BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Realisierung und Analyse des FIR-Filters

Laborbericht

angefertigt von

Robby Kozok, Nic Frank Siebenborn, Pascal Kahlert

in dem Fachbereich VII – Elektrotechnik - Mechatronik - Optometrie –
für das Modul Digitale Signalverarbeitung III
der Beuth Hochschule für Technik Berlin im Studiengang
Elektrotechnik - Schwerpunkt Elektronische Systeme

Datum 18. Dezember 2015

Lehrkraft
Prof. Dr.-Ing Marcus Purat Beuth Hochschule für Technik

Einleitung

Inhaltsverzeichnis

Kapitel 1

Realisierung des FIR-Filters

1.1 Aufgabenstellung

Die Aufgabenstellung bestand darin, ein Mittelwertfilter zu realisieren und auf verschieden Weisen zu implementieren. Anschließend wurden die Implementierungen auf Performance untersucht und verglichen. Es handelt sich jedes mal um ein Mittelwertfilter vierter Ordnung.

1.2 Durchführung

Gemäßder Umdrucks wurde ein Projekt erstellt und die entsprechenden Dateien angefügt. In einem ersten Teil wurde dann die Sprungantwort simmulativ durch händisches verändern der Eingangswerte ermittelt und aufgenommen. In der vollenden Tabelle sind die von uns manipulierten Werte und die in sDAC1L resultierenden Werte aufgezeigt.

Eingang	Ausgang
0	0
25716	5143
25716	10286
25716	15429
25716	20572
25716	25716
25716	25716

Im nachvollgenden Bild ist diese Annäherung grafisch dargestellt. Man erkennt gut, dass sich der Ausgangswert sukzessive und linear dem Eingangswert annähert. Dies passiert in fünf Schritten, wie es für ein solches Filter üblich ist und entspricht damit den Erwartungen

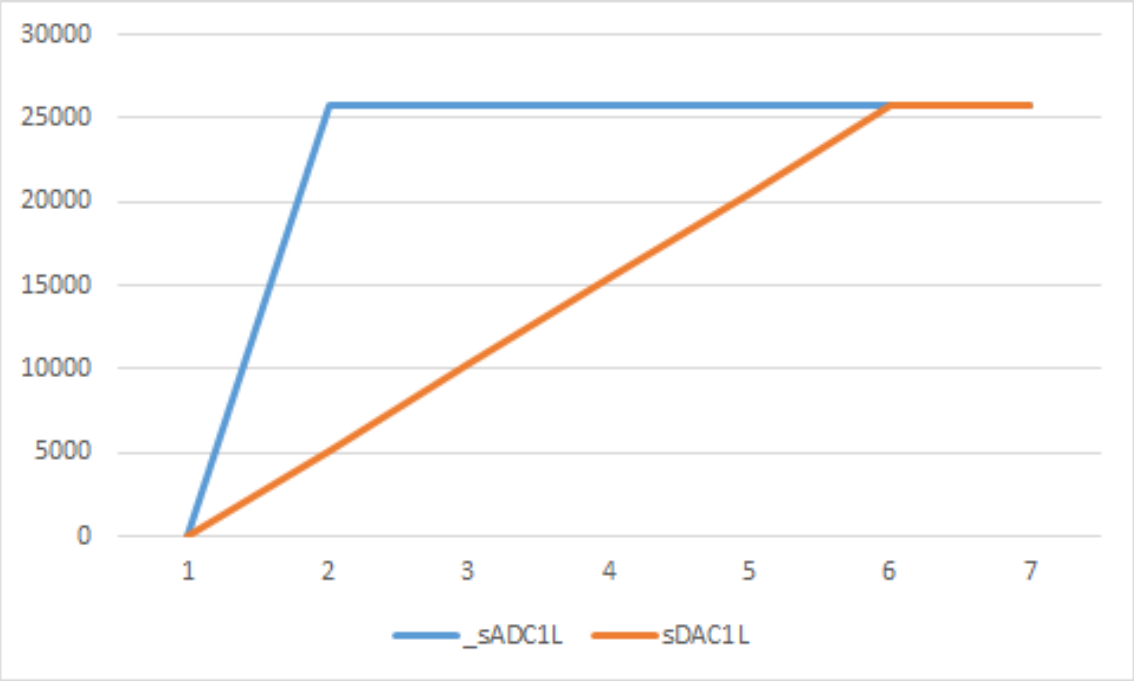

Abbildung 1.1: Sprungantwort des FIR Filters_Mode_Sprungantwort1.png

In der Vorbereitung wurde die Sprungantwort eines Filters vierter Ordnung ermittelt, welche auf dem folgenden Bild zu sehen ist.

