Laborversuch

Digitale Filter

Zu einem der wichtigsten Bausteine in einem digitalen Signalverarbeitungssystem gehören lineare digitale Filter.
Man unterscheidet rekursive und nichtrekursive (transversale) digitale Filter. Also Filter mit einer unendlich langen Impulsantwort (Infinite-duration Impulse Response, IIR) und Filter mit einer endlich langen Impulsantwort (Finite-duration Impulse Response, FIR).
Im ersten Teil dieses Laborversuchs sollen digitale Filter mit Hilfe von MATLAB entworfen und untersucht werden.
Der zweite Teil hat die Realisierung eines nichtrekursiven digitalen Filters, d.h. die Implementierung auf dem DSK6713 zum Gegenstand.

1. Theorie

1.1 Systemfunktion digitaler Filter

Digitale Filter lassen sich allgemein durch eine Übertragungsfunktion der Form

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_1 + b_2 \cdot z^{-1} + ... + b_{n_b+1} \cdot z^{-n_b}}{a_1 + a_2 \cdot z^{-1} + ... + a_{n_a+1} \cdot z^{-n_a}} \tag{1.1}$$

beschreiben.

Diese Notation wird auch beim Filterentwurf in MATLAB benutzt. Die Ordnung des Filters beträgt $n = \max(n_b, n_a)$, wobei für kausale Systeme $n_b \leq n_a$ gilt – also der Nennergrad maßgebend ist.

Die zugehörige Differenzengleichung lautet:

$$a_1 \cdot y(k) + a_2 \cdot y(k-1) + ... + a_{n_a+1} \cdot y(k-n_a) = b_1 \cdot x(k) + b_2 \cdot x(k-1) + ... + b_{n_b+1} \cdot x(k-n_b) \tag{1.2}$$

bzw.:

$$a_1 \cdot y(k) = b_1 \cdot x(k) + b_2 \cdot x(k-1) + ... + b_{n_b+1} \cdot x(k-n_b) - a_2 \cdot y(k-1) - ... - a_{n_a+1} \cdot y(k-n_a) \tag{1.3}$$

Bei der Differenzengleichung handelt es sich um eine Rechenvorschrift, die gleichzeitig einen direkten Weg für das Vorgehen bei der Implementierung liefert. Der Koeffizient a_1 muss dabei noch den Wert 1 erhalten. Dies wird durch Normierung aller Koeffizienten auf a_1 erreicht. Die Differenzengleichung lautet dann:

$$y(k) = d_1 \cdot x(k) + d_2 \cdot x(k-1) + ... + d_{n_d+1} \cdot x(k-n_d) - c_2 \cdot y(k-1) - ... - c_{n_c+1} \cdot y(k-n_c) \qquad (1.4)$$

Der gegenwärtige Wert der Ausgangsfolge $y(k)$ wird dabei aus dem gegenwärtigen Wert der Eingangsfolge $x(k)$ sowie aus vergangenen Werten von Eingangs- und Ausgangsfolge berechnet.

Die Übertragungsfunktion lautet dann:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{d_1 + d_2 \cdot z^{-1} + ... + d_{n_d+1} \cdot z^{-n_d}}{1 + c_2 \cdot z^{-1} + ... + c_{n_c+1} \cdot z^{-n_c}} \qquad (1.5)$$

In der Literatur ist meist ebenfalls diese Form der Darstellung zu finden.

Die Eigenschaften des Systems werden durch die Koeffizienten b_n und a_n bzw. d_n und c_n festgelegt.

Eine mögliche Struktur ist die Direkt-Form 1:

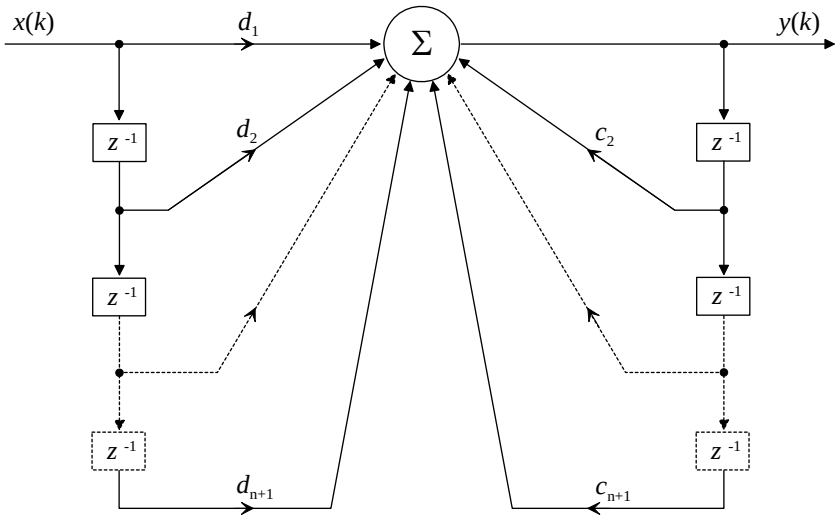


Abbildung 1.1: Allgemeine Filterstruktur Direkt-Form 1

Die Systemfunktion (1.5) beschreibt ein rekursives digitales Filter (IIR-Filter), gleichzeitig aber auch den „allgemeinen" Fall eines digitalen Filters. Setzt man nämlich im Nenner alle Koeffizienten $c_n = 0$, dann bleibt mit dem Zähler der sog. transversale Teil der Systemfunktion übrig. Er beschreibt ein nichtrekursives digitales Filter (FIR-Filter), das sog. Transversalfilter. Der Nenner enthält demnach den rein rekursiven Teil eines digitalen Filters.

Eine weitere Form der Übertragungsfunktion erhält man nach der Multiplikation von Zähler und Nenner mit z^n wobei $n = \max(n_d, n_c)$:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{d_1 \cdot z^n + d_2 \cdot z^{n-1} + ... + d_{n_d+1}}{z^n + c_2 \cdot z^{n-1} + ... + c_{n_c+1}} \qquad (1.6)$$

In dieser Form lässt sich sehr gut die verbleibende n-fache Polstelle eines FIR-Filters im Ursprung erkennen (Zur Erinnerung: im Nenner sind dann alle Koeffizienten $c_n = 0$, d.h. nur z^n bleibt übrig).

Die Übertragungsfunktion lässt sich aber auch über die Wurzeln der Polynome im Zähler (Nullstellen) und Nenner (Pole) angeben:

$$H(z) = \frac{Y(z)}{X(z)} = k \cdot \frac{(z - z_{o1}) \cdot (z - z_{o2}) \cdot \ldots \cdot (z - z_{on_b})}{(z - z_{\infty 1}) \cdot (z - z_{\infty 2}) \cdot \ldots \cdot (z - z_{\infty n_a})} \tag{1.7}$$

In MATLAB stehen zur Umformung von der einen in die andere Darstellung (transfer function - tf ⇔ zp - zero pole gain) folgende Funktionen zur Verfügung: `tf2zp`, `zp2tf`

Es kann aber auch eine getrennte Umformung von Zähler oder Nenner erfolgen. Dafür stehen folgende Funktionen zur Verfügung: `roots`, `poly`

Es besteht dabei folgende Konvention, die bei der Übergabe an andere Funktionen beachtet werden muss (z.B. bei `freqz`, `zplane` … s.u.):

Die Koeffizienten von Polynomen werden als Zeilenvektor (Form: 1 x n) gespeichert und dadurch als solche erkannt. Im Gegensatz dazu werden die Wurzeln eines Polynoms als Spaltenvektor (Form: n x 1) abgelegt.

1.2 FIR-Filter

1.2.1 Systemfunktion und Struktur

Die Systemfunktion eines nichtrekursiven digitalen Filters der Ordnung n lautet also:

$$H(z) = \frac{Y(z)}{X(z)} = b_1 + b_2 \cdot z^{-1} + \ldots + b_{n+1} \cdot z^{-n} \tag{1.8}$$

Daraus kann direkt die in Abbildung 1.2 dargestellte Struktur abgeleitet werden. Die Koeffizienten b_i mit $i = 1, 2, 3, \ldots, n+1$ stellen dabei die Antwort des Systems auf einen Einheitsimpuls dar, die sog. Impulsantwort.

