

STUDIENARBEIT

Sommersemester 2001

Digitale Signalverarbeitung

Lärmunterdrückung in Gegensprechanlagen

Daniel Kägi, Michael Meyer

Inhaltsverzeichnis

1.0	Abstract	2
2.0	Überblick	3
3.0	Aufgabenstellung	4
4.0	Signalanalyse	6
4.1	Fahrzeugmotoren	6
4.1.1	Personenwagen Mazda 626 2.0 Ottomotor	7
4.1.2	Mercedes Lieferwagen Dieselmotor	8
4.1.3	Volvo Lastkraftwagen Dieselmotor	9
4.2	Sprechen und Hören	10
4.2.1	Sprachanalyse	10
4.2.2	Gehör	11
5.0	Ansätze zur Lärmunterdrückung	12
5.1	Verschiebung	12
5.2	Beamforming	14
5.3	Adaptive Verfahren	15
5.3.1	Zwei Mikrofone	15
5.3.2	Ein Mikrofon	17
5.4	Mehrkanalverfahren	18
6.0	Realisation	22
6.1	Aufbau	22
6.2	Bändereinteilung des Mehrkanalsystems	24
6.3	GUI	26
6.4	Abschliessende Beurteilung	27
7.0	Erläuterungen zum Programmcode	28
7.1	Aufbau	28
7.2	Implementationshinweise	28
7.3	Compilieren	28
8.0	Zusammenfassung	29
9.0	Referenzen	31
10.0	Anhang	32
10.1	Verzeichnisstruktur	32
10.2	Abbildungsverzeichnis	33
10.3	Quellcode	34
10.3.1	C-Quellcode	34
10.3.2	m-Quellcode	45

1.0 Abstract

In vielen Kommunikationssystemen ist die Sprachübertragung einer der wichtigsten Anwendungszwecke. Dabei ist an der Schnittstelle zwischen Mensch und Kommunikationssystem meistens ein Mikrofon für die Signalaufnahme zuständig. Das Mikrofon kann jedoch nicht zwischen menschlicher Sprache und Störgeräuschen unterscheiden und zeichnet deshalb die ganze Geräuschkulisse auf. Wenn dadurch die Verständlichkeit beeinträchtigt wird, hervorgerufen durch eine zu stark beeinflussende Störquelle, kann dies zu Kommunikationsschwierigkeiten führen.

In dieser Arbeit wird ein Lärmunterdrückungs-System für eine Gegensprechanlage realisiert und vorgestellt. Diese Anlage wird bei einer Firmeneinfahrt zur Anmeldung benutzt. Dabei wird das Störgeräusch meist von einem Motor (PW oder LKW) verursacht. Gängige Verfahren - wie z.B. das Beamforming, das adaptive Noise Cancelling oder das Filterbankensystem -, welche störende Geräusche herausfiltern, scheitern entweder an den tief- oder hochfrequenten Störeffekten oder an den nicht periodischen Signalanteilen.

Die Lösung, die hier vorgeschlagen wird, ist eine Kombination zweier Verfahren, die die jeweils hervorragenden Eigenschaften der einzelnen Systeme zusammensetzt. Die beiden in Serie geschalteten Entstörglieder sind die adaptive Geräuscherdrückung, als erstes Bindeglied in der Kommunikationskette, und das Mehrkanalsystem, welches das schon fast optimale Signal noch weiter von Störgeräuschen befreit.

Das überzeugende Resultat der ersten Tests im Labor macht sogar den Nachteil, dass zwei Mikrofone benötigt werden, nichtig.

2.0 Überblick

Der schematische Aufbau des gesamten Systems, wie wir es aufgebaut haben, ist in der Abbildung 1 ersichtlich.

Man kann davon ausgehen, dass die Mikrofone später in einer Säule oder etwas Ähnlichem montiert werden. Deshalb wurde beim Systemaufbau darauf geachtet, dass der vertikale Abstand zwischen den beiden Mikrofonen (Sennheiser, System K6 mit Mikrofonmodul ME62, Kugel-Charakteristik) ungefähr einen Meter beträgt, um einer realen Situation möglichst nahe zu kommen. Des Weiteren wurden keine speziellen Massnahmen getroffen, um die Mikrofone akustisch zu trennen. Das Mikrofon1 wurde auf die Störquelle gerichtet und das Mikrofon2 auf den Sprecher.

Die Geräuschdateien wurden auf dem DAT-Rekorder (Sony, TCD-D7) aufgezeichnet und auf einem PC in WAV-Dateien (Soundkarte: Crystal Audio System, Soundeditor: Goldwave Ver. 3.22) umgewandelt. Die Dateien sind via PC und dem Verstärker AMP1 (Sony, TA-F561R) auf zwei Lautsprecher (Technics, SB-3130) ausgegeben worden. Damit das ganze System nicht allzu statisch ist, sind die Lautsprecher zueinander versetzt aufgestellt worden. Somit gibt es kleine Laufzeitverschiebungen des Motorengeräusches, denn auch in realen Situationen ist die Geräuschquelle nicht punktuell. Die beiden Mikrofonensignale wurden mit dem Verstärker AMP2 (Technics, SU-Z150) verstärkt und auf die entsprechenden DSP-Eingangskanäle eingespielen (Blacktip PCI, Single ADSP-2106X PCI System Board). Das Ausgangssignal des DSPs wurde mit einem weiteren Verstärker AMP3 (Mikrofonverstärker des DS-Labors) geregelt und mittels Kopfhörer (Sennheiser, HD580) wiedergegeben.

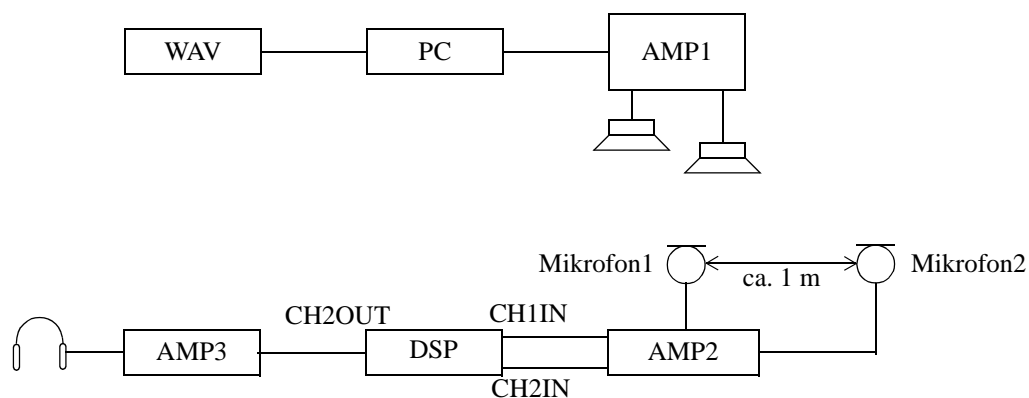


ABBILDUNG 1. Blockschaltbild des Aufbaus

3.0 Aufgabenstellung

Die Unterdrückung von Umweltlärm bei der Übertragung von Sprache aus einer lärmigen Umgebung ist ein Problem, das in verschiedenen Situationen aktuell ist. Zur Lösung sind verschiedene Verfahren aus der digitalen Signalverarbeitung bekannt.

Die klassische Störunterdrückungsmethode, die als "Adaptive Noise Cancelling" bekannt ist [5,6,7], enthält ein adaptives FIR-Filter, welches das Signal eines separaten, auf die Lärmquelle gerichteten Mikrofons so formt, dass es möglichst genau mit dem Störsignal, das dem Sprachsignal im Übertragungsweg überlagert ist, übereinstimmt. Nach einer Subtraktion erscheint dann das Nutzsignal störungsfrei (s. Abbildung 2).

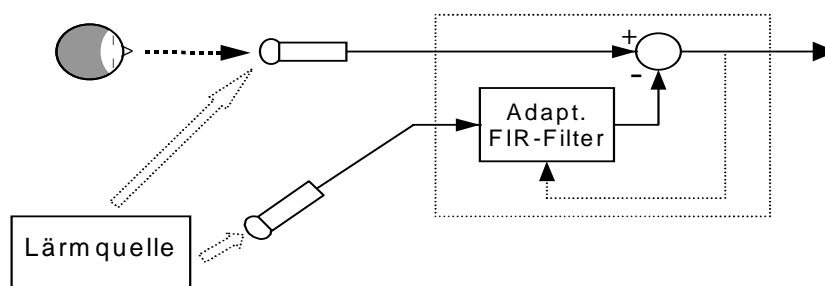


ABBILDUNG 2. Adaptive Noise Cancelling zur Lärmunterdrückung in einer Sprachübertragung

Ein weiteres Verfahren zur Unterdrückung von Lärm im Sprachsignal, das ohne Zweitmikrofon auskommt, nützt die grosse Dynamik, d.h. die grosse und rasche Pegelschwankung aus, die dem Sprachsignal eigen ist und es vom Lärm unterscheidet, da dieser in den meisten Fällen eher stationären Charakter hat. Das Spektrum des Gesamtsignals wird hier in eine grössere Anzahl von Teilbändern zerlegt. Von jedem Teilbandsignal wird die Dynamik geprüft: Ändert sich der Pegel nur langsam, so wird das Teilsignal als dem Lärm zugehörig interpretiert und vor der Wiederauswertung des Gesamtsignals aus den Teilbandsignalen abgeschwächt oder unterdrückt [6].

Weitere Verfahren, deren Anwendung spezifische Eigenschaften des Lärms ausnützen (z.B. Periodizität, Korrelation über grössere Zeitintervalle hinweg, Korrelation mit bekannten Mustern etc.) sind ebenfalls denkbar.

In dieser Aufgabe geht es um den folgenden konkreten Anwendungsfall: Bei der Einfahrt in das Areal einer Firma muss sich der Chauffeur des Lieferwagens über eine Gegensprechanlage anmelden bzw. identifizieren, damit von der Loge aus das Tor zum Firmenareal geöffnet werden kann. Der Chauffeur hält dazu sein Fahrzeug vor dem Tor an, öffnet das Fenster und spricht ins Mikrofon, das in einer Säule neben der Einfahrt angebracht ist - dies während der Motor seines Fahrzeuges läuft. Infolge des Motorenlärms ist die Sprache an der Gegenstation nur schwer verständlich. In dieser Studienarbeit ist die Wirksamkeit der oben erwähnten Lärmunterdrückungsmethoden in der beschriebenen Problematik zu untersuchen.

Aufgabe

Die vorliegende Aufgabe lässt sich grob in folgende Teilaufgaben aufteilen:

- Studium der Grundlagen der möglichen Lärmunterdrückungsverfahren
- Einarbeiten ins Labor-DSP-System
- Sichten, Inbetriebnahme und Anpassung der im Labor vorhandenen geeigneten Algorithmen im DSP-System (Adaptives System, Filterbank-Mehrkanal-Verfahren)
- Erstellen einer Versuchsanordnung zur Nachbildung der Anwendungssituation im Labor
- Austesten, Optimieren und Vergleichen der Algorithmen im Laboraufbau, ggf. auch im Feldversuch
- Ev. Einbezug weiterer Unterdrückungsverfahren

Bericht

Über die Arbeit ist eine Dokumentation zu verfassen. Alle verwendeten Quellen sind im Literaturverzeichnis dieses Berichts anzugeben (Hinweise im Text). Der Bericht ist in doppelter Ausführung abzugeben; ein Exemplar verbleibt am Labor für digitale Signalverarbeitung der HSR, das Doppel erhalten die Verfasser nach der Korrektur und der Bewertung zurück. Die erstellten und verwendeten Programme sowie der Text des Berichtes sind zur laborinternen Archivierung dem Laborassistenten zu übergeben.

Termine, Bedingungen

Gemäss Vorgaben und Terminplan des Vorstandes der Abteilung für Elektrotechnik.
Ausgabe der Aufgabenstellung, Beginn der Arbeit: Montag, 26.03.2001
Arbeitsplatz: DS-Labor 6.003 (zuständig: P. Roffler, Laborassistent)
Abgabe des Berichts, Ende der Arbeit: Freitag, 13. Juli 2001

Assistenz: P. Roffler, Laborassistent (Arbeitsplatz: MEO-Labor)
A. Rüegg (in dringenden Fällen; Arbeitsplatz: DS-Labor).

Rapperswil, 26. März 2001

Prof. Dr. A. Schüeli

Aufgabe in Zusammenarbeit mit der Firma Hugentobler Elektronik, Richterswil.
Kontakt: Th. Hugentobler, Lehrbeauftragter für das DS-Praktikum an der HSR.

4.0 Signalanalyse

In diesem Kapitel werden alle verwendeten Tonaufnahmen beschrieben und auf mögliche Ansätze zur Lärmunterdrückung untersucht. Die Analyse beinhaltet jeweils die Autokorrelation und das Leistungsdichtespektrum der Signale. Die Fouriertransformationen und Autokorrelationen wurden mit Hilfe von Matlab R12 berechnet. Diese Algorithmen wurden jeweils auf die gesamten 20 s angewendet, was zu langen Rechenzeiten führte. Alle Aufnahmen wurden mit dem folgenden Equipment aufgezeichnet und bearbeitet:

- Sennheiser Mikrofon System K6 mit Mikrofonmodul ME62 (Kugel-Charakteristik)
- Sony DAT-Recorder TCD-D7 mit 48 kHz / 16 Bit
- Crystal Audio System 44,1 kHz / 16 Bit
- Soundeditor Goldwave Ver. 3.22

Die verwendeten Tonaufnahmen befinden sich auf dem Datenträger im wav-Format. Sie sind 20 s lang und mit 44,1 kHz / 16 Bit von einem PC digitalisiert worden.

4.1 Fahrzeugmotoren

Die Aufnahmen der Fahrzeuggeräusche sind seitlich auf Fahrerhöhe und am oberen Rand des Radkastens gemacht worden. Die Motoren liefen im Standgas.

Es sei hier vorweggenommen, dass die Motordrehzahl f_m nicht den stärksten Anteil am Frequenzspektrum hat. Die stärkste Frequenz f_0 lässt sich in den meisten Fällen wie folgt berechnen. Sie entspricht der Zündfrequenz der Kerzen.

$$f_m = \frac{u}{60}$$

$$f_0 = n \cdot z \cdot f_m = \frac{n \cdot z \cdot u}{60}$$

$$u = \frac{60 \cdot f_0}{n \cdot z}$$

u: Umdrehungszahl [1/min]

z: Anzahl Zylinder

n: Viertaktmotor $n = 0,5$

f_0 : Stärkste Frequenz [Hz]

4.1.1 Personenwagen Mazda 626 2.0 Ottomotor

Um die klanglichen Unterschiede zwischen Dieselmotor und Ottomotor zu messen, wurde auch ein normaler Personenwagen aufgenommen und analysiert. Dies verhindert ein einseitiges Optimieren der Lärmunterdrückung auf einen speziellen Dieselmotor.

Das Leistungsdichtespektrum zeigt drei deutlich erkennbare Frequenzanteile bei 34 Hz, 68 Hz und 136 Hz. Diese rühren direkt von der Motordrehzahl her. Da dies ein 4-Zylinder Motor ist, ergibt die Berechnung der Leerlaufdrehzahl:

$$u = \frac{60 \cdot f_0}{n \cdot z} = \frac{60 \cdot 34 \text{ Hz}}{0,5 \cdot 4} = 1020 \frac{1}{\text{min}}$$

Da dieser Motor noch nicht ganz warm gelaufen war, ist das Resultat plausibel, obwohl es etwas hoch ist. Das recht breite Maximum der Leistungsdichte bei 34 Hz lässt darauf schliessen, dass die Drehzahl leichten Schwankungen unterworfen war.

Die Autokorrelation zeigt deutlich, dass kaum stochastische Anteile im Signal vorhanden sind, da die Maxima alle etwa gleich hoch sind. Im Weiteren ist ersichtlich, dass dieses Signal fast sinusförmig ist.

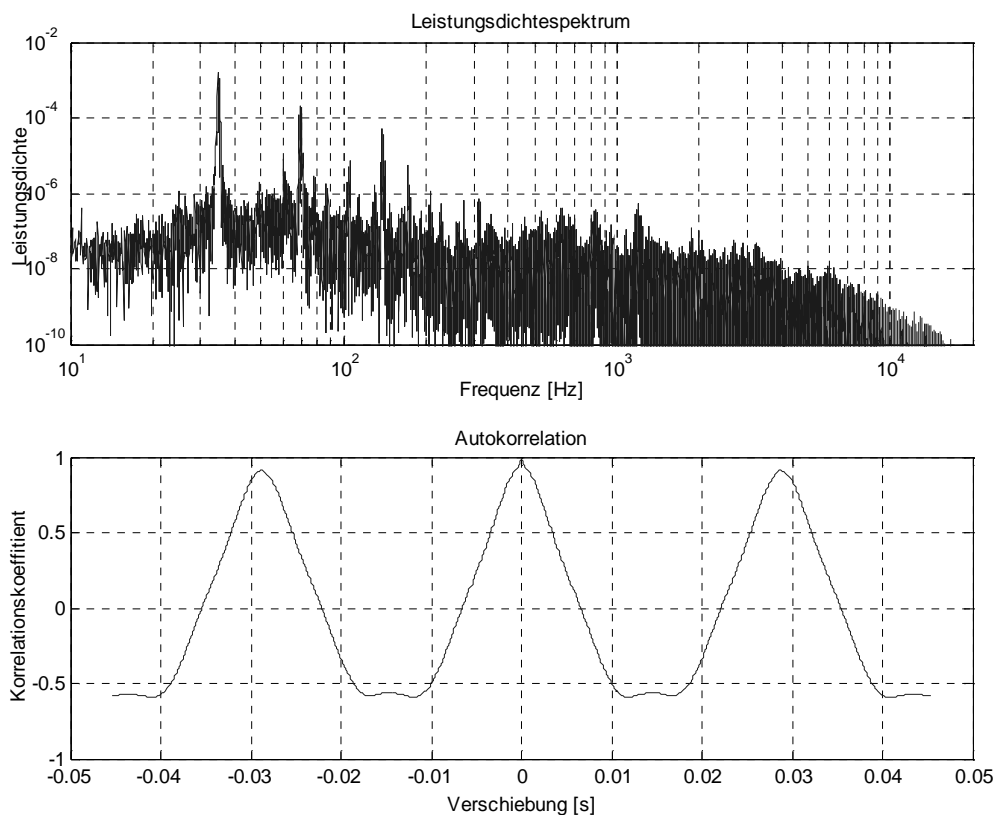


ABBILDUNG 3. Analyse eines Personenwagens

4.1.2 Mercedes Lieferwagen Dieselmotor

Dieser kleine Dieselmotor mit 4 Zylindern wurde ausgewählt, damit ein Vergleich mit grossen Dieselmotoren möglich ist.

Das Leistungsdichtespektrum zeigt eine kleine Abnormalität. Für die Berechnung der Umdrehungszahl darf keine der stärksten Komponenten bei 60 Hz, 120 Hz und 168 Hz verwendet werden, sondern die Schwächere bei 22 Hz.

$$u = \frac{60 \cdot f_0}{n \cdot z} = \frac{60 \cdot 22 \text{ Hz}}{0.5 \cdot 4} = 660 \frac{1}{\text{min}}$$

Mit Hilfe der Autokorrelation lässt sich nochmals bestätigen, dass sich dieses Signal nicht vollständig mit den Erwartungen deckt. Die starken Oberwellen sind deutlich erkennbar. Im Weiteren produziert dieser Motor einen gut erkennbaren, stochastischen Signalanteil. Auf Grund der schmalen Maxima ist ersichtlich, dass die Drehzahl während der Aufnahme konstant war.

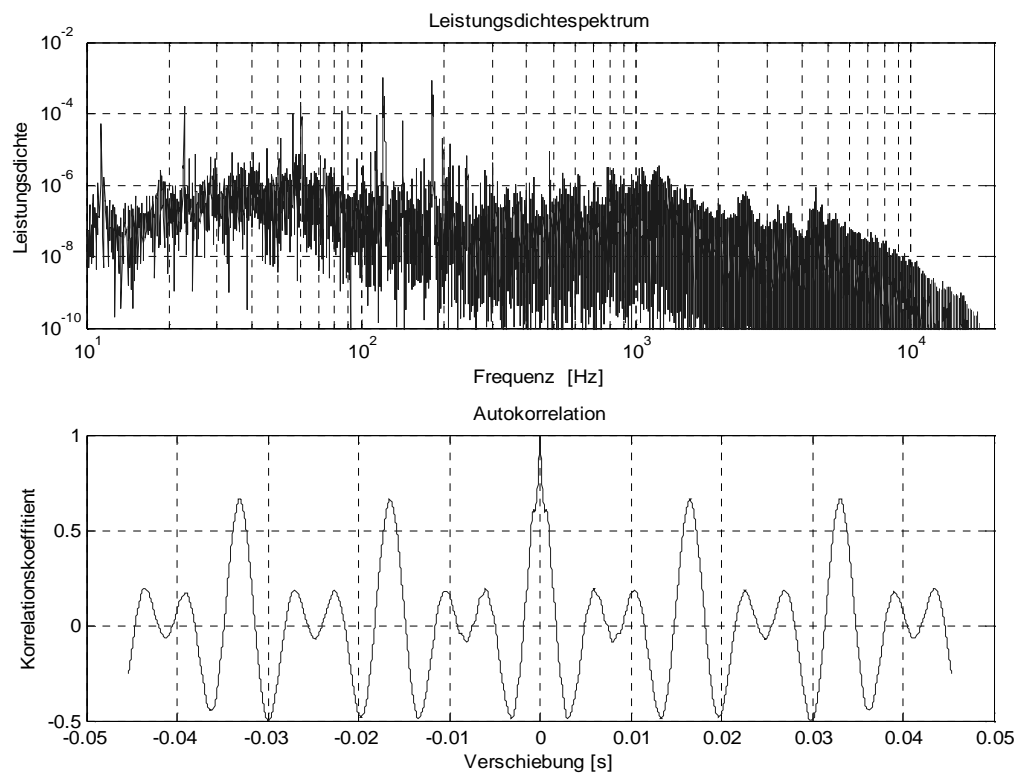


ABBILDUNG 4. Analyse eines kleinen Lieferwagens

4.1.3 Volvo Lastkraftwagen Dieselmotor

Dies ist ein grosser Dieselmotor neueren Baujahres mit 8 Zylindern. Er sollte von den bisher beschriebenen Signalen die grössten Probleme bei der Störgeräuschunterdrückung verursachen.