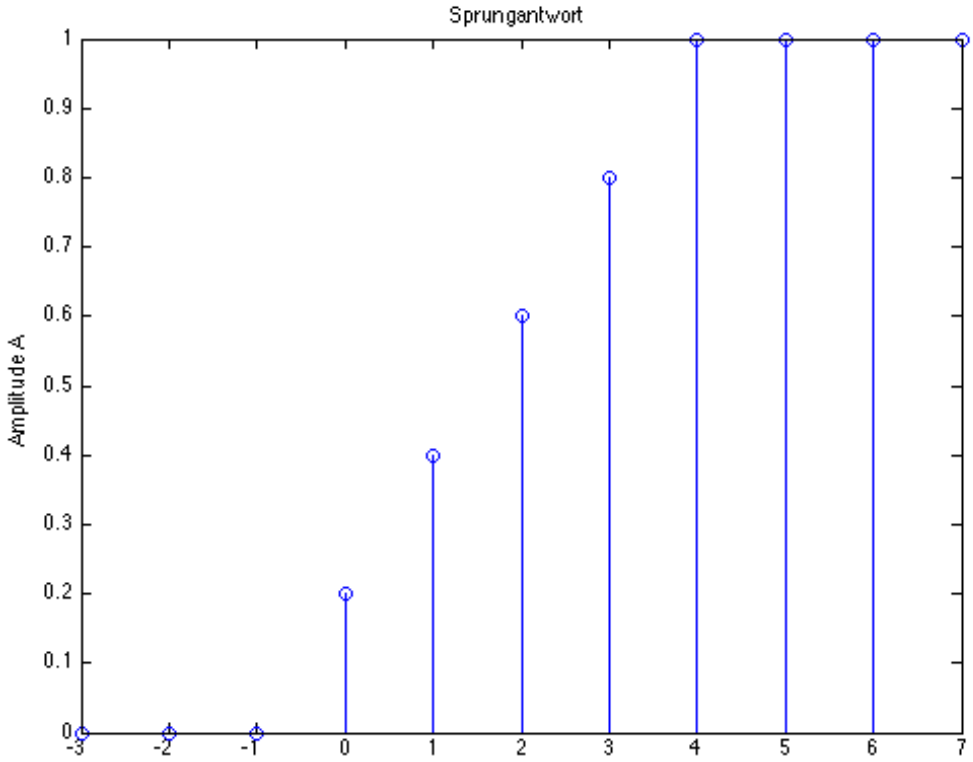

Abbildung 1.2: Sprungantwort des FIR Filters aus der Vorbereitung_Mode_SprungantwortMittelwert.png

Trotz der Unterschiedlichen Darstellungsarten ist eindeutig zu sehen, dass sich die Filter gleich verhalten.
Es wurde eine Programmzyklenzahl von 1672 ermittelt.
Im nächsten Schritt wird diese Herangehensweise wiederholt, diesmal allerdings mit Zahlen im 1.15 Format. Dazu wurden die Variablen auf den Typ short Integer im C-Code angepasst.

```
1  #include "fir.h"
2
3  short fir(short sInput, FIRstate *pFIR) {
4
5          int k;
6  //      float acc; //Datentyp gemaess Aufgabe angepasst
7          int acc;
8
9          *(pFIR->p++)=sInput;    // store current sample in delayline
10         if ( pFIR->p >= pFIR->d + pFIR->N) pFIR->p-=pFIR->N;
11
12         for (acc=0,k=0;k<pFIR->N;k++) {
13                 acc += (*(pFIR->p++) * pFIR->h[k]);
14                 if ( pFIR->p >= pFIR->d + pFIR->N) pFIR->p-=pFIR->N;
15         }
16
17         return (short)(acc >> 15); //rightshift um nur hoeherwertige Bits zu verwenden
18 }
```

fir.c

Wie zu sehen ist wurden im Vergleich zur Ursprungsversion zwei Änderungen vorgenommen: acc wurde zu einem Integer und der Typecast im return-statement wurde um einen Rechtsshift erweitert. Die erste Änderung erklärt sich aus der Aufgabe da mit short Integer gerechnet werden sollte, wobei acc das ergebnis einer Multiplikation ist, welche so viele Bits hat wie beide Summanden zusammen, deshalb int statt short. Der Rechtsshift sorgt dafür, dass nur die 15 höherwertigen Bits zum short getypecastet werden.

Wir überprüften die Funktionsweise wie oben und erhalten folgende Messwerte:

0	0
0,5	0,1
0,5	0,2
0,5	0,3
0,5	0,4
0,5	0,5
0,5	0,5

woraus folgende Grafik entsteht

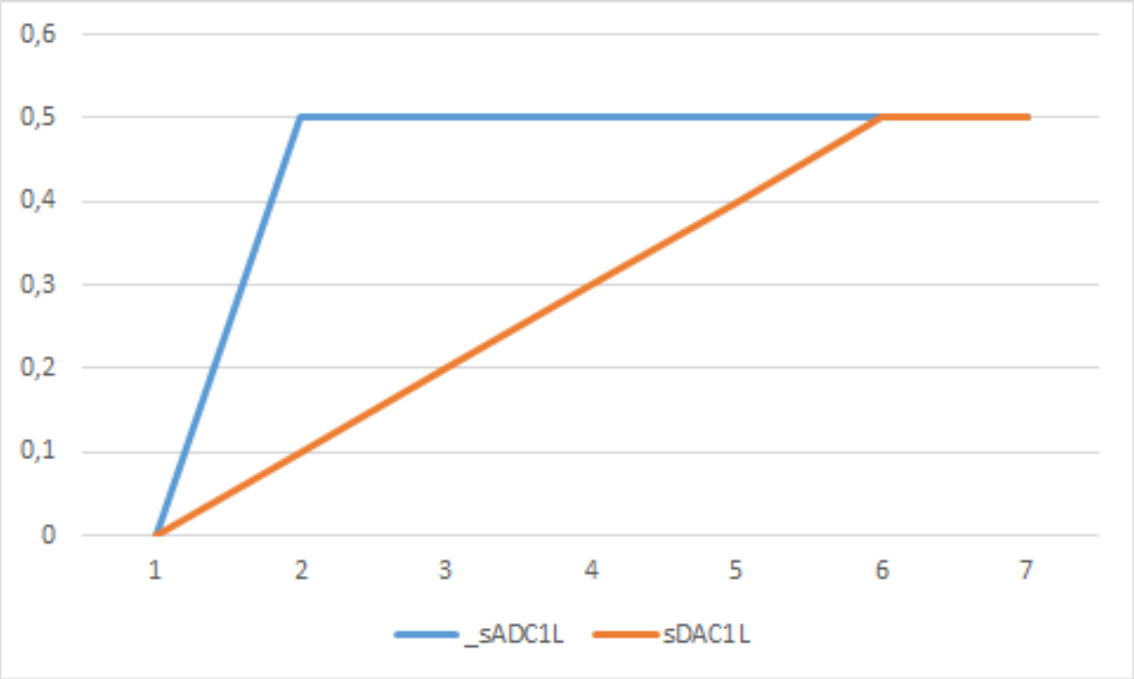

Abbildung 1.3: Sprungantwort des FIR Filters im short Integer Format_Mode_Sprungantwort2.png