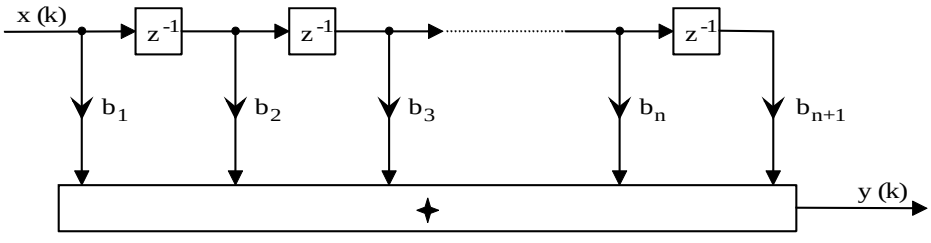


Abbildung 1.2: FIR-Filterstruktur

1.2.2 Hinweis zur Realisierung

Die Struktur in Abbildung 1.2 nennt sich Direktform eines FIR-Filters oder auch Transversalfilter und wird in einem Signalprozessor auf diese Weise realisiert. Die erforderlichen Zustands- und

Koeffizientenspeicher werden dabei meist als sog. Modulo-Speicher (Ringspeicher) ausgeführt (nähere Erläuterungen dazu siehe Kapitel 3.3 und: Infoblatt ‚Code Composer Studio…').
Bei den Zustandsspeichern hat die Modulo-Adressierung den Vorteil, dass nicht ganze Speicherbereiche umkopiert werden müssen, sondern dass nur der Zeiger, der auf die aktuelle Position zeigt, am Ende eines Durchlaufs um eine Position zurückgestellt werden muss, um mit dem nächsten neu ankommenden Sample (nächster Filterdurchlauf) den letzten (ältesten) Zustandsspeicher zu überschreiben.

1.2.3 Vor- und Nachteile von FIR-Filtern

Der Hauptvorteil von FIR-Filtern ist die Stabilität. Aufgrund des Fehlens einer Rückkopplung (d.h. des Fehlens von Polen außerhalb des Ursprungs) sind sie immer stabil.

Weiter kann nur mit Hilfe von FIR-Filtern ein linearer Phasengang erzeugt werden. Die Gruppenlaufzeit eines solchen Filters ist dann konstant und beträgt $n/2 \cdot T_a$, bei geradem Filtergrad n also ein ganzzahlig Vielfaches der Abtastperiodendauer T_a.
Die Nullstellen liegen dabei entweder direkt auf dem Einheitskreis der z-Ebene oder symmetrisch zum Einheitskreis (Spiegellage). Dort, wo sich Nullstellen direkt auf dem Einheitskreis befinden, springt die Phase um 180° und die Gruppenlaufzeit ist damit nicht definiert. Dies spielt jedoch keine Rolle, da das Filter an diesen Stellen ja sperrt.

Der Hauptnachteil von FIR-Filtern ist die gegenüber IIR-Filtern nötige höhere Filterordnung, um ähnliche Filterverläufe (Flankensteilheiten) zu erhalten. Dies ist leicht einzusehen, stehen doch gegenüber IIR-Filtern nur Nullstellen zum Einstellen des Filterverlaufs zur Verfügung. Die Ordnung liegt daher meist um den Faktor 10 und mehr höher. Damit einher geht eine entsprechend höhere Gruppenlaufzeit.

1.2.4 Beispiel eines FIR-Filters

Als ein erstes Beispiel eines FIR-Filters soll das Ihnen bereits aus Laborversuch 2 des Signale & Systeme Labors bekannte Mittelungsfilter dienen. Der Amplitudengang des Filters wird deshalb als bekannt vorausgesetzt.
Dort hatten Sie L Abtastwerte aufsummiert und das Ergebnis anschließend durch L dividiert. Die Koeffizienten b_i waren also alle 1. Die Division am Ende hätte man aber auch in die Koeffizienten verschieben können, die dann alle den Wert 1/L erhalten hätten. Da Sie L Abtastwerte berücksichtigt hatten, war die Ordnung n des Filters also L-1. Man erhält dadurch ebenfalls L-1 Nullstellen.

Für L = 10 lauten diese:

```
 0.8090 + 0.5878i
 0.8090 - 0.5878i
 0.3090 + 0.9511i
 0.3090 - 0.9511i
-1.0000
-0.8090 + 0.5878i
-0.8090 - 0.5878i
-0.3090 + 0.9511i
-0.3090 - 0.9511i
```

Sie liegen, wie in Abbildung 1.3 dargestellt, auf dem Einheitskreis der z-Ebene. Im Ursprung lässt sich ebenfalls die n-fache Polstelle des Filters erkennen (wie oben beschrieben).

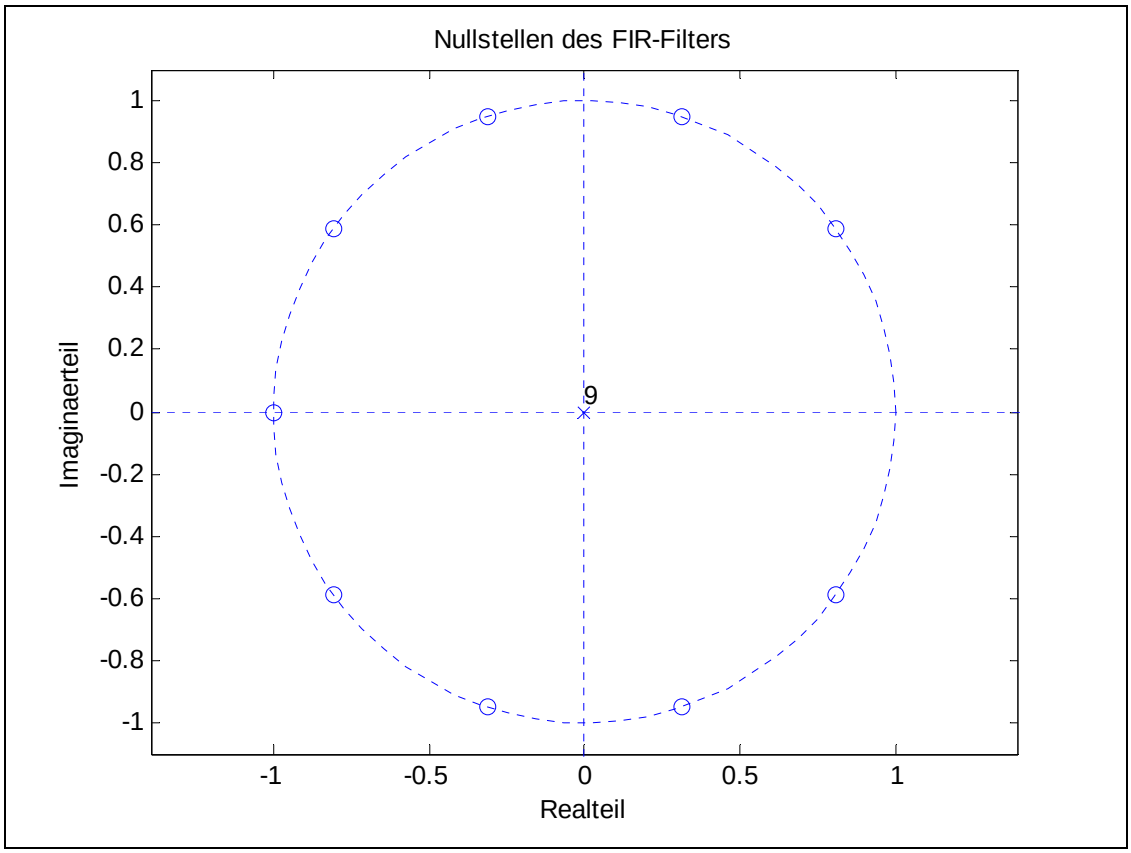


Abbildung 1.3: Nullstellen eines Mittelungsfilters neunter Ordnung

2. Filterentwurf mit MATLAB

2.1 Entwurf von FIR-Filtern

2.1.1 Funktionen für den FIR-Filterentwurf

Die Systemfunktion $H(z)$ eines diskreten, nichtrekursiven Systems lässt sich nicht, wie es bei einem rekursiven System der Fall ist, durch eine bekannte Abbildung aus der Systemfunktion eines kontinuierlichen Systems $H(s)$ ableiten (Stichwort: Konforme Abbildung – bilineare Transformation, siehe Kapitel 2.3.1). Es werden deshalb verschiedene Approximationsverfahren angewandt, um nichtrekursive Systeme zu erhalten. Dabei lassen sich beliebige Formen des Amplitudengangs, z.B. auch mehrfache Sperr- und Durchlassbereiche erzielen.

In MATLAB stehen dafür folgende m-files zur Verfügung: `fir1`, `fir2`
`firls`, `firpm` (früher `remez`)
`fircls`, `fircls1`
`cfirpm` (früher `cremez`)
`firrcos`

Bis auf `cfirpm` erzeugen alle oben genannten Entwurfsmethoden FIR-Filter mit linearer Phase. Nur mit `cfirpm` können auch FIR-Filter mit nichtlinearer Phase entworfen werden.