Die Analyse des Leistungsdichtespektrums zeigt, wie bereits der Personenwagen, drei starke Frequenzanteile. Sie befinden sich aber bei 30 Hz, 60 Hz und 90 Hz. Dieses Fahrzeug hat also eine relativ tiefe Leerlaufdrehzahl, was sehr wahrscheinlich für das bekannte, typische Dieselmotorgeräusch, das „Nageln“, mitverantwortlich ist.

$$u = \frac{60 \cdot f_0}{n \cdot z} = \frac{60 \cdot 30\text{Hz}}{0.5 \cdot 8} = 450 \frac{1}{\text{min}}$$

Die Autokorrelation zeigt einen deutlichen stochastischen Signalanteil. Anhand des Maximums der Leistungsdichte lässt sich erkennen, dass die Drehzahl dieses Motors nahezu konstant war.

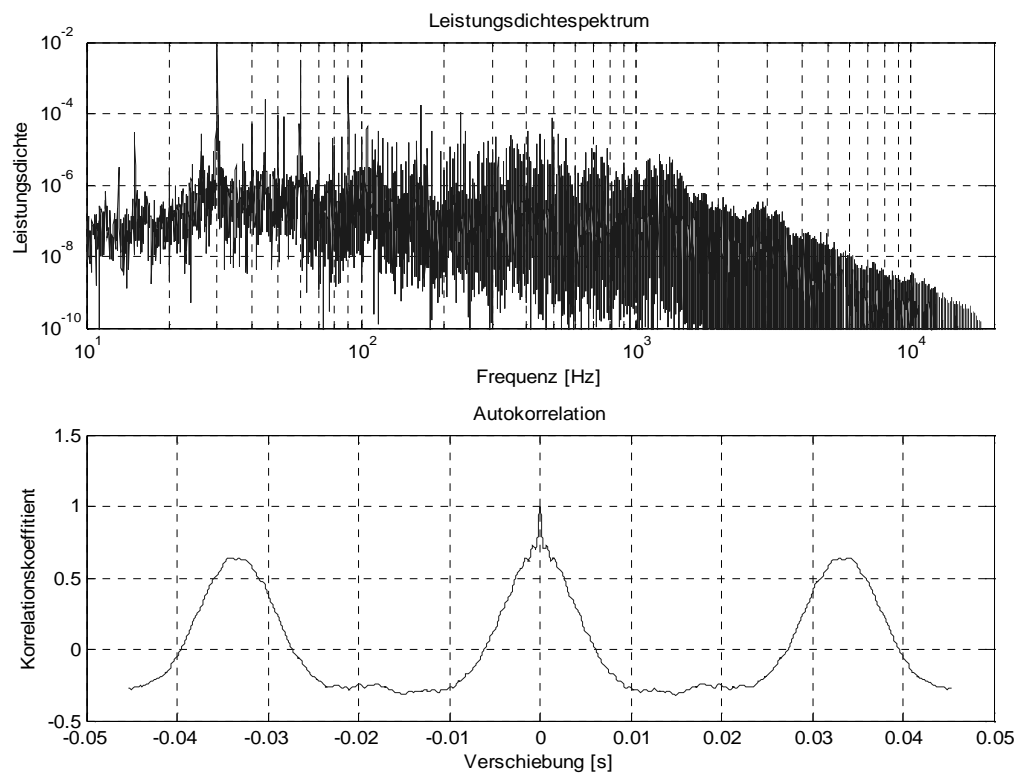


ABBILDUNG 5. Analyse eines Lastkraftwagens

4.2 Sprechen und Hören

Dieses Kapitel beschreibt die Charakteristik der menschlichen Sprache und des menschlichen Gehörs, soweit sie für diese Arbeit relevant sind. Dabei wird auch auf die Konsequenzen für die Lärmunterdrückung eingegangen.

4.2.1 Sprachanalyse

Das hier besprochene Sprachbeispiel stammt von einem männlichen Deutschschweizer.

Das Leistungsdichtespektrum zeigt, dass dieses Sprachsignal hauptsächlich Anteile im Bereich 80 Hz bis 400 Hz hat. Der Bereich von 700 Hz bis 6 kHz ist hingegen relativ schwach ausgebildet. Das Ohr ist dort jedoch besonders empfindlich.

Die Autokorrelation zeigt deutlich, dass das Sprachsignal nur in einem kleinen Bereich von ca. 15 ms korreliert. Bei weiterer Verschiebung zeigt sich nur noch eine unbedeutende Korrelation. Diese Eigenschaft ist vorallem für adaptive Lärmunterdrückungsverfahren mit nur einem Mikrofon relevant, da die Sprache und die Störung nur anhand dieses Kriteriums unterschieden werden können.

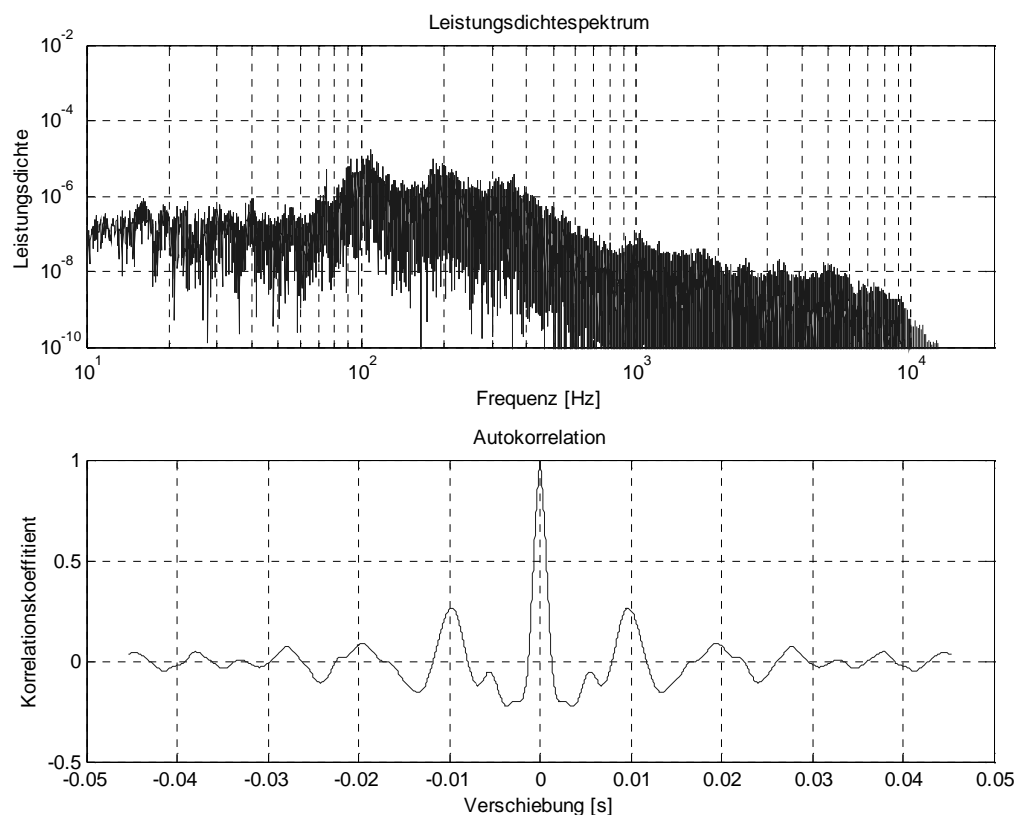


ABBILDUNG 6. Analyse der menschlichen Sprache

4.2.2 Gehör

Das menschliche Gehör zeigt abhängig von der Frequenz und dem Schalldruck starke Empfindlichkeitsschwankungen. Als für die Sprachverständlichkeit wichtigen Frequenzbereich wird allgemein der Bereich zwischen 300 Hz bis 3 kHz angesehen. Dies lässt sich anhand der unten abgebildeten Empfindlichkeitskurven (siehe Abbildung 7) erklären. Das Gehör ist zwischen 0 Hz und 100 Hz sehr unempfindlich. Mit zunehmendem Abstand zur Hörschwelle wird die Empfindlichkeit zwischen 300 Hz und 3 kHz annähernd linear.

Dies hat Konsequenzen für die Lärmunterdrückung, denn tieffrequente Lärmbestandteile können vernachlässigt werden, solange der Gesamtpegel nicht extrem hoch ist. Die höherfrequenten Anteile, ab etwa 200 Hz, bestimmen jedoch in viel höherem Ausmass das Lärmempfinden, da das Ohr in diesem Bereich empfindlicher ist. Dieser Umstand muss bei der Beurteilung der weiter oben besprochenen Leistungsdichtespektren berücksichtigt werden. Da diese aber nur mit grossem Aufwand dem menschlichen Ohr entsprechend gewichtet werden können, muss davon ausgegangen werden, dass nur der Bereich ab ca. 200 Hz für die Lärmunterdrückung relevant ist. Die obere Grenze wird durch die zur Verfügung stehende Bandbreite bestimmt. Diese wird in einem üblichen sprachverarbeitenden System wohl nicht höher als ca. 8 kHz liegen.

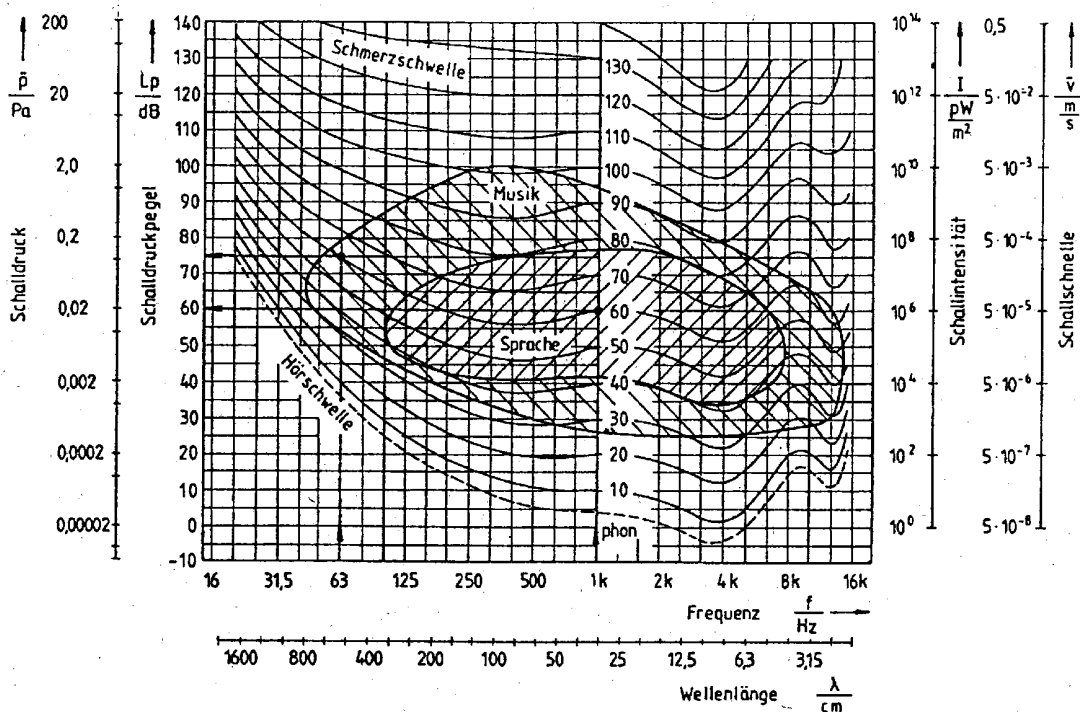


ABBILDUNG 7. Normalkurven gleicher Lautstärkepegel und Hörschwelle für Sinustöne im freien Schallfeld bei binauralem Hören nach Robinson und Dadson (1959).

5.0 Ansätze zur Lärmunterdrückung

Es gibt verschiedene Ansätze, ein Geräusch in einem Nutzsignal zu unterdrücken und es wurden auch schon diverse Semesterarbeiten und Literatur zu diesem Thema geschrieben. Deshalb wurden die vorhandenen Systeme der Reihe nach in Betrieb genommen und eingehend mit Motorengeräuschen und Sprache getestet.

Zusätzlich zu den bestehenden Algorithmen sind stets auch einfache Möglichkeiten gesucht worden, um das Problem zu lösen. Unter anderem sind das eine einfache Verschiebung im Zeitbereich oder einfache Auslöschungen im Frequenzbereich. Auf die einzelnen Verfahren wird im Folgenden ausführlich eingegangen.

5.1 Verschiebung

In den Signalspektren von Motorgeräuschen kann man einen direkten Zusammenhang zur Drehzahl erkennen. Wenn man davon ausgeht, dass das Störgeräusch hauptsächlich aus der Motordrehzahl und deren Vielfachen besteht, kann man ein System konstruieren, das genau diese Frequenzen herausfiltert. Das Erkennen der Funktionsweise ist im Zeitbereich und anhand eines Blockschaltbildes am einfachsten.

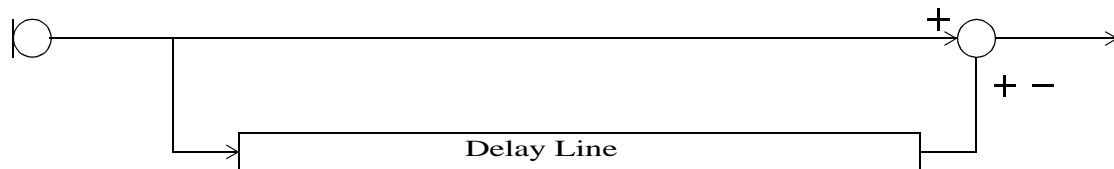


ABBILDUNG 8. Blockschaltbild des Verschiebungs-Verfahrens

Das Eingangssignal wird so weit verzögert, bis es sich wieder mit sich selbst deckt. D.h. wenn das Signal halbperiodisch verschoben wird, wird es addiert. Bei einer Verschiebung um eine ganze Periode wird es subtrahiert. Die nötige Verzögerungszeit lässt sich z.B. mit Hilfe der Autokorrelation bestimmen. Mit einem DSP ist ein solches System mit sehr wenig Rechenaufwand implementierbar. Anhand der Fouriertransformierten der Übertragungsfunktion lässt sich erkennen, dass dieses System eine Grundfrequenz und alle Vielfachen davon vollständig unterdrückt.

$$h(t) = \delta(t) \pm \delta(t - \tau)$$

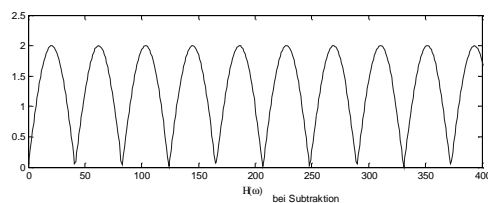
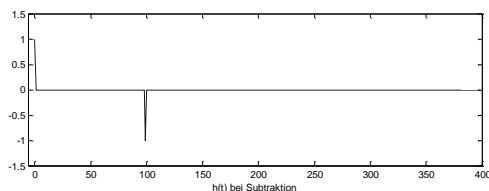
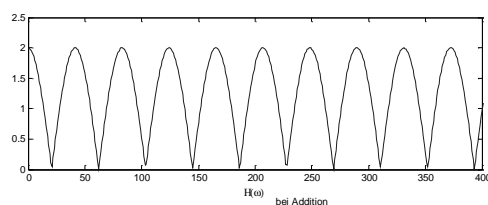
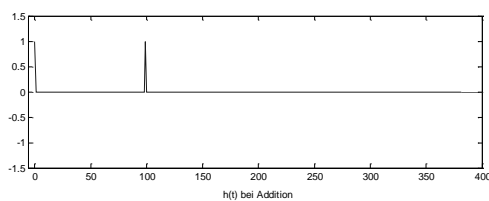


$$H(s) = (1 \pm e^{-s\tau})$$

$$H(s) = e^{-\frac{s\tau}{2}} \left(e^{\frac{s\tau}{2}} \pm e^{-\frac{s\tau}{2}} \right)$$

$$H(j\omega) = e^{-\frac{j\omega\tau}{2}} \left(e^{\frac{j\omega\tau}{2}} \pm e^{-\frac{j\omega\tau}{2}} \right)$$

$$|H(j\omega)| = \left| 2 \cos\left(\frac{\omega\tau}{2}\right) \right| \quad \text{bzw.} \quad |H(j\omega)| = \left| 2 \sin\left(\frac{\omega\tau}{2}\right) \right|$$



Die Anwendung dieses Verfahrens auf ein reales Motorengeräusch ist allerdings wenig befriedigend. Dies aus folgenden zwei Gründen:

- Die höherfrequenten, stochastischen Anteile im Signal sind viel störender als die tieffrequenten, periodischen Signalanteile.
- Es ist schwierig, die Motorperiode ganz exakt zu messen. Es bleiben also immer Reste davon im Signal.

Diese einfache Verschiebung des Signals reduziert die Amplitude der Störung markant. Das menschliche Ohr erkennt allerdings nur eine Klangveränderung des Ausgangssignals von einem „Nageln“ hin zu einem „Surren“, welches allerdings subjektiv nicht als angenehmer wahrgenommen wird. Ein weiteres Problem zeigt sich bei der Anwendung dieses Systems auf Sprache. Da Motorengeräusche sehr tieffrequent sind, muss auch die Verzögerungszeit sehr gross gewählt werden. Dies führt zu einem nicht mehr akzeptablen Echo des Sprachsignals.

Die beschriebenen Probleme machen das Verschiebungsverfahren für die Motorenlärmunterdrückung untauglich. Es kann aber als Erklärungsmodell für die Halleffekte des weiter unten beschriebenen, komplexeren Mehrkanalsystems dienen.

5.2 Beamforming

Eine weitere Möglichkeit, einen Sprecher in einer lärmigen Umgebung besser zu verstehen, ist das sog. Beamforming. Grundlage dieser Lösung ist eine Richtwirkung mindestens zweier Mikrofone, welche in kleinem Abstand d zueinander platziert werden müssen. Durch geschicktes Verrechnen der Mikrofonsignale unter Einsatz von Verzögerungselementen (siehe Abbildung 9) kann eine Richtcharakteristik erreicht werden. Die Richtkeule wird auf den Sprecher gerichtet und bewirkt eine Unterdrückung der sich daneben befindenden Störquellen.

Ein solches System ist aber nur erfolgsversprechend, wenn sich der Sprecher räumlich klar von der Störquelle trennen lässt. Dies ist unter Umständen schlicht nicht gegeben, wenn z.B. die Kabine des Fahrzeugs mit dem Motor mitschwingt oder sonstige Carosserieteile scheppern.

Der Testaufbau hat gezeigt, dass eine überzeugende Leistung dieses Systems mit nur zwei Mikrofonen nicht erzielbar ist. Mehr als zwei Mikrofone können aber wegen des grossen Hardwareaufwandes nur in Ausnahmefällen in der endgültigen Implementation verwendet werden. Hinzu kommt, dass wohl mit einem guten Richtmikrofon ein ähnliches Resultat erzielt werden könnte.

Auf Grund dieser eher schlechten Voraussetzungen schien es nicht erfolgsversprechend, diesen Ansatz weiter zu verfolgen.

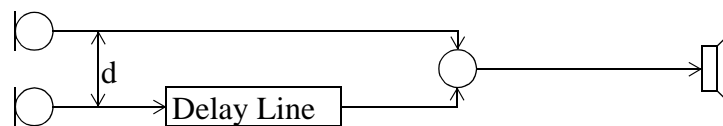


ABBILDUNG 9. Blockschaltbild eines einfachen Beamforming

5.3 Adaptive Verfahren

Das adaptive Verfahren hat den Vorteil, dass sich das System selbstständig an die immer wechselnden Einsatzumgebungsbedingungen anpasst [7]. Diese Fähigkeit wird durch einen rekursiven Systemaufbau erreicht, der die Parameter fortlaufend optimiert. Es wurde in der Literatur schon oft erwähnt und ist deshalb auch theoretisch gut untersucht worden.

5.3.1 Zwei Mikrofone

Da zwei Mikrofone benötigt werden, hat dies sicher einen nachteiligen Mehraufwand an Hardware zur Folge. Im Weiteren müssen die Signale so gut wie möglich akustisch von einander getrennt sein. Kann dies nicht eingehalten werden (d.h. das Mikrofon1 nimmt auch Sprache auf), so äussert sich dies in einer Verzerrung des Sprachsignals. Bezüglich der Geräuschunterdrückung kann ein gutes Resultat erzielt werden.

Das Funktionsprinzip ist relativ einfach. Das Mikrofon1 ist voll auf die Störquelle gerichtet, d.h. in diesem Anwendungsfall auf den Motor und sollte keine Sprache mitaufzeichnen. Das Mikrofon2 wird auf den Sprecher gerichtet. Dabei ist es kaum zu vermeiden, dass bei diesem Mikrofon auch das Störgeräusch mitaufgezeichnet wird - dies ist ja auch die Ausgangslage dieser Semesterarbeit.

Das Störsignal von Mikrofon1 läuft durch einen FIR(Transversal)-Filter mit N+1 Koeffizienten und wird vom gestörten Sprachsignal (Mikrofon2) subtrahiert. Daraus resultiert ein Fehlersignal, welches zugleich auch das Ausgangssignal darstellt. Dieses Signal besteht theoretisch und vereinfacht aus:

$$\text{Mikrofon2} - \text{Mikrofon1} = \text{Ausgangssignal}$$

bzw.

$$\text{Sprache} + \text{Motorengeräusch} - \text{Motorengeräusch} = \text{Sprache}$$

Das Ausgangssignal wird nun durch den normierten Least-Mean-Square (LMS) Gradientenalgorithmus [7] verändert und dieser passt die Koeffizienten des FIR-Filters an. Je weniger der LMS-Algorithmus die Koeffizienten nachkorrigieren muss, desto besser ist die Ähnlichkeit des Ausgangssignals mit dem reinen Sprachsignal.

