TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

Analyse der Implementierbarkeit einer aktiven Schallunterdrückungslösung in Hardware

Ausgeführt am Institut für Technische Informatik

Embedded Computing Systems
E182-2

der Technischen Universität Wien

unter der Anleitung von
Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Andreas Steiniger

durch

Christian Varga

Csokorgasse 62/1/3, 1110 Wien

28. August 2006
Datum

Unterschrift (Student)

Kurzdarstellung

Durch die enormen Fortschritte der Halbleitertechnik sind aktive Schallunterdrückungsmethoden ökonomisch realisierbar geworden. Durch die gesteigerte Leistungsfähigkeit verfügbarer Mikroprozessoren und dem Einsatz von dedizierter Hardware – wie zum Beispiel FPGAs – ist es möglich geworden flexible und portable Systeme zur aktiven Dämpfung von Schallwellen zu entwickeln.

Während jedoch in vielen Abhandlungen nur die Theorie eine Rolle spielt, soll diese Arbeit dazu dienen, die praktischen Aspekte einer Realisierung zu untersuchen.

Zu diesem Zweck sollen die Algorithmen der LMS (*least mean square*) Familie untersucht werden, um eine Aussage über die Güte einer erreichbaren Schalldämpfung zu erhalten. Es soll weiters die Auswirkung der zugrunde liegenden Arithmetik unter die Lupe genommen werden. Diese Notwendigkeit resultiert aus der Komplexität der Implementierung einer Gleitkommalösung. Dazu werden Filter mit Gleitkomma- und Fixkommaarithmetik in Hinsicht auf Konvergenz verglichen.

Nachdem gezeigt wurde, daß der Einsatz dieser Algorithmen durchaus vertretbar ist, soll ein weiterer Schritt in Richtung Hardware gesetzt werden. Dabei wird der Aufbau schematisch dargestellt und die einzelnen Komponenten auf ihr Übertragungsverhalten hin überprüft. Durch diese Daten und die Erfahrungen aus den Simulationen, kann nun eine Aussage über die Realisierung getroffen werden.

Im Laufe der Untersuchung werden Grenzen aufgezeigt, die besonders das Konvergenzverhalten und die Arten von Schall, welche bedämpft werden können, betreffen. Es ist im Laufe der Arbeit bestätigt worden, daß es *nur theoretisch* möglich ist *beliebigen* Schall zu eliminieren und, daß die Wahl der Parameter des Filters von entscheidender Bedeutung ist.

Abstract

The enormous progress in the field of semiconductor technology made active active noise cancellation (ANC) techniques economically feasible. The improved performance of modern microprocessors, and the usage of dedicated hardware – like FPGAs – provide techniques for the development of flexible and portable systems to fight unwanted acoustic noise.

Many papers only focus on the theoretical aspects of the problem. This work is intended to lighten up the practical issues that have to be faced, when considering a practical realisation of an ANC system.

For this reason, the widely known LMS (*least mean square*) algorithm and algorithms based on LMS will be investigated. The goal is to show what can be reached in the field of ANC. Furthermore the effect of the arithmetic operations needed to perform any kind of digital filtering will be examined. This has to be done because of the fact, that floating point solutions need much more processing power (and hardware), than their fixed point counterparts. The most interesting thing here is to compare the speed of convergence of the algorithm in both cases and to analyse the effects of numerical problems on the algorithm's behaviour.

After proving that fixed point arithmetics suffice to solve the given problem to full content, another step towards hardware realisation will be taken. By analysing the impact of the transfer functions of the different components involved, a statement on convergence of the solution as a whole can be made.

During this enquiry borders are discovered, which have a severe limiting impact on the possibilities of ANC. Especially the speed of convergence of the algorithm and the phase shift of the analogue component transfer paths tend to be problematic. It has been affirmed, that the cancellation of *arbitrary* noise can only be done *in theory*, and that the selection of the filter parameters is of major importance.

Danksagungen

In erster Linie danke ich meinem Betreuer, Prof. Andreas Steininger, welcher mich besonders auf dem Gebiet der Elektrotechnik und Messtechnik geduldig unterstützt hat und mir wertvolle Anregungen gab.

Weiters möchte ich Peter Tummeltshammer danken, welcher mir bei meinem Einstieg in VHDL geholfen hat und mir Hilfestellung, bei der Verwendung der benötigten Programme, zur Seite gestanden ist.

Ich danke außerdem Gottfried Fuchs, der mir ebenfalls, beim Aufbau der Testschaltungen, wichtige Tips und Ratschläge gegeben hat.

Mein Dank gilt auch dem Institutspersonal, insbesondere Edeltraud Sommer und Thomas Handl, welche mich bei Fragen und Problemen immer tatkräftig unterstützt haben.

Ich möchte an dieser Stelle, allen Personen, die an der Erstellung dieser Arbeit beteiligt waren und sich nicht hier finden ebenfalls meinen herzlichen Dank aussprechen.

Christian VARGA
Wien, im August 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Die Auswirkung von schädlichem, beziehungsweise unerwünschtem Schall, in weiterer Folge *Lärm* genannt, auf den Menschen sind mittlerweile in das Blickfeld der Forschung und der Medien gerückt. Besonders laute Schallquellen, wie Verkehrslärm und Fluglärm [1] [2], stehen dabei im Vordergrund. Doch auch Lärm im Umfeld des Arbeitsplatzes stellt ein hohes Risiko für die Gesundheit dar, da hier der Mensch dem Lärm für einen langen Zeitraum ausgesetzt ist.

Der Grundtenor der Ergebnisse zeigt dabei, daß Lärm eine ernsthafte Bedrohung für Körper und Geist darstellt. Die Symptome reichen dabei von Gehörschädigung (bis hin zum Tinnitus in Extremfällen), Störungen des Herz-Kreislaufsystems, Schlafstörungen, sowie Nachlassen der geistigen Leistungsfähigkeit. Aus diesem Grund schreibt der Gesetzgeber die Errichtung von Lärmschutzanlagen in Gebieten hoher Belastung [3] vor.

Auch im automotiven Bereich sind Arbeiten erstellt worden, um dem im Fahrgastraum auftretenden Lärm zu dämpfen. In Abbildung 1 sind verschiedene geräuschmindernde Systeme dargestellt, welche beispielsweise im HONDA Legend zum Einsatz gelangen. Zu erkennen sind passive Systeme wie Schalldämpfer im Auspuff-Endtopf, verschiedene Lager und Abdeckungen. Das aktive System arbeitet unter Verwendung der im Dachhimmel des Fahrzeugen angebrachten Mikrophone und die Lautsprecher.

Eine andere Art Motorgeräusche zu dämpfen ist in [4] dargestellt. Hierbei gelangen Aktuatoren zum Einsatz, welche direkt das Dachblech in Gegenschwingungen versetzen.

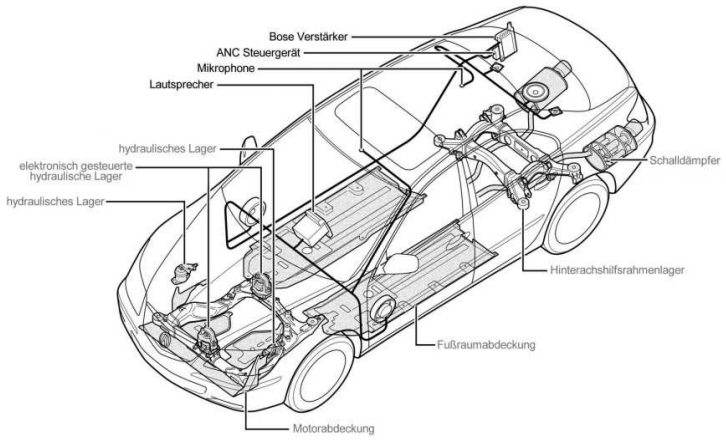

Abbildung 1: Schallreduzierung im automotiven Bereich

Einen weiteren Einsatzbereich zeigt [5], wo aktive Systeme zur Verbes-

serung der Kommunikation für Drive-In Schalter eingesetzt werden. Dabei sollen Störgeräusche eliminiert werden, um die Verständlichkeit zu erhöhen.

Auch an Schallunterdrückungssystemen für Aufzüge wurde bereits gearbeitet [6]. Wie man also sehen kann umfasst der Einsatzbereich für Schallunterdrückungssysteme ein breites Spektrum von Anwendungen. Einen sehr guten Überblick bietet [7].

1.2 Die Idee

Aktive Schallunterdrückung wird durch die Erzeugung eines Anti-Schall-Signals erreicht [8], welches dem unerwünschten Signal in Frequenz und Amplitude möglichst gleicht (im Idealfall vollkommen gleich ist), jedoch um 180° phasenverschoben ist, so daß es das ursprüngliche Signal auslöscht. Eine genauere Erklärung findet sich im Abschnitt 2.3.

1.2.1 Definitionen

Grundsätzlich stellt sich der Aufbau einer Schallunterdrückungslösung wie in Abbildung 2 dar. Auf der linken Seite findet sich sich der Störschall, während auf der rechten Seite der wahrgenommene Schall zu sehen ist.

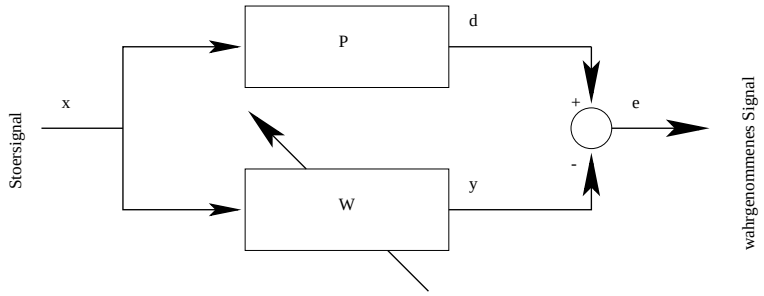

Abbildung 2: Genereller Aufbau eines Schallunterdrückungssystems

Als Eingangsparameter findet sich der Störschall als Signal x. Im Optimalfall ist dies der Schall, welcher direkt an der Störquelle gemessen wird. Die Übertragungsfunktion zur Position an welcher der Schall wahrgenommen wird, wird durch den so genannten *Primärpfad* P modelliert. Das Signal nach dem Primärpfad wird mit d bezeichnet und wird von *desired* abgeleitet. Es handelt sich hierbei also um das Signal, welches es zu eliminieren gilt.

Der Primärpfad kann im Allgemeinen nur abgeschätzt werden und wird durch W modelliert. Je besser die Abschätzung, umso besser funktioniert die Schallunterdrückung. Der Ausgang des Filters ist mit y bezeichnet. Die Auslöschung findet durch Inversion von y statt und ist durch das Minuszeichen nach y erkennbar. Die Notwendigkeit der Inversion ergibt sich aus der Tatsache, daß akustische Signale immer *addiert* werden.

Der Schall, der nach der (meist partiellen) Auslöschung noch wahrzunehmen ist findet sich im Signal e (*error*) wieder. Dieses Signal gilt es zu minimieren.

Damit dies möglich wird, muß gelten, daß $y = d$. Dies wiederum, kann nur durch die Anpassung des Filters W an den Primärpfad erfolgen. Es muß also weiters gelten, daß $W = P$. Sind diese Bedingungen erfüllt, so kann der Fehler e effektiv minimiert werden.

Die Schwierigkeit liegt nun in der Abschätzung des Primärpfades P. Zusätzlich kommt hinzu, daß sich dieser über die Zeit ändern kann. Im Wesentlichen gibt es zwei Ansätze zur Lösung dieses Problems.

1.2.2 Analoger Ansatz

Analoge Lösungen bieten einen reduzierten Bauteilaufwand und sind in der Realisierung meist weitaus einfacher. In billigen Lösungen werden daher in der Regel analoge Schaltungen eingesetzt.

Dabei wird weniger der Primärpfad betrachet, sondern die Inversion – also die Phasenverschiebung. Diese wird meist über eine Operationsverstärkerschaltung realisiert, welche als Allpass mit variabler Verzögerung arbeitet.

Der große Nachteil besteht in der Flexibilität der Schaltung. Die besten Ergebnisse werden erzielt, wenn ein schmalbandiges Signal gleichbleibender Frequenz die Lärmquelle darstellt. So zum Beispiel für Motoren oder Lüfter. Ändert sich jedoch nun die Frequenz, so kann eine optimale Funktionsweise nicht mehr gewährleistet werden.

Der Grund hierfür ist in Abbildung 3 ersichtlich. Die drei Diagramme zeigen Eingangssignal, Ausgangssignal (Anti-Schall), sowie die Summe der beiden vorhergehenden Signale (also den Restschall). Die Periodenlänge des Sinus beträgt 2π. Der Parameter ϕ soll die Phasenverschiebung des Allpass bezeichnen und beträgt für den optimalen Fall π. Die Verschiebung wird für die einzelnen Diagramme nicht verändert.

- Optimale Konfiguration (Periode = π)
- Halbe Frequenz (Periode = 2π)
- Doppelte Frequenz (Periode = $\frac{\pi}{2}$)

Wie man sehen kann, ist bei der halben Frequenz kaum noch eine Dämpfung gegeben. Das Restsignal schwankt zwischen ± 0.5. Schlimmer jedoch ist der Sachverhalt bei doppelter Frequenz. Hier beträgt die Periodendauer genau die, die im Allpass eingestellte Verzögerung. Damit kommt es zu keiner Auslöschung, sondern genau das Gegenteil ist der Fall – das Eingangssignal ist nun in Betrag und Phase gleich dem Ausgangssignal und führt zu einer Verdoppelung des Signals. Dies kann in der Anwendung verheerende Folgen haben, da das Signal effektiv um den Faktor 2 verstärkt wird.

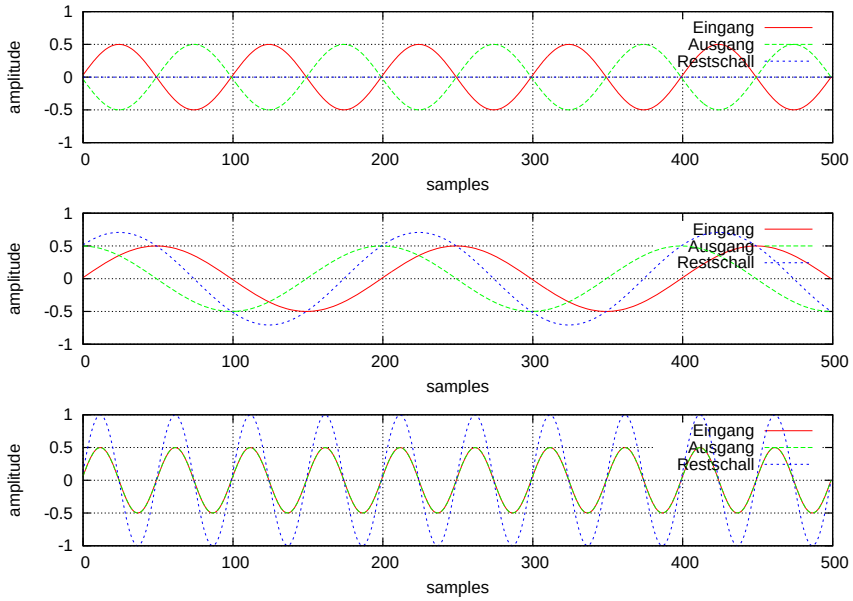

Abbildung 3: Auswirkung unterschiedlicher Frequenzen bei Verwendung eines Allpassfilters

Wie man deutlich erkennt, sind die einfachen analogen Realisierungen lediglich für sehr schmalbandige Frequenzbereiche einsetzbar. Sollte die Frequenz sich ändern, kann es aufgrund der statischen Konfiguration zu großen Problemen kommen. Zudem wird die aktive Modellierung des Primärpfades außer Acht gelassen. Bei einer Änderung der Übertragungsfunktion des Pfades kommt es zu weiteren Problemen.

1.2.3 Digitaler Ansatz

Um das Antischall-Signal digital zu erzeugen, bedient man sich üblicherweise einer aktiven Filterlösung, wie sie zum Beispiel in Abbildung 4 dargestellt ist. Dabei wird als digitales Element ein *adaptiver Filter*, bestehend aus einem *adaptiven Algorithmus* und einem *Filter* eingeführt. Dies ist im Bild als punktiertes Kästchen dargestellt.

Die Grundfunktion des aktiven Filters ist eine Anwendung der *Systemidentifikation*. Es gilt, den Primärpfad rechnerisch zu modellieren und die Koeffizienten des Filters W entsprechend zu stellen.

Für die Anpassung des adaptiven Filters werden der aktuelle Fehler e, sowie das Eingangssignal x herangezogen. Das Bestreben des adaptiven Algorithmus besteht darin den Fehler e zu minimieren. Die Algorithmen, welche für den adaptiven Teil verwendet werden, sind im Abschnitt 2.5.5ff beschrieben.

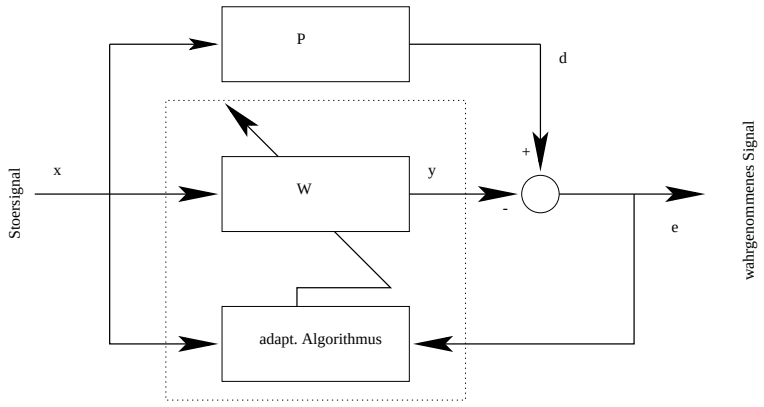

Abbildung 4: Schallunterdrückung mittels aktiver Filter

1.3 Der Aufbau

Wie in [9] und [10] vorgeschlagen, ergibt sich ein möglicher Aufbau für das Geräuschunterdrückungsverfahren, wie in Abbildung 5 dargestellt.

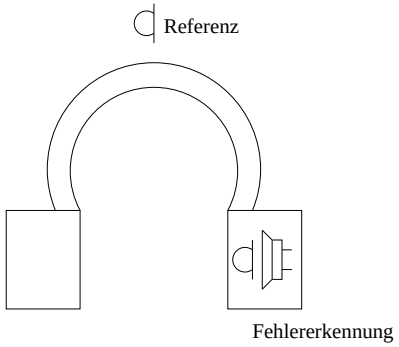

Abbildung 5: Der Aufbau

Auf dem Bügel befindet sich das Referenzmikrophon, welches den Umgebungsschall aufnimmt – also dem Eingang x entspricht. Die Kopfhörerkapseln dämpfen hochfrequente Schwingungen und verringern dadurch den Aufwand der aktiven Lösung. Über den Lautsprecher wird der Gegenschall ausgestrahlt und gleichzeitig von einem Fehlererkennungsmikrophon wieder aufgefangen, welches nun die Summe der eindringenden Störgeräusche und Gegenschallsignal (also y) aufnimmt und als e an die aktive Lösung weiterreicht. Damit ist der Regelkreis nun geschlossen. Das Mikrophon sollte möglichst nahe am Lautsprecher platziert werden, um auftretende Phasenverschiebungen durch die Luftstrecke zu minimieren.

2 Theorie

2.1 Schallunterdrückungsmethoden

Da diese Arbeit den Themenkreis Schallunterdrückung behandelt, wird im Folgenden erläutert, welche Möglichkeiten zur Verfügung stehen und wie diese genutzt werden können.

2.2 Passive Geräuschunterdrückung

Die einfachste Methode um Schall zu reduzieren, ist der Einsatz passiver Lösungen. Darunter fallen Lärmschutzwände neben Autobahnen, Ohrschützer (sei es als Ohrstopfen oder als Kopfhörerausführung) sowie sämtliche schalldämmenden Materialien.

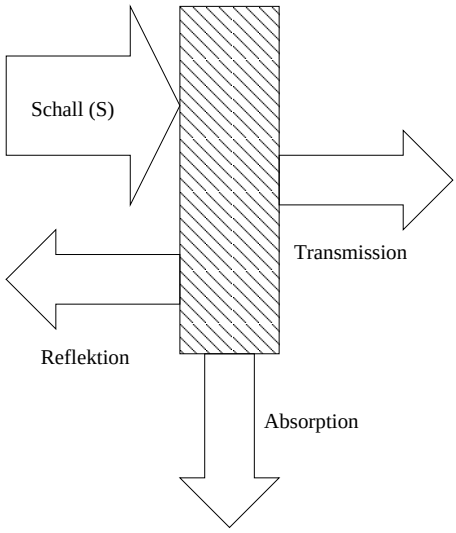

Abbildung 6: Arten der Schalldämmung

In Abbildung 6 sind die möglichen Wege des eintreffenden Schalls (**S**) dargestellt [11]. Ein Teil des Schalls wird vom Material zurückgeworfen (**Reflexion**), ein Teil vom Material absorbiert (**Absorption**).

Der letzte Teil, welcher für diese Betrachtungen am interessantesten ist, ist der Anteil an Schall, welcher durch das Material dringt (**Transmission**). Dies geschieht dadurch, indem das Material selbst, durch den Schall, zum Schwingen gebracht wird – man spricht dabei vom sogenannten *Körperschall*. Hier wird auch klar, weshalb hohe Frequenzen stärker bedämpft werden als Schallwellen niedriger Frequenz. Die Eigenfrequenz der Materialien ist, aufgrund der Masse des Werkstoffes, im Idealfall sehr niedrig. Um das Material zum Mitschwingen anzuregen, muß die Erregerfrequenz möglichst nahe an die Eigenfrequenz kommen. Daher vermögen Schallwellen hoher Frequenz

diese nicht zum Mitschwingen zu bewegen und werden größtenteils absorbiert.

Der Schall, der die Barriere durchdringt, ist ein wesentlicher Bestandteil dieser Arbeit. Schließlich ist es dieser Schall, welcher vom Menschen wahrgenommen wird.

Als konkreteres Beispiel, für die Übertragungscharakteristik bestimmter Schallschutzmaßnahmen, soll ein Gehörschutz samt zugehörigem Diagramm dienen - siehe dazu Abbildung 7 (Quelle: *www.sonicshop.de*).

Abbildung 7: Passiver Gehörschutz - Übertragungsfunktion

Wie man am Diagramm erkennen kann, ist die passive Lösung für tiefe Töne mehr durchlässig als für Töne hoher Frequenz. Der Gehörschutz wirkt also ähnlich wie ein Tiefpaß, wobei auch tiefe Töne, bis zu einem gewissen Maße, bedämpft werden. Möchte man nun auch die störenden Geräusche mit niedriger Frequenz entfernen, so werden passive Lösungen schnell unhandlich und sperrig.

Vorteile:

- einfacher Aufbau
- keine beweglichen oder aktiven Teile
- gute Dämpfung für hohe Frequenzen

Nachteile:

- niedrige Frequenzen werden weitaus geringer gedämpft
- sollen niedrige Frequenzen stark bedämpft werden, dann werden passive Lösungen schnell unhandlich

2.3 Aktive Geräuschunterdrückung

In Abschnitt 1.2 wurde bereits ein Einblick in die Funktionsweise von aktiven Schallunterdrückungsmethoden gewährt. Die Darstellung in Abbildung 2 ist jedoch unvollständig, da die Wirkung der aktiven Bauteile außer Acht gelassen wurde.

Tatsächlich wird durch den Einsatz aktiver Elemente ein zusätzlicher Pfad eingeführt – der sogenannte *Sekundärpfad* $S(z)$. In Abbildung 8 findet sich dieser nach der Gegenschallerzeugung. Hier fließen die Effekte der notwendigen Verstärker, Sensoren (Mikrophone) und Aktuatoren (Lautsprecher) ein.

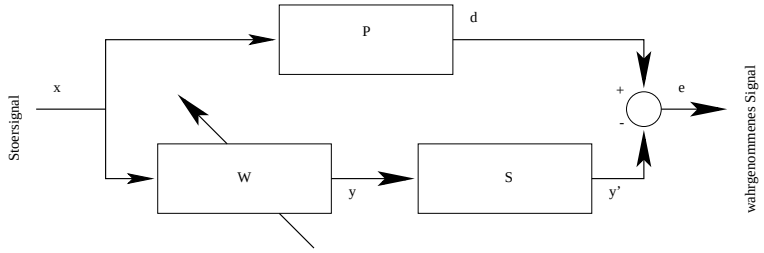

Abbildung 8: Schallunterdrückungssystem unter Einbeziehung des Sekundärpfades

Dies ist insbesonders für digitale Aufbauten wichtig. Als Schnittstellen zwischen analoger (Mikrophon zur Aufnahme des Stör- und Fehlersignals, sowie der Lautsprecher zur Ausstrahlung des Gegenschalls) und digitaler Domäne, müssen *Analog/Digital Wandler*(ADC) beziehungsweise *Digital/Analog Wandler*(DAC) eingesetzt werden. Hinzu kommen Tiefpassfilter, welche einerseits die Bandbreite des Eingangssignals, sowie *Aliasing*-Effekte im Ausgangssignal filtern.