Beim Entwurf von FIR-Filtern mit linearer Phase werden vier verschiedene Kategorien (I-IV) anhand der Filterordnung (gerade / ungerade) und der Symmetrie der Koeffizienten unterschieden, weil sich dadurch z.T. Einschränkungen bezüglich des möglichen Filterverlaufs ergeben. So kann z.B. ein Filter mit ungerader Ordnung n und symmetrischer Impulsantwort kein Hochpass- oder Bandsperrfilter sein (vgl. Literatur FIR-Filtertyp II).
Diese Kriterien (Symmetrie der Koeffizienten und gerade / ungerade Filterordnung) können übrigens bei der Realisierung von FIR-Filtern aufwandsreduzierend ausgenutzt werden.

2.1.2 Entwurf mit fir1.m

Mit Hilfe der MATLAB-Funktion `fir1` können die Koeffizienten für ein linearphasiges FIR-Filter der Ordnung n nach der Fenstermethode bestimmt werden (standardmäßig wird das Hamming-Fenster verwendet, auf nähere Angaben zu diesem Verfahren soll hier verzichtet werden).
Die normierte Grenzfrequenz `Wn` liegt im Bereich 0…1, wobei 1 der halben Abtastfrequenz entspricht ($f_a/2$ wird auch Nyquistfrequenz f_n genannt). Allerdings liegt die normierte Grenzfrequenz hier nicht wie gewohnt bei einer normierten Verstärkung von -3 dB, sondern bei -6 dB.

Aufruf der Funktion:

```
b = fir1(n, Wn, 'ftype');
```

Dabei können folgende Filtertypen durch Angabe der entsprechenden Zeichenkette für `'ftype'` gewählt werden:

Tiefpass: keine Angabe oder `'low'`
Bandpass: keine Angabe oder `'bandpass'`
Hochpass: `'high'`
Bandsperre: `'stop'`

Standardmäßig erfolgt eine Normierung der Koeffizienten, so dass der Amplitudengang des Filters im Durchlassbereich 0 dB erreicht.

Bei Bandfiltern muss `Wn` einen Vektor mit mindestens zwei Bandgrenzen `w1` und `w2` enthalten (`Wn = [w1 w2]`). Bei Angabe weiterer Bandgrenzen werden sog. Multibandfilter erzeugt (siehe MATLAB-Hilfe, z.B.: `help fir1`).

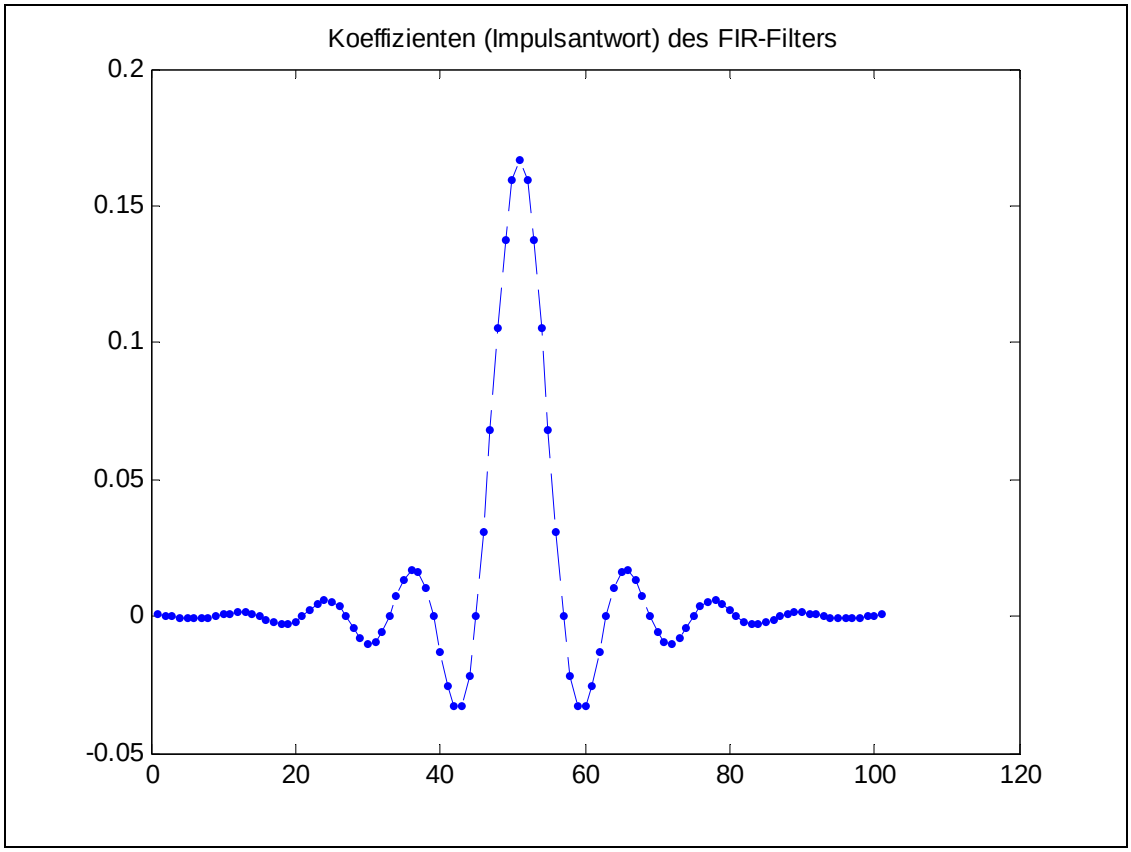
Für Hochpass- und Bandsperrfilter muss eine gerade Ordnung vorgegeben werden. Nötigenfalls wird die gerade Ordnung durch Erhöhen der übergebenen Ordnung um 1 erzwungen (siehe Filterkategorien I-IV).

Die Gruppenlaufzeit der erzeugten Filter beträgt $n/2 \cdot T_a$ (linearphasiges Filter).

Beispielentwurf:

```
% Ordnung n:
n = 100;  % Anzahl der Nullstellen in der kompl. Ebene

fa = 48000;  % Abtastfrequenz in Hz passend zur Signalverarbeitungs-Hardware wählen
fn = fa/2;  % Nyquistfrequenz

Wn = 4000/fn;  % -6 dB – Grenzfrequenz in Hz

FIRkoeff = fir1(n, Wn, 'low');  % TP-Filter
```

```
% Alternativ der Entwurf eines Bandpasses:
%pass1 = 8000;  % -6 dB – Grenzfrequenz in Hz
%pass2 = 16000;  % -6 dB – Grenzfrequenz in Hz
%Wn = [pass1/fn pass2/fn];
%FIRkoeff = fir1(n, Wn, 'bandpass');  % BP-Filter

% Darstellung der Koeffizienten
% -----------------------------
fig = figure(1);
plot(FIRkoeff, '--')
hold on;
plot(FIRkoeff, '.')
hold off
title('Koeffizienten (Impulsantwort) des FIR-Filters')
```



2.1.3 Berechnung des Frequenzgangs mit freqz.m

Die Analyse des Frequenzgangs eines diskreten Systems kann z.B. mit Hilfe der MATLAB-Funktion `freqz` erfolgen. Es liegt eine Systemfunktion nach (1.1) zugrunde. Das m-file erwartet, dass ihm Koeffizienten für Zähler und Nenner der Übertragungsfunktion übergeben werden. Der Koeffizientenvektor für den Nenner muss für ein FIR-Filter also eine 1 gefolgt von der passenden Anzahl 0-en enthalten (`a = [1 0 0 ... 0]`). Die Berechnung des Frequenzgangs basiert auf der schnellen Fouriertransformation.

Aufruf der Funktion:

```
[h, w] = freqz(b, a);
```

Der komplexe Frequenzgang $H(e^{jw})$ wird, ohne eine optional mögliche Angabe, für 512 Punkte im Bereich 0 bis π berechnet und in h gespeichert. In w werden die zugehörigen Stützpunkte abgelegt.
Die möglichen Optionen der Funktion `freqz` bezüglich der Frequenz(-einteilung) sind vielfältig (siehe MATLAB-Hilfe, z.B.: `help freqz`).

