Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

David Hemmer

Eine kopfhörerbasierte, virtuelle Audioumgebung zur räumlichen Positionierung mehrerer Schallquellen mit einer mobilen SoC-Plattform

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

David Hemmer

Eine kopfhörerbasierte, virtuelle Audioumgebung zur räumlichen Positionierung mehrerer Schallquellen mit einer mobilen SoC-Plattform

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master 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 25. November 2013

Thema der Masterarbeit

Eine kopfhörerbasierte, virtuelle Audioumgebung zur räumlichen Positionierung mehrerer Schallquellen mit einer mobilen SoC-Plattform

Stichworte

Audio, digitale Signalverarbeitung, Embedded System, System-on-Chip, HW/SW Co-Design, FPGA, FIR-Filter, Binaural, 3D Sound, räumliches Hören, HRTF, kopfbezogene Übertragungsfunktionen, Echtzeit, virtuelle Klangwelt, Matlab/Simulink, Xilinx SG

Kurzzusammenfassung

Diese Masterarbeit befasst sich mit der Entwicklung einer mobilen SoC-Plattform zum Erzeugen einer virtuellen Klangwelt. Die virtuellen Schallquellen werden über kopfbezogene Übertragungsfunktionen (HRTF) in Echtzeit berechnet. Das System kann bis zu 8 virtuelle Schallquellen simulieren. Durch die Feststellung der Kopfposition des Zuhörers relativ zur Position der virtuellen Schallquellen mithilfe eines 3-Achsen-Kompasssensors wird die Audiofilterung gesteuert. Zur Steuerung des SoC wurde eine Serveranwendung entwickelt. Diese stellt alle benötigten Informationen zur Verfügung und verschickt diese an das SoC. Auf diese Weise lässt sich ein beliebiges Audiosignal in einem virtuellen, dreidimensionalen Raum um den Zuhörer positionieren und bewegen.

Der Embedded-System-Enwicklungsprozess gliedert sich in die Erstellung des Hardwarebeschleunigers für die Audiofilterung mithilfe des Xilinx Systemgenerators auf Basis der Matlab/Simulink-Entwicklungsumgebung. Zur Optimierung der HRTF-Filterung werden die Interaurale Laufzeitdifferenz und die Interaurale Pegeldifferenz getrennt und separat voneinander verarbeitet. Dieser so entwickelte Hardwarebeschleuniger wurde in die entwickelte System-on-Chip-Plattform integriert. Das SoC verfügt über eine kabellose Schnittstelle, um mit dem Server zu kommunizieren. Über diese Schnittstelle werden alle Informationen (Position von Schallquellen, Audiostreams) ausgetauscht.

Der Server verfügt über eine einfache Szenensteuerung zur Positionierung von Schallquellen. Die Audioquellen werden durch eine Schnittstelle aufgenommen und an das SoC verschickt. Für das Audiostreaming wurde ein eigenes Protokoll entwickelt. Der weitere Informationsaustausch zwischen dem SoC und Server geschieht über OpenSoundControl-Nachrichten.

Das entwickelte System unterstützt die Echtzeit-Audiofilterung von bis zu acht Audioquellen. Ebenso unterstützt es die Echtzeit-Positionserhaltung der virtuellen Schallquellen für die vollen 360° der horizontalen Kopfdrehung sowie Kopfneigungen im Bereich von -41° bis + 37°.

Title of the paper

A headphone-based virtual audio environment for spatial positioning of multiple sound sources using a mobile SoC platform

Keywords

Audio, digital signal processing, embedded systems, system-on-chip, HW / SW co-design, FPGA, FIR filter, binaural, 3D sound, spatial hearing, HRTF, head-related transfer functions, real-time, virtual world of sound, Matlab / Simulink, Xilinx SG

Abstract

This thesis deals with the development of an mobile SoC platform for creating a virtual soundscape. The virtual sound sources are calculated using head-related transfer function (HRTF) in real time. The system is able to simulate up to 8 virtual sound sources. By identifying the head position of the listener relative to the position of the virtual sound sources using a 3-axis compass sensor the audiofiltering is controlled. To control the SoC a server application was developed. This provides all the information needed and sends it to the SoC. This way any audio signal within a virtual three-dimensional space around the listener can be positioned and moved.

