h[i] * delays[i];
                }

//cast and write Left and Right channel to short
                AIC23_data.channel[LEFT] = (short) (FIR_accu32_LP >> 15);
                AIC23_data.channel[RIGHT] = (short) (FIR_accu32_HP >> 15);

//write back to DA-Converter
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, AIC23_data.both);

        return;
}


void main()
{

        IRQ_globalDisable();

        DSK6713_init();

        hAIC23_handle=DSK6713_AIC23_openCodec(0, &config);
        DSK6713_AIC23_setFreq(hAIC23_handle, fs);

        MCBSP_config(DSK6713_AIC23_DATAHANDLE,&AIC23CfgData);

        MCBSP_start(DSK6713_AIC23_DATAHANDLE, MCBSP_XMIT_START | MCBSP_RCV_START |
        MCBSP_SRGR_START | MCBSP_SRGR_FRAMESYNC, 220);

        CODECEventId= MCBSP_getXmtEventId(DSK6713_AIC23_DATAHANDLE);

//initilasing the Array with Zero
        for(i=0; i < N;i++){
        delays[i] = 0;
        }

        IRQ_map(CODECEventId, 5);
```

```
        IRQ_reset(CODECEventId);
        IRQ_globalEnable();
        IRQ_nmiEnable();
        IRQ_enable(CODECEventId);
        IRQ_set(CODECEventId);