Die Berechnung des Amplituden- und Phasengangs erfolgt durch folgende Anweisungen:

`amplitude = abs(h);` oder in dB: `amplitude = 20*log10(abs(h));`

`phase = angle(h);` bzw.: `phase = unwrap(angle(h));`

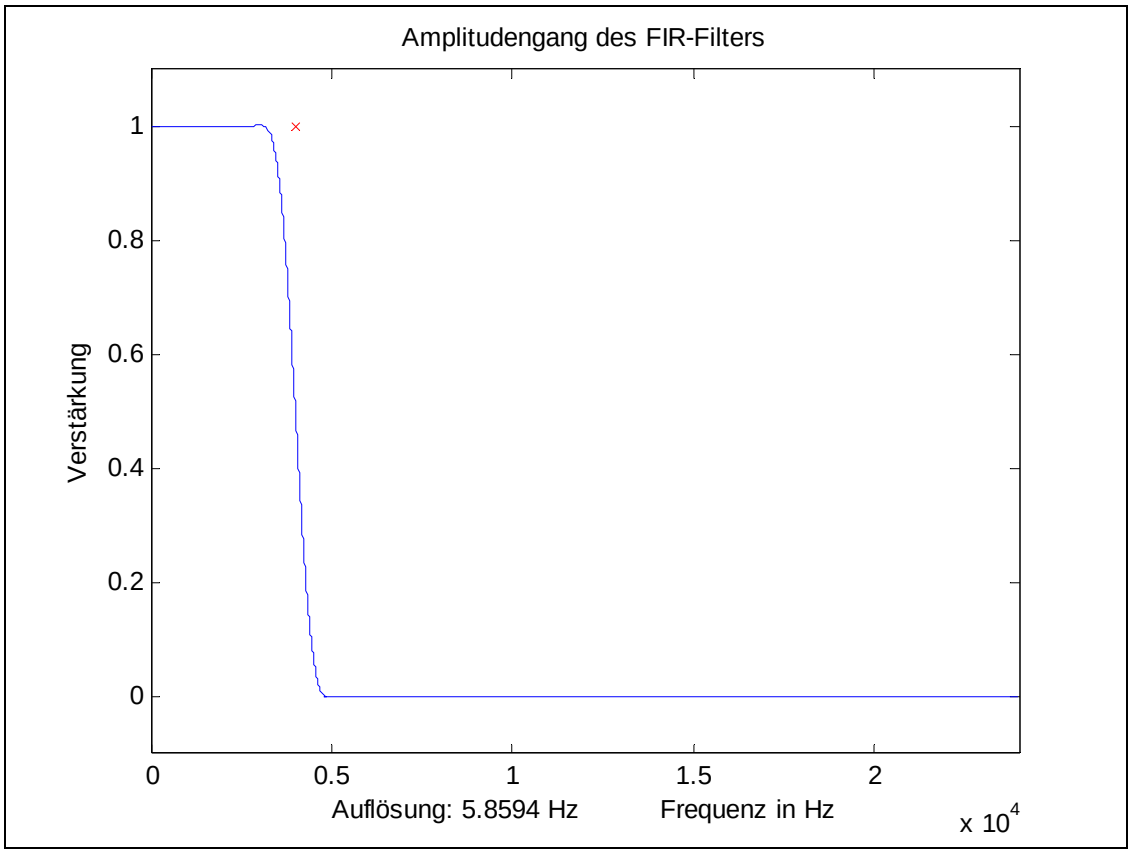
(Erläuterungen zum Befehl `unwrap` siehe MATLAB-Hilfe, z.B.: `help unwrap`)

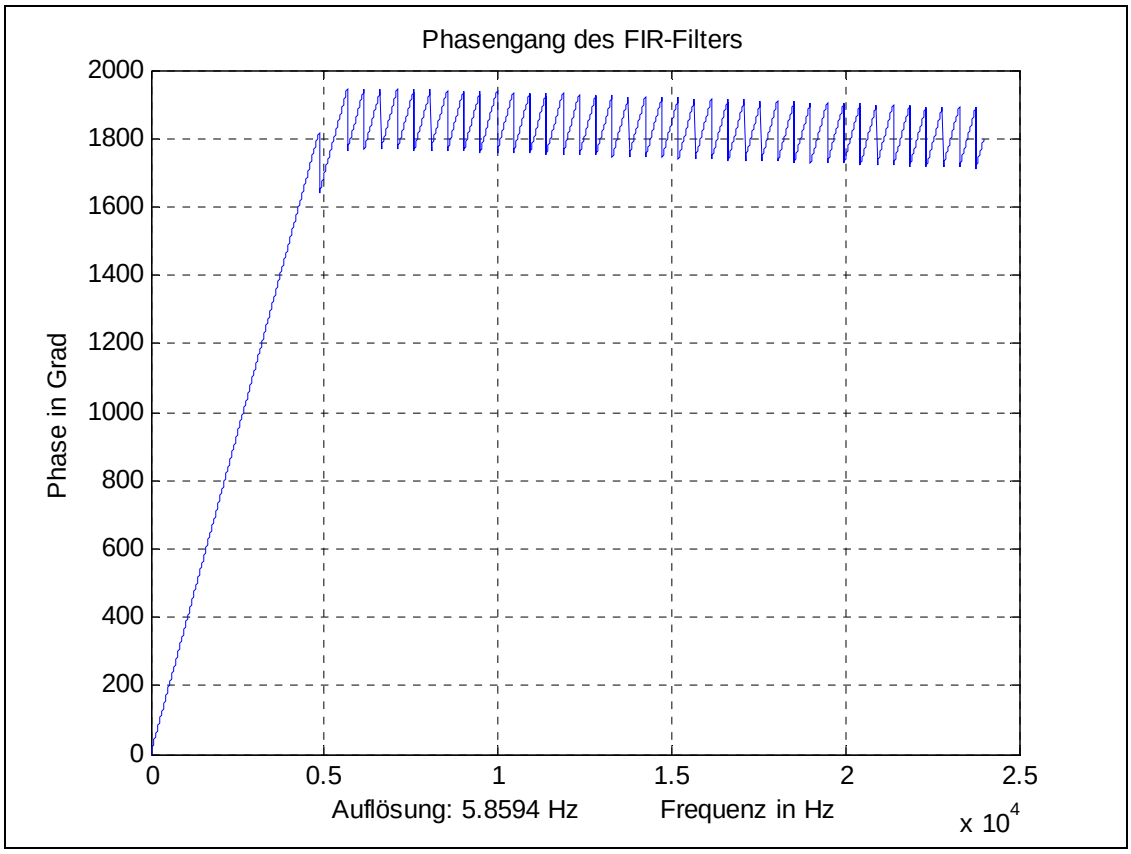
Hinweis: Dem englischen Begriff `phase` entspricht im Deutschen der Phasenwinkel. Dem Phasengang (auch Phase oder Phasenmaß genannt) entspricht hingegen `-phase` !

Die Darstellung des Amplituden- und Phasengangs erhalten Sie über die bekannten plot-Befehle.