The embedded system development process is divided into the creation of the hardware accelerator for audio filtering using the Xilinx System generator based on Matlab / Simulink. To optimize the HRTF filtering the interaural time difference and interaural level difference are separated and processed separately from each other. This hardware accelerator has been integrated in the system-on-chip platform. The SoC includes a wireless interface to communicate with the server. Via this interface all informations (position of sound sources, audio streams) will be exchanged.

The server has an simple scene control for positioning the sound sources. The audio sources are received by an interface and sent to the SoC. For the audio streaming an own protocol was developed.The leftover information exchange occurs via OSC messages between the SoC and the server.

The developed system supports real-time audio filtering of up to eight audio sources. It also supports real-time position maintenance of the virtual sound sources for the full 360° of the horizontal plane and a vertical head movement of -41° to +37°.

Inhaltsverzeichnis

1 Einleitung

Unsere Lieblingsmusik begleitet uns dank Smartphone und MP3-Player ständig. Ob auf dem Weg zur Arbeit oder beim Sport, viele Menschen hören ihre Musik mobil über Kopfhörer. Dieses gehört zu unserem Leben wie das Smartphone, welches uns alle Informationen aus dem Internet überall und mobil zur Verfügung stellt.

Der Fortschritt ist nicht aufzuhalten. Das Smartphone entwickelt sich zu einem erweiterten und virtuellen Begleitgerät. Das aktuellste Beispiel ist Google Glass, welches im Juli 2012 vorgestellt wurde und voraussichtlich im 4. Quartal 2013 auf dem Markt erhältlich ist [Goo]. Google Glass ist ein Miniatur-Computer, der auf einem Brillenrahmen montiert ist und über ein Head-Up-Display im Sichtfeld des Trägers Informationen einblendet. Somit werden alle nur erdenklichen Informationen in die reale Welt eingebunden und es entsteht eine noch nicht da gewesene erweiterte Realität. Die Brille wird über Sprache gesteuert, welche ein einfaches Interaktionssystem zwischen Mensch und Maschine darstellt. Durch die Brille wird der Benutzer nicht in der Mobilität eingeschränkt und so verschmelzen diese sehr gut mit der realen Umwelt.

Ein Nachteil dieses Systems ist die Einschränkung auf das visuelle Sinnesorgan. Ein weiteres Sinnesorgan des Menschen ist das Gehör, welches nur minimal mit berücksichtigt wird. Dabei ist das menschliche Gehör ganz automatisch bei jeder Wahrnehmung beteiligt. Es erlaubt uns viele Informationen über unsere Umgebung unterbewusst wahrzunehmen. Dazu zählen Informationen des Raumes, in dem wir uns befinden, anhand der Raumakustik. Ebenso bestimmen wir über unser Gehör die Position und Entfernung zu Schallquellen. Durch eine Erweiterung der aktuellen virtuellen und erweiternden Realität um das Gehör würde ein besserer Informationsfluss entstehen. Nicht alle Informationen müssten visuell sondern könnten auch akustisch eingebunden werden.

Ein entsprechendes Augmented Reality Audio (ARA) System muss die Realität sehr genau nachbilden. Das bedeutet, dass ein solches System die virtuellen Schallquellen in Echtzeit berechnet und in die Realität einbindet. In Bezug auf virtuelle und erweiterte Realitätssysteme darf ein solches System die Mobilität des Benutzers nicht beeinträchtigen. Durch ein ARA-System wird die reale oder virtuelle Umwelt mit Informationen angereichert. Ebenso wäre aber auch ein neuartiges Musikerlebnis denkbar, indem die Musikinstrumente beziehungsweise Schallquellen räumlich eingebunden werden und somit ein Konzerteindruck entsteht. Daraus folgt eine Vielzahl von neuen Einsatzgebieten für uns Menschen. Zum einen lassen sich bestehende Systeme wie Google Glass erweitern. Zum anderen ist ein solches System auch ein neuartiges Auditory Display, welches eine Vielzahl von neuen Anwendungsbereichen erschließt.