Es können zwei Parameterwerte eingestellt werden, die direkten Einfluss auf das adaptive System haben. Es sind dies die Korrekturschrittweite des LMS-Algorithmus und die Anzahl der Filterkoeffizienten. Wird letzteres klein gehalten, braucht das Verfahren wenig Rechenleistung des DSPs, aber das erzielte Resultat ist nicht sehr befriedigend. Erhöht man die Anzahl der Koeffizienten, verhält sich das System entsprechend besser, d.h. die Qualität des Ausgangssignals steigt an. Die Korrekturschrittweite wird benötigt, um den minimalen Fehler beim LMS-Algorithmus je nach dem schneller und ungenauer oder langsamer und dafür präziser zu erreichen. Der LMS-Algorithmus ist eigentlich eine Vereinfachung des MSE (Mean-Square-Error) Verfahrens [7]. Bei der MSE-Variante gilt es, ein Gleichungssystem zu lösen, um den minimalen (quadratischen) Fehler zu erhalten. Beim LMS-Gradientenalgorithmus wird die Lösung iterativ gesucht. Dabei hat die Korrekturschrittweite einen wesentlichen Einfluss auf die Geschwindigkeit dieses Verfahrens.

In der Abbildung 10 sieht man einen Querschnitt der quadratischen Fehlerfläche Q entlang eines Koeffizienten c_n . Diese Kurve entsteht, wenn die quadratische Gleichung des Koeffizienten c_n bezüglich des MSEs aufgestellt wird. Die Korrekturschrittweite α hat nun, wie aus den Formeln ersichtlich, Einfluss auf die Korrektur Δc_n .

Wird die Korrekturschrittweite gross gewählt, passt sich der Algorithmus schnell an, mit dem Nachteil einer Qualitätseinbusse. Das System kann in diesem Fall das Minimum nur per Zufall exakt anfahren; es kann sogar instabil werden. Umgekehrt wird die Güte besser, wenn die Schrittweite klein gehalten wird. Dies hat den negativen Einfluss auf die Anpassungsgeschwindigkeit, denn es müssen mehr Korrekturen berechnet werden, um in die Nähe des Minimums zu gelangen.

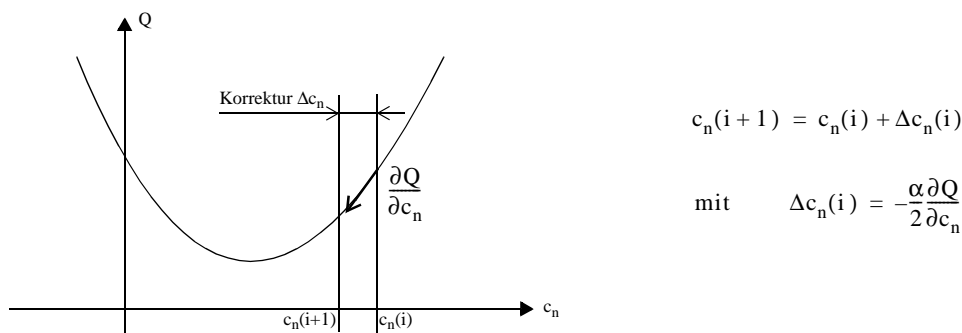


ABBILDUNG 10. Korrektur des Koeffizienten c_n in Richtung Minimum

Bei Versuchen hat sich gezeigt, dass die Anzahl Koeffizienten mindestens 200 aber besser über 1000 betragen sollte, um ein gutes Resultat zu erhalten. Das System selber erzeugt aber ein starkes Rauschen, das sich klar aus den gegebenen Signalen hervorhebt. Die Ursache ist unklar. Es könnte sich um Aliasing-Probleme handeln. Ein ebenfalls auftretendes Pfeifen und Surren stammt wahrscheinlich von der Abtastfrequenz und der parallel laufenden Matlab m-Datei, die der Kontrolle des DSPs dient. Die so erzeugten PCI-Zugriffe auf die DSP-Karte könnten den empfindlichen Analogteil stören.

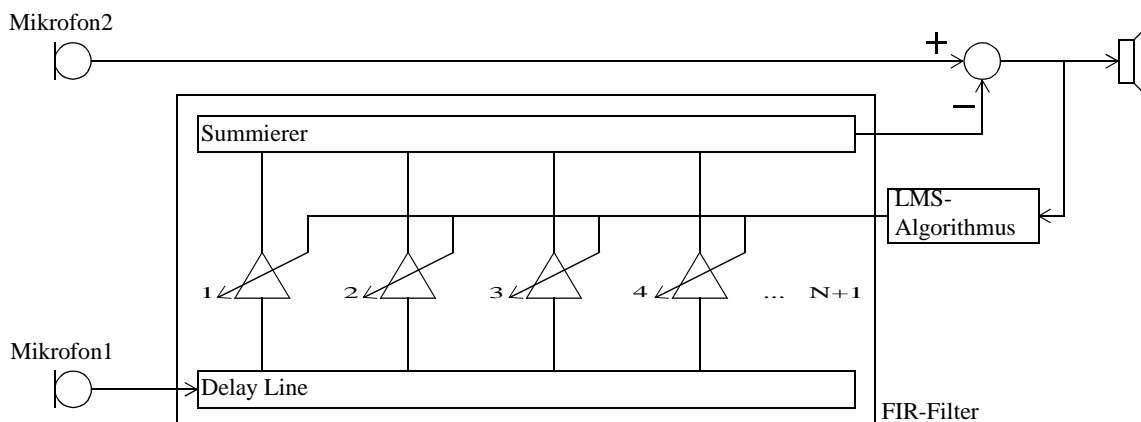


ABBILDUNG 11. Blockschaltbild des adaptiven Systems mit zwei Mikrofonen

5.3.2 Ein Mikrofon

Der Nachteil, dass zwei Mikrofone benötigt werden, kann umgangen werden, wenn das Signal des zweiten Mikrofons vom ersten Signal abgeleitet wird. Dabei wird das gestörte Sprachsignal verzögert und danach dem adaptiven System eingespielt. Die Verzögerungszeit muss so gewählt werden, dass das Sprachsignal nicht mehr korreliert [12]. Auf die Frequenz des Motorengeräusches muss keine Rücksicht genommen werden, da dessen Autokorrelation periodisch ist.

Ansonsten ist die Funktionsweise der des oben beschriebenen adaptiven Systems gleich.

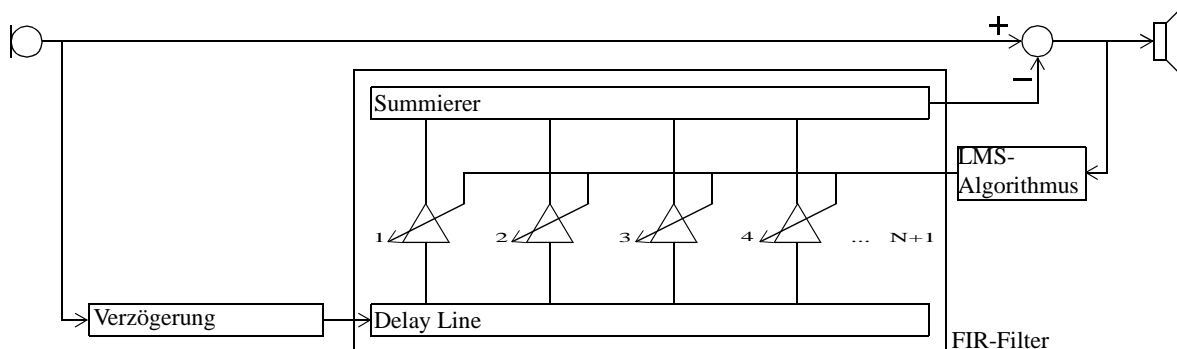


ABBILDUNG 12. Blockschaltbild des adaptiven Systems mit einem Mikrofon

Dieses adaptive System mit vorgeschalteter Verzögerung lässt sich auch aus einem anderen Blickwinkel betrachten. Es entspricht bis auf die grosse Verzögerung einem Prädiktor, bzw. einem Dekorrelator. Dieser hat die Fähigkeit, periodische Signalanteile herauszufiltern und am Ausgang, ein unkorreliertes Signal zu liefern.

Wenn nun dafür gesorgt wird, dass die Verzögerung vor dem Transversalfilter so gross ist, dass die Sprache nicht mehr mit sich selbst korreliert, kann dieses System Sprache ungehindert übertragen, während periodische Störungen unterdrückt werden.

Die Hörtests mit dieser Anordnung haben ergeben, dass periodische Störungen wie Sinus-, Rechteck- und Dreieckssignale unabhängig von der Frequenz restlos eliminiert werden. Die Tests mit realem Motorenlärm waren aber enttäuschend. Dies lässt sich einfach durch die vorhandenen stochastischen Signalanteile im Motorenlärm erklären, denn diese können das System ungehindert passieren. Dieser Ansatz ist deshalb für die Motorenlärmunterdrückung nicht geeignet.

5.4 Mehrkanalverfahren

Das Mehrkanalverfahren war bereits Bestandteil einer Diplomarbeit [1] und ist dort umfassend beschrieben worden. An dieser Stelle soll nur eine kurze Beschreibung, soweit sie für das Verständnis erforderlich ist, und eine Beurteilung in Hinsicht auf die Unterdrückung von Motorenlärm stehen.

Das Mehrkanalverfahren nutzt die hohe Dynamik der menschlichen Sprache aus, um ein Lärmunterdrückungsverfahren zu realisieren. Wie das weiter unten stehende Blockschaltbild (Abbildung 13) erkennen lässt, arbeitet der Algorithmus im Frequenzbereich. Das Sprachsignal wird zuerst fouriertransformiert und dann in einzelne Frequenzbänder zerlegt, welche theoretisch beliebige Grössen haben können. Diese werden dann auf ihre Dynamik hin untersucht, indem jeweils die maximale und minimale Gesamtleistung, die in einem Band über mehrere Messungen hinweg vorgekommen ist, verglichen wird. Dieser Dynamikmesswert wird verwendet, um den Bändern unterschiedliche Dämpfungsfaktoren zuzuordnen. Vor dem Rücktransformieren in den Zeitbereich werden die Bänder wieder zusammengefügt.

Das System erlaubt folgende Modifikationen, wobei die ersten vier im Betrieb verändert werden können:

- Maximale Dämpfung
- Maximale Dynamik
- Absinkfaktor
- Anstiegsfaktor
- Breite, Verteilung und Anzahl der Bänder

Die zwei Erstgenannten haben Einfluss auf die Wirksamkeit der Störunterdrückung und die folgenden zwei beeinflussen die Adaptionsgeschwindigkeit.

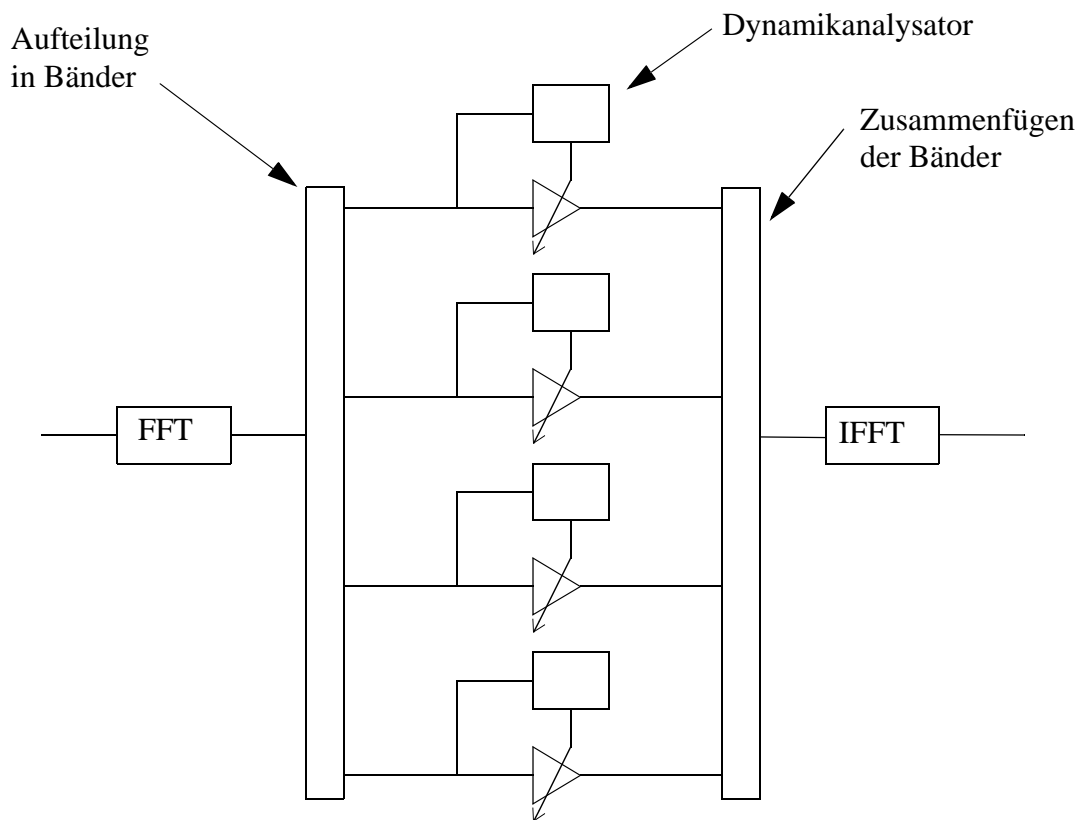


ABBILDUNG 13. Blockschaltbild des Mehrkanalsystems

Dieses Verfahren führt zu einem prinzipiellen Frequenzgang, welcher etwa wie in Abbildung 14 aussehen könnte und abhängig vom Eingangssignal ständig ändert.

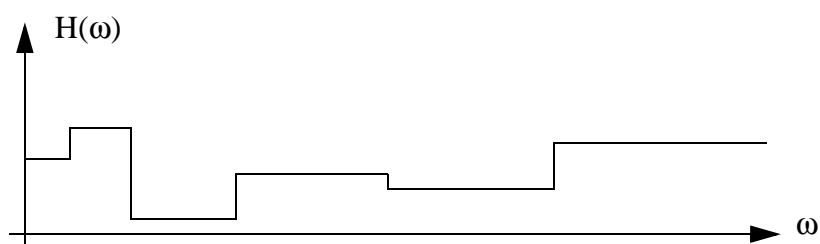


ABBILDUNG 14. Möglicher Frequenzgang des Mehrkanalsystems

Die Autoren der Diplomarbeit [1] stellten einen nicht erklärbaren Halleffekt fest, der zudem von der beigemischten Störung abhängig war. Dieser Effekt kann folgendermassen erklärt und hergeleitet werden.

Beim einfachsten Fall einer Störung mit vielen starken Oberwellen sieht der idealisierte Frequenzgang des Mehrkanalverfahrens wie folgt aus.

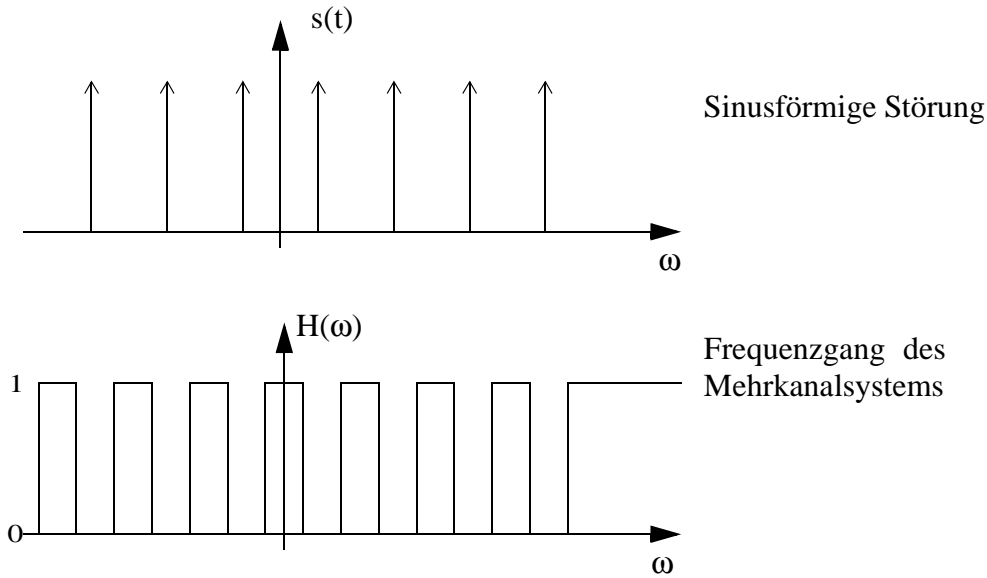


ABBILDUNG 15. Ein einfacher Fall einer Störung

$H(\omega)$ lässt sich in die Stossantwort $h(t)$ transformieren:

$$g(t) = \frac{\omega_g}{\pi} \cdot \frac{\sin(\omega_g t)}{\omega_g t}$$

Da der Frequenzgang aber periodisch ist, muss die Stossantwort des Systems diskret sein:

$$h(t) = \frac{\omega_g}{\pi} \cdot \frac{\sin(\omega_g t)}{\omega_g t} \cdot \sum_{n=-\infty}^{\infty} \delta\left(t - \frac{n\pi}{2\omega_g}\right)$$

$$h(n) = \frac{\omega_g}{\pi} \cdot \frac{\sin\left(\frac{n\pi}{2}\right)}{\frac{n\pi}{2}}$$

Dieses Resultat lässt sich auch mit Abbildung 16 grafisch verdeutlichen:

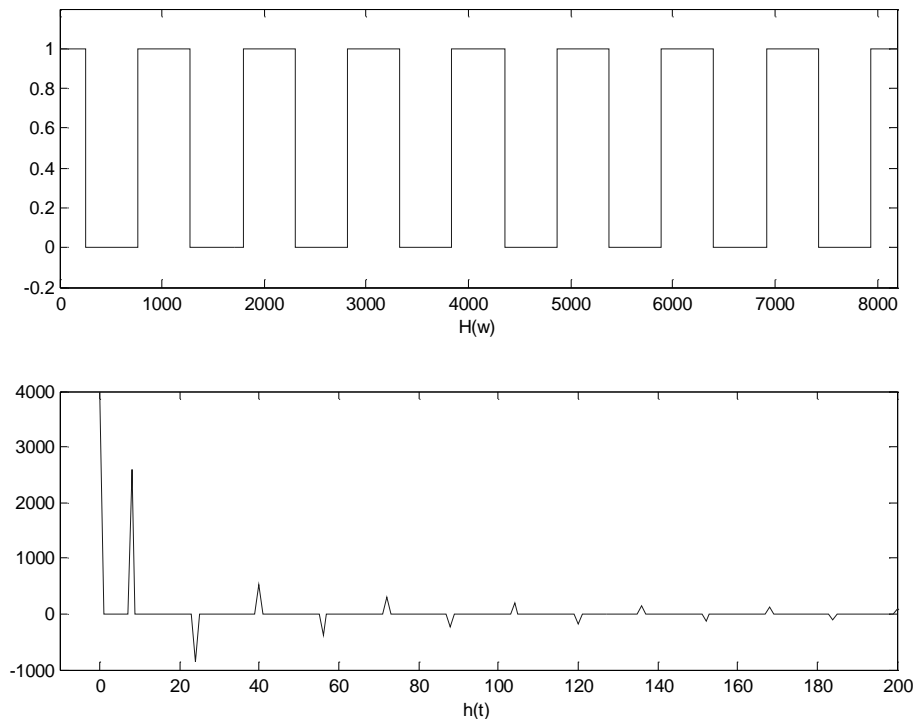


ABBILDUNG 16. oben: Frequenzgang des Mehrkanalsystems
 unten: Resultierende Stossantwort des Mehrkanalsystems

Anhand der idealisierten Stossantwort $h(t)$ des Mehrkanalsystems bei anliegender oberwellenreicher Störung erkennt man, dass dem Nutzsignal zwangsläufig eine Art Hall hinzugefügt wird, da $h(t)$ aus einer abklingenden Folge von Dirac-Stößen besteht. Diese Gesetzmässigkeit hat sich bereits beim Verschiebverfahren gezeigt. Ein Kammfilter führt unabhängig von der exakten Kurvenform, also immer, zu einem Hall bzw. zu einem Echo. Die Klangcharakteristik dieses Halls ist von der Art der Störung und von den aktuellen Parametern des Mehrkanalsystems abhängig.

Aus diesen Überlegungen ergibt sich, dass das Hallproblem etwas entschärft werden kann, wenn verhindert wird, dass immer gleich breite Frequenzbänder bedämpft werden. Dies lässt sich am einfachsten erreichen, indem das ganze Frequenzspektrum in verschieden breite Frequenzbänder aufgeteilt wird.

Das Mehrkanalverfahren eignet sich trotzdem sehr gut, um beliebige stationäre Störungen zu unterdrücken. Um aber optimale Ergebnisse hinsichtlich der Geräuschunterdrückung und des Hallverhaltens zu erreichen, sind umfangreiche Hörversuche mit der gegebenen Störquelle nötig, damit alle Parameter korrekt eingestellt werden können. Das Mehrkanalsystem arbeitet umso besser, je lauter und deutlicher gesprochen wird.

6.0 Realisation

6.1 Aufbau

Der physikalische Aufbau ist schon im Überblick beschrieben, sodass hier nur noch auf den Aufbau der DSP-Software eingegangen wird.

