Bachelorarbeit

David Hemmer

Eine SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern für positionsveränderliche virtuelle Schallquellen.

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

David Hemmer

Eine SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern für positionsveränderliche virtuelle Schallquellen.

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Wolfgang Fohl
Zweitgutachter: Prof. Dr. -Ing. Bernd Schwarz

Abgegeben am 09.09.2011

Thema der Bachelorarbeit

Eine SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern für positionsveränderliche virtuelle Schallquellen

Stichworte

Audio, AC97, digitale Signalverarbeitung, Embedded System, FPGA, FIR-Filter, Binaural, 3d Sound, räumliches Hören, HRTF, kopfbezogene Übertragungsfunktionen, Echtzeit-Interpolation, System-on-Chip, HW/SW Co-Design

Kurzzusammenfassung

Diese Bachelorarbeit befasst sich mit der Entwicklung eines Embedded Systems zur kontinuierlichen Interpolation zwischen HRTF-Filtern in Echtzeit. Durch die Feststellung der Kopfposition eines Zuhörers relativ zur Position der virtuellen Schallquelle mittels eines 3-Achsen-Kompasssensors wird die Interpolation zwischen den HRTF-Stützstellenfiltern gesteuert.

Der Entwicklungsprozess gliedert sich in die Entwicklung einer SoC-Plattform, welche die HRTF-Filterung in Echtzeit bearbeiten kann. Ein weiterer Entwicklungsschritt ist die Implementierung der C-Routinen zur Steuerung der HRTF-Filterung sowie zur Interpolation zwischen HRTF-Filtern. Zur Optimierung der HRTF-Filterung werden die interaurale Laufzeitdifferenz und die interaurale Pegeldifferenz getrennt und separat voneinander verarbeitet.

Title of the paper

A SoC platform for continuous interpolation of HRTF filters for variable position of virtual sound

Keywords

Audio, AC97, digital signal processing,Embedded System, FPGA, FIR-Filter, binaural, 3d sound, spatial hearing, HRTF, Head-Realated-Transfer-Function, realtime interpolation, System-on-Chip, HW/SW Codesign

Abstract

This thesis deals with the development of an embedded system for continuous interpolation of HRTF filters in real time. The interpolation between the HRTF filters is controlled by identifying the position of a listener's head relative to the position of the virtual sound source using a 3-axis compass sensor.

The development process is divided into the development of a SoC platform which is able to do the HRTF filtering in real time. Another part of the development is the implementation of the C routines to control the HRTF filter and to interpolate between the HRTF filters. To optimize the HRTF filtering the Interaural-Time-Difference (ITD) and the Interaural-Level-Difference (ILD) are separated and processed separately from each other.

Inhaltsverzeichnis

1 Einleitung

Die virtuelle und erweiterte Realität wird in Zukunft eine noch größere Rolle in unserem Alltag einnehmen. Schon heute gibt es in der Industrie viele Anwendungsbereiche für solche Systeme, wie zum Beispiel Simulatoren und digitale Produktentwicklungsanlagen. Aber auch im Heimbereich kommen immer mehr solcher Systeme zum Einsatz. Das Wohnzimmer wird durch interaktive Spielekonsolen zu einer virtuellen Welt. Durch neue Interaktionsmöglichkeiten und die dadurch resultierende Bewegungsfreiheit des Benutzers verschwimmt die reale Welt mit der virtuellen Welt. Durch Head-Mounted Displays können Benutzer heute mobil in visuelle virtuelle Welten eintauchen. Somit lässt sich das visuelle Sinnesorgan gut überlisten.

Das zweite wichtige Sinnesorgan des Menschen ist das Gehör. Dieses erlaubt uns viele Informationen über unsere Umgebung wahrzunehmen. Dabei kann das Gehör viele Informationen eines Raumes durch die Raumakustik erkennen. Wir Menschen können des Weiteren aber auch die Position von Schallquellen im Raum bestimmen. In diesem Bereich gibt es zurzeit viele Forschungsschwerpunkte, allerdings auch schon erste Produkte wie beispielsweise die Headzone Kopfhörerserie der Firma Beyerdynamic [Bey11]. Zum einem sollen in den Systemen die Realität so genau wie möglich nachgebildet werden, andererseits aber darf dabei nicht vernachlässigt werden, dass die Systeme die Berechnungen in Echtzeit durchgeführen müssen. Ein weiterer wichtiger Punkt ist die Mobilität der Systeme, um die Benutzerinteraktion mit der virtuellen und erweiterten Realität nicht zu beeinträchtigen.

Für die beiden Schwerpunkte Echtzeitfähigkeit und Mobilität eignen sich Embedded Systems besonders. Über eine SoC[1]-Plattform mit FPGA-Chip können die rechenaufwendigen Audiofilterungen in schnelle parallele Hardwarebeschleuniger ausgelagert werden. Ein Embedded System kann in Größe, benötigter Hardware und Ressourcenbedarf angepasst werden, was einen erheblichen Vorteil in Bezug auf die Mobilität eines solchen Systems ergibt. [Nat11]

Diese Bachelorarbeit reiht sich in eine Vielzahl von Forschungsausarbeitungen ein, die sich mit binauraler Klangerzeugung beschäftigen. Hierzu zählt zum Beispiel die Zerlegung der einzelnen HRTF-Filter (FIR-Filter hoher Ordnung) in ein Allpassfilter und ein minimalphasiges Filter geringerer Ordnung [Zöl02] sowie ressourcensparenden Interpolationmethoden von HRTFs [CL09].

Das Ziel dieser Arbeit ist die Entwicklung einer SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern für positionsveränderliche virtuelle Schallquellen. (siehe Abbildung 1) Dabei wird auf die Erkenntnisse aus der Masterarbeit von Jan Kuhr [Kuh10] aufgebaut.

[1] Abkürzungen sind im Abkürzungsverzeichnis im Anhang erläutert

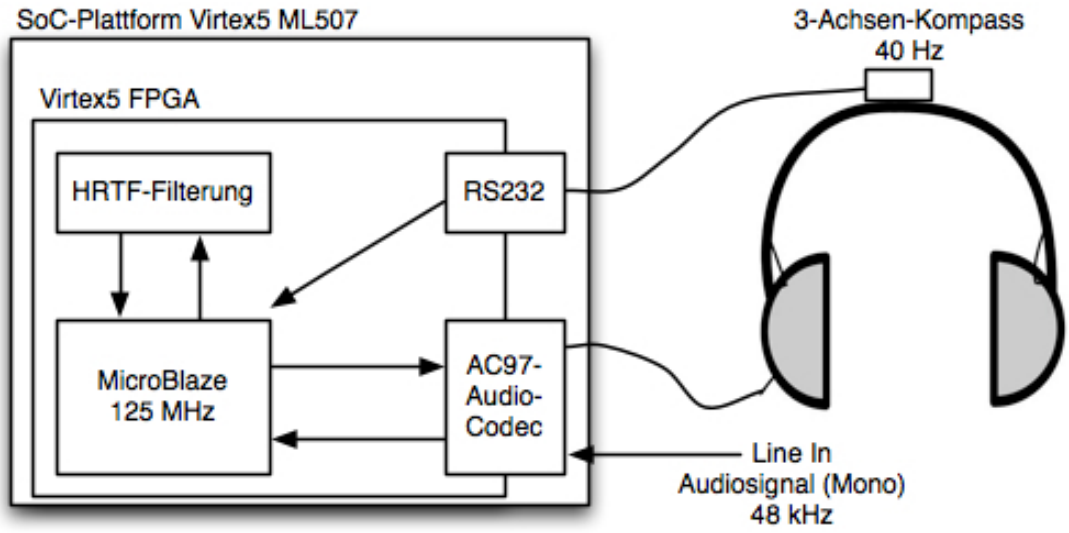

Abbildung 1: Übersicht des entwickelten SoCs mit 3-Achsen-Kompass und Stereo-Kopfhörer

Die Schwerpunkte dieser Arbeit konzentrieren sich auf:

- Entwicklung einer SoC-Architektur auf einem Virtex5, die für digitale HRTF-Filterung in Echtzeit ausgelegt ist.
- HRTF-Filterung mit getrennter Verarbeitung der interauralen Laufzeitdifferenz (ILT) und interauralen Pegeldifferenz (ILD). Dabei wird auf zwei HRTF-Messreihen zurückgegriffen.
- Softwareseitige Steuerung der HRTF-Filterung mit Interpolation zwischen den HRTF-Stützstellenfiltern.

Die theoretischen Grundlagen für diese Ausarbeitung werden in Kapitel 2 behandelt. In Kapitel 3 werden die Problemstellungen des Systems und dessen Lösungsansätze vorgestellt. Die Berechnungen der HRTF-Filterkoeffizienten mit dem Schwerpunkt der getrennten Verarbeitung der Laufzeitdifferenz und der Pegeldifferenz werden in Kapitel 5 behandelt. Die Entwicklung der SoC-Plattform wird in Kapitel 6 erläutert. Die umgesetzten Softwarekonzepte für das 3D-Audio-System werden in Kapitel 7 beschrieben.

2 Grundlagen

In diesem Kapitel sollen die wichtigsten Grundlagen für das Themengebiet dieser Arbeit erklärt werden. Auf diese Grundlagen wird im weiteren Verlauf dieser Ausarbeitung immer wieder Bezug genommen.

2.1 Binaurales Hören

Das menschliche Gehör ist eines der wichtigsten Sinneswahrnehmungsorgane. Dabei kann das menschliche Gehör akustische Ereignisse mittels Frequenz und Schalldruck wahrnehmen. Der Frequenzbereich von Schallwellen ist in einem Bereich von ca. 16 Hz bis ca. 20 kHz hörbar, wobei dieser auch vom Schalldruckpegel anhängig ist. Die Wahrnehmung von Sound ist von Mensch zu Mensch unterschiedlich und ist massgeblich vom Alter der Person abhängig.

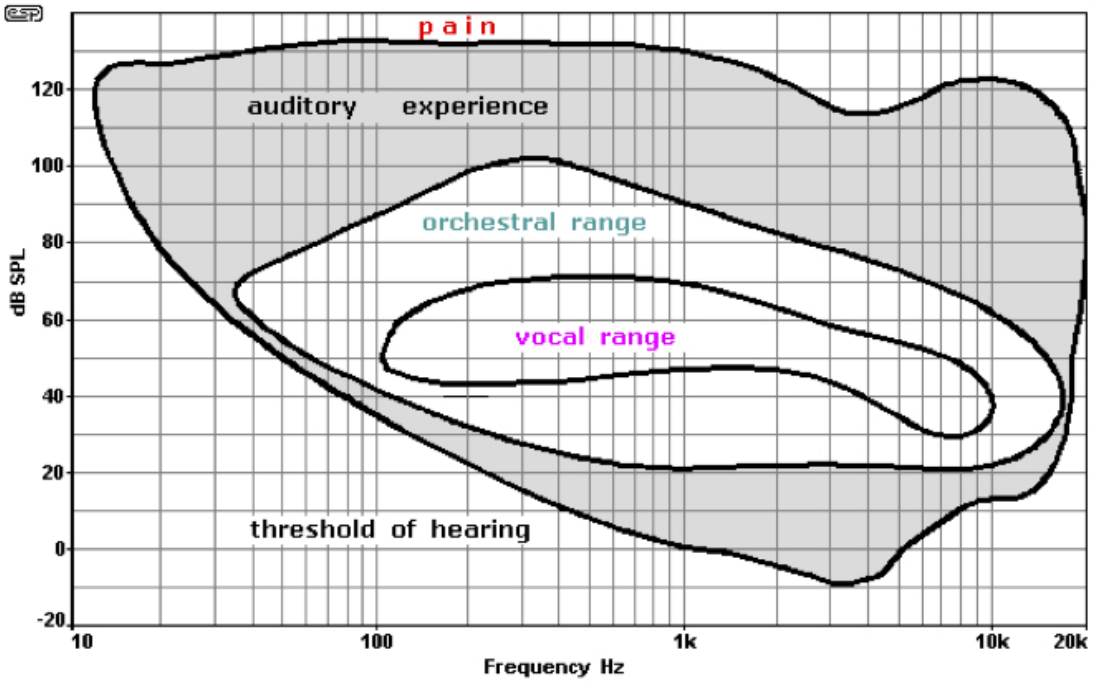

Abbildung 2: Hörbereich des Menschen Quelle: [Ell06]

Um Schall mit einer bestimmten Frequenz wahrzunehmen, muss diese ein Mindestschalldruckpegel aufweisen. Dieser ist bei 100 dB bei etwa 16 Hz und geht herunter bis zu -10 dB bei 3 kHz [2]. Der grau eingefasste Bereich in der Abbildung 2 stellt den Bereich dar, in

[2] die Frequenz bei der das Gehör am empfindlichlichsten ist

dem das menschliche Gehör Schallwellen wahrnimmt. Alles was über den grauen Bereich hinausgeht, ist für das menschliche Gehör schmerzhaft. Der Bereich unterhalb des grauen Bereiches, ist für den Menschen nicht wahrnehmbar.

Das menschliche Gehör bietet noch eine weitere Eigenschaft zum Hören. Es kann auch, die Richtung aus dem der Schall kommt, räumlich wahrnehmen. Dieses ist eines der wichtigsten Orientierungsinstrument des Menschen. Um die Richtung der Schallquelle zu orten, spielt die physikalische Struktur der beiden Ohren, in Kombination mit dem Gehirn, eine große Rolle.

Durch die Anordnung der beiden Ohren zueinander entsteht eine Laufzeitdifferenz. Diese Laufzeit zwischen beiden Ohren ist ein wichtiges Merkmal zur Ortung von Schallquellen. Dieses wird in der Literatur als Interaurale Laufzeitdifferenz (engl: ITD Interaural-Time-Difference) bezeichnet. Wenn eine Schallquelle im 90° Winkel zum Kopf steht (links vom Kopf), erreicht der Schall dieser Schallquelle zuerst das linke Ohr. Um das rechte Ohr zu erreichen wird der Schall um den Kopf herum geleitet. Daraus entsteht eine zusätzliche Wegstrecke. Dieser Zeitunterschied zwischen beiden Ohren (Δt) wird vom Gehirn ermittelt und unterstützt den Ortungsvorgang der Schallquelle. Die kleinste Laufzeitdifferenz, die das Ohr-Gehirn-System erkennt, liegt bei ca. 10 μs. Die maximale Laufzeitdifferenz, zwischen beiden Ohren beträgt ca. 0.63 ms und wird auch Hornbostel-Wertheimer Konstante genannt. Diese Zeitdifferenz zwischen beiden Ohren entspricht einem Ohrabstand bzw. eine Wegstreckendifferenz von ca. 21.5 cm.

Ein weiteres Erkennungsmerkmal für die Ortung von Schallquellen ist die Interaurale Pegeldifferenz (engl: ILD Interaural-Level-Difference). Diese beschreibt die Pegeldifferenz oder auch Lautstärkenunterschiede eines bestimmten Signals zwischen beiden Ohren. Die Differenz wird wie bei der Interauralen Laufzeitdifferenz durch die Eigenschaft des menschlichen Kopfes verursacht. Der Kopf hat eine Dämpfungswirkung auf den Schalldruck des Signals. Wenn ein Signal direkt auf das linke Ohr trifft, wird diese durch den Kopf gedämpft, bevor es auf der rechten Seite des Kopfes vom rechten Ohr wahrgenommen wird. Diese Dämpfung des Signals ist dabei auch noch frequenzabhängig. Das menschliche Gehör kann durch die Interaurale Pegeldifferenz eine Veränderung von bis zu 3° von der Kopfmitte erkennen.

Die Interaurale Laufzeitdifferenz und Interaurale Pegeldifferenz ist ein guter Ansatz für eine grobe Positionsbestimmung auf der horizontalen Ebene. Eine Verbesserung der Ortung von Schallquellen wird durch die freie Bewegung des Kopfes erzielt. Durch die ITD und ILD ist es nicht möglich zu unterscheiden ob eine Schallquelle von vorne oder hinten auf dem Kopf gerichtet ist. Um die Schallquelle dennoch zu lokalisieren bewegt der Mensch sein Kopf.

2.2 Head Related Transfer Funktion

Die Head-Related Transfer Function (HRTF) wird im Deutschen auch als kopfbezogene Übertragungsfunktion bezeichnet. Diese Funktion beschreibt den natürlichen Filterprozess, den das menschliche Gehör aufgrund seiner Beschaffenheit auf eintreffende Schallwellen anwendet. Die Filterung ist dabei abhängig von der Position der eingehenden Schallquelle, sowie der daraus resultierenden Interferenzen mit Ohrmuscheln, Kopf und Schultern. Wobei die Position sich durch die Entfernung und die Winkel der horizontalen und vertikalen Ebene zum Kopfmittelpunkt ändert. Diesen Vorgang kann man als die Berechnung der binauralen Impulsantwort bezeichnen. Demzufolge versucht die kopfbezogene Übertragungsfunktion, ein akustisch dreidimensionales Bild, durch der Intensitäts- und Laufzeitunterschiede, zu erzeugen.[Kuh10]

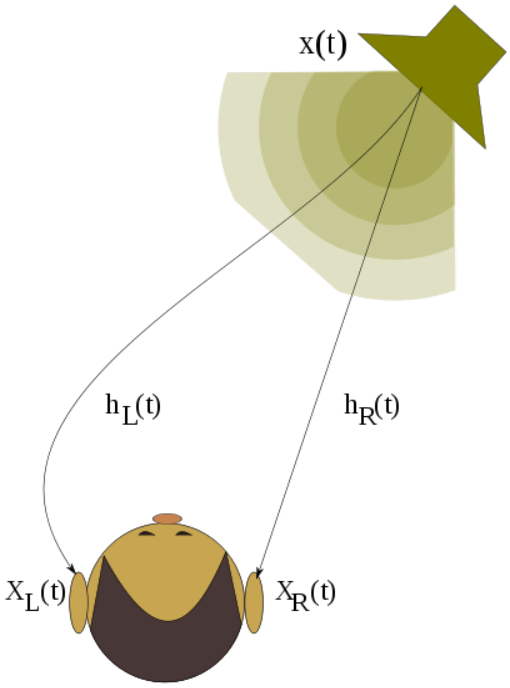

Abbildung 3: Head-Related Transfer Function Quelle: [Wik11]

Für die Messung der HRTF wird in einem reflexionsarmen Raum ein Testsignal auf ein Kunstkopf mit je einem Mikrofon in jedem Ohr aufgezeichnet. Der Kunstkopf bildet die physikalischen Eigenschaften des Menschen nach. Das Testsignal wird von verschiedenen Richtungen auf den Kunstkopf geschickt und von den beiden Mikrofonen in den Ohren aufgezeichnet. Die mit den Mikrofonen aufgenommen Signale entsprechen annähernd den

Schall-Signalen, wie wir Menschen dieses Testsignal hören würden. Durch Dekonvolution (inverse Faltung) der aufgezeichneten Signale mit den Testsignalen im Frequenzbereich erhält man die jeweilige HRTF. Wendet man die inverse Fourier-Transformation auf eine HRTF an, erhält man die kopfbezogene Impulsantwort (engl. Head- Related-Impulse-Response HRIR). Diese HRIRs können direkt als Koeffizientensätze für die FIR-Filterung 2.3 eines Audiosignals verwendet werden. Die Schallquelle eines beliebigen auf diese Weise gefilterten Audiosignals kann ein Hörer an dem Punkt orten, für den die Messung der HRIRs vorgenommen wurde. [Kuh10][Zöl02]

2.2.1 HRTF-Messreihe KEMAR vom MIT

Das “Massachusetts Institute of Technology“ kurz MIT hat 1994 eine umfassende Messreihe für Head Related Transfer Funktion (HRTF) durchgeführt. Der Name dieser Messreihe ist KEMAR und steht frei zur Verfügung und ist über die Internetseite des MIT herunterladbar [MIT11]. Die Messreihe beinhaltet die Impulsantworten für das rechte und linke Ohr. Bei den Messungen betrug die Entfernung zwischen dem Kunstkopf und der Schallquelle (Lautsprecher) 140 cm. Die Winkel aus denen Messungen durchgeführt wurden sind für die Höhenposition von -40° bis + 90° und für den Azimuth von 0° bis 355°. Das durch den Lautsprecher ausgestrahlte Signal war ein deterministisches zufallsgeneriertes weißes Rauschen mit einer Abtastrate von 44.1 kHz. Zu der Messreihe gibt es noch sehr viele hilfreiche MATLAB-Skripte, die auf der MIT-Internetseite zu finden sind. Im weiteren Verlauf dieser Arbeit wird diese Messreihe nur noch Kemar benannt.

2.2.2 HRTF-Messreihe aus der Bachelorarbeit S. Sima

Die Bachelorarbeit "HRTF Measurements and Filter Design vor a Headphone-Based 3D-Audio System“ von S. Sima bietet ein funktionsfähiges MATLAB-System zur Berechnung von HRTFs aus Kunstkopf HRIR Messungen. Dabei wurde fast der gleiche Messaufbau wie bei der Messreihe des MITs benutzt. Der Unterschied zum Messaufbau des MITs war, dass das ausgestrahlte Signal ein Sinussignal war. Bei diesen Messungen wurden der Phasengang und die Pegeldifferenz gemessen. Es gibt zwei Messreihen. Die erste Messreihe hat für die Elevation -41°, +45° und 0° je die Messung für die Azimuth von 0° bis 180° mit einer Schrittweite von 30°. Die zweite Messreihe hat für die Elevation +37° und 0° je die Messungen für die Azimuth von 0° bis 180° mit einer Schrittweite von 15°.[Sim08]

2.3 Digitale Signalfilterung mit FIR-Filtern

Um HRTF nachzubilden benötigt man digitale Filter. Die einfache Struktur der beim HRTF Verfahren ermittelten Filterkoeffizienten, beziehungsweise die Tatsache, dass die HRIR eine endliche Impulsantwort ist, legt die Verwendung eines Finite-Impulse-Response Filters (FIR) nahe.[Kuh10] Die allgemeine Beschreibung von FIR-Filtern ist im Zeitbereich durch die Differenzengleichung gegeben, die jeden Eingangssignalwert x[n] mit einem speziellen Koeffizienten c_k gewichtet. Die Anzahl der zu speichernden vorherigen Eingangssignalwerte x[n-k] ist die Filterordnung N, sodass insgesamt für jede Ausgangssignalaktualisierung L = N+1 Produkte zu addieren sind (L ist die Filterlänge).[RS09]

$$y[n] = \sum_{k=0}^{N} c_k * x[n-k] \tag{1}$$

Die Impulsantwort ist das Ausgangssignals eines FIR-Filters bei dem am Eingang ein Einheitsimpuls zugeführt wird. Die anderen zugeführten Pulse sind '0'. Die Antwort des Filters ist gleich seinen Filterkoeffizienten. Durch die Impulsantwort wird getestet ob der Filter richtig implementiert wurde.
Die Sprungantwort eines FIR-Filters ist das Ausgangssignal, bei dem am Eingang immer einem maximal positiven Puls zugeführt wird. Das Ausgangssignal zeigt ob es Überlaufe in der internen Addition der Multiplikation auftreten. Die Sprungantwort ist das zeitliche Integral der Impulsantwort.

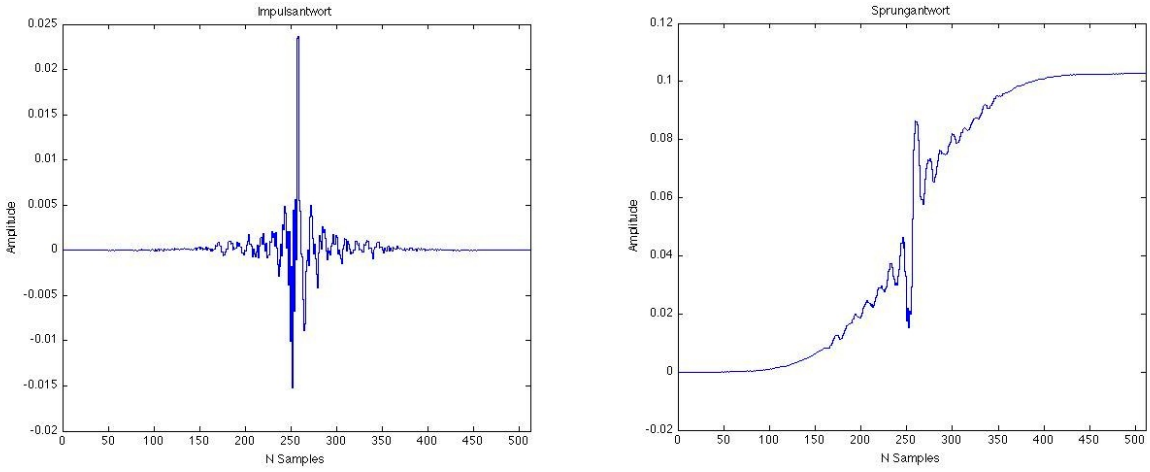

Abbildung 4: Impuls- und Sprungantwort eines FIR-Filters

Die Gruppenlaufzeit ist die Zeit, die ein Signal vom Eingang des Filters bis zur Veränderung am Ausgang braucht. Dabei ist die Gruppenlaufzeit frequenzabhängig. Filter mit konstanter Gruppenlaufzeit haben die positive Eigenschaft, dass sie Signale im Durchlassbereich nicht verzerren, sondern nur verzögern.[Grü08]

2.3.1 Normierung von Filterkoeffizienten

Mit der Filterskalierung oder auch unter dem Begriff Normierung der Koeffizienten sollen Überlaufeffekte in der Addition der Produkte verhindert werden. Ein Überlauf im positiven Wertebereich ändert das Vorzeichenbit der 2er-Komplement-Zahl und sie wird zur negativen Zahl verfälscht. Beim Überlauf im negativen Wertebereich ist dieser Effekt umgedreht.

Ein Skalierungsansatz ist, dass das Ausgangssignal y[n] bei begrenzten Eingangssignal |x[n]|<= 1 (absolute Intervallgrenze) immer innerhalb des gewählten Intervalls bleibt.

$$|y[n]| \leq \sum_{k=0}^{N} |c_k| \tag{2}$$

Diese Aussage gilt, da im Worst-Case |x[n]| maximal 1 ist. Wird die rechte Seite mit modifizierten Koeffizienten kleiner 1 eingestellt, dann bleibt das FIR-Filter überlauffrei. Dieses Verfahren wird L_1-Norm genannt.

$$L_1 = \sum_{k=0}^{N} |c_k| \tag{3}$$

Um zu gewährleisten, dass die rechte Seite der Gleichung immer kleiner 1 ist, werden die Filterkoeffizienten durch L_1 dividiert. Daraus ergibt sich eine Verkleinerung der Filterverstärkung. Die L_1-Norm ist der konservativste Ansatz, um Filterkoeffizienten zu normieren.