Bei den verwendeten Tiefpässen zweiter Ordnung erreicht die Verschiebung einen Wert von 90 Grad, bei der Grenzfrequenz f_c. Zum besseren Verständnis sei hier auf Abschnitt 7.3.4, insbesondere Abbildung 45 verwiesen. Dort ist der gemessene Frequenz- und Phasengang des Filters in einem Bodediagramm dargestellt.

Die *physikalische Grenze* jeder aktiven Lösung stellt die maximale Phasenverschiebung des Sekundärpfades zum Primärpfad dar. Ein System ist *kausal*, wenn der Primärpfad (also die Übertragungsfunktion vom Referenzmikrophon zu der Stelle, an der der Schall wahrgenommen wird) eine größere Verschiebung aufweist als der Sekundärpfad.

Sollte der elektrische Pfad – also der Sekundärpfad – eine Verschiebung einbringen, welche vom Primärpfad nicht kompensiert werden kann, so können aperiodische Signale *nicht mehr ausgelöscht werden*.

In diesem Fall ist der aktive Filter nicht mehr in der Lage die Verzögerung durch Anpassung wieder wettzumachen. Die Leistungsfähigkeit des aktiven

Filters wird stark geschmälert, denn dann können nur noch periodische Signale bedämpft werden, deren Periodendauer mindestens doppelt so groß ist, wie die durch den aktiven Filter erreichbare Verzögerung.

Als Beispiel soll folgendes Szenario dienen (siehe auch Abbildung 9):

- Primärpfad - Verzögerung: 2 ms
- Sekundärpfad - Verzögerung: 5 ms
- Störgeräusch: Dirac Impuls mit 1 ms Länge

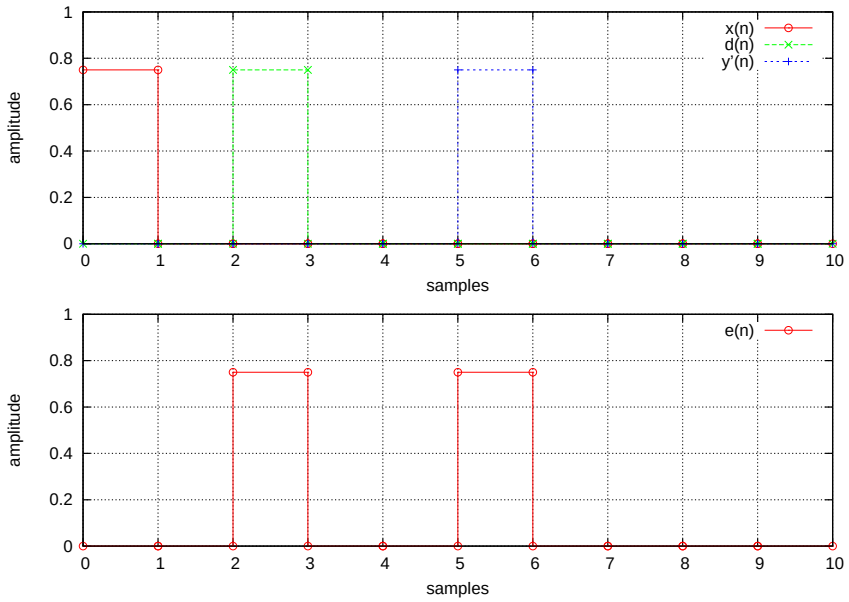

Abbildung 9: Beispiel zu Kausalität

Über den Sekundärpfad kann kein sinnvolles Gegensignal mehr augegeben werden. Die Differenz der Verzögerungen, Sekundär- nach Primärpfad, beträgt 3 ms. Das heißt, daß das Antischallsignal *frühestens* 1 ms nach dem Eintreffen des Störgeräusches ausgestrahlt werden kann. Der Gegenschall kommt also 2 ms zu spät.

Generell kann also folgendes festgestellt werden: Ist die Differenz der Verzögerungen Sekundärpfad minus Primärpfad *größer* als Null, so ist das System nicht mehr kausal. Nur wenn die Differenz *kleiner* als Null ist, ist Kausalität gegeben und der Algorithmus kann das Ereignis noch bevor der Schall die Zielzone erreicht, bearbeitet und vorhergesagt werden.

Vorteile:

- bessere Auslöschung niederfrequenter Störgeräusche möglich

Nachteile:

- aufwändiger in der Realisierung
- hohe Korrelation notwendig
- physikalische Limitierung durch Einsatz von Tiefpassfiltern hoher Ordnung

2.4 Hybrider Lösungsansatz

Sieht man sich nun die Eigenschaften der aktiven und passiven Lösungen an, so wird man schnell erkennen, daß die Kombination der beiden Techniken ein gangbarer Weg ist, um das Beste aus beiden Welten zu erhalten.

Hierbei wird eine passive Lösung zur Bedämpfung der hohen Frequenzen und eine aktive Lösung für die Bedämpfung tiefer Frequenzen gewählt. Die Beschränkung der aktiven Lösung läßt sich durch die verwendeten Tiefpassfilter nicht vermeiden.

Zur Veranschaulichung werden Kopfhörer herangezogen, welche beide Techniken einsetzen. In Abbildung 10 (Quelle: Sennheiser und AKG) sind Kopfhörersyteme abgebildet, welche aktive Geräuschunterdrückung verwenden. Die höheren Frequenzen werden dabei durch die Polsterung bedämpft, während der Restschall über eine aktive Lösung verringert wird.

Abbildung 10: Sennheiser PXC300 und AKG K28NC

Laut Datenblatt des Sennheiser Modells beträgt die passive Dämpfung etwa -15dB bis -25dB, im Bereich von Frequenzen über 1200Hz, während die aktive Lösung die Schallwellen mit einer Frequenz unter einem kHz bis zu -15dB abschwächt. Die Technologie wird unter dem Namen $NoiseGard^{TM}$ vermarktet. Zum Modell von AKG ließen sich leider keine detaillierten Informationen finden.

Um die Funktionsweise noch weiter zu verdeutlichen, beachte man Abbildung 11. In der Graphik sind die zugeteilten Frequenzbereiche für den passiven und aktiven Filterteil ersichtlich. Der aktive Filter wird in der Region bis etwa f_g benötigt, da der passive Filter noch zu wenig Dämpfung

liefert. Erst ab f_g ist die Filterwirkung des passiven Filters ausreichend, um auf die aktive Lösung verzichten zu können. f_g bezeichnet dabei die Grenzfrequenz bei der die Dämpfung der einen Lösung, den anderen Teil ersetzen kann. Dabei ist es wichtig festzustellen, daß diese Grenze nicht scharf gezogen ist.

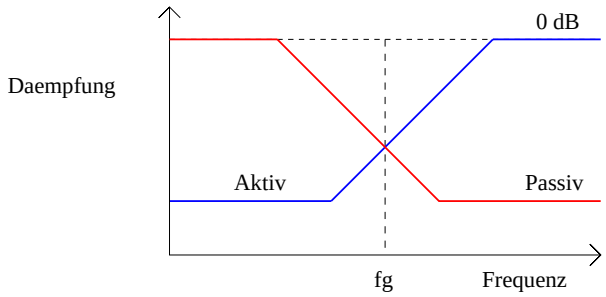

Abbildung 11: Filterbereiche

2.5 Digitale Filter

2.5.1 Allgemeines

Digitale Filter bilden das mathematische Pendant zu analogen Lösungen. Der Aufwand ist jedoch meist um einiges höher. Mindestens drei Teile werden benötigt, um mit einem digitalen Filter eine Filterfunktion zu realisieren [12]:

- Analog/Digital Wandler (inklusive Analog-Beschaltung)
- DSP (*Digital Signal Processor*)
- Digital/Analog Wandler (inklusive Analog-Beschaltung)

Der DSP bildet das Herz des Filters und führt die eigentlichen Berechnungen aus. In [13] [14] finden sich Implementierungen auf DSP Bausteinen. Die Wandler dienen als Interface zum DSP, der Verstärkung und analogen Vorfilterung des Signals. Diese ist unumgänglich, da der A/D Wandler mit einer fixen Abtastfrequenz betrieben wird und Frequenzen über der Abtastfrequenz störend eingehen. Umgekehrt wird für die D/A Wandlung ein sogenannter Anti-Aliasing Filter benötigt. Der Grund hierfür liegt in der Stufigkeit des vom D/A Wandlers gelieferten Signals, welches einen hohen Oberwellenanteil aufweist.

Generell können zwei Klassen von digitalen Filtern unterschieden werden:

- FIR - Filter

- IIR - Filter

Diese beiden Filterklassen haben beide ihre individuellen Vor- und Nachteile, sowie bestimmte Charakteristika, auf welche in den folgenden zwei Abschnitten kurz näher eingegangen werden soll. Eine ausgezeichnete Einführung in die digitale Signalverarbeitung und eine weitaus erschöpfendere Behandlung der Filterklassen findet sich in [15].

2.5.2 Symbole und Abkürzungen

$x(n)$	skalarer Signalwert zum Zeitpunkt, bzw. Sample n
$\mathbf{x}(n)$	Signalvektor zum Zeitpunkt, bzw. Sample n
$P(z)$	Übertragungsfunktion (z-Transformation)
z^{-1}	Signalverzögerung um ein Sample

2.5.3 FIR - Filter

Diese Klasse bezeichnet Transversalfilter mit *endlicher Sprungantwort*. Das bedeutet, daß nach einem Impuls und einer darauffolgenden Stille (Folge von Nullwerten) das Ausgangssignal des Filters auch wieder auf den Nullpegel einschwenkt. Daher auch der Name *FIR*, welcher für *finite impulse response* steht.

Diese Tatsache ergibt sich aus der Funktionsweise des Filters: zur Berechnung werden Koeffizienten mit den Eingangswerten multipliziert und das Ergebnis addiert. Es wird also, mathematisch gesehen, eine *Faltung* des Koeffizientenvektors mit dem Eingangsvektor durchgeführt. Damit ist leicht erkennbar, daß nach einem Sprung das Ergebnis der Filterung sich wiederum auf Null einpendelt und schließlich bei diesem Wert verharrt. Die Dauer des Abklingvorganges ist grundsätzlich von der Ordnung des Filters (also der Anzahl der Koeffizienten) abhängig.

Die Berechnungsvorschrift für einen Filter *erster Ordnung* ist durch Gleichung 2.1 gegeben:

$$y(n) = wx(n) \tag{2.1}$$

$x(n)$ ist das Eingangssignal zu einem bestimmten Zeitpunkt n, analog dazu $y(n)$ das Ausgangssignal. w ist der Filterkoeffizient. Oft wird der Terminus Ordnung mit der Anzahl der Koeffizienten gleichgesetzt.

Filter beliebiger Ordnung greifen auf einen Vektor von Eingangssignalen und einem Koeffizientenvektor zurück. Die mathematische Formulierung ist durch Gleichung 2.2 (N ist die Ordnung des Filters) gegeben:

$$y(n) = \sum_{i=1}^{N} x(n-i)w(i) \tag{2.2}$$

Der Ausgangswert wird also durch wiederholtes Multiplizieren und Aufaddieren (*multiply-accumulate*, MAC) gebildet. Ein Filter mit Ordnung N benötigt also N MACs. Die Vektorschreibweise dieser Gleichung ist wie folgt:

$$y(n) = \mathbf{w}\mathbf{x}^T(n) \tag{2.3}$$

Graphisch werden Filter zumeist wie in Abbildung 12 dargestellt. Die Schreibweise z^{-1} bezeichnet dabei eine Verzögerung von einem Sample.

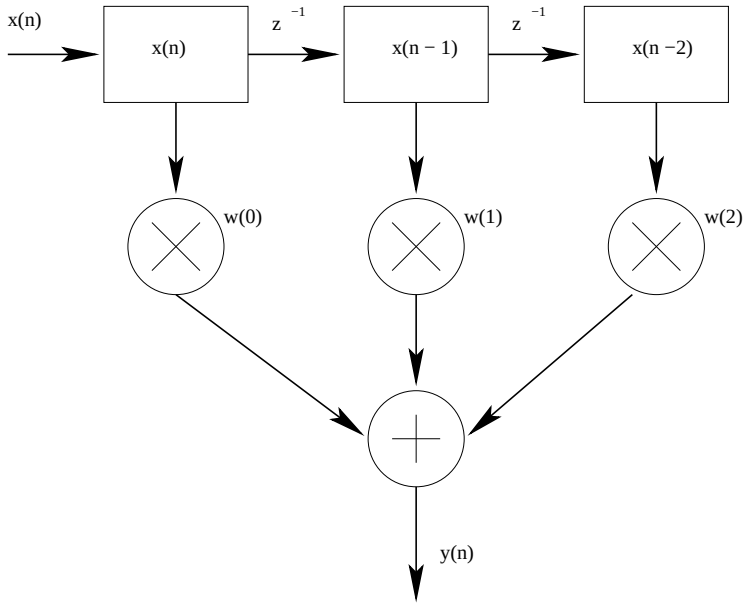

Abbildung 12: FIR Filter 3. Ordnung

Bei den hier vorgestellten Filtern ändern sich die Koeffizienten nicht, sondern bleiben konstant über die Zeit.

Filter die ihre Koeffizienten ändern, besitzen also einen zeitabhängigen Koeffizientenvektor. Dabei werden die Koeffizientenwerte kontinuierlich neu berechnet. Es gilt meist eine Fehlerfunktion zu minimieren, beziehungsweise ein bestimmtes Optimalitätskriterium zu erfüllen. Beispielsweise versucht der im Abschnitt 2.5.5 vorgestellte *LMS* Algorithmus den quadratischen Fehler zu minimieren.

Der Hauptvorteil der FIR Filter ist ihre geringe mathematische Komplexität, sowie ihre hohe Resistenz gegenüber numerischen Fehlern. Diese Tatsache ergibt sich daraus, daß keine Ergebniswerte (also $y(n)$) für aktuelle Berechnungen herangezogen werden. Dadurch ist es nicht möglich numerische Fehler zu akkumulieren.

Es wurden bereits Arbeiten [16] erstellt, um den Aufwand der Multiplikation in Hardware zu reduzieren. Wie man also sehen kann stellt die Klasse der FIR Filter ein sehr mächtiges Werkzeug dar.

2.5.4 IIR - Filter

Da in dieser Arbeit keine IIR Filter verwendet worden sind, sollen sie, nur der Vollständigkeit halber, Eingang in diese Arbeit finden.

IIR steht dabei für *infinite impulse response*. Das bedeutet, daß es nicht notwendigerweise gegeben ist, daß der Filter nach einem Impuls in endlicher Zeit wieder zum Nullpegel zurückkehrt. FIR Filter berücksichtigen nur *zurückliegende Eingangssignale*, während IIR Filter auch *zurückliegende Ausgangssignale* in die Berechnung mit einbeziehen.

IIR Filter haben einige wesentliche Nachteile, welche bei der Überlegung, welcher Filtertypus eingesetzt werden sollte, eine wichtige Rolle spielen.

Durch die Rückkopplung (*feedback*) des Ausgangssignales in die laufende Berechnung, beeinflussen numerische Fehler die Filterung wesentlich. Dadurch ist die Wahrscheinlichkeit, daß der Filter instabil wird und anfängt zu oszillieren, wesentlich höher als bei einem FIR Filter. Auch bieten IIR Filter keinen linearen Phasengang. Ihre Komplexität stellt bei der Analyse der Ausgabe den Entwickler vor eine große Herausforderung.

Der Grund, weshalb IIR Filter dennoch eine wichtige Rolle spielen, ist die Tatsache, daß die Effizienz viel höher ist, als bei einem FIR Filter. Es sind beispielsweise weniger Multipliationen erforderlich, um ein vergleichbares oder in vielen Fällen besseres Filterverhalten im Vergleich zu einem FIR Filter zu erhalten.

Da bei diesem Projekt jedoch nur Fixpunktgenauigkeit für die Berechnungen verfügbar ist, würden die numerischen Fehler den Algorithmus instabil machen. Aus diesem Grund wurden bei der Implementierung nur FIR Filter in Betracht gezogen.

2.5.5 Der LMS Algorithmus

Einer der bekanntesten Algorithmen auf dem Gebiet der digitalen Filter, ist der *Least Mean Square* Algorithmus – in Folge LMS-Algorithmus genannt. Durch seine einfache Implementierbarkeit ist er besonders geeignet, um auf Systemen mit geringer rechnerischer Komplexität eingesetzt zu werden. Werfen wir nun einen Blick auf die Funktionsweise des LMS:

Die folgenden Gleichungen, welche die Arbeitsweise des Algorithmus beschreiben sind weitaus ausführlicher in [17] behandelt. Dennoch soll im Folgenden die grundsätzliche Funktionsweise erläutert werden.

Der Algorithmus bedient sich im Wesentlichen eines Eingangssignalvektors $\mathbf{x}(n)$, eines Koeffizientenvektors $\mathbf{w}(n)$, sowie einer Konstanten μ. Zusätzlich gibt es noch die Werte $e(n)$, welcher das Fehlersignal beschreibt, sowie $d(n)$, welcher das zu unterdrückende Signal repräsentiert (siehe auch Abschnitt 1.2.1).

Während der Durchführung des Algorithmus ist dieser nun bestrebt, den Wert von $e(n)$ zu minimieren (im besten Fall ist $e(n) = 0$), indem der Ko-

effizientenvektor $\mathbf{w}(n)$ verändert wird. Dabei wird der *mittlere quadratische Fehler* (engl. *Mean Square Error*) der in Gleichung 2.4 dargestellt ist minimiert. Für die Anwendung zur Schallunterdrückung wird dabei $e(n)$ aus der Differenz $d(n) - y(n)$ berechnet.

$$J(n) = e^2(n) = [d(n) - \mathbf{w}(n)\mathbf{x}^T(n)]^2 \tag{2.4}$$

Der Algorithmus versucht das Minimum der quadratischen Fehleroberfläche zu erreichen. Dabei wird die *Gradientenmethode über den steilsten Abstieg* verwendet. Das Aktualisieren der Koeffizienten kann auch mit Hilfe des Fehlers angegeben werden:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \left(-\frac{\delta J_n}{\delta \mathbf{w}(n)} \right) \tag{2.5}$$

Dabei ist μ eine Konstante und bestimmt das Konvergenzverhalten des LMS Algorithmus über ihren Wert maßgeblich. Bei höheren Werten konvergiert der Algorithmus schneller, bei kleineren Werten langsamer. μ darf jedoch nicht beliebig gewählt werden, möchte man sicherstellen, daß der LMS Algorithmus konvergiert. Abschätzungen dieses Wertes sind weiter unten gegeben.

Der Ableitungsterm $-\frac{\delta J_n}{\delta \mathbf{w}(n)}$ berechnet sich zu $-2e(n)\mathbf{x}(n)$ und nach Einsetzen in Gleichung 2.5 erhält man die Form des Koeffizientenupdates für den LMS Algorithmus:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n) \tag{2.6}$$

Die Initialisierung des Eingangsvektors, sowie des Koeffizientenvektors ist in Gleichung 2.7 beschrieben. Man beachte jedoch, daß auch andere Initialisierungen möglich sind, je nachdem, wie viel Information zu Beginn des Algorithmus bereits zur Verfügung steht. Dies kann zum Beispiel eine Vorkenntnis der Werte des Koeffizientenvektors beinhalten.

$$\mathbf{x}(0) = \mathbf{w}(0) = [0 \quad 0 \quad \dots \quad 0]^T \tag{2.7}$$

Der nächste Schritt ist nun die Ausführung der Berechnungsschritte, welche in Gleichung 2.8 beschrieben sind.

$$\begin{aligned} &\forall n \geq 0 : \\ &\quad e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n) \\ &\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n) \end{aligned} \tag{2.8}$$

Wie aus den Gleichungen ersichtlich, ist der Aufbau des Algorithmus ziemlich einfach. Zu Beginn werden die Eingangs- und Koeffizientenvektoren auf Null gesetzt. Danach wird der Fehler berechnet, der sich aus der Differenz

des gewünschten Signales und des gefilterten Eingangssignales ergibt. Nur für den Fall des optimalen Koeffizientenvektors $\mathbf{w}_{opt}$ ergibt sich $e_{opt}(n) = 0$ und damit $\mathbf{w}(n+1) = \mathbf{w}(n)$ im nächsten Schritt. Ist jedoch $e(n) \neq 0$ dann wird der Koeffizientvektor erneut angepasst.

Wie bereits erwähnt steuert der Parameter μ Konvergenzverhalten der Koeffizienten. Der Wert von μ *muß* der Ungleichung 2.9 genügen, damit der LMS-Algorithmus konvergiert. N bezeichnet die Ordnung des Filters und $P_{average}$ die mittlere Eingangsleistung. Diese kann durch die Summe der Quadrate des Eingangsvektors errechnet werden – dies ist im Abschnitt zum NLMS Algorithmus genauer dargestellt.

$$0 < \mu < \frac{2}{N \cdot P_{average}} \tag{2.9}$$

2.5.6 Der NLMS Algorithmus

Der *Normalized Least Mean Square* Algorithmus – in Folge NLMS Algorithmus genannt – ist eine Erweiterung des LMS Algorithmus, um dessen Konvergenzverhalten zu verbessern. Die Konvergenz des LMS Algorithmus wird durch die Wahl des Parameters μ bestimmt. Dieser Parameter wird zu Beginn der Abarbeitung festgelegt und wird während der Berechnung nicht verändert.

Genau hier setzt der NLMS Algorithmus an. μ ist nun nicht mehr konstant, sondern wird für jeden Zeitpunkt n neu berechnet. Gleichung 2.10 zeigt die Berechnung des Parameters μ zu einem Zeitpunkt $n \geq 0$.

$$\mu(n) = \frac{\beta}{\gamma + N \cdot P_{average}} \qquad 0 < \beta < 2 \tag{2.10}$$

$$\mu(n) \in [\frac{\beta}{\gamma + N \cdot P_{average}} \quad \dots \quad \frac{\beta}{\gamma}] \tag{2.11}$$

Neu hinzugekommen sind die Parameter β und γ. Über diese Parameter ist es möglich, zu Zeitpunkten an denen die mittlere Eingangsleistung $P_{average}$ auf Null fällt, ein maximales $\mu(n)$ zu definieren. Mit $P_{average} = 0$ reduziert sich Gleichung 2.10 zu:

$$\mu(n) = \frac{\beta}{\gamma} \tag{2.12}$$

Sowohl β als auch γ müssen ungleich Null sein. Wäre β gleich Null, so würden die Koeffzienten sich nie ändern. Schwerwiegender würde sich jedoch γ gleich Null auswirken. Im Falle, daß $P_{average} = 0$ würde dies zu $\frac{\beta}{0}$ führen und damit nicht berechnet werden können.

Die mittlere Eingangsleistung kann wiederum durch folgende Gleichung abgeschätzt werden:

$$P_{average} = \frac{\mathbf{x}^T(n)\mathbf{x}(n)}{N} \tag{2.13}$$

Die vollständige Beschreibung des NLMS Algorithmus sieht also nun folgendermaßen aus:

$$\text{Initialisierung: } \mathbf{x}(0) = \mathbf{w}(0) = [0 \quad 0 \quad \dots \quad 0]^T \tag{2.14}$$

$$\begin{aligned} &\forall n \geq 0 : \\ &\quad e(n) = d(n) - \mathbf{x}^T(n)w(n) \\ &\quad \mu(n) = \frac{\beta}{\gamma + N \cdot P_{average}} = \frac{\beta}{\gamma + \mathbf{x}^T(n)\mathbf{x}(n)} \\ &\mathbf{w}(n+1) = \mathbf{w}(n) + \mu(n)e(n)\mathbf{x}(n) \end{aligned} \tag{2.15}$$

2.5.7 Der FxLMS Algorithmus

Der *Filtered-X Least Mean Squares* Algorithmus [18] - in Folge FxLMS Algorithmus genannt – ist eine Spezialisierung der LMS Algorithmen für ein gegebenes Problem – in diesem Fall aktive akustische Schallunterdrückung.

Um eine sinnvolle Schallreduzierung zu erreichen, ist es wichtig ein hohes Maß an Korrelation zwischen dem Fehlersignal $e(n)$ und dem Eingangssignal $x(n)$ zu gewährleisten. Durch Verzögerungen im Sekundärpfad ist dies jedoch nicht immer möglich.

Der naive Ansatz wäre hier, den Sekundärpfad $S(z)$ durch eine Inversion zu eliminieren. Dies ist jedoch nur möglich, wenn es sich beim Sekundärpfad um ein *Minimalphasensystem* handelt. Dies ist in der Regel jedoch nie der Fall, da ein Minimalphasensystem dadurch charakterisiert ist, daß es keine Verzögerung aufweist. Jede Übertragungsfunktion läßt sich in zwei Bestandteile zerlegen: das besagte Minimalphasensystem, sowie einen Allpass (also eine fix definierte Verzögerung). Erst wenn der Allpassteil wegfallen kann, ist die Übertragungsfunktion minimalphasig. Dieser wird jedoch aufgrund der Laufzeiten der Bauteile nie der Fall sein. Also bedient sich der FxLMS Algorithmus (Abbildung 13) einer anderen Methode.