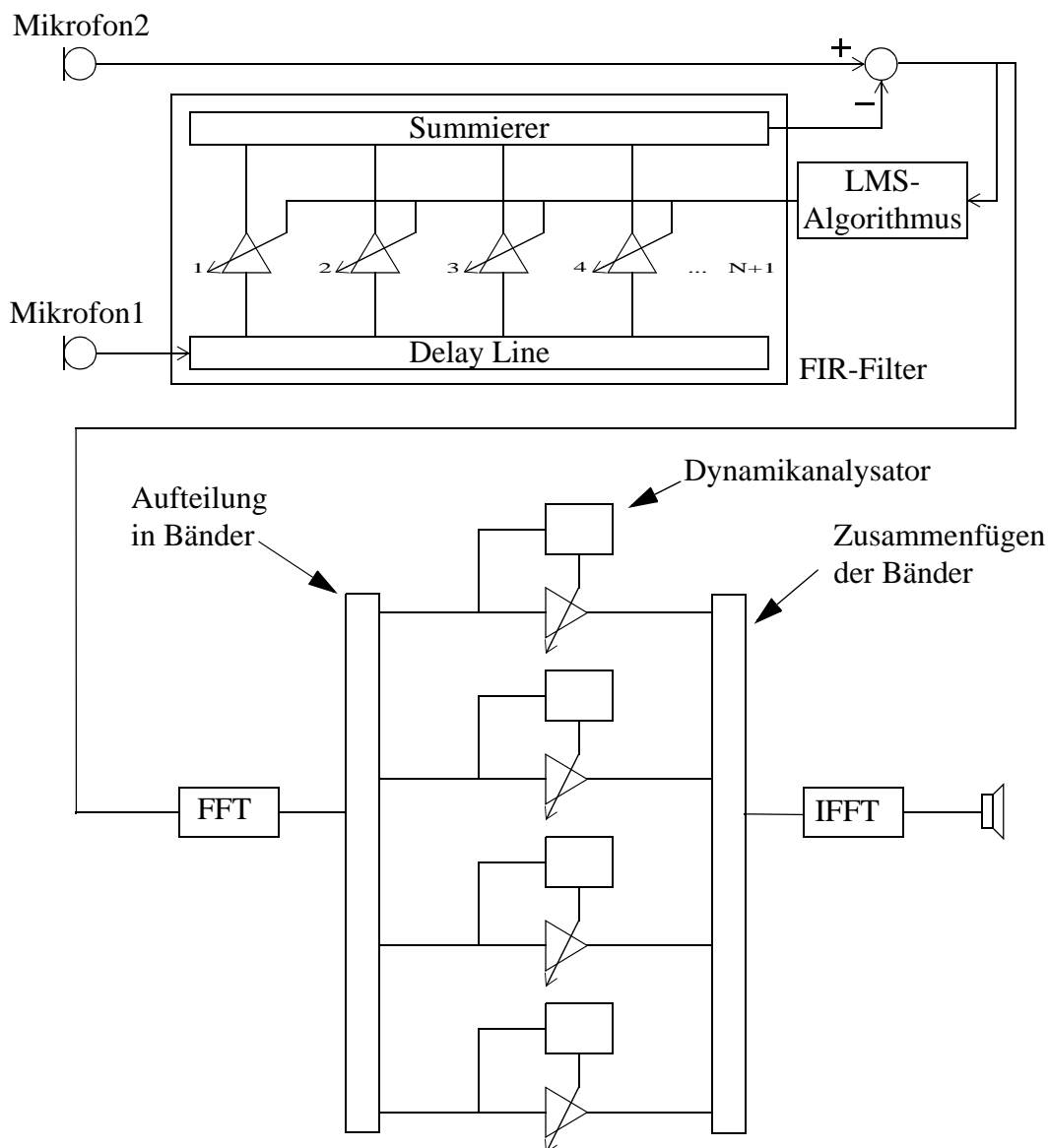


ABBILDUNG 17. Blockschaltbild des Gesamtsystems

Da das Mehrkanalverfahren und das adaptive System bei Versuchen mit Motorenlärm die besten Lärmunterdrückungsergebnisse erbrachten, liegt der Gedanke nahe, diese beiden Systeme zu kombinieren. Das Funktionsprinzip dieser beiden Varianten wurde bereits weiter oben erklärt. An dieser Stelle soll nur noch auf Aspekte eingegangen werden, die für die endgültige Realisation relevant sind. Da das adaptive System zwei Eingänge besitzt,

wurde es mit den beiden Mikrofonen verbunden. Zur Verbesserung des Resultats kann dann das Mehrkanalsystem nachgeschaltet werden

Die beiden Systeme ergänzen sich optimal. Das adaptive System eliminiert tieffrequente Störungen fast perfekt. Die stochastischen, höher frequenten Anteile der Störung können dann problemlos durch das Mehrkanalsystem unterdrückt werden.

Zusätzlich erreicht man, dass das System in den Sprechpausen praktisch stummgeschaltet wird, weil der stochastische Teil des Motorenlärms kaum Dynamik aufweist, dafür aber über den ganzen Frequenzbereich verteilt ist. Dies führt dazu, dass das Mehrkanalsystem alle Bänder stark abschwächt.

Das System arbeitet mit einer Abtastfrequenz von 22050 Hz, damit bei grösseren Stückzahlen auch billigere, im PC-Bereich übliche Standardkomponenten, z.B. DA-AD-Wandler zum Einsatz kommen könnten, ohne dass das Gesamtverhalten gravierend ändern würde.

Der Rechenleistungsbedarf des Mehrkanalsystems ist unabhängig von den einstellbaren Parametern, da die Anzahl der Bänder nur durch Änderung des Quellcodes und durch nachträgliche Neukompilation geändert werden kann. Im realisierten System arbeitet das Mehrkanalsystem mit 30 Bändern. Eine weitere Erhöhung der Bänderanzahl scheint nur sinnvoll, wenn gleichzeitig auch die Länge der FFT von momentan 512 Punkten auf 1024 Punkte erhöht würde, damit auch die tatsächlich realisierbare Frequenzauflösung entsprechend gut ist. Dafür reicht aber die zur Verfügung stehende Rechenleistung nicht aus. Das adaptive System benötigt für gute Ergebnisse vergleichsweise viel Rechenleistung. Allerdings lässt sich diese einfach durch Veränderung der Filterlänge den Bedürfnissen anpassen.

Die Schrittweite, bzw. die Anpassungsgeschwindigkeit sind wegen des normierten LMS Algorithmus unproblematisch, solange das System stabil bleibt. Wie schon bei der Beschreibung der Signale gezeigt wurde, sind die Störungen annähernd stationär. Die Motordrehzahlen sind nahezu konstant und vom Heranfahen des Fahrzeugs zur Gegensprechanlage bis zum Beginn des Sprechens vergeht sicher genügend Zeit, damit sich das adaptive System auf die Störung einstellen kann. Die Schrittweite kann sehr klein gewählt werden.

Die optimalen Parametereinstellungen lauten wie folgt:

Maximale Dämpfung: 40 dB
Maximale Dynamik: 50 dB
Absinkfaktor: 0.97
Anstiegsfaktor: 1.03

Die zur verfügbare Rechenleistung reicht beim adaptiven System für 400 Koeffizienten aus.

6.2 Bändereinteilung des Mehrkanalsystems

Das Mehrkanalsystem wurde mit 30 Bändern realisiert. Die Einteilung geschah unter drei Aspekten. Zum einen muss die Hörcharakteristik des menschlichen Ohres berücksichtigt werden. Dies führt zu einer Einteilung mit kleinen Abständen im tieffrequenten Bereich und grossen Abständen im hochfrequenten Bereich. Zum anderen sollten die Bänder nicht alle die gleiche Breite haben, um Halleffekte so gut als möglich zu verhindern. Im Weiteren muss berücksichtigt werden, dass die Frequenzauflösung von der FFT-Länge abhängt. Es müssen immer zwei Spektrallinien innerhalb eines Bandes zu liegen kommen. Diese drei Bedingungen können mit folgenden Spezifikationen erfüllt werden:

- Es werden 30 Bänder realisiert
- 512-Punkte-FFT ergibt einen minimalen Bandabstand von 86.1 Hz
- Band 1 beginnt bei 0 Hz
- Band 10 reicht bis 1000 Hz
- Band 20 reicht bis 3000 Hz
- Band 30 reicht bis 11025 Hz

Durch diese Punkte kann ein Polynom gelegt werden.

$$f = ax^4 + bx^3 + cx^2 + dx^1 + e$$

a = 0.0218214
 b = -0.471786
 c = 3,878580
 d = 86,57140
 e = 0.000000

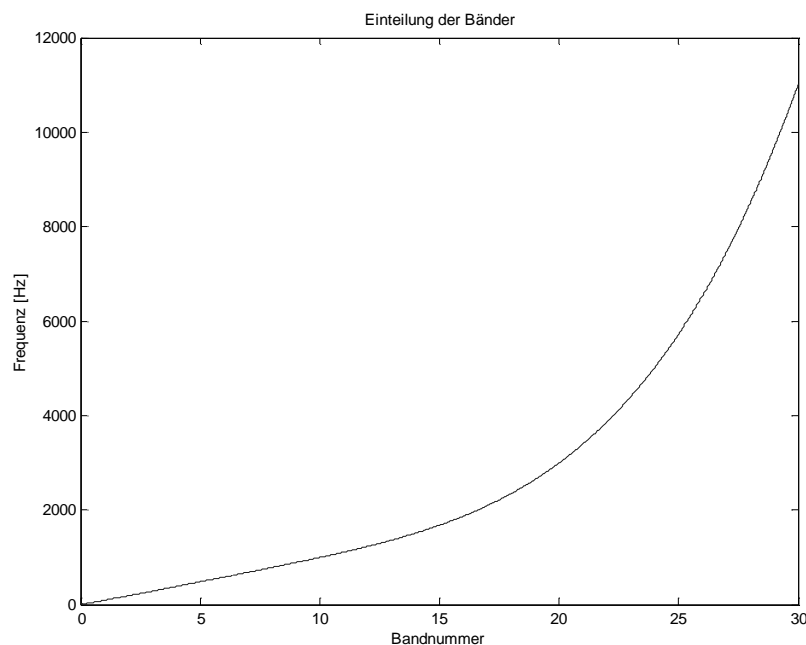


ABBILDUNG 18. Graf zu der Bändereinteilung

Dies ergibt folgende Bändereinteilung:

TABELLE 1. Bändereinteilungen

Bandnummer	Bandanfang	Bandende	Bandbreite
1	0	90	90
2	90	185	95
3	185	284	99
4	284	384	100
5	384	484	100
6	484	585	101
7	585	687	102
8	687	789	102
9	789	893	104
10	893	1000	107
11	1000	1113	113
12	1113	1235	122
13	1235	1368	133
14	1368	1516	148
15	1516	1684	168
16	1684	1876	192
17	1876	2097	221
18	2097	2354	257
19	2354	2653	299
20	2653	3000	347
21	3000	3403	403
22	3403	3870	467
23	3870	4409	539
24	4409	5030	621
25	5030	5741	711
26	5741	6553	812
27	6553	7476	923
28	7476	8521	1045
29	8521	9700	1179
30	9700	11025	1325

Die Tabelle der Frequenzbänder zeigt, dass unter Einbezug der FFT-Auflösung von 43,07 Hz trotzdem Bänder mit gleicher Breite entstehen. Die Auflösung rührt von der Division von der Abtastfrequenz durch die Anzahl Punkte der FFT her [1].

Der entstehende Hall konnte aber trotzdem vermindert werden.

6.3 GUI

Die grafische Oberfläche des Systems (Abbildung 19) wurde grösstenteils vom bestehenden adaptiven System übernommen und mit zusätzlichen Einstellmöglichkeiten erweitert. So sind neu wegen des Mehrkanalverfahrens auch die maximale Dämpfung, die maximale Dynamik, der Absink- und der Anstiegsfaktor einstellbar. Um diese Einstellungen optisch schneller zu erfassen, wurden Schieberegler implementiert, deren Voreinstellungen auf das System optimiert wurden. Die Regler können via Schieber, Pfeiltasten, Klicken auf den Balken oder durch Direkteingabe des Wertes in das Feld oberhalb, eingestellt werden, auch während dem das System in Betrieb ist. Die beiden Systeme, Adaptiv- und Mehrkanalverfahren, können zu beliebiger Zeit ein- und ausgeschaltet werden. Dies hat den Vorteil, die einzelnen Systeme unabhängig voneinander zu testen bzw. dessen Wirkung zu erfahren und ermöglicht einen Quervergleich zwischen Originalsignal und bearbeitetem Signal. Der Korrekturschritt und die Anzahl der Koeffizienten gehören zum adaptiven Teil, wobei nur die Korrekturschrittweite während des laufenden Systems angepasst werden kann. Die Übersteuerungsanzeige leuchtet rot auf, wenn die Kanäle übersteuert werden. Die Auslastungsanzeige hilft, die Auslastung des DSPs in einem regulären Bereich zu halten, wobei die Anzahl Koeffizienten für eine zu grosse Auslastung verantwortlich ist.

Die Koeffizienten können über den Menüpunkt File ausgedruckt werden. Das Spektrum (Abbildung 20) der Koeffizienten des adaptiven Teils kann ebenfalls ausgedruckt bzw. betrachtet werden, wenn das Menü Spektrum angewählt wird. Um vom Spektrumfenster wieder in das Lärmunterdrückungsfenster zu gelangen, muss das Fenster einfach geschlossen werden.

Das grafische Teilfenster mit den Koeffizienten im Hauptfenster passt sich während des Betriebs laufend den aktuellen Koeffizientenwerten an. Überschreitet der erste Koeffizient den Wert 20, werden die Koeffizienten automatisch auf Null zurückgesetzt. Wenn dies geschieht, ist das System offensichtlich instabil geworden. Es muss nicht neu gestartet werden.

Das Kontrollkästchen Marker stellt die einzelnen Koeffizienten als Matlab-Marker dar.

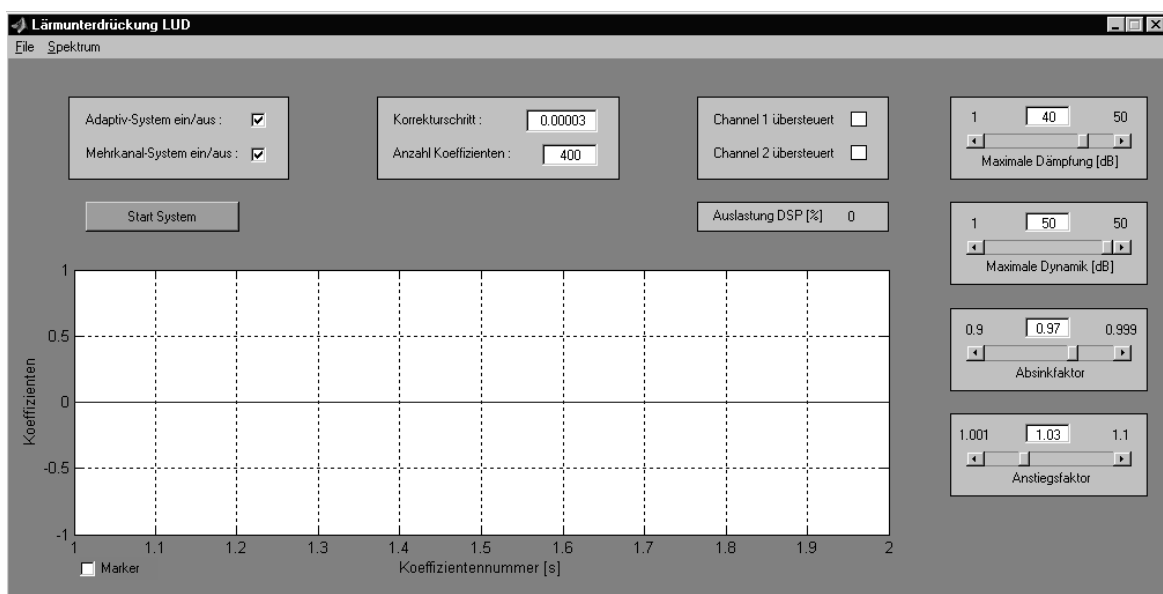


ABBILDUNG 19. Hauptfenster

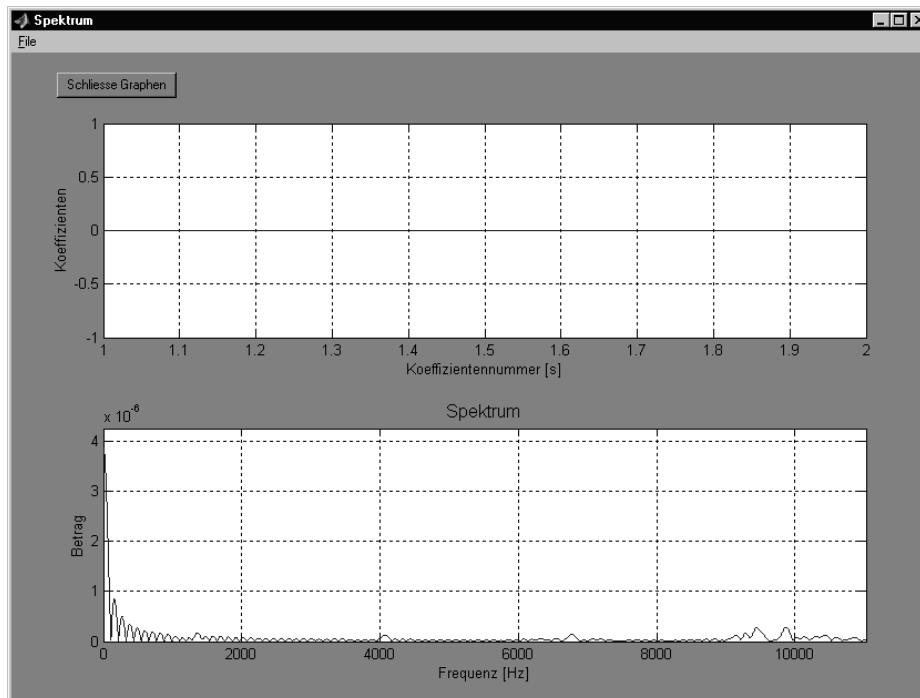


ABBILDUNG 20. Spektrum der Koeffizienten

6.4 Abschliessende Beurteilung

Die objektiven Hörtests haben ergeben, dass das so aufgebaute Lärmunterdrückungssystem bei optimaler Einstellung aller Parameter sehr gute Ergebnisse liefert. Es ist in der Lage, Motoren- und anderen Lärm praktisch vollständig zu unterdrücken, so dass die Sprache deutlich aus den Störgeräuschen hervortritt und verständlicher wird. Allerdings wird dabei auch das Sprachsignal leicht verändert. Durch das adaptive System können Verzerrungen hervorgerufen werden, allerdings nur, wenn die beiden Mikrofone ungenügend akustisch getrennt bzw. zu nahe beieinander sind. Das Mehrkanalsystem führt zu einem unvermeidlichen Halleffekt. Dieser lässt sich durch geeignete Wahl der Parameter minimieren.

7.0 Erläuterungen zum Programmcode

7.1 Aufbau

Das gesamte DSP-Programm besteht aus einer einzigen C-Datei, welche jedoch mehrere Header-Dateien benötigt. Die ganze Berechnung des Algorithmus findet in einer Interrupt-Service-Routine (ISR) statt. Um den Code trotzdem übersichtlich zu gestalten, werden in der ISR mehrere Funktionen aufgerufen. Die Main-Funktion besteht nur aus der Initialisierung und einer leeren Endlosschleife.

7.2 Implementationshinweise

Die Implementation des Lärmunterdrückungsprogrammes LUD beruht im Wesentlichen auf zwei bereits bestehenden Systemen. Im Einzelnen sind dies ein adaptives System [2] und ein Mehrkanalsystem [1]. Sie werden in dieser Anwendung logisch hintereinander geschaltet. Da das adaptive System jeden Input-Sample einzeln bearbeitet und wieder ausgibt, das Mehrkanalsystem aber ganze Blöcke bearbeitet und ausgibt, ist die Verbindung nicht trivial.

Das Problem kann gelöst werden, wenn man ganze Datenblöcke einliest und dann die sich darin befindenden Samples in einer Schleife einzeln vom adaptiven System bearbeiten lässt. Die veränderten Datenblöcke können dann vom Mehrkanalsystem weiterverwendet werden.

Das adaptive System besteht ausschliesslich aus Assembler-Code, da nur so genügend hohe Filterlängen erzielt werden können. Dieser Code macht Gebrauch von sehr vielen Prozessorregistern, die auch der C-Compiler benutzt. Um hier eine doppelte Nutzung von Registern durch den Compiler und den von Hand geschriebenen Assemblercode zu verhindern, benutzt letzterer den alternativen Registersatz des DSPs. Dies hat den grossen Vorteil, dass keine Register auf dem Stack gesichert werden müssen und auch keine Register unbeabsichtigt überschrieben werden können, da die beiden Teile isoliert voneinander ablaufen. Das Umschalten der Register benötigt 1-2 Taktzyklen [8, 10].

7.3 Compilieren

Das DSP-Programm sollte mit der Optimierungsoption `-O` compiliert werden, obwohl die Option `-O2` ca. 15% schnelleren Code produziert. Dieser Code führt aber zu einem kräftigen Einschaltgeräusch beim Starten des Programms, welches mit zusätzlichen Schaltungsmassnahmen unterdrückt werden müsste. Der weniger stark optimierte Code startet hingegen absolut geräuschlos.

8.0 Zusammenfassung

Lärm und Störgeräusche können auf zwei prinzipiell verschiedenen Wegen in ein Sprachübertragungssystem gelangen. Einerseits durch Einkopplung während der Signalverarbeitung und Übertragung und andererseits direkt am Mikrofon. Die zweite Variante ist besonders in lauter Umgebung problematisch und kann zu grossen Verständigungsschwierigkeiten führen. Mit Hilfe eines geeigneten DSP-Systems kann die Sprachqualität deutlich gesteigert werden. Da es verschiedene Lösungsansätze in der Literatur gibt, mussten zuerst einige Verfahren auf ihre Wirksamkeit hin getestet werden. Bei Lärmunterdrückungssystemen spielen immer auch die Charakteristik des Gehörs und der Sprache eine Rolle. Diese mussten deshalb auch berücksichtigt werden.