Ein weiteres Verfahren zur Normierung von Filtern ist die L_2-Norm. Bei dieser Norm kann es zu Überläufen in der Addition der Produkte kommen. Dabei ist der Faktor durch den die Koeffizienten geteilt werden, wie folgt definiert.

$$L_2 = \sqrt{\sum_{k=0}^{N} |c_k|^2} \tag{4}$$

3 Problemstellungen und Lösungsansätze für die SoC-Plattform

In diesem Kapitel werden die Problemstellungen des zu entwickelnden Echtzeit-Audio-Systems aufgezeigt, sowie die Erläuterung der Lösungsansätze.

3.1 Aufgabenverteilung in der SoC-Plattform um die Echtzeitaudioverarbeitung zu gewährleisten

Im späteren System sollen die HRTF-Filterung in Echtzeit bei einer Abtastrate von 48 kHz verarbeitet werden. Dieses hat zur Folge, dass die komplette Audioverarbeitung in unter 20,8 μs abgearbeitet werden muss. Das entspricht ca. 2600 Takten, wenn das Virtex5-System mit dem Maximaltakt von 125 MHz betrieben wird. Deswegen wird überlegt, wie die Aufgabenverteilung im Embedded System verteilt werden muss, um die Echtzeitaudioverarbeitung zu sichern.

Die komplette Echtzeitsignalverarbeitung wird vollständig innerhalb der ISR des Audio-Codecs abgearbeitet. Dabei ist zu gewährleisten, dass die Audio-ISR deutlich schneller ist, als das ein neues Audiosample im System eintrifft. Um zu garantieren, dass die ISR schneller arbeitet, werden die rechenaufwendigen Aufgaben, wie zum Beispiel die HRTF-Filterung, in Hardwarebeschleuniger ausgelagert. Des Weiteren werden alle Steueraufgaben und Bedingungen aus der ISR in die Mainroutine ausgelagert. Folglich ist die Aufgabe der Mainroutine das System auf dem aktuellsten Zustand zu halten. Dazu zählt, das Nachladen von Filterkoeffizienten, das Abfragen der Position des Kompasses, sowie die Benutzerinteraktion. Die aktuellen Systeminformation werden der Audio-ISR über Variablen von der Mainroutine bereitgestellt. Somit ist gewährleistet, dass die ISR schnellstmöglich arbeitet.

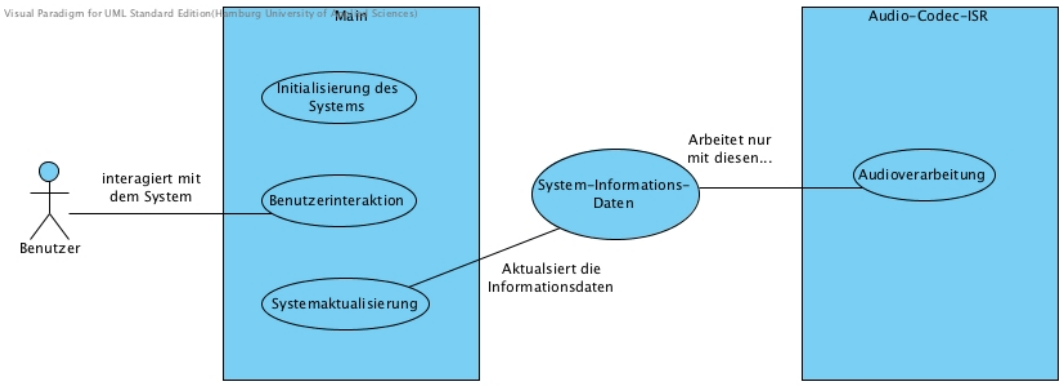

Abbildung 5: Aufgabenverteilung im Echtzeit-Audio-System

3.2 Konzepte der HRTF-Filterverwaltung

Um im späteren System die HRFT-Filterung durchzuführen, gibt es verschiedene Konzepte.

Da für jede Winkelposition (vertikal und horizontal) sich die HRTF-Filterung verändert, könnten mittels MATLAB aus den HRTF-Messreihen für jede Position die entsprechenden Filterkoeffizienten errechnet werden. Dieses würde eine große Anzahl von Filterkoeffizienten zur Folge haben. Bei der Annahme das die HRTFs im Evaluationwinkel-Bereich von -30° bis +30° und einmal um den Kopf (360°) in 1°-Schritten dem System zur Verfügung gestellt werden sollen, würden dieses 43200 Filterkoeffizientensätze zur Folge haben. Das ist in keiner Weise in der zu entwickelnden SoC-Plattform zu realisieren.

Ein anderer Ansatz ist es zu bestimmten Winkelpositionen die HRFT-Filterkoeffizeintensätze dem System zur Verfügung zu stellen. Mit diesen Stützstellen kann das System durch Interpolation für jede Position das zu erwartende Filterergebnis errechnen. Die SoC-Plattform könnten die vertikale Ebene durch 3 Filterkoeffzientensätze zur Verfügung gestellt bekommen. Die horizontale Ebene um den Kopf herum, könnte dem System mit einer Schrittweite von 30° bereitgestellt werden. Mit dieser Annahme kann die Filteranzahl deutlich reduziert werden. Es würden für beide Ohren jeweils 3 vertikale mit dazugehörigen 12 horizontalen Filtern zur Verfügung gestellt. Demnach würde die SoC mit nur 72 Filtersätzen auskommen. (3 vertikale Ebenen mit je 12 horizontalen Filtern für je das rechte und linke Ohr)

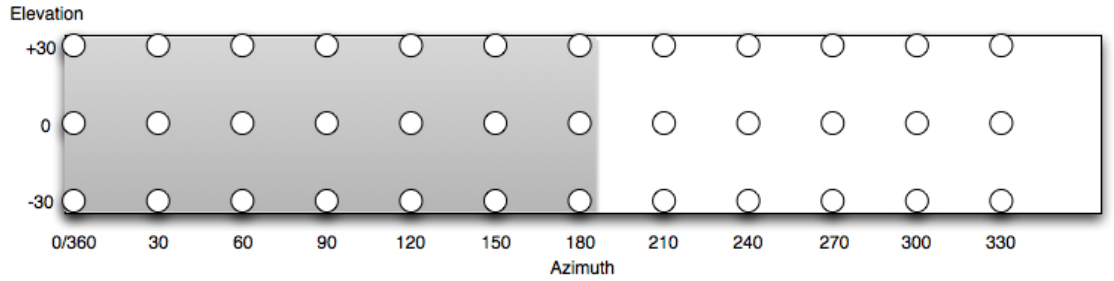

Abbildung 6: HRTF-Stützstellen-Filter Anordnung in der SoC-Plattform

Durch einfache Spiegelung der Filterkoeffizienten für den linken und rechten Kanal, reduziert sich die Anzahl der Filter pro HRTF auf 7 Filterkoeffizientensätze pro Ebene. Demzufolge würden für die vertikale Ebenen nur die Filtersätze von 0° bis 180° benötigt (graue Bereich Abbildung 6). Positionen größer 180° lassen sich durch die Spiegelung der Kanäle errechnen. Das hat zur Folge, dass nur noch 42 Filtersätze dem System zur Verfügung gestellt werden müssen. (3 vertikale Ebenen mit je 7 horizontalen Filtern für je das rechte und linke Ohr, ergibt zusammen 42 Filter)

Diese 42 Filter könnten mit Hilfe von FIR-Filter auf Basis DSP-Einheiten des Virtex5 ML507 ohne Probleme realisiert werden. Durch die Verwendung der in der Masterarbeit von Jan Kuhr [Kuh10] entwickelten FIR-Stereo-Filter, wird der rechte und linke Kanal zusammen in einem FIR-Stereo-Filter gefiltert. Dadurch würde sich die Filteranzahl auf 21 FIR-Stereo-Filter reduzieren. Mittels Interpolation wird für jede Position das Filterergebnis aus diesen 21 FIR-Stereo-Filter errechnet. Da im späteren System die HRTF in Echtzeit verarbeitet werden sollen, müssten die Filter über ein schnelles Businterface angeschlossen werden. Dazu bietet der MicroBlaze, der auf dem Virtex5 arbeitet, nur 16 FSL-Schnittstellen. Infolgedessen ist dieser Ansatz nicht ohne Weiters umsetzbar.

Um für eine bestimmte Position das Filterergebnis zu errechnen, benötigt man dazu die 4 Filter-Quellnachbarn, die um die Position herumliegen. Somit könnte jede Position aus diesen 4 HRTF-Filterkoeffizientsätzen berechnet werden. Bei einer Änderung der Position müssten die Filter geändert werden. Das ist mit den entwickelten FSL-FIR-Stereo-Filtern machbar, da diese über die Möglichkeit verfügen ihre Koeffizentensätze im laufenden Betrieb auszutauschen. Das Nachladen der Koeffizienten eines Filters benötigt in laufenden Betrieb ca. 160 μs. (Messung siehe Anhang C) Im schlechtesten Fall sind 3 Filter zu aktualisieren. Das entspricht einer vertikalen und horizontalen Positionsänderung über die Grenzen des aktuellen Filterbereiches. Während der Aktualisierung der Filter kann die Audioverarbeitung nicht mehr garantiert werden. Um dieses Problem zu umgehen, könnten zwei dieser 4-Filterblöcke implementiert werden. Bei einer Positionsänderung könnte ein Filterblock aktualisiert werden. Solange würde noch mit dem alten Filterblock gearbeitet. Nach Durchführung des Updates, könnte auf den aktuellen Filterblock umgeschaltet werden. Hierbei ist zu beachten, dass immer alle Filter mit einem neuesten Audiosample versorgt werden, da sonst erst das Ergebnis nach der Filterlängenanzahl von Samples richtig ist. Diese Methode hat zum Vorteil, dass nur 8 FIR-Stereo-Filter ins System für die HRTF Filterung integriert werden müssen. Der Nachteil ist, dass ziemlich häufig die Filterkoeffizienten neu geladen werden müssen. Dieses Update wurde im normalen Betrieb ca. 480μs dauern. Des Weiteren ist der algorithmische Aufwand sehr groß. Es müsste auch ein komplexer Aufwand in der ISR betrieben werden da geschaut werden muss, aus welchen Filtern das Filterergebnis errechnet werden muss.

Ein weiterer Ansatz wäre, dass die Filter einer vertikalen Ebene immer zusammengefasst würden. Damit brauchte man immer 2 Filterebenen aus 7 HRTF-Filtern, um für die Position das Ergebnis zu interpolieren. Ein Filteraustausch wird nur stattfinden, wenn die mittlere vertikale Ebenen durchbrochen wurde. Ein großer Vorteil dieses Verfahren ist, dass die Änderungen immer durch die mittlere Ebene verlaufen. Während der Filteraktualisierung kann mit dieser Ebene weitergearbeitet werden. Somit wird nie mit Filtern gearbeitet, die zurzeit mit neuen Filterkoeffizienten ausgestattet werden. Im Vergleich zu dem davor beschriebenen Verfahren, sind zwar immer 7 Filter nachzuladen, aber es ist davon auszugehen, dass die mittlere Ebene deutlich weniger oft durchbrochen werden wird, da der Kopf mehr horizontal Bewegungen macht als Vertikale. Daraus folgt, dass 14 FSL-FIR-Stereo-Filter

benötigt werden. Somit lassen sich im späteren System 2 weitere FIR-Filter anschließen, die für andere Aufgaben nutzbar sind. (Verzögerungsglied, Kopfhörerkompensation).

3.3 Echtzeitinterpolation zwischen den HRTF-Stützstellenfilter

Da die HRFT-Filterergebnisse im späteren System über Stützstellen errechnet werden sollen, wird das System für die entsprechende Position das Ergebnis interpoliert. Dabei wird das HRFT-Ergebnis durch lineare-Interpolation aus den vier Quellnachbarn errechnet. Siehe Abbildung 7.

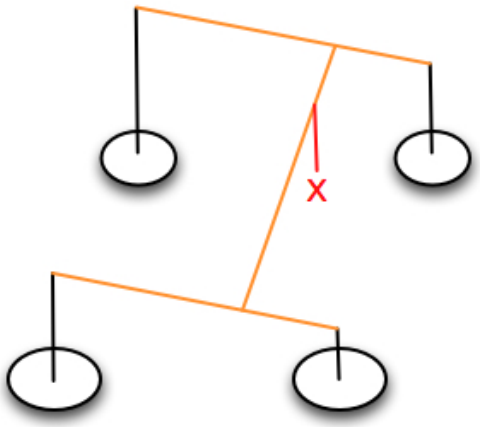

Abbildung 7: bilineare Interpolation zwischen 4 HRTF-Stützstellen

Dieses Verfahren nennt sich bilineare Interpolation und ist ein gängiges Verfahren [DSAS11] zur Interpolation zwischen Stützstellen. Normalerweise wird diese Berechnung in einem Fließkommabereich zwischen 0,0 und 1,0 berechnet, sodass das Mischverhältnis sehr präzise durch den Nachkommabereich dargestellt wird. Dieses ist ein Problem, da der MicroBlaze in seiner Standardeinstellung eine Integer-ALU ist. Die Berechnung mit Floiting-Point wäre zwar auch mit dieser ALU realisierbar, aber würde deutlich mehr Zeit in Anspruch nehmen. Folglich wäre eine Echtzeit-Audio-Verarbeitung von HRTFs nicht realisierbar. Der MicroBlaze könnte mit einer Floating-Point-ALU ausgerüstet werden. So könnte die Berechnung beschleunigt werden. Der Hardwarebedarf würde dabei aber deutlich größer ausfallen. Ein besserer Ansatz ist es durch geschicktes Multiplizieren und Dividieren diese Berechnungen zu realisieren. Da jedes Audio-Sample nur 16 Bit lang ist und die MicroBlaze-Recheneinheit eine 32 Bit ALU, bleiben für die Berechnung noch 16 Bit über. Die Division ist eine rechenaufwendig Operation für die ALU und wird im späteren Echtzeit-Audio-System durch Bit-Schiebe-Operation ersetzt. Die dadurch erzielbare Genauigkeit reicht trotzdem für dieses System aus.

4 Entwicklungsumgebungen des SoC's

Im Rahmen dieser Bachelorarbeit wird auf die Erkenntnisse, die in der Masterarbeit von Jan Kuhr [Kuh10] erworben wurden, aufgebaut. Daraus folgt, dass die Hardware sowie Software Ebene neu aufgesetzt werden muss. Dies bedarf der Verwendung von unterschiedlichen Entwicklungsplattformen.

4.1 SoC-Plattform Xilinx Virtex5 ML507

Als Hardwareplattform für die Entwicklung wird das Xilinx Virtex5 ML507 ausgewählt, welches zuvor auch schon bei der Masterarbeit von Jan Kuhr zum Einsatz gekommen ist. Dieses Entwicklungsboard ist für System-On-Chip (SoC) Systeme ausgelegt. Es ermöglicht leistungsfähige parallelisierbare FPGA-Hardwaremodule einzubinden. Der integrierbare Softprozessor (MicroBlaze) kann diese FPGA-Hardwaremodule über Software ansprechen. Der Vorteil eines Softprozessors wie dem MicroBlaze ist, dass dieser im FPGA aufgebaut wird und damit für das entsprechende System optimiert werden kann. Der MicroBlaze ist ein RISC Microcontroller und kann auf dem Virtex5 mit einem Maximaltakt von 125 MHz betrieben werden. Im Fall dieser Arbeit werden die komplizierten algorithmischen Berechnungen in schnelle FPGA-Module ausgelagert und direkt als IP-Core in den Softprozessor integriert. Somit übernimmt der Prozessor logiklastige Steueraufgaben.

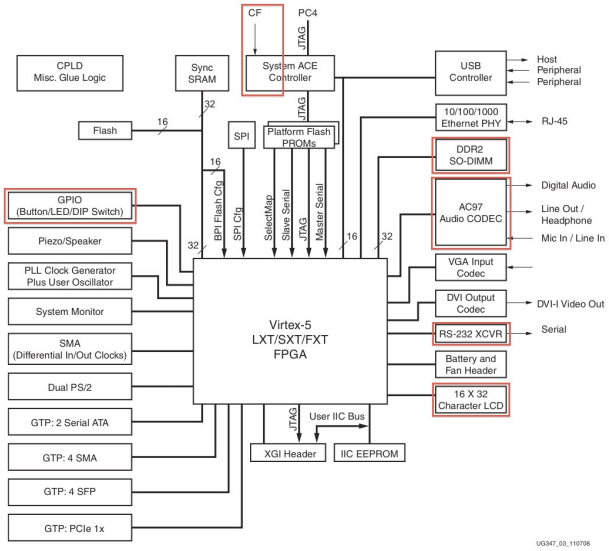

Abbildung 8: Blockdiagramm Virtex5 ML507 [Xil11b]

Wie die Grafik 8 zeigt, ist dieses System sehr flexibel einsetzbar und lässt sich für den Aufgabenbereich in dem es eingesetzt werden soll individuell konfigurieren. Für die zu

entwickelnde SoC-Plattform werden folgende Komponenten benötigt.

- AC97 Audio Codec
- DDR2-SO-DIMM für Programm-Code und die HRTF-Koeffizientensätze
- CF-Card Schnittstelle zum Einlesen der Koeffizientensätze für die HRTF-Filter
- RS-232 für Ausgabe an einem Terminal sowie die Kommunikation mit dem Kompass
- LCD Display für Ausgaben des Systems
- GPIO wie Push-Buttons, LEDs und Dip-Switches

4.2 Xilinx Platform Studio

Die Xilinx Platform Studio (XPS) Entwicklungsumgebung ist ein Teil aus dem Xilinx-Embedded-Development-Kit 12.4 (EDK). Diese ermöglicht Hardware-Entwicklern ein sehr individuelles Embedded-Prozessor-System zu entwickeln. Die grafische Oberfläche (Abbildung 9) bietet viele Assistenten, wie z.B. den Base System Builder, mit dessen Hilfe der Benutzer sehr bequem sein System entwickeln kann.

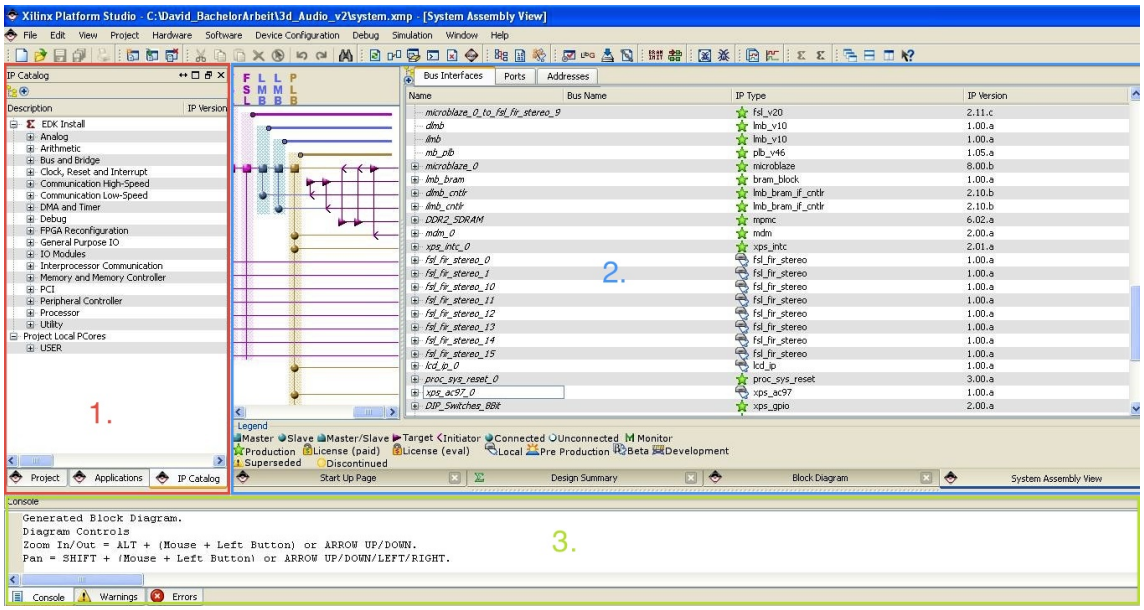

Abbildung 9: Benutzeroberfläche Xilinx Platform Studio (XPS)

Die XPS Oberfläche teilt sich in 3 Bereiche auf.
Der erste Bereich (1.) ist der Projekt-Informations-Bereich. Dieser beinhaltet drei Unterbereiche. Der Applications Tab, IP Catalog und Project Tab. Der Applications Tab bietet eine kleine Entwicklungsumgebung für Software Projekte. Eine bessere Entwicklungsumgebung für Software ist das SDK, welches auch als C-Entwicklungsumgebung in dieser Arbeit zum

Einsatz gekommen ist. [4.3]
Im IP Catalog werden alle IP-Cores des EDK und die selbst entwickelten IP-Cores aufgelistet.
Der Project Tab ist sozusagen der Projekt Explorer. Er listet Projekt-Dateien wie die Microprozessor Hardware Specification (MHS), Microprozessor Software Specification (MSS) und die User Constraints File (UCF) auf.

Der zweite Bereich der Oberfläche (2.) ist die System Assembly View. Die linke Seite dieses Bereiches zeigt die Busübersicht des Systems. Über den Tab Bus-Interface lassen sich IP-Core an bestimmte Bus-Systeme anbinden. Der Port Tab gibt die Möglichkeit die Verdrahtung des Systems zu bearbeiten. Über den Tab Addresses lässt sich das System mit entsprechenden Addressen versehen. Dieses ist notwendig, da der MicroBlaze ein Memory Mapped IO System ist.

Der letzte Bereich (3.) ist die Konsolenausgabe. Auf ihr werden die Ergebnisse der Synthese ausgegeben.

Für genauere Informationen ist das Buch "Embedded Systems Design with Platform FPGAs" von Ron Sass Andrew und G. Schmidt zu empfehlen[SS10].

4.3 Xilinx Software Development Kit

Das Xilinx Software Development Kit (SDK) bietet eine Eclipse Entwicklungsumgebung zum Erstellen von C/C++ Software Projekten für eingebettete MicroBlaze-Prozessoren. Das SDK arbeitet direkt mit dem Hardware Design des XPS und erzeugt aus diesen Daten ein Board Support Packages (BSP). Im SDK gibt es weitere Tools, wie zum Beispiel einen Linker-Script-Generator und einen komfortablen Debugmodus.

4.4 MATLAB Entwicklungsumgebung

Die MATLAB Entwicklungsumgebung [Mat] und die gleichgenannte Programmiersprache erlauben es numerische Lösungen für mathematische Probleme zu finden. MATLAB eignet sich auch für die Entwicklung von Algorithmen, da diese sofort in MATLAB getestet werden können. Dabei stellt MATLAB eine Vielzahl von Visualisierung- und Analyse-Tools zur Verfügung, mit deren Hilfe die Ergebnisse grafisch dargestellt werden können. Viele weitere Toolboxen machen MATLAB zu einem komfortablen Werkzeug, um zum Beispiel Signal- und Bildverarbeitungs-Algorithmen zu testen.

5 MATLAB Prototypenentwicklung

Die an der HAW in der Bachelorarbeit von S. Sima erzeugten HRTF [2.2.2] und die von Massachusetts Institute of Technology unter dem Namen Kemar veröffentlichen HRTF [2.2.1] liegen dieser Ausarbeitung zu Grunde. In beiden HRTFs wird die Laufzeitdifferenz und Pegeldifferenz zusammen in einem Koeffizienten-Filtersatz verarbeitet. In dieser Arbeit sollen die Laufzeitdifferenz und Pegeldifferenz getrennt voneinander verarbeitet werden. Ein Vergleich dieser vier HRTF-Koeffizientensätze soll mit der SoC-Plattform möglich sein. Um diese vier Koeffizientensätze komfortabel zu erzeugen, wurde der MATLAB-File "main.m" geschrieben, über welches die Berechnung der entsprechenden HRTF gesteuert wird.

In den folgenden Unterkapiteln werden die Berechnungen der HRTF erläutert. Das Kapitel 5.3 beschreibt den Ansatz der getrennten ILD und ITD Filterung.

5.1 Berechnung der KEMAR Filterkoeffizienten

Wie schon im Kapitel 2.2.1 beschriebenm gibt es eine große Messreihe für HRIR vom MIT. In dieser Messreihe wurden Messungen für die Elevation -30°, 0° und +30° und für den Azimuth in 30° Schritten um den Kopf gemacht. Um die Koeffizientensätze für die gewünschten Position zu errechnen wurde das Skript "calculate_hrir_kemar.m" benutzt. Dieses errechnet die Koeffizienten für die Filter durch Faltungsmethoden im Frequenzbereich.

```
sp_fft= fft(speaker,NFILT);
ir_fft= fft(iresp,NFILT);
coeff = ifft( sp_fft ./ ir_fft ) ;
coeff = coeff .* wnd;
coeff = coeff / norm_faktor;
```

Dabei wird über ***NFILT*** die gewünschte Filterlänge eingestellt. In diesem Projekt wird eine Filterlänge von 512 Koeffizienten gewünscht. Dieses hat sich in der Masterarbeit von Jan Kuhr als eine gute Filterlänge für HRTF-Filter herausgestellt. Die Variable ***speaker*** beinhaltet das Signal, welches über den Lautsprecher ausgestrahlt wurde. ***iresp*** ist das Signal, welches je nach Position des Kopfes zum Lautsprecher von den Mikrophonen aufgenommen wurde. Durch Faltung dieser Signale erhält man die Koeffizienten für das Filter. Die Koeffizienten werden daraufhin mithilfe der Fensterfunktion ***wnd*** und durch Normierung mit ***norm_faktor*** für die SoC-Plattform aufbereitet.

Da bei der Aufnahme der HRIR mit einer Abtastrate von 44,1 kHz gearbeitet wurde und die spätere SoC-Plattform mit einer Abtastrate von 48 kHz arbeitet, müssen die errechneten Koeffizientensätze noch angepasst werden. Diese sieht wie folgt aus.

```
coeff = resample(coeff,48000,44100);
coeff = coeff(1:NFILT);
```

Das sogenannte Resamplen der Koeffizienten wird mit der MATLAB-Funktion *resample()* gemacht. Durch das Resamplen von 44.1 kHz auf 48 kHz erhält man einen längeren Koeffizientensatz. Darum wird die Variable ***coeff*** auf eine maximale Länge von ***NFILT*** verkleinert. Dieses hat kaum eine Auswirkung auf die Filtereigenschaft des Filters, da die wichtigsten Koeffizienten in der Mitte des Koeffizientensatz liegen.