Durch die Nachbildung des Sekundärpfades $\hat{S}(z)$ wird dem adaptiven Algorithmus eine Signalbasis zur Verfügung gestellt, welche die späteren Einwirkungen des Pfades $S(z)$ bereits enthält. Damit ist eine möglichst akkurate Berechnung des Ausgangssignales möglich. Dabei ergibt sich die Gleichung für das Koeffizientenupdate wie folgt:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu e(n)\mathbf{x}'(n) \tag{2.16}$$

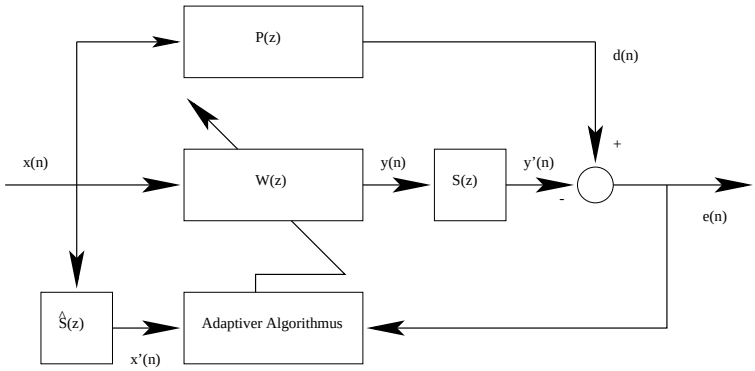

Abbildung 13: Der FxLMS Algorithmus

Das eingehende Signal $x(n)$ wird vorgefiltert und steht dann als $x'(n)$ zur Verfügung. Diese Filterung hat dem FxLMS Algorithmus zu seinem Namen verholfen. Mathematisch ausgedrückt sieht das Ganze so aus:

$$x'(n) = \sum_{i=1}^{M} c(i)x(n-i) \tag{2.17}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu e(n)\mathbf{x}'(n) \tag{2.18}$$

Hier ist $c(i)$ das i-te Element des Koeffizientenvektors der Sekundärpfadabschätzung. M stellt dabei die Ordnung des Sekundärpfadfilters dar. Die Gleichungen sind auf den LMS Algorithmus zugeschnitten, können jedoch leicht für den NLMS angepasst werden. Dabei ist nur das Koeffizientenupdate zu verändern.

Es ist weiters zu beachten, daß $x'(n)$ nur dem LMS Teil zur Verfügung gestellt wird. Der FIR Teil des aktiven Filters erhält das ungefilterte Eingangssignal $x(n)$.

Damit der Filter $\hat{S}(z)$ möglichst genaue Ergebnisse liefert, muß dieser zu Anfang kalibriert werden. Dies kann durch einen Rauschgenerator erfolgen, welcher an den Lautsprecher und den Filter angebunden ist. Durch Analyse des Signals, errechnet nun der Filter über das Fehlererkennungsmikrophon die Übertragungskoeffizienten und ist damit auf den Sekundärpfad eingestellt. Um den Filter $\hat{S}(z)$ akkurat zu halten, muß die Kalibrierung bei geänderten Bedingungen (zum Beispiel Verrutschen des Kopfhörers) wiederholt werden. Eine laufende Kalibrierung ist nicht empfehlenswert, da dies ein konstantes Rauschen zur Folge hätte und ebenso als störend empfunden werden würde.

Die Applikation des FxLMS Algorithmus ist jedoch nicht unproblematisch. Welche konkreten Probleme auftreten können, wird im Abschnitt 4.4 untersucht.

3 Validierung der Algorithmen

3.1 Implementierung in OCTAVE

Der erste Schritt, ist die Implementierung der Algorithmen in OCTAVE (`www.octave.org`), einer frei verfügbaren Implementierung des MATLAB Befehlssatzes. Dies soll dazu dienen greifbare Ergebnisse zu erhalten und ein Gefühl für die Funktionsweise der verwendeten Algorithmen zu entwickeln.

Es wurde, in Hinsicht auf die spätere Realisierung, darauf geachtet Konstrukte zu verwenden, welche später einfach in einer prozeduralen Sprache realisiert werden können.

3.1.1 Der FIR-Filter

Zu Beginn wurde ein allgemeiner FIR Filter erstellt, auf welchen die Implementierungen des LMS und NLMS Algorithmus zurückgreifen. Die Arbeitsweise eines FIR Filters läßt sich wie folgt in OCTAVE beschreiben:

```
function [nh, out] = fir(h, c, x)
  % update tap vector
  for j = length(c) - 1:-1:1,
    nh(j + 1) = h(j);
  end;
  nh(1) = x;

  % compute filtered signal
  out = c * nh';
```

Es ist anzumerken, daß die Funktion keinen Zustand hat, sondern lediglich auf, von außen eingebrachten, Vektoren und Werten operiert. Die Ausgabe der Funktion beinhaltet die neue Wertekette, sowie das gefilterte Ausgangsignal. Gut erkennbar ist das Einbringen des neuen Inputwertes in die Kette `h`, sowie die Berechnung des Ausgangssignals über Multiplikation mit den Koeffizientenvektor `c`.

3.1.2 Der LMS Algorithmus

Im nächsten Schritt wurde der LMS Algorithmus, aufbauend auf dem FIR-Filter, realisiert:

```
function [nc, nh, out] = lms(h, c, x, cx, e)
  % define mu
  mu        = 0.125;
  [nh, out] = fir(h, c, x);

  % perform coefficient update
```

```
nc = c + 2 * mu / length(h) * e * h;
```

Die wesentliche Neuerung hier ist, daß der Koeffizientenvektor `c` verändert wird und als neuer Vektor `nc` zurückgeliefert wird. Der Fehler fließt dabei in das Koeffizientenupdate mit ein.

3.1.3 Der NLMS Algorithmus

Der NLMS Algorithmus ist nur eine geringfügige Änderung gegenüber dem LMS Algorithmus und unterscheidet sich durch die Berechnung der neuen Koeffizienten:

```
% estimate input power
power = h * h';
% perform coefficient update
nc = c + 2 * beta / (power + gamma) * e * h;
```

Hier ist die Berechnung der Eingangsleistungsabschätzung `power` das tragende Element. Dadurch ist es möglich `beta` beinahe beliebig zu wählen, da es in Folge laufend angepasst wird. Diese Eigenschaft macht den NLMS Algorithmus besonders wertvoll.

3.1.4 Testen der Implementierung

Der Test der Implementierungen wurde als Systemidentifikation ausgeführt. Dabei wurde als Eingangssignal weißes Rauschen gewählt und als zu identifizierende Funktion ein Allpass, mit einer konstanten Verzögerung von 10 Samples.

In Abbildung 14 sind die Signalverläufe über die ersten 1000 Samples, zu sehen. Die erste Zeile bildet das Eingangssignal, darunter ist das Signal nach dem Allpass zu finden. Zeile drei zeigt das Ausgangssignal des LMS Blocks. Die letzte Zeile zeigt die Differenz der vom FIR gefilterten Signals und des vom LMS gelieferten Signals. Es ist gut zu erkennen, wie sich der LMS Algorithmus an den Filter annähert. Als Parameter μ wurde der Wert 0.3 appliziert. Wie man sieht, konvergiert der Algorithmus recht langsam.

Nun soll die selbe Ausgangssituation mit dem NLMS Algoritmus getestet werden. Das Ergebnis ist in Abbildung 15 zu sehen.

Der Parameter β wurde wiederum auf 0.3 festgelegt, jedoch wird er im Laufe der Anpassung verändert. Der Sicherheitsparameter γ wurde mit 0.001 initialisiert. In der letzten Kurve ist der Verlauf des Wertes des Wertes von $\mu(n)$ in logarithmischer Skalierung dargestellt. Durch die, durch den Allpass induzierte Nullfolge, am Anfang, läßt sich das Wirken des Parameters γ sehr schön erkennen. Da die Leistungsabschätzung für die ersten Samples Null ergeben würde, würde eine Division durch Null auftreten. So wird dies aber in eine Division durch $0 + \gamma$ umgewandelt. In der „heißen“ Phase der

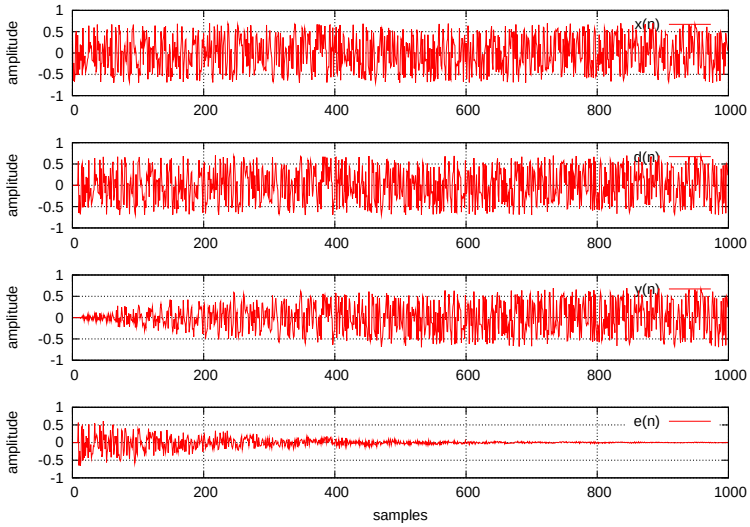

Abbildung 14: Evaluierung der OCTAVE LMS Implementierung

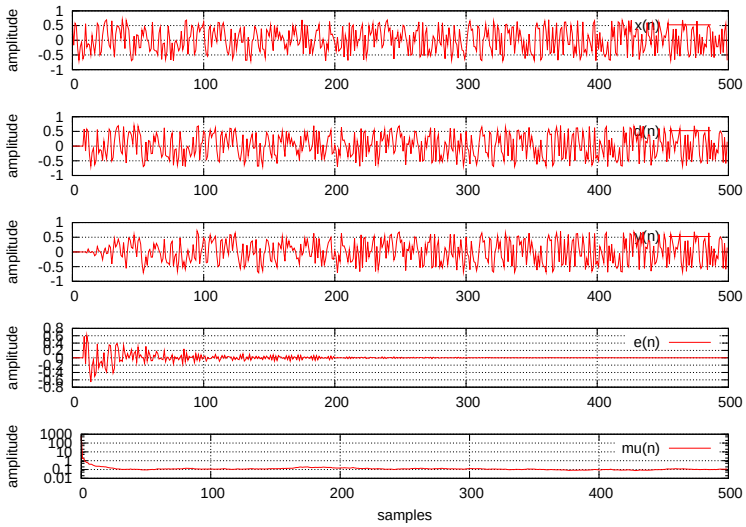

Abbildung 15: Evaluierung der OCTAVE NLMS Implementierung

Anpassung ist $\mu(n)$ recht groß, da der Fehler entsprechend groß ist. Je besser jedoch die Anpassung, umso kleiner wird der Fehler und in weiterer Folge pendelt sich $\mu(n)$ um den Wert 0.1 ein.

Auffallend ist die Konvergenzgeschwindigkeit. Lag diese beim LMS mit gleichen Parametern noch bei etwa 1000 Samples, so ist nun schon nach etwa 100 Samples der Fehler sehr klein. Durch die laufende Anpassung kann eine schnelle Konvergenz zu Beginn und eine sehr feine Annäherung, im weiteren Verlauf, beobachtet werden. In diesem Fall ist der Geschwindigkeitsvorteil sogar bei Faktor 10.

Nachdem gezeigt werden konnte, daß die Algorithmen ihren Dienst zur Zufriedenheit verrichten, wurde nun eine Implementierung in C++ ins Auge gefasst. Diese soll zur weiteren Untersuchung bestimmter Eigenschaften der LMS Familie dienen.

3.2 Implementierung in C++

3.2.1 Allgemeines

Vor der Implementierung in VHDL wurde zuerst eine C++-Version erstellt. Ziel war es, eine möglichst objektorientierte Lösung zu finden, um die einzelnen Komponenten später auf die Hardwareeinheiten abzubilden. Grundsätzlich sollten also folgende Komponenten verfügbar sein:

- Eingabe (`CInput`)
- Ausgabe (`COutput`)
- Filterung (`CFilter` und `CFilter2`)

Diese Elemente sind als abstrakte Klassen definiert. Von diesen Klassen leiten sich die spezialisierten Versionen ab.

Intern werden die Samples als `double` weitergereicht, also als 32-bit Gleitkommazahlen. Der Bereich der Zahlen ist auf das Intervall von -1 bis +1 beschränkt. Auch ist nur monaurale Verarbeitung möglich.

In den folgenden Abschnitten soll eine kurze Zusammenfassung der verfügbaren Klassen gegeben werden, um die Möglichkeiten die das Framework bietet darzustellen.

3.2.2 Eingabeobjekte

Für die Eingabe wurden drei spezialisierte Klassen erzeugt:

- `CRAWInput`
- `CWAVInput`
- `CBuffer`

`CRAWInput` ist eine Klasse zum Lesen von rohen Eingabedaten. Das heißt, daß die Daten als `double` Werte vorliegen. Das Format ist implizit mono, also wird lediglich eine Tonspur verwendet. Es werden keine Daten über die Abtastrate gespeichert – diese muß man manuell über das `CFormat` Objekt festlegen.

Eine Verbesserung bietet die Klasse `CWAVInput`, welche imstande ist eine Standard PCM Datei zu lesen. Die Parameter werden aus den Headern gelesen und intern im Objekt gespeichert. Sollten mehrere Tonspuren vorhanden sein, so wird implizit die erste Tonspur gelesen.

Die Klasse `CBuffer` implementert sowohl Eingabe als auch Ausgabe. Sie stellt einen Puffer bereit, der es ermöglicht Audiodaten zwischenzuspeichern und an nachgeschaltete Objekte weiterzureichen.

Alle Objekte halten die Audioinformation auf Abruf im Speicher. Auch besteht die Möglichkeit den Datenstrom ,,zurückzuspulen", um sich das wiederholte Öffnen des Eingabestromes zu ersparen. Dies ist jedoch nicht für alle Objekte anwendbar.

3.2.3 Ausgabeobjekte

Für die Ausgabe existieren mehrere Möglichkeiten:

- `CRAWOutput`
- `CWAVOutput`
- `CTeeOutput`
- `CgnuplotOutput`
- `CBuffer`

`CRAWOutput` ist das Pendant zu `CRAWInput`. Dies ist eine sehr simple Form die Samples zu speichern. Es werden die `double` Werte sequentiell ohne zusätzliche Information in die Datei geschrieben.

`CWAVOutput` bietet die Möglichkeit eine PCM Datei, mitsamt Header, zu schreiben, um die Weiterverarbeitung in anderen Programmen zu gewährleisten. Die Formatdaten werden dabei mittels eines `CFormat` Objektes übergeben.

`CTeeOutput` dient als Verteiler, um einen Ausgang aufzuspalten. Dazu können mehrere Objekte vom Basistyp `COutput` registriert werden. Alle Daten die an das Objekt gesendet werden, werden an die registrierten Komponenten weitergegeben.

`CgnuplotOutput` ist eine Möglichkeit die Daten direkt als Graph in GNUplot darzustellen. Dazu wird ein Skript und eine Datendatei erstellt. Das Skript wird in GNUplot ausgeführt und dient dazu die Daten aus der anderen Datei zu lesen und darzustellen.

`CBuffer` implementiert ebenso die Klasse `COutput`, um die, in der vorangegangenen Sektion beschriebene, Funktion zu erfüllen.

3.2.4 Filterobjekte

Die Filter bilden das Herzstück des vorliegenden Frameworks. Es sind folgende Spezialisierungen verfügbar:

- `CDelayFilter`
- `CAddFilter`
- `CSubFilter`

- `CLMSFilter`
- `CNLMSFilter`

`CDelayFilter` ist ein einfacher Filter, der eine Stille vor den Datenstrom hängt und damit die Audiodaten um eine gegebene Anzahl von Samples verzögert. Es wird also ein Allpass realisiert.

`CAddFilter` und `CSubFilter` sind Filter zum Addieren, beziehungsweise subtrahieren zweier Datenströme. Die Aktion wird dabei sampleweise ausgeführt.

`CLMSFilter` ist eine Implementierung des LMS Algorithmus. Der Parameter μ sowie die Ordnung des Filters können, hierbei angegeben werden.

Analog dazu ist die Klasse `CNLMSFilter` eine Implementierung des NLMS Algorithmus. Die Parameter μ, γ sowie die Ordnung des Filters können hierbei zusätzlich angegeben werden.

3.2.5 Besonderheiten der Implementierung

Das Filterframework arbeitet blockorientiert, wobei die Blockgröße über Definitionen festgelegt wird. Alle Operationen werden also sequentiell auf die geblockten Daten angewandt.

3.2.6 Ein kurzes Beispiel

Um eine Anwendung des Frameworks zu zeigen, soll hier ein kurzer Ausschnitt aus dem Testprogramm folgen:

```
CWAVInput speech;
CWAVInput noise;
speech.openFile(SPEECH);
noise.openFile(NOISE);

CFormat inputformat = noise.getFormat();

// combine speech and noise
CBuffer speech_and_noise;
CAddFilter add;
add.filter(speech, noise, speech_and_noise);
```

Hier wird ein Sprachsignal mit einem Störsignal gemischt. Durch den Aufruf von `filter()` wird die Abarbeitung gestartet. Das Ergebnis steht danach im Puffer `speech_and_noise`.

3.3 Testkonfiguration des C++ Frameworks

Es soll an dieser Stelle der Einsatz des LMS Algorithmus für die Problemstellung *signal enhancement* gezeigt werden. Es gilt, aus einer Mischung von Stör- und Nutzsignal, das Nutzsignal zu extrahieren. Der Aufbau ist in Abbildung 16 dargestellt, welche aus [17] Seite 10 entnommen ist.

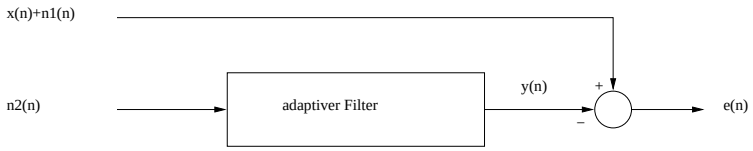

Abbildung 16: signal enhancement mit LMS

Wie ersichtlich ist, werden an den Eingängen des LMS die Signale $x(n)+n_1(n)$ und $n_2(n)$ angelegt. Dabei ist $x(n)$ das Nutzsignal, $n_1(n)$ und $n_2(n)$ sind korrelierte Störsignale.

Betrachtet man nun die Zusammenhänge, so wird die Funktionsweise deutlich (unter Berücksichtigung von $n_1(n) \approx n_2(n)$):

$$y(n) \approx n_1(n)$$
$$e(n) = x(n) + n_1(n) - y(n) = x(n) + n_1(n) - \tilde{n}_1(n) \approx x(n)$$

Während also $y(n)$ eine Annäherung beziehungsweise Nachbildung des Störsignals darstellt, findet sich am Ausgang $e(n)$ das vom Störsignal befreite Nutzsignal.

3.3.1 Resultate

Das vorliegende Framework wurde verwendet, um die Eigenschaften der beiden LMS Algorithmen, in Hinblick auf reale Audiodaten, näher zu beleuchten. Als Nutzsignal wurde dabei eine Datei mit 4.6 Sekunden Länge genommen. Neben einem Sprachsignal findet sich auch eine sinusförmige Schwingung darin. Das Störsignal bildet weißes Rauschen gleicher Länge. Die Konfiguration des Versuches ist im vorhergehenden Abschnitt beschrieben.

Zur Vereinfachung wurden die Signale $n_1(n)$ und $n_2(n)$ gleichgesetzt. Damit ist vollständige Korrelation gewährleistet und man kann sich rein auf das Konvergenzverhalten des Algorithmus – unter idealen Bedingungen – konzentrieren.

Die Abtastrate der Dateien beträgt 44.1 kHz und es wurde für den Test keine Beschränkung der Bandbreite vorgenommen.

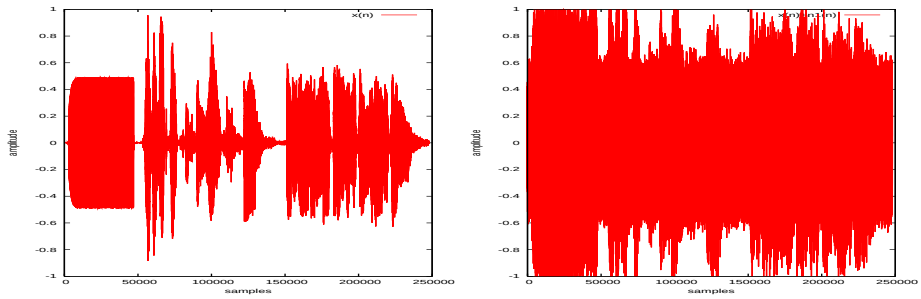

Abbildung 17: Nutzsignal und Summe Nutzsignal+Störsignal

3.3.2 LMS

Zuerst wurde der LMS Algorithmus unter die Lupe genommen. Dabei sollten die Auswirkungen von μ und die Ordnung des Filters beleuchtet werden.

In Abbildung 18 ist die gefilterte Ausgabe auf der linken Seite zu sehen, wobei als Parameter $\mu = 0.1$ und als Filterordnung $N = 1$ gewählt wurden. Wenn man das Nutzsignal aus Abbildung 17 betrachtet, so wird man schnell feststellen, daß der Sinuston recht stark „ausgefranst“ ist.

In Abbildung 18 auf der rechten Seite sind die ersten 1000 Samples des Fehlersignals abgebildet. Wie man erkennen kann, so benötigt der LMS Algorithmus etwa 500 Samples zum Einschwingen.

Bei der verwendeten Abtastrate von 44.1 kHz, entspricht das also einer Zeitspanne von 0.0113 Sekunden. Dies ist ein verschwindend kurzer Zeitraum. Man darf allerdings nicht vergessen, daß die vorgesehene Abtastrate in der Hardware weitaus geringer sein wird. Beispielsweise würde für eine Abtastfrequenz von 8 kHz der Einschwingvorgang etwa 0.0625 Sekunden betragen – also schon fast eine Zehntelsekunde.

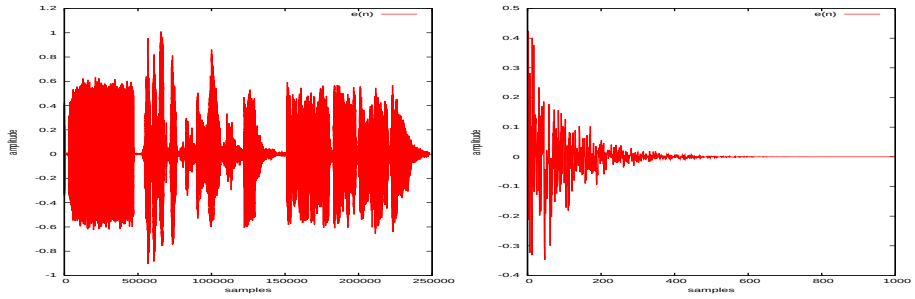

Abbildung 18: C++ LMS mit $\mu = 0.1$, $N = 1$

Im Folgenden soll also nun versucht werden die Qualität der Filterung zu verbessern. Dazu wird als neues μ ein Wert von 0.01 gewählt, also ein Zehntel der ursprünglichen Konfiguration. Die Filterordnung von $N = 1$ wird dabei beibehalten. Das Resultat ist in Abbildung 19 zu finden. Wie

aus der Abbildung zu entnehmen ist, ist der Verlauf der ersten Sinusschwingung viel ruhiger, das Rauschen während der Schwingung ist nun kaum mehr wahrnehmbar. Auf der rechten Seite jedoch erkennt man, daß die Einschwingphase nun viel länger ist. Der Algorithmus benötigt also mehr als 1000 Samples zum Einschwingen.

Dies läßt den Schluß zu, daß bei kleineren Werten von μ der Algorithmus zwar längere Zeit benötigt, das Ergebnis jedoch genauer an den optimalen Koeffizientensatz w_{opt} herankommt.

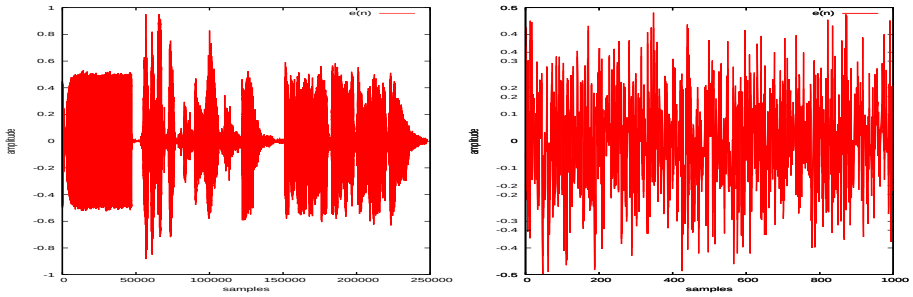

Abbildung 19: C++ LMS mit $\mu = 0.01$, $N = 1$

Zuletzt soll noch untersucht werden, welchen Einfluß die Ordnung des Filters auf das Ergebnis hat. Dazu soll nun die Ordnung N auf 10 erhöht werden. Abbildung 20 zeigt das Ergebnis. Qualitativ ist es die beste Lösung. Das Rauschen ist nun vollständig (das heißt nicht mehr wahrnehmbar) vom Nutzsignal getrennt worden. Dies ist auch auf der rechten Seite erkennbar. Diese stellt die Differenz des Nutzsignals und der gefilterten Ausgabe dar. Der große Nachteil hierbei ist der sehr lange Einschwingvorgang, von ungefähr 40000 Samples, also etwa einer Sekunde. Dies ist auf die Division $\frac{\mu}{N}$ zurückzuführen, welche μ durch die Ordnung 10 dividiert.