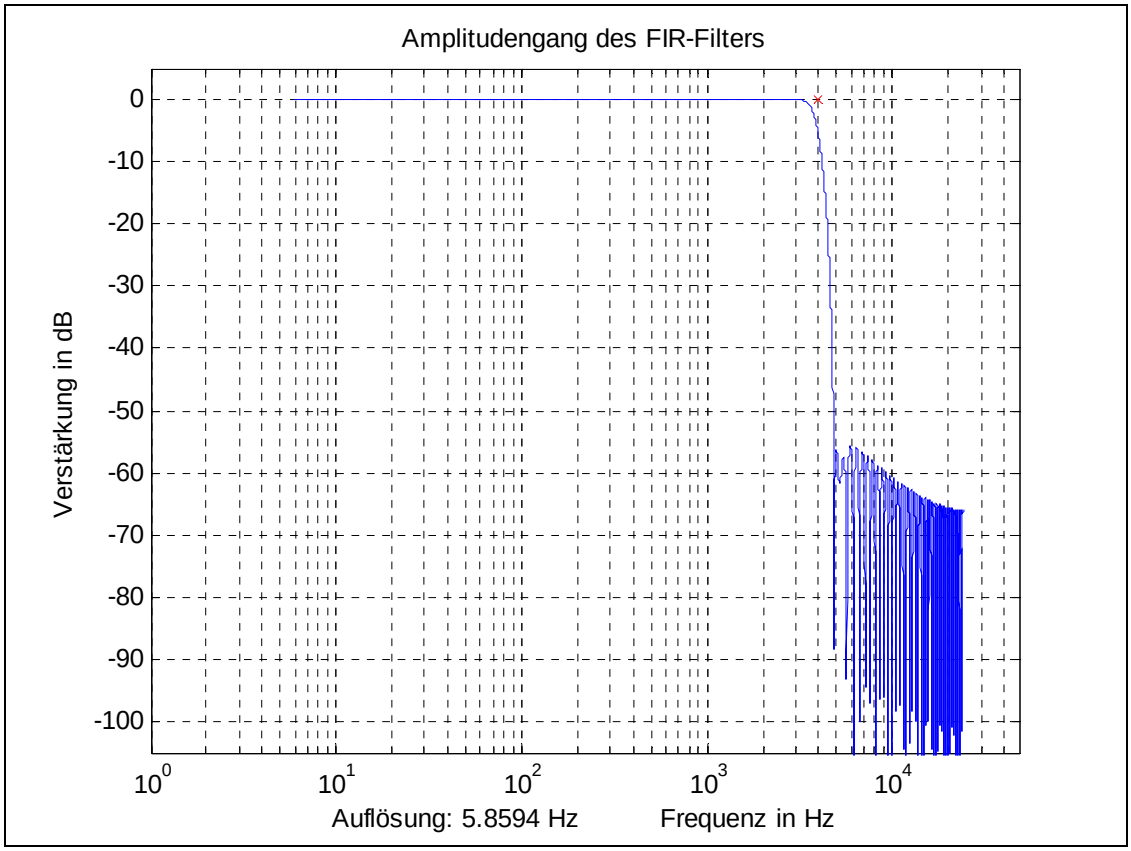
Fortsetzung des Beispielentwurfs:

```
% Zunächst den Nennerkoeffizientenvektor erstellen:
% (nötig zur korrekten Rechnung mit freqz, grpdelay, zplane)
a = zeros(size(FIRkoeff));  % oder:  a = zeros(1, n+1);
a(1) = 1;

% freqz über Aufruf mit Koeffizientenvektoren
cntW = 4096;
% Standardmäßig (ohne zusätzliche Parameterangabe cntW) werden nur 512 Frequenzpunkte
% berechnet!
[Hfir, Wfir] = freqz(FIRkoeff, a, cntW);  % mit 0 <= Wfir <= pi

df = Wfir(2)/pi*fn;  % oder:  df = fn/length(Wfir);
% df (delta_f) ist der Abstand zwischen den Frequenzpunkten in Hz
% d.h. df entspricht der Frequenzauflösung in Hz

amplFIR = abs(Hfir);

phaseFIR = angle(Hfir);


% Darstellung des Amplitudengangs (linear)
% ----------------------------------------

fig = figure(fig+1);
plot(Wfir/pi*fn, amplFIR, 'b');
axis([0 fn -0.1 1.1]);
title('Amplitudengang des FIR-Filters')
ylabel('Verstärkung')
xlabel(['Auflösung: ',num2str(df),' Hz          Frequenz in Hz'])
hold on;
plot(Wn*fn,1,'rx')
hold off;


% Darstellung der Phase
% ---------------------

fig = figure(fig+1);
plot(Wfir/pi*fn, -unwrap(phaseFIR)/pi*180)
title('Phasengang des FIR-Filters')
ylabel('Phase in Grad')
xlabel(['Auflösung: ',num2str(df),' Hz          Frequenz in Hz'])
grid
```


Amplitudengang des FIR-Filters
Verstärkung
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
Auflösung: 5.8594 Hz
Frequenz in Hz
x 10^4


Phasengang des FIR-Filters
Phase in Grad
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
0.5
1
1.5
2
2.5
Auflösung: 5.8594 Hz
Frequenz in Hz
x 10^4

```
% Darstellung des Amplitudengangs (logarithmisch)
% ------------------------------------------------
fig = figure(fig+1);
semilogx(Wfir/pi*fn, 20*log10(amplFIR), 'b');
axis([1 fa -105 5]);  % 1 Hz bis fa, -105 dB bis +5 dB
title('Amplitudengang des FIR-Filters')
ylabel('Verstärkung in dB')
xlabel(['Auflösung: ',num2str(df),' Hz          Frequenz in Hz'])
grid
hold on;
plot(Wn*fn,0,'rx')
hold off;
```



Zu Beachten sind die beiden Darstellungen des Amplitudengangs. Aus der linearen Darstellung folgt die oft gehörte Aussage zur Charakterisierung (bzw. zum Wunschverlauf) eines Filters (z.B. eines Anti-Aliasing-Tiefpassfilters): „Keine Dämpfung im Durchlassbereich und Null sonst“.
Das dabei genauer betrachtet Null eben nicht Null ist, wird in der logarithmischen Darstellung des Amplitudengangs deutlich. Und was ca. 55 dB Dämpfung bedeuten weis jeder, der das Rauschen beim Abspielen von Compact-Cassetten ohne Rauschunterdrückung noch im Ohr hat.

2.1.4 Darstellung der Null- und Polstellen mit zplane.m

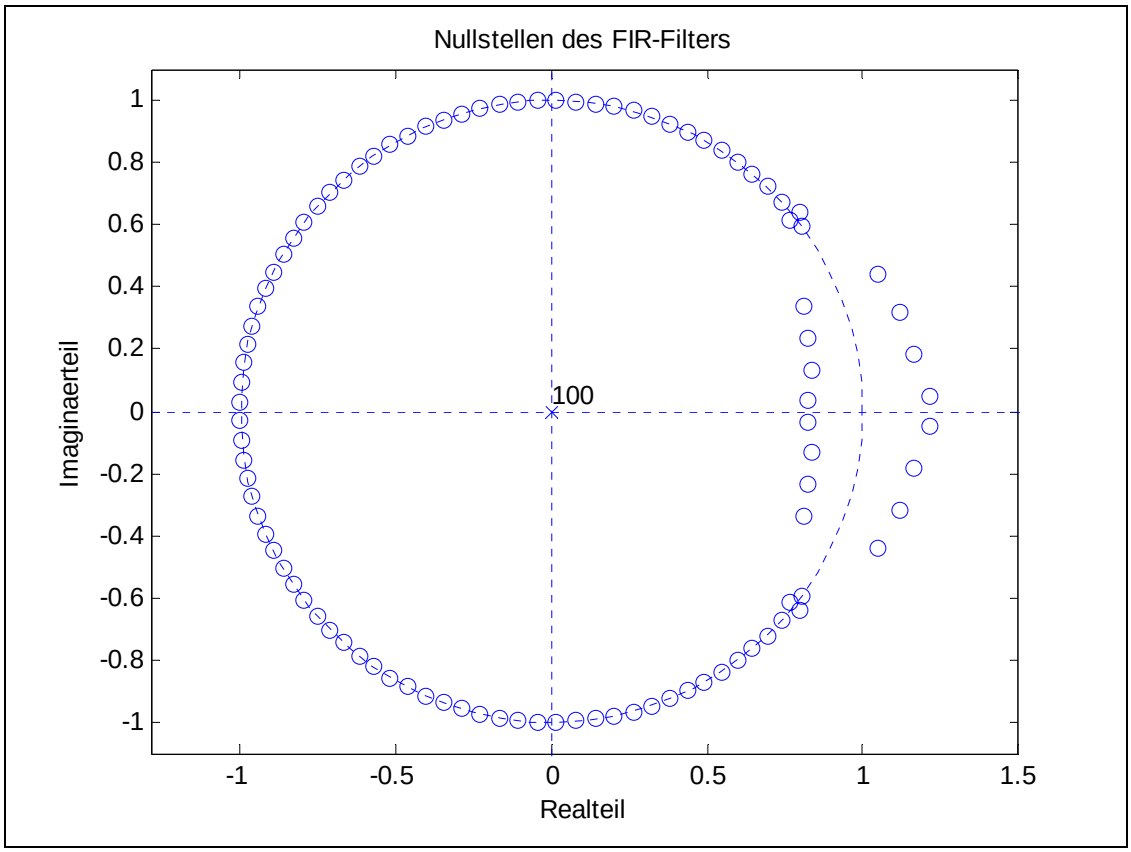
Die Darstellung der Null- und Polstellen eines diskreten Systems kann z.B. mit Hilfe der MATLAB-Funktion `zplane` erfolgen. Das m-file erwartet, dass ihm entweder Koeffizienten für Zähler und Nenner der Übertragungsfunktion übergeben werden oder bereits daraus berechnete Null- und Polstellen.
Bei der Übergabe von Koeffizienten muss, wie bei der MATLAB-Funktion `freqz`, der Koeffizientenvektor für den Nenner eines FIR-Filters eine 1 gefolgt von der passenden Anzahl 0-en enthalten (`a = [1 0 0 ... 0]`).