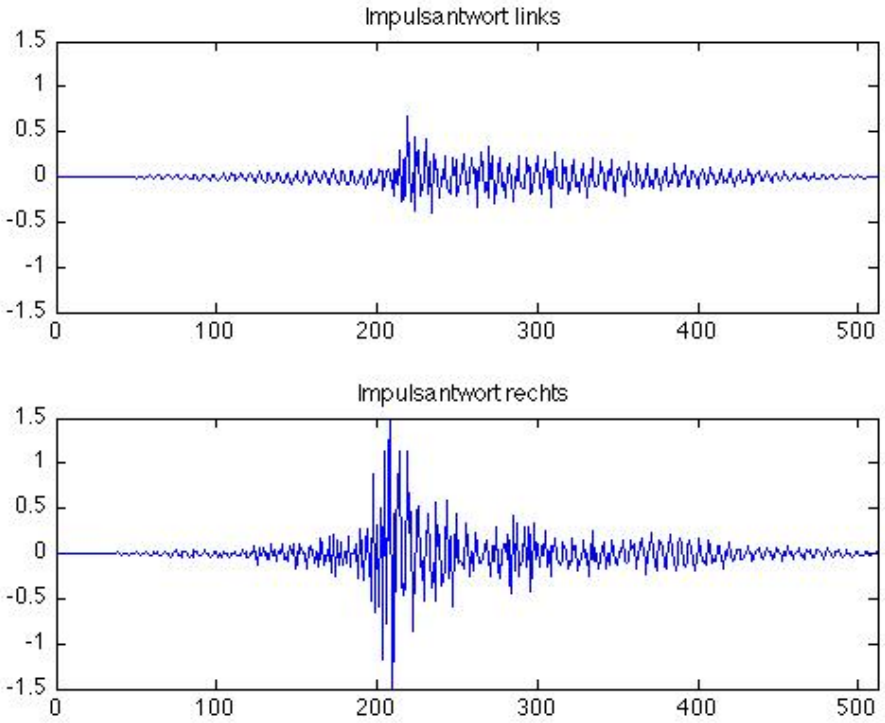

Abbildung 10: linke und rechte Impulsantwort der Kemer-HRTF bei e0a30

Die Grafik 10 zeigt die linke und rechte Impulsantwort der Kemar HRTF bei Elevation von 0° und Azimuth von 30°. Dabei sind die ITD und ILD sehr gut zu erkennen. Die rechte Impulsantwort hat eine viel größere Amplitude im Vergleich zur linken Impulsantwort. Dieses ist die Eigenschaft der interaurale Pegeldifferenz. Die Laufzeitdifferenz zwischen beiden Impulsantworten ist ebenfalls gut zu erkennen. Die Impulsantwort des rechten Kanals fängt bei ca. 200 Samples an, wobei die Impulsantwort des linken Kanals um ca.

16 Samples verzögert anfängt. Diese 16 Samples erzeugen eine Zeitdifferenz von 333 μs. Diese entspricht eine Schallwegstrecke von ca. 11,3 cm und erscheint mit der in Kapitel 5.3.1 physikalisch errechneten Laufzeitdifferenz zwischen beiden Ohren als plausibel.

5.2 Berechnung der SIMA Filterkoeffizienten

An der HAW Hamburg wurde in der Bachelorarbeit “HRTF Measurements and Filter Design for a Headphone-Based 3D-Audio System“ [Sim08] von S. Sima eine HRTF-Messreihe durchgeführt. Um aus dieser Messreihe FIR-Filter Koeffizienten zu errechnen wird das MATLAB-Skript “calculate_coeff_sima.m“ aus der Bachelorarbeit von S. Sima benutzt. Auf eine Erläuterung dieses MATLAB-Skripts wird verzichtet, weil dieses ausführlich in der Bachelorarbeit von S. Sima erklärt wird.

Die nächste Grafik (Abb. 11) zeigt die Impulsantworten des rechten und linken Kanals auf der Elevation Ebene 0° mit einem Azimuth Winkle von 90°.

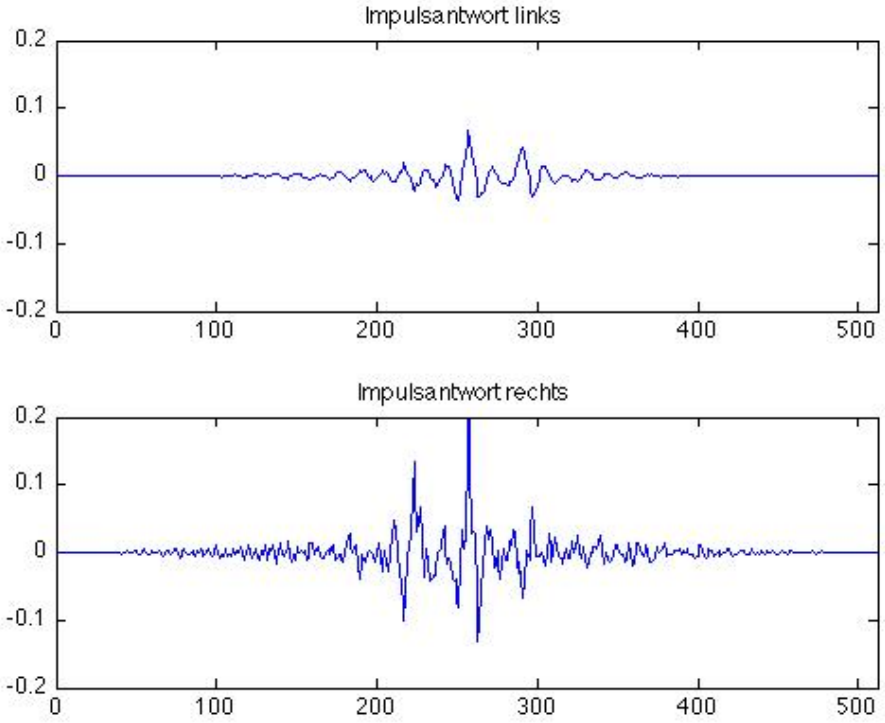

Abbildung 11: linke und rechte Impulsantwort der Sima-HRTF bei e0a90

Die Grafik zeigt ebenfalls sehr gut die interaurale Laufzeitdifferenz und die Pegeldifferenz.

5.3 HRTF-Filteransatz mit getrennten Laufzeitdifferenzen- und Pegeldifferenzen-Filterung

In dieser Arbeit soll die Laufzeitdifferenz und die Pegeldifferenz getrennt voneinander verarbeitet werden. Daraus ergibt sich, dass die SoC-Plattform für diese Anwendung zwei Filter-Typen benötigt. Ein FIR-Filter, der die Filterung für die Pegeldifferenz übernimmt, sowie ein Filter, der die Laufzeitdifferenz erzeugt.
Durch die getrennte Verarbeitung der ILD und ITD ergibt sich ein wesentlicher Vorteil. Dieser besteht darin, dass die Filter für die Pegeldifferenz eine konstante Gruppenlaufzeit haben. Filter mit konstanten Gruppenlaufzeiten verzerren Signale im Durchlassbereich nicht, sondern verzögern diese nur. [Grü08] Die Laufzeitdifferenz wird über ein Verzögerungsglied realisiert.

5.3.1 Physikalischer Ansatz zur Bestimmung der Laufzeitdifferenz

Um eine Vorstellung für die Laufzeitdifferenz zwischen dem linken und rechten Ohr zu bekommen, wurde sich dem Problem mithilfe eines physikalischen Modells genähert.

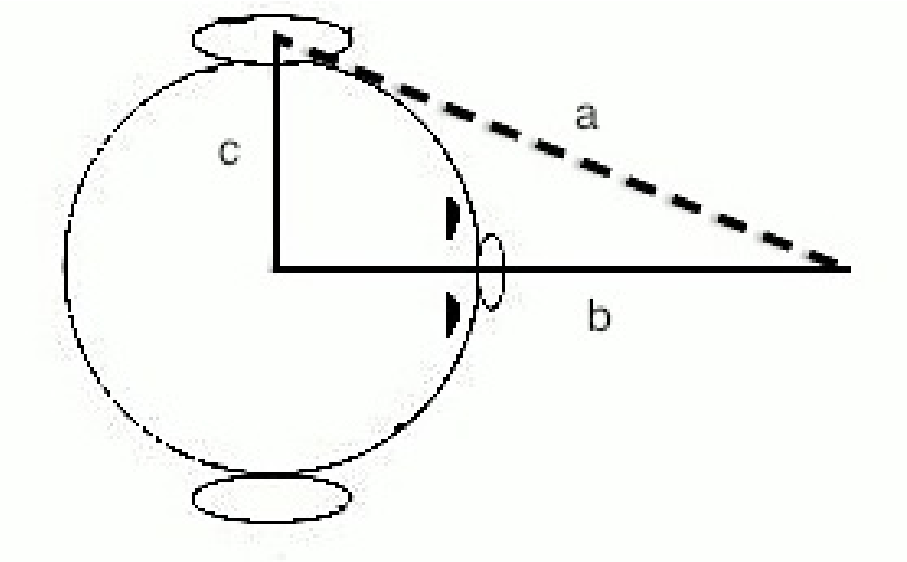

Abbildung 12: Physikalischer Ansatz für die Laufzeitdifferenz über den Kosinussatz

Gesucht ist die Entfernung der Schallquelle zum Ohr (Gerade a). Diese lässt sich durch den Kosinussatz errechnen. Die Formel dafür ist wie folgt:

$$a = \sqrt{b^2 + c^2 - 2 * b * c * cos(\alpha)} \tag{5}$$

Dabei ist der Abstand von Mittelpunkt des Kopfes bis zum Ohr die Gerade c. Diese wird für die Berechnung auf 10 cm festgelegt. Die Entfernung, der Geraden b, von Kopfmittelpunkt bis zum Lautsprecher ist 140 cm lang. Dieses ist auch die Entfernung aus den beiden zugrunde liegenden Messreihen. Da bei HRTF ein Azimuthwinkel von 0° eine frontale Schallquelle entspricht muss der Winkel α für das linke Ohr um -90° verschoben werden. Die Verschiebung für das rechte Ohr ist +90°. Die Entfernung zwischen linkem Ohr und einer sich nach links drehenden Schallquelle sieht wie folgt aus.

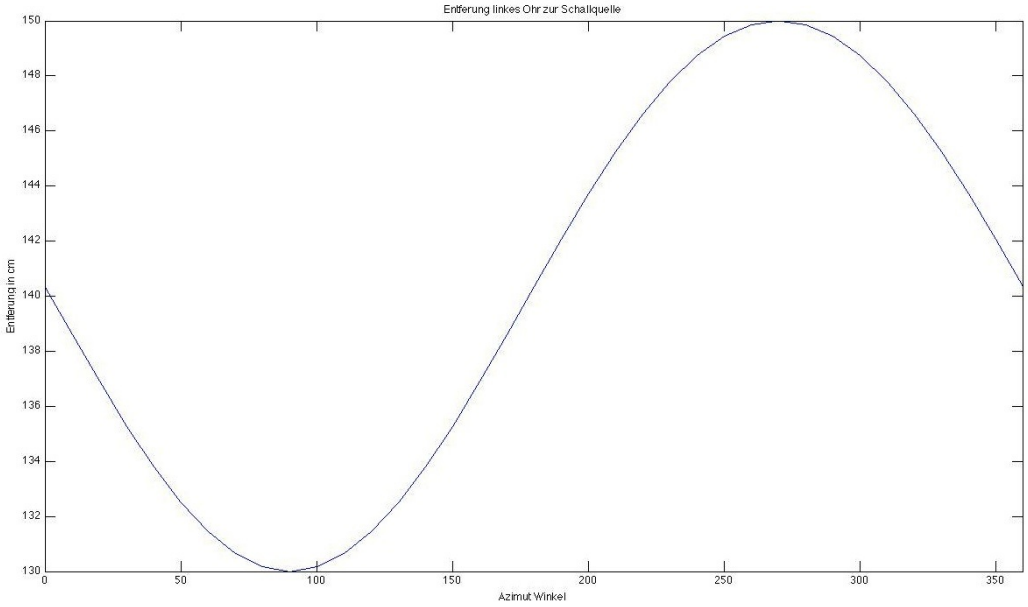

Abbildung 13: Entfernung linkes Ohr zu einer sich nach links drehenden Schallquelle

Diese Grafik 13 ist eine Annäherung an das zu erwartende Ergebnis, welches bei dem beiden zugrunde liegenden Messreihen für HRTF erwartet wird. Es wird erwartet, dass bei 0°, 180° und 360° die Laufzeiten für beide Ohren identisch sind. Zwischen 0° bis 180° sollte es nur minimale Abweichungen in der Entfernung (Laufzeit) geben, da der Schall ohne Störung das Ohr trifft. Bei einem Azimuthwinkel zwischen 180° und 360° wird davon ausgegangen, dass die gemessene Laufzeit nach oben abweicht. Bei der Berechnung wurde davon ausgegangen, dass der Schall durch den Kopf weitergeleitet wird. Dieses ist in der Realität nicht zu erwarten, weil der Schall um dem Kopf herumgeleitet wird, bevor er das Ohr erreicht. Dieser zusätzliche Weg sollte sich in den Messungen wiederspiegeln.

5.3.2 Berechnung der Laufzeitdifferenz aus der KEMAR Messreihe

Bei den Kemar Datensätzen kann nicht direkt auf die Laufzeitdifferenz zurückgegriffen werden. Aus diesem Grund wurden die Filterkoeffizienten, wie im Kapitel 5.1 berechnet. Mit der MATLAB-Funktion *grpdelay()* wurde die Gruppenlaufzeit der einzelnen Koeffizientensätze errechnet. Die nachfolgende Grafik zeigt den Frequenzgang und die Gruppenlaufzeit eines Kemar-Filter-Koeffizientensatzes.

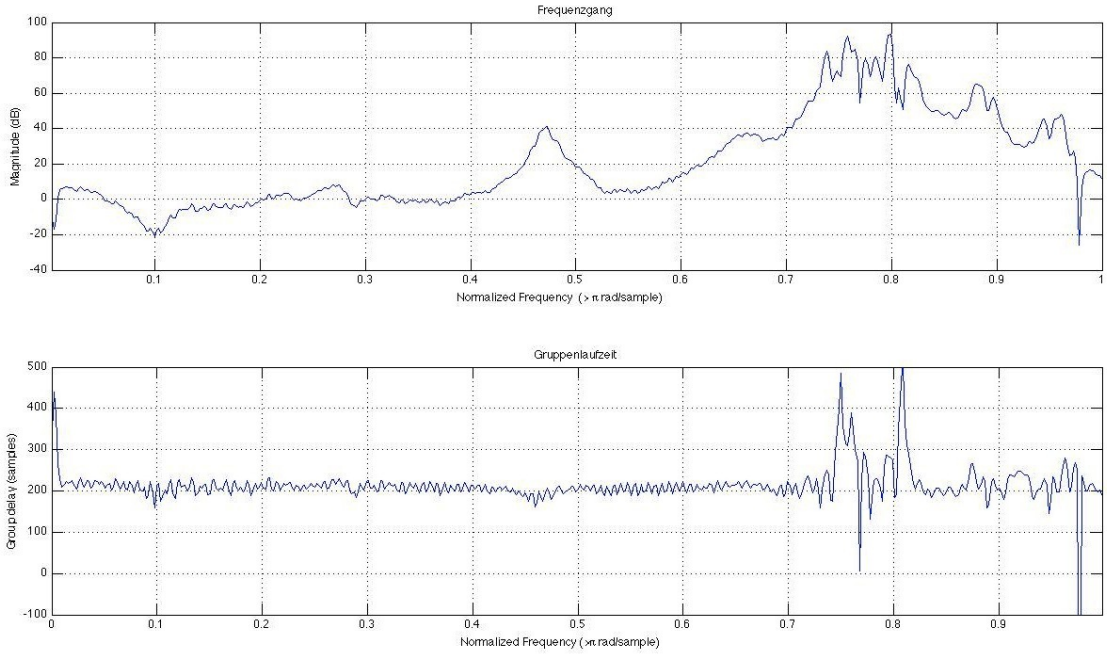

Abbildung 14: Frequenzgang und Gruppenlaufzeit im Vergleich bei einem Kemar-Filtersatzes

In Abbildung 14 ist zu sehen, dass die Gruppenlaufzeit nicht konstant, sondern von der Frequenz abhängig ist. Bei einer großen Änderung im Frequenzgang springt die Gruppenlaufzeit extrem. Um die Gruppenlaufzeit jenes Filterkoeffizientensatzes zu errechnen, wurde der Median der Gruppenlaufzeit gewählt. Der Median ist der mittlere Wert aus einer sortierten Wertereihe. Um die gleiche Laufzeitdifferenz der einzelnen Koeffizientensätze zu bekommen, wurden die Koeffizienten verschoben. Die Verschiebung hat keine große Veränderung auf die Filtereigenschaft, da die ersten Koeffizienten keine Auswirkung auf die Filterung haben. Durch die Verschiebung der Filterkoeffizienten wird die Größe des Filters verkleinert. Der Filter wir deswegen wieder auf eine Filterlänge von 512 angepasst, indem dieser mit Nullen aufgefüllt wird. Nach diesem Verfahren ist die Gruppenlaufzeit aller Filter bei einer fast gleichen Gruppenlaufzeit. Die maximale Abweichung beträgt $\pm$ 0.5 Samples. Der Verschiebungsfaktor, um den die Filterkoeffizienten verschoben wurden, entspricht der ITD in Samples und sieht wie folgt aus.

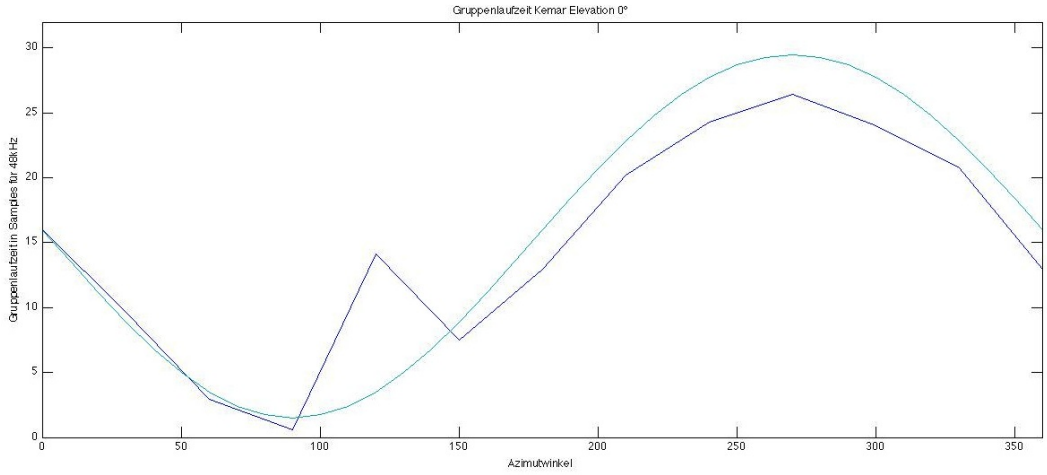

Abbildung 15: ITD der Kemar Messreihe im Bezug zur errechneten ITD

Die Grafik 15 zeigt die zu erwartete Laufzeitdifferenz im Vergleich zur errechneten Laufzeitdifferenz aus der Kemar-Messreihe. Dabei ist die Laufzeitdifferenz schon in Samples umgerechnet, bei einer Abtastfrequenz von 48 kHz. Bei einem Azimuthwinkel von 120° weist die Kemar Laufzeitdifferenz eine gewaltige Abweichung auf. Im Vergleich zu den benachbarten HRTF-Filter ist dieser Filter deutlich lauter, was vermuten lässt, dass es sich um einen Fehler in der Messreihe handelt. Verwunderlich ist des Weiteren, dass die Gruppenlaufzeit ab 180° nicht größer als die errechnete Laufzeitdifferenz ist. Bei der Berechnung der zu erwartenden Laufzeitdifferenz wurde davon ausgegangen, dass der Schall durch den Kopf geleitet wird. Dieses ist in der Wirklichkeit nicht gegeben, sondern der Schall wird um den Kopf herumgeleitet. Aus diesem Grund wurde erwartet, dass die gemessene Laufzeitdifferenz größer als die errechnete Laufzeitdifferenz ist.

5.3.3 Berechnung der Laufzeitdifferenz aus der SIMA Messreihe

Bei der Messreihe Sima wurde der Phasengang und die Pegeldifferenz separat gemessen. Die Filterkoeffizienten für die Pegeldifferenz (ILD) werden normal berechnet. Dabei gibt es nur einen Unterschied: In der Berechnung wird jeder Filtersatz mit einer konstanten Verzögerung berechnet (***fpr_base***).

```
b_r = real(ifft(far_r .* exp(j * fpr_base)));
```

Diese Änderung hat zufolge, dass alle HRTF-Koeffizientensätze die gleiche Gruppenlauf-

zeit haben. Dieses ist für die interaurale Pegeldifferenz (ILD) gewünscht. Um die interaurale Laufzeitdifferenz (ITD) zu errechnen, wurden diese Koeffizientensätze mit den Koeffizientensätzen aus der normalen Berechnung in Bezug auf die Gruppenlaufzeit verglichen. Die daraus errechneten Gruppenlaufzeiten werden in der folgenden Grafik mit der zu erwartenden Gruppenlaufzeit dargestellt.

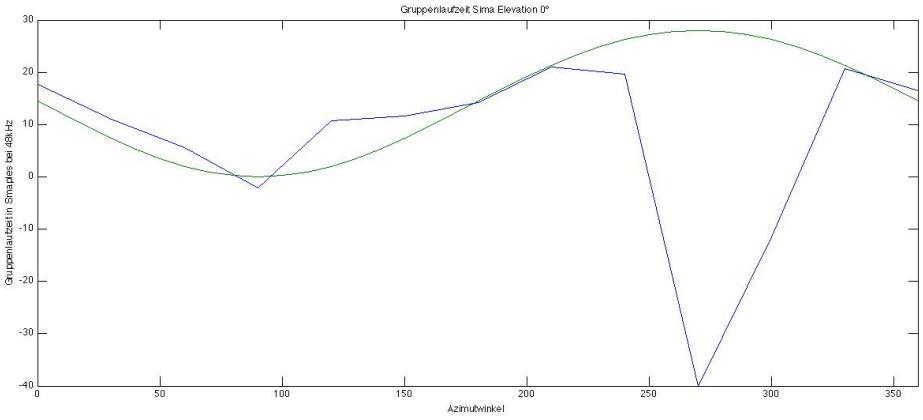

Abbildung 16: ITD der Sima Messreihe im Bezug zur errechneten ITD

Auch in dieser Grafik ist zu sehen, dass es bei den Messungen im Bereich von 240° bis 300° mehrere Fehlmessungen gegeben hat.

Für das spätere System wurde daher entschlossen eine einfache Möglichkeit des Austausches der Filterkoeffizienten für den ITD-Filter zu implementieren. Durch Hörtests wird im System getestet mit welchem Verfahren sich ein gutes Hörergebnis erzielen lässt.

5.4 Normierung der HRTF-Filterkoeffizienten für den Einsatz in der SoC-Plattform

Es gibt mehrere Möglichkeiten die Normierung von Filterkoeffizienten anzugehen. Zwei Methoden wurden in Kapitel 2.3.1 vorgestellt.

Mit dem MATLAB Befehl *norm(coeff,1)* lässt sich die L_1-Norm aus den Koeffizienten ***coeff*** errechnen. Da alle HRTF-Filterkoeffizienten einer Messreihe zusammen betrachtet werden müssen, sind alle Filterkoeffizienten durch den gleichen Normierungsfaktor zu teilen. Der Normierungsfaktor ist dabei das Maximum aller L_1-Normierungsfaktoren aus der HRTF-Messreihe.

Für die L_2-Norm stellt MATLAB den Befehl *filternorm(coeff,2)* zur Verfügung. Dabei ist das gleiche Vorgehen mit der Wahl des Normierungsfaktors wie bei der L_1 Normierung zu beachten. Die Ergebnisse dieser beiden Normierungen werden in der folgenden Tabelle 1 dargestellt.

	L_1-Norm	L_2-Norm
SIMA	25.8202	3.4944
SIMA mit konstanter Laufzeitdifferenz	14.8388	3.5545
Kemar	753.6478	45.9072
Kemar mit konstanter Laufzeitdifferenz	753.6478	45.9072

Tabelle 1: L_1 und L_2 - Normierungsfaktor der HRTFs

Tests dieser beiden Normierungsverfahren haben kein gutes Hörergebnis vermittelt. Im Fall der L_1-Norm war trotz lautem Eingangssignal das gefilterte Ausgangssignal kaum noch hörbar. Bei der Normierung mit dem L_2-Normfaktor gab es Übersteuerungs-Effekte. Aus diesem Grund wurde ein anderes Normierungsverfahren angewandt. Dabei wurden verschiedene Testsignale wie Pop Musik und Sprache ausgesucht. Diese Testsignale wurden mittels MATLAB auf den vollen Wertebereich von -1 bis +1 normiert. Diese so erzeugten Testsignale wurden mittels MATLAB FIR-Filter mit allen errechneten HRTF-Filterkoeffizienten gefiltert. Als Normierungsfaktor wurde das größte absolute Ausgangssample gewählt. In der Tabelle 2 werden die Normierungsfaktoren dieses Verfahrens dargestellt.

	Pop	Klassic	Sprache1	Sprache2	Pink-Noise	White-Noise
SIMA	5.0857	1.7485	6.1116	3.1418	3.1418	2.1908
SIMA g	4.5526	1.7367	6.2636	3.3583	2.8817	2.5918
Kemar	22.0082	1.9039	18.5253	3.2724	18.8902	30.6716
Kemar g	22.0082	1.9039	18.5253	3.2724	18.8902	30.6716

Tabelle 2: Normierungsfaktoren der HRTFs

Die Tabelle 2 zeigt sehr gut, dass die verschiedenen HRTF-Filter mit unterschiedlichen Testsignalen andere Normierungsfaktoren liefern. In der Praxis hat sich gezeigt, dass eine Normierung mit dem Pop-Musik-Normierungsfaktor die beste Lösung ist. Daraufhin wurden alle Koeffizientensätze mit dem entsprechenden Normierungsfaktor berechnet.

6 Entwicklung der SoC-Plattform

In diesem Kapitel wird der Aufbau der zu entwickelnde SoC-Plattform erläutert. Dabei zeigt die Abbildung 17 die zu entwickelnde SoC-Plattform.