Es gibt verschiedene Ansätze um eine solche Audioumgebung zu erzeugen. Zum einen den schallquellenbasierten Ansatz. Für jede Schallquelle werden die Berechnungen zur

Simulation der Schallquelle durchgeführt. Um mehrere dieser Schallquellen zu simulieren, sind für jede Schallquelle die Berechnungen durchzuführen. Um das Gesamtergebnis zu erhalten, ist es notwendig alle Ergebnisse zu vereinen und abzuspielen.

Ein anderer Ansatz ist es, Surround-Setups als Grundlage zu nehmen. Mit diesen lässt sich eine Berechnung durchführen, um einen räumlichen Eindruck zu erzeugen. Auf Basis dieses Verfahrens sind zuletzt mehrere Produkte vorgestellt worden. Zum einen HEADPHONE SURROUND 3D von der Firma New Audio Technology [Tec]. Die aktuelle Virtualisationstechnologie der Firma lässt den Hörer über Kopfhörer jedes beliebige Lautsprechersystem von Stereo bis Surround erleben. Die Produkte sind zunächst für Inhalts-Hersteller wie Komponisten, Audioproduzenten und Filmsound-Designer vorgesehen. Zum anderen stellte kürzlich das Fraunhofer IIS "Cingo"vor [IIS]. Mit Cingo können Tablets, Smartphones oder andere mobile Geräte überzeugenden Surround-Sound über die eingebauten Lautsprecher oder über angeschlossene Kopfhörer wiedergeben. Für optimale Klangergebnisse kombiniert Cingo mehrere Technologien. Diese beinhalten den virtuellen Rundumklang, einen Lautstärkeoptimierer und eine Klangregelung, die dem Gerät und den Kopfhörern angepasst werden. Damit wird sowohl Surround- als auch Stereo-Material überzeugend räumlich wiedergegeben. Das Fraunhofer IIS vertreibt Cingo als fertige Software-Implementierung für Hersteller von mobilen Geräten, für Chipsatz-Hersteller und für Anbieter von Multimedia-Diensten.

Diese Arbeit befasst sich mit der Umsetzung eines schallquellenbasierten Ansatzes. Dabei werden die Berechnungen in Echtzeit auf einem Embedded-System vorgenommen und über normale Stereo-Kopfhörer abgespielt. In einem definierten dreidimensionalen Raum rund um den Zuhörer lassen sich mehrere Schallquellen frei positionieren. Um den Augmented Reality Audio Ansatz zu vervollständigen, wird auch die Bewegung des Hörers mit einbezogen. Zur Verifikation des Mobile Augmented Reality Audio Systems (MARA) wird am Ende der Entwicklung ein Hörtest mit mehreren Testpersonen durchgeführt.

2 Grundlagen

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

2.1 Binaurales Hören

Das menschliche Gehör ist eines der wichtigsten Sinnesorgane. 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 ebenfalls vom Schalldruckpegel abhängig ist. Die Wahrnehmung von Sound ist von Mensch zu Mensch unterschiedlich und ist maßgeblich vom Alter der Person abhängig.