Das schliesslich realisierte System besteht aus einem adaptiven System mit zwei Mikrofonen und einem Mehrkanalsystem, das auf der Sprachdynamik beruht. Diese beiden Teile arbeiten unabhängig voneinander und können einzeln überbrückt werden. So kann die jeweils günstigste Konfiguration durch Einstellen der Parameter evaluiert werden.

Das adaptive System ist aus der Literatur bereits bekannt und verhält sich prinzipiell unproblematisch, da es bereits gut untersucht und analysiert wurde. Es ergänzt das Mehrkanalsystem optimal, weil es tieffrequente, periodische Störungen gut unterdrücken kann. Diese sind zwar für das menschliche Gehör kaum als solche wahrnehmbar, haben aber einen hohen Pegel und können die Anpassung des Mehrkanalsystems beeinflussen. An die Adaptionsgeschwindigkeit müssen keine hohen Anforderungen gestellt werden, weil die auftretenden Störungen nur sehr langsam ihre Charakteristik ändern.

Das im Frequenzbereich arbeitende Mehrkanalsystem teilt das Frequenzspektrum in 30 Bänder auf. In jedem Band wird dann die Dynamik bestimmt und das Band danach entsprechend abgeschwächt. Wenn kein Sprachsignal anliegt, ist das System praktisch stumm geschaltet. Das Mehrkanalsystem kann allerdings zu starken Sprachverfremdungen führen, wenn es nicht optimal eingestellt ist. Besonders störend kann der entstehende Halleffekt werden, dessen Ausmass allerdings mit geeigneten Einstellungen minimiert werden kann. Ganz eliminieren kann man ihn nicht, da er ein Nebeneffekt des Algorithmus ist.

Das Problem der Motorlärmunterdrückung in Gegensprechanlagen konnte so befriedigend gelöst werden, wobei allerdings für eine maximale Unterdrückung zwei Mikrofone nötig sind. Wenn weniger grosse Ansprüche an die Sprachqualität gestellt werden, kann auch nur mit dem Mehrkanalsystem gearbeitet werden. Es wird dann nur ein Mikrofon benötigt.

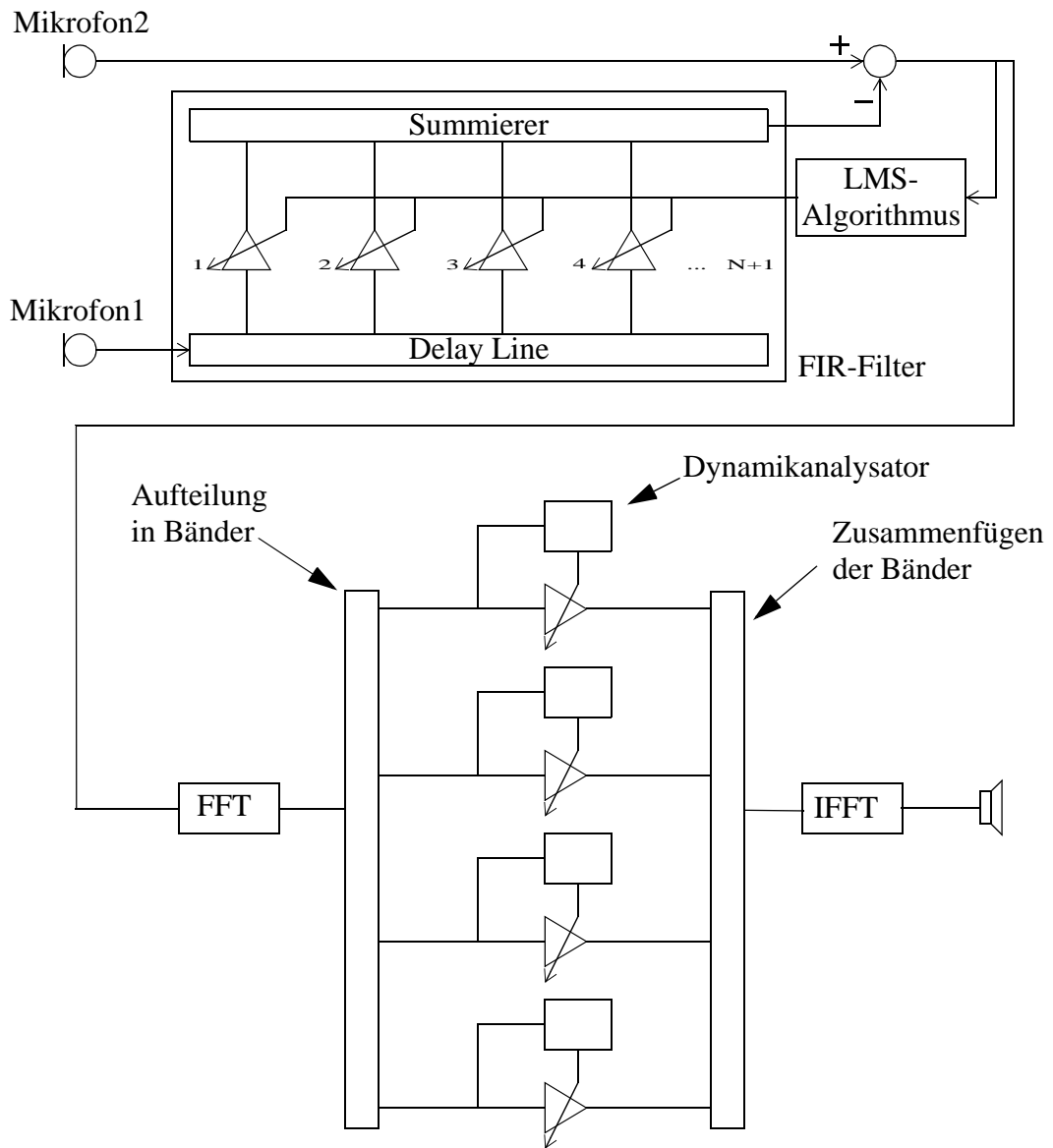


ABBILDUNG 21. Blockschaltbild des realisierten DSP-Systems

9.0 Referenzen

- [1] Mehrkanalige Störgeräuschunterdrückung, A. Eggenberger / M. Jud
Diplomarbeit Hochschule Rapperswil, 1999
- [2] Adaptives System, Wassim G. Najm, Analog Devices, DSP Division
Modifiziert von A. Rüegg Hochschule Rappeswil.
- [3] Digitale Sprachsignalverarbeitung, P. Vary / U. Heute / W.Hess
B. G. Teubner Stuttgart, 1998
- [4] Digital Speech Processing, A. Nejat Ince
Kluwer Academic Publishers Boston / Dordrecht / London 1995
- [5] The Digital Signal Processing Handbook, Vijay K. Madisetti / Douglas B. Williams
CRC Press / IEEE Press, 1998
- [6] Autographie Digitale Signalverarbeitung, A. Schüeli
Hochschule Rapperswil
- [7] Praktikumsversuch Adaptives FIR-Filter, Digitale Signalverarbeitung, A. Schüeli
Hochschule Rapperswil
- [8] ADSP-2106X SHARC User's Manual,
Analog DevicesInc., 1997
- [9] ADSP-21000 Family C Runtime Library Manual,
Analog DevicesInc., 1995
- [10] ADSP-21000 Family C Tools Manual,
Analog DevicesInc., 1995
- [11] ADSP-21000 Family Assembler Tools & Simulator Manual,
Analog DevicesInc., 1995
- [12] Spürnasen Adaptive Filter in der Signalverarbeitung, Prof. Dr. B.Wirnitzer /
Chr. Schönig / W.Seipp
ELRAD 8/1994 S.28
- [13] A simple adaptive first-order differential microphone, Gary W. Elko, Anh-Tho
Nguyen Pong
AT&T Bell LaboratoriesAcoustics Research Department
- [14] Graphics and GUIs with Matlab, second edition, Patrick Marchand
CRC Press, 1999

10.0 Anhang

10.1 Verzeichnisstruktur

Stammverzeichnis

bericht.fm	Dokumentation
berichtIVZ.fm	Inhaltsverzeichnis
berichtAVZ.fm	Abbildungsverzeichnis
bericht.pdf	Dokumentation im pdf-Format

Unterverzeichnisse

\Bilder	*.*	Diverses Bildmaterial
\Code	lud.m lud.c lud.21k sharc.dll	Matlab-GUI C-Quellcode Ausführbarer Code Sharc-Bibliothek
\Matlab	*.fig *.m	Diverse Figuren im Matlab-Format Matlab-Code für diverse Figuren
\Sound	*.wav	Diverse Soundfiles

10.2 Abbildungsverzeichnis

ABBILDUNG 1.Blockschaltbild des Aufbaus.....	3
ABBILDUNG 2.Adaptive Noise Cancelling zur Lärmunterdrückung in einer Sprachübertragung	4
ABBILDUNG 3.Analyse eines Personenwagens.....	7
ABBILDUNG 4.Analyse eines kleinen Lieferwagens.....	8
ABBILDUNG 5.Analyse eines Lastkraftwagens.....	9
ABBILDUNG 6.Analyse der menschlichen Sprache	10
ABBILDUNG 7.Normalkurven gleicher Lautstärkepegel und Hörschnelle für Sinustöne im freien Schallfeld bei binauralem Hören nach Robinson und Dadson (1959).....	11
ABBILDUNG 8.Blockschaltbild des Verschiebungs-Verfahrens	12
ABBILDUNG 9.Blockschaltbild eines einfachen Beamforming	14
ABBILDUNG 10.Korrektur des Koeffizienten c_n in Richtung Minimum.....	16
ABBILDUNG 11.Blockschaltbild des adaptiven Systems mit zwei Mikrofonen	16
ABBILDUNG 12.Blockschaltbild des adaptiven Systems mit einem Mikrofon.....	17
ABBILDUNG 13.Blockschaltbild des Mehrkanalsystems	19
ABBILDUNG 14.Möglicher Frequenzgang des Mehrkanalsystems	19
ABBILDUNG 15.Ein einfacher Fall einer Störung	20
ABBILDUNG 16.oben: Frequenzgang des Mehrkanalsystems unten: Resultierende Stossantwort des Mehrkanalsystems.....	21
ABBILDUNG 17.Blockschaltbild des Gesamtsystems	22
ABBILDUNG 18.Graf zu der Bändereinteilung.....	24
ABBILDUNG 19.Hauptfenster	26
ABBILDUNG 20.Spektrum der Koeffizienten.....	27
ABBILDUNG 21.Blockschaltbild des realisierten DSP-Systems	30

10.3 Quellcode

10.3.1 C-Quellcode

```

/*****
* LUD - Lärmunterdrückung
*****
*Author  : Daniel Kägi, Michael Meyer
*Version : 1.0
*Datum   : 13.06.2001
*
*****
*Basiert auf der Diplomarbeit 'Mehrkanalige Störgeräuschunterdrückung' von
*Mischa Jud & Adrian Eggenberger 1999 und dem Adaptiven Algorithmus von
*Analog Devices und André Rüegg HSR
*****/

/* ADSP-21060 System Register bit definitions */
#include <def21060.h> // register definitions
#include <21060.h> // flag / timer definitions
#include <signal.h> // interrupt
#include <sport.h> // serial port
#include <trans.h> // rfft, ifft
#include <math.h> // fmod
#include <macros.h> // MIN, circular-buffer
#include "ctrlregs.h"
#include "bitsibb.h" // bitsi board
#include "bitsi.h"
asm("#include <def21060.h>");

/*****
* Konstanten und Typedef's
*****/
#define CP_PCI 0x20000 /* Program-Controlled Interrupts bit */
#define CP_MAF 0x1ffff /* Valid memory address field bits */

/* macros */
#define SetIOP(addr, val) (* (int *) addr) = (val)
#define GetIOP(addr) (* (int *) addr)

/* Convert a 16 bit Integer value (the 16 left bits in a 32 bit word x)
to a 32 bit Integer value */
#define convertleft16to32(result, x) \
asm("%0=fext %1 by 16:16 (se);" : "=d" (result) : "d" (x));

/* Convert a 16 bit Integer value (the 16 right bits in a 32 bit word x)
to a 32 bit Integer value */
#define convertright16to32(result, x) \
asm("%0=fext %1 by 0:16 (se);" : "=d" (result) : "d" (x));

#define MAX_U_AD 2.75 /* maximum input voltage for A/D converter */
#define MAX_U_DA 3.0 /* maximum output voltage on D/A converter */

/* DMA chaining Transfer Control Blocks */
typedef struct {
    unsigned** cp; /* Chain Pointer to next TCB */
    unsigned c; /* Count register */
    int im; /* Index modifier register */
    unsigned * ii; /* Index register */
} _tcb_;

#define NUM_BUFS 2 /* number of IFFT buffers needed */
#define TCB_BUFS 3 /* number of TCB buffers needed */
#define HALF_FFT_SIZE256 /* half FFT length */
#define FFT_SIZE512 /* FFT length */
#define rfft rfft512
#define ifft ifft512

#define RECHTECK 1
#define HALBSINUS 2
#define ANZAHL_BAENDER 30

#define SetIOP(addr, val) (* (int *) addr) = (val)
#define GetIOP(addr) (* (int *) addr)

#define MAX_U_AD 2.75 /* maximum input voltage for A/D converter */

```

```

#define MAX_U_DA 3.0 /* maximum output voltage on D/A converter */
#define PI 3.14159265359
#define MAXTAPS 1000 /* Max. number of filtercoefficients */
asm("#define MAXTAPS 1000");

#define fc 22050.0

#define NORM (MAX_U_AD / 32767.0)
#define CORR (32767.0 / (MAX_U_DA*FFT_SIZE))

/*****
/* Matlab Variablen */
/*****
volatile float Absinkfaktor=0.97; /*Parameter des MKS-Systems*/
volatile float Anstiegsfaktor=1.03;
volatile int MaxDaempfung=40;
volatile int MaxDynamik=50;
//volatile int BaenderVariante=1; /*wählt die Bändeinteilung aus*/
volatile int clocks; /*Anzahl Taktcyclen pro Interupt*/
volatile int adapon=1; /*Adaptive System on/off */
volatile int mkon=1; /*MKS System on/off*/
volatile int sync_flag=1; /*Synchronisation mit Matlab*/
volatile int taps = 400; /* Number of filtercoefficients */
volatile float stepsize = 0.00003; /* Stepsize for correction of the system */
volatile int currentinfo[2]={0,0};

/*****
* Variablen für das Mehrkanalsystem *
/*****
int rx0_buf[TCB_BUFS][HALF_FFT_SIZE]; /* receive buffer */
int tx0_buf[TCB_BUFS][HALF_FFT_SIZE]; /* transmit buffer */

_tcb_rx0_tcb[TCB_BUFS] = { {0, HALF_FFT_SIZE, 1, 0}, /* receive tcb */
{0, HALF_FFT_SIZE, 1, 0}, /* receive tcb */
{0, HALF_FFT_SIZE, 1, 0} }; /* receive tcb */

_tcb_tx0_tcb[TCB_BUFS] = { {0, HALF_FFT_SIZE, 1, 0}, /* transmit tcb */
{0, HALF_FFT_SIZE, 1, 0}, /* transmit tcb */
{0, HALF_FFT_SIZE, 1, 0} }; /* transmit tcb */

/*FFT variables: rin -> real input, rout->real output, iout-> imaginary output*/
float FFT_rin[FFT_SIZE],FFT_rin[FFT_SIZE],FFT_rout[FFT_SIZE],FFT_iout[FFT_SIZE];

/*IFFT variables: rin->real inp,iin->imag inp,rout->real outp,iout->imag outp*/
float
IFFT_rin[FFT_SIZE],IFFT_iin[FFT_SIZE],IFFT_rout[FFT_SIZE],IFFT_iout[FFT_SIZE];

int tx_ptr; /* Pointer to the newer transmit DMA Buffer */
int rx_ptr = 0; /* Pointer to the newer receive DMA Buffer */

volatile float ch1=0;
volatile float ch2=0;
volatile float out=0;
float outalt[HALF_FFT_SIZE];
float workvar[FFT_SIZE];
float outvar[FFT_SIZE];
float pastoutvar[HALF_FFT_SIZE];

int BAENDER[ANZAHL_BAENDER][2] =
{{0,90},{90,185},{185,284},{284,384},{384,484},
{484,585},{585,687},{687,789},{789,893},{893,1000},
{1000,1113},{1113,1235},{1235,1368},{1368,1516},{1516,1684},
{1684,1876},{1876,2097},{2097,2354},{2354,2653},{2653,3000},
{3000,3403},{3403,3870},{3870,4409},{4409,5030},{5030,5741},
{5741,6553},{6553,7476},{7476,8521},{8521,9700},{9700,12000}};

float Windowfactors[FFT_SIZE]; // Variablen der Fensterfunktion

struct WerteImBandStrukt
{
int Erster;
int Letzter;
int Anzahl;
};

float Baender_Frequenzabstand; // Variablen der Baenderfunktion
struct WerteImBandStrukt WerteImBand [ANZAHL_BAENDER];
float BandMittelWerte[ANZAHL_BAENDER];
float BandMinimalWerte[ANZAHL_BAENDER];
float BandMaximalWerte[ANZAHL_BAENDER];

```

```

float  BandDynamikWerte[ANZAHL_BAENDER];
float  BandFaktorenWerte[ANZAHL_BAENDER];
float  SpektralFaktoren[FFT_SIZE];

/*****
 * Variablen für das Adaptive System
 *****/

float dm deline_data[MAXTAPS];      /* circular delay line buffer      */
float pm weights[MAXTAPS];          /* circular weight buffer (Filtercoef.) */

float d, e, y;                       /* d(n), e(n), y(n)                */
float pn = 0;                        /* Input power of u(n)              */
#define C 0.001;                      /* For calculating the input power  */
float c = C;
float c_ = 1.0-C;
float dlx, dly, pole=0.98;           /* DC Blocking Filter              */

/*****
 * Funktionen *
 *****/

//-----
// void ArraysInit(void)
// initialisiert Arrays mit 0
//-----
void ArraysInit(void)
{
int t;
for(t=0;t<HALF_FFT_SIZE;t++) // Initialisierung der Read und Write - Buffer
{
rx0_buf[0][t] = 0;
rx0_buf[1][t] = 0;
rx0_buf[2][t] = 0;
tx0_buf[0][t] = 0x0;
tx0_buf[1][t] = 0x0;
tx0_buf[2][t] = 0x0;
outalt[t]=0.0;
pastoutvar[t]=0.0;
}
}

//-----
// void Fenster_UseWindowfactors ( float* _InVector, float* _OutVector )
// Fensterfunktion auf einen Vektor anwenden
//-----
void Fenster_UseWindowfactors(float* _InVector,float* _OutVector)
{
int z1;
for (z1=0;z1<FFT_SIZE;z1++)
{
_OutVector[z1] = _InVector[z1] * Windowfactors[z1];
}
}

//-----
// void Fenster_Create_Windowfactors ( unsigned char _Windowfunction )
// Windowfaktoren in der entsprechenden Form und Laenge generieren
//-----
void Fenster_Create_Windowfactors ( unsigned char _Windowfunction )
{
int z1 = 0;
double pi = 3.14159265359;

switch (_Windowfunction)
{
case RECHTECK: for (z1 = 0;z1 < FFT_SIZE;z1++)
{
Windowfactors[z1] = 1;
}
break;
case HALBSINUS: for (z1 = 0;z1 < FFT_SIZE>>1;z1++)
{
Windowfactors[z1] = sin(z1*pi/FFT_SIZE);
}
for (z1 = 0;z1 < FFT_SIZE>>1;z1++)
{
Windowfactors[FFT_SIZE-1-z1] = Windowfactors[z1];
}
break;
}
}

```

```

// Defaultmaessig wird Rechteckfenster erzeugt
default :      for (z1 = 0; z1 < FFT_SIZE; z1++)
                {
                    Windowfactors[z1] = 1;
                }
}

//-----
// void Baender_Calculate_WerteImBand ()
// Aufteilen des Spektrums in Baender
//-----
void Baender_Calculate_WerteImBand ()
{
    int Band;
    int WertNr;
    float Frequenz;

    Baender_Frequenzabstand = fc / FFT_SIZE;
    for(Band=0; Band<ANZAHL_BAENDER; Band++)
    {
        Frequenz = 0;
        WerteImBand [Band].Erster = -1;
        WerteImBand [Band].Letzter = -1;
        for(WertNr= 0; WertNr<HALF_FFT_SIZE; WertNr++)
        {
            if ((WerteImBand[Band].Erster==-1)&&(Frequenz>=BAENDER[Band][0]))
            {
                WerteImBand[Band].Erster=WertNr; // Erster Wert des Bandes
            }
            if (Frequenz < BAENDER[Band][1])
            {
                WerteImBand[Band].Letzter = WertNr; // Letzter Wert des Bandes
            }
            Frequenz += Baender_Frequenzabstand; // naechste Frequenz errechnen
        }
        WerteImBand[Band].Anzahl=WerteImBand[Band].Letzter-
        WerteImBand[Band].Erster+1;
    }
}

//-----
// void Baender_Get_BandAmplitudenMittelwerte ( float* _Spektrum_Real,
// float* _Spektrum_Imag, BandWerteStrukt *_BandMittelwerte)
// Aus einem Frequenzspektrum werden die Bandmittelwerte beechnet
//-----
void Baender_Get_BandAmplitudenMittelwerte ( float* _Spektrum_Real, float*
_Spektrum_Imag, float* _BandMittelwerte)
{
    int Band;
    int WertNr;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ ) // Baender durchzaehlen
    {
        // Amplitudenwerte eines Bandes quadratisch addieren

        for(WertNr=WerteImBand[Band].Erster;WertNr<=WerteImBand[Band].Letzter;WertNr++)
        {
            //Spectrallinien aufsumieren
            _BandMittelwerte[Band]+=( _Spektrum_Real[WertNr]*_Spektrum_Real
            [WertNr]+_Spektrum_Imag[WertNr]*_Spektrum_Imag [WertNr]);
        }
        // Wurzel ziehen
        _BandMittelwerte[Band]=sqrt(_BandMittelwerte[Band]/
        WerteImBand[Band].Anzahl);
    }

//-----
// void Baender_Get_BandMinimalwerte (struct BandWerteStrukt *_BandMittelwerte,
// struct BandWerteStrukt *_BandMinimalwerte)
// Bestimmen der neuen Bandmaximalwerte aus den Bandmittelwerten und früheren
// Bandmaximalwerten
//-----
void Baender_Get_BandMaximalwerte (float* _BandMittelwerte, float*
_BandMaximalwerte)
{
    int Band;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ ) // Baender durchzaehlen
    {
        if (_BandMaximalwerte[Band]*Absinkfaktor <= _BandMittelwerte[Band])
        {
            _BandMaximalwerte[Band] = _BandMittelwerte[Band];
        }
    }
}