Wieder sind alle Werte wie erwartet. Die Messung ergab eine Programmzyklenzahl von 764, was eine deutliche Performanceverbesserung gegenüber des Versuches zuvor ist, was sich darin begründet, dass der DSP für 16-Bit-Operationen optimiert ist.
In einem weiteren Optimierungsschritt wird das Filter nun in Assembler umgesetzt um Optimierungen zu nutzen die der Compiler sonst nicht nutzt, wir erwarten eine schnellere Abarbeitung. Dazu wurde die Datei fir.asm im Bereich für die Funktion _fir: angepasst:

```
19 //!! Begin of Setting I, L, and B regs
20 B1 = P1; //Basis: Erstes Verzoegerungsglied, hier delay line.
21 R1 = R1 << 1; //Mal 2, weil Short Int 2 Byte hat.
22 L1 = R1; // Laenge festlegen auf 2 mal 2 Byte.
23 I1 = P2; // I1 auf Read/Write legen
24 I2 = P0; // I2 auf Filterkoeffizienten
25 //!! End of Setting I, L, and B regs
```

fir.c

Die Register werden so konfiguriert, dass die Funktionalität des Filters abgebildet wird, vergleiche hierzu den kompletten Code siehe Anhang. I2 beinhaltet dabei die übergebenen Filterkoeffizienten und wird mit I1 multipliziert, was den aktuell übergebenen Werten entspricht. L1 ist die Länge des zyklischen Speichers und muss damit der Ordnung des Filters multipliziert mit dem Speicherbedarf eines Wertes, hier 2 Byte multipliziert werden. Der Anfang des Speichers muss logischer Weise der Anfang des Filters sein, hier Parameter P1.

Auf die Aufnahme einer Messreihe und der entsprechenden Grafik wird der Übersichtlichkeit halber ab hier verzichtet, da die Versuche gezeigt haben, dass die Messreihen jedes mal identisch aussehen, lediglich die Programmzyklenzahl hat sich verändert. Sie beträgt in diesem Fall 100, was einer weiteren Performanceverbesserung und unseren Erwartungen aus oben gegebenem Grund entspricht.

Der letzte Optimierungsschritt nutzt die Fähigkeit des DSPs zur parallelen Abarbeitung zweier MAC-Operationen aus. Dazu wurde die Funktion zum Aufruf der Filterfunktionalität wie in der Aufgabe beschrieben ersetzt, die isr.c zum Aufruf der Stereofunktion durch Auskommentieren verändert und fir.asm wie folgt angepasst. Im Teil _fir_stereo: wurden Register wie folgt angepasst.

```
27 //!! Begin of Setting I, L, and B regs
28 B1 = P1; //Basis: Erstes Verzoegerungsglied, hier delay line.
29 R1 = R1 << 2; // 4*R1 da 32 Bit -> 8Bit*4=32
30 L1 = R1; // Laenge festlegen auf 4 mal 2 Byte.
31 I1 = P2; // I1 auf Read/Write legen
32 I2 = P0; // I2 auf Filterkoeffizienten
33 //!! End of Setting I, L, and B regs
```

fir.c

Diese konfiguration entspricht der selben wie in der Aufgabe zuvor, mit dem Unterschied, dass der Speicher doppelt so lang ist. Dies begründet sich darin, dass nun zwei Werte gleichzeitig bearbeitet werden.
Auch in diesem Fall wurde die Programmlaufzeit ermittelt und ist mit 52 Zyklen die schnellste, auch dies haben wir auf Grund der nun parallelen Abarbeitung erwartet.

1.3 Auswertung

Es soll die maximal mögliche Anzahl an Ausführungszyklen für die Laborsituation anhand der ermittelten Programmzyklenzahlen berechnet werden.

Art der Realisierung	Anzahl Programmzyklen
Fliesskommarealisierung in C	1672
Festkommarealisierung in C	764
Festkommarealisierung in asm	100
Festkommarealisierung in asm mit SIMD	52

Bei einer Abtastfrequenz von $f_{abtast} = 48kHz$und einer Befehlsfrequenz von $f_{Befehl} = 500MHz$ ergibt sich eine maximale Programmzyklenzahl von

$$N_{ZyklMax} \leq f_{Befehl}/f_{abtast} \leq 10416 \tag{1.1}$$

Zur Berechnung der maximalen Filterordnung in den zwei Fällen der C-Implementierung genügt ein einfacher Dreisatz:

$$N_{OrdnMaxCFloat} + 1 \leq 5 * 10416/1672 \leq 31,15 \text{ ergibt } N_{OrdnMaxCFloat} = 30 \tag{1.2}$$

$$N_{OrdnMaxC1.15} + 1 \leq 5 * 10416/764 \leq 68,17 \text{ ergibt } N_{OrdnMaxC1.15} = 67 \tag{1.3}$$

Bei den Realisierungen in Assembler ist der Overhead hingegen zu beachten, da die Schleifenabarbeitung genau einen Befehlstakt beträgt. Daher wird von der genutzten Zyklenzahl zwei mal die Filterkoeffizientenzahl abgezogen(zwei da der linke und der rechte Audiokanal berechnet werden müssen). Und das ergebnis dann halbiert, wegen der nicht parallelen Berechnung.

$$N_{OrdnMaxAsm} + 1 \leq (10416 - (100 - 2 * 5)/2) \leq 5163 \text{ ergibt } N_{OrdnMaxAsm} = 5162 \tag{1.4}$$

Bei der Berechnung dieser Werte mit SIMD-Befehlen fällt der Faktor 2 natürlich weg.

$$N_{OrdnMaxAsmSIMD} + 1 \leq 10416 - (52 - 5) \leq 10369 \text{ ergibt } N_{OrdnMaxAsmSIMD} = 10368 \tag{1.5}$$

Die Berechnungen stützen die Erwartung, dass die Programme immer performanter wurden und dadurch auch höherwertige Filter verarbeiten können.

Kapitel 2

Analyse des FIR-Filters

2.1 Aufgabenstellung

Im zweiten Teil des Laborversuchs sollte das Verhalten des Filters mit den Ergebnissen der Vorbereitung verglichen werden.
Außerdem soll ein FIR-Filter höherer Ordnung, mithilfe eines MatLab-Tools entworfen werden.