Aufruf der Funktion:

```
zplane(b, a);
```

Fortsetzung des Beispielentwurfs:

```
% Darstellung der Nullstellen in der z-Ebene
% ----------------------------------------

fig = figure(fig+1);
zplane(FIRkoeff, a)
title('Nullstellen des FIR-Filters')
ylabel('Imaginaerteil')
xlabel('Realteil')
```



Vergleichen Sie die Lage der Nullstellen mit dem Amplituden- und Phasengang.

2.1.5 Erzeugen der Koeffizientendatei mit fprintf.m

Die Koeffizientendatei zum Einbinden in ein Software-Projekt für das DSK6713 kann wie folgt erzeugt werden:

Fortsetzung des Beispielentwurfs:

```
% Erzeugen der Koeffizientendatei:

fid = fopen('FIRkoeff.h', 'w');  % file-id
fprintf(fid,'const float FIRkoeff[] = {\n');
fprintf(fid,'%1.7e,\n',FIRkoeff(1:end-1));
fprintf(fid,'%1.7e};\n',FIRkoeff(end));
fclose(fid);
```

2.2 Vorbereitung, Durchführung, Aufgaben und Fragen

2.1 Entwerfen Sie ein FIR-Bandpassfilter mit Hilfe der MATLAB-Funktion `fir1`.

Die Bandgrenzen legen Sie dabei so fest, dass Sie bei der späteren Implementierung des Filters auf dem DSK6713 bei einer Abtastfrequenz von f_a = 32 kHz die Wirkungsweise des Filters auch gehörmäßig gut überprüfen können. Das bedingt zum einen, ein sinnvolles Frequenzband zu wählen (also z.B. nicht 10 kHz bis 14 kHz) und zum anderen, dass bei gegebener Ordnung des Filters von n = 130 zu beiden Seiten des Durchlassbereichs eine ausreichende, d.h. ähnlich hohe Dämpfung erreicht wird.

Vorbereitung:

Übernehmen Sie die MATLAB-Befehle des Beispielentwurfs in ein m-file und passen Sie die Parameter entsprechend an.
Sofern beim Lesen noch nicht geschehen, sollten Sie sich dabei auch in die Funktionsweise des Matlab-Scripts einarbeiten, d.h. den Sinn der einzelnen Befehle, deren Ergebnisse und das Zusammenwirken erfassen und verstehen!

Testen Sie einige verschiedene Bandgrenzen mit Hilfe des von Ihnen erstellten m-files. Die logarithmische und die lineare Darstellung des Amplitudengangs kann dabei sinnvoll sein. Als Anhaltspunkt für einen gehörmäßig sinnvollen Durchlassbereich kann die NF-Bandbreite des öffentlichen Fernsprechnetzes dienen.
Erreichen Sie bei einer solchen Vorgabe eine zu beiden Seiten des Durchlassbereichs ausreichende, d.h. ähnlich hohe Dämpfung?
Falls nein, passen Sie die untere Bandgrenze entsprechend an.

Woraus und mit Hilfe welcher Berechnungsschritte erhält man den Amplitudengang eines FIR-Filters?

Enthält das Pol-Nullstellendiagramm eines FIR-Filters Polstellen?
Wenn ja, wie viele und wo liegen sie?

2.2 Hat die Abtastfrequenz einen Einfluss auf die erreichbare Dämpfung?
Ändern Sie dazu die Abtastfrequenz in Ihrem m-file ab (bei entsprechend gewählten Bandgrenzen z.B. auf f_a = 12 kHz). Die logarithmische und die lineare Darstellung des Amplitudengangs kann bei der Beantwortung der Frage hilfreich sein. Erläutern Sie Ihre Antwort auch anhand der Pol-Nullstellendarstellung in der z-Ebene.

2.3 Testen Sie den Einfluss der Filterordnung auf die erreichbare Dämpfung.
Halbieren und verdoppeln Sie dazu den Vorgabewert für die Ordnung. Sehen Sie sich auch jeweils die Pol-Nullstellendarstellung in der z-Ebene an.

2.4 Erzeugen Sie einen Ausdruck des Amplitudengangs des von Ihnen letztendlich ausgewählten Filters (n = 130, f_a = 32 kHz !), so dass ein späterer Vergleich mit der Realisierung auf dem DSK6713 möglich wird.

Kennzeichnen Sie im Ausdruck des Amplitudengangs die Sperr- und Übergangsbereiche sowie den Durchlassbereich (die dazu benötigten Frequenzen ergeben sich aus den -3 dB Punkten und den Schwellen für minimale Dämpfung in den beiden Sperrbereichen).

2.5 Erzeugen Sie die Koeffizientendatei (siehe Kapitel 2.1.5).

Die Fragen sind handschriftlich in der Versuchsvorbereitung bzw. Ihrem Versuchsprotokoll zu beantworten!

2.3 Entwurf von IIR-Filtern

2.3.1 Funktionen für den IIR-Filterentwurf

Die Systemfunktion $H(z)$ eines diskreten, rekursiven Systems lässt sich durch eine konforme Abbildung aus der Systemfunktion eines kontinuierlichen Systems $H(s)$ ableiten (Stichwort: bilineare Transformation).
Man geht dabei von den Koeffizienten eines analogen Tiefpassfilter-Prototypen (normiert auf die Grenzfrequenz) aus, transformiert diese dann nach Bedarf in einen anderen Filtertyp (Hochpass, Bandpass, …) und legt die gewünschte Grenzfrequenz fest. Danach erfolgt mit Hilfe der bilinearen Transformation die Konvertierung des kontinuierlichen Systems $H(s)$ in ein diskretes System $H(z)$.
Die bilineare Transformation erfüllt die Bedingung, die linke s-Halbebene ins Innere des Einheitskreises der z-Ebene sowie die $j\Omega$-Achse der s-Ebene auf den Einheitskreis der z-Ebene abzubilden und damit aus einem stabilen Bezugssystem ebenfalls ein stabiles digitales System zu gewinnen.
Die Bilineartransformation ist eine umkehrbar eindeutige konforme Abbildung. Sie lautet:

$$z_{\infty/0} = \frac{1 + \frac{1}{2} \cdot T_a \cdot s_{\infty/0}}{1 - \frac{1}{2} \cdot T_a \cdot s_{\infty/0}} \quad \text{bzw} \quad . \quad s_{\infty/0} = \frac{2}{T_a} \cdot \frac{1 - \frac{1}{z_{\infty/0}}}{1 + \frac{1}{z_{\infty/0}}} \tag{2.1}$$

Die bilineare Transformation hält zwar die im Analogen vorgegebene 3dB-Grenzfrequenz ein, führt aber bei höheren Grenzfrequenzen (ab ca. $f_a/8$) zu einer Erhöhung der Flankensteilheit wegen der nichtlinearen Verzerrung der Frequenzachse.

Eine weitere Möglichkeit der Konvertierung stellt die Impulsinvariantenmethode dar. Sie ist allerdings nicht für die Konvertierung von Hochpass- und Bandsperrfiltern geeignet.

In MATLAB besteht aber auch die Möglichkeit, die Koeffizienten für ein diskretes, rekursives System auf verschiedene Art und Weise direkt zu ermitteln.

Für den Entwurf von IIR-Filtern stehen in MATLAB insgesamt folgende m-files zur Verfügung:

`lp2lp`, `lp2hp`, `lp2bp`, `lp2bs`

`bilinear`, `impinvar`

`butter`, `maxflat`, `cheby1`, `cheby2`, `ellip`, `besself`

`yulewalk`, `prony`, `lpc`, `stmcb`, `invfreqz`, `invfreqs`

2.3.2 Analyse von IIR-Filtern

Die Analyse des komplexen Frequenzgangs und die Darstellung der Null- und Polstellen kann mit den bereits in Kapitel 2.1 für FIR-Filter vorgestellten Funktionen erfolgen. Da IIR-Filter, im Gegensatz zu (den meisten) FIR-Filtern, keine lineare Phase haben und damit die Gruppenlaufzeit nicht konstant ist, kann bei der Analyse von IIR-Filtern zusätzlich die Darstellung der Gruppenlaufzeit von Interesse sein.

2.3.3 Berechnung der Gruppenlaufzeit mit grpdelay.m

Die Berechnung der Gruppenlaufzeit eines diskreten Systems kann z.B. mit Hilfe der MATLAB-Funktion `grpdelay` erfolgen. Es liegt eine Systemfunktion nach (1.1) zugrunde. Das m-file erwartet, dass ihm Koeffizienten für Zähler und Nenner der Übertragungsfunktion übergeben werden. Der Koeffizientenvektor für den Nenner muss für ein FIR-Filter also eine 1 gefolgt von der passenden Anzahl 0-en enthalten (`a = [1 0 0 ... 0]`).