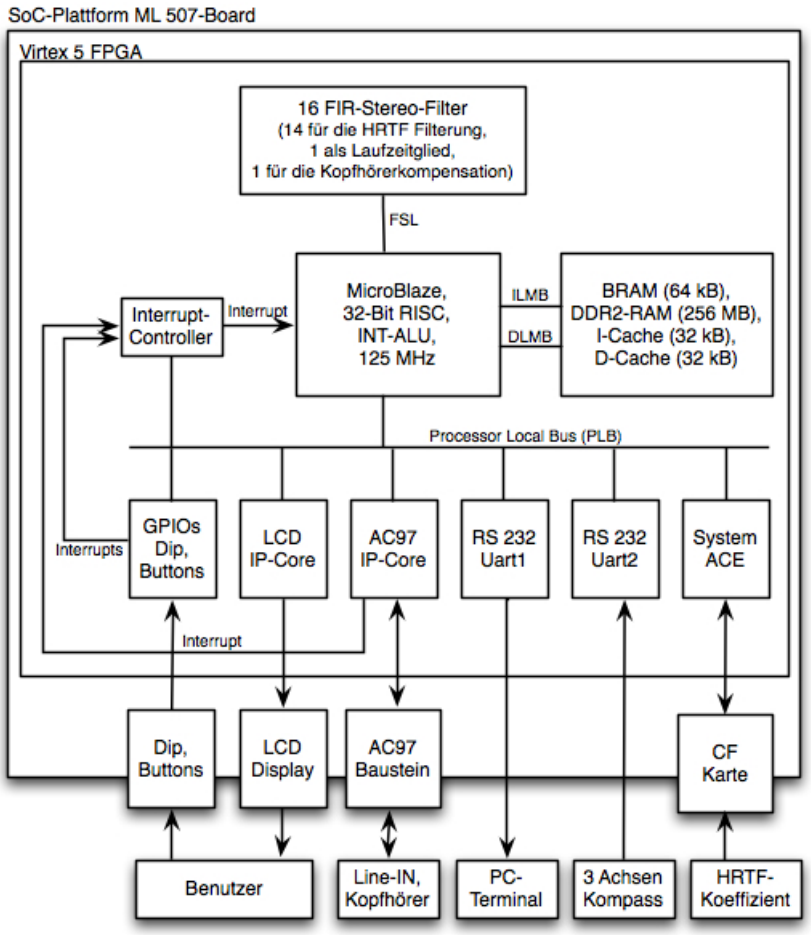

Abbildung 17: SoC-Architektur im Virtex5 FPGA mit MicroBlaze Softcore-Prozessor und allen Komponenten

In dem folgenden Unterkapitel 6.1 werden zuerst die für das System notwendigen IP-Cores beschrieben. In Kapitel 6.2 wird die Entwicklung des MicroBlaze mithilfe des Xilinx XPS-Entwicklungsumgebung beschrieben. In Kapitel 6.3 werden die Änderungen am MicroBlaze und das Hinzufügen der notwendigen Komponenten beschrieben.

6.1 IP cores

Ein IP-Core ist ein FPGA-Modul oder auch Beschleuniger-Modul genannt, welches direkt in den MicroBlaze integriert werden kann. Die Xilinx XPS-Entwicklungsumgebung bietet dafür den “Create or Import Peripheral“-Assistenten. Dieser Assistent bietet eine Vielzahl von Einstellungsmöglichkeiten. Im Assistenten lässt sich zuerst einstellen, über welches Bus-Interface der spätere IP-Core angesteuert werden soll. Dazu stehen in der Version 12.4 des XPS drei verschiedene Businterfaces zur Auswahl. Der Processor Local Bus kurz PLB, die Fast Simplex Link Schnittstelle kurz FSL sowie die AXI-Bus Schnittstelle. Des Weiteren können Einstellungen bezüglich des IP-Interfaces wie zum Beispiel S/W-Register angeben werden. Eine Vielzahl von weiteren Einstellungen sind über diesen Assistenten möglich. Nach dem Durchlaufen dieses Assistenten erzeugt dieser einen IP-Core mit den gewünschten Einstellungen in welchen dann das FPGA-Beschleuniger-Modul eingebunden werden kann. Die IP-Cores bestehen aus mehreren wichtigen Dateien, die in einer Ordnerstruktur abgelegt sind. Im Unterordner data gibt es die mpd-Datei, in der die Schnittstellen des IP-Cores beschrieben sind. Des Weiteren befindet sich in diesem Ordner noch die pao-Datei, welche geändert wird, wenn ein IP-Block aus mehreren Quellcodedateien besteht. Im Unterordner hdl befinden sich die vom Assistent erzeugten Quellcodedateien.

6.1.1 Entwicklung des LCD IP-Cores

Um das auf dem Board zur Verfügung stehende “Tianma LCD Display Module“ [Mic09] in das System einzubinden, wird für dieses LCD Display ein neuer IP-Core erzeugt. Dieser lässt sich wie schon beschrieben über den “Create or Import Peripheral“-Assistenten der XPS Entwicklungsumgebung erzeugen. Der LCD-IP-Core wird über den Processor Local Bus angesteuert und mit einem User Logic Software Registern ausgestattet. Sonst sind für den LCD-IP-Core keine weiteren Einstellungen im Assistenten zu treffen. Nach dem Erzeugen des IP-Cores befindet sich seine Ordnerstruktur im Unterverzeichnis “pcores/ip-name“ des Projektordners. Nach dem Erzeugen des LCD-IP-Core muss das LCD Interface hinzugefügt werden. Das LCD wird über 7 Bit angesteuert. Drei dieser Bit sind Steuerbit. Das E-Bit, welches den Start eines neuen Lese-, bzw. Schreib-Befehls kennzeichnet. Das RW-Bit gibt an, ob gelesen oder geschrieben wird, sowie ein RS-Bit, das ein Register auswählt. Die restlichen 4 Bit sind Datenbits.

Diese 7 Bit werden dazu zuerst in der Microprozessor Peripheral Definition (MPD) Datei hinzugefügt. Dazu wird die Datei um einen neuen Ausgangsport erweitert.

```
## Ports
PORT lcd = "", DIR = O, VEC = [0:6]
```

Dieser Port wird auch in der lcd.vhd Datei hinzugefügt und in der Port Map eingetragen.

```
--USER ports added here
   lcd                          : out std_logic_vector(0 to 6);
   ...
```

```
port map(
--USER ports mapped here
   lcd                              => lcd,
```

In der Top-Entity (user_logic.vhd) wird dieser Port ebenfalls bekannt gemacht und als Signal hinzugefügt. Dazu wird diese Datei ebenfalls erweitert2.

```
--USER ports added here
   lcd                            : out std_logic_vector(0 to 6);
--USER signal declarations added here, as needed for user logic
   signal lcd_i                           :std_logic_vector(0 to 6);
```

Somit sind in allen Datein die 7 Bit des LCD Display bekannt. Die Daten für die LCD-Pins werden über den PLB-Bus an den IP-Core geschickt. Dieser leitet diese an die entsprechenden Pins weiter. Um diesen Ablauf zu gewährleisten, wird der PLB-Interface-Process des IP-Cores angepasst.

```
  lcd_PROC : process(Bus2IP_Clk) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk='1' then
      if Bus2IP_Reset='1' then
        lcd_i<=(others=>'0');
      else
        if Bus2IP_WrCE(0)='1' then
          lcd_i<=Bus2IP_Data(25 to 31);
        end if;
      end if;
    end if;
  end process lcd_PROC;
  lcd<=lcd_i;
```

Somit ist der LCD-IP-Core fertiggestellt und wird im Verlauf der Entwicklung dieser SoC-Plattform in den MicroBlaze hinzugefügt. Dieses wird im Kapitel 6.3.4 erklärt.

6.1.2 Aufbau des AC97-Codec-IP-Core

Auf dem Virtex5 ML-507 befindet sich ein AC97 Codec Baustein [DEV05]. Dieser AC97 Codec Baustein ist in der Lage analoge Stereo-Audiosignale mit einer maximalen Abtastrate von 48 kHz abzutasten. Die Wiedergabe von digitalen Audiodaten kann über analoge Ausgänge ebenfalls mit bis zu 48 kHz erfolgen. Die Eingangs und Ausgangs Sample-Rate können voneinander unterschiedlich sein und zwischen 7040 Hz und maximal 48 kHz mit einer Schrittweite von 1 Hz liegen. Die AD-Wandler und DA-Wandler lassen sich mit 16 Bit oder 20 Bit Genauigkeit betreiben.

Der AC97-IP-Core aus der Masterarbeit von Jan Kuhr kann nicht mehr verwendet werden, da dieser über die nicht mehr vorhandene OPD-Schnittstelle angeschlossen wurde. Aus diesem Grund wurde nach Alternativen gesucht. In einem Xilinx-Übungs-Lab für “Partial Reconfiguration Flow“ wurde in Lab 6 eine Alternative gefunden [Xil11a]. Der in der

Übung verwendete IP-Core für den AC97 Codec ist für den gleichen AC97 Code Baustein des Virtex5 ML507 geschrieben worden. Er wird über die PLB-Schnittstelle angeschlossen und lässt sich in das neue System ohne weiteren Aufwand einbinden. Da für diesen IP-Core keine Dokumentation zur Verfügung steht, wurde dieser IP-Core einer gründlichen Kontrolle unterzogen.
Der Aufbau wird in der folgenden Abbildung 18 dargestellt.

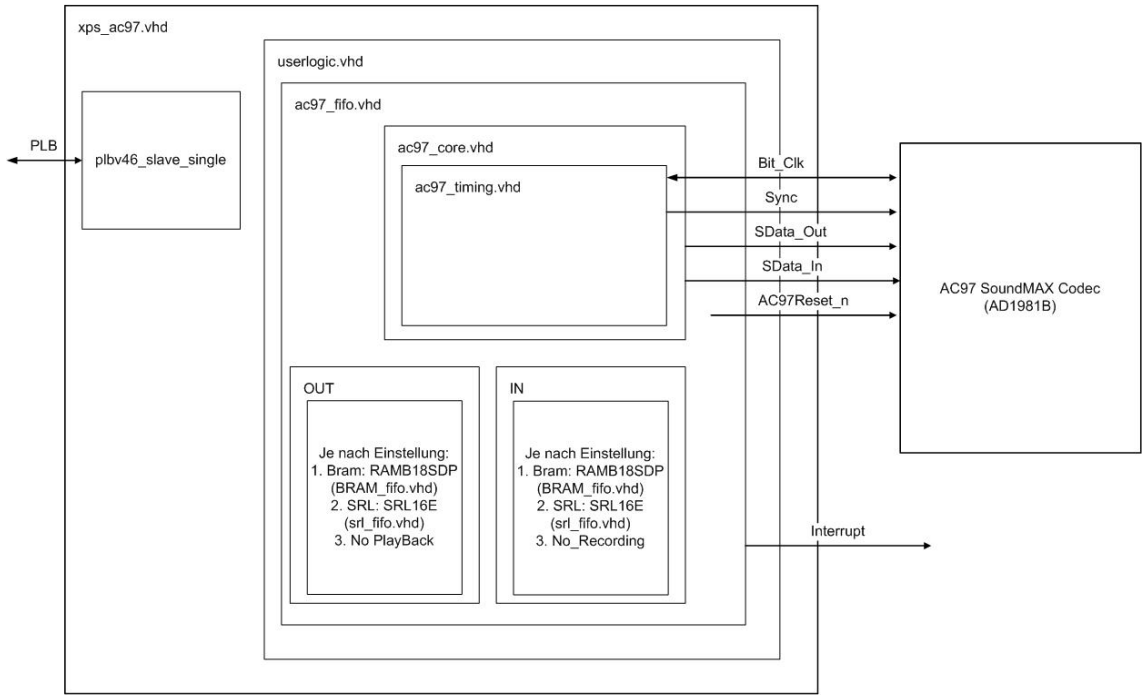

Abbildung 18: Aufbau des AC97-IP-Cores

Die Top-Entity des AC97-IP-Cores wird in der “xps_ac97.vhd“ beschrieben. Sie instanziiert auf der einen Seite die Schnittstelle zum MicroBlaze über den Processor Local Bus (PLB), sowie auf der anderen Seite für die Ansteuerung des AC97 Bausteins über die “userlogic.vhd“. Sie verbindet den PLB Bus mit dem FPGA-Modul zur Steuerung des AC97 Bausteins. Des Weiteren sind in ihr die Pins zur Steuerung des Bausteins rausgeführt.

Die Datei “userlogic.vhd“ instanziiert nur die “ac97_fifo.vhd“. In der Datei “ac97_fifo.vhd“ wird die Grundlogik des IP-Cores beschrieben. Durch die Bits 27 bis 29 des ***BUS2IP_Addr*** aus dem PLB-Interface wird der IP-Core gesteuert. Die nachfolgende Tabelle 3 zeigt die Bedeutung der 3 Bit bei Lese- und Schreib-Zugriffen.

BUS2IP_Addr2(27 to 29)	Read	Write
0x0	OUT_FIFO	IN_FIFO
0x4	STATUS	CONTROL
0x8	AC97_READ	AC97_WRITE
0xC	/	AC97_CNTRL

Tabelle 3: Registeradressen des AC97-IP-Cores bei Schreib- und Lesezugriffen

Beim einem Lesebefehl mit der Adresse 0x0 wird das nächste Stereo-Sample aus den Record FIFO über die PLB-Schnittstelle an den MicroBlaze versendet. Mit der Adresse 0x0 und einem Write-Zugriff wird ein Stereo-Sample in das PlayBack FIFO geschrieben. Das Status-Register, welches bei einem Lesezugriff mit der IP-Adresse 0x4 über den PLB-Bus verschickt wird, ist wie folgt aufgebaut.

Bit	Name	Beschreibung
0	in_FIFO_full	gesetzt wenn PlayBack-FIFO voll ist
1	in_FIFO_empty	gesetzt wenn PlayBack-FIFO leer ist
2	out_FIFO_empty	gesetzt wenn Record-FIFO leer ist
3	out_Data_exists	gesetzt wenn Record-FIFO Daten enthält
4	regiser_access_bussy	gesetzt wenn der AC97-Codec-Baustein bussy ist
5	IP_CLK_codes_rdy	gesetzt wenn der AC97-Codec-Baustein ready ist
6	in_FIFO_underrun	gesetzt wenn es ein Unterlauf im PlayBack-FIFO geben hat
7	out_FIFO_overrun	gesetzt wenn es ein Überlauf im Record-FIFO gegeben hat
8	IP_CLK_ac97_reg_error	gesetzt wenn der AC97-Codec-Baustein ein Fehler hat
9	in_FIFO_interrupt_en	gesetzt wenn Interrupt auf PlayBack-FIFO freigegeben ist
10	out_FIFO_interrupt_en	gesetzt wenn Interrupt auf Record-FIFO freigegeben ist
12-21	in_FIFO_level	Füllstand des PlayBack-FIFOs
22-31	out_FIFO_level	Füllstand des Record-FIFOs

Tabelle 4: Status-Register des AC97-IP-Cores

Mit der Adresse 0x4 und einem Schreibzugriff auf dem AC97-IP-Core wird das Control-Register beschrieben. Über dieses Register wird der IP-Core gesteuert. Das Control-Register ist wie folgt aufgebaut. (Tabelle 5)

2

Bit	Name	Beschreibung
31	clear_in_FIFO	löschen des PlayBack FIFO
30	clear_out_FIFO	löschen des Record FIFO
29	in_FIFO_interrupt_en	PlayBack FIFO mit einem Interrupt versehen
28	out_FIFO_interrupt_en	Record FIFO mit einem Interrupt versehen
27	ac97_reset_i	Reset des AC97-Baustein

Tabelle 5: Control-Register des AC97-IP-Cores

Über die IP-Core Defines ***C_PlayBack*** und ***C_Record*** wird die Aufnahme beziehungsweise die Ausgabe des IP-Cores freigeschaltet. Durch Setzen dieser Defines auf 1 werden diese aktiv geschaltet. Durch das ***C_USE_BRAM*** Define wird ausgewählt welche Art von Zwischenspeicher verwendet wird. Wenn ***C_USE_BRAM*** = 1 dann wird der AC97-IP-Core mit einem Block-Ram als Zwischenspeicher instanziiert. Ist ***C_USE_BRAM*** = 0 ist der Zwischenspeicher für die Samples ein Schieberegister.

Die oben genannten Defines lassen sich später beim Hinzufügen des IP-Cores in die MicroBlaze einstellen.

Des Weiteren ist der AC97-IP-Core mit einem Interrupt-Controller zum Erzeugen von Interrupts ausgestattet. Dieser befindet sich ebenfalls in der “ac97_fifo.vhd“ Datei. Die Tabelle 6 zeigt, auf welche Ereignisse ein Interrupt ausgelöst werden kann. Durch das Setzten des ***C_INTR_LEVEL*** Defines wird der Interrupt-Modus ausgewählt. Über das Control-Register werden die Signale ***in_fifo_interrupt_en*** und ***in_fifo_interrupt_en*** gesetzt, die dann den Interrupt-Controller freischalten.

C_INTR_LEVEL	***in_fifo_interrupt_en*** = 1	***in_fifo_interrupt_en*** = 1
0	Kein Interrupt	Kein Interrupt
1	Interrupt wenn PlayBack-FIFO halb voll ist	Interrupt wenn Record-FIFO halb voll ist
2	Interrupt wenn PlayBack-FIFO voll ist	Interrupt wenn Record-FIFO leer ist

Tabelle 6: Interrupt-Modus des AC97-IP-Cores

6.1.3 Anpassung der FSL-Schnittstelle für den FIR-Stereo-Filter

Seit der Arbeit von Jan Kuhr haben sich die Versionen des MicroBlaze und der FSL-Schnittstelle verändert. Bei der Einbindung des FSL-FIR-Stereo-Filter-IP-Cores sind Fehler bei der Kommunikation über die FSL-Schnittstelle aufgetreten. Aus diesem Grund wurde entschlossen, die FSL-Schnittstelle für den FIR-Stereo-Filter auf den neuesten FSL-Stand zu bringen.

Der unidirektionale Fast Simplex Link ist eine direkte Verbindung zwischen MicroBlazes oder zwischen MicroBlaze und schnellem FPGA-Beschleuniger. Er arbeitet nach einem FIFO-basierten Master-Slave Prinzip, wobei die Master-Schnittstelle die Daten in den FIFO schreibt und die Slave-Schnittstellen diese Daten aus dem FIFO liest. Durch die Unidirektionalität besteht eine FSL-Schnittstelle immer aus zwei FSL-Interfaces, um die wechselseitige Kommunikation zu gewährleisten. Jedes Paar besteht auf beiden Seiten aus einem Master sowie einem Slave. In der aktuellen Version des MicroBlaze stehen 16 solcher "Stream Link Interface Paare" mit je einer Datenbusbreite von 32 Bit zur Verfügung.

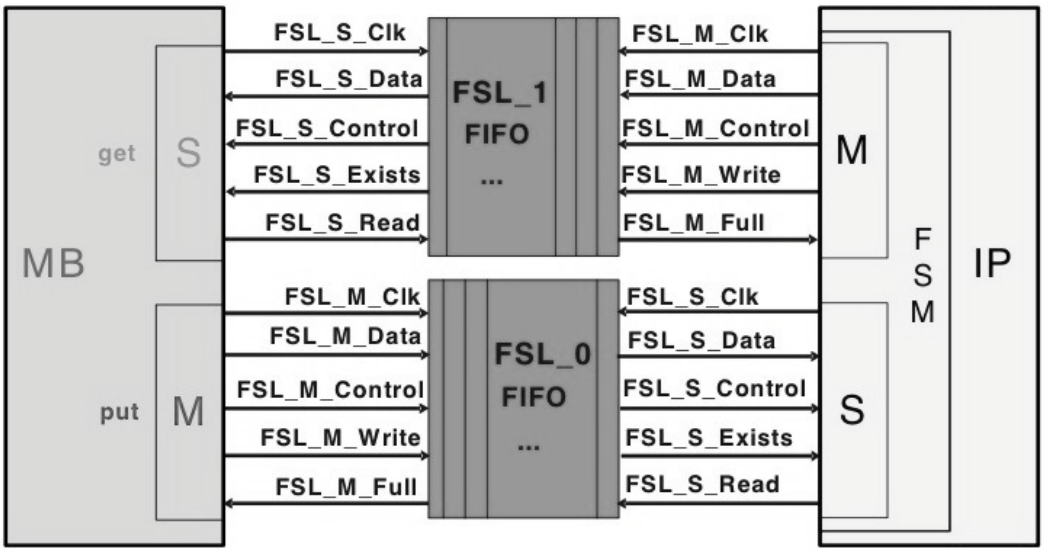

Abbildung 19: FSL-Schnittstelle zwischen MicroBlaze und IP-Core [Sch11]

Die wichtigen Signale der FSL-Schnittstelle sind in der Tabelle 7 dargestellt.

Signal Name	Beschreibung
FSL_M_Data	Daten Input richtung FIFO (schreiben)
FSL_M_Control	Ein 1 Bit breites Steuersignal, das zu jeden FSL_M_Data-Signal mit abgespeichert wird
FSL_M_Write	Steuersignal für das FIFO das ein neues Datum am Eingang anliegt
FSL_M_Write	Ausgangsbit das bei vollen FIFO auf "1" springt
FSL_S_Data	Daten Output richtung IP oder MicroBlaze (lesen)
FSL_S_Control	Das mit dem Datum abgespeicherte Control-Bit
FSL_S_Read	Steuersignal für das FIFO das ein Datum aus dem FIFO gelesen werden möchte
FSL_S_Exists	Ausgangsbit das bei vorhanden Daten im FIFO "1" zeigt

Tabelle 7: FSL-Signale und dessen Bedeutung

Mittels des "Create or Import Peripheral"-Assistenten wurde ein neuer IP-Core mit Fast Simplex Link (FSL) Interface erzeugt. Der Name dieses IP-Cores lautet "fsl_fir_stereo_v1_00_a". Die einzige Einstellung ist, dass die FSL-Schnittstelle der Version 2.11.c ein Input und Output FSL Interface bekommt. Die Anzahl von Input und Output Words ist dabei 1. Diese Angabe erstellt eine FSM, die immer nur ein Datum von dem FSL-FIFO liest. Nach dem Durchlaufen des Assistenten ist das FSL Interface im Projektordner unter dem Ordner pcores zu finden.

Als nächsten Schritt wird die Datei "FIR_BRAM_STEREO.vhd" aus dem alten IP-Core in den neuen IP-Core-Ordner hdl/vhdl hinzugefügt. Diese Datei enthält den in der Masterarbeit von Jan Kuhr entwickelten FIR-Stereo Filter, der auch in diesem IP-Core verwendet wird. Dazu wird die durch den Assistenten erzeugte Topentity angepasst. Diese befindet sich im Pcores Unterordner hdl/vhdl unter dem Namen "fsl_fir_stereo.vhd". Dazu wird der FIR-BRAM-STEREO-Filter als Komponente hinzugefügt.

```
component  FIR_BRAM_STEREO is
generic(
  C_ORDER: positive := 511); -- Filterordnung
port
 (CLK: in std_logic;                 -- Taktsignal
  RESET: in std_logic;               -- asynchron, active-high
  REQ: in std_logic;                 -- Starte Berechnung
  INPUT: in std_logic_vector(31 downto 0);  -- 32 Bit Eingangssignal
  NEW_COEFF: in std_logic;           -- neuer koeffizient
  CONTROL: in std_logic;             -- CONTROL=1 laedtKoeffizienten
  OUTPUT: out std_logic_vector(31 downto 0); -- 32 Bit Ausgangssignal
  ACK: out std_logic);               -- Signalisiere Datenuebergabe
end component;
```

Nachdem die Komponente hinzugefügt wurde, wird diese mit der FSL-Schnittstelle verbunden. Dazu befindet sich in der Top-Entity eine Finite-State-Maschine. Diese wird umgeschrieben um die Steuerung zwischen FSL und FIR-Stereo-Filter zu übernehmen. Die Abbildung 20 zeigt den dafür entwickelten Automaten.

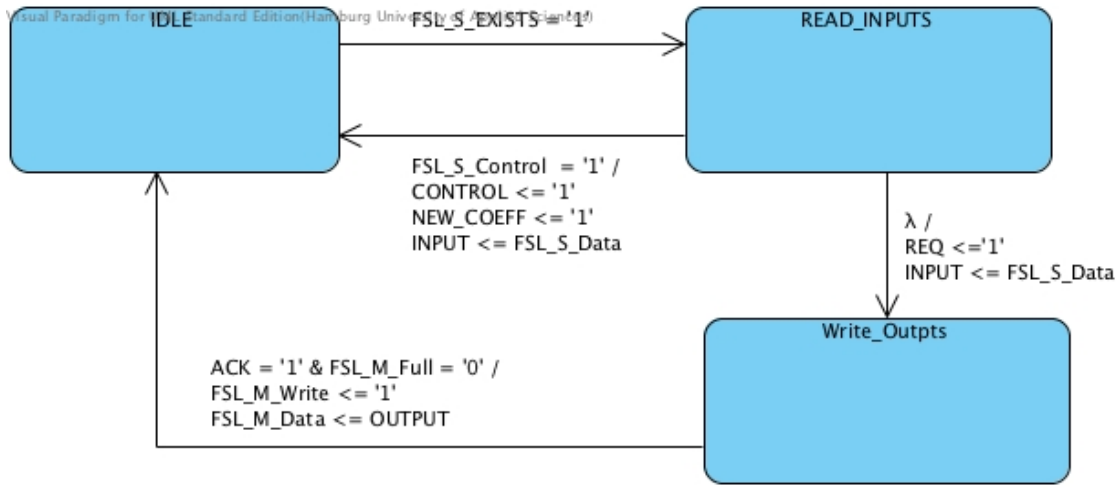

Abbildung 20: Finite-State-Maschine des FSL-FIR-Stereo-Filters

Der Startzustand ist der **Idle**-Zustand. Solange kein Datum im FIFO der FSL Slave Schnittstelle anliegt, bleibt dieser Zustand aktuell und wartet auf das Ereignis ***FSL_S_EXITS*** = "1". Bei diesem Ereignis wechselt der Automat in den Zustand **READ_INPUTS**. Dieser Zustand lässt sich über 2 Ereignisse verlassen. Das erste Ereignis ist gegeben, wenn das Bit ***FSL_S_Control*** = "1" gesetzt ist. Dieses Signal wird genutzt um anzugeben, ob das Datum, was aktuell aus dem FIFO genommen wird, ein Sample oder ein Filter-Koeffizient ist. Ein 1-Signal dieses Bits gibt an, dass das Datum als Koeffizient für den FIR-Filter zu verstehen ist. Um den FIR-BRAM-Stereo-Filter dieses mitzuteilen müssen die Bits ***CONTROL*** und ***NEW_COEFF*** auf 1 gesetzt werden. Das Signal ***FSL_S_Data*** wird auf das Input Signal des Filters gelegt. Nach dieser Transition ist der Zustand wieder Idle und es wird auf ein neues Datum aus dem FIFO gewartet. Die zweite Transition aus dem **READ_INPUT** Zustand ist, dass das aktuelle Datum als ein zu filterndes Audio-Sample zu verstehen ist. Dazu wird am Filter das Bit ***REQ*** = "1" gesetzt und das Filter-Input Signal braucht die Information des ***FSL_S_Data*** Signals. Daraufhin startet der Filter die Filterung. Der Zustand ändert sich daraufhin in den Zustand **Write_Outputs**. In diesem Zustand wird gewartet, bis der Filter mit der Filterung fertig ist. Dieses wird über das Signal ***ACK*** des Filter gekennzeichnet. Zur Sicherheit, dass der FIFO Richtung MicroBlaze nicht überfüllt ist, muss noch die Abfrage ***FSL_M_Full*** = "0" erfüllt sein. Wenn beide Bedingungen eintreffen, wird das gefilterte Ergebnis über die Schnittstelle (***FSL_M_Data***) an den MicroBlaze verschickt. Mit dem Signal ***FSL_M_Write*** = "1" wird dem FIFO mitgeteilt, dass ein neues Datum aufgenommen

werden soll. Der Zustand nach dieser Transaktion ist der **Idle** Modus.