2.2 Durchführung

Zur Analyse des in 1 erstellten Filters wurde ein Sinussignal an den Eingang des Filters angelegt werden. Durch alternieren der Frequenzen des Signals, konnten wir dann die Nullstellen anhand der Amplitude des Ausgangssignals ermitteln.
Diese Analyse wurde dann mit Aufnahme des Amplitudenganges vertieft.
Zum aufnehmen der Sprungantwort wurde dann ein Rechtecksignal mit 2000Hz. Diese Frequenz wurde ausgewählt, da ein zu schnelles Signal dazu führen würden, dass die Sprungantwort des Filters nicht in eine komplette Periode des Rechtecksignals passen würde.

Im zweiten Teil dieses Versuchteils, wurde ein Filter höherer Ordnung mit einem Filter Design and Analysis (FDA)-Tool entworfen. Dabei sollte folgende Parameter genutzt werden:

- Equiripple-Charakteristik
- Passband-Frequenz: 3400 Hz
- Stoppband-Frequenz: 4000 Hz
- Abtastfrequenz: 48000 Hz
- Passband-Welligkeit: 2dB
- Stoppband-Dämpfung: 80dB

Die dadurch erzeugten Filter Koeffizienten wurden dann im fractional Format in die bereits vorhandene C-Header-Datei eingefügt. Dies ist in dem untenstehenden Quellcode-Ausschnitt zu sehen.

```
1  // Definition der Filterkoeffizienten
2  #define N_FILT 184 // Anzahl der Koeffizienten
3
4  const short coef[N_FILT] = {
5          -5,     -12,     -25,     -43,     -67,     -98,    -132,    -168,    -200,
6        -226,    -239,    -237,    -215,    -175,    -119,     -51,      21,      89,
7         144,     179,     189,     173,     134,      79,      18,     -39,     -81,
8        -100,     -92,     -59,      -7,      54,     112,     155,     174,     163,
9         124,      64,      -5,     -69,    -115,    -132,    -114,     -64,       9,
10         91,     164,     213,     224,     193,     124,      31,     -70,    -155,
11       -206,    -209,    -160,     -66,      56,     181,     282,     334,     323,
12        245,     112,     -52,    -214,    -336,    -390,    -355,    -231,     -36,
13        196,     416,     575,     630,     555,     347,      33,    -336,    -686,
14       -939,   -1017,    -864,    -454,     204,    1058,    2024,    2996,    3856,
15       4500,    4843,    4843,    4500,    3856,    2996,    2024,    1058,     204,
16       -454,    -864,   -1017,    -939,    -686,    -336,      33,     347,     555,
17        630,     575,     416,     196,     -36,    -231,    -355,    -390,    -336,
18       -214,     -52,     112,     245,     323,     334,     282,     181,      56,
19        -66,    -160,    -209,    -206,    -155,     -70,      31,     124,     193,
20        224,     213,     164,      91,       9,     -64,    -114,    -132,    -115,
21        -69,      -5,      64,     124,     163,     174,     155,     112,      54,
22         -7,     -59,     -92,    -100,     -81,     -39,      18,      79,     134,
23        173,     189,     179,     144,      89,      21,     -51,    -119,    -175,
24       -215,    -237,    -239,    -226,    -200,    -168,    -132,     -98,     -67,
25        -43,     -25,     -12,      -5
26 };// Von MatLab generierte Filter Koeffizienten
```

process_data_KompFIR.c

In Zeile 2 wurde die Anzahl der Koeffizienten festgelegt und in Zeile 4 bis Zeile 26 wurden die Koeffizienten eingetragen, außer diesen Änderungen war es nicht notwendig den Quellcode anzupassen.

2.3 Auswertung des Mittelwert FIR-Filter

Für diesen Aufgabenteil haben wir jeweils zwei Minimalfälle und zwei normale Fälle ausgewählt, alle Eingangssignale haben eine Amplitude von 1V. Diese sind in Abbildung 2.1 bis Abbildung 2.4

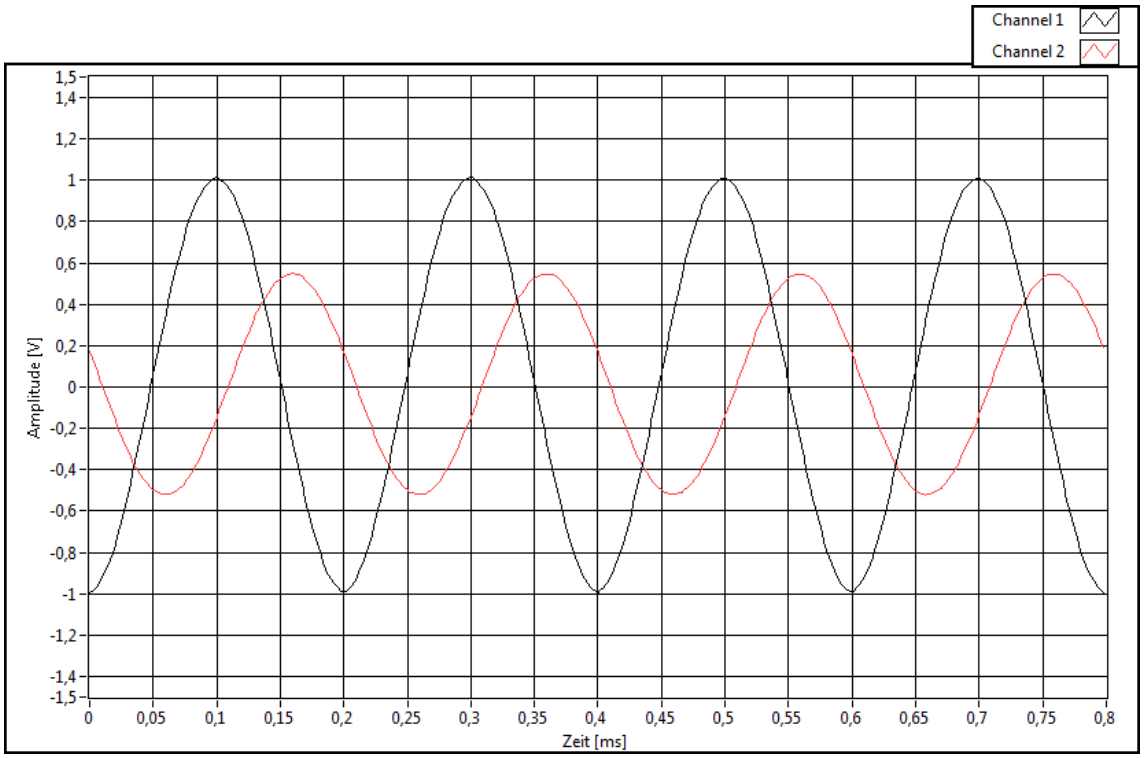

Abbildung 2.1: Frequenz: 5 kHz

In Abbildung 2.2 und Abbildung 2.4 sind die Auswirkungen der Nullstellen des Filters zu sehen. Dort ist das Ausgangssignal fast Null. In Abbildung 2.1 und Abbildung 2.3 ist zu sehen, dass das Signal gedämpft ist aber deutlich größer Null ist. Diese Ergebnisse entsprechen der Vorbereitung.