Aufruf der Funktion:

```
[gd, w] = grpdelay(b, a);
```

Die Gruppenlaufzeit wird, ohne eine optional mögliche Angabe, für 512 Punkte im Bereich 0 bis π berechnet und in `gd` gespeichert. In `w` werden die zugehörigen Stützpunkte abgelegt.
Die möglichen Optionen der Funktion `grpdelay` bezüglich der Frequenz(-einteilung) sind vielfältig (siehe MATLAB-Hilfe, z.B.: `help grpdelay`)

3. Implementierung eines FIR-Filters auf dem DSK6713

3.1 Quell-Code-Vorgaben

Angelehnt an den letzten Laborversuch stehen Ihnen folgende Quellcode-Dateien im Projektordner 5 zur Verfügung:

\Project5\fir.c	Rumpfprogramm, das von Ihnen erweitert werden soll
\Project5\init_mcbsp_codec.c	Initialisierung der McBSPs und des Codecs
\Project5\vectors3.asm	Interrupt-Vektor-Tabelle
\Project5\c6713dsk.h	Include-Datei (C6713 und DSK Definitionen)
\Project5\dsk6713.cmd	Linker-Konfigurationsdatei

3.2 Die Windows-Audio-Software

Auf den Labor-PCs steht Ihnen neben dem Windows Media Player zum Abspielen von CDs und Wave-Dateien auch wieder das Programmpaket AudioTester zur Verfügung (siehe Laborversuch 2, Kapitel 1.2 des Signale & Systeme Labors).

3.3 FIR-Filteralgorithmus in C

In der Regel stellen DSPs für die in der Signalverarbeitung oft benötigten **Ringspeicher** eine entsprechende Adressierlogik zur Verfügung. Auch bei den DSPs von TI ist das der Fall.

Diese Adressierlogik ermöglicht das Inkrementieren und Dekrementieren insbesondere auch an den Ringspeichergrenzen und erzwingt dann den Sprung ans andere Ende des Ringspeichers. In Assembler wird diese Adressierung über die Initialisierung und Verwendung spezieller Register angesprochen. In C steht dafür bisher keine Implementierung zur Verfügung. (weitere Erläuterungen dazu siehe Infoblatt ‚Code Composer Studio…').
Es ist deshalb in der Literatur, wenn es um die Realisierung von FIR-Filtern in C geht, meist nur der Weg über das Speicherumkopieren der Zustandsspeicher am Ende jedes Filterdurchlaufs zu finden.
Das stellt aber für ein Signalverarbeitungssystem in der Regel einen sehr ineffizienten Weg dar. Sind es doch oft die vielfältigen Speicherzugriffe von peripheren Einheiten (Serielle Hochgeschwindigkeits-Ports, DMAs usw.) und der CPU (möglicherweise auch mehreren), die sich gegenseitig blockieren und so die Gesamtperformance des Systems limitieren. (In der Regel wird in einem SVS ja nicht nur ein Filter berechnet.)

Ein möglicher anderer Weg ist die Nachbildung dieser sog. Moduloadressierung in C. Das bedeutet allerdings, dass man den Zeiger auf den Ringspeicher der Abtastwerte während der Filterberechnung permanent überwachen muss (was, insbesondere bei FIR-Filtern hoher Ordnung, wiederum ineffizient ist).
Nach kurzer Überlegung findet man aber durchaus auch Wege, diese Moduloadressierung während der Filterberechnung zu umgehen, sie also nicht in C nachzubilden.
Eine Möglichkeit besteht darin, den Zustandsspeicher zweimal hintereinander anzulegen. Dadurch kann bei der Filterberechnung einfach linear über den eigentlichen Beginn des Ringspeichers – dieser ist also der Zweite der beiden Zustandsspeicher – hinaus adressiert (dekrementiert) werden und trotzdem werden die richtigen Daten in der richtigen Reihenfolge gelesen. Der Nachteil besteht hier im sich verdoppelnden Speicherbedarf für die Zustandsspeicher.
Eine andere Möglichkeit stellt die Verwendung von zwei `for`-Schleifen bei der Filterberechnung dar. Auch dabei muss dann, trotz der Verwendung eines Ringspeichers einfacher Größe, der Zeiger auf den Speicher nicht mehr permanent überprüft werden. Welche der möglichen Filterberechnungsvarianten im konkreten Fall effizienter ist, hängt von mehreren Faktoren ab, insbesondere vom verwendeten DSP, von der Filterlänge und vom Compiler.

Voraussetzungen beim Empfangsinterrupt:

Die Empfangs-Interruptserviceroutine inkrementiert den globalen Zeiger `j` auf den Ringspeicher bevor der neue Abtastwert in den Speicher geschrieben wird (`++j`). Man überschreibt dadurch also den ältesten im Speicher vorhandenen Abtastwert mit dem neuen. Der globale Zeiger zeigt danach auf diesen neuen Abtastwert, mit dem dann die Filterberechnung beginnen kann.
Diese Vorgehensweise setzt wiederum voraus, dass der Zeiger vor dem Schreibzugriff und damit vor dem Inkrementieren nicht auf das letzte Element des Ringspeichers zeigt. Das bedeutet, dass die Zeigerverwaltung in der Empfangs-ISR vor dem Schreibbefehl erfolgen muss (siehe Beispiel-Code unten).

Filterberechnung mit zwei `for`-Schleifen:

Die erste der beiden Filterberechnungsschleifen berechnet die Teilsumme vom aktuellen (neuen) Abtastwert bis zum Ringspeicheranfang. Dann wird der Zeiger auf das Ringspeicherende gestellt und die Summenberechnung mit der zweiten Filterberechnungsschleife bis zum ältesten im Ringspeicher vorhandenen Abtastwert fortgeführt. Der Koeffizientenspeicher wird dabei linear durchadressiert, beginnend mit dem ersten Element `FIRkoeff[0]`.

Beispiel-Code:

Initialisierungen:

```
.
.
#include "FIRkoeff.h"  // FIR-filtercoefficient-table with floating-point
                       // single-precission-coeffs. between +/-1.0
                       // const float FIRkoeff[] = {...};
.
.
#define L  ??  // L = n+1 !
.
.
float left_sample_in_buffer[L];
float right_sample_in_buffer[L];
int j = 0;
int start_calc = 0;

unsigned int data_word = 0;  // output for XINT
.
.
```

Empfangs-ISR:

```
interrupt void isr_rint(void)
{
  unsigned int data_word_in;

  // setup the circular pointer before writing to the buffers
  if (j >= (L-1)) j -= L;  /* if true then j=-1 */

  // write samples to global sample circular buffers:
  // increment pointer before writing to buffers (pre-increment)
  // (due to the codec sample clock this write takes place every 1/fa_Hz)
  data_word_in = *(unsigned volatile int *)McBSP1_DRR;
  left_sample_in_buffer[++j] = (float)((data_word_in >> 16) & 0x0000FFFF);
  right_sample_in_buffer[j] = (float)(data_word_in & 0x0000FFFF);

  start_calc = 1;  // start-flag for calculation in main
}
```

Filterberechnung:

```
void main()
{
  register int calc_j;  // local variable increases execution speed !
  register int k;
  register float l_sp_y = 0.0;  // left single precission y
  register float r_sp_y = 0.0;  // right single precission y
  register int start_index;
  .
  .
  while (1)
  {
    if (start_calc)
    {
      start_calc = 0;
      calc_j = j;
      l_sp_y = 0.0;
      r_sp_y = 0.0;

      // FIR-Filter calculation with floating-point-coeffs:

      start_index = calc_j+1;

      // 1. part (pre wrap part)
      // calculate from actual sample-buffer-index (newest sample) until j=0
      for (k=0; k<start_index; k++)
        l_sp_y += (FIRkoeff[k] * sample_buffer[calc_j]);
        r_sp_y += (FIRkoeff[k] * sample_buffer[calc_j--]);
```

```
      // wrap around to highest sample-buffer-index
      calc_j = L-1;
      // 2. part (post wrap part)
      // calculate from j=L-1 (top end of sample-buffers) until
      // buffer-index is start_index
      // (last [j--] corrects the pointer to the starting position)
      for (k=start_index; k<L; k++)
        l_sp_y += (FIRkoeff[k] * sample_buffer[calc_j]);
        r_sp_y += (FIRkoeff[k] * sample_buffer[calc_j--]);
      // output for XINT:
      data_word = (((int)l_sp_y) << 16) | (((int)r_sp_y) & 0x0000FFFF);
    }
  }
}
```

3.4 Durchführung, Aufgaben und Fragen

3.1 Erstellen Sie mit Code Composer Studio v5 ein neues Projekt für die DSK6713-Plattform (vgl. Laborversuch Mittelungsfilter, Kapitel 3.2, Absatz 3.2).
Fügen Sie die zur Verfügung gestellten Quellcode-Dateien dem Projekt hinzu.
Nehmen Sie die nötigen Einstellungen bei den Build Options vor (Target processor version 6700, später eventuell Heap Size und Stack Size anpassen).