Daraufhin wird die MPD-Datei und POA-Datei des IP-Cores angepasst.
Bei der MPD-Datei wird ein Parameter hinzugefügt. Dieser Parameter gibt an, welcher Ordnung der FIR-Filter ist und lässt sich später über das Xilinx Platform Studio individuell anpassen.

```
## Generics for VHDL or Parameters for Verilog
PARAMETER C_ORDER = 511, DT = INTEGER, RANGE = (1:511)
```

In der POA-Datei wird die "FIR_BRAM_STEREO.vhd" Datei eingetragen. Dabei ist zu beachten, dass diese vor der Top-Entity aufgelistet wird.

```
lib fsl_fir_stereo_v1_00_a FIR_BRAM_STEREO vhdl
lib fsl_fir_stereo_v1_00_a fsl_fir_stereo vhdl
```

Somit ist der neue FSL-FIR-Stereo IP-Core fertig und wird getestet. Da die Filtereinheit schon in der vorangegangenen Arbeit mit Hilfe von ModelSim getestet wurde, wird auf diesen Test verzichtet. Um die FSL Schnittstelle zu testen wurde ein FIR-Stereo-Filter mit einem Koeffizientensatz geladen. Daraufhin wurden die Impulsantwort und Sprungantwort ausgewertet und mit dem in MATLAB errechneten Ergebnissen verglichen.

Auf der linken Seite der Abbildung 21 ist die Impulsantwort zu sehen, die rechte Seite zeigt die Sprungantwort. Dabei ist die oberste Zeile die in MATLAB errechnete Impuls- oder Sprungantwort. Darunter ist das Ergebnis der SoC-Plattform dargestellt. Die letzte Zeile zeigt den Unterschied zwischen MATLAB und der SoC-Plattform. Dabei liegt der Fehler zwischen beiden Ergebnisse bei maximal einem LSB. Das LSB ist in unserm Fall 2^{-15}. Dieser Fehler lässt sich durch unterschiedliche Quantisierung erklären.

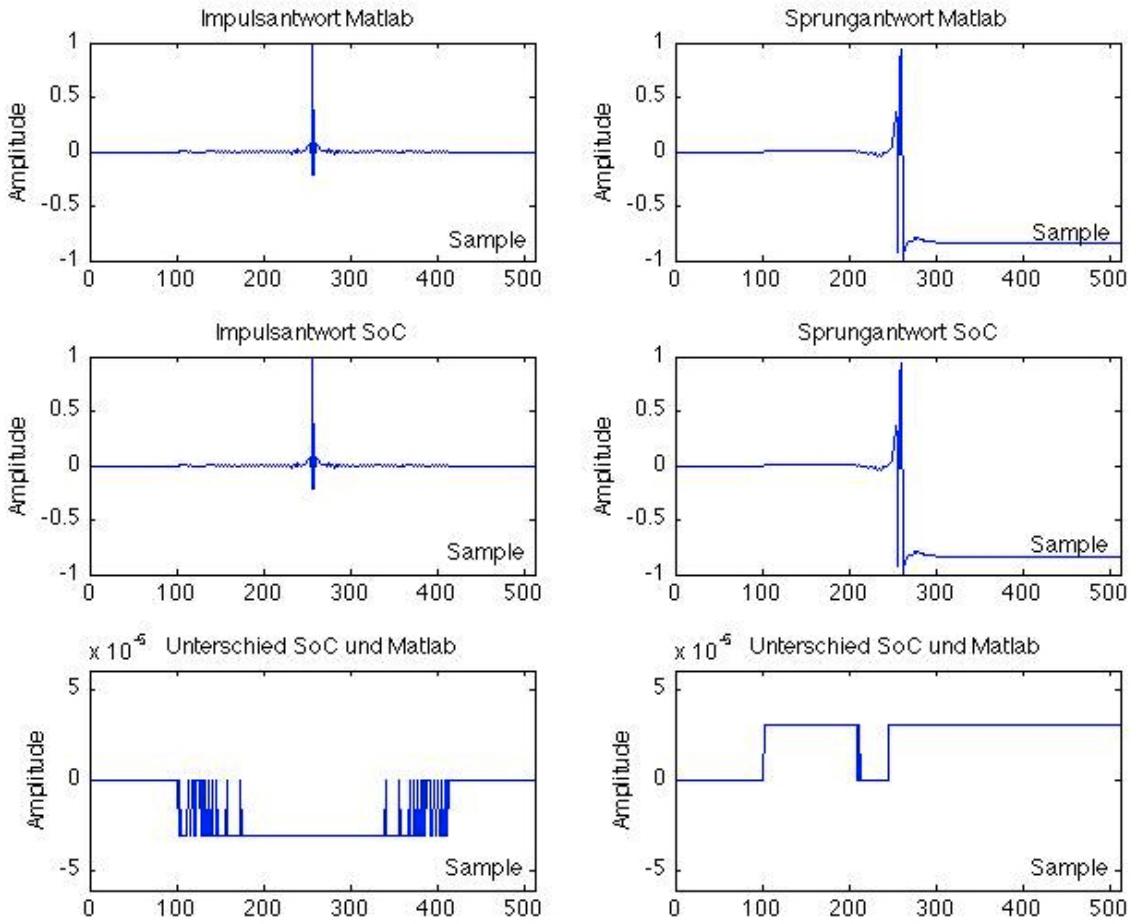

Abbildung 21: Vergleich FSL-FIR-Filter und MATLAB: Impuls- und Sprungantworten eine FIR-Filters

6.2 Konfiguration von MicroBlaze und Standardschnittstellen

Mit dem “Base System Builder wizard“-Assistenten unter XPS lassen sich die Grundeinstellungen des MicroBlaze vornehmen. In der Version 12.4 des Xilinx Platform Studio lassen sich zwei Arten von Busschnittstellen auswählen. Die Processor Local Bus (PLB) und die Advanced eXtensible Interface (AXI). Für das jetzige System wird der MicroBlaze über die PLB Schnittstelle mit den restlichen Hardware-Komponenten verbunden. Der MicroBlaze wird als Single Prozessor System eingerichtet und mit dem maximalen Tack von 125 MHz getaktet. Des Weiteren wird der Interne BRAM des MicroBlaze auf 64 KB eingestellt. Dieser interne BRAM ist direkt im MicroBlaze eingebunden. Der große Vorteil ist das dieser BRAM ein sehr schneller Dual-Port-RAM ist. Er wird in diesem System eher für Testzwecke benutzt, da davon auszugehen ist, dass die große Anzahl von Filterkoeffizienten deutlich mehr Speicherplatz braucht und diese somit in einen externen RAM

ausgelagert werden müssen. Im weiteren Verlauf der Konfiguration des MicroBlaze werden noch folgende Schnittstellen und Hardware-Module hinzugefügt.

- RS232 UART1 für den Anschluss an den Entwicklungscomputers
- RS232 UART2 für den Anschluss des Positionssensors (Kompass)
- DDR2 SDRam
- 8Bit LEDs zur Statusanzeige
- 5Bit Push-Buttons
- 8Bit Dip-Switches zur Statuseinstellung
- SystemACE Controller zum Zugriff auf die CF-Speicherkarte

Die letzte wichtige Einstellung durch den “Base System Builder wizard“-Assistenten ist die Einstellung der Caches. Der Daten- und Instruktion-Cache wird auf jeweils 32 KB gestellt. Die beiden Caches bekommen ein eigenen BRAM-Block. Dabei ist nur zu beachten, dass die beiden Caches über Cacheleitungen mit dem DDR2-RAM verbunden werden.

6.3 Anbindung der Hardware-Module

Dieses Kapitel erläutert wie der MicroBlaze um die weiteren IP-Cores erweitert wird.

6.3.1 Anpassung des MicroBlaze für die SoC-Plattform

Nachdem die Grundeinstellung des MicroBlaze vorgenommen wurden, wird dieser für das gewünschte System angepasst. Dazu werden zuerst die IP-Cores angepasst, welche durch den “Base System Builder wizard“-Assistenten hinzugefügt wurden.

Dazu werden in der System Assembly View die RS232-Uart Schnittstellen angepasst. Die beiden RS232 Schnittstellen werden mit einer Bitrate von 19200 bps und 8 Datenbits konfiguriert. Diese Bitrate resultiert aus der maximal angegeben Bitrate des 3-Achsen Kompass [Oce10]. Für die Datenübertragung zum seriellen Terminal des Entwicklungscomputers reicht diese Bitrate auch vollkommen aus.

Die drei Standard-GPIO-Module LEDs, Push-Buttons und DIP-Switches wurde schon bei den Grundeinstellungen des MicroBlaze hinzugefügt. Die Push-Buttons und DIP-Switches sollen dem Benutzer der SoC-Platform die Möglichkeit geben das System bequemer zu

steuern. Die DIP-Switches sollen in diesem System zur Einstellung wechselnder Systemeigenschaften verwendet werden. Dies ist nötig, da im System später verschiedene Arbeitsweisen verglichen werden sollen. Die Push-Buttons sollen im System als zweite Möglichkeit der Steuerung verwendet werden. Somit ist es später möglich die Steuerung mittel Kompass oder Push-Buttons durchzuführen. Die Push-Button-Steuerung bietet dabei eine bessere Testmöglichkeit des Systems.

Es gibt verschiedene Verfahren Änderungen an den GPIOs der Buttons und des Dip-Switches zu erkennen.

Eine gute Variante zum Erkennen von Änderungen am Dip-Switch wäre das sogenannte Polling. Dabei würde in der Main-Routine der Zustand des GPIOs abgefragt und durch ein Vergleich des Zustandes mit dem vorangegangen Zustand könnten Änderungen festgestellt werden. Ein Dip hat nur den Zustand Ein (1) oder Aus (0). Da der Zustand in der Main-Routine abgefragt wird, lässt sich die Zeit bis zum Erkennen einer Änderung nicht vorhersagen. Diese wäre im System aber nicht weiter tragisch, da durch das Dip-Switch das System in ein anderen Arbeitsmodus gesetzt wird. Die Echtzeit-Audioverarbeitung würde in keiner Weise beeinträchtigt.

Um einen Tastendruck an den GPIO-Buttons zu erkennen eignet sich dieses Verfahren nicht. Solange die Taste gedrückt wird, ist der Zustand Ein (1). Sobald die Taste wieder losgelassen wird, ist der Zustand Aus (0). Durch Polling könnte man solche Ereignisse verpassen. Die Lösung des Problems ist das Auslösen eines Interrupts. Bei einer Änderung des Zustandes einer Taste würde somit immer ein Interrupt ausgelöst. Um keine wirkliche Auswirkung auf die Echtzeitverarbeitung des Systems zu haben, wird die ISR kurz gehalten. Dies könnte durch Setzten eines Flags geschehen. Die ISR setzt ein Flag, welches von der Main-Routine überprüft wird. Da das Drücken eines Tasters keine Echtzeitanforderung ist, ist das Polling des Flags unkritisch.

Für das System wurde sich entschlossen beide GPIO-Eingänge gleich aufzubauen. Da die Buttons nur über Interrupts abfragbar sind, werden auch die Dips per Interrupt abgefragt. Die ISR dieser GPIOs wird im späteren System ein Event-Flag setzen, welches durch die Hauptroutine ausgewertet wird. Um das DIP-Switch und die Push-Buttons als Interruptquelle einzustellen, muss der Interrupt übers Configure IP Menü freigeschaltet werden. Die Anbindung an das System wird im Kapitel des Interruptcontroller (6.3.3) beschrieben.

Standardmäßig sind die GPIOs als Input/Output eingestellt. Dieses ist für die drei genannten GPIOs nicht notwendig. Da die LEDs nur als Ausgabe genutzt werden sollen und die Push-Buttons sowie das Dip-Switch nur als Eingang genutzt werden.

Um die Richtung der GPIOs zu steuern gibt es im IP-Core xps_gpio ein Tri-State der die Richtung schaltet. Über die System Assembly View im Unterpunkt Ports wird einge-

stellt, ob der GPIO als Input/Output oder als Input oder als Output betrieben werden soll. Da die LEDs nur als Output arbeiten sollen, wird bei dem LED IP-Core die Connection aus **GPIO_IO** gelöscht und durch eine neue externe Connection in **GPIO_IO_O** ersetzt. Das gleiche Vorgehen ist bei den anderen beiden Eingangs-GPIOs (Button, Dip-Switch) zu wa"hlen, mit dem Unterschied, dass die neuen externen Connections mit **GPIO_IO_I** verbunden werden.

6.3.2 Anbindung des Oszilloskop-Moduls

Um im späteren System Zeitmessungen durchzuführen, wird ein weiteres GPIO Modul hinzugefügt. Dieser IP-Core befindet sich im IP Catalog im Unterordner General Purpose IO. Mittels Add IP wird eine neuer GPIO zum MicroBlaze hinzugefügt. Dieser jetzt neu hinzugefügte IP-Core bekommt den Namen Oszi_4Bit und wird an den PLB Bus angeschlossen. Über Configer IP wird der Oszi_4Bit IP-Core angepasst. Dabei wird der Channel 1 des GPIO-Moduls mit 4 Bit eingestellt. Nachdem Erzeugen des neuen GPIOs wird der 4-Bit breite Ausgang aus dem FPGA-Modul herausgeführt. Dazu wird im Tab Ports der System Assembly View des Oszi-4Bit-IP-Core eine neue externe Connection an den **GPIO_IO_O** Ausgang angeschlossen. Nach diesen Arbeitsschritt sieht die MHS Datei dafür wie folgt aus.

```
BEGIN xps_gpio
 PARAMETER INSTANCE = Oszi_4Bit
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81420000
 PARAMETER C_HIGHADDR = 0x8142ffff
 PARAMETER C_GPIO_WIDTH = 4
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO_O = Oszi_4Bit_GPIO_IO_O
END
```

Um die Ausgänge des IPs auf die richtigen Ausgangspins zu legen, wird die UCF-Datei erweitert. Die UCF Datei befindet sich im Anhang B.

6.3.3 Hinzufügen des Interruptcontrollers

Der MicroBlaze hat nur ein Interrupt-Eingang. Um wie in diesem System mehrere Interruptquellen verwalten zu können benötigt das System einen Interruptcontroller. Dieser Interruptcontroller kann bis zu 32 Interruptquellen mit verschieden Prioritäten verwalten. Siehe Abbildung 22.

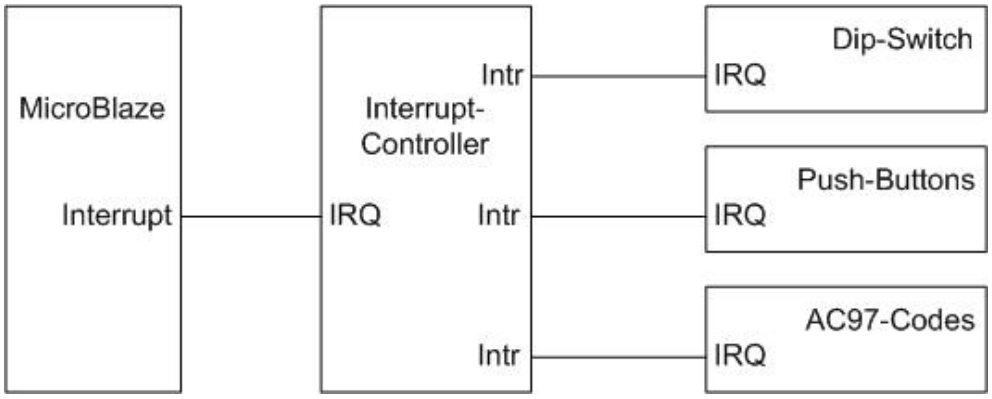

Abbildung 22: Mehrere Interrupt-Quellen mittels Interuptcontroller an eine MicroBlaze angeschlossen

Über den IP-Catalog im Unterordner Clock, Reset, Interrupt wird der XPS Interrupt Controller dem System hinzugefügt. Dieser wird ebenfalls über den PLB Bus an das System angebunden. (**SPLB** = mb_plb). Im Tab Port der System Assembly View wird dieser Interrupt-Controller mit dem MicroBlaze verbunden. Dazu wird im Interruptcontroller xps_intc_0 der **IRQ** Pin mit einer neuen Connection verbunden. Diese Verbindung erhält den Namen microblaze_0_Interrupt. Im Tab des microblaze_0 wird diese Verbindung mit dem Interrupt-Pin verbunden. Im Tab xps_intc_0 werden jetzt noch die zurzeit zur Verfügung stehenden Interruptquellen angeschlossen. Dazu müssen an den Intr die Interruptquellen aufgelistet werden. Die XPS Entwicklungsumgebung bietet dafür das kleine Tool Interrupt Connection Dialog an. Über dieses Tool werden die beiden bisherigen Interrups des DIP-Switches und der Push-Buttons an den Interruptcontroller angeschlossen und gleich mit einer Priorität versehen.

```
BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT Intr = Push_Buttons_5Bit_IP2INTC_Irpt &
             DIP_Switches_8Bit_IP2INTC_Irpt
 PORT Irq = microblaze_0_Interrupt
END
```

6.3.4 Anbindung des LCD-IP-Cores

Um den Status des Systems auszugeben, ist das auf dem Board vorhandene LCD-Display perfekt zu gebrauchen. Der im Kapitel 6.1.1 erzeugte IP-Core für das LCD wird dazu in den MicroBlaze integriert. Der erzeugte IP-Core wird dafür im Unterordner pcores des Projektordners abgelegt. Danach ist dieser über den IP-Catalog/Projekt local PCores/User zu finden und wird dem System hinzugefügt. Wie auch bei den andern IP-Cores wird dieser IP-Core über den PLB-Bus angebunden. Im Untertab Ports wird der Port LCD mit einer neuen externen Connection verbunden.

```
BEGIN xps_intc
 PARAMETER INSTANCE = lcd_ip_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xcf400000
 PARAMETER C_HIGHADDR = 0xcf40ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT lcd = lcd_ip_0_lcd
END
```

Der LCD-Port wird daraufhin mit den Pins des LCD-Displays verbunden. Dazu wird die UCF-Datei erweitert. UCF-Datei siehe Anhang B. Die Pins des Virtex5 Boards wurden aus der Virtex5-ML507-Schematic Datei entnommen [Xil08a].

6.3.5 Anbindung AC97-IP-Cores

Das Virtex5 ML507 besitzt ein AC97-Codec-Baustein. Der im Kapitel 6.1.2 beschrieben IP-Core wird dafür in dem MicroBlaze hinzugefügt. Dazu wird der AC97-IP-Core im Unterordner pcores des Projektes abgelegt. Nachdem lässt sich der IP-Core über den User-Unterordner des IP-Catalogs hinzufügen. Auch dieser IP-Core wird über den PLB Bus mit dem MicroBlaze verbunden. Über Configer IP muss der AC97-IP-Core nun eingestellt werden. Da der AC97-Codec Audio-Samples lesen und abspielen soll, werden die beiden Defines **C_PlayBack** und **C_Record** auf 1 gesetzt. Das spätere System wird zur Echtzeitaudioverarbeitung eingesetzt. Aus diesem Grund genötigt der AC97-IP-Core keinen großen Zwischenspeicher und es wird auf einen großen BRAM-Block verzichtet. Stattdessen wird ein kleines Schieberegister als Zwischenspeicher für die Audio-Samples genutzt. Diese Einstellung wird dem IP-Core über das Setzen einer 0 für das Define **C_USE_BRAM** mitgeteilt. Der AC97-IP-Core soll Interrupts erzeugen. Durch das Setzten einer 1 für das **C_intr_level** erzeugt dieser IP-Core Interrupts, wenn der Zwischenspeicher zur Hälfte gefüllt ist. Über den Tab Ports in der System Assembly View werden jetzt die Ein und Ausgänge verbunden. Der Port Interrupt des AC97-IP-Cores bekommt eine neue Verbindung. Diese Connection wird daraufhin an den Interrupt-Controller angeschlossen. Dabei ist zu beachten, dass dieser Interrupt die höchste Priorität bekommt. Die Ports **AC97Reset_n**, **Bit_Clk**, **SData_In**, **SData_Out** und **Sync** werden jeweils mit einer neuen externen Connection verbunden. Daraufhin werden diese Pins an den AC97-Baustein angeschlossen. Die UCF Datei wird dazu

erweitert. Siehe Anhang B. Die MHS-Datei für den AC97-Codec-IP-Core sieht wie folgt aus.

```
BEGIN xps_ac97
 PARAMETER INSTANCE = xps_ac97_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xc9c00000
 PARAMETER C_HIGHADDR = 0xc9c0ffff
 PARAMETER C_MEM0_BASEADDR = 0xc6e00000
 PARAMETER C_MEM0_HIGHADDR = 0xc6e0ffff
 PARAMETER C_INTR_LEVEL = 1
 PARAMETER C_USE_BRAM = 0
 BUS_INTERFACE SPLB = mb_plb
 PORT Interrupt = xps_ac97_0_Interrupt
 PORT Bit_Clk = xps_ac97_0_Bit_Clk
 PORT AC97Reset_n = xps_ac97_0_AC97Reset_n
 PORT Sync = xps_ac97_0_Sync
 PORT SData_Out = xps_ac97_0_SData_Out
 PORT SData_In = xps_ac97_0_SData_In
END
```

6.3.6 Anbindung der FSL-FIR-Stereo-Filter

Der im Kapitel 6.1.3 erläuterte FIR-Stereo-Filter wird über zwei unidirektionale FSL-Busse mit dem Embedded System verbunden. Zuvor wird der MicroBlaze mit entsprechenden FSL-Schnittstellen versehen. Durch Doppelklick auf den MicroBlaze in der System Assembly View öffnet sich der “XPS Core Config“-Assistent. Im Unterbereich Bus dieses Assistenten wird das Select Stream Interface auf FSL gesetzt. Die Anzahl der Stream Links wird auf die maximale Anzahl von FSL-Bus-Interfaces des MicroBlaze (16 Stück) gesetzt. Der FSL-FIR-STEREO IP-Core wird im Unterordner pcores abgelegt und ist somit im IP-Catalog/User aufgelistet. Die Filter werden mit einen weiteren Assistenten (Hardware/Configure Coprocessor) an das System angeschlossen. Durch Auswahl des FSL-FIR-Stereo IP-Core wird dieser mit ADD in das System übernommen. Dabei fügt der Assistent auch die nötigen FSL Master und Slave Interfaces hinzu und verbindet diese sofort dem IP-Core. Dieses erleichtert das Hinzufügen erheblich.
Nach dem Assistent sieht die MHS-Datei wie folgt aus.

```
BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 ...
 PARAMETER C_FSL_LINKS = 16
 BUS_INTERFACE SFSL0 = fsl_fir_stereo_0_to_microblaze_0
 BUS_INTERFACE MFSL0 = microblaze_0_to_fsl_fir_stereo_0
 ...
END
```

```
BEGIN fsl_v20
 PARAMETER INSTANCE = fsl_fir_stereo_0_to_microblaze_0
 PARAMETER HW_VER = 2.11.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT FSL_Clk = clk_125_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN fsl_v20
 PARAMETER INSTANCE = microblaze_0_to_fsl_fir_stereo_0
 PARAMETER HW_VER = 2.11.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT FSL_Clk = clk_125_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN fsl_fir_stereo
 PARAMETER INSTANCE = fsl_fir_stereo_0
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE MFSL = fsl_fir_stereo_0_to_microblaze_0
 BUS_INTERFACE SFSL = microblaze_0_to_fsl_fir_stereo_0
 PORT FSL_Clk = clk_125_0000MHzPLL0
END
```

Alle bis auf ein Filter werden mit einer Filterlänge von 512 betrieben. Dieses wird dem IP-Core über das Define **C_ORDER** mitgeteilt. Dabei sind 14 dieser Filter für die HRTF-Filterung vorgesehen (Kapitel 3.2). Ein Filter wird für die in der Bachelor-Arbeit von S. Sima entwickelten Kopfhörer-Kompensation eingesetzt. Der Laufzeitfilter wird auf eine Filterlänge von 32 gesetzt. Diese Filterlänge wurde durch den physikalischen Ansatz sowie mit der Annahme, dass die Wegstrecke zwischen beiden Ohren maximal 21,5 cm beträgt, errechnet. Dabei würde bei einer Samplerate von 48 kHz für den ITD-Filter eine Filterlänge von 30,2 ausreichen.

7 Softwareentwicklung für des Embedded Systems

In diesem Kapitel wird die Entwicklung der C-Anwendung für das Embedded System beschrieben. Dabei kommt das Xilinx Software Development Kit (SDK) (Kapitel 4.3) zum Einsatz. In das C-Projekt werden Module aus der Masterarbeit von Jan Kuhr [Kuh10] eingebunden. Dabei handelt es sich um die C-Funktion für die Auswertung der Kompasswerte, sowie Funktionen zum Lesen und Schreiben auf der CF-Karte.

Für die zu entwickelnde Anwendung wird zunächst ein neues C-Projekt in Xilinx SDK erzeugt. Dieses Projekt muss für diese Anwendung angepasst werden. Die Hardware wird durch ein Board Support Package bereitgestellt. Ebenso muss das Linker Skript und vieles mehr eingestellt werden (Kapitel 7.1).

Erst nach den Projekteinstellungen ist es möglich die Anwendung zu schreiben, da die Hardware jetzt über entsprechende Treiber angesprochen werden kann. Durch die Initialisierung werden Einstellungen an den Hardware-Modulen vorgenommen. Dieses geschieht direkt nach dem Programmstart in der Main-Routine und wird im Kapitel 7.2 beschrieben.

Daraufhin wird das Programm zur 3D-Audio-Echtzeitverarbeitung geschrieben. Hierbei kommen Software-Erweiterungen für die Benutzerinteraktion, Interrupt-Service-Routinen und die Funktion zum Audio-Signal-Crossfading zum Einsatz. Des Weiteren kommt Logik zum Umschalten des Systems zwischen den vier zu vergleichen HRTF-Ansätzen zum Tragen.

7.1 3DAudioSystem Software-Application-Project

Innerhalb des Xilinx SDK wird die Entwicklung von mehreren Software-Applikationen pro Hardwaredesign unterstützt. Es wird ein neues Software-Application-Projekt mithilfe der SDK Entwicklungsumgebung angelegt.