Die Verzögerung zwischen Eingangssignal und Ausgangssignal ist, wie bereits im ersten Laborversuch, dem Versuchsaufbau zu verschulden, allerdings treten nun durch den Filter wiederum Verzögerungen auf. Diese Verzögerungen sind damit zu erklären, dass der FIR Filter den Ausgang immer aus den letzten N_FILT Werten bestimmt. Damit folgt auch das ausgegebene Signal diesem Muster.

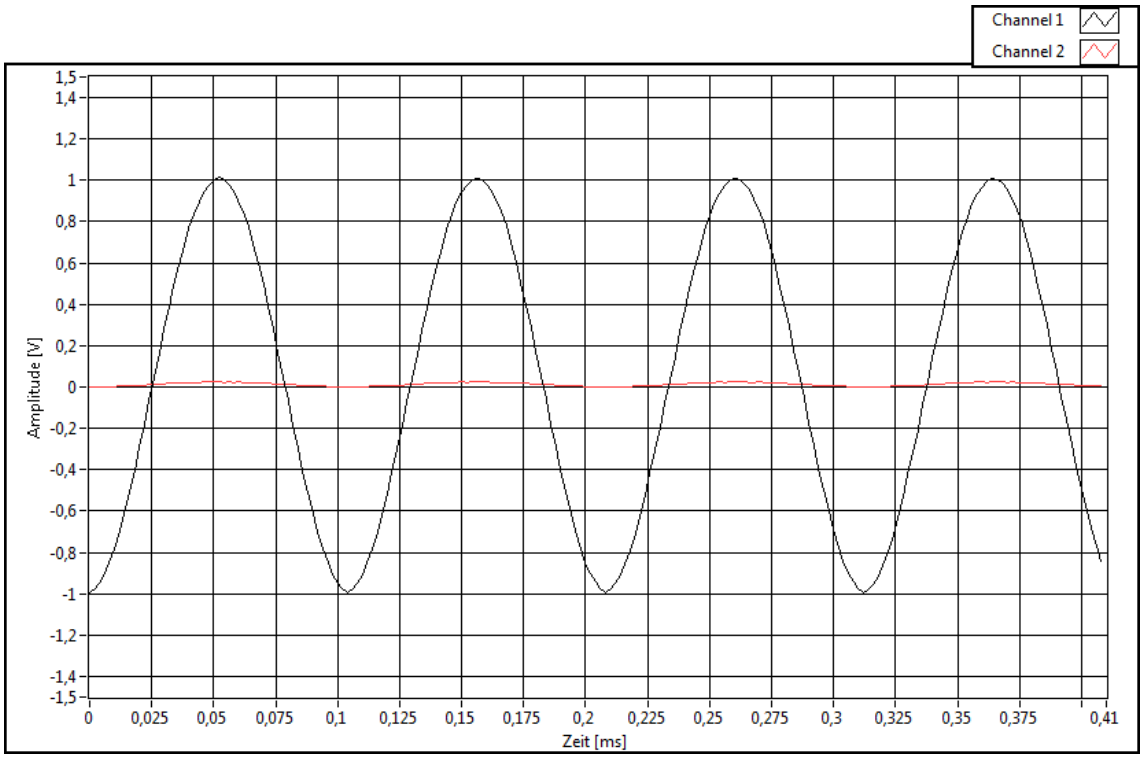

Abbildung 2.2: Frequenz: 9,6 kHz

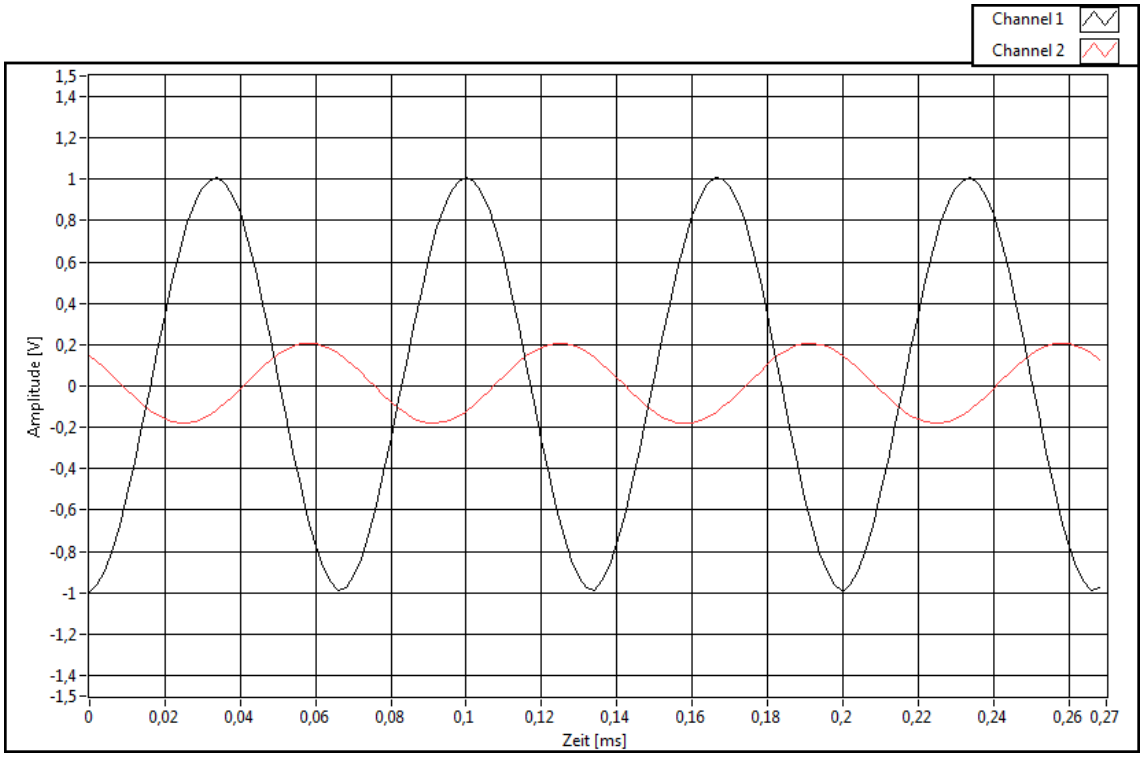

Abbildung 2.3: Frequenz: 15 kHz

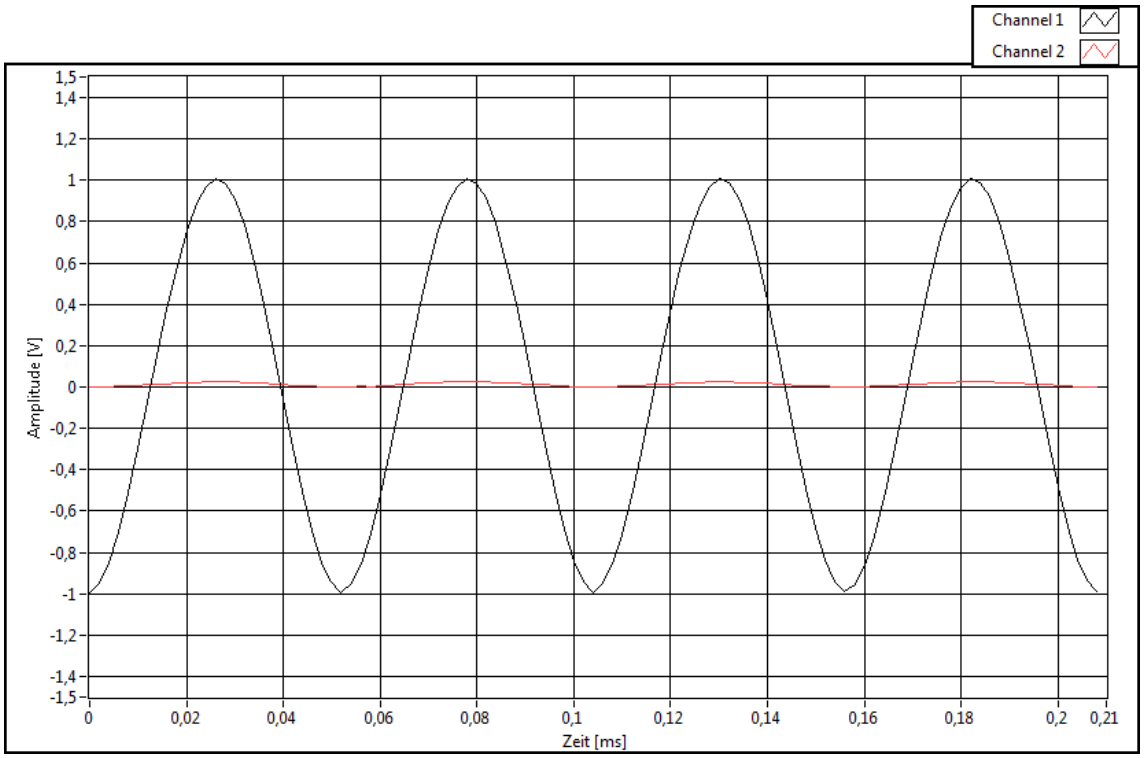