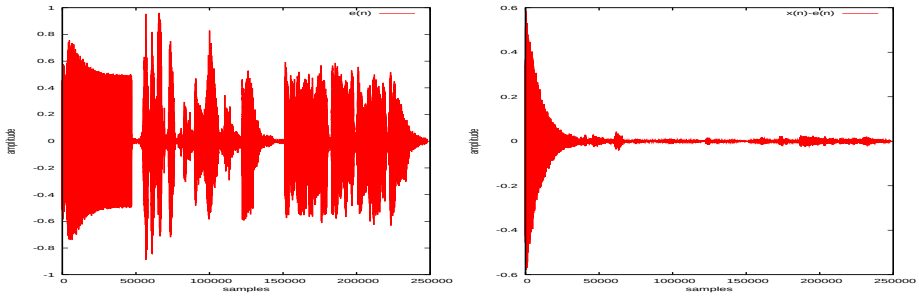

Abbildung 20: C++ LMS mit $\mu = 0.01$, $N = 10$

Wie sich erkennen läßt, ist trotz des geringen rechnerischen Aufwandes, die Qualität des LMS Algorithmus sehr gut. Durch die statische Wahl von μ, kann der Algorithmus jedoch nur sehr schwerfällig auf plötzliche Ände-

rungen reagieren und benötigt für kleine Werte von μ eine lange Zeit zum Einschwingen. Im nächsten Abschnitt wird der NLMS Algorithmus untersucht, welcher genau diese Schwachstelle ausmerzen sollte.

3.3.3 NLMS

Nun sollen die Eigenschaften des NLMS Algorithmus betrachtet werden. Als Basis soll die in Abbildung 20 dargestellte Wertekombination dienen. Der NLMS Algorithmus liefert folgendes Ergebnis (siehe Abbildung 21).

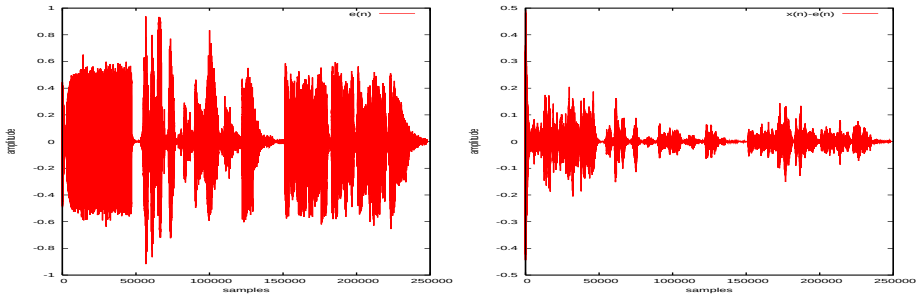

Abbildung 21: C++ NLMS mit $\beta = 0.01$, $N = 10$

Wie man an der rechten Seite erkennen kann, so ist die Einschwingphase wesentlich kürzer, als beim LMS Algorithmus. Die Differenz der Signale jedoch ist wesentlich größer, als beim vergleichbaren LMS Pendant. Eine Verbesserung ergibt sich erst wenn β halbiert wird, also auf 0.005. Das Resultat zeigt Abbildung 22. Hier wird der Vorteil der normalisierten Variante deutlich. Vergleicht man die Abbildungen 20 und 22, so erkennt man, daß der Einschwingvorgang, im Vergleich zum LMS, von etwa 25000 Samples auf etwa 5000 reduziert worden ist – eine Verkürzung um den Faktor 5. Das Differenzsignal liegt dabei in der gleichen Größenordnung.

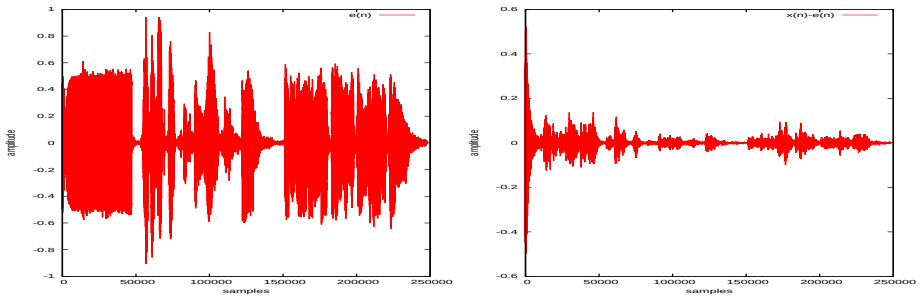

Abbildung 22: C++ NLMS mit $\beta = 0.005$, $N = 10$

Als letztes Beispiel sei die Flexibilität des NLMS Algorithmus demonstriert. Dazu wurde das Störgeräusch in der Amplitude modifiziert. Das

Signal wurde sprunghaft um -3dB, -6dB, +6dB und +3dB verstärkt. Das gefilterte Störsignal, sowie der Verlauf von $\mu(n)$ sind in Abbildung 23 dargestellt. Die Parameter β und N wurden dabei nicht verändert. Wie man anhand der Bilder in Abbildung 23 sehen kann, stellt der Algorithmus $\mu(n)$ entsprechend der Eingangsleistung nach.

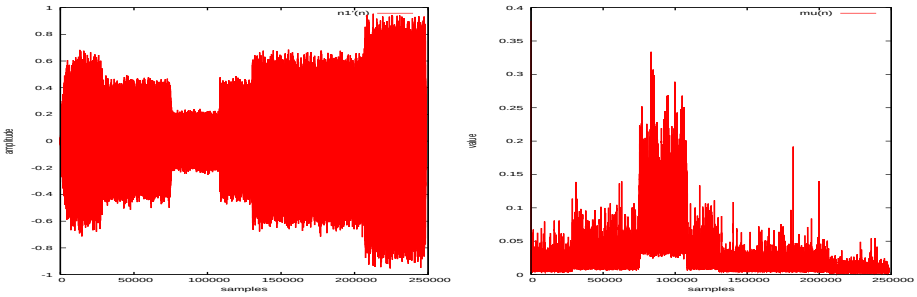

Abbildung 23: Modifiziertes Störsignal und $\mu(n)$

4 Untersuchung des FxLMS Algorithmus

4.1 Allgemeines

Dieser Abschnitt soll sich mit der grundlegenden Analyse der Funktionsweise des FxLMS Algorithmus auseinandersetzen. Dabei sollen die Zusammenhänge zwischen den einzelnen Pfaden beleuchtet werden.

Insbesondere interessant für eine eventuelle Realisierung, sind die Qualität der Abschätzung des Sekundärpfades, sowie der Einfluß einer sich ändernden Übertragungsfunktion.

Ein weiterer Punkt ist die Untersuchung bezüglich der Phasenlage. Es ist zu bestimmen, wie weit das Signal verzögert sein kann und welchen Einfluß es auf das resultierende Gegenschallsignal hat.

Diese Experimente werden unter Zuhilfenahme der im Abschnitt 3.1 vorgestellten Funktionsblöcke durchgeführt. Die C++ Implementierung läßt leider keine Realisierung einer geschlossenen Regelschleife zu, da eine aktive Rückführung der Ausgangssignale an den Eingang des LMS-Objektes notwendig wäre. Diese ist jedoch nicht implementiert.

4.2 OCTAVE Implementierung

Es werden Primär- und Sekundärpfad modelliert. In erster Näherung wurde dabei auf eine Kalibrierung der Abschätzung $\hat{S}(z)$ verzichtet und stattdessen eine Kopie der Koeffizienten verwendet.

Der Kern der Implementierung ist in der `for` Schleife zu finden, welche an dieser Stelle noch einmal genauer betrachtet werden soll:

```
% index of current sample is n
for n = 1:length(num_samples),
  % step 1: calculate C(z) => xf
  [fir_c_hist, xf] = fir(fir_c_hist, fir_c_coef, x(n));

  % step 2: calculate LMS coeffcient update based on xf and e
  %         compute W(z) based on x => y
  [lms_w_coef, lms_w_hist, y] =
    lms(lms_w_hist, lms_w_coef, x(n), xf, e);

  % step 2a:
  % invert anti-noise
  y = -y;

  % step 3: compute P(z) => d
  [fir_p_hist, d] = fir(fir_p_hist, fir_p_coef, x(n));

  % step 4: compute H(z) => yf
```

```
  [fir_h_hist, yf] = fir(fir_h_hist, fir_h_coef, y);

  % step 5: compute residual error => e
  e = d + yf;
end;
```

Der Grundgedanke der Implementierung bestand darin, den Zusammenhang der Pfade deutlich zu machen. Als Basis, beziehungsweise „Eingabe“, dient der Vektor $x(n)$. Dieser bildet den Referenzschall.

Der erste Schritt ist die Berechnung des Filterwertes für die Sekundärpfadabschätzung. Der so entstandene Wert `xf` wird nun dem Regelteil des Hauptfilters zugeführt. Der Filterteil wird jedoch mit dem ungefilterten Eingangssignal versorgt. Als Zwischenschritt wird das Ausgabesignal `y` des Filters invertiert. Dieser Schritt ist notwendig, da der reine LMS Algorithmus dazu nicht in der Lage ist.

Damit ist die Antischallgenerierung abgeschlossen. Nun wird über den Primärpfad, welcher ja vom Hauptfilter nachgebildet werden soll, das Eingangssignal verändert. Zugleich wird auch das Gegenschallsignal einer weiteren Filterung unterworfen – nämlich durch den Sekundärpfad.

Den Abschluß bildet die (akustische) Addition beider Signale. Der verbleibende Signalteil ist der Fehler, welcher im nächsten Durchlauf wieder an den adaptiven Algorithmus des Hauptfilters gereicht wird.

4.3 Resultate

Als Basisexperiment soll als Eingangssignal eine Sinusschwingung mit Amplitude 0.7 und einer Periode von 100 Samples dienen. Der Primärpfad wird außer Acht gelassen – Allpass, Verzögerung gleich eins (dies ist der minimal erreichbare Wert). Der Sekundärpfad (und natürlich auch seine Kopie) ist ebenfalls ein Allpass, wiederum mit Verzögerung gleich eins.

In Abbildung 24 sind die Signalverläufe für die minimale Phasenverschiebung von einem Sample zu sehen. Auf der y-Achse ist die Amplitude des Signals aufgetragen, währen sich auf der x-Achse die Samples finden. Wie man erkennen kann, wird das Fehlersignal $e(n)$ schon nach wenigen Durchläufen (sprich Samples) Null. Die Gegenschallwellen $y(n)$ und $yf(n)$ lassen sehr schön den gewünschten Verlauf erkennen.

4.4 Probleme

In diesem Abschnitt soll über die Simulationen gezeigt werden, an welchen Stellen Probleme auftreten können. Die Simulationen zeichnen ein *sehr* vereinfachtes Bild der Realität. Damit kann also geschlossen werden, daß sobald die Simulation Probleme zeigt, bei einer Implementierung, diese Probleme zu erwarten sind, wenn nicht sogar in gravierenderer Form.

Die Betrachtungen sollen sich auf folgende Fälle erstrecken:

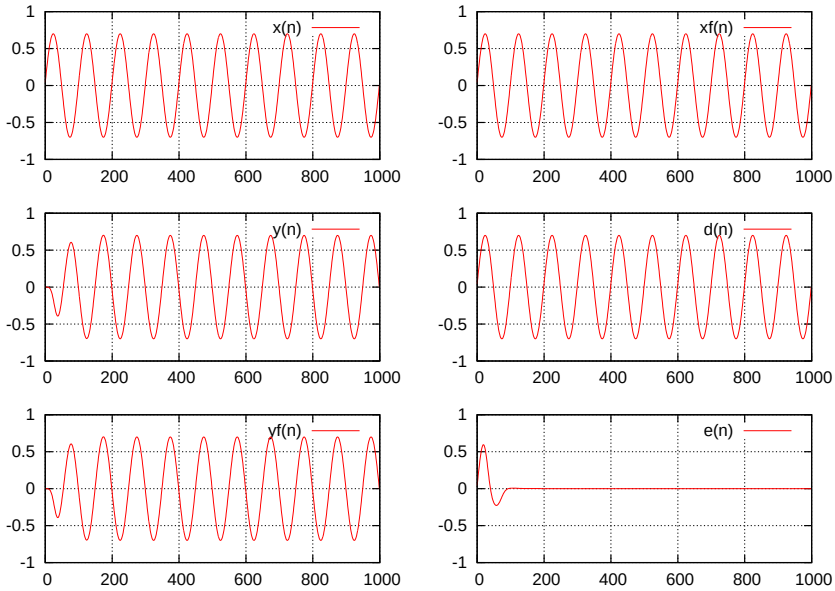

Abbildung 24: FxLMS - Verzögerung 1

- Primärpfad
- Sekundärpfad (mit Sekundärpfad gleich Abschätzung)
- Sekundärpfad (Verschiebung in Phase zwischen Abschätzung und realem Pfad)
- Qualität der gelieferten Eingangssignale

Als adaptiver Filter wurde der LMS Algorithmus mit einer Schrittweite von $\mu = 0.35$ gewählt. Wie sich zeigen wird, ist diese Wahl nicht immer klug.

4.4.1 Primärpfad - Phase

Für diese Betrachtung wurde der Sekundärpfad eliminiert – die Verzögerung auf den Wert eins gesetzt. Damit reduziert sich der FxLMS Algorithmus zu einer simplen Systemidentifikation.

Wie in Abschnitt 3.1.4 gezeigt, hängt die Adaptionsfähigkeit für eine gegebene Phase lediglich von der Anzahl zur Verfügung stehender Taps – also der Filterordnung ab. Für eine hinreichende Anzahl von Taps konvergiert der Algorithmus also immer.

4.4.2 Sekundärpfad - Phase

In diesem Abschnitt sollen nun verschiedene Werte für die Phasenlage des Sekundärpfades getestet werden. Dabei wird lediglich die Auswirkung der Verzögerung betrachtet.

Erhöht man schrittweise die Phasenverschiebung, so wird man feststellen, daß bereits bei einer Verzögerung von 7 Samples die Fehlerfunktion länger nachschwingt. Der Knackpunkt ist schließlich bei 8 Samples erreicht. Man kann sehen, wie die Signale $y(n)$ und $e(n)$ in den negativen Bereich abdriften. Dies ist in Abbildung 25 dargestellt.

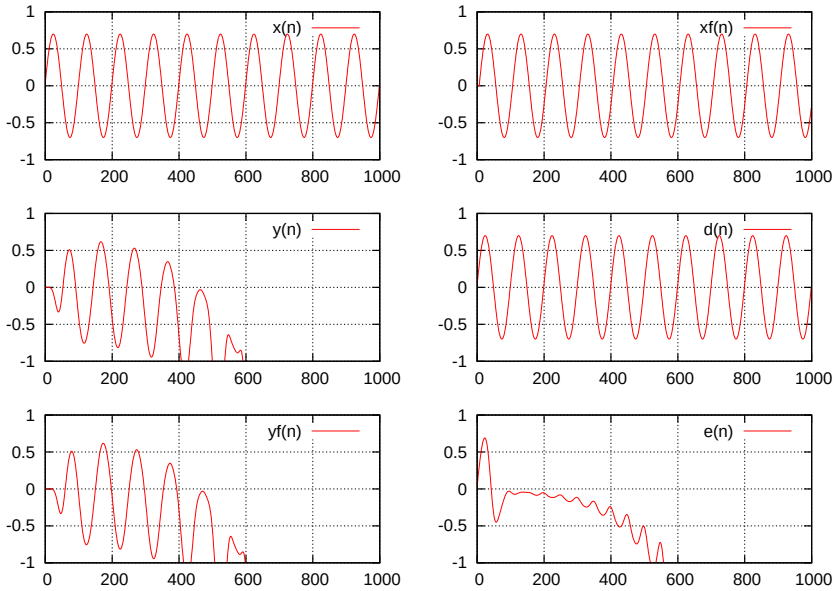

Abbildung 25: FxLMS - Verzögerung 8

Erhöht man die Phasenlage weiter, so wird die Divergenz beschleunigt und der Algorithmus wird vollkommen unbrauchbar. Die Verzögerung entspricht dabei einer Phase in Bezug auf die verwendete Frequenz von 28°.

An dieser Stelle zeigt sich ein großes Problem des hier implementierten FxLMS Algorithmus: Der Sekundärpfad darf nur eine minimale Phasenverschiebung aufweisen. Ansonsten divergiert der Algorithmus extrem schnell und kommt seiner eigentlichen Aufgabe in keinster Weise nach.

4.4.3 Phasenlage Sekundärpfad - Sekundärpfad Abschätzung

Nun soll die Auswirkung der Phasenlage des Sekundärpfades und seiner Abschätzung betrachtet werden. Dazu soll der Primärpfad wiederum nicht berücksichtigt werden, ebenso der Sekundärpfad.

Lediglich die Abschätzung soll nun einer Verzögerung unterworfen werden. In Abbildung 26 ist das Ergebnis dargestellt. Dabei wurde als Phasenlage die Abschätzung um 28° (also 8 Samples) verschoben.

Wie klar zu erkennen ist, treten sehr ähnliche Probleme wie bei der Verzögerung des Sekundärpfades im vorherigen Abschnitt auf.

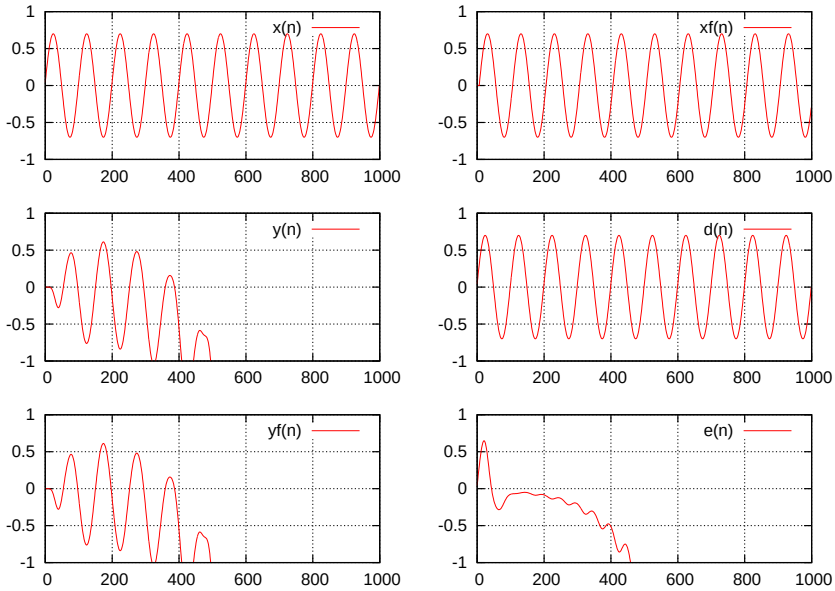

Abbildung 26: FxLMS - Sekundärpfad mit Phasenverschiebung

4.4.4 Entschärfung des Problems

Wie es scheint ist der FxLMS mit großen Problemen behaftet. Doch es gibt auch eine Lösung um der Divergenz des Algorithmus entgegenzuwirken. Die Lösung ist in [19] zu finden.

$$\mu_{max} = \frac{1}{P_{avrg_{x'}}(N + \Delta)} \tag{4.1}$$

Gleichung 4.1 beschreibt die Antwort auf die vorhandenen Probleme. Der Parameter, welcher die Adaptionsgeschwindigkeit beeinflußt – μ – wird begrenzt. Zusätzlich zu der in Gleichung 2.9 auf Seite 24 definierten Bedingung wird Δ in die Berechnung eingeführt. Dieser Wert gibt die maximal erwartete Phasenverschiebung in Samples des Sekundärpfades an.

Zum Testen wird folgendes Setup verwendet:

- LMS Filter
- Anzahl der Taps (N): 128
- Sinus-Schwingung mit Periodendauer von 100 Samples und Amplitude 0.7
- Verschiebung des Sekundärpfades (Δ): 25 Samples (entspricht 90° Phase)

- Primärpfad weist keine Phasenverschiebung auf

Damit kann man μ_{max} für die angegebene Sinusschwingung, wie folgt berechnen:

$$P_{avrg_{x'(n)}} = \frac{1}{N}\sum_{i=0}^{128} x_n(i)^2 = 0.1914$$

$$\mu_{max} = \frac{1}{0.1914(128+25)} = 0.0341$$

In Experimenten konnte jedoch nachgewiesen werden, daß die angegebene Parametrisierung, für diese Werte konvergiert. Als Beispiel soll das Resultat für einen Wert von 0.0341 für μ gezeigt werden. Die zugehörige Abbildung 27 zeigt die Signalverläufe.

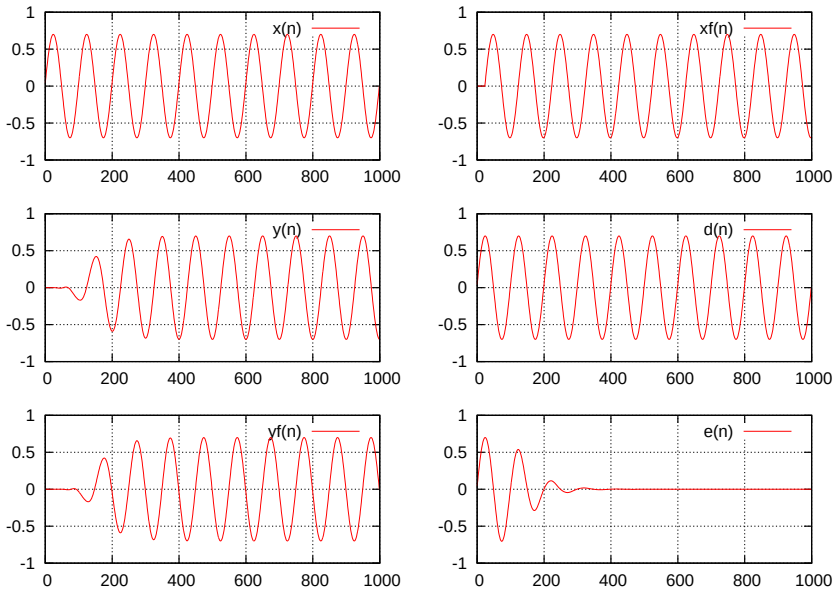

Abbildung 27: FxLMS - korrigiertes $\mu = 0.05$

Die hohe Verzögerung wirkt sich also, im Gegensatz zu Experimenten, mit größerem μ nicht aus. Selbst bei weiterer Steigerung, bis hin zu einer Phasendifferenz von 180° konvergiert der Algorithmus immer noch. Jedoch wird die Zeit, die der Algorithmus benötigt um zu konvergieren, entprechend länger. Sind beim obigen Beispiel nur etwa 250 Samples nötig um einen Fehler von annähernd Null zu erhalten, sind es bei 50 Samples Verzögerung schon knapp über 400 Samples.

Zusätzlich muß bemerkt werden, daß die Phasenverschiebung von der Frequenz abhängt. Es gilt also: je höher die maximale Frequenz, welche

bedämpft werden soll, ist, desto geringer muß die Phasenverschiebung des Sekundärpfades sein. Ansonsten wird der Algorithmus nicht konvergieren.

In den Versuchen konnte gezeigt werden, daß mit weißem Rauschen als Eingangssignal selbst nach 10000 Samples keine merkliche Dämpfung zu erzielen ist. Es ist also unbedingt erforderlich die Eingangsfrequenzen auf einen angemessenen Wert zu begrenzen.

4.4.5 Einsatz des NLMS Algorithmus in Hinblick auf Δ

Die im vorherigen Abschnitt gezeigte Tatsache wirkt sich auch direkt auf den Einsatz des NLMS Algorithmus, als apdaptiven Filter im FxLMS, aus. Wie gezeigt, wird durch den NLMS der Parameter μ auf den höchsten Wert bei dem der Algorithmus konvergiert gestellt.

Dies ist nun insofern ein Problem, da der Parameter Δ (der ja die Phasenverschiebung des Sekundärpfades wiederspiegelt) nicht in die Berechnung einfließt. Ohne die Einbindung von Δ divergiert der Algorithmus sehr schnell. Es muß also sichergestellt werden, daß der Parameter in die Berechnung von $\mu(n)$ aufgenommen werden muß. Dies kann auf zwei Wege geschehen:

- direkte Einbindung in die Division des Koeffizientenupdates
- Begrenzung des berechneten μ auf den maximal zugelassenen Wert μ_{max}

Die erste Methode bringt den zusätzlichen Aufwand einer Addition und Multiplikation. Aufgrund der verfügbaren Rechenzeit, stellt dies kein Problem dar. In Hinblick auf einen FPGA würde es jedoch wiederum einiges an zusätzlicher Logik (gleichbedeutend mit Hardware) bedeuten.