Ein Board Support Package (BSP) wird erzeugt. Dieses BSP enthält die Low-Level-Treiber für das in Xilinx EDK entwickelte Hardware-Design. Da das entwickelte Hardware-Design ein Memory Mapped I/O System ist, benötigt man die entsprechenden Adressen der Hardware-Module. Dazu wird im BSP die Datei “xparameters.h“ erzeugt. Diese Datei enthält Defines und die Adressen jeder Hardware-Komponente. Des Weiteren werden Treiber für bestimmte Hardware-Komponenten erzeugt. In diesem Projekt wird ein zusätzlicher Treiber für das System ACE Interface hinzugefügt. Dieses Xilinx-Treiber-Interface wird zum Zugriff auf das FAT-Dateisystem der CF-Card verwendetet. Auf der CF-Card werden im späteren System die Koeffizientensätze gespeichert, die für das System benötig werden. Es ist möglich, den Ressourcenverbrauch der Bibliotheksfunktionen, vor der Kompilierung, in das Softwaresystem einzustellen. Die maximale Anzahl gleichzeitig geöffneter Dateien

im späteren System wird auf 1 gesetzt. (**CONFIG_MAXFILES**=1). Da innerhalb der prozeduralen Dateiverarbeitung nie mehr als eine Datei zurzeit bearbeitet wird. **CONFIG_WRITE** wird auf true gesetzt, somit ist im späteren System das Schreiben auf die CF-Card gewährleistet. Der Parameter **CONFIG_BUFCACHE_SIZE** wird auf 1024 Byte reduziert, da jeder Koeffizientensatz aus 512 Koeffizienten besteht, die jeweils 2 Byte groß sind. (512 * 2 Byte = 1024 Byte). Durch diese Änderung des Treibers wird der Speicherbedarf der Bibliothek verringert und somit auch die Größe der Applikation.

Die zu entwickelnde Software übersteigt die 64 kB Grenze des internen BRAMs und wird aus diesem Grund in den DDR2-RAM ausgelagert. Die Ausführung der Systemsoftware innerhalb des DDR2-RAMs macht die Anpassung des Linker-Skripts notwendig. Standardmäßig wird das komplette C-Projekt mit dem ausführbaren Code, allen Variablen, Konstanten sowie der Stack und Heap in den internen 64 kB großen BRAM gelinkt. Mithilfe des Linker-Skript-Assistenten lassen sich die Adress- und Speicherbereiche der Software in dem 256 MB großen DDR2-RAM verlagern. Durch die Verlagerung des Programms aus dem BRAM in dem DDR2-RAM wird die Ausführgeschwindigkeit deutlich verlangsamt. Da das Hardware-Design aber über Caches auf den DDR2-RAM zugreift, wird die Zugriffsgeschwindigkeit wieder deutlich verbessert und stellt kein Problem dar.

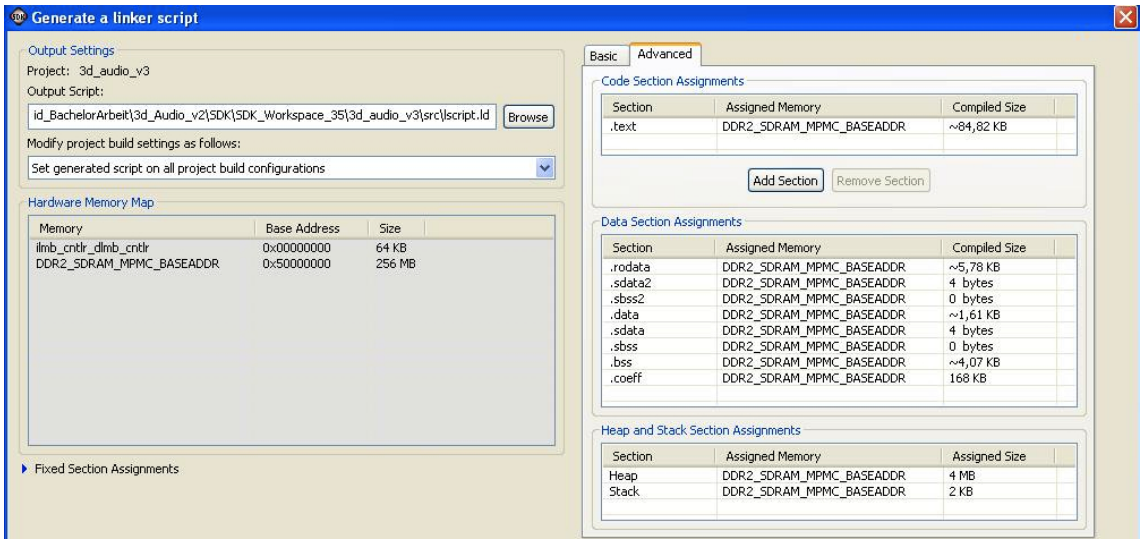

Abbildung 23: Das Linker-Script für die zu entwickelnde Software

Es wird ein neuer Linker-Bereich hinzugefügt. Diese Section heißt .coeff. In ihr werden im späteren System alle Koeffizienten abgelegt.

Die Heapgröße für das spätere System wird vergrößert. Im System sollen Tests der Filterung durchgeführt werden. Hierzu ist geplant, dass Audio-WAVE Dateien von der CF-Card gelesen und daraufhin gefiltert werden. Die gelesenen Audio-WAVE-Datein sowie das Ergebnis der Filterung müssen zwischengespeichert werden. Als Zwischenspeicher empfiehlt sich der Heap Bereich. Durch *malloc()* lassen sich der Größe der Audiodatei entsprechende Speicherblöcke allokieren. Das gefilterte Ergebnis lässt sich zurück auf die CF-Card schreiben, bevor der Speicherbereich freigegeben wird.

Eine weitere wichtige Einstellung des Projektes ist die Optimierungsstufe. Diese lässt sich über die Projekteigenschaften einstellen. Der im SDK eingebundene gcc-Kompilier unterstützt verschiedene Optimierungsstufen. In diesem Projekt wird die Optimierungsstufen -O2 eingeschaltet. Ohne diese Optimierung ist die Geschwindigkeit des Programmes deutlich zu langsam.

7.2 Initialisierung der SoC-Plattform

Die Initialisierung der verschiedenen Hardware-Module des Embedded Systems, sowie das Laden der Koeffizienten wird direkt nach dem Start der *main()*-Routine durchgeführt. Dazu wurde die Methode *init()* geschrieben.

Als Erstes werden die beiden Caches aktiviert. Dazu gibt es die Funktion *Xil_ICacheEnable()* und *Xil_DCacheEnable()*, die durch das Board Support Package zur Verfügung gestellt werden. Danach arbeitet das Embedded System mit den beiden Caches für Daten und Instruktion.

Mittels der Methode *LCDInit()* und *LCDOn()*, die in der Datei “lcd.c“ implementiert wurden, lässt sich das LCD-Display initialisieren und aktiv schalten. Somit kann das System im späteren Betrieb den aktuellen Zustand über das LCD-Display ausgeben.

Daraufhin lässt sich der AC97-Baustein sowie der dazugehörige AC97-IP-Core einstellen. Dazu wurde die Methode *ac97_init()* in der Datei “ac97.c“ geschrieben. Diese Methode resetet zuerst den AC97-Codec-Baustein und wartet, bis dieser wieder bereit zum Arbeiten ist. Anschließend werden alle nicht erforderlichen Eingänge, Ausgänge und sowie nicht gewünschte Funktionen ausgeschaltet. Der gewünschte Line-IN Eingang und Line-Out Ausgang wird freigeschaltet und die Abtastrate des Bausteins auf 48 kHz gesetzt. Zum Einstellen der Register des Bausteins gibt es die Funktion *XAC97_WriteReg(baseaddr,register offset, data)*. Nach jedem Schreiben auf dem AC97-Baustein mittels dieser Funktion, muss gewartet werden, bis dieser wieder aufnahmebereit ist. Dazu gibt es die Funktion *XAC97_AwaitCodecReady()*. Beim Ändern der Abtastrate ist zu beachten, dass zuvor alle AD-Wandler bzw. DA-Wandler ausgeschaltet werden, bevor die Abtastrate geändert wird. Sonst ist beim Einstellen des AC97-Bausteins nichts zu beachten. Alle Register und sonstig-

en Einstellungen des AC97-Codec-Bausteins können dem Datenblatt entnommen werden [DEV05].

Der 3-Achsen-Kompass wird mittels der zweiten Seriellen Schnittstelle (Uart2) an die SoC-Plattform angeschlossen. Die Initialisierung wird über den Befehlsaufruf *XUartLite_Initialize()* des BSPs durchgeführt.

Damit sind alle Hardware-Komponenten-Initialisierungen bis auf die Initialisierung der Interrupts durchgeführt. Im Anschluss daran werden die Koeffizienten von der CF-Card gelesen. Dafür wurde die Datei “coeff.c“ entwickelt. Die Speicherung der Koeffizienten jedes HRTFs (Sima, Kemar, Sima ITD, Kemar ITD) wird in zwei globalen dreidimensionalen Short-Integer Arrays abgelegt.

```
short hrtfptrArrL<TYPE> [X][Y][Z] __attribute__ ((section (".coeff")));
short hrtfptrArrR<TYPE> [X][Y][Z] __attribute__ ((section (".coeff")));
```

Die feste Definition der Werte für X,Y, Z im Programmcode sorgt für eine Allokation des notwendigen Speicherplatzes zur Kompilierungszeit.

Dabei ist die X-Dimension, die Anzahl der verfügbaren Elevationsebenen. In allen HRTF gibt es 3 verschiedene Elevationwinkel. Darum ist X = 3. Zum Zugriff auf der Elevationsebene muss ein Mapping der Arrayindizes auf die tatsächlichen Winkelwerte stattfinden. Dabei wird der negative Elevationwinkel auf den Index 0 abgebildet. Die Elevationwinkel 0° wird auf den Index 1 abgebildet. Auf positiven Elevationwinkel wird mit dem Index 2 zugegriffen.

Die zweite Dimension Y gibt die Anzahl der Koeffizientsätze pro Elevationsebene an. Um Speicherplatz zu sparen, gibt es nur die Koeffizientsätze von 0° bis 180°. Die Filterergebnisse für Winkel größer 180° wird durch einfache Spiegelung der Filterergebnisse für das linke und rechte Ohr errechnet. Da alle HRTF-Messreihen mit einem Azimuthwinkel von 30° arbeiten, werden auch nur die Winkel 0°, 30°, 60°, 90°, 120°, 150°, 180° benötigt. Darum ist Y = 7. Der Zugriff wird wieder durch Mapping der Arrayindizies auf den tatsächlichen Winkelwert stattfinden.

Die Z-Dimension ist gleichbedeutend mit der Anzahl von Koeffizienten pro Koeffizientsatz und daher ist Z = 512.

Der Speicherbedarf für alle Koeffizienten beträgt in diesem System 168 kByte. (4 Filtersätze * 3 Elevation-Ebenen * 7 Azimuthwinkel * 2 Kanäle * 512 Koeffizienten * 16 Bit)

Die Funktion *readBinaryCoeffFile()* aus der Masterarbeit von Jan Kuhr wird genutzt um die Koeffizienten von der CF-Card einzulesen und an dem entsprechenden Speicherplatz zu speichern. Dabei hat sich herausgestellt das je kleiner der Pfad zu den Koeffizienten-

Datein ist, die Geschwindigkeit des Einlesens deutlich schneller wird. Die Funktion *load_filter(Elevation-Ebene)* inizaliziert die Filter der SoC-Plattform mit den entsprechenden HRTF-Filterkoeffizienten.

Zum Schluss der Initialisierung des Systems werden jetzt die Interrupts freigegeben. Die beiden GPIOs (Push-Buttons und Dip-Switch) arbeiten im späteren System über Interrupts. Mit dem Befehl *XIntc_RegisterHandler(INTC_ADDRESSE,Interrupt ID, ISR-Name, ISR argument)* wird die ISR an die Interrupts angebunden. Die ISRs werden im Kapitel 7.3 beschrieben. Die Interrupt-Controller Baseadresse sowie die jeweilige Interrupt-ID werden aus der Header-Datei “xparameters.h“ entnommen. Des Weiteren werden die Interrupts der beiden GPIOs freigeschaltet. Dazu wird das Global Interrupt Enable Register (GIE) und IP Interrupt Enable Register (IER) gesetzt. Das Setzen dieser Register erfolgt mit dem Befehl *XGpio_WriteReg(Baseaddress, Reg-Offset, Maske)*. Somit sind die beiden GPIOs mit dem Interrupt-Controller verbunden sowie der Interrupt freigeschaltet.

Die AC97-ISR wird im Kapitel 7.5 beschrieben. Diese ISR wird mittels dem Befehl *XIntc_RegisterHandler()* angebunden. Da der Interrupt des AC97-IP-Core standardmäßig ausgeschaltet ist, wird dieser mit der Methode *ac97_out_interrupt_en()* freigeschaltet. Diese Funktion gibt den Interrupt für das Record-FIFO frei.

Jetzt sind alle Interrupts mit den entsprechenden ISR verbunden und alle Interruptquellen freigeschaltet. Somit können die Interrupteingänge des Interruptcontrollers freigeschaltet werden. Dieses wird mit der Funktion *XIntc_EnableIntr()* aus dem BSP erledigt. Der Interrupt des Interruptcontrollers wird über die Funktion *XIntc_MasterEnable()* freigeschaltet. Nun wird der Interrupteingang am MicroBlaze freigeschaltet. Dazu stellt das Board Support Package die Methode *microblaze_enable_interrupts()* zu Verfügung. Hiernach sind die Interrupts für das System freigeschaltet.

7.3 Benutzerinteraktion - DIP-Switch / Push-Button ISR

Das Dip-Switch und die Push-Buttons werden zur Benutzerinteraktion genutzt. Um die Echtzeitaudioverarbeitung nicht zu beeinflussen, werden die Interrupt Service Routinen sehr kurz gehalten.

Bei Änderungen am Dip-Switch wird ein Interrupt ausgelöst. Die dazugehörige ISR *dip_int_handler()* wird aufgerufen. Diese setzt ein Flag, welches von der Main-Routine abgefragt wird. Ist das Dip-Flag gesetzt, wird der Zustand des Dips gelesen und ausgewertet. Die nachfolgende Messung zeigt, dass die Dip-Switch-ISR nur 270 ns Zeit braucht. Dabei ist zu beachten, dass das Setzen und Löschen eines Pins ca. 250 ns braucht. Diese ISR hat somit keine Auswirkung auf die Echtzeit-Audio-Verarbeitung.

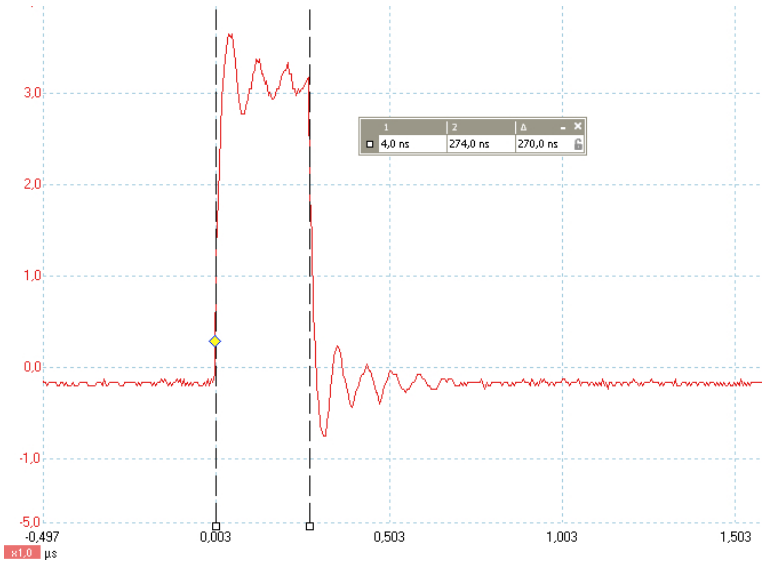

Abbildung 24: Zeitmessung der Dip-Switch-ISR

Die Push-Buttons erzeugen bei Betätigung ebenso einen Interrupt, der die Ausführung der ISR *push_button_int_handler()* triggert. Die Push-Buttons dienen dem Benutzer als zweite Möglichkeit, die Position der Schallquelle zu bewegen. Durch Drücken der Push-Button rechts oder links wird die horizontale Ebene um 10° weitergezählt. Es wird beachtet, dass der Wertebereich immer zwischen 0° bis 360° ist. Beim einem Tastendruck der Push-Buttons unten oder oben wird die vertikale Ebene um 5° verändert. Beachtet wurde, dass die verscheiden HRTF-Messreihen auf unterschiedlichen Intervalle beruhen. Die Sima Messreihen haben das Intervall -41° bis +37°. Die Kemar Messreihen arbeiten von -30°. bis +30°. Wie auch bei der horizontalen Ebene wurde darauf geachtet, dass die Intervallgrenzen je nach Zustand des Systems (Sima, Kemar) nicht überschritten werden. Durch

Drücken von PUSH_BUTTON_CENTER werden beide Winkel auf 0° zurückgesetzt, was einer Klangposition genau vor dem Zuhörer, in Köpfhöhe, entspricht. Die über die Push-Buttons erzeugte Position wird dem System über Variablen zur Verfügung gestellt. Die Main-Routine kann daraufhin, je nach Zustandseingabe des Dips, entscheiden, ob über Button-Position oder die Kompass-Position verarbeitet werden soll. Auch in dieser ISR wurde darauf geachtet, dass der Zeitbedarf der ISR sehr klein ist ,um die Echtzeitverarbeitung des 3D-Audio-System nicht zu beeinträchtigen.

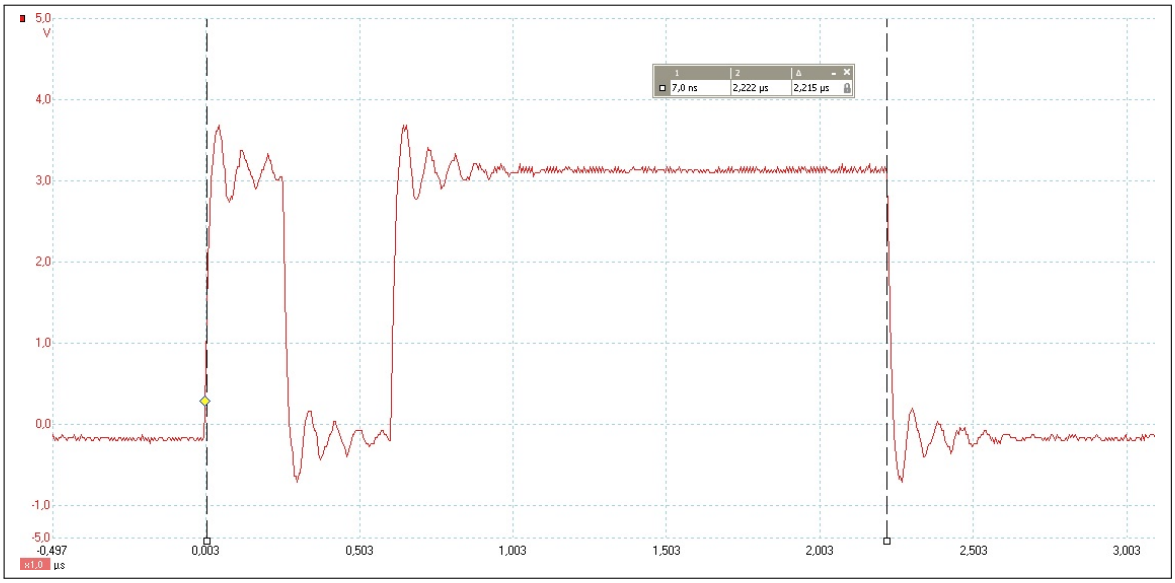

Abbildung 25: Zeitmessung der Push-Button-ISR

Die gesamte Laufzeit der Push-Button-ISR beträgt 2,2 μs. Im ersten Bereich von ca. 250 ns wird der PIN hoch und wieder runter gesetzt. Der nächste Bereich bis ca. 600 ns ist der Bereich der ISR, wo die Register des GPIOs ausgelesen werden. Danach erfolgt die Berechnung der neuen Position. Rechnet man die Zeit für das Setzen des Pins aus der gemessen Zeit heraus, braucht die ISR ca. 1,8 μs. Die ISR hat somit auch keine Auswirkung für die Echtzeit-Audio-Verarbeitung.

7.4 Main-Routine für den MicroBlaze

Nach der Initialisierung der SoC-Plattform arbeitet die Main-Routine in einer Dauerschleife. Der Ablauf dieser Dauerschleife wird in dem Ablaufdiagramm 26 gezeigt.

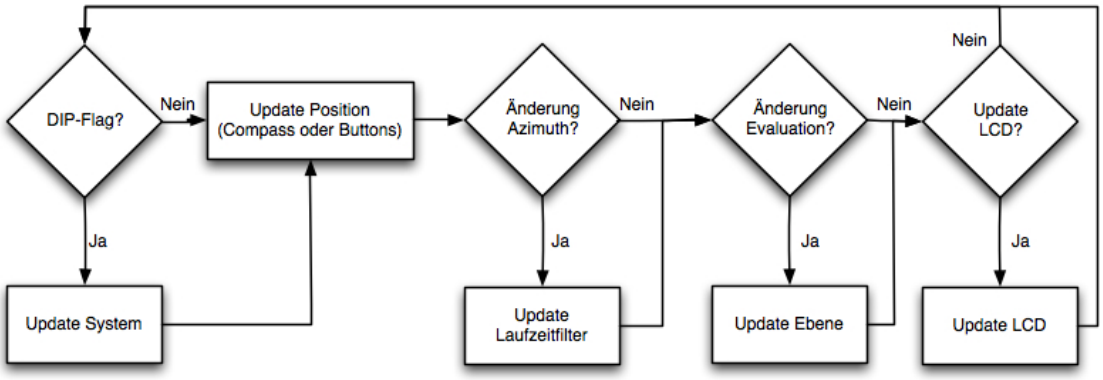

Abbildung 26: Ablaufdiagramm der Main-Routine

Zuerst wird geschaut, ob es ein Interrupt an dem Dip-Switch gegeben hat. Ist das der Fall wird die Methode *updateSystem()* aufgerufen. Diese liest den Zustand des Dips aus und vergleicht diesen mit dem aktuellen Zustand des Systems. Bei einem Unterscheid wird das System dementsprechend geändert. Dabei haben die Dips folgende Bedeutung.

Dip	High	Low
1	HRTF-Filterung aktiv	LoopBack
2	Kemar-Filtersatz	Sima Filtersatz
3	ITD und ILD getrennt verarbeitet	In einem Filter verarbeitet
4	bilineare Interpolation	Interpolation horizontal
5	Button-Steuerung	Kompass-Steuerung
6	/	/
7	/	/
8	Kopfhörerkompensation aktiviert	Kopfhörerkompensation deaktiviert

Tabelle 8: Dip-Switch und dessen Bedeutung für die SoC-Plattform

Danach wird geschaut mit welchen Positionsverfahren (Buttons oder Kompass) gearbeitet wird. Die Position wird der Audio-ISR über globale Variablen mitgeteilt.

Anschließend wird geschaut, ob sich der Azimuthwinkel geändert hat. Sollte dies der Fall sein und das System zurzeit mit dem Verzögerungsglied arbeitet, werden für diese Position

die neuen Filterkoeffizienten für den Laufzeitfilter errechnet. Zu jeder Azimuthwinkel-Ebene liegen dem System die ermittelte Laufzeitdifferenz vor. Durch eine Interpolation zwischen den beiden benachbarten Azimuthwinkel-Laufzeiten lässt sich die ITD für den entsprechenden Positionswinkel bestimmen. Die so ermittelte ITD in Samples wird jetzt prozentual zwischen den benachbarten Filterkoeffizienten verteilt. Somit ist die Berechnung des Laufzeitgliedes fertiggestellt. Beachtet wird, dass das Nachladen des Filters immer direkt nach Beendigung der Audio-ISR geschieht. Daher ist sichergestellt, dass das System nie auf den Verzögerungsfilter zugreift, wenn dieser zurzeit neu geladen wird. Die Abbildung 27 zeigt diesen Ablauf. Der erste blaue Ausschlag zeigt das Erkennen einer neuen Position. Daraufhin berechnet die Main-Routine die neuen ITD-Filter-Koeffizienten. Beim zweiten blauen Ausschlag ist diese Berechnung beendet und das System wartet bis die nächste Audio-ISR (roter Ausschlag) abgelaufen ist, bevor das Nachladen der Filter stattfindet (letzter blauer Bereich). Dieser Vorgang ist im Normalfall im laufenden Betrieb innerhalb von 3 Audiosamples durchgelaufen.

Abbildung 27: Zeitbedarf zum Errechnung der ITD und das Nachladen des Laufzeitfilters

Im nächste Schritt ist zu schauen, ob sich auch der Evaluationwinkel geändert hat. Ist dies der Fall, wird geprüft, ob die mittlere Filterebene durchbrochen wurde und somit die Filterebenen getauscht werden müssen.

Als Letztes wird kontrolliert, ob es nötig ist das LCD zu aktualisieren. Das LCD wird bei großen Änderungen wie Filtertypenänderung oder ca. alle 2 Sekunden aktualisiert.

Zu den Wiederholungsraten wurden Messungen durchgeführt. Diese befinden sich im Anhang C. Wenn das System mit dem Kompass arbeitet, dauert ein Durchlauf der Main-Routine ca. 1,9 μs. Wenn mit den Buttons gearbeitet wird, dauert derselbe Durchlauf nur

760 ns. Dieses ergibt eine Differenz von ca. 1,2 μs, die für das Abfragen der Position des Kompasses benötigt wird.

7.5 Echtzeitaudioverarbeitung in der AC97-ISR

In der Abbildung 28 ist das Ablaufdiagramm der Audioverarbeitung, die in der AC97-Codec-ISR durchgeführt wird, dargestellt.