Abbildung 2.4: Frequenz: 19,2 kHz

Die Nullstellen lassen sich des weiteren im Amplitudengang ablesen, dieser ist in 2.5 zu sehen. Dort ist bei ungefähr 9,6 kHz ein Einbruch auf -68dB und ein Einbruch auf -65dB bei 19,2 kHz zu sehen.
Auffällig, aber nicht weiter überraschend, ist der Verlauf des Signals bei ca. 24 kHz. Dieser folgt aus dem Tiefpassverhalten des Codecs.

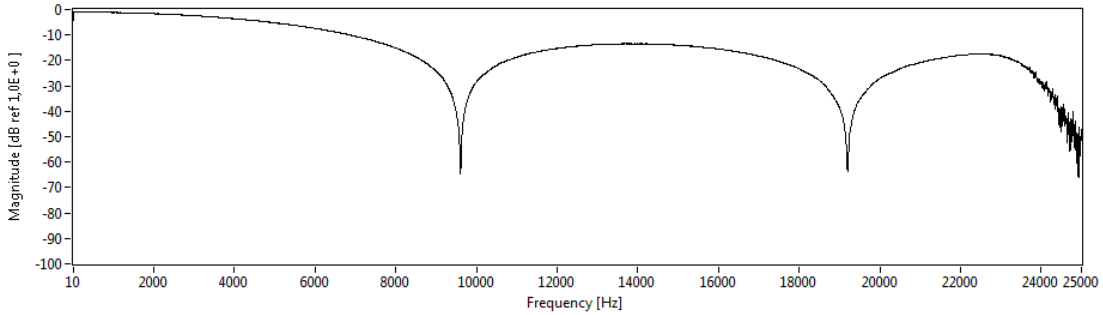

Abbildung 2.5: Amplitudengang des FIR Mittelwertfilters

Im weiteren Verlauf soll nun die Sprungantwort des Filters analysiert werden.