Die zweite Methode würde auf einem FPGA viel Logik nach sich ziehen, da Vergleiche teuer sind – sprich viele Gatter benötigen. Als Alternative könnte man jedoch eine Bitmaske aufsetzen und das berechnete μ mit dieser Maske über ein logisches UND verknüpfen. Dies ist in Bezug auf generierte Hardware sehr billig, hat jedoch den Nachteil, daß der Begrenzungswert nur in ganzen negativen Potenzen von 2 angegeben werden kann. Beispielsweise könnte man einen Wert von 0.0625, also 2^{-4}, als Bitmaske für eine $4.x$ Fixkommazahl (mit $x \geq 4$), als `0000.0001` angeben. Für Werte von $x > 4$ werden die verbleibenden Stellen mit `1` aufgefüllt.

Durch die UND Verknüpfung würden die Bits ausmaskiert werden, welche zu einer Fixkommazahl größer der geforderten führen würden. Das Rechnen mit Fixkommazahlen wird im nächsten Abschnitt genauer erklärt.

Es ist jedoch auf jeden Fall eine Modifikation durchzuführen, oder ein LMS Algorithmus mit entprechend angepasstem μ einzusetzen.

4.4.6 Qualität der anliegenden Signale

Eine weitere Fehlerquelle kann die Qualität der Eingangssignale darstellen. Dies ist besonders in Hinblick auf eine Realisierung in Hardware interessant. Dabei können folgende zwei Fälle (auch kombiniert) als Fehler auftreten:

- Anliegen einer Offsetspannung am Eingang des ADCs
- elektrische Störung des Eingangssignals (Rauschen)

Zwecks Analyse der Problematik wurde das OCTAVE-Skript aus Abschnitt 4.2 modifiziert, um Überlagerungen an den Signalen zu simulieren. Dabei kann nun ein konstanter Offset, sowie ein Rauschsignal zum Testen verwendet werden.

Als Grundeinstellung wurde ein sinusförmiges Signal für $x(n)$, mit einer Periode von 600 Samples und einer Amplitude von 0.7 gewählt. Die hohe Periodendauer wurde gewählt, um Störungen besser sichtbar zu machen. Der Primärpfad weist wiederum keine Phasenverschiebung auf, womit $d(n) = x(n)$ gilt. Der Sekundärpfad weist eine Verzögerung von 15 Samples auf.

Der erste Versuch bestand darin, dem Fehlersignal $e(n)$ einen additiven Offset von 0.1 und ein weißes Rauschen mit Amplitude 0.2 aufzuprägen. Die Wahl dieser Werte stellt sicher, daß das Signal den Wertebereich von ± 1 nicht verläßt.

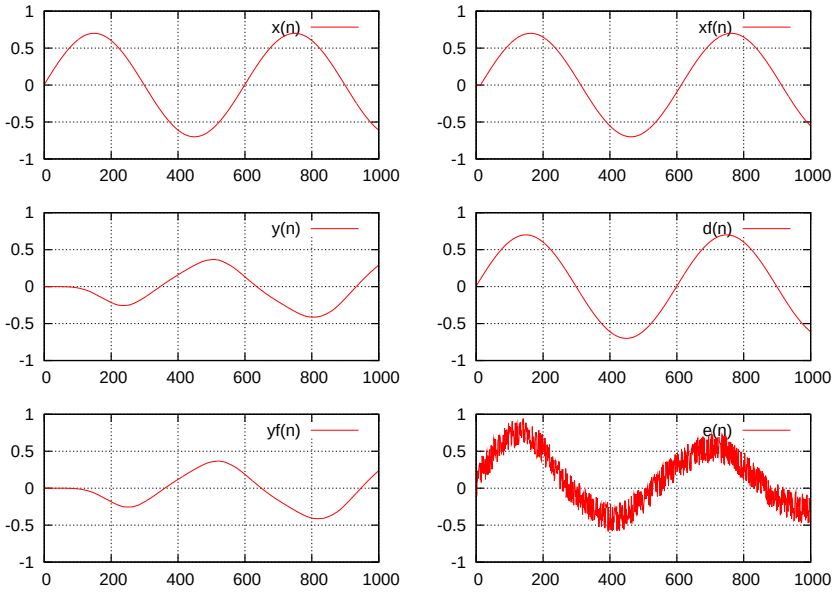

Abbildung 28: FxLMS - Gestörtes $e(n)$

Abbildung 28 zeigt das Ergebnis des Versuches. Wie man an der Graphik (insbesondere am Signal $y(n)$) erkennen kann, beeinflussen Störungen

die Konvergenzfähigkeit des Algorithmus nicht. Das Rauschen läßt sich lediglich ganz leicht am Signalverlauf erkennen. Dies fällt jedoch auch nur bei entsprechend langer Periodendauer auf.

Der nächste Versuch untersucht zusätzlich die Auswirkungen eines gestörten Eingangssignales $x(n)$. Als Störsignal wurde weißes Rauschen, wie im vorherigen Versuch verwendet. Dabei ist anzumerken, daß es sich dabei um verschiedene Rauschsignale handelt, welche nicht zueinander in Bezug stehen.

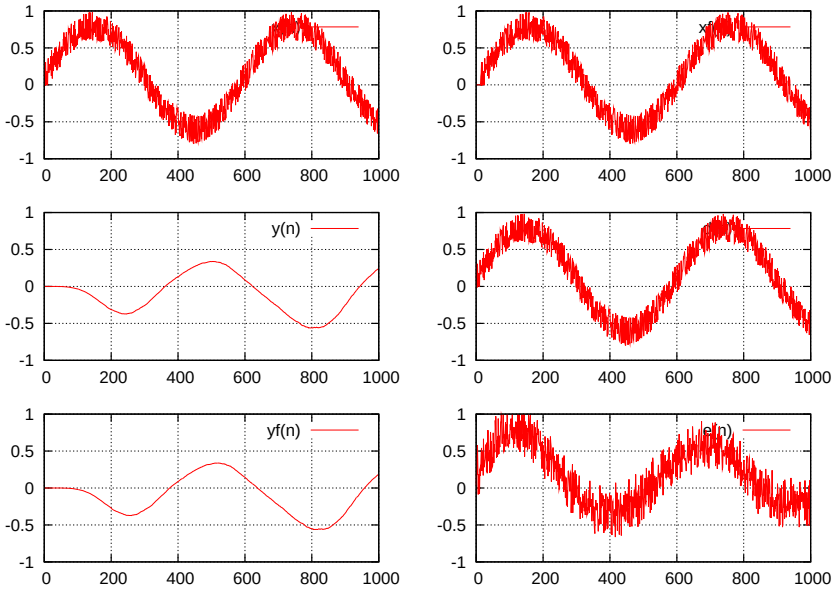

Abbildung 29: FxLMS - Gestörtes $x(n)$ und $e(n)$

Auch hier zeigt das Ergebnis in Abbildung 29, keine Beeinträchtigung der Konvergenz. Es fällt jedoch auf, daß das tatsächliche Fehlersignal die Störungen des Eingangssignales übernimmt. Der Grund hierfür liegt, wie in Abschnitt 4.4.4 gezeigt, in der hohen Frequenz und fehlenden Periodizität des Rauschens. Der LMS ist nicht mehr in der Lage dieses Störsignal zu beseitigen. Damit bleibt es als Fehler erhalten.

5 Implementierung in VHDL

5.1 Fixpunktarithmetik

Aufgrund der weitaus geringeren Komplexität als bei der Gleitpunktdarstellung wurde als Zahlendarstellung die Fixpunktdarstellung gewählt. Eine Fixpunktzahl besteht aus einer zuvor festgelegten Anzahl an Vor- und Nachkommastellen (QI und QF als Bits). Die Gesamtlänge WL ergibt sich aus der Summe der Stellen, also $WL = QI + QF$. Da die Position des Komma bereits implizit festgelegt ist, wird dafür kein Bit benötigt.

Der nächste Schritt ist die Festlegung der Stellen. Durch die Anzahl der Bits wird der mögliche Wertebereich definiert. Für den Ganzzahlanteil ist die Rechnung recht einfach: Die größte darstellbare Ganzzahl ergibt sich aus $2^{QI} - 1$, für vorzeichenlose Zahlen. Sollen die Zahlen vorzeichenbehaftet sein, so muß man ein Bit zur Speicherung des Vorzeichens vorsehen und die größte Ganzzahl α ergibt sich wie in Gleichung 5.1 dargestellt.

$$-2^{QI-1} \leq \alpha \leq 2^{QI-1} - 1 \tag{5.1}$$

Der Nachkommateil QF ist etwas komplizierter. Wie schon erwähnt besteht er aus den verbleibenden Bits der Wortlänge WL. Die Anzahl der Bits bestimmt die Auflösung ϵ der Nachkommastellen. Diese berechnet sich laut Gleichung 5.2.

$$\epsilon = \frac{1}{2^{QF}} \tag{5.2}$$

Setzt man beide Teile zusammen, so erhält man folgende Beziehung (Gleichung 5.3) für die darstellbaren Zahlen α.

$$-2^{QI-1} \leq \alpha \leq 2^{QI-1} - 2^{-QF} \tag{5.3}$$

[20] zeigt weiters einige Beispiele zum Rechnen mit Fixpunktzahlen mit versetzten Dezimalstellen. Darauf soll hier jedoch nicht weiter eingegangen werden, da das festgelegte Zahlenformat für *alle* verwendeten Zahlen gelten wird. Die Notation Q$I.F$ soll jedoch übernommen werden, um Fixkommazahlen mit I Ganzzahlstellen, sowie F Nachkommastellen zu bezeichnen. Es soll auch an dieser Stelle festgelegt werden, daß für den Ganzzahlanteil I bereits das Vorzeichenbit hält - also errechnet sich Anzahl der Bits für die Ganzzahlen eigentlich zu $I - 1$.

Nun sollen die drei benutzten Grundrechenarten besprochen werden, da diese für die Fixkommadarstellung implementiert werden müssen. Im Projekt sind diese im Paket `fixpoint_arith` zu finden.

5.1.1 Der Datentyp

Als Datentyp kommt hierbei der Typ `std_logic_vector` zum Einsatz. Die Wertigkeit der einzelnen Stellen wird dabei durch die Konstanten `QI` und `QF` definiert. Dadurch kann eine Fixkommazahl folgendermaßen auf einen `std_logic_vector` abgebildet werden (für eine 4.12 Fixkommazahl):

$$S \underbrace{III}_{QI-1} . \underbrace{FFFFFFFFFFFF}_{QF} \tag{5.4}$$

S ist dabei das Vorzeichenbit, I sind die Bits, welche für die Ganzzahlen benötigt werden und F die Nachkommabits.

5.1.2 Addition und Subtraktion

Die Summe (oder die Differenz) zweier Fixpunktzahlen Q$I.F$ ergibt wiederum eine Zahl mit Q$I.F$. Es ist jedoch zu beachten, daß ein Überlauf auftreten kann. Um nun zwei Zahlen zu addieren, beziehungsweise zu subtrahieren, wird ein Addierer benötigt. Dieser ist als Funktion `FRACADD` (beziehungsweise `FRACSUB` für die Subtraktion) zu finden.

Grundsätzlich ist es möglich die Addition über Kaskadierung mehrerer *Volladierer* auszuführen (unter Berücksichtigung der Regeln in Gleichung 5.5ff).

$$s = i0 \oplus i1 \oplus ci \tag{5.5}$$

$$co = (i0 \wedge i1) \vee (i0 \wedge ci) \vee (i1 \wedge ci) \tag{5.6}$$

Das eingesetzte *Stratix II* FPGA bietet jedoch intern fertige Addierer für Binärarithmetik an. Durch die Verwendung der Datentypen `signed` und `unsigned`, ist es möglich, diese direkt im VHDL-Code zu instanzieren und zu verwenden. Dies kann mittels type-casts erreicht werden. Dabei wird die interne Struktur des Datentypes nicht beachtet. Dies ist jedoch durch die Aufteilung der Bits nicht notwendig.

Subtraktion kann sehr einfach, unter Verwendung der *Zweierkomplements*, durchgeführt werden. Um positive Zahlen in die Zweierkomplementdarstellung umzurechnen, gibt es zwei Möglichkeiten (siehe auch [21] Seite 87ff):

- Das Einerkomplement der Zahl berechnen und 1 addieren.
- Vom LSB alle Ziffern bis einschließlich der ersten '1' kopieren und für den Rest das Einerkomplement bilden.

Das Zweierkomplement erlaubt eine Darstellung negativer Zahlen. Damit erklärt sich nun auch die Subtraktion. Sie ist eine Addition, wobei der

zweite Summand vor der Berechnung, in die Zweierkomplementdarstellung umgerechnet wird. Danach wird einfach eine Addition auf die Summanden angewendet. Die Komplementbildung wird durch die Funktion `COMP` bereitgestellt.

5.1.3 Multiplikation

Multiplikation kann als eine Folge von Addition und Schiebeoperationen aufgefasst werden. Dabei ist jedoch zu beachten, das Q$x.y$ Zahlen als Ergebnis eine Zahl der Gestalt Q$2x.2y$ liefern. Das heißt, daß die Zwischendarstellung mit einer höheren Genauigkeit arbeiten muß. Durch *Skalierung* kann das Zwischenergebnis wieder in eine Q$x.y$ Zahl zurückgewandelt werden. Betrachten wir dazu ein Beispiel:

Gegeben seien zwei Q1.5 Zahlen, $\alpha = 0.01010$ und $\beta = 0.10001$. Zuerst wird eine Multiplikation der beiden Werte durchgeführt und man erhält $\alpha * \beta = \gamma = 00.0010101010$. Dabei ist es wichtig festzustellen, daß es sich dabei um eine Q2.10 Zahl handelt. Um diese wieder auf eine Q1.5 Zahl zu reduzieren, wird γ um 5 Bits nach rechts geschoben. Damit erhält man $\gamma' = 00.00101$, also eine Q2.5 Zahl. Verwirft man nun das erste Bit, so erhält man die gewünschte Zahl $\gamma'' = 0.00101$. Durch das Verwerfen von Stellen, ist das Ergebnis natürlich bis zu einem gewissen Grad ungenau. Dies muß bei Fixkommazahlen in Kauf genommen werden, sollte jedoch in der Implementierung keine Schwierigkeiten bereiten.

Eine weitere Besonderheit ist die Behandlung des Vorzeichenbits. Vor der Multiplikation kann anhand der Vorzeichen der Faktoren festgestellt werden, welches Vorzeichen das Ergebnis haben wird. Die Gleichung 5.7 zeigt, wie das Vorzeichen mittels exklusiv-oder ermittelt werden kann.

$$s_{Ergebnis} = s_A \oplus s_B \tag{5.7}$$

Nachdem man nun das Vorzeichen kennt, kann man beide Faktoren in die positive Darstellung umwandeln. Je nach errechnetem Vorzeichen muß, nach Abschluß der Operation, das Ergebnis wieder in die Zweierkomplementdarstellung transformiert werden (sobald das Ergebnis vorzeichenbehaftet ist).

Die Implementierung benutzt wiederum die zur Verfügung gestellte Funktionalität des `unsigned` Datentyps. Die Speicherung des Vorzeichens und die Anpassung des Ergebnisses werden von der Funktion `FRACMULT` zur Verfügung gestellt.

5.1.4 Division

Die Division kann meist nur durch einen verhältnismäßig hohen Hardwareaufwand realisiert werden. Es existieren jedoch zwei Möglichkeiten, um den Rechenaufwand zu reduzieren:

- Verschiebung um die Wertigkeit des ersten Bit ungleich Null
- Einsatz eines ROM Bausteines, welcher die Kehrwerte für alle möglichen Eingangswerte enthält

Die erste Methode benutzt die Tatsache, daß eine Verschiebung der Bits des Dividenden um die Wertigkeit des ersten Bits im Divisor ungleich Null, eine angenäherte Division dargstellt. Durch Verschieben der Bits des Dividenden um n Stellen, kann eine Division durch 2^n erreicht werden. Der große Nachteil dieser Methode ist die Ungenauigkeit, welche durch das Weglassen der niedrigeren Stellen entsteht.

Die zweite Methode basiert auf der Umformulierung der Division wie in Gleichung 5.8:

$$\frac{a}{b} = a\frac{1}{b} \tag{5.8}$$

Dabei ist es möglich den Ausdruck $\frac{1}{b}$ im Voraus zu berechnen und als ROM Baustein abzulegen. Dazu wird für alle möglichen Werte für b der zugehörige Wert $\frac{1}{b}$ berechnet. Dies kann zum Beispiel im *Stratix* FPGA als asynchrones ROM synthetisiert werden. Dabei bezeichnet der Wert b die Adresse, an welcher das Resultat hinterlegt ist.

Dabei ist jedoch zu beachten, daß der Zugriff über Signale erfolgt und damit zwischen `process`, beziehungsweise `wait` Statements gepackt werden muß. Dies führt zu einer längeren Laufzeit der Berechnung, da der Zugriff auf die Speicher abgewartet werden muß. Weiters ist auch der Aufwand an Hardware nicht zu vernachlässigen. Beispielsweise benötigt eine Wortbreite von 16 Bit $2^{16} = 65536$ Einträge – dies ist bereits der maximal zulässige Wert für das Stratix FPGA. Zudem steigt dieser Aufwand für höhere Wortbreiten exponentiell.

Aus diesem Grund und aufgrund der Tatsache, daß es sich um einen Regler handelt, welcher in der Lage ist die Ungenauigkeiten zu kompensieren, wurde von der Implementierung eines ROMs Abstand genommen und die zwar weniger genaue, aber wesentlich effizientere Methode der Verschiebung der Bits des Dividenden gewählt.

5.2 Generierung von Pseudozufallszahlen

Zur Kalibrierung der Sekundärpfadabschätzung (siehe Abschnitt 2.5.7), ist es notwendig einen Rauschgenerator zu implementieren. Da Rauschen aber im Grunde genommen eine möglichst zufällige Folge von Zahlen ist, kann man sich eines Zufallszahlengenerators bedienen, um *weißes Rauschen* (jede Frequenz mit gleicher Amplitude vertreten) zu erzeugen.

Den Grundgedanken liefert der Algorithmus aus [22], welcher auf der Methode der „*primitiven Polynome modulo 2*“ beruht. Dabei wird die Eigenschaft besonderer Polynome ausgenutzt Bitfolgen zu permutieren und

zwar in einer Weise, daß alle möglichen Kombinationen durchlaufen werden. Dabei handelt es sich aber um eine deterministische Folge – das heißt, daß die Kombinationen immer in der selben Reihenfolge erzeugt werden.

Die maximale Anzahl von Permutationen für n Stellen ergibt sich aus $2^n - 1$. Polynome die diese Anzahl erreichen müssen *primitiv prim* sein. Dies bedeutet, daß das Polynom $p_n(x)$ ein Faktor des Polynoms $x^n + 1$ sein muß. Es darf dabei jedoch *nur* von diesem speziellen n ein Faktor sein und keinem kleineren n!

Den Ursprung bildet der *seed* – eine Zahl ungleich Null, welche als Ausgangspunkt für alle folgenden Pseudozufallszahlen dient. Es ist jedoch wichtig festzuhalten, daß das erzeugte Muster wiederum deterministisch ist. Der *seed* legt lediglich den Einsprungpunkt in die Folge fest. Übliche Methoden zum Randomisieren des *seeds* sind Berechnungen aus der aktuellen Zahl von Ticks oder Prozesskennummern. Hier ist der Wert jedoch fix vorgegeben.

Bei diesem Projekt werden Pseudozufallszahlen mit der Länge von 12 Bit benötigt. Die ersten Versuche wiesen jedoch einen erheblichen Schwachpunkt auf: Die maximale Anzahl von Permutation von $2^{12} - 1 = 4095$ erwies sich als zu kurz um als Rauschen wahrgenommen zu werden.

Der Grund hierfür liegt in der Bitlänge. Betrachtet man zum Beispiel die Sampling-Rate einer CD – also 44kHz – unter der Annahme, daß zu jedem Zeitpunkt eine neue „Zufallszahl“ errechnet wird, dauert eine vollständige Permutation für 12-Bit Vektoren:

$$t_{perm} = \frac{4095}{44100} = 93ms \hat{=} 10.769Hz \tag{5.9}$$

Dies ist für den Menschen nicht mehr als Rauschen erkennbar, da sich die Periode der Pseudozufallszahlen bestenfalls als verrauschtes Brummen erkennbar macht. Abhilfe schafft hier die interne Berechnung einer viel längeren Bitfolge – zum Beispiel mit 24 Bit. Von dieser Folge werden jedoch nur die unteren 12-Bit als Pseudozufallszahl ausgelesen.

Der Vorteil liegt auf der Hand. Durch die höhere Bitlänge ist die Periodendauer viel höher – die Wiederholungsfrequenz ist viel niedriger. Damit wird ein monotones Rauschen erzeugt. Man verliert jedoch die Eigenschaft, daß jede Folge *genau einmal* vorkommt. Abbildung 30 stellt die Fourieranalyse des erzeugten Signals dar.

Für tatsächliches weißes Rauschen müßte der obere Abschluß der Kurve eine horizontale Linie bilden. Man kann jedoch sehen, daß der Verlauf nur nahezu horizontal ist und sich im Bereich von -10 bis -20dB bewegt (die Analyse wurde unter Zuhilfenahme des Programmes *Audacity* durchgeführt). Damit ist klar, daß es sich um kein reines, weißes Rauschen handelt. Für Kalibrierungszwecke ist die Qualität jedoch ausreichend.

Für diesen Rauschgenerator wurde das Polynom aus Gleichung 5.10 verwendet um eine Bitfolge mit 24 Stellen zu permutieren. Weitere Polynome für andere Bitlängen (bis 100) sind in [22] zu finden.

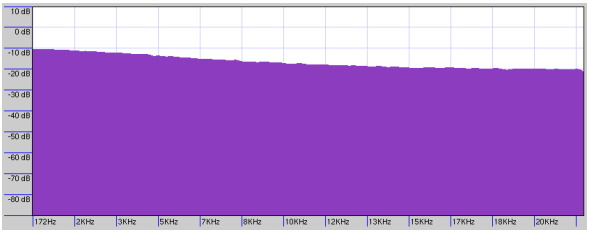

Abbildung 30: Fourier-Transformation des Rauschgenerators

$$p_{24}(x) = x^{23} + x^5 + x^0 \tag{5.10}$$

Der Algorithmus für die Durchführung einer Berechnung ist sehr einfach beschrieben:

- Initialisierung: Ein Bitvektor x mit einer Folge aus Nullen und Einsen, wobei zumindest eine Eins enthalten sein muß
- Berechne den Wert x'_0 durch Anwendung des Polynomials aus 5.10 (für 12 Bit Werte), indem die Exponenten als Stellenangaben im Bitvektor interpretiert werden und durch *exklusiv-oder* verknüpft werden
- Schiebe die Stellen des Bitvektors um eine Stelle nach links und hänge x'_0 an die geschobene Bitfolge x an

Die Realisierung läßt sich sehr elegant in VHDL formulieren (siehe auch den Quellcode zum Entity `RNG`). `random` bezeichnet dabei den Bitvektor, den es zu permutieren gilt.

```
-- XORs
bit :=  random(23) xor
        random(5)  xor
        random(0);
-- shifting
random := random(22 downto 0) & bit;
```

5.3 FIR-Filter in VHDL

FIR-Filter werden hier zwar nicht direkt eingesetzt, sind jedoch integraler Bestandteil des LMS Algorithmus. Es wurden zwei Architekturen implementiert.

- `FIR_sim`
- `FIR`

`FIR_sim` dient lediglich zur Simulation des FIR-Filters. Berechnungen werden in einem Schritt durchgeführt.

Dies ist jedoch für die Realisierung des FIR-Filters in einer Schaltung nur begrenzt möglich, da nur eine relativ geringe Anzahl von DSP-Elementen zur Verfügung steht. DSP-Elemente sind spezialisierte, bereits in Hardware vorhandene Einheiten, wie zum Beispiel Addierer und Multiplizierer. Würde man nun die `FIR_sim` Architektur synthetisieren, so würde für jede Multiplikation ein DSP-Element benötigt werden. Bei hohen Filterordnungen würde die Anzahl der benötigten Elemente, die Anzahl der tatsächlich vorhandenen übersteigen. Es ist also wichtig, die Anzahl der verfügbaren Elemente im Auge zu behalten. Diesem Sachverhalt wurde in der `FIR` Architektur Rechnung getragen.

Daher wurde die Komplexität auf Blöcke aufgeteilt, welche in aufeinanderfolgenden Taktzyklen abgearbeitet werden. Dies ist aufgrund der Struktur des FIR Filter immer möglich, da er seriell abgeartbeitet werden kann. Genauere Informationen zur Implementierung finden sich im nächsten Abschnitt.