    while(1);
}
```

Amplitudengänge Weichenfilter

Zum Aufnehmen der Amplitudengänge wurde nun wieder der Spektrumanalyser mit dem Board verbunden und der Amplitudengang vom linken (Tiefpass) und rechten (Hochpass) Kanal aufgenommen. Die Amplitudengänge sind Abbdildung **??** entnehmbar.

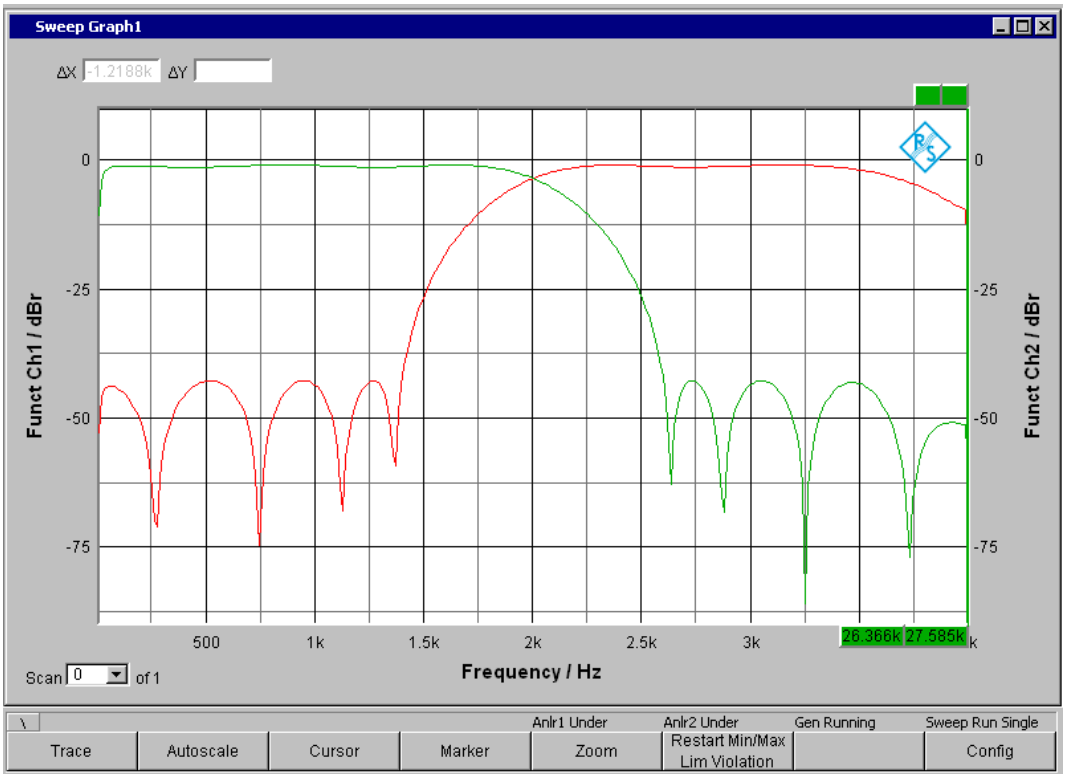

Abbildung 14: Amplitudengänge Weichenfilter

Insgesamt entsprechen die beiden Amplitudengänge der Erwartung. Der Hochpass ist genau der gespiegelte Verlauf des Tiefpasses. Beide Sperrbereiche zeigen die gleiche Ripple-Charakteristik und Sperrdämpfung, allerdings auch gespiegelt.
Wenn man genau hinsieht, ist allerdings im Durchlassbereich des Hochpasses am Ende bereits der Einfluss des Amplitudenganges des DSP-Boardes zu erkennen. Dieser Amplitudengang der sich nun dort Überlagert wurde bereits in Abbildung **??** aufgenommen.

Veränderung mittlerer Filterkoeffizient

Das laufende Programm wurde nun angehalten und der mittlere Koeffizient um 1 verringert. Zu beachten war auch hier die vorher verwandte Skalierung mit 2^{15}, sodass der Koeffizient um diese Zahl verringert werden musste. Der Wert des mittleren Koeffizienten betrug 17713 und wurde auf -15055 geändert. Der aufgenommene Amplitudengang ist in Abbildung **??** dargestellt.

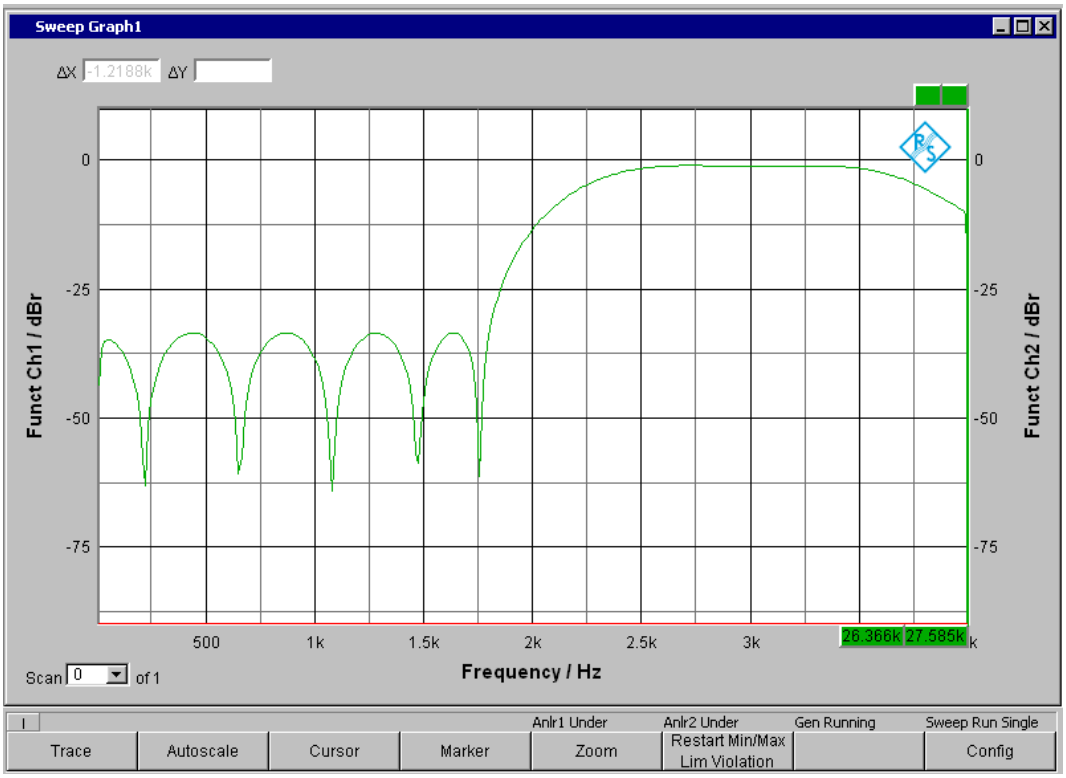

Abbildung 15: Amplitudengang geänderter Tiefpass

Durch ändern des mittleren Koeffizienten des Tiefpasses, entsteht nun ein Hochpass. Dieses Verfahren ist allerdings nur bei ungerader Anzahl von Koeffizienten möglich, weil sonst die symmetrieeigenschaften verändert werden.

3 Fazit

In diesem Praktikum sollten nun das erste Mal eigene Filter entworfen werden. Dabei wurden alle Designschritte durchlaufen. Vom Schreiben eines MATLAB Skriptes über die Simulation bis hin zur lauffähigen Implementierung auf dem DSP mit anschließender Validierung des vorher simulierten Filters.
Interessant war zu sehen, dass die vorher in MATLAB simulierten Filter sich auf dem Board oft nicht ganz so verhielten, wie man es nach der Simulation erwartete. Das zeigte sich bei uns besonders in dem entworfenem Tiefpass zu Beginn des Praktikums. Innerhalb der Simulation mit MATLAB hielt der berechnete Filter die gegebenen Spezifikationen ein. Lies man den Filter nun auf dem Board laufen, wurden die Spezifikationen mit dem Filter nicht mehr eingehalten.
Wichtig ist also auch nach der theoretischen Berechnung des Filters eine Verifikation von diesem durchzuführen.Damit er unter den gegeben Einflüssen auch richtig arbeitet. Alternativ bestünde natürlich die Möglichkeit, die Einflüsse bereits in der Simulation zu berücksichtigen, wenn kein geeignetes Objekt zum Testen zur Verfügung steht. In unserem Fall hätte dann der Einfluss des Amplitudenganges des Boardes und die Rundungsfehler des DSP berücksichtigt werden müssen.
Auch wichtig ist die Messung der CPU-Zeit der ISR. Besonders bei größeren Projekten ist darauf zu achten, die ISR möglichst kurz zu halten. Über die Dauer der ISR kann dann auch die maximale Arbeitsgeschwindigkeit abgeschätzt werden, wie im Praktikum auch geschehen. Allerdings gilt diese, wie schon erwähnt, nur unter optimalen Bedingungen, wenn der Prozessor wirklich nur die ISR abarbeitet, was in der Praxis wohl eher selten der Fall sein wird.

Abbildungsverzeichnis