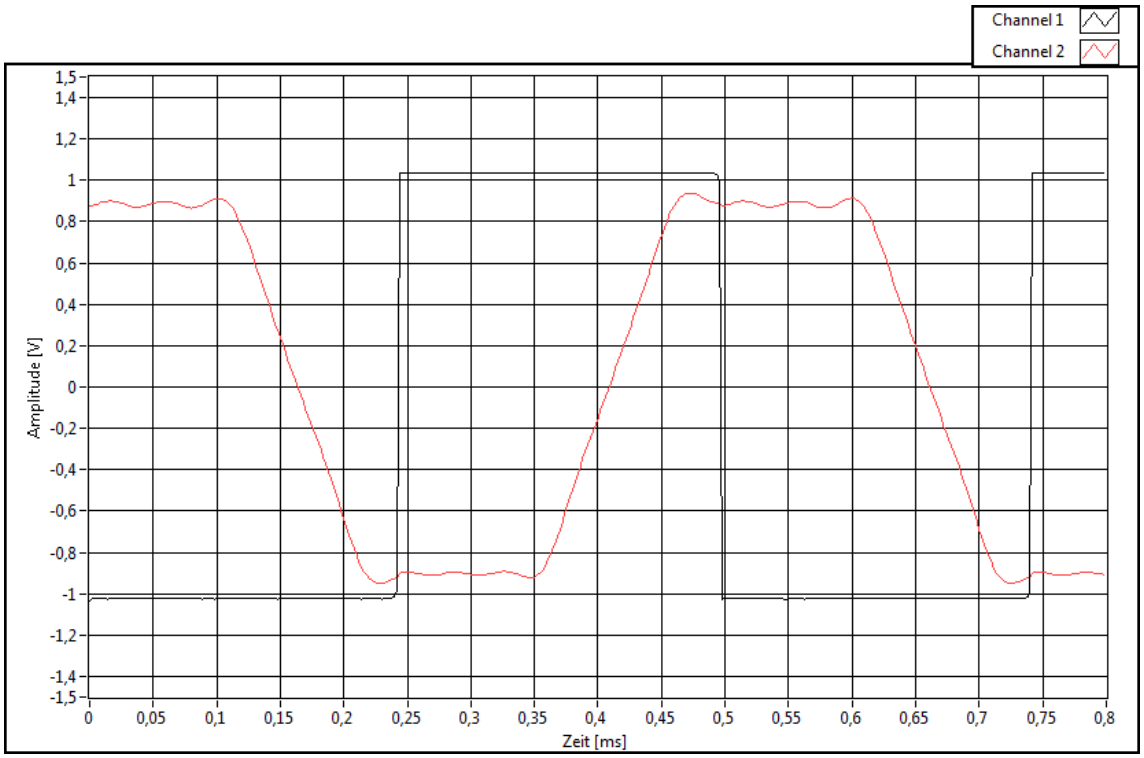

Abbildung 2.6: Sprungantwort des FIR Mittelwertfilters

Der Sprung ist mit dem schwarzen Graphen dargestellt und die Sprungantwort mit dem roten Graphen. Der lineare und leicht verzögerte Anstieg der Sprungantwort entspricht dem verhalten eines FIR Filter. In der Theorie müsste diese lineare Gerade eine Treppenform haben, diese Form wird allerdings durch das Tiefpassverhalten des Systems geglättet. Wie bereits bei der Analyse der Ausgangssignale erwähnt tritt eine Anstiegszeit von N_FILT Werte auf, in diesem Beispiel sind dies 5 Werte, also 5 Abtastwerte.

Damit lässt sich die Anstiegszeit berechnen.

$$T_{rise} = N\_FILT * \frac{1}{f_{Abtast}} \tag{2.1}$$

Diese Anstiegszeit deckt sich, mit leichter Abweichung, mit Abbildung 2.6.

2.4 Auswertung des FIR-Filter höherer Ordnung

Das Matlab FDA-Tool erzeugt neben den Koeffizienten auch die Graphen welche das Verhalten des erzeugten Filters darstellen. Der ideale Amplitudengang des erzeugten Filters ist in Abbildung 2.7 zu sehen.

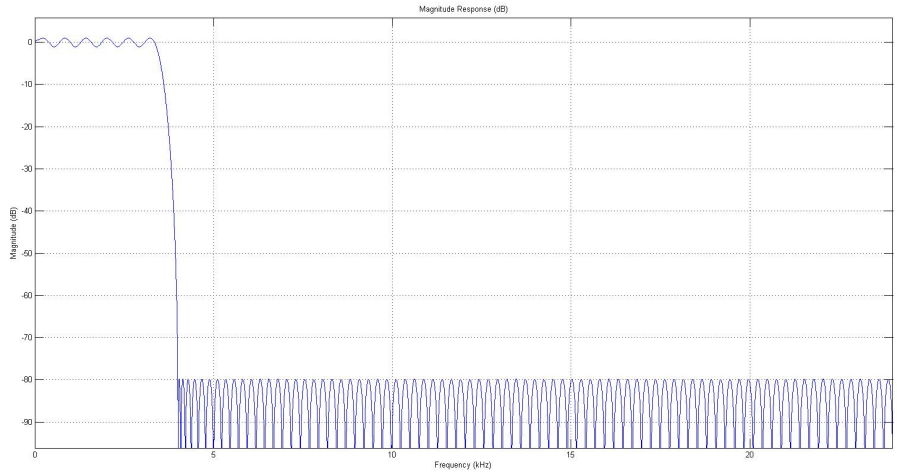

Abbildung 2.7: Idealer Amplitudengang des erzeugten Filters

Im Vergleich dazu soll nun in Abbildung 2.8 der reale Amplitudengang gezeigt werden.