Die sonstige Umsetzung des FIR-Filters ist recht simpel. Benötigt wird eine Registerkette, welche die eingegangenen Signalwerte speichert – beziehungsweise durchschiebt.

Die Koeffizienten müssen vor der Synthese im Quellcode angegeben werden. Die Berechnung der Koeffizienten kann, zum Beispiel, mit Hilfe der Programme OCTAVE/MATLAB oder ScopeFIR erfolgen.

5.4 Der LMS Algorithmus in VHDL

Nachdem nun alle zum Aufbau einer LMS-Entität benötigten Elemente vorhanden sind, kann diese realisiert werden. Die schematische Darstellung ist in Abbildung 31 ersichtlich.

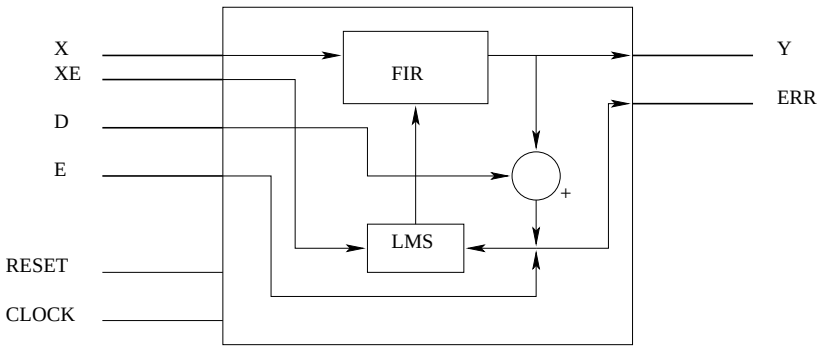

Abbildung 31: LMS Entity

Die interne Arbeitsweise der Entität ist als *state-machine* realisiert. Um die verfügbaren Ressourcen möglichst effizient zu nutzen und dabei die Zy-

klenzeit dennoch kurz zu halten, wurde das FIR-Schieberegister in Blöcke zu je 4 Registern aufgeteilt. Pro Zyklus werden 4 DSP-Einheiten des Stratix FPGAs verwendet, um die Berechnungen durchzuführen. Damit ist eine minimale Filterordnung von 4 gegeben und kann ausschließlich Vielfache dieses Wertes annehmen.

`X` und `D` sind Signaleingänge, `ERR` liefert den gegenwärtigen Fehlerwert und `E` ist ein optionaler Eingang für das Zuführen externer Fehlerinformation. Die Verwendung des Einganges hängt vom Zustand der Leitung `USE_EXT_E` ab. Wird diese auf 0 gezogen, so wird `ERR` aus `D - Y` berechnet. `Y` bezeichnet dabei den gefilterten Wert von `X`. Wird `E` als Eingang benutzt, so wird der adaptive Filter nach den, an dem Eingang `E` anliegenden, Signalwerten gestellt. Die Bezeichnungen der Signale wurden nach der, im Abschnitt 3.3, festgelegten Nomenklatur gewählt.

Zusätzlich dient der Zustand des Einganges `HOLD` dazu die Koeffizienten „einzufrieren“. Dies ist dazu gedacht, um den LMS einmalig zu kalibrieren und danach mit dem aktuellen Koeffizientensatz fortzufahren. Diese Notwendigkeit wird beim Betrachten der Modellierung des Sekundärpfades im FxLMS Algorithmus ersichtlich.

Der Eingang `XE` stellt eine Besonderheit dar. In Hinblick auf die Implementierung des FxLMS Algorithmus mußte die Möglichkeit geschaffen werden, Filterung und Koeffizientenupdate zu trennen. Während die Werte am Eingang `X` lediglich den Filter passieren und dabei durch die aktuellen Koeffizienten modifiziert werden, dienen die Werte an `XE` zur Veränderung der Koeffizienten. Es ist also möglich den Algorithmus von der Filterung zumindest teilweise abzutrennen.

Eine andere Möglichkeit wäre, beide Teile als Entitäten aufzuspalten – sprich FIR-Filter und LMS-Algorithmus. Das Bindeglied wären hier die Koeffizienten, welche vom LMS an den FIR-Filter übergeben werden. Das Problem hierbei ist, die Anzahl der Signalleitungen, welche zwischen den Entitäten verlaufen würden. Bei einer Filterordung von 128 und einer Wortbreite von 16 Bit wären dies 2048 nötige Verbindungen. Um diesen Aufwand zu reduzieren, könnte man die Koeffizienten in einem RAM Speicher ablegen. Dies wurde jedoch im vorliegenden Projekt nicht realisiert.

Um den LMS zu starten, wird am Eingang `CALC` für einen Taktzyklus eine ’1’ erwartet. Sobald der Rechenvorgang abgeschlossen ist und gültige Werte an den Ausgängen anliegen, wird dies durch eine ’1’ am Ausgang `READY` angezeigt.

Die Eingänge werden jeweils an der *steigenden* Flanke von `CLOCK` gelesen und verabeitet.

Das Rücksetzen aller Register erfolgt durch Anlegen einer ’1’ am `RESET` Eingang. Solange `RESET` ’1’ ist, werden keine Berechnungen durchgeführt.

Die Anzahl der Filterblöcke wird vor der Synthese durch den generischen Parameter `blocks` eingestellt und muß zumindest 1 sein. Ebenso muß der Parameter μ bereits zu Anfang festgelegt werden. Dabei ist zu beachten, daß

der generische Parameter `mu2` heißt und den Wert 2μ in Fixpunktdarstellung erhalten soll.

5.5 Der NLMS Algorithmus in VHDL

Der NLMS Algorithmus ist prinzipiell gleich aufgebaut wie LMS, jedoch wurde hier die Berechnung des nun zeitabhängigen Parameters $\mu(k)$ hinzugenommen.

Dabei stellt jedoch die Division durch die Eingangsleistungsabschätzung ein nicht zu unterschätzendes Problem dar. Wie in Abschnitt 5.1.4 dargestellt gibt es grundsätzlich zwei Lösungsansätze. Aus Gründen der Effizienz wurde auf die Implementierung eines ROMs verzichtet und der MSB-Shift Ansatz gewählt.

Die Laufzeit des NLMS ist etwas länger als die des LMS, da die Abschätzung der Eingangsleistung zusätzliche $blocks/4 + 2$ Zyklen benötigt.

6 Simulationen in VHDL

6.1 Allgemeines zu den Simulationen

Die Simulationen wurden unter Verwendung von *ModelSim* durchgeführt. Zwecks Vergleich wurden auch Simulationen mit dem frei verfügbaren Programm *ghdl* durchgeführt, jedoch dauerte die Ausführung der Simulation wesentlich länger als mit *ModelSim*. Aus diesem Grund wurden alle Simulationen mit *ModelSim* durchgeführt.

Die Ausgangssituation ist den Experimenten in Abschnitt 3.3.1 sehr ähnlich. Es wurden lediglich die Signale bedämpft, um Überläufe zu vermeiden. Das Rauschen wurde um -3dB, das Nutzsignal um -6dB vor der additiven Mischung abgeschwächt.

Da die VHDL Implementierung Fixpunktarithmetik einsetzt wurde für die Wortlänge $QI + QF$ der Wert 16 genommen. Dabei entfallen 4 Bit auf Vorzeichen und Ganzzahlteil, die restlichen 12 Bit auf die Nachkommastellen. Dieser Wert ist nicht optimal, da die Werte intern auf das Intervall [-1, 1] beschränkt sind. Durch diese Normierung ist sichergestellt, daß die Ergebniswerte das definierte Intervall nicht verlassen. Die drei Vorkommastellen (das vierte Bit ist das Vorzeichen) sind nur als Sicherheit für minimale Überläufe vorgesehen (beispielsweise bei Fehlabschätzung des Parametes $\mu(n)$ im NLMS), sollten jedoch nie zum Einsatz kommen.

Um die Algorithmen im vorhinein simulieren zu können, wurde das Framework um zwei zusätzliche Klassen erweitert. Diese (`CVHDLInput` und `CVHDLOutput`) dienen dazu, die interne Darstellung des Frameworks in eine einfache, textuelle Darstellung von `BIT_VECTOR`en umzuwandeln. Dabei können die Parameter QI und QF frei definiert werden. Auf diese Weise konnten, mit ein paar Codezeilen, Umwandlungsprogramme von VHDL-Daten nach WAVE und zurück geschrieben werden – es handelt sich hierbei um die Programme `vhdl2wav` und `wav2vhdl`. Die dabei generierten Daten sehen für *Q4.12* Fixkommazahlen so aus:

```
1111111101011101
1111111100011001
0000010011010000
usw.
```

Diese Daten können nun in VHDL eingelesen werden. Dabei wird folgendes Konstrukt benutzt (hier gezeigt am Beispiel der Datei für den Eingang `X`):

```
file X_IN : text open read_mode is "lms_x_in.dat";
...
while not endfile(X_IN) loop
    readline (X_IN, li);
```

```
    read(li, READ_X);

    X <= READ_X;
    ...
end loop;
```

6.2 LMS

Der erste Versuch wurde mit Filterordnung 1 und einem μ von 0.0625 durchgeführt. Abbildung 32 zeigt das gefilterte Eingangssignal, sowie einen Ausschnitt (die ersten 25000 Samples) um das Einschwingverhalten sichtbar zu machen. Wie zu erkennen ist, arbeitet der LMS Algorithmus auch unter Einsatz von Fixpunktarithmetik sehr zuverlässig.

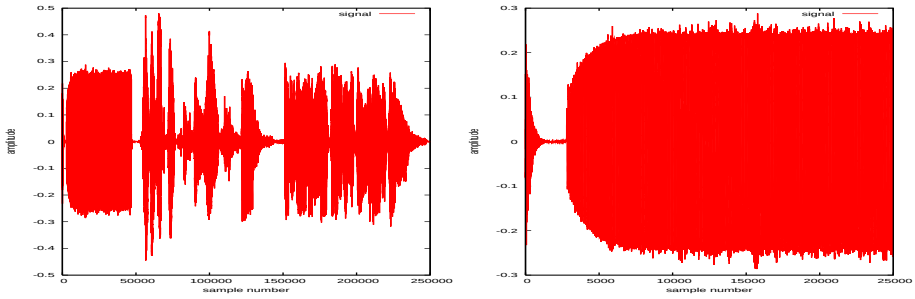

Abbildung 32: VHDL LMS - $N = 1$, $\mu = 0.0625$, $WL = 4.16$

Ein Problem ist jedoch der Wertebereich der Variablen. Während in einer Gleitkommaimplementierung die Divergenz bei bestimmten Werten sehr gut beobachtet werden kann, können Überläufe in der Fixkommaimplementierung bei der Diagnose fehlerhaft interpretiert werden. Diese Überläufe können vor allem beim Einsatz des LMS Algorithmus gegeben sein – für den Fall einer falschen Dimensionierung des Parameters μ.

Bei Einsatz des NLMS Algorithmus wird dieses Problem jedoch sehr effektiv umgangen.

6.3 NLMS

Die Simulation des NLMS Algorithmus verlief ebenfalls sehr positiv. Dabei ist zusätzlich festzustellen, daß die Konvergenz etwas besser war als im Falle des LMS. Dies ist jedoch nicht weiter verwunderlich, da auch die OCTAVE und die C++ Simulation dies vorhergesagt haben.

Es ist weiterhin anzumerken, daß die numerische Ungenauigkeit, aufgrund der verwendeten Fixkommadarstellung, keinen großen Einfluß auf das Koeffizientenupdate nimmt. In bester Näherung erhält man für die Koeffizienten Werte, welche an maximal 4 von 28 Bitstellen vom Sollwert abweichen.

Dies wird trotz der höchst ungenauen Division erreicht. Es ist also bestätigt, daß der NLMS – in Punkto Konvergenz – dem simplen LMS Algorithmus einiges voraus hat.

6.4 FxLMS

Die letzte Simulation bildet die Quintessenz. Hier soll der FxLMS Algorithmus getestet werden. Dabei wurde sehr ähnlich der OCTAVE Simulation vorgegangen. Dabei werden nun keine fixen Sekundärpfade verwendet, sondern ein tatsächlicher Rauschgenerator und ein fixer FIR Sekundärpfad.

Die Kalibrierung erfolgt während der ersten 800 Samples. Die ist in Abbildung 33 auch sehr gut erkennbar. Der Sekundärpfad wird aus Gründen besserer Konvergenz vom NLMS Algorithmus gestützt.

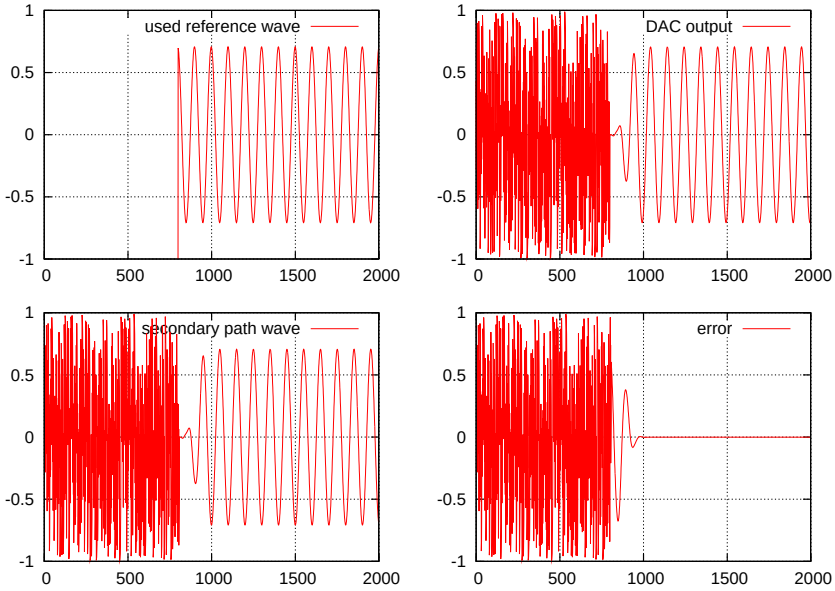

Abbildung 33: FxLMS - VHDL Simulation

Nach der Kalibrierungsphase wird das Referenzsignal – in diesem Fall ein Sinus - in den (nicht vorhandenen) Primärpfad eingespeist. Trotz der Verfälschung des Signals wird, durch die gute Annäherung der Abschätzung, sehr schnell Konvergenz erreicht. Dabei ist zu beachten, daß als Kernfilter ein LMS gesteuerter Filter zum Einsatz kam. Es wurde bewußt auf den Einsatz des NLMS Algorithmus verzichtet, da dieser den Parameter μ für diese Anwendung sehr ungünstig beinflussen würde.

Betrachtet man die Abbildung, so erkennt man, daß bereits nach etwa 200 Samples die Regelung eingeschwungen ist. Damit konnte also die Funktion des FxLMS - zumindest in der Simulation – gezeigt werden.

Es wurden des weiteren Versuche mit verschiedenem μ durchgeführt.

Dabei wurde festgestellt, daß der FxLMS Algorithmus in VHDL um einiges empfindlicher ist, als seine Gleitkomma-Pendants. Es ist also besondere Sorgfalt, bei der Auswahl des Parameters walten zu lassen.

Die Versuchsausgänge sollen an dieser Stelle nicht graphisch dargestellt werden, ähneln sie doch den Ausgängen der OCTAVE Versuche mit zu hohem μ. Es kommt noch gravierend hinzu, daß Überläufe in der Fixpunktarithmetik, vergleichbar mit einer Implementierung eines C `unsigned` Datentyps mit einem Wertebereich von ± 1 sind. Also führen Werte über 1 sofort zu negativen Werten und umgekehrt. Von diesem Zustand kann sich der Algorithmus nicht mehr erholen und divergiert.

6.5 Ergebnis

Man kann anhand der VHDL Simulationen erkennen, daß die präsentierten Algorithmen ihren Gleitkomma – Pendants kaum nachstehen. Damit ist auch gezeigt, daß es nicht nötig ist einen DSP zu benutzen, um die LMS Familie einsetzen zu können – zumindest nicht in der hier dargestellten FIR Form.

Es ist jedoch darauf zu achten die Parameter genau zu wählen – besonders in Hinblick auf eventuelle Überläufe der Zahlenbereiche. In dieser Hinsicht reagieren die Algorithmen weitaus empfindlicher als Implementierungen höherer Zahlengenauigkeit.

Die Algorithmen wurden auch auf Synthetisierbarkeit geprüft und auch hier gab es keine Probleme. Auch die anschließende Simulation, welche hier nicht noch einmal dargestellt wird, liefert die erwarteten Ergebnisse.

Damit soll nun der Softwareteil beendet werden und das weitere Augenmerk auf den Aufbau einer möglichen Hardwarerealisierung gerichtet werden.

7 Der Testaufbau

7.1 Grundsätzliches

Nachdem bis jetzt lediglich Ergebnisse aus den Simulationen vorliegen, soll nun überprüft werden, ob diese Ergebnisse auch in der Realität erzielt werden können. Dazu ist es notwendig, die externe Beschaltung zu untersuchen – insbesondere die Übertragungsfunktion der einzelnen Blöcke.

Abbildung 34 zeigt die benötigten Komponenten für das Experiment. Die punktierten Blöcke fassen eine komplette Transformation vom digitalen zum analogen Teil, beziehungsweise umgekehrt, zusammen.

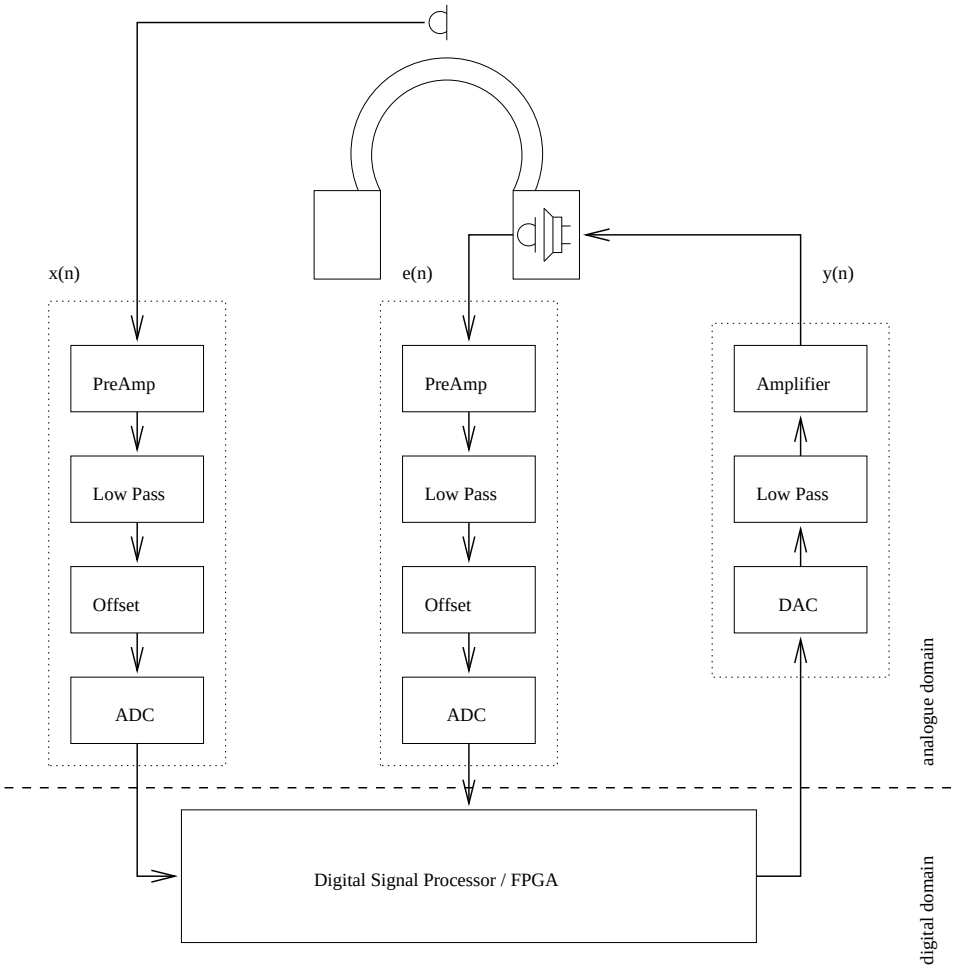

Abbildung 34: Hardware Beschaltung

Die Eingänge der Schaltung bilden die Mikrophone. Es werden vier Blöcke durchlaufen. Eine genaue Beschreibung der Hardware findet sich in den, in den Klammern angeführten, Abschnitten.

- Vorverstärker (Abschnitt 7.3.1)
- Tiefpassfilter (Abschnitt 7.3.2)
- Addition der für den ADC benötigten Offsetspannung (Abschnitt 7.3.3)
- Analog/Digital Wandler (Abschnitt 7.2.3)

Der Ausgang zum Lautsprecher hin wird von drei Blöcken gebildet. Details finden sich wiederum in den Abschnitten in den Klammern.

- Digital/Analog Wandler (Abschnitt 7.2.4)
- Tiefpassfilter (Anschnitt 7.4.3)
- Verstärker (Abschnitt 7.4.4)

7.2 Verwendete Hardware

7.2.1 Allgemeines

Nach der Vorstellung des grundsätzlichen Aufbaus im vorhergehenden Abschnitt, sollen nun die Komponenten im Einzelnen beleuchtet werden.

7.2.2 Altera Stratix II FPGA / MJL Stratix Board

Um den Aufwand des Aufbaus einer eigenen Steuerplatine zu umgehen, wurde auf eine bereits vorhandene Experimentierplatine zurückgegriffen – und zwar auf das Stratix Development Kit von MJL [23].

Die Platine stellt einen FPGA vom Typ *Stratix EP1S25* zur Verfügung. Zusätzlich bietet das Board Erweiterungsmöglichkeiten für Zusatzplatinen, sowie LEDs und Digitalanzeigen für die Interaktion mit dem Benutzer.

Die Taktung ist mit 10 MHz vorgesehen. Auf dem Board wird der FPGA aber mit einem Takt von 33 MHz betrieben. Es ist jedoch möglich intern die Frequenz der Uhr über PLL Schaltkreise auf 10 MHz zu reduzieren. Dies ist in [24] ausführlich beschrieben.

Den Kern der Platine bildet das Altera Stratix EP1S25 FPGA. Dieses FPGA bringt bereits von Haus aus einen Grundstock von DSP Funktionen mit. Darunter fallen Addierer, Multiplizierer und MAC-Einheiten. Der größte Teil des Aufbaus wird jedoch nicht genutzt. Die Platine ist in Abbildung 35 dargestellt.

Die Platine bietet eine Vielzahl von Anschlußmöglichkeiten – von USB, RS232, über VGA bis hin zum Tastatur/Maus Anschluß ist alles vertreten. Zudem besteht die Möglichkeit auf 4MB Flash-Speicher und 8MB SDRAM Speicher zurückzugreifen. Details finden sich in [23].

Der Download der Schaltung erfolgt über ein JTAG Interface, in Verbindung mit *ByteBlaster MV* oder *MasterBlaster* Verkabelung. Das VHDL Design wurde mit den folgenden Programmen realisiert:

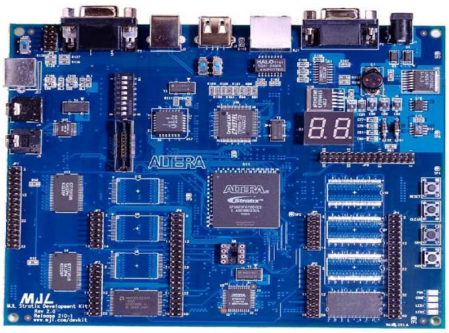

Abbildung 35: MJL Stratix Board

- ModelSim 6.0AE - Simulator
- Synplify 8.1 - Synthese
- Quartus II - Realisierung und Download der Schaltung

Durch die Einbettung von DSP Elementen in das FPGA, bietet das EP1S25 FPGA einen zusätzlichen Anreiz. Es wurde gezeigt [25], daß die Stratix Familie durchaus in der Lage ist, für Signalverarbeitungsaufgaben herangezogen zu werden.

7.2.3 ADC ADCS7476

Dieser Analog/Digital Wandler wird mit einer Versorgung von +5V betrieben. Die maximale Leistungsaufnahme beträgt dabei 10mW. Durch Aufrufen des Stromsparmodus kann dieser Wert auf 5μW gesenkt werden. Dieses Feature wird jedoch bei diesem Projekt nicht ausgenutzt.