```

```

else
{
    _BandMaximalWerte[Band] = _BandMaximalWerte[Band]*Absinkfaktor;
}
}
}

//-----
// void Baender_Get_BandMinimalWerte (struct BandWerteStrukt *_BandMittelwerte,
// struct BandWerteStrukt *_BandMinimalWerte)
// Bestimmen der neuen Bandminimalwerte aus den Bandmittelwerten und früheren
// Bandminimalwerten
//-----
void
Baender_Get_BandMinimalWerte(float*_BandMittelwerte,float*_BandMinimalWerte)
{
    int Band;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ ) // Baender durchzaehlen
    {
        if(_BandMinimalWerte[Band]*Anstiegsfaktor>_BandMittelwerte[Band])
        {
            _BandMinimalWerte[Band]=_BandMittelwerte[Band];
        }
        else
        {
            _BandMinimalWerte[Band]=_BandMinimalWerte[Band]*Anstiegsfaktor;
        }
    }
}

//-----
// void Baender_Get_BandDynamik (struct BandWerteStrukt *_BandMaximalWerte,
// struct BandWerteStrukt *_BandMinimalWerte,
// struct BandWerteStrukt *_BandDynamikWerte)
// Bestimmen der Banddynamik aus Maximal- und Minimalwerten
//-----
void Baender_Get_BandDynamik( float*_BandMaximalWerte,float*_BandMinimalWerte,
float*_BandDynamikWerte)
{
    int Band;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ ) // Baender durchzaehlen
        { // Dynamik errechnen
            _BandDynamikWerte[Band]=20*log10f(_BandMaximalWerte[Band]/
            _BandMinimalWerte[Band]);
        }
}

//-----
// void Baender_Get_BandFaktoren (struct BandWerteStrukt *_BandDynamikWerte,
// struct BandWerteStrukt *_BandFaktorenWerte)
// Bestimmen der Bandfaktoren aus der Dynamik des Bandes
// (Dynamik-Daempfung-Kennlinie)
//-----
void Baender_Get_BandFaktoren(float*_BandDynamikWerte,float*_BandFaktorenWerte)
{
    int Band;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ ) // Baender durchzaehlen
    {
        // Bandfaktoren berechnen
        _BandFaktorenWerte[Band]=1/powf(10,0.05*(MaxDaempfung-((float)MaxDaempfung/
(float)MaxDynamik)*MIN(MaxDynamik,_BandDynamikWerte[Band])));
    }
}

//-----
// void Baender_Get_SpektralFaktoren ( struct BandWerteStrukt
// _BandFaktorenWerte, double *_SpektralFaktoren )
// Fuer jeden Wert des Spektrums wird sein Spektralfaktor berechnet
//-----
void Baender_Get_SpektralFaktoren( float*_BandFaktorenWerte,
float*_SpektralFaktoren)
{
    int Band;
    int WertNr;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ ) // Baender durchzaehlen
    {
        // Werte der ersten Haelfte des Spektrums
        for(WertNr=WerteImBand[Band].Erster;WertNr<=WerteImBand[Band].Letzter;WertNr++)
        {
            _SpektralFaktoren[WertNr] = _BandFaktorenWerte[Band];
        }
    }
}

```

```

// Werte der zweiten Haelfte des Spektrums
for(WertNr=FFT_SIZE-WerteImBand[Band].Letzter-1; WertNr<=FFT_SIZE-
WerteImBand[Band].Erster-1; WertNr++)
{
    _SpektralFaktoren[WertNr] = _BandFaktorenWerte[Band];
}
}

//-----
//voiBaender_Calculate_Spektrum(float*_SpektralFaktoren,float*_Spektrum_In_Real,
//float*_Spektrum_In_Imag, float*_Spektrum_Out_Real, float*_Spektrum_Out_Imag)
// Ein Frequenzspektrum wird mit Spektralfaktoren multipliziert
//-----
void Baender_Calculate_Spektrum(float*_SpektralFaktoren,float*_Spektrum_In_Real,
float*_Spektrum_In_Imag, float*_Spektrum_Out_Real, float*_Spektrum_Out_Imag)
{
    int WertNr;
    for(WertNr=0; WertNr<FFT_SIZE; WertNr++) // Spektralwerte durchzaehlen
    {
        _Spektrum_Out_Real[WertNr]=_SpektralFaktoren[WertNr]*_Spektrum_In_Real[WertNr];
        _Spektrum_Out_Imag[WertNr]=_SpektralFaktoren[WertNr]*_Spektrum_In_Imag[WertNr];
    }
}

//-----
// void Baender_Preset_BandWerte(struct BandWerteStrukt*_BandWerte,double _Wert)
// Daten im BandWerteStrukt koennen vorgesetzt werden
//-----
void Baender_Preset_BandWerte (float *_BandWerte, float _Wert)
{
    int Band;
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        _BandWerte[Band] = _Wert;
    }
}

```

```

//-----
// void nlms(void)
// LMS algorithm implemented with a transversal FIR filter structure
//-----
void nlms( )
{
    asm("
        bit set MODE1 (SRD1H|SRD1L|SRD2H|SRD2L|SRRFH|SRRFL|SRCU);
        nop; /*alternativer Registersatz ein*/

        f1=dm(_ch2);
        f0=dm(_ch1);

        // Normalize Step Size
        // p(n)=(1-c)*p(n-1)+c*u(n)^2
        f4=dm(_pn); ! f4=p(n-1)
        f8=dm(_c_); ! f8=1-c
        f4=f8*f4; ! f4=(1-c)*p(n-1)
        f2=f0*f0; ! u(n)^2
        f12=dm(_c); ! f2=c
        f8=f12*f2; ! f8=c*u(n)^2
        f12=f4+f8; ! f12=p(n)=(1-c)*p(n-1)+c*u(n)^2
        dm(_pn)=f12;

        // Clear adaptive filter and mute the output
        // if the first Filtercoefficient is to big
        // if the filter is running away, it is faster in a decedent state
        // (not overdrived) if he comes back
        f8=pm(0,i8); ! f8=w(n)
        f2=20.0; // Nicht gut? Kann Koeffizient grösser 20 erzeugt werden?
        f8=abs f8;
        f8=f8-f2;
        if lt jump(pc,powerok); ! if (abs(w(n))<20) powerok
        // Clear the adaptive filter and mute the output if the inputpower p(n)
        // is too big.
        // Not absolut necessary if constat C is enough little. Adaptive filter
        // starts softer (it takes less time to be out of overdrive), if he is
        // switched on after he is switched off.
    ");
}

```

```

powerbad:
    f8=0.0;
    r2=dm(_taps);          ! Number of filtercoefficients
    lcntr=r2, do clearfilter until lce;
    clearfilter:
    dm(i0,m0)=f8,pm(i8,m8)=f8;    ! u(i)=0, w(i)=0
    r8=0;
    dm(TX0)=r8;              ! write to IO-Port
    rti;
powerok:
    //p(n)=p(n)+delta
    f8=0.005;
    f12=f12+f8;

    // mue=STEPSSIZE/(p(n)+delta)
    // floating-point division using an iterative convergence algorithm
    // See ADSP-2106x Sharc User's Manual
    f11=2.0;
    f2=dm(_stepsize);
    f2=recips f12, f7=f2;      ! Get 8 bit seed f2=1/p(n)
    f12=f2*f12;              ! f12=p(n)'=p(n)*seed(1/p(n))
    f7=f2*f7, f2=f11-f12;    ! f7=STEPSSIZE*seed(1/p(n)), f2=2.0-p(n)'
    f12=f2*f12;              ! f12=(2.0-p(n)')*p(n)'
    f7=f2*f7, f2=f11-f12;    ! f7=(2.0-p(n)')*STEPSSIZE*seed(1/p(n)),
    ! f2=2.0-(2.0-p(n)')*p(n)'
    f12=f2*f12;              ! f12=p(n)'=(2.0-(2.0-p(n)')*p(n)')
    ! *(2.0-p(n)')*p(n)'
    f7=f2*f7, f2=f11-f12;    ! f7=(2.0-(2.0-p(n)')*p(n)')
    ! *(2.0-p(n)')*STEPSSIZE*seed(1/p(n)),
    ! f2=2.0-p(n)'
    f7=f2*f7;                ! f2=2.0-p(n)')*f7
    /*performs the following LMS algorithm implemented with a transversal
    *FIR filter structure
    *****
    * 1) y(n)= w.u ( . = dot_product), y(n)= FIR filter output      *
    * where w= [w0(n) w1(n) ... wN-1(n)]= filter weights            *
    * and u= [u(n) u(n-1) ... u(n-N+1)]= input samples in delay line*
    * n= time index, N= number of filter weights (taps)            *
    * 2) e(n)= d(n)-y(n), e(n)= error signal & d(n)= desired output *
    * 3) wi(n+1)= wi(n)+STEPSSIZE*e(n)*u(n-i), 0 <=i<= N-1      *
    *****

```

Written by: Wassim G. Najm , Analog Devices, DSP division, April 2 1991
 Calling parameters (inputs):
 f0= u(n) = input sample
 f1= d(n) = desired output

Altered registers:
 f0, f1, f4, f6, f7, f8, f12, f13

Computation cycles:
 lms_alg: 3N+8 per iteration, lms_init: 12+N

Results (outputs):
 f13= y(n)= filter output
 f6= e(n)= filter error signal
 i8 -> Program Memory Data buffer of the filter weights

Memory usage:
 pm code= 29 words, pm data= N words, dm data= N words
 */

```

lms_alg:
    dm(i0,m0)=f0, f4=pm(i8,m8); /*store u(n) in delay line, f4=w0(n)*/
    f8=f0*f4, f0=dm(i0,m0), f4=pm(i8,m8); /*f8=u(n)*w0(n),f0=u(n-1),f4=w1(n)*/

    f12=f0*f4,f0=dm(i0,m0),f4=pm(i8,m8);/*f12=u(n-1)*w1(n),f0=u(n-2),f4=w2(n)*/
    r6=dm(_taps);
    r13=3;
    r6=r6-r13;
    lcntr=r6, do macs until lce;
    macs:f12=f0*f4, f8=f8+f12, f0=dm(i0,m0), f4= pm(i8,m8); /*f12= u(n-i)*wi(n), f8=
    sum of prod, f0= u(n-i-1), f4= wi+1(n)*/
    f12=f0*f4, f8=f8+f12; /*f12= u(n-N+1)*wN-1(n)*/
    f13=f8+f12; /*f13= y(n)*/
    dm(_y)=f13;
    f6=f1-f13;          /* f6= e(n) */

    f1=f6*f7, f4=dm(i0,m0); /*f1= STEPSSIZE*e(n), f4= u(n)*/
    f0=f1*f4, f12=pm(i8,m8); /*f0= STEPSSIZE*e(n)*u(n), f12= w0(n)*/
    r8=dm(_taps);
    r8=r8-1;
    lcntr=r8, do update_weights until lce;
    f8=f0+f12,f4=dm(i0,m0),f12=pm(i8,m8);/*f8=wi(n+1),f4=u(n-i-1),f12=wi+1(n)*/

```

```

update_weights:
    f0=f1*f4, pm(i9,m8)=f8; /*f0= STEPSIZE*e(n)*u(n-i-1), store wi(n+1)*/
    f8=f0+f12, f0=dm(i0,1); /*f8=wN-1(n+1), i0->u(n+1) location in delay line*/
    pm(i9,m8)=f8; /*store wN-1(n+1)*/

    // DC Blocking Filter
    f4=dm(_dlx); // Differentiator
    dm(_dlx)=f6;
    f6=f6-f4;
    f4=dm(_pole); // Leaky Integrator
    f13=dm(_dly);
    f4=f4*f13;
    f6=f4+f6;
    dm(_dly)=f6;

    f4=dm(_y);
    dm(_out)=f6;

    bit clr MODE1 (SRD1H|SRD1L|SRD2H|SRD2L|SRRFH|SRRFL|SRCU);
    nop; /*alternativer Registersatz aus*/
");
}

//-----
// serial port receive DMA complete
// übernimmt die komplette Berechnung
//-----
void spr0_asserted( int sig_num )
{
    int temp;
    int index;

    timer_set (0xff, 0xffffffff);
    timer_on ();

    asm(" // cache wieder aktivieren*/
        bit clr MODE2 CADIS;
        bit clr MODE2 CAFRZ;
        nop;
    ");
}

/* find pointer to actual receive and transmit buffer */
rx_ptr = fmodf(rx_ptr, 3.0); // Pointer to the newer DMA Read Buffer
tx_ptr = fmodf(rx_ptr+2, 3.0); // Pointer to the next DMA Write Buffer

for (index=0; index<HALF_FFT_SIZE; index++)
{
    convertleft16to32 (temp,rx0_buf[rx_ptr][index]);
    currentinfo[1]=temp;
    ch1= (NORM * ((float)(temp)));
    convertright16to32(temp,rx0_buf[rx_ptr][index]);
    currentinfo[0]=temp;
    ch2= (NORM * ((float)(temp)));
    nlms(); // benutzt ch1,ch2,out
    if(!adapon) out=ch2; // Bypass
    workvar[index + HALF_FFT_SIZE]=out;
    workvar[index]=outalt[index];
    outalt[index]=out;
}

Fenster_UseWindowfactors(workvar,workvar);
rfft(workvar, FFT_rout, FFT_iout);

Baender_Get_BandAmplitudenMittelwerte(FFT_rout,FFT_iout,BandMittelWerte);
Baender_Get_BandMaximalWerte(BandMittelWerte,BandMaximalWerte);
Baender_Get_BandMinimalWerte(BandMittelWerte,BandMinimalWerte);
Baender_Get_BandDynamik (BandMaximalWerte,BandMinimalWerte,BandDynamikWerte);
Baender_Get_BandFaktoren(BandDynamikWerte,BandFaktorenWerte);
Baender_Get_SpektralFaktoren(BandFaktorenWerte,SpektralFaktoren);

if (mkon)
{ // Neues manipuliertes Spektrum berechnen

Baender_Calculate_Spektrum(SpektralFaktoren,FFT_rout,FFT_iout,IFFT_rin,IFFT_iin)
;
}
}

```

```

else
{ //Bypass
  for(index=0;index<FFT_SIZE;index++)
  {
    IFFT_rin[index] = 0.2*FFT_rout[index];
    IFFT_iin[index] = 0.2*FFT_iout[index];
  }
}

ifft(IFFT_rin,IFFT_iin,IFFT_rout,IFFT_iout);
Fenster_UseWindowfactors(IFFT_rout,outvar);

/* fill chained DMA transmit buffer with result */
for (index=0;index<HALF_FFT_SIZE;index++)
{
  tx0_buf[tx_ptr][index] = (unsigned)(pastoutvar[index]+outvar[index]*CORR) &
0x0000ffff ;
  pastoutvar[index] = outvar[index + HALF_FFT_SIZE]*CORR;
}
rx_ptr++; /* increment pointer to next buffer address */

clocks = 0xffffffff-timer_off();
}

//-----
// void setup_sport0( void )
// Serial Port 0 initialisieren
//-----
void setup_sport0( void )
{
  /* Configure SHARC serial port */
  /* TRANSMIT CONTROL REGISTER */
  sport0_iop.txc.mdf = 0; /* multichannel frame delay (MFD) */
  sport0_iop.txc.schen = 1; /* Tx DMA chaining enable */
  sport0_iop.txc.sden = 1; /* Tx DMA enable */
  sport0_iop.txc.lafs = 0; /* Late TFS (alternate) */
  sport0_iop.txc.ltfs = 1; /* Active low TFS */
  sport0_iop.txc.ditfs = 0; /* Data independent TFS */
  sport0_iop.txc.itfs = 0; /* Internally generated TFS */
  sport0_iop.txc.tfsr = 1; /* TFS Required */

  sport0_iop.txc.ckre = 0; /* Data and FS on clock rising edge */
  sport0_iop.txc.gclk = 0; /* Enable clock only during transmission*/
  sport0_iop.txc.iclk = 0; /* Internally generated Tx clock */
  sport0_iop.txc.pack = 0; /* Unpack 32b words into two 16b tx's */

  sport0_iop.txc.slen = 31; /* Data word length minus one */
  sport0_iop.txc.sendn = 0; /* Data word endian 1 = LSB first */
  sport0_iop.txc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
  /* Data type specifier */
  sport0_iop.txc.spen = 1; /* Enable (clear for MC operation) */

  /* RECEIVE CONTROL REGISTER */
  sport0_iop.rxc.nch = 31; /* multichannel number of channels - 1 */
  sport0_iop.rxc.mce = 0; /* multichannel enable */
  sport0_iop.rxc.spl = 0; /* Loop back configure (test) */
  sport0_iop.rxc.d2dma = 0; /* Enable 2-dimensional DMA array */
  sport0_iop.rxc.schen = 1; /* Rx DMA chaining enable */
  sport0_iop.rxc.sden = 1; /* Rx DMA enable */
  sport0_iop.rxc.lafs = 0; /* Late RFS (alternate) */
  sport0_iop.rxc.ltfs = 1; /* Active low RFS */
  sport0_iop.rxc.irfs = 0; /* Internally generated RFS */
  sport0_iop.rxc.rfsr = 1; /* RFS Required */
  sport0_iop.rxc.ckre = 0; /* Data and FS on clock rising edge */
  sport0_iop.rxc.gclk = 0; /* Enable clock only during transmission*/
  sport0_iop.rxc.iclk = 0; /* Internally generated Rx clock */
  sport0_iop.rxc.pack = 0; /* Pack two 16b rx's into 32b word */

  sport0_iop.rxc.slen = 31; /* Data word length minus one */
  sport0_iop.rxc.sendn = 0; /* Data word endian 1 = LSB first */
  sport0_iop.rxc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
  /* Data type specifier */
  sport0_iop.rxc.spen = 1; /* Enable (clear for MC operation) */

  /* Enable sport0 xmit & rcv irqs */
  interruptf(SIG_SPR0I, spr0_asserted);

  /* Set up Transmit Transfer Control Block for chained DMA */
  tx0_tcb[0].ii = tx0_buf[0]; /* DMA source buffer address */
  tx0_tcb[1].ii = tx0_buf[1]; /* DMA source buffer address */
  tx0_tcb[2].ii = tx0_buf[2]; /* DMA source buffer address */
}

```

```

tx0_tcb[0].cp = &tx0_tcb[1].ii; /* define ptr to next TCB (point to self) */ //-----
tx0_tcb[1].cp = &tx0_tcb[2].ii; /* define ptr to next TCB (point to self) */ // void wait()
tx0_tcb[2].cp = &tx0_tcb[0].ii; /* define ptr to next TCB (point to self) */ // wartet einen Moment (Delay)
                                                                    //-----
SetIOP(CP2, (((int)&tx0_tcb[0].ii) & CP_MAF) | CP_PCI);
                                                                    void wait()
                                                                    {
                                                                    /* define ptr to current TCB (kick off DMA) */
                                                                    /* (SPORT0 transmit uses DMA ch 2) */
                                                                    volatile unsigned int q;
                                                                    volatile int a=0;

                                                                    for(q=0;q<0xffff0;q++)
                                                                    {
                                                                    a++;
                                                                    }
                                                                    }

/* Set up Receive Transfer Control Block for chained DMA */
rx0_tcb[0].ii = rx0_buf[0]; /* DMA destination buffer address */
rx0_tcb[1].ii = rx0_buf[1]; /* DMA destination buffer address */
rx0_tcb[2].ii = rx0_buf[2]; /* DMA destination buffer address */

rx0_tcb[0].cp = &rx0_tcb[1].ii; /* define ptr to next TCB (point to self) */
rx0_tcb[1].cp = &rx0_tcb[2].ii; /* define ptr to next TCB (point to self) */
rx0_tcb[2].cp = &rx0_tcb[0].ii; /* define ptr to next TCB (point to self) */

SetIOP(CP0, (((int)&rx0_tcb[0].ii) & CP_MAF) | CP_PCI);
                                                                    //-----
                                                                    // Hauptprogramm
                                                                    //-----
                                                                    void main ( void )
                                                                    {
                                                                    /* GET BASE LOCATION OF DMS3 (BITSI) */
                                                                    init_ms_bases(); /* Initialize memory select bases */
                                                                    ArraysInit();

                                                                    Fenster_Create_Windowfactors ( HALBSINUS );

                                                                    sync_flag=0;
                                                                    wait();
                                                                    while(!sync_flag)
                                                                    {}
                                                                    wait();
                                                                    Baender_Calculate_WerteImBand (); // Bandwerte bestimmen

                                                                    // Startwerte setzten, so dass der Effekt von Beginn an zu wirken beginnt
                                                                    Baender_Preset_BandWerte(BandMinimalWerte,1000);
                                                                    Baender_Preset_BandWerte(BandMaximalWerte,-1000);
                                                                    }
}

//-----
// void setup_bitsibb( void )
// Initialisiert bitsibb
//-----
void setup_bitsibb( void )
{
BITSIBB_TIM0 = BB_TIM(fc);
BITSIBB_CTL = 1; /* Enable only one D/A */
BITSIBB_CLKSEL = 0x0844; /* Ch 1-2 Tim 0, Ch 3-4 Tim 1 */
}