Bringen Sie zunächst das Programm als "**Durchreicher**" zum Laufen. Beachten Sie dabei, dass über den ersten DIL-Schalter (`USER_SW0`) eine Umschaltmöglichkeit zwischen Durchreichen der Abtastwerte und Filtern vorgesehen ist. Den DIL-Schalter zunächst also nicht drücken!

Anmerkungen:

Die Abtastwerte des Stereo-Codecs werden durch die Receive-Interruptserviceroutine als Gleitkommazahlen in getrennten Ringspeichern für den linken und den rechten Kanal abgelegt. Die Berechnung in `main()` wird nach jedem angekommenen Abtastwertepaar angestoßen.
Die Transmit-Interruptserviceroutine erwartet – wie in den bisherigen Projekten – ein 32 Bit breites Datenwort (bestehend aus zwei 16 Bit Integer-Werten) zur Übergabe der Samples an den Stereo-Codec. Um die Übergabe bzw. den Zugriff auf die Übergabespeicher-stelle(n) eleganter zu gestalten, wird in diesem Projekt zunächst eine Struktur definiert, die im nächsten Schritt in einer Union benutzt wird:

```
struct samples
{
  short left;
  short right;
};
union stereo
{
  unsigned int lr_word;
  struct samples channel;
};
union stereo audio;
```

Der Zugriff auf das 32 Bit breite Datenwort (beispielsweise in der Transmit-ISR) erfolgt dann über:

```
audio.lr_word
```

Und der Zugriff auf die beiden 16 Bit breiten Komponenten (Samples) erfolgt über:

```
audio.channel.left
audio.channel.right
```

Beim Umkopieren der Samples (beispielsweise beim "Durchreichen") muß also eine Typkonvertierung (von `float`) auf `short` vorgenommen werden.

In der Receive-ISR wird ebenfalls eine solche Union benutzt, dort allerdings lokal (siehe C-Quellcode-Datei *fir.c*).

3.2 Testen Sie kurz die korrekte Funktion mit einem Sinussignal aus dem Funktionsgenerator des Software-Pakets AudioTester (siehe Laborversuch Mittelungsfilter, Kapitel 2.1, Abs. 2.4 und 2.5) und mit passenden Musikbeispielen über den Kopfhörer.
(Achtung: Einstellungen am Mischpult überprüfen! – siehe Infoblatt ‚Erläuterungen zur Audioverkabelung')
Achten Sie jeweils darauf, dass der A/D-Wandler korrekt ausgesteuert wird, d.h. ermitteln Sie die Aussteuergrenze. Zu Ihrer Hilfe ist dazu in der Receive-Interruptserviceroutine eine Minimalpegelanzeige über drei LEDs des DSK implementiert. Sie zeigen die jeweils maximale Aussteuerung der A/D-Wandler im Codec an (- 30 dBFS, - 10 dBFS und Peak). Bei korrekter Aussteuerung sollte die `USER_LED3` (Peak-Anzeige) gerade nicht aufleuchten.
Überprüfen Sie auch den korrekten Zweikanalbetrieb, d.h. die Links-/Rechts-Trennung (Stereobetrieb) mit Hilfe der Balance-Schieber des Windows-Sound-Wiedergabegeräts Lautsprecher, die Sie über einen Klick auf das Lautsprechersymbol in der Windows-Taskleiste und in dem sich öffnenden Fenster durch einen weiteren Klick auf das große Lautsprechersymbol ganz oben erreichen (dort über die Seite Pegel → Balance, siehe Laborversuch Mittelungsfilter, Kapitel 3.2, Abs. 3.4).

3.3 Erweitern Sie die C-Quellcode-Datei *fir.c* um die als Kommentar vorgegebene **FIR-Filterberechnung** über zwei `for`-Schleifen.
Denken Sie auch an die Koeffizientendatei und alle benötigten Variablen (beachten Sie dabei auch die in `main()` mit dem Schlüsselwort `register` vereinbarten).

3.4 Was bewirkt das **Schlüsselwort `register`** bei den Variablenvereinbarungen in `main()`? Warum wurde es bei diesen Variablen angegeben? (Vorbereitung!)

3.5 Compilieren/Linken Sie das Programm wieder, und laden Sie es auf das DSK. Testen Sie wieder kurz die korrekte Funktion mit einem Sinussignal aus dem (Software-) Funktionsgenerator waveGen und mit passenden Musikbeispielen über den Kopfhörer.

3.6 Ermitteln Sie den **Amplitudengang** Ihres Filters mit Hilfe der Wobbel-Funktion (Sweep-Measurement) des Messprogramms audioTester (siehe Laborversuch Mittelungsfilter, Kapitel 3.2, Abs. 3.4).
Stimmt der Betragsgang mit dem von Ihnen erwarteten Verlauf überein?

Erzeugen Sie einen Ausdruck des Amplitudengangs, so dass ein Vergleich mit dem Ergebnis des Filterentwurfs aus Kapitel 2.2 Abs. 2.4 möglich ist.

3.7 Generieren Sie mit Hilfe Ihres MATLAB m-files einige andere Koeffizientensätze (beispielsweise auch Hochpass- oder Tiefpassfilterverhalten) und testen Sie diese ebenfalls. (Achten Sie bei einer Variation der Ordnung auf die nur begrenzt zur Verfügung stehende Rechenzeit.)

Ermitteln Sie in diesem Zusammenhang empirisch die **maximal mögliche Filterlänge**

indem Sie eine Koeffizientendatei mit z.B. 180 Einträgen erzeugen und den Wert für L, ausgehend von 131, in einer geeigneten Schrittweite erhöhen bzw. verändern.
Ein Überschreiten der zur Verfügung stehenden Rechenzeit macht sich durch Verzerrungen bemerkbar (beim Test also genau hinhören!). Natürlich weicht das mit reduzierter Koeffizientenzahl erzielte Filterergebnis vom Wunschverlauf ab, passieren kann aber nichts (z.B. Schwingen o.ä.), da FIR-Filter ja immer stabil sind.

Anmerkung:

Die hier ermittelte maximale Filterlänge lässt sich durch verschiedene Optimierungsmaßnahmen weiter steigern (beispielsweise durch diverse Compiler-Einstellungen oder durch Verwenden einer in Assembler programmierten, optimalen Filter-Routine. TI bietet dafür die sog. DSP-LIB mit diversen Routinen: sprc121.zip mit spru657c.pdf).
Insbesondere wirkt sich ein Abschalten der Debug-Möglichkeit in der Entwicklungsumgebung stark auf die erreichbare Performance des Systems aus.

Compiler-Einstellungen: Project -> Build Options... -> C6000 Compiler

und dort unter

- Basic Options: Debugging model (none)
  Optimization Level
- Optimizations: Optimize for speed

Diese Dinge müssen/sollen hier aber nicht berücksichtigt werden.

3.8 Kommentieren Sie die vorgegebene Filterberechnungsroutine jetzt wieder aus und programmieren Sie stattdessen die **direkte Form nach Abbildung 1.2** bzw. nach Gl. 1.4 mit jeweils nur einer `for`-Schleife und der dann nötigen "Zeigerverwaltung" (beachten Sie dabei die vorgegebenen Variablenvereinbarungen in `main()` mit dem Schlüsselwort `register`).

3.9 Wiederholen Sie die Ermittlung der maximal möglichen Filterlänge. Wie viele Koeffizienten lassen sich jetzt noch berücksichtigen?

3.10 Wiederholen Sie die Ermittlung der maximal möglichen Filterlänge, diesmal aber ohne die Angabe des Schlüsselwortes `register` bei den Variablenvereinbarungen in `main()`. Wie viele Koeffizienten lassen sich jetzt noch berücksichtigen?

Die Fragen sind handschriftlich in der Versuchsvorbereitung bzw. Ihrem Versuchsprotokoll zu beantworten!