Die Wortbreite beträgt für diesen Typ 12 Bit. Die Wandlungsgeschwindigkeit beträgt 1 MSPS und wird durch Einsatz der *successive-approximation* Technik, in Verbindung mit einem *charge-redistribution DAC*, erreicht. In Bild 36 ist die schematische Darstellung des internen Aufbaus dargestellt:

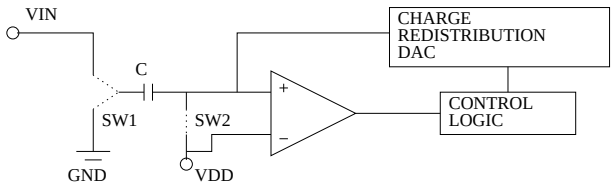

Abbildung 36: Funktionsweise des ADC

In der Funktionsweise werden zwei Modi unterschieden:

- Track Mode (`SW1` zu `Vin`, `SW2` geschlossen)

- Hold Mode (`SW1` zu `GND`, `SW2` offen)

Im *track mode* wird der Sampling-Kondensator `C` von der zu messenden Spannung `Vin` geladen. Der Komparator wird durch `SW2` auf einem ausgeglichenem Level gehalten.

Im *hold mode* findet die eigentliche Konversion statt. Hier stellt der Komparator das Herzstück dar. Erst wenn dieser Komparator balanciert ist, gilt der analoge Wert als digitalisert. Dabei wird vom *charge-redistribution DAC* Ladung vom Kondensator entnommen oder hinzugefügt. Eine Übersicht über verschiedene Typen von ADCs kann in [26] nachgelesen werden.

Der Wechsel vom *track mode* zum *hold mode* wird durch den Wechsel des Wertes der Leitung `/CS` von 1 auf 0 initiiert. Dabei wird der digitale Wert sukzessive an den fallenden Flanken von `SCLK` auf dem Pin `SDATA` ausgegeben. Nach der dreizehnten fallenden Flanke springt der ADC wieder in den *track mode* zurück und verbleibt bis zur nächsten fallenden Flanke von `/CS` in diesem Zustand.

7.2.4 DAC AD5320

Dieser Digital/Analog Wandler bietet bei einer Versorgung von +5V eine slew-rate von 1V/Sekunde. Der interne Ausgangsverstärker ist in der Lage, eine Last von 2kΩ parallel zu 1nF zu `GND` zu betreiben.

Die Wortbreite ist wie beim ADC auf 12 Bit festgelegt und realisiert eine Ausgangsspannung von 0V bei `0x000` bis V_{DD} bei `0xFFF`. Optimal wäre jedoch eine symmetrische Ausgangsspannung um 0 Volt. Die im Datenblatt [27] vorgeschlagene Beschaltung ist in Abbildung 37 dargestellt. Dadurch wird eine symmetrische Spannung am Ausgang ermöglicht und gleichzeitig das Signal verstärkt.

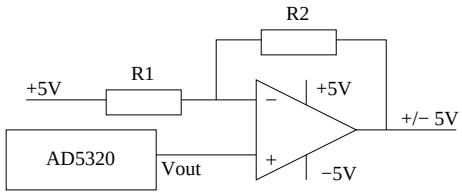

Abbildung 37: Beschaltung des DACs

Es wurde jedoch, aus Gründen minimaler Phasenverschiebung, auf den Einsatz der Beschaltung verzichtet. Die notwendige Transformation wird durch den Koppelkondensator `C3` erreicht.

Die hardwareseitige Ansteuerung des DAC erfolgt über die VHDL-Entität `DA_Interface`. Dabei ist zu beachten, daß die Übertragung eines Samples mindestens 12 SCLK Zyklen in Anspruch nimmt, zuzüglich einer einzuhaltenden Ruhephase zwischen den Samples von mindestens 33ns.

7.3 Mikrophon-Ankoppelungs Platine

Um den vom ADC geforderten Eingangsspannungbereich von 0 bis 5V zu erreichen sind einige Vorkehrungen zu treffen. Dabei sind jedoch folgende Randbedingungen zu beachten:

- die Ausgangsspannung des Mikrophons auf einen höheren Pegel bringen
- sicherstellen, daß die Eingangsspannung den zulässigen Bereich des ADCs nicht verläßt

Ein Lösungsansatz ist in [28] dargestellt. Eine wichtige Ergänzung dabei ist die Erweiterung der Schaltung für Elektret-Mikrophone. Bei diesem Projekt wurden Mikrophonkapseln vom Typ `60D82` von *Panasonic* verwendet. Die Kennwerte sind wie folgt gegeben:

Frequenzbereich:	20 - 20000 Hz
Empfindlichkeit:	6mV/Pa/1kHz/$\pm$4dB
Ausgangsimpedanz:	2kΩ, R_L: 2.2kΩ
Signal/Rauschabstand:	$>$ 58dB
Koppelkondensator:	0.1 - 4.7 μF
Stromversorgung:	1.5 - 10V/0.5 mA

7.3.1 Der Vorverstäker

Die Schaltung zur Anbindung eines Elektret-Mikrophons, ist in Abbildung 38 gezeigt. Elektret-Mikrophone sind gepolte Bauteile. Intern ist ein FET-Verstärker vorhanden um die schwachen Signale gleich an Ort und Stelle zu verstärken.

Die Versorgung des internen Verstärkers erfolgt über `R1`. Am linken Anschluß des Kondensators `C2` stellt sich nun das Ausgangssignal des Mikrophons, samt überlagertem Gleichspannungsanteil ein. Dieser ist jedoch für die Verstärkung störend und wird mit Hilfe des Kondensators, der als Hochpass (aufgrund der hohen Kapazität ist die Grenzfrequenz jedoch sehr niedrig) arbeitet, entfernt.

Als erste Verstärkerstufe gelangen Operationsverstärker vom Typ `NE5532` zum Einsatz[29]. Dabei handelt es sich um einen Dual-Operationsverstärker im DIL-8 Gehäuse, welcher über ein sehr gutes Rauschverhalten verfügt und damit ideal zum Verstärken des schwachen Mikrophonsignals geeignet ist. Die frequenzabhängige Verstärkung beginnt erst ab 10^5 Hz zu sinken.

Um das Mikrophon zu entlasten ist nach dem Koppelkondensator der erste Operationsverstärker des `NE5532` (`IC1A`) als Impedanzwandler geschalten. Es ist keine Offsetkorrektur notwendig, da der Operationsverstärker intern auf *unity gain* – also Verstärkung 1 – kompensiert ist.

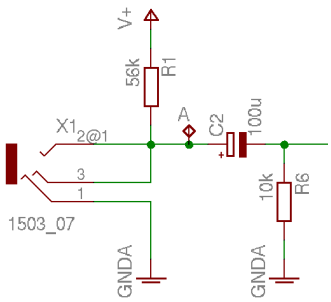

Abbildung 38: Mikrophon-Ankoppelung

Der zweite Operationsverstärker (in weiterer Folge als *OpAmp* bezeichnet) ist als invertierender Verstärker geschalten und erreicht durch das Widerstandsverhältnis $R3 : R2$ eine Verstärkung nach Formel 7.1. Dabei ist jedoch anzumerken, daß hier nicht die volle Verstärkung ausgeschöpft wird, sondern lediglich ein Pegel von $\pm 100\text{mV}$ erreicht werden soll.

$$A_{U,max} = -\frac{R3}{R2} = -\frac{220k}{10k} = -22 \tag{7.1}$$

Durch die eher geringe Verstärkung, werden Verzerrungen des OpAmp umgangen und es wird eine Anbindung an eine bereits existierende Schaltung ermöglicht, welche zum Wandeln des Signals in eine für den ADC verträgliche Form bewerkstelligt. Es ist wichtig, die Verstärkung nicht zu hoch zu wählen, da sonst störende Interferenzen ebenfalls mitverstärkt werden. Besser ist es, die Verstärkung auf mehrere Stufen aufzuteilen.

7.3.2 Der Tiefpass

Der Aufbau gestaltet sich um den Quad-OpAmp Baustein `TL084`. Dieser wird in der DIL-14 Ausführung verwendet und zeichnet sich durch eine sehr hohe *slew-rate* von $16\text{V}/\mu\text{s}$ aus.

Dieser Filter hat den Zweck hochfrequente Störanteile aus dem Signalgemisch zu entfernen. Dies ist wichtig, um den Samplingvorgang so rein als möglich zu halten.

Dabei stellt sich nun die Frage, welcher Filtertypus für diese Aufgabe am besten geeignet ist. Zwei verbreitete Typen sind dabei in Abbildung 39 gezeigt – *Butterworth* und *Chebyshev*.

Wie aus diesen Bildern ersichtlich ist, bietet der Chebyshev Filter eine hohe Flankensteilheit, in Bezug auf den Übergang zwischen *Passband* und *Stoppband*. Dies wird jedoch durch ein hohes *ripple* (also Verzerrungen) im Passband erkauft.

Wesentlich zahmer zeigt sich der Butterworth-Filter, welcher keine hohe Flankensteilheit hat, dafür aber im Passband eine nahezu lineare Übertra-

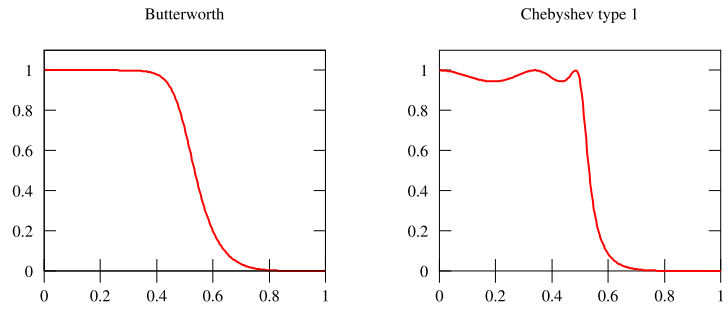

Abbildung 39: Vergleich Butterworth- und Chebyshev-Filter

gung zeigt. Die Ordnung des Filters bestimmt dabei die Steilheit maßgeblich. Dies ist in Abbildung 40 gezeigt. Wie dabei zu sehen zu ist steigt die Steilheit mit zunehmender Ordnung.

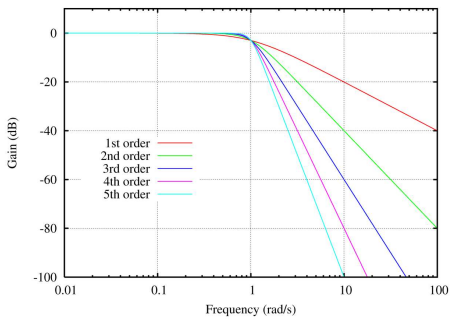

Abbildung 40: Verschiedene Ordnungen des Butterworth-Filters

Um den Bauteilaufwand gering zu halten, wurde ein Filter der Ordnung 2 gewählt. Nun gilt es die Werte für den Filter zu erhalten. Gemäß [30] und [31] greift man auf eine *Sallen-Key* Konfiguration zurück. Unter Berücksichtigung von Vereinfachung 2 erhält man folgende Beziehungen:

$$\begin{aligned} R1 &= mR \\ R2 &= R \\ C1 &= C \\ C2 &= nC \\ f_c &= \frac{1}{2\pi RC\sqrt{mn}} \end{aligned}$$

Sei $R = 16k\Omega$, $m = 0.5$ und $C = 10\text{nF}$, $n = 2$, dann folgt daraus die Cutoff-Frequenz $f_c = 993\text{Hz}$. Die Verstärkung im Passband beträgt hier 1.

Der verwendete Aufbau der Schaltung ist in Abbildung 41 gezeigt. Dabei

wurden jedoch etwas abweichende Werte verwendet. Diese sind aus dem Schaltbild ersichtlich.

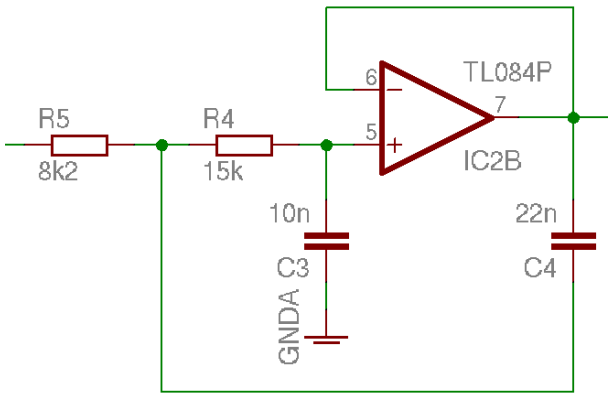

Abbildung 41: Mikrophon-Filterstufe

Der Versuchsaufbau lieferte für diese Konfiguration das Schirmbild, wie in Abbildung 42. Die verwendete Frequenz beträgt hier 1kHz. Das Eingangssignal (`Ch1`) bildet eine Sinusschwingung mit $1V_{pp}$, das Ausgangssignal ist unter `Ch2` zu finden. Die Amplitude beträgt dabei $710mV_{pp}$ – also der -3dB Punkt.

Die Phasenverschiebung zwischen dem Eingangssignal und dem Ausgangssignal beträgt etwa 250μs. Dies entspricht den Erwartungen, da die Phasendrehung bei der Grenzfrequenz 90° nacheilend beträgt.

7.3.3 Der Spannungschieber

Den letzten Teil bildet die Verschiebung des Signals in den gültigen Bereich von 0 bis +5 Volt. Das Schaltbild in Abbildung 43 zeigt den Aufbau.

Über `R11` Pin `E` wird das Signal des Butterworth-Filters in die Schiebestufe eingespeist. Die Amplitude des Signals kann hier ebenfalls über den Trimmer nachgestellt werden.

Über `R12` kann nun die Offsetspannung justiert werden und zwar im Bereich von -V_{ss} bis 0V.

Der Operationsverstärker `IC1D` arbeitet als Summierverstärker und addiert zum Signal die eingestellte Offsetspannung (das negative Vorzeichen ergibt sich aus der Arbeitsweise als invertierender Verstärker).

Es ist absolut wichtig, die Schaltung so zu justieren, daß der zulässige Spannungsbereich für den ADC nicht verlassen wird, da dies die Zerstörung des Bausteines zur Folge haben könnte. Ebenso darf die Speisespannung 5V nie überschreiten, da der Baustein sofort überhitzen würde und zerstört werden würde.

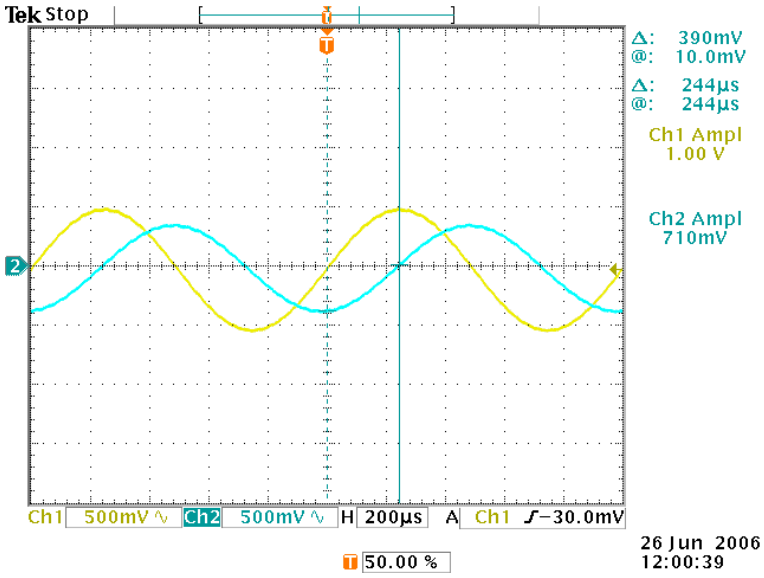

Abbildung 42: Butterworth-Filter bei 1kHz

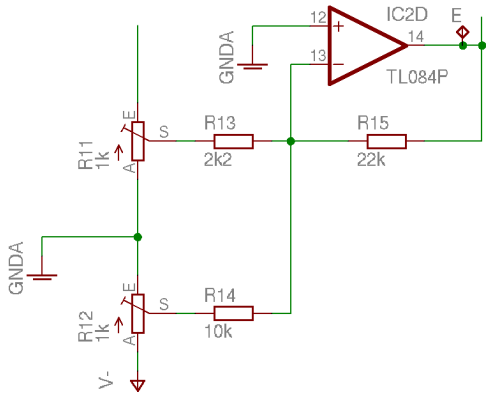

Abbildung 43: Mikrophon-Spannungsschiebestufe

Der ADC selbst wird vom Stratix Board gespeist. Auf Block `JP7` kann eine Versorgungsspannung von +5V abgegriffen werden – die maximale Belastung beträgt hierbei 50mA. Der ADC ist sehr empfindlich in Hinsicht auf die Eingangsspannung. Diese dient gleichfalls als Referenz und sollte möglichst ruhig sein. Im Aufbau konnte jedoch, trotz Parallelschaltung der im Datenblatt angegebenen Kapazitäten, kein 100%-ig zufriedenstellender Betrieb erreicht werden. Die Auswirkungen sind in Abbildung 44 ersichtlich. Die Abbildung ist ein Ausschnitt aus der Kommunikation des FPGAs mit den angeschlossenen ADC und DAC Komponenten.

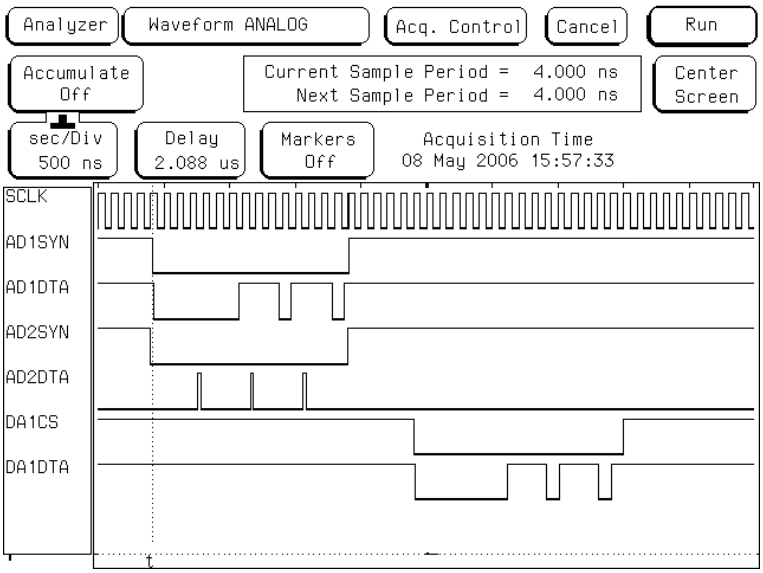

Abbildung 44: Spikes in den Logikwerten

Durch die mangelnde Pufferung reißen die Logikwerte zum Teil zu schnell wieder ab und es ergeben sich Spikes, welche vom FPGA nicht verwertet werden können. Außerdem kommt es zu einer Verfälschung der gemessenen Werte. Um diesem Problem entgegenzuwirken, wurde der Kondensator `C6` in die Datenleitung eingebracht. Er dient dazu, das Signal „logisch 1“ lange genug aufrechtzuerhalten, um als tatsächliche „logische 1“ erkannt zu werden.

Die Taktgebung des Systems ist in der ersten Zeile ersichtlich (Bezeichner `SCLK`). In der Zeile mit der Bezeichnung `AD1DTA` sind reguläre Logikwerte erkennbar, während in Zeile `AD2DTA` kein Kondensator verwendet worden ist. Der Unterschied ist klar ersichtlich.

Diese Lösung ist nicht für den dauernden Betrieb vorgesehen und sollte, durch einen Aufbau auf einer Printplatte, nicht mehr von nöten sein.

7.3.4 Justierung

Die Justierung der Schaltung erfolgt über die zwei Potentiometer `R11` und `R12`. Ursprünglich war noch ein Potentiometer für die Vorverstärkung vorgesehen, jedoch wurde dieses wieder verworfen, da sich gezeigt hat, daß bei einer Verstärkung von über 50 zu viel Störeinflüsse in die Leitung aufgenommen werden. Deshalb wird der erste Verstärker mit einer maximalen Verstärkung von -22 betrieben.

Über Potentiometer `R11` stellt man das Signal auf einen Wert von etwa 4.5V peak-to-peak ein. Es ist nicht empfohlen das Signal auf $5V_{pp}$ zu justieren, da lautere Geräusche über diesen Wert hinaus verstärkt werden könnten und damit den ADC schädigen könnten.

Die Offset-Spannung wird über `R12` geregelt. Dieses Potentiometer ist so einzustellen, daß das Ausgangssignal im zulässigen Bereich von 0 bis 5V verbleibt. Es sei angemerkt, daß die Koppelung des Oszilloskopes auf Gleichspannungen (DC) gestellt sein muß, da man sonst keine Verschiebung des Signals messen kann.

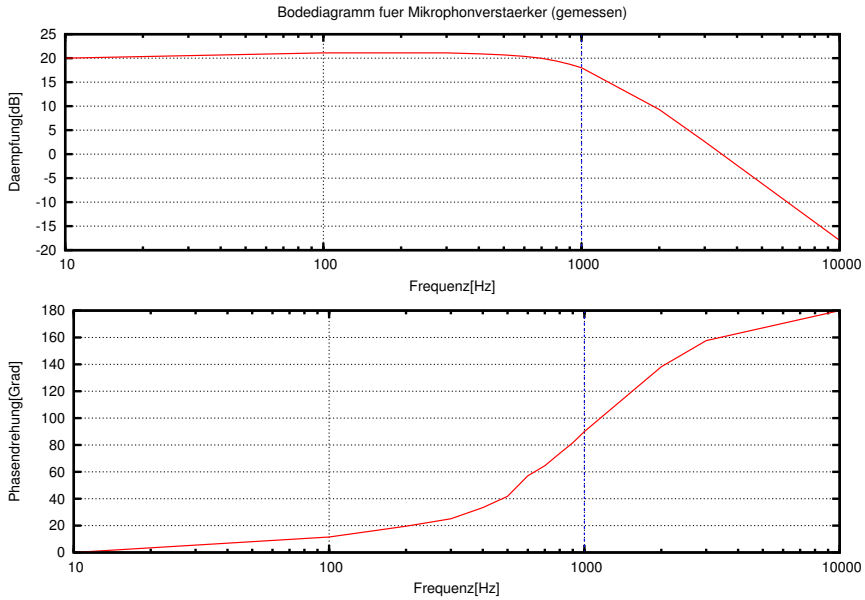

Abbildung 45: Frequenz- und Phasengang der ADC Anbindung

Der Phasengang der Schaltung ist beinahe ausschließlich durch die Phasenverschiebung des Butterworth-Filters bestimmt. In Bild 45 ist das Bodediagramm für die gesamte Schaltung ersichtlich.

7.4 Kopfhörer-Verstärker Platine

Um das vom DAC gelieferte Signal im Kopfhörer wiederzugeben, muß dieses erst symmetrisch gemacht und dann verstärkt werden. Zusätzlich ist es

wichtig, die diskreten Spannungswerte, die vom DAC ausgegeben werden, zu glätten. Dies soll wiederum durch ein Filter realisiert werden. Der Aufbau soll drei Aufgaben erfüllen:

- Die Wandlung des Spannungsbereiches von 0-5V in ±2.5V
- Filterung der Aliasing Effekte des DAC
- Verstärkung des Signales zur Wiedergabe

7.4.1 Der Anschluß an den DAC

Wie in 7.2.4 nachzulesen ist, liefert der DAC eine Ausgangsspannung im Bereich von 0 bis 5V. Die Umwandlung des Spannungsbereiches erfolgt mit der im Datenblatt des DACs angegebenen Schaltung, welche auch in Abbildung 37 auf Seite 64 zu sehen ist. Die Ausgangsspannung für ein Datenwort `D` (im Bereich von 0 bis 4095) kann laut Formel 7.2 berechnet werden:

$$V_O = \left(\frac{10 \times D}{4096}\right) - V_{DD} \tag{7.2}$$

Dabei gilt die Bedingung, daß der Wert von `R1` gleich dem Wert von `R2` ist. Durch die Beschaltung wird ein Ausgangswert von $-V_{DD}$ bei `0x000` und $+V_{DD}$ bei `0xFFF` erreicht.

7.4.2 Die Eingangsstufe

Der Ausgang des DAC ist mit einem Impedanzwandler verbunden. Die Qualität des Signals hängt stark von der Belastung des Ausganges des DACs ab. Durch den Impedanzwandler wird der Ausgang des DAC entlastet.

7.4.3 Der Anti-Aliasing Filter

Der DAC Wandler erzeugt keine kontinuierliche Ausgangsspannung, sondern lediglich eine abgestufte Spannung mit der Schrittweite von:

$$V_{step} = \frac{V_{DD}}{2^{12}} \tag{7.3}$$

Bei 5V ergibt das einen Wert von 0.0012V pro Stufe und ist unter dem Schlagwort „Aliasing“ [32] bekannt. Da Rechtecksignale einen hohen Oberwellenanteil enthalten, welche sich als störendes Signal in der Schaltung bemerkbar machen, wird ein Filter, wie in Abschnitt 7.3.2, eingesetzt. Das Schaltbild kann ebenfalls dort eingesehen werden. Die Auswirkung des Butterworth Filters auf das vom DAC gelieferte Signal, zeigt das Schirmbild in Abbildung 46.