```

```
/* set constant registers */
asm(" bit set MODE1 (SRD1H|SRD1L|SRD2H|SRD2L|SRRFH|SRRFL|SRCU);
    nop;          /*alternativer Registersatz ein*/

    m0=-1;
    b0=_deline_data;
    r2=dm(_taps);
    l0=r2;

    m5=0;
    m6=1;
    m7=-1;
    m13=0;
    m14=1;
    m15=-1;

    m8=1;
    b8=_weights;
    l8=r2;
    b9=b8;
    l9=l8;
    f7=dm(_stepsize);

    bit clr MODE1 (SRD1H|SRD1L|SRD2H|SRD2L|SRRFH|SRRFL|SRCU);
    nop;          /*alternativer Registersatz aus*/
");

wait();
setup_bitsibb();    /* initialize hardware */
wait();
setup_sport0();
wait();

sync_flag=0;
wait();
while(1)
{
// idle();
}
}
```



```

global h_dynam_max           % Maximalwert der Dynamik
global h_dynam_val          % Aktueller Wert der Dynamik
global h_dynam_text         % Beschriftung Max Dynamik
global h_absinkf_frame      % Rahmen um Absinkfaktor-Slider
global h_absinkf_slidr      % Absinkfaktor-Slider
global h_absinkf_min        % Minimalwert des Absinkfaktors
global h_absinkf_max        % Maximalwert des Absinkfaktors
global h_absinkf_val        % Aktueller Wert des Absinkfaktors
global h_absinkf_text       % Beschriftung Absinkfaktor
global h_anstiegf_frame     % Rahmen um Anstiegsfaktor-Slider
global h_anstiegf_slidr     % Anstiegsfaktor-Slider
global h_anstiegf_min       % Minimalwert des Anstiegsfaktors
global h_anstiegf_max       % Maximalwert des Anstiegsfaktors
global h_anstiegf_val       % Aktueller Wert des Anstiegsfaktors
global h_anstiegf_text     % Beschriftung Anstiegsfaktor
global h_achse              % Achse für Korrelationsgraph
global h_graph              % Korrelationsgrafik
global h_graphtitel         % Titel des Graphen
global h_marker             % Ein- und Ausschalten der Marker auf Graph
global h_menueprint         % Auswahl Print in Menüleiste
global h_menuegraph         % Auswahl Graphen in Menüleiste
global h_titelfig           % Fig. für die Eingabe eines Titels für den Graph
global h_graphgui           % Figur welche die Spektren enthält
global h_editspektrum       % Auswahl für Spektrum
global h_spektrumachse     % Achse for Grafik vom Spektrum
global h_spektrum           % Grafik mit Spektrum
global h_copyachse          % Kopie der Achse der Korrelationsgrafik
global h_copygraph          % Kopie der Korrelationsgrafik (für h_spektrum)
global h_copygraphtitel    % Titel der Kopie der Korrelationsgrafik
global h_back2adapt         % Graphen des Spektrum verlassen
global h_menueprintspek     % Auswahl Print in Menüleiste bei Graphen
global resultat            % Resultat der Korrelation
global fc_res               % Abtastfreq. mit der Korrelation gemacht wurde
global currentmode          % Aktueller Modus: mode = 1: Korrelatorfenster
                             % mode = 2: Graphenfenster
global h_parameter         % Enthät alle Händel die etwas mit der Parameter
                             % eingabe zu zun haben

%=====
darkgray = [0.5 0.5 0.5];      % Farbvektor Dunkelgrau
%darkergray = [0.45 0.45 0.45]; % Farbvektor Dunkelgrau
lightgray = [0.753 0.753 0.753]; % Farbvektor Hellgrau
green = [0.5 0.7 0.5];
red =[0.7 0.5 0.5];
white = [1 1 1];
%black = [0 0 0];
groundcol = darkgray;          % Hintergrundfarbe
%linecolor = black;           % Farben der Linien
%framecolor = white;          % Hintergrundfarbe für Verzögerungsglieder
                             % und Summen

%=====
% Kontrolle, ob Function-Argument vorhanden ist
if nargin == 0
    command = 'Initialize';
end

%=====
% Initialisierung
%=====
if strcmp(command, 'Initialize')
    % Kontrolle, ob GUI nur einmal erzeugt wurde
    gui_exist = findobj('Tag','h_adaptgui');
    if gui_exist > 0
        warndlg('Programm is already running!','Warning Message');
    else
        currentpath = which('lud.m'); % Pfad des Programms suchen
        % lud.m hinten vom Pfad entfernen
        currentpath = currentpath(1:length(currentpath)-5);
        currentmode = 1; % Aktuelles Fenster ist Korrelatorfenster
        h_parameter = [];
    end
end

```

```

%-----
% Grundfenster
%-----
    extend_x = 0.95;
    extend_y = 1.4;
    a=[extend_x extend_y extend_x extend_y];
    b=[0      1-extend_y    0    0];
    h_adaptgui = figure;
    set (h_adaptgui, 'Tag', 'h_adaptgui', ...
        'Units', 'normalized', ...
        'Color', groundcol, ...
        'PaperOrientation', 'landscape', ...
        'PaperPosition', [ 0.25 1.6 8+8/4 6+6/4 ], ...
        'NumberTitle', 'off', ...           % Figurnummer ausblenden
        'Name', 'Lärmunterdrückung LUD', ... % Tiltel des Hauptfensters
        'Position', [0.01 0.30 0.93/extend_x 0.85/extend_y], ...
        'BackingStore', 'off', ...         % Damit Animation schneller ist
        'Resize', 'off', ...
        'CloseRequestFcn', 'lud(''Programmende'')', ...
        'MenuBar', 'none');               % Standartmenü ausblenden

%-----
% UIcontrol-Objekte erzeugen
%-----
    %-----
    % Grundauswahl
    %-----
    xalignpos = - 0.03;
    yalignpos = - 0.03;

%-----
    % Erzeugen des Start-Button
    xpos = xalignpos + 0.1;
    ypos = yalignpos + 0.80;
    h_startbutton = uicontrol('Parent', h_adaptgui, ...
        'Units', 'normalize', ...
        'BackgroundColor', green, ...
        'CallBack', 'lud(''Start'')', ...
        'FontSize', 8, ...
        'Position', b+a.*[xpos ypos 0.14 0.04], ...
        'String', 'Start System', ...

        'Style', 'pushbutton', ...
        'Tag', 'Start');

% Erzeugen des Stopp-Button
    h_stoppbutton = uicontrol('Parent', h_adaptgui, ...
        'Units', 'normalize', ...
        'BackgroundColor', red, ...
        'CallBack', 'lud(''Stopp'')', ...
        'FontSize', 8, ...
        'Position', b+a.*[xpos ypos 0.14 0.04], ...
        'String', 'Stopp System', ...
        'Style', 'pushbutton', ...
        'Tag', 'Stopp');

%-----
    % Erzeugen Rahmen für System ein/aus
    xpos = xalignpos + 0.1;
    ypos = yalignpos + 0.93;
    h_korrektur_frame = uicontrol('Parent', h_adaptgui, ...
        'Units', 'normalize', ...
        'BackgroundColor', lightgray, ...
        'Position', b+a.*[xpos-0.015 ypos-0.06 0.20 0.11], ...
        'Style', 'frame', ...
        'Tag', 'Parameter');

% Erzeugen Beschriftung Adaptiv-System ein/aus
    h_adsysein_text = uicontrol('Parent', h_adaptgui, ...
        'Units', 'normalize', ...
        'BackgroundColor', lightgray, ...
        'FontSize', 8, ...
        'Position', b+a.*[xpos ypos 0.14 0.03], ...
        'HorizontalAlignment', 'left', ...
        'String', 'Adaptiv-System ein/aus :', ...
        'Style', 'text', ...
        'Tag', 'Systemein');

% Erzeugen Beschriftung Mehrkanal-System ein/aus
    ypos = ypos - 0.045;
    h_mkseysein_text = uicontrol('Parent', h_adaptgui, ...
        'Units', 'normalize', ...
        'BackgroundColor', lightgray, ...

```

```

'FontSize',8 , ...
'Position', b+a.*[xpos ypos 0.14 0.03], ...
'HorizontalAlignment', 'left', ...
'String', 'Mehrkanal-System ein/aus :', ...
'Style', 'text', ...
'Tag', 'Systemein');

% Erzeugen Checkbox Adaptiv-System ein/aus
xpos = xpos + 0.15;
ypos = ypos + 0.047;
h_adsysein_checkbox = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'CallBack', 'lud(''adSysein'')', ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.03 0.03], ...
    'Value', 1, ...
    'Style', 'checkbox', ...
    'Tag', 'adSysein');

% Erzeugen Checkbox Mehrkanal-System ein/aus
ypos = ypos - 0.045;
h_mkseyein_checkbox = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'CallBack', 'lud(''mkSysein'')', ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.03 0.03], ...
    'Value', 1, ...
    'Style', 'checkbox', ...
    'Tag', 'mkSysein');

%-----
% Erzeugen Rahmen für adaptives Systemparameter
xpos = xalignpos + 0.38;
ypos = yalignpos + 0.93;
h_parameter_frame = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'Position', b+a.*[xpos-0.015 ypos-0.06 0.22 0.11], ...
    'Style', 'frame', ...

    'Tag', 'Parameter');
h_parameter = [h_parameter, h_parameter_frame];

% Erzeugen Beschriftung Korrekturschritt
h_korrektur_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize',8 , ...
    'Position', b+a.*[xpos ypos 0.11 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Korrekturschritt :', ...
    'Style', 'text', ...
    'Tag', 'Korrekturschrittkonstante');

% Erzeugen Beschriftung Anzahl der Koeffizienten
ypos = ypos - 0.045;
h_leng_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize',8 , ...
    'Position', b+a.*[xpos ypos 0.125 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Anzahl Koeffizienten :', ...
    'Style', 'text', ...
    'Tag', 'Filterlänge');
h_parameter = [h_parameter, h_leng_text];

% Erzeugen des Korrekturschritt Auswahlfeldes
xpos = xpos + 0.135;
ypos = ypos + 0.047;
h_korrektur_edit = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'CallBack', 'lud(''Korrekturschritt'')', ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos-0.015 ypos 0.065 0.03], ...
    'String', '0.00003', ...
    'Style', 'edit', ...
    'Tag', 'Korrekturschritt');

```

```

% Erzeugen des Koeffizienten Auswahlfeldes
ypos = ypos - 0.045;
h_leng_edit = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'Callback', 'lud(''Filterlänge'')', ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.05 0.03], ...
    'String', '400', ...
    'Style', 'edit', ...
    'Tag', 'Filterlänge');
h_parameter = [h_parameter, h_leng_edit];

%-----
% Erzeugen Rahmen für DSP Auslastung
xpos = xalignpos + 0.67;
ypos = yalignpos + 0.80;
h_auslastung_frame = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'Position', b+a.*[xpos-0.015 ypos 0.175 0.04], ...
    'Style', 'frame', ...
    'Tag', 'Parameter');

% Erzeugen Beschriftung der Auslastung
ypos = ypos + 0.002;
h_auslastung_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.11 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Auslastung DSP [%] :', ...
    'Style', 'text', ...
    'Tag', 'Auslastung');

% Erzeugen der DSP-Auslastungsanzeige
xpos = xpos + 0.1;
ypos = ypos + 0.009;
h_auslastung_show = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.05 0.02], ...
    'String', '0', ...
    'Style', 'text', ...
    'Tag', 'Auslastung');

%-----
% Erzeugen Rahmen für Übersteuerungsanzeige
xpos = xalignpos + 0.67;
ypos = yalignpos + 0.93;
h_uebersteuert_frame = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'Position', b+a.*[xpos-0.015 ypos-0.06 0.175 0.11], ...
    'Style', 'frame', ...
    'Tag', 'Übersteuert');

% Erzeugen Beschriftung Channel 1 übersteuert
h_uebersteuert1_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.115 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Channel 1 übersteuert', ...
    'Style', 'text', ...
    'Tag', 'übersteuert1');

% Erzeugen Beschriftung Channel 2 übersteuert
ypos = ypos - 0.045;
h_uebersteuert2_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.115 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Channel 2 übersteuert', ...
    'Style', 'text', ...
    'Tag', 'übersteuert2');

```

```

% Erzeugen Channel 1 übersteuert Anzeige
xpos = xpos + 0.125;
ypos = ypos + 0.055;
h_uebersteuert1_show = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'Position', b+a.*[xpos ypos 0.015 0.02], ...
    'Style', 'frame', ...
    'Tag', 'übersteuert1');

% Erzeugen Channel 2 übersteuert Anzeige
ypos = ypos - 0.045;
h_uebersteuert2_show = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'Position', b+a.*[xpos ypos 0.015 0.02], ...
    'Style', 'frame', ...
    'Tag', 'übersteuert2');

%-----
% Erzeugen des Rahmen für max Dämpfung-Slider
xpos = xalignnpos + 0.9;
ypos = yalignnpos + 0.93;
h_daempfung_frame = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'Position', b+a.*[xpos-0.015 ypos-0.06 0.180 0.11], ...
    'Style', 'frame', ...
    'Tag', 'Maximale Dämpfung');

% Erzeugen des max Dämpfung-Sliders
h_daempfung_slldr = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'callback','lud('Daempfung Slider Moved')',...
    'style','slider',...
    'fontsize', 6,...
    'value', 40,... % Anfangswert
    'min',1,'max',50,...
    'SliderStep', [1/49 5/49],...
    'position', b+a.*[xpos ypos-.02 0.150 0.02]);

h_daempfung_min = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'style','text',...
    'string',num2str(get(h_daempfung_slldr,'min')),...
    'position', b+a.*[xpos-0.005 ypos+0.012 0.025 0.02]);
h_daempfung_max = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'style','text',...
    'string',num2str(get(h_daempfung_slldr,'max')),...
    'position', b+a.*[xpos+0.127 ypos+0.012 0.025 0.02]);
h_daempfung_val = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'callback','lud('Daempfung Change Value')',...
    'style','edit',...
    'fontsize', 6,...
    'string',num2str(get(h_daempfung_slldr,'value')),...
    'position', b+a.*[xpos+.054 ypos+0.01 0.04 0.027]);
ypos = ypos - 0.045;

% Erzeugung Beschriftung max Dämpfung-Slider
h_daempfung_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize',6 , ...
    'Position', b+a.*[xpos+0.014 ypos-0.01 0.13 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Maximale Dämpfung [dB]', ...
    'Style', 'text', ...
    'Tag', 'MaxDämpfung');

%-----
% Erzeugen des Rahmen für max Dynamik-Slider
xpos = xalignnpos + 0.9;
ypos = yalignnpos + 0.79;
h_dynamik_frame = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'Position', b+a.*[xpos-0.015 ypos-0.06 0.180 0.11], ...

```

```

'Style', 'frame', ...
'Tag', 'Maximale Dynamik');

% Erzeugen des max Dynamik-Sliders
h_dynam_sldr = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'callback','lud(''Dynam Slider Moved'')',...
    'style','slider',...
    'fontsize', 6,...
    'value', 50,... % Anfangswert
    'min',1,'max',50,...
    'SliderStep', [1/49 5/49],...
    'position', b+a.*[xpos ypos-.02 0.150 0.02]);
h_dynam_min = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'style','text',...
    'string',num2str(get(h_dynam_sldr,'min')),...
    'position', b+a.*[xpos-0.005 ypos+0.012 0.025 0.02]);
h_dynam_max = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'style','text',...
    'string',num2str(get(h_dynam_sldr,'max')),...
    'position', b+a.*[xpos+0.127 ypos+0.012 0.025 0.02]);
h_dynam_val = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'callback','lud(''Dynam Change Value'')',...
    'style','edit',...
    'fontsize', 6,...
    'string',num2str(get(h_dynam_sldr,'value')),...
    'position', b+a.*[xpos+.054 ypos+0.01 0.04 0.027]);
ypos = ypos - 0.045;

% Erzeugung Beschriftung max Dynamik-Slider
h_dynam_text = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize',6 , ...
    'Position', b+a.*[xpos+0.018 ypos-0.01 0.13 0.03], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Maximale Dynamik [dB]', ...
    'Style', 'text', ...
    'Tag', 'MaxDynam');

%-----
% Erzeugen des Rahmen für Absinkfaktor-Slider
xpos = xalignpos + 0.9;
ypos = yalignpos + 0.65;
h_absinkf_frame = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'Position', b+a.*[xpos-0.015 ypos-0.06 0.180 0.11], ...
    'Style', 'frame', ...
    'Tag', 'Absinkfaktor');

% Erzeugen des Absinkfaktor-Sliders
h_absinkf_sldr = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'callback','lud(''Absinkf Slider Moved'')',...
    'style','slider',...
    'fontsize', 6,...
    'value', 0.97,... % Anfangswert
    'min',0.9,'max',0.999,...
    'SliderStep', [1/99 10/99],...
    'position', b+a.*[xpos ypos-.02 0.150 0.02]);
h_absinkf_min = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'style','text',...
    'string',num2str(get(h_absinkf_sldr,'min')),...
    'position', b+a.*[xpos-0.005 ypos+0.012 0.025 0.02]);
h_absinkf_max = uicontrol('parent', h_adaptgui,...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'style','text',...
    'string',num2str(get(h_absinkf_sldr,'max')),...
    'position', b+a.*[xpos+0.125 ypos+0.012 0.03 0.02]);
h_absinkf_val = uicontrol('parent', h_adaptgui,...

```

```

'Units', 'normalize', ...
'BackgroundColor', white, ...
'callback','lud('Absinkf Change Value')',...
'style','edit',...
'fontsize', 6,...
'string',num2str(get(h_absinkf_sldr,'value')),...
'position', b+a.*[xpos+.054 ypos+0.01 0.04 0.027]);
ypos = ypos - 0.045;

% Erzeugung Beschriftung Absinkfaktor-Slider
h_absinkf_text = uicontrol('Parent', h_adaptgui, ...
'Units', 'normalize', ...
'BackgroundColor', lightgray, ...
'FontSize',6 , ...
'Position', b+a.*[xpos+0.045 ypos-0.01 0.1 0.03], ...
'HorizontalAlignment', 'left', ...
'String', 'Absinkfaktor', ...
'Style', 'text', ...
'Tag', 'Absinkfak');

%-----
% Erzeugen des Rahmen für Anstiegsfaktor-Slider
xpos = xalignpos + 0.9;
ypos = yalignpos + 0.51;
h_anstiegf_frame = uicontrol('parent', h_adaptgui,...
'Units', 'normalize', ...
'BackgroundColor', lightgray, ...
'Position', b+a.*[xpos-0.015 ypos-0.06 0.180 0.11], ...
'Style', 'frame', ...
'Tag', 'Anstiegsfaktor');

% Erzeugen des Anstiegsfaktor-Sliders
h_anstiegf_sldr = uicontrol('parent', h_adaptgui,...
'Units', 'normalize', ...
'BackgroundColor', lightgray, ...
'callback','lud('Anstiegf Slider Moved')',...
'style','slider',...
'fontsize', 6,...
'value', 1.03,... % Anfangswert
'min',1.001,'max',1.1,...
'SliderStep', [1/99 10/99],...

'position', b+a.*[xpos ypos-.02 0.150 0.02]);
h_anstiegf_min = uicontrol('parent', h_adaptgui,...
'Units', 'normalize', ...
'BackgroundColor', lightgray, ...
'style','text',...
'string',num2str(get(h_anstiegf_sldr,'min')),...
'position', b+a.*[xpos-0.005 ypos+0.012 0.025 0.02]);
h_anstiegf_max = uicontrol('parent', h_adaptgui,...
'Units', 'normalize', ...
'BackgroundColor', lightgray, ...
'style','text',...
'string',num2str(get(h_anstiegf_sldr,'max')),...
'position', b+a.*[xpos+0.125 ypos+0.012 0.03 0.02]);
h_anstiegf_val = uicontrol('parent', h_adaptgui,...
'Units', 'normalize', ...
'BackgroundColor', white, ...
'callback','lud('Anstiegf Change Value')',...
'style','edit',...
'fontsize', 6,...
'string',num2str(get(h_anstiegf_sldr,'value')),...
'position', b+a.*[xpos+.054 ypos+0.01 0.04 0.027]);
ypos = ypos - 0.045;

% Erzeugung Beschriftung Anstiegsfaktor-Slider
h_anstiegf_text = uicontrol('Parent', h_adaptgui, ...
'Units', 'normalize', ...
'BackgroundColor', lightgray, ...
'FontSize',6 , ...
'Position', b+a.*[xpos+0.042 ypos-0.01 0.1 0.03], ...
'HorizontalAlignment', 'left', ...
'String', 'Anstiegsfaktor', ...
'Style', 'text', ...
'Tag', 'Anstiegfak');

%-----
% Achsen für Filterkoeffizientengragh
xpos = xalignpos + 0.09;
ypos = yalignpos + 0.4;
h_achse = axes('Parent', h_adaptgui, ...
'Visible', 'on', ...
'Position', b+a.*[xpos ypos 0.74 0.35], ...