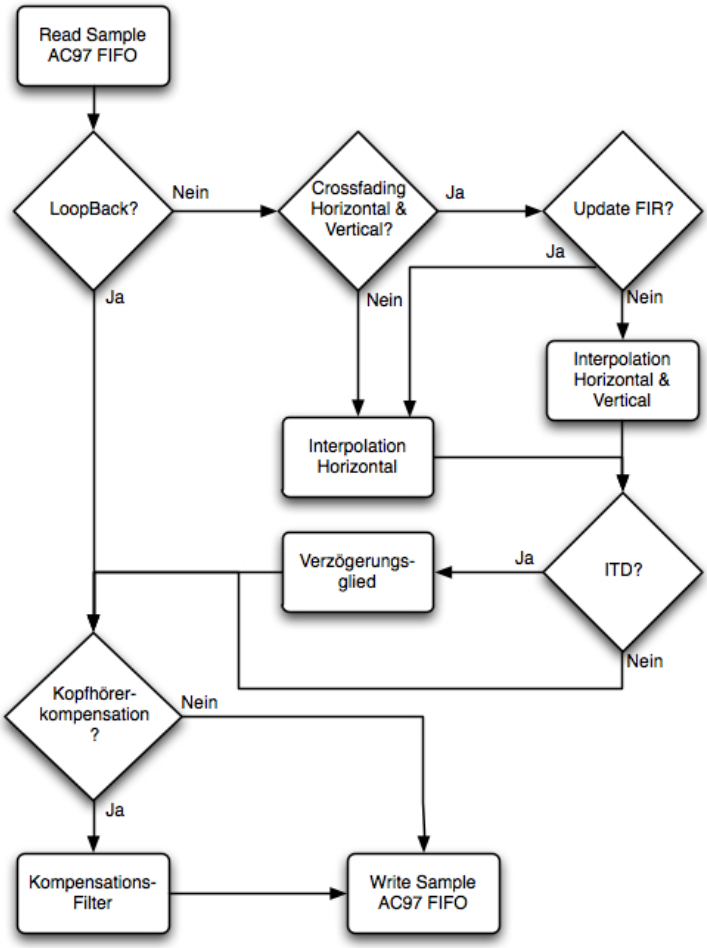

Abbildung 28: Ablaufdiagramm der AC97-ISR

Die komplette Audioverarbeitung wird in der ISR der Audio-Codecs durchgeführt. Nach einem Interrupt des AC97-Codec-IP-Cores wird über den PLB-Bus ein Audio-Sample aus dem Record-FIFO des IP-Cores gelesen.

Daraufhin wird entschieden, wie die Verarbeitung weitergeht. Dafür gibt es drei Möglichkeiten:

- Digitales Loopback
- HRTF-Filterung mit bilineare Interpolation auf der horizontalen Ebene
- HRTF-Filterung mit bilineare Interpolation mit allen 4 Quellnachbarn. Sollten die Filter gerade mit neuen Filterkoeffizienten geladen werden, wird so lange auf die Interpolation der horizontalen Ebene umgeschaltet.

Bei der bilinearen Interpolation wurde darauf geachtet, dass keine Floation-Point-Berechnung durchgeführt wird. Die Berechnung arbeitet nur mit Multiplikation und Schiebeoperation. Somit wird eine schnelle Berechnung ermöglicht, dabei verschlechtert sich aber die Genauigkeit der Interpolation. In diesem System gibt es eine Abweichung von ca 3,47%.

Nach der Filterung mit bilinearer Interpolation wird geprüft mit welchen HRTF-Filter-Typen gearbeitet wurde. Wurden zuvor nur die Pegeldifferenz der HRTF gefiltert, wird das gefilterte und interpolierte Signal durch den Laufzeitfilter gefiltert um eine vollständige HRTF-Filterung durchzuführen.

Zum Schluss der Audioverarbeitung wird geprüft, ob die Kopfhörerkompensation für die in diesem Projekt verwendeten Sennheiser HD 600 Kopfhörer eingeschaltet ist.

Die nachfolgende Tabelle zeigt den Zeitbedarf für die jeweiligen Einstellungen. Anzumerken ist, dass die Zeitmessung leichte Schwankungen ergeben haben. Es macht einen großen Unterschied, ob die Main-Routine den Kompass abfragt oder nicht. Da die AC97-ISR keine Unterschiede bezüglich der Kompass- oder der Button-Steuerung macht lässt sich diese Zeitdifferenz nur auf die Caches zurückgeführen. Dieses hat aber keine Auswirkung auf die Echtzeitaudiobearbeitung, da die Audio-ISR deutlich schneller ist als das neue Audiosamples eintreffen.

Modus	Zeitbedarf der ISR
Loopback	680 ns
Loopback + Kopfhörerkompensation	710 ns
HRTF-Filterung + bilineare Interpolation (horizontal + vertical)	8,44 μs - 10,20 μs
HRTF-Filterung + bilineare Interpolation (horizontal + vertical) + ITD	8,78 μs - 10,69 μs
HRTF-Filterung + bilineare Interpolation (horizontal)	7,28 μs - 7,71 μs
HRTF-Filterung + bilineare Interpolation (horizontal) + ITD	7,80 μs - 7,96 μs

Tabelle 9: Zeitbedarf der AC97-ISR in den verschieden Modis

8 Fazit

Das Ziel dieser Bachelorarbeit war die Entwicklung einer SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern zu entwickeln. Dabei wurde auf Erkenntnisse aus der Masterarbeit von Jan Kuhr aufgebaut. Um für jede Position um den Kopf herum ein Filterergebnis zu haben, wurde aus den benachbarten HRTF-Filter-Stützstellen das Ergebnis in Echtzeit bilinear interpoliert. Der letzte Punkt in dem Projekt war es, die Laufzeitdifferenz und Pegeldifferenz der HRTF getrennt voneinander zu verarbeiten, um ein besseres Klangergebnis zu erzeugen.

Die Positionierbarkeit von Schallquellen beschränkt sich auf den horizontalen 360° Bereich um den Kopf des Hörers, mit einer Schrittweite von 1°. Der vertikale Bereich hängt von den verwendeten Filtertypen ab. Bei der Sima-HRTF ist der vertikale Wertebereich von +37° bis -41°, bei den Kemar-HRFT von +30° bis -30°. Bei beiden Filtertypen ist die vertikale Schrittweite ebenfalls 1°. Die Positionierbarkeit wird über die bilinear Interpolation zwischen den Quellstützstellen in Echtzeit errechnet. Die Qualität des Systems bezüglich der so gefilterten und errechneten Audiosignale, sind zum Erkennen von Positionen einzelner Schallquellen, als gut zu bewerten.

Der zweite neue Ansatz war die Laufzeitdifferenz und Pegeldifferenz der HRTF-Filterung separat zu filtern. In beiden HRTF-Messreihen gab es Fehlmessungen bei den Laufzeitdifferenzen. Aus diesem Grunde konnte nicht auf diese Messreihen zurückgegriffen werden und die Laufzeitdifferenzen wurde mittels eines physikalischen Modells errechnet. Aber auch mit diesem Ansatz hat sich die HRTF-Filterung im System hörbar verbessert.

Das ganze System wurde so ausgelegt, dass es schnellstmöglich auf Positionsveränderungen von dem Kompass reagiert ohne die Echtzeitaudioverarbeitung zu beeinflussen. Dabei kann das System eine deutlich höhere Aktualisierungsrate der Kompass-Position verarbeiten, als der in diesem System eingesetzte Kompass erreicht.

Das Embedded System bietet ausreichende Ressourcen (Prozessorleistung, Speicherkapazität, freiem Platz im FPGA-Chip), sodass auch noch weitere Erweiterungen des Systems möglich sind. Dabei könnte als nächster Schritt die entwickelte Audioverarbeitung in einem eigenem IP-Core implementiert werden. Dieses hätte zum Vorteil, dass nur eine FSL-Schnittstelle benötigt wird und nicht wie zurzeit 15 FSL-Schnittstellen. Somit könnten weitere Entwicklungsschritte wie, die Entfernung zur Schallquelle, mit in das System implementiert werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AC97	Audio Codec 97
AD-Wandler	Analog-Digital-Wandler
BRAM	Block Random Access Memory
BSP	Board Support Package
CF	Compact Flash
DA-Wandler	Digital-Analog-Wandler
dB	Dezibel
DDR	Double Data Rate
DIP	Dual In-Line Package
DSP	Digital Signal Processing / Processor
EDK	Embedded Development Kit
FAT	File Allocation Table
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSL Bus	Fast Simplex Link Bus
FSM	Finite-State Machine
GPIO	General Purpose In Out
HAW	Hochschule für Angewandte Wissenschaften
HRIR	Head Related Impulse Response (kopfbezogene Impulsantwort)
HRTF	Head Related Transfer Function (kopfbezogene Transferfunktion)
HW	Hardware
Hz	Hertz
ILD	Interaural-Level-Difference (Pegeldifferenz)
IP-Core	Intellectual Property Core
IRQ	Interrupt Request
ISR	Interrupt Service Routine
ITD	Interaural-Time-Difference (Laufzeitdifferenz)
JTAG	Joint Test Action Group
LCD	Liquid Cristal Display
LMB Bus	Local Memory Bus
LSB	Least Significant Bit
MAC	Multiply Accumulate
MHS File	Microprocessor Hardware Specification File
MIT	Massachusetts Institute of Technology
MPD File	Microprocessor Peripheral Definition File
MSB	Most Significant Bit
OPB Bus	On Chip Peripheral Bus
PLB Bus	Processor Local Bus
POA File	Peripheral Analyze Order File

RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SDK	Software Development Kit
SLR	Schieberegister
SoC	System on Chip
UART	Universal Asynchronous Receiver Transmitter
UCF File	User Constraints File File
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Literatur

[Bey11] BEYERDYNAMIC: *Headzone - professional 5.1 monitoring headphone systems*. http://www.beyerdynamic.de/kopfhoerer-headsets/kopfhoerer-headsets/headzone/headzone-systemueberblick-und-anwendungsgebiete.html, 01.09.2011

[CL09] CARTY, B. ; LAZZARINI, V.: *Binaural hrtf based süatialization: new approaches and implementation.* Proc. of the 12th Int. Conference on Digital Audio Effects, 2009

[DEV05] DEVICES, ANALOG: *AC97 SoundMAX Codec AB1981B*. 2005

[DSAS11] DEMANT, Christian ; STREICHER-ABEL, Bernd ; SPRINGHOFF, Axel: *Industrielle Bildverarbeitung: Wie optische Qualitätskontrolle wirklich funktioniert*. 3. Auflage. Springer-Verlag Berlin Heidelberg, 2011 (ISBM 978-3642130960)

[Ell06] ELLIOTT, Rod: *Frequency, amplitude and db*. 2006

[Grü08] GRÜNINGEN, Daniel von: *Digitale Signalverarbeitung*. Carl Hanser Verlag, München, 2008 (ISBM 978-3-446-41463-1)

[Kuh10] KUHR, Jan: *Hardware-Software-Codesign eines Echtzeit-Audiosignalverarbeitungssystems zur räumlichen Positionierung virtueller Schallquellen über Stereokopfhörer*. Masterarbeit (HAW-Hamburg), 2010

[Mat] MATHWORKS: *Mathworks Webseite*. http://www.mathworks.de, 31.08.2011

[Mic09] MICROELECTRONICS, TIANMA: *Specification for LCD Module TM162VCA6*. 2009

[MIT11] MIT: *Kemar*. http://sound.media.mit.edu/resources/KEMAR.html, 22.08.2011

[Nat11] *Nationale Roadmap Embedded Systems*. http://www.bitkom.org/files/documents/NRMES-2009-einseitig.pdf, 03.09.2011

[Oce10] OCEANSERVER: *O. T. Inc. Oceanserver technology digital compass Users Guide*. http://www.ocean-server.com, 2010

[RS09] REICHARDT, Jürgen ; SCHWARZ, Bernd: *VHDL-Synthese Entwurd digitaler Schaltungen und Systeme*. 5. Auflage. Oldenbourg Wissenschaftsverlag, München, 2009 (ISBM 978-3-486-58987-0)

[Sch11] SCHWARZ, Bernd: *High Performance Embedded Computing*. Wahlfach HAW-Hamburg (Informatik), 2011

[Sim08] SIMA, Sylvia: *HRTF Measurements and Filter Design for a Headphone-Based 3D-Audio System*. Bachelorarbeit (HAW-Hamburg), 2008

[SS10] SASS, Ron ; SCHMIDT, Andrew G.: *Embedded Systems Design with Platform FPGAs*. Morgan Kaufmann, 2010 (ISBN 978-0-12-374333-6)

[web11] *DAFX Conference*. http://www.dafx.de, 01.09.2011

[Wik11] WIKIPEDIA: *Head-Related Transfer Function*. http://de.wikipedia.org/wiki/HRTF, 22.08.2011

[Xil08a] XILINX: *ML50X schematics*. 2008

[Xil08b] XILINX: *System ACE CompactFlash Solution*. 2008

[Xil10a] XILINX: *EDK 12.3 Concepts, Tools, and Techniques*. 2010

[Xil10b] XILINX: *LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c)*. 2010

[Xil10c] XILINX: *OS and Libraries Document Collection*. 2010

[Xil11a] XILINX: *Partial Reconfiguration Flow*. 2011

[Xil11b] XILINX: *UserGuide ML505/ML506/ML507 Evaluation Platform*. 2011

[Zöl02] ZÖLZER, Udo: *digital audio effects (DAFX)*. 1. Auflage. John Wiley and Sons, 2002 (ISBM 978-0471490784)