Oben ist das Signal zu sehen, welches vom DAC geliefert wird. Es handelt sich dabei um einen Sägezahn mit einer Periode von 14.2 ms. Dieses

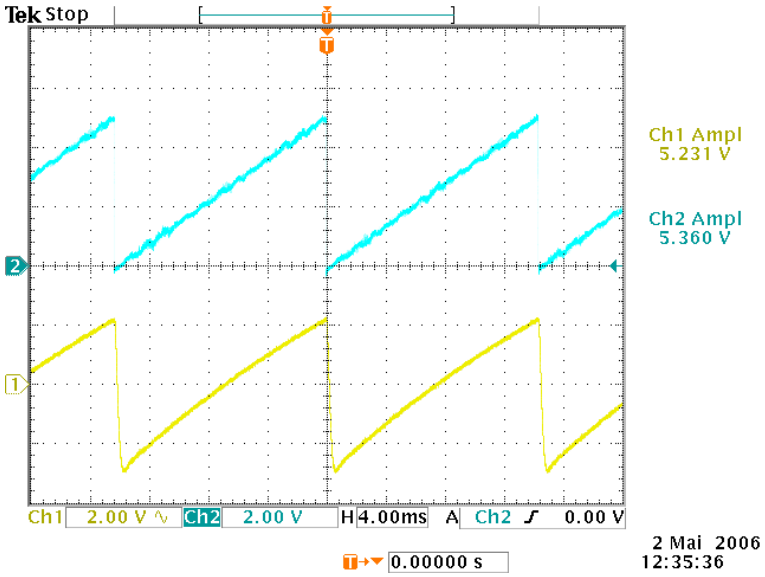

Abbildung 46: Gefiltertes DAC Signal

Signal wird auf dem FPGA generiert und an den DAC weitergereicht. In der unteren Hälfte ist das gefilterte Signal, nach Verlassen des Butterworth Filters, zu sehen. Wie man leicht erkennen kann, ist der Verlauf wesentlich glatter als der des gelieferten Signals. Die Oberwellen werden also sehr effektiv bedämpft.

7.4.4 Der Endverstärker / Justierung

Der Endverstärker ist sehr einfach aufgebaut. Nach dem Filter wird ein Impedanzwandler geschalten, um die Filterstufe möglichst nicht zu beeinflussen. Über den Kondensator `C3` wird das Signal von störenden Gleichspannungsanteilen befreit, welche den Kopfhörer Schaden zufügen könnten. Durch Anlegen einer Gleichspannung würde die Spule konstant bestromt werden und im schlimmsten Fall überhitzen.

Die Justierung der Schaltung gestaltet sich sehr einfach. Der Ausgangspegel wird über das Potentiometer des Ausgangsverstärkers eingestellt. Dazu wird ein sauberes, sinusförmiges Signal mit $5V_{pp}$, Offset +2.5V angelegt. Die Lautstärke wird nun über das Ausgangspotentiometer `R4` geregelt.

7.5 Der Kopfhörer

Als Kopfhörer gelangt der SONY MDR-V500 zum Einsatz. Dabei handelt es sich um einen professionellen Studiokopfhörer, der möglichst geringe Verzerrungen aufweisen sollte. Es ist generell wichtig, darauf zu achten, einen

Kopfhörer ohne intergrierte Filter oder Weichen zu benutzen, da diese den Phasen- und Frequenzgang zusätzlich verändern könnten.

Die Kenndaten des verwendeten Kopfhörers sind wie folgt:

- Frequenzgang: 10 - 25000Hz
- Impedanz: 24Ω
- maximale Leistung: 1000mW

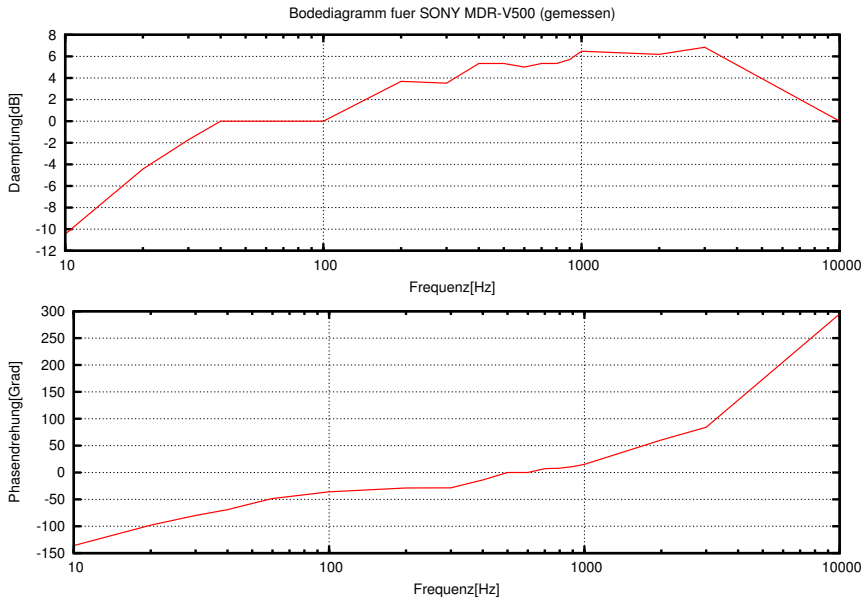

Abbildung 47: Frequenz- und Phasengang des SONY MDR-V500

Das Bodediagramm wurde per Hand erstellt und ist in Abbildung 47 dargestellt. Für die Messung wurde der Verstärkerteil des Mikrophonverstärkers genutzt (unter Auslassung des Butterworth Filters). Der Kopfhörer wurde direkt mit dem Frequenzgenerator, verbunden um inherente Phasendrehungen zu umgehen. Für den verwendeten Kopfhörer ist die Ausgangsleistung des Frequenzgenerators vollkommen ausreichend.

Als Referenz für die Dämpfung wurde der Wert bei 100Hz genommen. Das heißt, daß die Dämpfung für manche Frequenzen auch positiv werden kann. Für die Messung wurde ein Mikrophon in der Kopfhörerkapsel positioniert und mit einer sehr einfachen Operationsverstärkerschaltung auf einem messbaren Signalpegel transformiert. Dies war aufgrund, des vom Mirkrophon gelieferten, sehr schwachen Signals notwendig. Die Schaltung selbst weist keine messbare Phasenverschiebung auf, um die erhaltenen Ergebnisse nicht zu verfälschen.

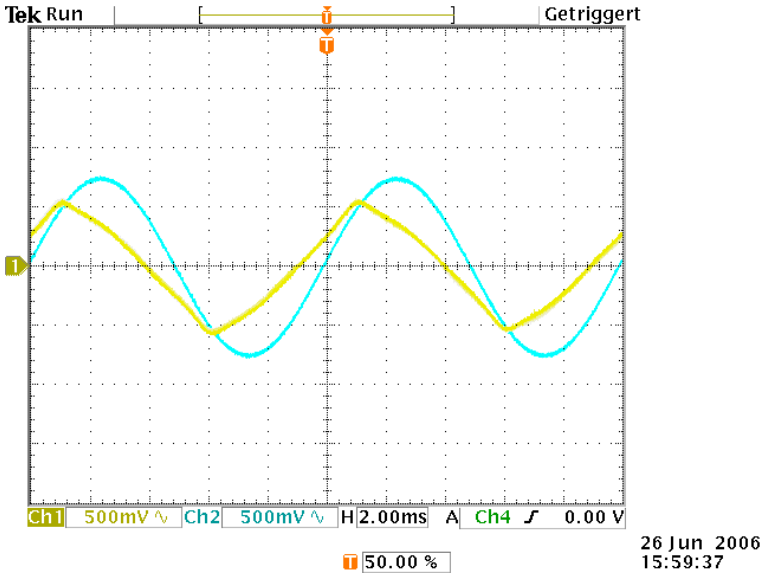

Abbildung 48: Verzerrungen durch den Kopfhörer

Es wurde bei den Messungen mit dem Kopfhörer festgestellt, daß bei manchen Frequenzen der Verlauf des reinen Sinus beeinflußt wird und beinahe in ein Dreiecksignal übergeht. Ein Schirmbild (Abbildung 48) stellt diesen Sachverhalt dar. Die Messfrequenz beträgt 100 Hz. Durch den Frequenzbereich des Mikrophones, kann dieses als Fehlerursache ausgeschlossen werden. Diese Änderung des Signals muß also vom Kopfhörer bedingt sein.

8 Ergebnisse des Hardwareaufbaus

8.1 Allgemeines

Die im vorhergehenden Abschnitt gezeigten Messungen und Diagramme wurden provisorisch auf Steckbrettern aufgebaut. Dieser Umstand ergibt sich aus der Tatsache, daß der endgültige Aufbau der Hardware, nicht primäres Ziel war.

Durch diesen einfachen Aufbau ergaben sich jedoch, im Laufe der Messungen, unvorhergesehen Nebeneffekte. Es war trotz der geringen Verstärkung der Mikrophoneingänge ein nicht unbeträchtliches Störsignal am Ausgang zu messen. Auch konnte leider keine korrekte Arbeitsweise des Filters beobachtet werden. Gründe für dieses Fehlverhalten sollen im nächsten Abschnitt analysiert werden.

8.2 Problemdiagnose

In Abbildung 49 sind im oberen Bereich die Ausgangssignale des Referenz- und Fehlermikrophons nach den jeweiligen Verstärkerstufen zu sehen. Die horizontale Linie bei +2.5V zeigt die ideale Nullinie nach dem Spannungsschieber.

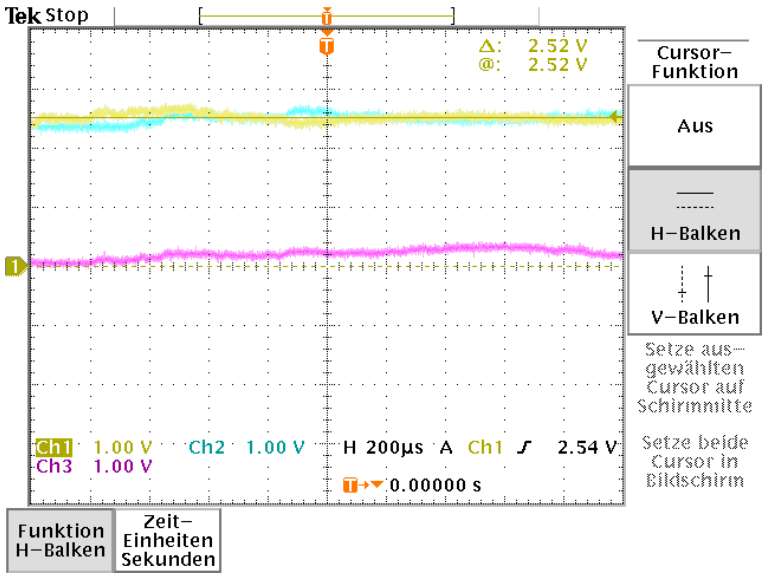

Abbildung 49: Störungen im Ruhebetrieb der Verstärkers

Man erkennt deutlich Schwankungen, beziehungsweise Rauschen mit einer Amplitude von etwa 0.25V. Dies sollte jedoch laut Abschnitt 4.4.6 für den Algorithmus kein nennenswertes Hindernis darstellen.

Gravierender jedoch wiegt der Umstand, daß das Lautsprechersignal ebenfalls hohe Schwankungen aufweist. Dies schlägt sich im Regelkreis als konstanter Fehler nieder. Abbildung 50 zeigt das Lautsprechersignal (im unteren Teil) in feinerer Auflösung. Hierbei ist jedoch wichtig anzumerken, daß dieser Fehler vom System nicht erkannt wird, da er von außen in das System induziert wird.

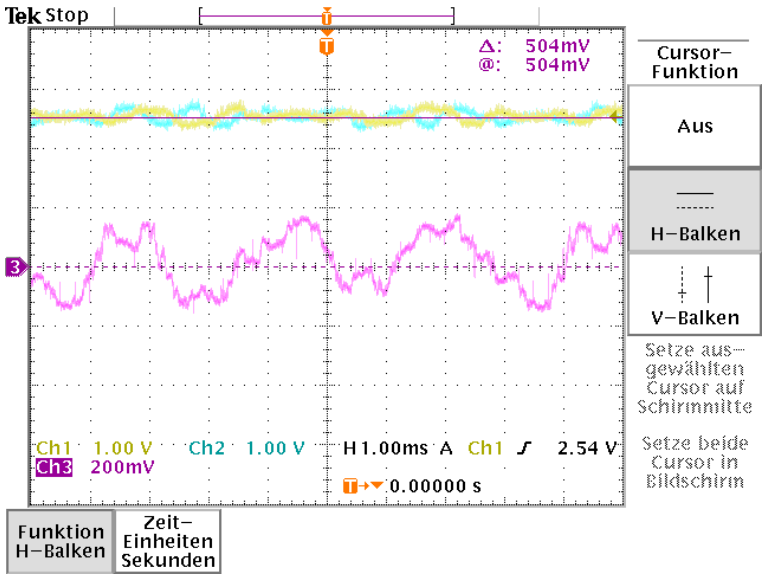

Abbildung 50: Vergrößerte Darstellung des Lautsprechersignals

Durch dieses konstante, jedoch nicht vorhersehbare, Fehlersignal ist der Regelalgorithmus in der Realität nicht in der Lage zu konvergieren, bzw. ein brauchbares Ergebnis zu liefern.

Durch genauere Analyse der Störsignale konnte festgestellt werden, daß zwei Komponenten maßgeblich an dessen Entstehung beteiligt sind:

Zum einen handelt es sich hierbei um den Netzbrumm (also eine Sinusschwingung mit der Frequenz 50 Hz), welcher von den umgebenden Geräten durch Leitungen abgestrahlt wird. Dies ist insbesondere kritisch, da der gesamte Aufbau nicht geschirmt ist.

Die zweite Komponente ist das Signal, welches als Taktgeber für die Wandlerbausteine benutzt wird. Dieses macht sich mit Spannungsspitzen im Signal bemerkbar und tritt mit einer Frequenz von 10 MHz auf.

Auch ist, bedingt durch den sehr einfachen Aufbau der Verstärker, ein temperaturabhängiger Drift im Ausgangsspannungspegel festzustellen. Dieser Umstand ergibt sich auch aus der etwas unglücklichen Wahl der Wandlerbausteine. Bedingt durch den Eingangsspannungsbreich der Bausteine, welcher zwischen 0 und 5 Volt liegen muss, war es nicht möglich Koppelkondensatoren zum Entfernen eines Gleichspannungsanteils zu verwenden.

8.3 Lösungsvorschläge

Der erste verbesserungsfähige Punkt wäre die Wahl der Analog/Digital, beziehungsweise Digital/Analog Wandler. Gerade bei der Verwendung beziehungsweise Verarbeitung von Audiosignalen, bietet sich der Einsatz von Bausteinen mit symmetrischen Eingang an. In diesem Fall nämlich entspricht die Nullinie genau dem Signalwert 0. Dadurch ist es nicht notwendig, weder das Eingangs- noch das Ausgangssignal um einen Gleichspannungsanteil zu verschieben. Ebenso können Kondensatoren zur Koppelung eingesetzt werden. Der Nachteil hierbei jedoch wäre die Verwendung einer symmetrischen Spannungsversorgung. Dies könnte in Hinblick auf Portabilität ein Problem darstellen.

Ferner sollte der gesamte Aufbau auf einer Platine erfolgen und es muß eine strikte Trennung zwischen Analogteil und Digitalteil erfolgen. Dies dient dazu das Übersprechen des Taktsignals, in den analogen Teil, zu unterbinden. Es muß weiters gewährleistet sein, daß die Stromversorgung möglichst glatt und ruhig ist. Im vorliegenden Aufbau wurden die 5 Volt, welche zur Versorgung der Wandlerbausteine dienen, direkt vom MJL Stratix Board genommen. Dieser Punkt würde jedoch bei Batterieeinsatz entfallen. Um elektrische Störungen aus der Umgebung abzuschwächen, wäre es sinnvoll den fertigen Aufbau in einem Metallgehäuse zu platzieren.

Es ist auch festzustellen, daß die verwendete Konfiguration lediglich Butterworth-Filter zweiter Ordnung einsetzt. Hier würde sich, zum Beispiel, der Einsatz von Filterbausteinen höherer Ordnung empfehlen.

Weitere zu prüfende Punkte wären eine genauere Analyse der eingesetzten Mikrophone, sowie des Kopfhörers. Es ist jedoch im Allgemeinen schwer, aussagekräftige Datenblätter zu diesen Bausteinen zu finden. Insbesondere zum Phasengang von Kopfhörern finden sich kaum Dokumente. Auch für den hier eingesetzten Typus konnten lediglich eigenhändig Abschätzungen erstellt werden.

9 Schlußfolgerungen

In der hier vorliegenden Arbeit konnte gezeigt werden, daß durch den Einsatz adaptiver digitaler Filter, die Unterdrückung von störendem Schall, durchaus im Bereich des Möglichen liegt.

Durch die Implementierung in verschiedenen Sprachen, konnte von mehreren Seiten die Funktionsweise und Effektivität der Algorithmen gezeigt werden. Es wurde dabei klar, daß die Genauigkeit der verwendeten Zahlendarstellung eine untergeordnete Rolle spielt. Dies ist insbesondere für den Einsatz in portablen Geräten, wo meist nur eine begrenzte Rechenleistung zur Verfügung steht, wichtig.

Bedingt durch die räumliche Anordnung der aktiven Komponenten, konnten klare Grenzen aufgezeigt werden. Dies umfasst insbesondere die Arten von Signalen, welche effektiv ausgelöscht beziehungsweise unterdrückt werden können. Während bei der klassischen Anwendung der aktiven Schallunterdrückung sehr lange Primärpfade die Regel sind, ist bei der hier gezeigten Anwendung gerade dies nicht der Fall. Daraus folgt, daß der Sekundärpfad *immer* länger ist, als der Primärpfad. Dies führt dazu, daß es realistisch nicht möglich ist, beliebige Signale zu eliminieren.

Durch diese Tatsache wird sich der Anwendungsbereich personenbezogener Schallunterdrückungslösungen immer nur auf periodische Signale erstrecken können. Dies spiegelt sich auch im gegenwärtigen Stand der Technik wieder. Die häufigsten Anwendungsgebiete finden sich deshalb in der Dämpfung von Motorgeräuschen (Flugzeug, Bahn), welche periodische Elemente beinhalten.

Eine sehr wichtige Einsicht ist die Tatsache, daß der benötigte Rechenaufwand auf digitaler Seite sehr gering ist. Die Grenzen werden fast ausschließlich vom analogen Teil definiert. Erst wenn es möglich wäre, die Laufzeiten der analogen Elemente zu verkürzen, könnte auch das Spektrum an dämpfbaren Signalen erweitert werden. Dies ist jedoch in der Praxis wohl schwer oder kaum realisierbar.

Eine weitere wichtige Erkenntnis ist, daß der Aufbau eine entscheidende Rolle für die Funktionstüchtigkeit der Schallunterdrückungslösung spielt. Bei der Auslegung und Realisierung des analogen Teils, muß hohe Sorgfalt, auf eine möglichst hohe Signalqualität, gelegt werden. Dies betrifft besonders den Ausgang zum Lautsprecher hin, welcher das Gegenschallsigal ausgibt. Durch das Induzieren von Störungen kann es zu einer Divergenz des Filters kommen, durch welchen die Wirkungsweise, im schlimmsten Fall, umgekehrt wird.

Es konnte durch die Implementierung in VHDL gezeigt werden, daß es durchaus möglich ist, den digitalen Teil mit sehr geringem Aufwand direkt in Hardware zu lösen. Dennoch muß leider gesagt werden, daß digitale Signalprozessoren einen besseren Weg darstellen. Der große Vorteil, der sich hier bietet, ist die Zusammenfassung der Schnittstellen und der Berech-

nung in einem einzigen Baustein. Dies wird jedoch auf Kosten von eigentlich nicht benötigter Hardware, innerhalb des Bausteins, erkauft. Durch die hohe Verfügbarkeit und die geringen Stückpreise relativiert sich dieser Aufwand wieder.

Abschließend kann also gesagt werden, daß Schallunterdrückungslösungen, auf digitaler Basis, sehr flexibel an bestehende Probleme angepasst werden können. Jedoch sind sie durch den unumgänglichen Einsatz von analogen Bauteilen nur bedingt in der Lage, ihr volles Potential auszuspielen.

10 Anhang

Literatur

[1] Jens Ortscheid and Heidemarie Wende. Studie zu Fluglärmwirkungen. Umweltbundesamt Berlin, 2000.

[2] M. Meis, A. Schick, M. Klatte, and M. Bullinger. Auswirkungen von Transrapid- und Flugverkehrsgeräuschen auf Gedächtnisleistungen: Clustering als Schall-Gedächtnis Mediator. *Fortschritte der Akustik - DAGA 98*, pages 718–719, 1998.

[3] Bundesministerium für wirtschaftliche Angelegenheiten. Dienstanweisung Lärmschutz an Bundesstraßen, 1999.

[4] Tom Weyer and Hans Peter Monner. *Motor- und Aggregate-Akustik*, chapter PKW Innenlärmreduzierung durch aktive Beruhigung der durch die Motorharmonischen erregten Dachblech-Schwingungen. expert-Verlag, 2003.

[5] Larry Schultz. Drive-Up noise cancellation, 2000. Portland State University, ECE 510.

[6] J. Landaluze, I. Portilla, N. Cabezon, A. Martinez, and R. Reyero. Application of Active Noise Control to an elevator cabin. In 15^{th} *Triennal World Congress, Barcelona*. IFAC, 2002.

[7] Dieter Guicking. Aktive Lärm und Schwingungsminderung - Von der Kuriosität zum technischen Produkt. In *Aktive Lärmbekämpfung und Schwingungsabwehr*, Essen, 1995. Haus der Technik, e.V.

[8] Sen M. Kuo, Issa Panahi, Kai M. Chung, Tom Horner, Mark Nadeski, and Jason Chyan. Design of Active Noise Control Systems with the TMS320 Family. Technical report, 1996.

[9] Chu Moy. Active noise reduction headphone systems. http://www.headwize.com/tech/anr_tech.htm, 2001.

[10] Gottfried Fuchs and Peter Tummeltshammer. Machbarkeitsstudie für digitale Schallunterdrückung. Technical report, Vienna University of Technology, 2000. ECS Embedded Computing Systems Group.

[11] Dina Bruns. Schallschutzstoffe. Technical report, FH Nordakademie, Elmshorn, 2004.

[12] Steve Winder. *Analog and digital filter design*. Newnes - Elsevier Science(USA), second edition, 2002.

[13] R. W. Stewart, I. Swan, and D. Schmid. Multi-channel active noise cancellation using the DSP56001. In *Proceedings of the IEEE*, pages 185–188. IEEE, 1993.

[14] Marco Jennifer Patrick, Hany Ferdinando, and Resmana. Active Noise Cancellation Control by Filtered-X Least Mean Square Method using DSP starter kit TMS320C35. Technical report.

[15] Richard G. Lyons. *Understanding Digital Signal Processing.* Prentice Hall, second edition, 2004.

[16] Peter Tummeltshammer, James C. Hoe, and Markus Püschel. Multiple Constant Multiplication By Time-Multiplexed Mapping of Addition Chains. In *Design Automation Conference*, volume 41, pages 826–829, 2004.

[17] Paulo S. R. Diniz. *Adaptive Filtering - Algorithms and Practical Implementation.* Kluwer Academic Publishers, Boston / Dordrecht / London, second edition, 1997.

[18] Eric H. Anderson and Jonathan P. How. Active Vibration Isolation Using Adaptive Feedforward Control. Number I-97115B. American Control Conference, 1997.

[19] Sen M. Kuo and Dennis Morgan. Review of DSP algorithms for Active Noise Control. In *Proceedings of the IEEE*, volume 87, pages 943–973. IEEE, 1999.

[20] Erick L. Oberstar. Fixed Point Representation And Fractional Math. Technical report, Oberstar Consulting, 2004–2005.

[21] Johann Blieberger, Johann Klasek, Alexander Redlein, and Gerhard-Helge Schildt. *Informatik.* Springer-Verlag, Wien / New York, first edition, 1996.

[22] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C.* Cambridge University Press, first edition, 1988–1992.

[23] MJL Technology Ltd. *MJL Stratix Datasheet*, 2003.

[24] Altera. *General Purpose PLLs in Stratix and Stratix GX Devices*, 2001.

[25] Andrew Y. Lin and Karl S. Gugel. Feasability of fixed-point adaptive filters in FPGA devices with embedded DSP blocks. In *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 157–160. Applied Digital Design Laboratory, Dept. of ECE, University of Florida, 2003.

[26] Microchip Technology Inc. *Analog-to-Digital Converter Design Guide*, 2005.

[27] Analog Devices Inc. *AD5320 12-bit DAC in SOT-23*, 2000.

[28] Michael Ring. MC 68HC11 Experiment Proposals. Michigan State University, Department of Electrical and Computer Engineering, 1999.

[29] Philips. *NE/SA/SE5532/5532A data sheet*, 1997.

[30] James Karki. Analysis of the Sallen-Key architecture. Technical report, Texas Instruments, 1999.

[31] AN733: A Filter Primer, 2005.

[32] C. Marven and G. Ewers. *A simple approach to Digital Signal Processing*. Alden Press Limited, Oxford, 1994.