```

```

        'DrawMode', 'fast');
h_graph = plot([0 0]);
h_graphtitel = title('');
set(h_graphtitel, 'Visible', 'off', ... % Nur zum Drucken
    'FontSize', 12);
xlabel('Koeffizientennummer [s]');
ylabel('Koeffizienten');
grid;

% Erzeugen der Auswahl der Marker auf Graph
xpos = xalignpos + 0.12;
ypos = yalignpos + 0.76;
xpos = xalignpos + 0.095;
ypos = yalignpos + 0.34;
h_marker = uicontrol('Parent', h_adaptgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', darkgray, ...
    'Callback', 'lud(''Marker'')', ...
    'FontSize', 8, ...
    'Position', b+a.*[xpos ypos 0.07 0.03], ...
    'String', 'Marker', ...
    'Style', 'checkbox', ...
    'Value', 0, ...
    'Tag', 'marker');

%-----
% Erzeugen der Menüleiste
h_menueprint = uimenu('Parent', h_adaptgui, ...
    'label', '&File');
h_menuegraph = uimenu('Parent', h_adaptgui, ...
    'label', '&Spektrum');
h_parameter = [h_parameter, h_menueprint, h_menuegraph];

% Erzeugen von Print in der Menüleiste
h_print = uimenu(h_menueprint, ...
    'label', '&Print Koeffizienten', ...
    'Callback', 'lud(''Print'')');

% Erzeugen von Graph-Auswahl in der Menüleiste
h_editspektrum = uimenu(h_menuegraph, ...
    'label', '&Spektrum', ...

        'CallBack', 'lud(''Spektrum'')');

% Grundfenster für Spektren
h_graphgui = figure;
set(h_graphgui, 'Tag', 'h_graphgui', ...
    'Units', 'normalized', ...
    'Color', groundcol, ...
    'PaperPosition', [0.5 0.4 7.7 10.12], ...
    'NumberTitle', 'off', ... % Figurnummer ausblenden
    'Name', 'Spektrum', ... % Tiltel des Hauptfensters
    'Position', [0.05 0.147 0.809 0.76], ...
    'CloseRequestFcn', 'lud(''Spektrumverlassen'')', ...
    'MenuBar', 'none'); % Standartmenü ausblenden
h_spektrumachse = axes('Parent', h_graphgui, ...
    'Position', [0 0 0.74 0.35]);
h_back2adapt = uicontrol('Parent', h_graphgui, ...
    'Units', 'normalize', ...
    'BackgroundColor', darkgray, ...
    'Callback', 'lud(''Spektrumverlassen'')', ...
    'FontSize', 8, ...
    'Position', [0.05 0.93 0.13 0.04], ...
    'String', 'Schliesse Graphen', ...
    'Style', 'pushbutton');

% Erzeugen der Menüleiste
h_menueprintspek = uimenu('Parent', h_graphgui, ...
    'label', '&File');

% Erzeugen von Print in der Menüleiste
h_printspek = uimenu(h_menueprintspek, ...
    'label', '&Print Graph', ...
    'Callback', 'lud(''Print'')');

set(h_stoppbutton, 'Visible', 'off'); % Stoppbutton ausblenden
set(h_graphgui, 'Visible', 'off'); % Figur mit Spektren ausblenden

end

```

```

=====
% Callbacks
=====
%-----
% Callback für Korrekturschritt eingeben
%-----
elseif strcmp(command, 'Korrekturschritt')
    kors = str2num(get(h_korrektur_edit, 'String'));
    if isempty(kors),
        set(h_korrektur_edit, 'String', num2str(0.0003));
    elseif kors < 0.00000001,
        set(h_korrektur_edit, 'String', num2str(0.00000001));
    elseif kors > 1,
        set(h_korrektur_edit, 'String', num2str(1));
    end
    if running == 1,
        korr_res = str2num(get(h_korrektur_edit, 'String')); % Korrekturschritt
        sharc('dlflt', 'stepsize', korr_res); % Korrekturschritt auf DSP laden
    end

%-----
% Callback für Adaptiv-System ein/aus anwählen
%-----
elseif strcmp(command, 'adSysein')
    if running == 1,
        if get(h_adysein_checkbox, 'Value'),
            sharc('dl_int', 'adapon', 1);
        else
            sharc('dl_int', 'adapon', 0);
        end
    end

%-----
% Callback für Mehrkanal-System ein/aus anwählen
%-----
elseif strcmp(command, 'mkSysein')
    if running == 1,
        if get(h_mkseysein_checkbox, 'Value'),
            sharc('dl_int', 'mkon', 1);
        else
            sharc('dl_int', 'mkon', 0);
    end
end

```

```

end
end

%-----
% Callback für Anzahl der Filterkoeffizienten eingeben
%-----
elseif strcmp(command, 'Filterlänge')
    N = str2num(get(h_leng_edit, 'String'));
    if isempty(N),
        set(h_leng_edit, 'String', num2str(420));
    elseif N < 10,
        set(h_leng_edit, 'String', num2str(10));
    elseif N > 5000,
        set(h_leng_edit, 'String', num2str(5000));
    end

%-----
% Callback für maximale Dämpfung-Slider
%-----
elseif strcmp(command, 'Daemp Change Value')
    user_value = str2num(get(h_daemp_val, 'string'));
    if ~length(user_value)
        user_value = (get(h_daemp_sldr, 'max')+get(h_daemp_sldr, 'min'))/2;
    end
    user_value = min([user_value get(h_daemp_sldr, 'max')]);
    user_value = max([user_value get(h_daemp_sldr, 'min')]);
    set(h_daemp_sldr, 'value', user_value);
    set(h_daemp_val, 'string', num2str(get(h_daemp_sldr, 'value')));
    if running == 1,
        daemp = str2num(get(h_daemp_val, 'String'));
        sharc('dl_int', 'MaxDaempfung', daemp);
    end

elseif strcmp(command, 'Daemp Slider Moved')
    set(h_daemp_val, 'string', num2str(get(h_daemp_sldr, 'value')));
    if running == 1,
        daemp = str2num(get(h_daemp_val, 'String'));
        sharc('dl_int', 'MaxDaempfung', daemp);
    end
end

```

```

%-----
% Callback für maximale Dynamik-Slider
%-----
elseif strcmp(command, 'Dynam Change Value')
    user_value = str2num(get(h_dynam_val, 'string'));
    if ~length(user_value)
        user_value = (get(h_dynam_sldr, 'max')+get(h_dynam_sldr, 'min'))/2;
    end
    user_value = min([user_value get(h_dynam_sldr, 'max')]);
    user_value = max([user_value get(h_dynam_sldr, 'min')]);
    set(h_dynam_sldr, 'value', user_value);
    set(h_dynam_val, 'string', num2str(get(h_dynam_sldr, 'value')));
    if running == 1,
        dynam = str2num(get(h_dynam_val, 'String'));
        sharc('dl_int', 'MaxDynamik', dynam);
    end
elseif strcmp(command, 'Dynam Slider Moved')
    set(h_dynam_val, 'string', num2str(get(h_dynam_sldr, 'value')));
    if running == 1,
        dynam = str2num(get(h_dynam_val, 'String'));
        sharc('dl_int', 'MaxDynamik', dynam);
    end

%-----
% Callback für Absinkfaktor-Slider
%-----
elseif strcmp(command, 'Absinkf Change Value')
    user_value = str2num(get(h_absinkf_val, 'string'));
    if ~length(user_value)
        user_value = (get(h_absinkf_sldr, 'max')+get(h_absinkf_sldr, 'min'))/2;
    end
    user_value = min([user_value get(h_absinkf_sldr, 'max')]);
    user_value = max([user_value get(h_absinkf_sldr, 'min')]);
    set(h_absinkf_sldr, 'value', user_value);
    set(h_absinkf_val, 'string', num2str(get(h_absinkf_sldr, 'value')));
    if running == 1,
        absinkf = str2num(get(h_absinkf_val, 'String'));
        sharc('dlflt', 'Absinkfaktor', absinkf);
    end
elseif strcmp(command, 'Absinkf Slider Moved')
    set(h_absinkf_val, 'string', num2str(get(h_absinkf_sldr, 'value')));

    if running == 1,
        absinkf = str2num(get(h_absinkf_val, 'String'));
        sharc('dlflt', 'Absinkfaktor', absinkf);
    end

%-----
% Callback für Anstiegsfaktor-Slider
%-----
elseif strcmp(command, 'Anstiegf Change Value')
    user_value = str2num(get(h_anstiegf_val, 'string'));
    if ~length(user_value)
        user_value = (get(h_anstiegf_sldr, 'max')+get(h_anstiegf_sldr, 'min'))/2;
    end
    user_value = min([user_value get(h_anstiegf_sldr, 'max')]);
    user_value = max([user_value get(h_anstiegf_sldr, 'min')]);
    set(h_anstiegf_sldr, 'value', user_value);
    set(h_anstiegf_val, 'string', num2str(get(h_anstiegf_sldr, 'value')));
    if running == 1,
        anstiegf = str2num(get(h_anstiegf_val, 'String'));
        sharc('dlflt', 'Anstiegsfaktor', anstiegf);
    end
elseif strcmp(command, 'Anstiegf Slider Moved')
    set(h_anstiegf_val, 'string', num2str(get(h_anstiegf_sldr, 'value')));
    if running == 1,
        anstiegf = str2num(get(h_anstiegf_val, 'String'));
        sharc('dlflt', 'Anstiegsfaktor', anstiegf);
    end

%-----
% Callback für Start anwählen
%-----
elseif strcmp(command, 'Start')
    oldpath = cd; % aktueller Pfad Speichern
    cd(currentpath); % Pfad auf dieses Programm setzen
    fc_res = 22050; % Abtastfrequenz

    % Eingaben sperren
    set(h_parameter, 'Enable', 'off');

    % Parameter setzen
    korr_res = str2num(get(h_korrektur_edit, 'String')); % Korrekturschritt

```

```

adsysein = get(h_adsysein_checkbox, 'Value'); % Adapt-System ein/aus
mksysein = get(h_mksysein_checkbox, 'Value'); % MK-System ein/aus
N = str2num(get(h_leng_edit, 'String')) + 1; % Filterlänge
daempfung = str2num(get(h_daempfung_val, 'String')); % Max Dämpfung
dynamik = str2num(get(h_dynamik_val, 'String')); % Max Dynamik
absinkfaktor = str2num(get(h_absinkfaktor_val, 'String')); % Absinkfaktor
anstiegfaktor = str2num(get(h_anstiegfaktor_val, 'String')); % Anstiegfaktor

% Graph vorbereiten
set(h_graph, 'XData', (0:N-1), 'YData', zeros(1,N));

% Die Achsen werden nicht jedesmal neu gezeichnet, macht Animation schneller
set(h_graph, 'erasemode', 'xor');

% DSP-Board initialisieren
stat= sharc('open',0); % Öffnet Verb. zu DSP-Board
stat= [stat char(13) sharc('reset')]; % Hardreset
stat= [stat char(13) sharc('cfg_proc')]; % Prozessor konfigurieren
stat= [stat char(13) sharc('dl_exe','lud.21k')]; % Prog. 'lud.21k' laden
stat= [stat char(13) sharc('start')]; % Programm auf DSP starten

% Warten, bis DSP bereit zum Parameter laden
while sharc('ul_int', 'sync_flag'), end

% Parameter auf DSP laden
sharc('dl_flt', 'stepsize', korr_res); % Korrekturschritt auf DSP laden
sharc('dl_int', 'adapon', adsysein); % Adaptiv-System ein/aus
sharc('dl_int', 'mkon', mksysein); % MK-System ein/aus
sharc('dl_int', 'taps', N); % Filterlänge auf DSP laden
sharc('dl_int', 'MaxDaempfung', daempfung); % max Dämpfung auf DSP laden
sharc('dl_int', 'MaxDynamik', dynamik); % max Dynamik auf DSP laden
sharc('dl_flt', 'Absinkfaktor', absinkfaktor); % Absinkfaktor auf DSP laden
sharc('dl_flt', 'Anstiegfaktor', anstiegfaktor); % Anstiegfaktor auf DSP laden

% DSP melden, dass alle Para. geladen
sharc('dl_int', 'sync_flag', 1);

% Warten bis Korrelation auf DSP gestartet
while sharc('ul_int', 'sync_flag'), end
set(h_startbutton, 'Visible', 'off'); % Startbutton unsichtbar machen
set(h_stoppbutton, 'Visible', 'on'); % Stoppbutton sichtbar machen

% Variablen initialisieren
running = 1;
zaehler = 10;
ymin = 0;
ymax = 0;
while running,
    koef = sharc('ul_flts','weights',N); % Filerkoeffizienten laden
    currentinfo = sharc('ul_ints','currentinfo',2);
    % currentinfo[0]: current input value channel 2
    % currentinfo[1]: current input value channel 1

% Resultat der Korrelation
set(h_graph, 'YData', koef);

% Anzahl Zyklen für System
clocks = sharc('ul_int','clocks');

% Zeit für System
% 100 : ca. Anzahl Zyklen für Interruptaufruf und Zeitmessung selbst
% 256 : Blockgrösse der eingelesenen Werte
% 40e6 = 40Mhz = Taktfrequenz DSP
clocks = clocks + 100;

% Auslastung des Prozes. (in %) berechnen
auslast = clocks * fc_res / (40e6 * 256) * 100;

% % Auslastung anzeigen
set(h_auslastung_show, 'String', num2str(auslast, 3));

% Überprüfen, ob Eingänge übersteuern
if (currentinfo(2) < -32700) | (currentinfo(2) > 32700),
    set(h_uebersteuert1_show, 'BackgroundColor', 'red');
else
    set(h_uebersteuert1_show, 'BackgroundColor', white);
end
if (currentinfo(1) < -32700) | (currentinfo(1) > 32700),
    set(h_uebersteuert2_show, 'BackgroundColor', 'red');
else
    set(h_uebersteuert2_show, 'BackgroundColor', white);
end

```

```

% Zwischendurch prüfen, ob Achsen des Graphen noch in Bereich des Resul.
if zaehler > 20,
    % Achsen neu zeichnen
    minres = min(koef); % Minimaler Wert suchen
    maxres = max(koef); % Maximaler Wert suchen
    % Ist es nötig Achsen anzupassen ?
    if (minres < ymin*0.95) | (minres > 0.8*ymin) ...
        | (maxres > ymax*0.95) | (maxres < 0.8*ymax),
        ymin = minres-0.15*abs(minres); % Neues Minimum der y-Achse
        ymax = maxres+0.15*abs(maxres); % Neues Maximum der y-Achse
        if (abs(ymin) == NaN) | (abs(ymax) == NaN),
            ymin = -0.1; % Absichern falls System instabil
            ymax = 0.1;
        end
        if ymin == ymax, % Absichern falls ymin == ymax
            ymin = -0.1;
            ymax = 0.1;
        end
        set(h_achse, 'YLim', [ymin ymax]); % Neue Werte setzen
    end
    zaehler = 0;
else
    zaehler = zaehler + 1;
end
drawnow; % Änderungen zeichnen
end % Ende der Korrelationsschleufe
resultat = koef;

% Achsen optimal setzen
minres = min(resultat); % Minimaler Wert suchen
maxres = max(resultat); % Maximaler Wert suchen
ymin = minres-0.15*abs(minres); % Neues Minimum der y-Achse
ymax = maxres+0.15*abs(maxres); % Neues Maximum der x-Achse
if (abs(ymin) == NaN) | (abs(ymax) == NaN),
    ymin = -0.1; % Absichern falls System instabil
    ymax = 0.1;
end
if ymin == ymax, % Absichern falls ymin == ymax
    ymin = -0.1;
    ymax = 0.1;
end

set(h_achse, 'YLim', [ymin ymax]); % Neue Werte setzen
set(h_graph, 'erasemode', 'normal');
set(h_auslastung_show, 'String', num2str(0)); % Auslastung auf Null setzen
set(h_parameter, 'Enable', 'on'); % Eingaben freigeben
set(h_uebersteuert2_text, 'Enable', 'on');
set(h_stoppbutton, 'Visible', 'off'); % Stoppbutton unsichtbar machen
set(h_startbutton, 'Visible', 'on'); % Startbutton sichtbar machen
cd(oldpath); % Auf alten Pfad wechseln

%-----
% Callback für Stopp anwählen
%-----
elseif strcmp(command, 'Stopp')
    running = 0;

%-----
% Callback für Marker anwählen
%-----
elseif strcmp(command, 'Marker')
    if get(h_marker, 'Value'),
        set(h_graph, 'marker', 'o');
    else
        set(h_graph, 'marker', 'none');
    end

%-----
% Callback für Spektrum anwählen
%-----
elseif strcmp(command, 'Spektrum') ...
    currentmode = 2; % Aktuelles Fenster ist Graphfenster
    set(h_adaptgui, 'Visible', 'off'); % Figur mit Korrelator ausblenden
    set(h_graphgui, 'Visible', 'on'); % Figur mit Spektren einblenden

    % Korrelationsfigur kopieren
    h_copyachse = copyobj(h_achse, h_graphgui);
    set(h_copyachse, 'Position', [0.1 0.56 0.83 0.33]);
    set(h_spektrumachse, 'Position', [0.1 0.09 0.83 0.33]);
    h_copygraph = get(h_copyachse, 'Children');
    h_copygraphtitel = copyobj(h_graphtitel, h_copyachse);
    resultatleng = length(resultat);
    zeroNr = 2*4096-resultatleng;

```

```

resleng = 2*resultatleng-1;

% Frequenzachse skalieren
f = linspace(0, fc_res, resultatleng+zeroNr);

% Spektrum berechnen (zero padding)
spektrum = 1/fc_res*fft([resultat, zeros(1, zeroNr)]);

% Achse für Spektrum zur aktuellen Achse ändern
axes(h_spektrumachse);

% Sinnvolle Länge des Spektrums bestimmen
n = floor((resultatleng + zeroNr)/2);
maxval = max(abs(spektrum(1:n)));
while abs(spektrum(n)) < (0.005 * maxval),
    n=n-1;
end
n=n+1;
spektrum = spektrum(1:n);
f = f(1:n);

% Spektrum zeichnen
h_spektrum = plot(f, abs(spektrum));
axis([0 f(n) 0 1.1*max(abs(spektrum))]);
grid;
xlabel('Frequenz [Hz]');
ylabel('Betrag');
if strcmp(command, 'Spektrum'),
    spektitel = 'Spektrum';
end
h_spektitel = title(spektitel);
set(h_spektitel, 'FontSize', 12);

%-----
% Callback für Schliessen des Spektrums anwählen
%-----
elseif strcmp(command, 'Spektrumverlassen')
    set(h_adaptgui, 'Visible', 'on');    % Figur mit Korrelator einblenden
    set(h_graphgui, 'Visible', 'off');  % Figur mit Spektren ausblenden
    delete(h_copyachse);
    currentmode = 1;                  % Aktuelles Fenster ist Korrelator

```

```

%-----
% Callback für Print Graph anwählen
%-----
elseif strcmp(command, 'Print')
    if currentmode == 1,
        set(h_menueprint, 'Enable', 'off');
    elseif currentmode == 2,
        set(h_menueprintspek, 'Enable', 'off');
    end

% Erzeugt Fenster, um Titel für Graph einzugeben
h_titelfig = figure;
set (h_titelfig, ...
    'Units', 'normalized', ...
    'Color', lightgray, ...
    'NumberTitle', 'off', ...           % Figurnummer ausblenden
    'Name', 'Titel', ...              % Tiltel des Hauptfensters
    'Position', [0.2 0.75 0.5 0.1], ...
    'CloseRequestFcn', 'lud(''CloseTitelfig'')', ...
    'Resize', 'off', ...
    'WindowStyle', 'modal', ...
    'MenuBar', 'none');                % Standartmenü ausblenden

% Erzeugt Eingabeaufforderung
h_aufforderung = uicontrol('Parent', h_titelfig, ...
    'Units', 'normalize', ...
    'BackgroundColor', lightgray, ...
    'FontSize', 10, ...
    'Position', [0.02 0.48 0.8 0.4], ...
    'HorizontalAlignment', 'left', ...
    'String', 'Titel für den Graphen eingeben :', ...
    'Style', 'text');

```

```

% Erzeugen des Eingabefelds für den Titel
h_titel = uicontrol('Parent', h_titelfig, ...
    'Units', 'normalize', ...
    'BackgroundColor', white, ...
    'Callback', 'lud(''closetitle'')', ...
    'FontSize', 10, ...
    'Position', [0.02 0.22 0.81 0.31], ...
    'HorizontalAlignment', 'left', ...
    'Style', 'edit');

% Erzeugen des Ok-Button
h_titel_ok = uicontrol('Parent', h_titelfig, ...
    'Units', 'normalize', ...
    'BackgroundColor', darkgray, ...
    'Callback', 'lud(''closetitle'')', ...
    'FontSize', 10, ...
    'Position', [0.86 0.22 0.12 0.31], ...
    'String', 'OK', ...
    'Style', 'pushbutton');

%-----
% Callback für Titelfigure (vom Drucken) schliessen
%-----
elseif strcmp(command, 'CloseTitelfig')
delete(get(0,'CurrentFigure'))
    if currentmode == 1,
        set(h_graphtitel, 'Visible', 'off');
set(h_menueprint, 'Enable', 'on');
    elseif currentmode == 2,
        set(h_copygraphtitel, 'Visible', 'off');
        set(h_menueprintspek, 'Enable', 'on');
    end
drawnow;

%-----
% Callback für Titeleingabe des Graphen fertig
%-----
elseif strcmp(command, 'closetitle')
    h_titel = findobj(h_titelfig, 'Style', 'edit');
    neuertitel = get(h_titel, 'String');
    if currentmode == 1,
        set(h_graphtitel, ...
            'String', neuertitel, ...
            'Visible', 'on');
    elseif currentmode == 2,
        set(h_copygraphtitel, ...
            'String', neuertitel, ...
            'Visible', 'on');
    end
delete(h_titelfig); % Figur zur Eingabe des Titels löschen
print -noui;
    if currentmode == 1,
        set(h_graphtitel, 'Visible', 'off');
set(h_menueprint, 'Enable', 'on');
    elseif currentmode == 2,
        set(h_copygraphtitel, 'Visible', 'off');
        set(h_menueprintspek, 'Enable', 'on');
    end
drawnow;

%-----
% Callback für beenden des Programms
%-----
elseif strcmp(command, 'Programmende')
    delete(h_graphgui);
    delete(get(0,'CurrentFigure'))
end

%=====
% Zusätzliche Funktionen
%=====

```