A Hardware

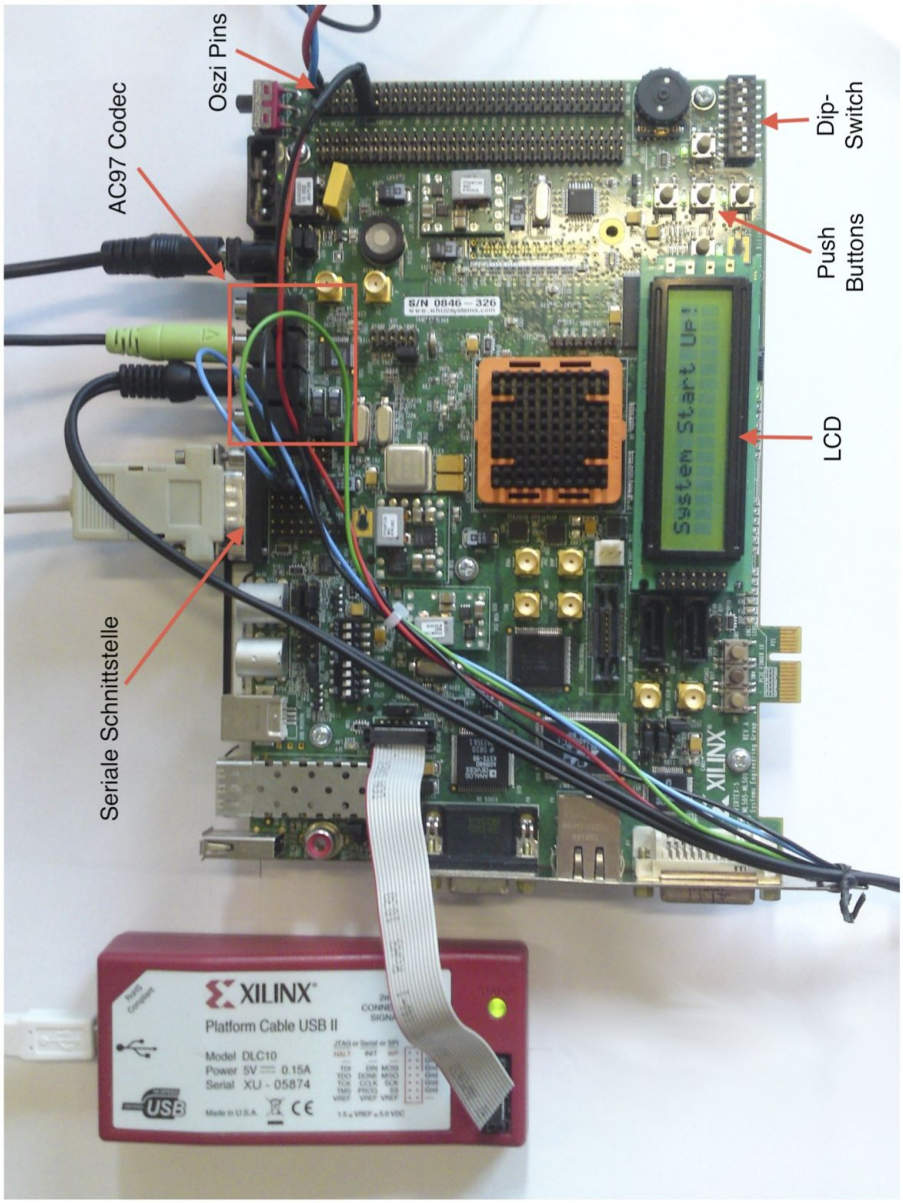

Abbildung 29: Virtex5 ML 507

Abbildung 30: Kopfhörer Sennheiser HD 600 + Kompass

B Embedded System

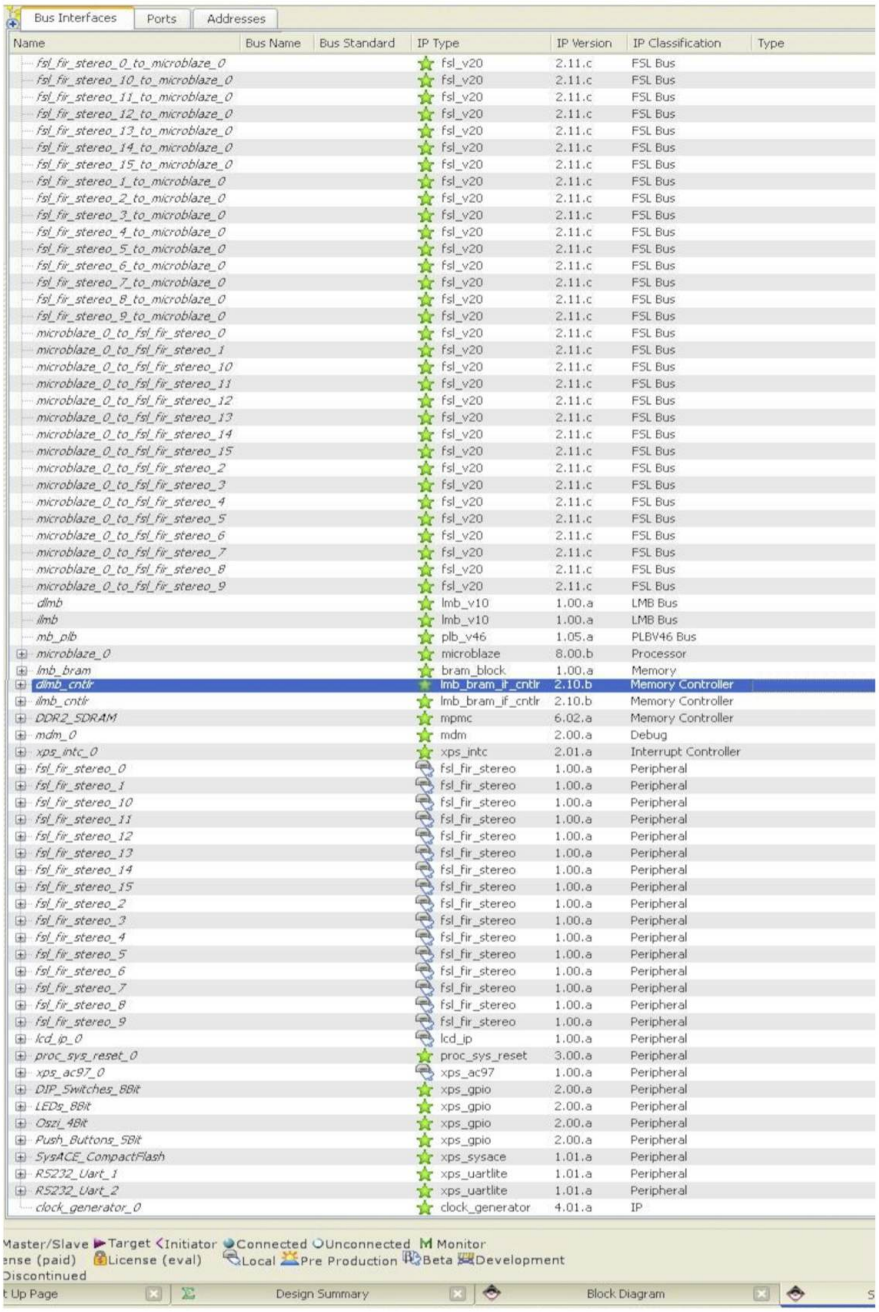

Abbildung 31: Bus Interfaces des Embedded System

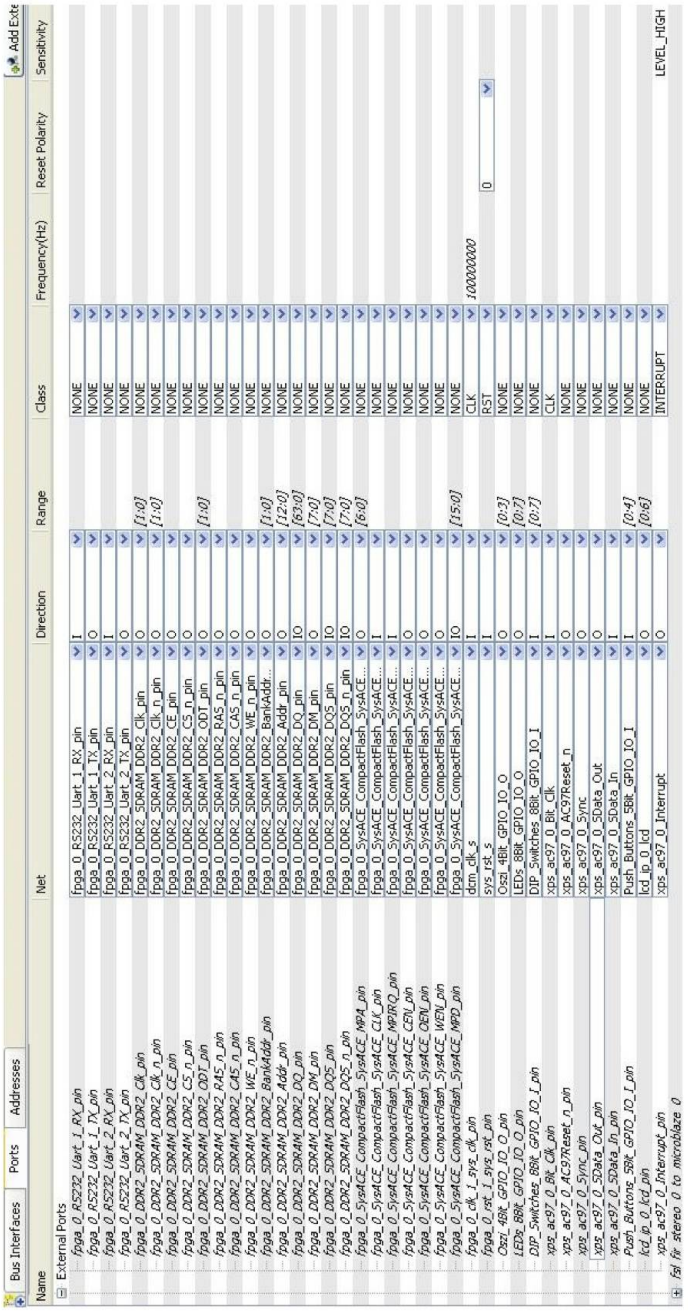

Abbildung 32: External Ports des Embedded System

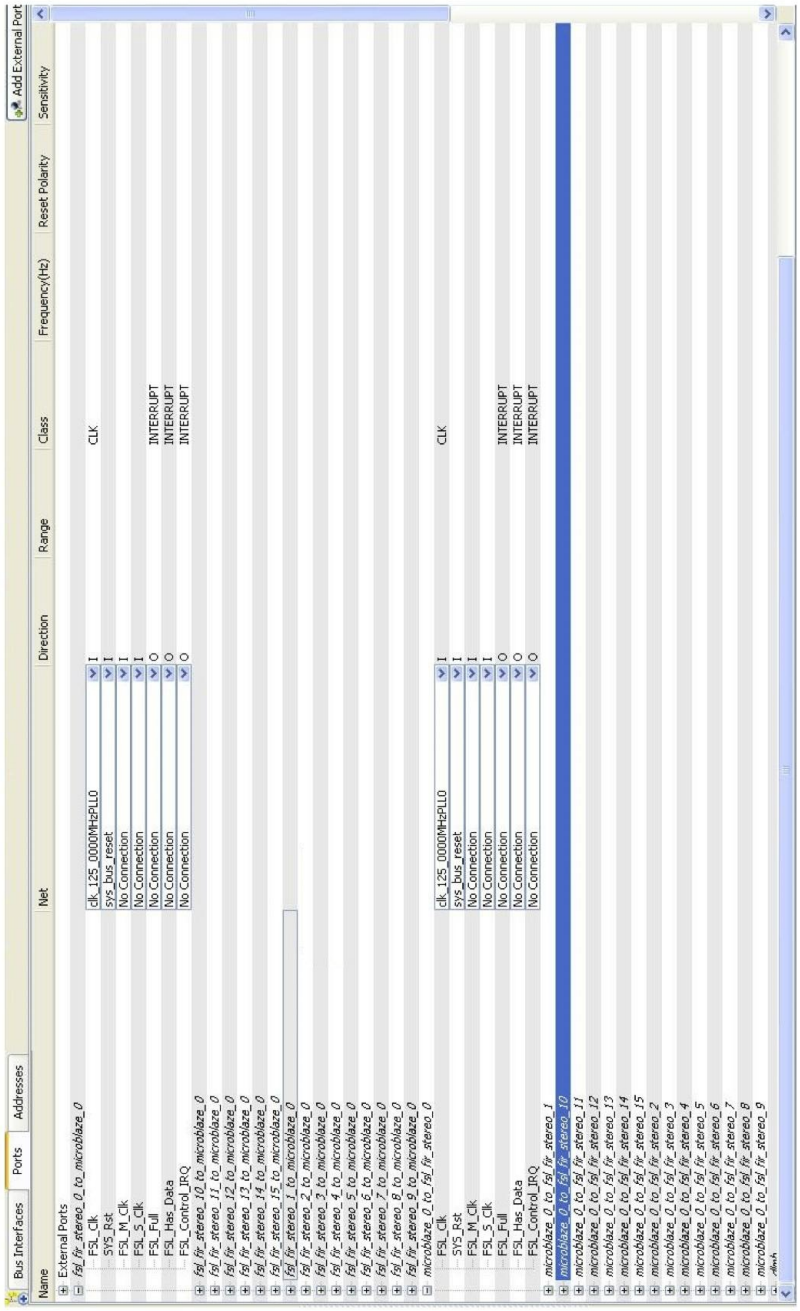

Abbildung 33: Ports (FSL Interfaces) des Embedded System

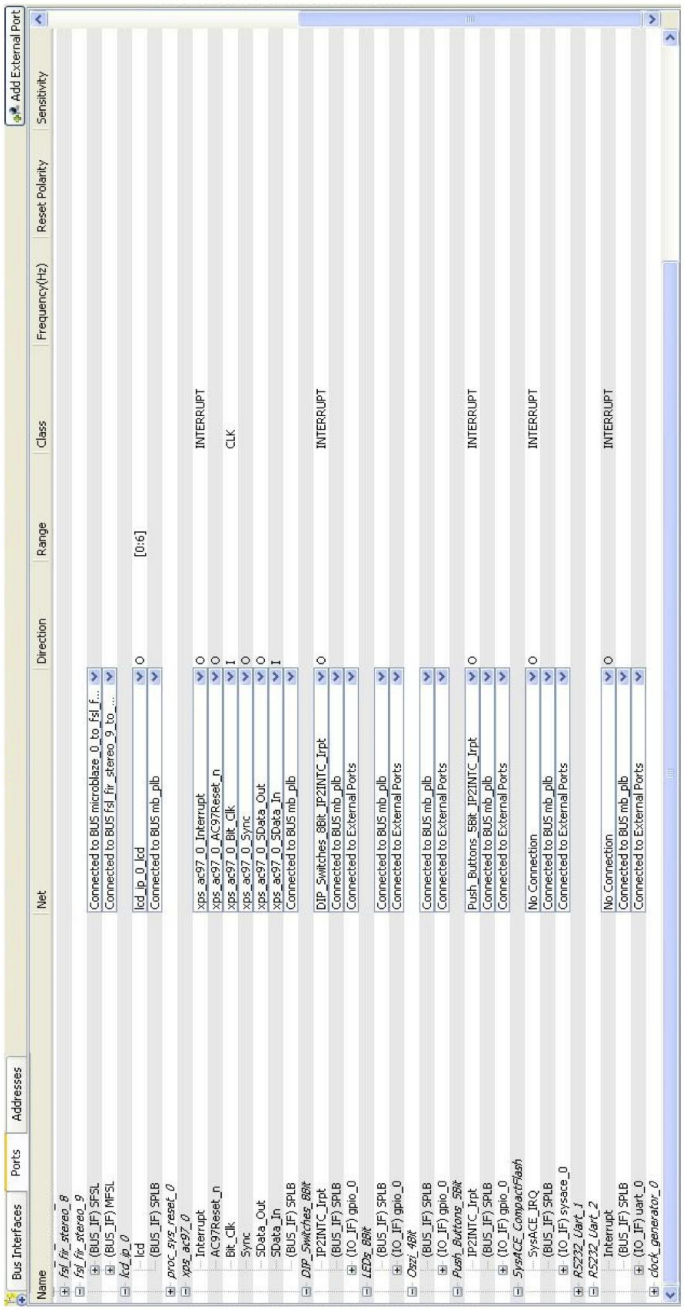

Abbildung 34: Ports des Embedded System

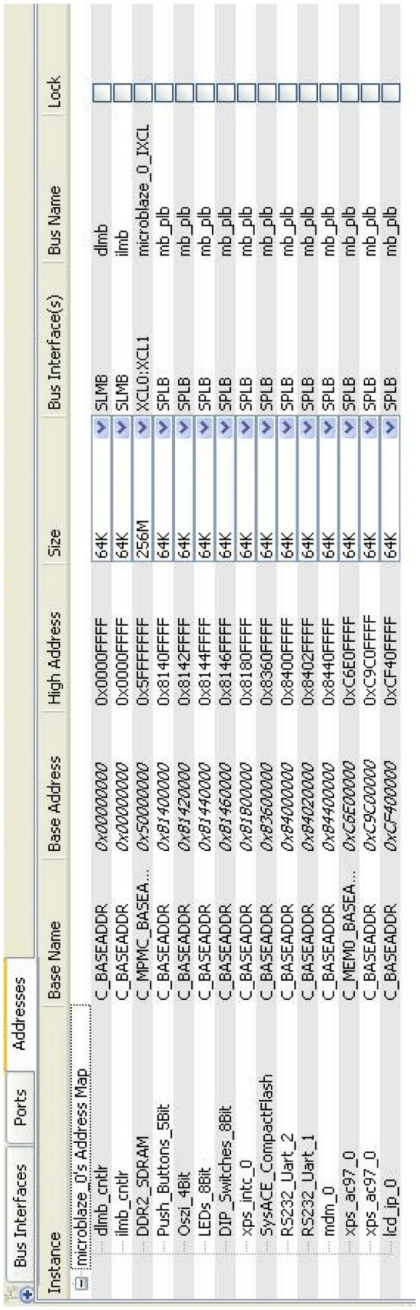

Abbildung 35: Addresses des Embedded System

Auszug UCF-File

```
NET fpga_0_RS232_Uart_1_RX_pin LOC = AG15 | IOSTANDARD=LVCMOS33;
NET fpga_0_RS232_Uart_1_TX_pin LOC = AG20 | IOSTANDARD=LVCMOS33;
NET fpga_0_RS232_Uart_2_RX_pin LOC = G10  | IOSTANDARD=LVCMOS33;
NET fpga_0_RS232_Uart_2_TX_pin LOC = F10  | IOSTANDARD=LVCMOS33;

NET LEDs_8Bit_GPIO_IO_O_pin<0> LOC = AE24 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<1> LOC = AD24 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<2> LOC = AD25 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<3> LOC = G16  | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<4> LOC = AD26 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<5> LOC = G15  | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<6> LOC = L18  | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW;
NET LEDs_8Bit_GPIO_IO_O_pin<7> LOC = H18  | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW;

Net Push_Buttons_5Bit_GPIO_IO_I_pin<0> LOC = AJ6 | IOSTANDARD=LVCMOS33 |
     PULLDOWN | SLEW=SLOW;
Net Push_Buttons_5Bit_GPIO_IO_I_pin<1> LOC = AJ7 | IOSTANDARD=LVCMOS33 |
     PULLDOWN | SLEW=SLOW;
Net Push_Buttons_5Bit_GPIO_IO_I_pin<2> LOC = V8  | IOSTANDARD=LVCMOS33 |
     PULLDOWN | SLEW=SLOW;
Net Push_Buttons_5Bit_GPIO_IO_I_pin<3> LOC = AK7 | IOSTANDARD=LVCMOS33 |
     PULLDOWN | SLEW=SLOW;
Net Push_Buttons_5Bit_GPIO_IO_I_pin<4> LOC = U8  | IOSTANDARD=LVCMOS18 |
     PULLDOWN | SLEW=SLOW;

Net DIP_Switches_8Bit_GPIO_IO_I_pin<0> LOC = U25  | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
Net DIP_Switches_8Bit_GPIO_IO_I_pin<1> LOC = AG27 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
Net DIP_Switches_8Bit_GPIO_IO_I_pin<2> LOC = AF25 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
Net DIP_Switches_8Bit_GPIO_IO_I_pin<3> LOC = AF26 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
Net DIP_Switches_8Bit_GPIO_IO_I_pin<4> LOC = AE27 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
Net DIP_Switches_8Bit_GPIO_IO_I_pin<5> LOC = AE26 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
Net DIP_Switches_8Bit_GPIO_IO_I_pin<6> LOC = AC25 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;
```

```
Net DIP_Switches_8Bit_GPIO_IO_I_pin<7> LOC = AC24 | IOSTANDARD=LVCMOS18
    | PULLDOWN | SLEW=SLOW;

Net lcd_ip_0_lcd_pin<0> LOC = AC9  | IOSTANDARD=LVCMOS33 | TIG |
    PULLDOWN; # LCD_E
Net lcd_ip_0_lcd_pin<1> LOC = J17  | IOSTANDARD=LVCMOS25 | TIG |
    PULLDOWN; # LCD_RS
Net lcd_ip_0_lcd_pin<2> LOC = AC10 | IOSTANDARD=LVCMOS33 | TIG |
    PULLDOWN; # LCD_RW
Net lcd_ip_0_lcd_pin<3> LOC = T11  | IOSTANDARD=LVCMOS33 | TIG |
    PULLDOWN; # LCD_DB7
Net lcd_ip_0_lcd_pin<4> LOC = G6   | IOSTANDARD=LVCMOS33 | TIG |
    PULLDOWN; # LCD_DB6
Net lcd_ip_0_lcd_pin<5> LOC = G7   | IOSTANDARD=LVCMOS33 | TIG |
    PULLDOWN; # LCD_DB5
Net lcd_ip_0_lcd_pin<6> LOC = T9   | IOSTANDARD=LVCMOS33 | TIG |
    PULLDOWN; # LCD_DB4

NET xps_ac97_0_Bit_Clk_pin LOC = AF18 | IOSTANDARD=LVCMOS33 | PERIOD=80;
NET xps_ac97_0_SData_In_pin LOC = AE18 | IOSTANDARD=LVCMOS33;
NET xps_ac97_0_AC97Reset_n_pin LOC = AG17 | IOSTANDARD=LVCMOS33 | TIG;
NET xps_ac97_0_SData_Out_pin LOC = AG16 | IOSTANDARD=LVCMOS33;
NET xps_ac97_0_Sync_pin LOC = AF19 | IOSTANDARD=LVCMOS33;
NET xps_ac97_0_Interrupt_pin LOC = G32 | IOSTANDARD=LVCMOS33;

Net Oszi_4Bit_GPIO_IO_O_pin<0> LOC = H33 | IOSTANDARD=LVCMOS33 |
    PULLDOWN;
Net Oszi_4Bit_GPIO_IO_O_pin<1> LOC = F34 | IOSTANDARD=LVCMOS33 |
    PULLDOWN;
Net Oszi_4Bit_GPIO_IO_O_pin<2> LOC = H34 | IOSTANDARD=LVCMOS33 |
    PULLDOWN;
Net Oszi_4Bit_GPIO_IO_O_pin<3> LOC = G33 | IOSTANDARD=LVCMOS33 |
    PULLDOWN;
```

FPGA Ressourcenverbrauch

Art	verfügbar	verwendet	in %
Number of Slice Registers	44800	9205	20%
Number of Slice LUTs	44800	18762	41%
Number of occupied Slices	11200	7327	65%
Number of LUT Flip Flop pairs used		23071	
Number of bonded IOBs	640	185	28%
Number of BlockRAM/FIFO	148	47	32%
Number of BUFG/BUFGCTRLs	32	7	21%
Number of IDELAYCTRLs	22	3	21%
Number of BSCANs	4	1	25%
Number of BUFIOs	80	8	10%
Number of DSP48Es	128	35	27%
Number of PLL_ADVs	6	1	16%

Tabelle 10: FPGA Ressourcenverbrauch

C Software

Benutzerinteraktion

Push-Button	Funktion
Push-Button Oben	Evaluationwinkel - 5
Push-Button Rechts	Azimuthwinkel - 10
Push-Button Unten	Evaluationwinkel + 5
Push-Button Links	Azimuthwinkel + 10
Push-Button Mitte	Position reset

Tabelle 11: Push-Buttons und dessen Bedeutung für die SoC-Plattform

Dip	High	Low
1	HRTF-Filterung aktiv	LoopBack
2	Kemar-Filtersatz	Sima Filtersatz
3	LTI und LDI getrennt verarbeitet	In einem Filter verarbeitet
4	bilineare Interpolation	Interpolation horizontal
5	Button-Steuerung	Kompass-Steuerung
6	/	/
7	/	/
8	Kopfhöerkompensation aktiviert	Kopfhöerkompensation deaktiviert

Tabelle 12: Dip-Switch und dessen Bedeutung für die SoC-Plattform

Zeitbedarf eines Main-Routine-Durchlaufs

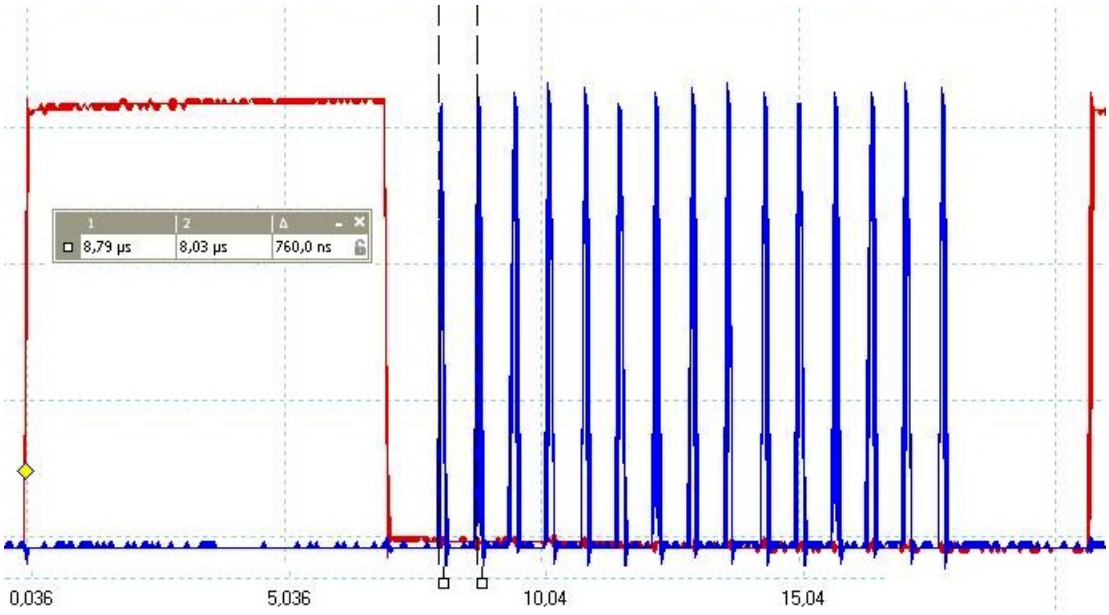

Abbildung 36: Zeitbedarf eines Main-Routine-Durchlaufs mit Buttons

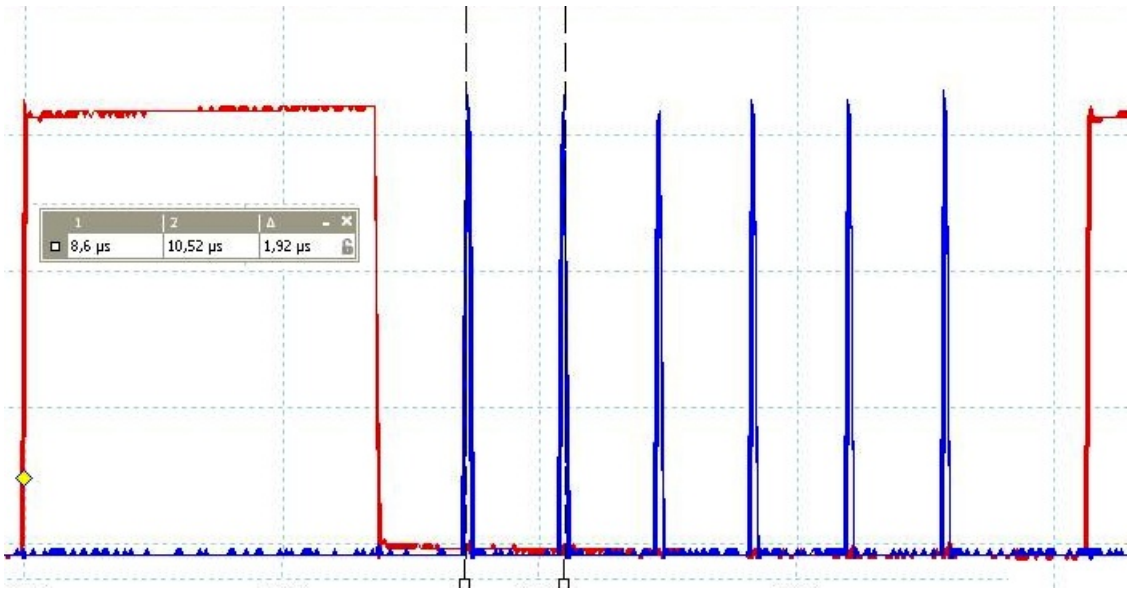

Abbildung 37: Zeitbedarf eines Main-Routine-Durchlaufs mit aktiven Kompass

Audio-ISR Messungen

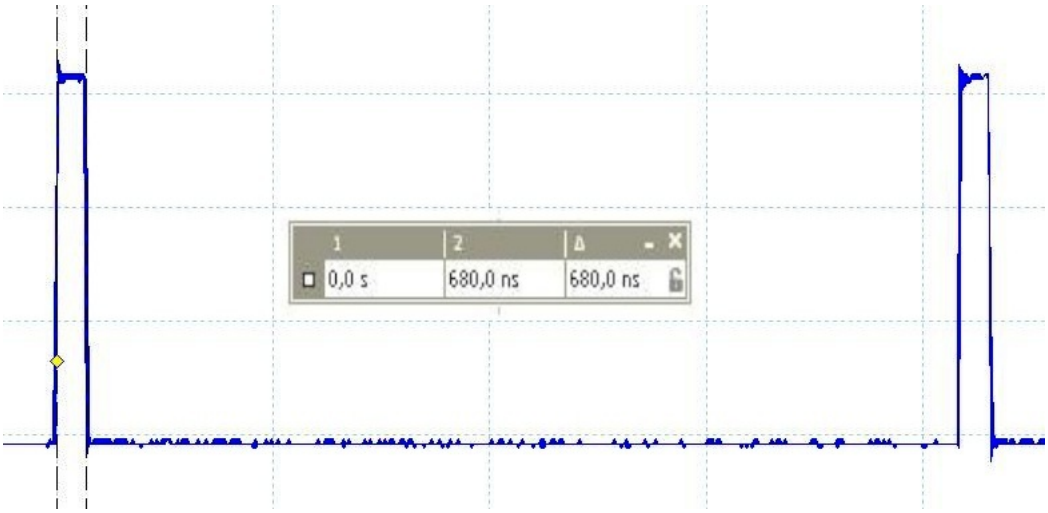

Abbildung 38: Zeitbedarf der AC97-ISR bei Loopback

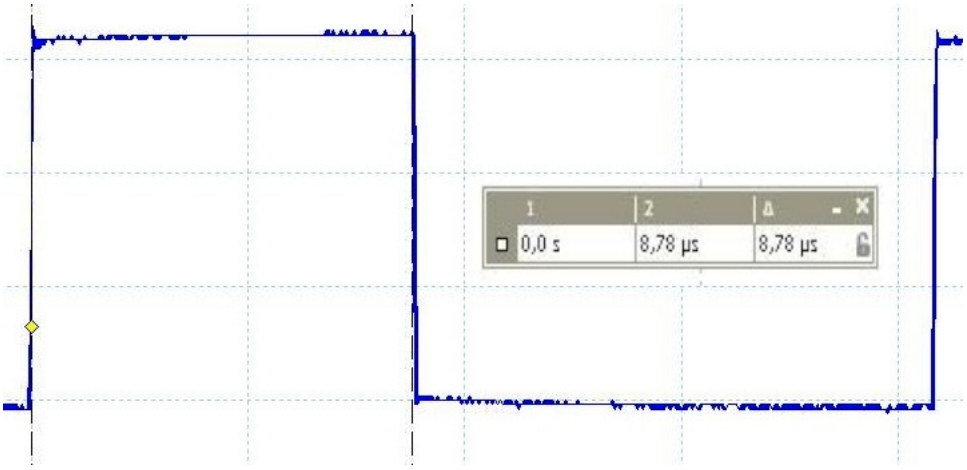

Abbildung 39: Zeitbedarf der AC97-ISR bei Crossfading mit Laufzeitglied und Steuerung über Buttons

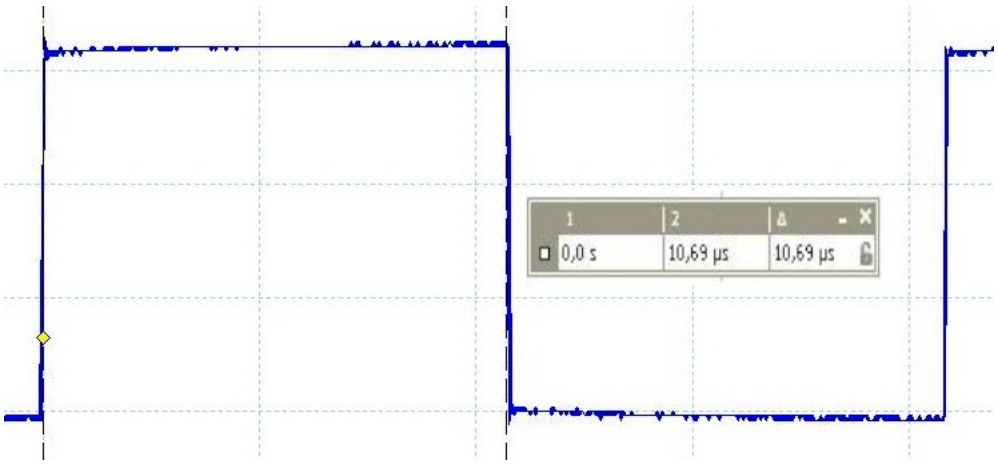

Abbildung 40: Zeitbedarf der AC97-ISR bei Crossfading mit Laufzeitglied und Steuerung über Kompass

Zeitmessung einer FIR-Filteraktualisierung

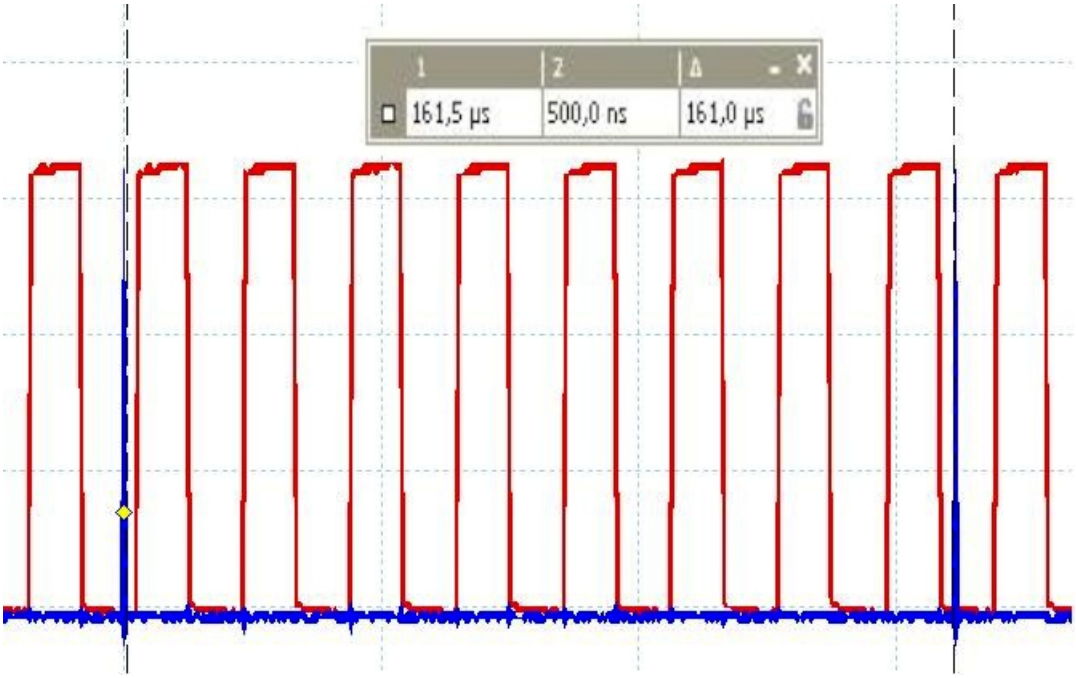

Abbildung 41: Zeitbedarf einer FIR-Stereo-Filter-Aktualisierung über die FSL-Schnittsellte im laufenden Betrieb

Impulsantworten der Sima-HRTF mit getrennter ILD und ITD Filterung aus der SoC-Plattform

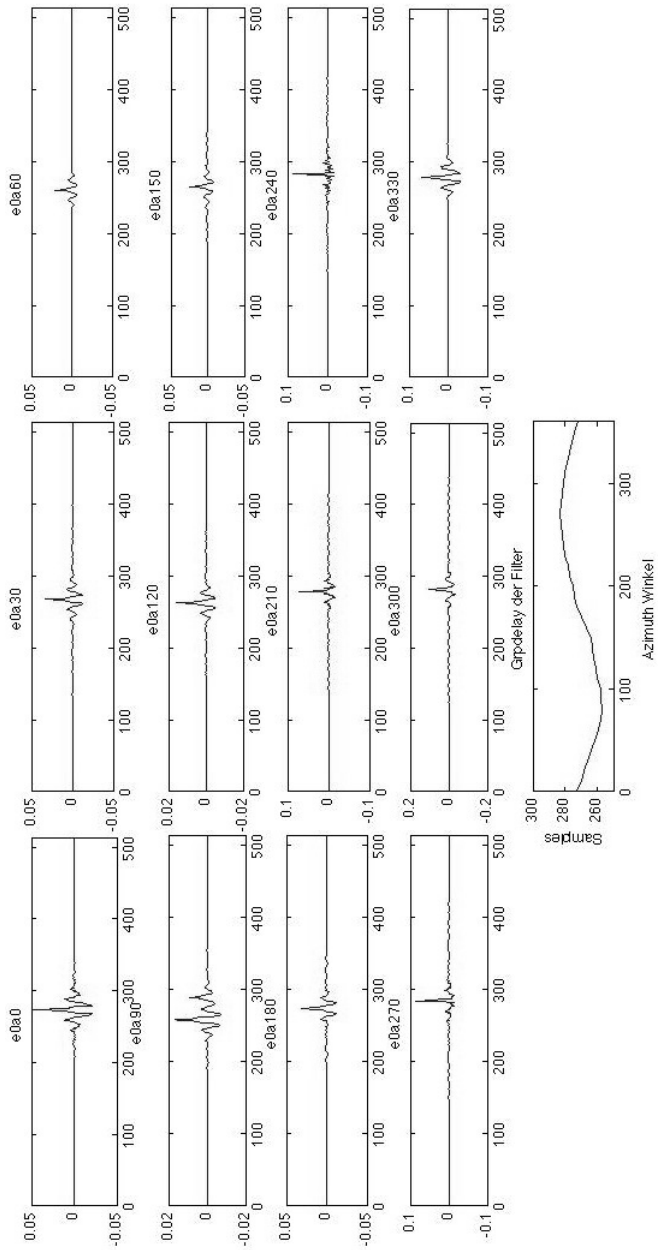

Abbildung 42: Impulsantworten der Sima-HRTF mit getrennter ILD und ITD Filterung aus der SoC-Plattform

D Inhalt der CD-Rom

- Bachelorarbeit “Eine SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern für positionsveränderliche virtuelle Schallquellen.“ (PDF)
- HRTF-Messungen von Sylvia Sima
- Kemar-Kunstkopf HRTF-Messungen des MIT
- MATLAB-Skripte zur Berechnung der Filterkoeffizienten
- Kopie des Dateisysteminhaltes der Compact-Flash-Card
- XPS-Projekt der SoC-Plattform
- SDK-Projekt der entwickelten Software
- Impulsantworten aller HRTF-Filter in 10° Schritten um dem Kopf
- Xilinx Dokumente und Manuals für das ML507 Evaluation-Board
- Diverse Dokumente (Artikel, Manuals, Application Notes, White Papers,. . .) zum behandelten Themengebiet

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 9. September 2011	
Ort, Datum	Unterschrift