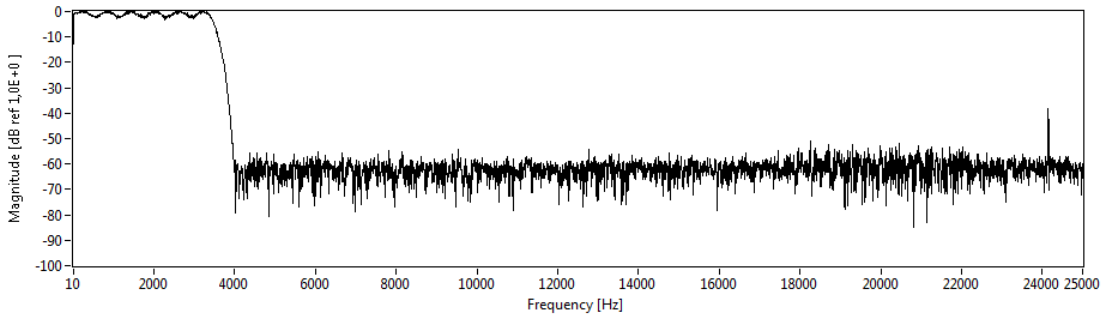

Abbildung 2.8: Realer Amplitudengang des erzeugten Filters

Beide Amplitudengänge ähneln einander. Im Durchlassbereich ist im realen Amplitudengang eine leichte Dämpfung zu erkennen. Diese liegt zwischen 1 dB und 2 dB und ist auf die Skalierung von ADC und DAC zurückzuführen. Die Grenzfrequenz beträgt in beiden Abbildungen ca. 4 kHz.
Im Sperrbereich ist deutlich zu erkennen das die Dämpfung des idealen Amplitudengangs nicht erreicht wird. Es werden lediglich -60 dB erreicht, im idealen Amplitudengang sind

es jedoch -80 dB.

Die Sprungantwort des idealen Filters ist in Abbildung 2.10 und die Sprungantwort des realen Filters in Abbildung 2.9 zu sehen.

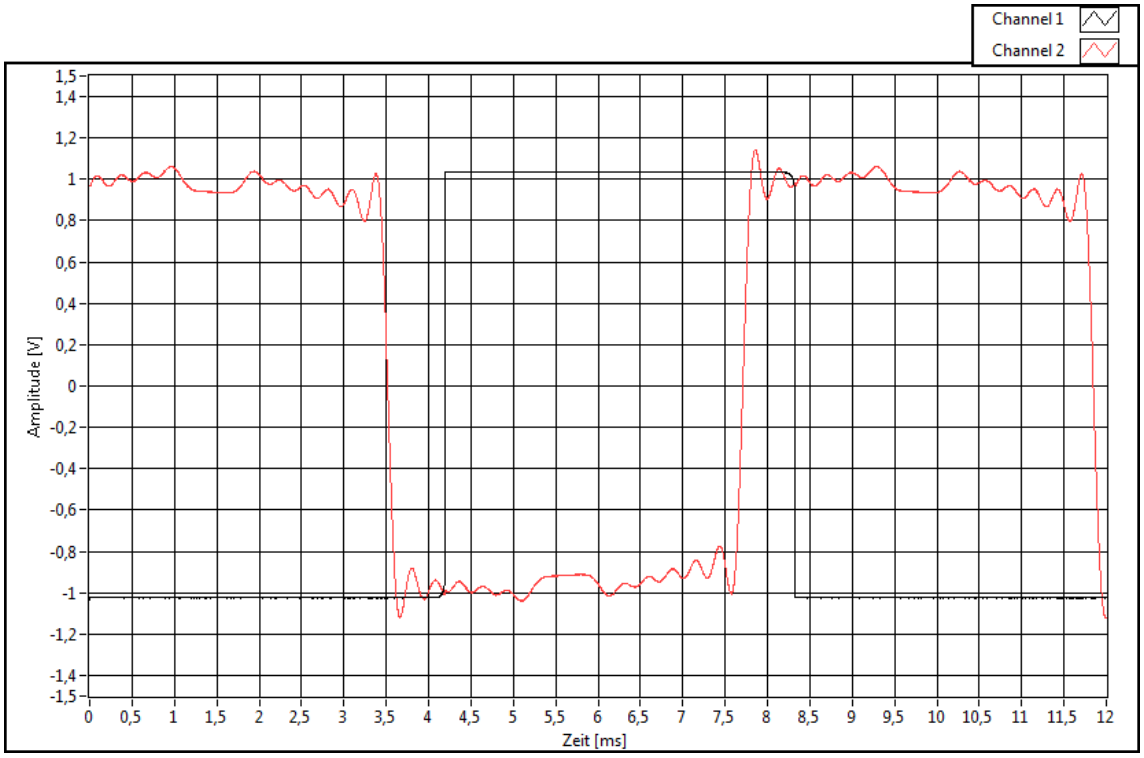

Abbildung 2.9: Reale Sprungantwort des erzeugten Filters

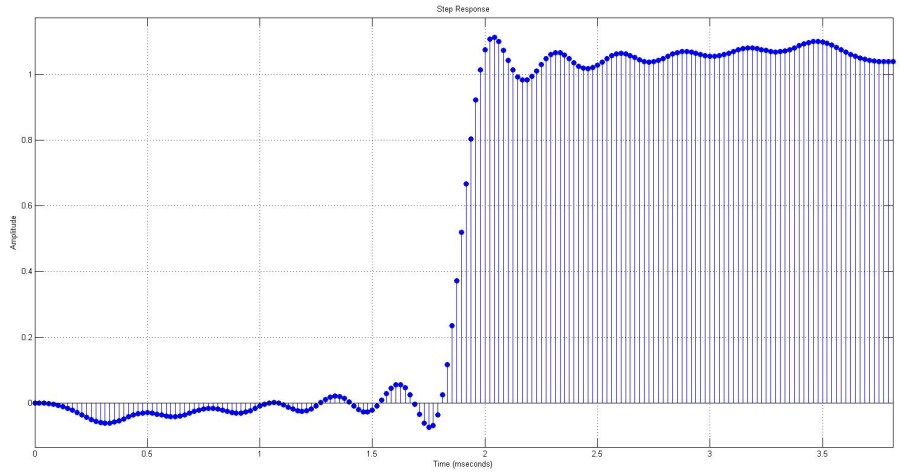

Abbildung 2.10: Ideale Sprungantwort des erzeugten Filters

Das Verhalten des realen Filters entspricht sehr genau unseren Erwartungen. Es ist lediglich eine leichte Dämpfung zu erkennen. Die Anstiegszeit, sowie das Überschwingverhalten sind fast identisch.

Anhang A

Quelltext-Dateien

Abbildungsverzeichnis

Abkürzungsverzeichnis

FDA Filter Design and Analysis. 7, 13