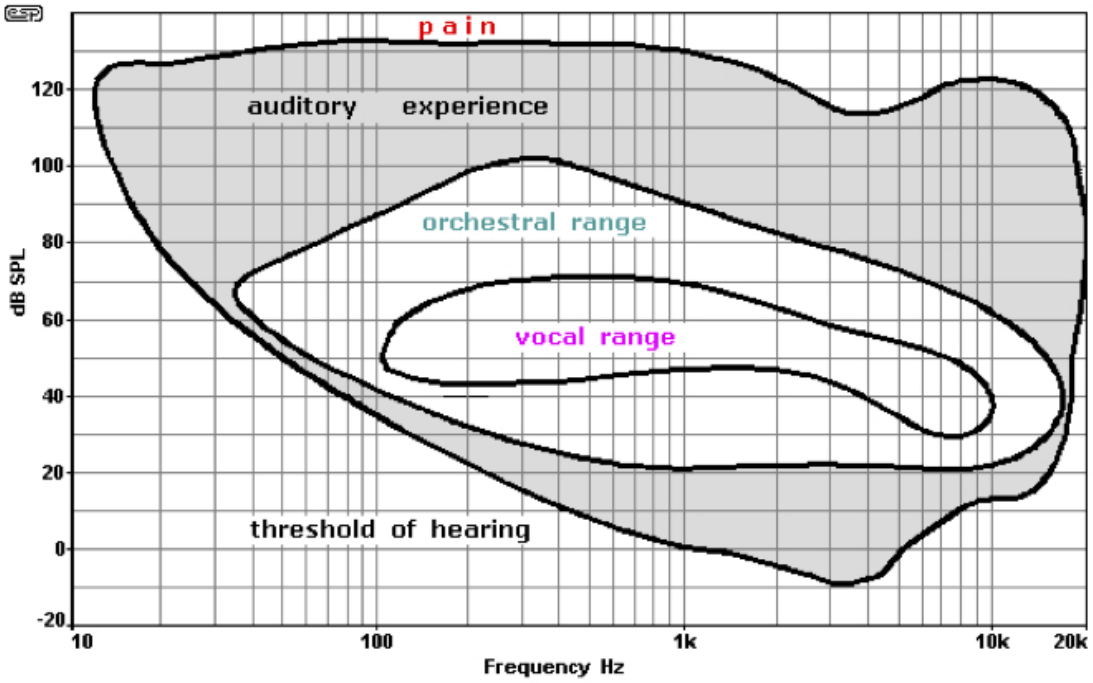


Abbildung 1: Hörbereich des Menschen Quelle: [Ell06]

Um Schall mit einer bestimmten Frequenz wahrzunehmen, muss diese einen Mindestschalldruckpegel aufweisen. Dieser ist bei 100 dB bei etwa 16 Hz und geht herunter bis zu -10 dB bei 3 kHz [1]. Der grau eingefasste Bereich in der Abbildung 1 stellt den Bereich dar,

[1] die Frequenz, bei der das Gehör am empfindlichsten ist

in dem das menschliche Gehör Schallwellen wahrnimmt. Alles was oberhalb des grauen Bereiches liegt, 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. Dieses ist die räumliche Wahrnehmung, aus welcher Richtung der Schall kommt und ist somit eines der wichtigsten Orientierungsinstrumente des Menschen. Um die Richtung einer Schallquelle zu orten, spielt die physikalische Struktur der beider Ohren in Kombination mit dem Gehirn eine große Rolle.

Durch die Anordnung beider Ohren zueinander entsteht eine Laufzeitdifferenz. Diese Laufzeitdifferenz 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. einer 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 dieses 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. Durch die interaurale Pegeldifferenz ist das menschliche Gehör in der Lage, eine Veränderung von bis zu 3° von der Kopfmitte zu erkennen.

Die interaurale Laufzeitdifferenz und interaurale Pegeldifferenz sind 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 nicht festzustellen, ob eine Schallquelle von vorne oder hinten auf den Kopf gerichtet ist. Um die Schallquelle dennoch zu lokalisieren, bewegt der Mensch seinen Kopf.

2.2 Head-Related Transfer Function

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 signalaussendenden 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 die Intensitäts- und Laufzeitunterschiede zu erzeugen.

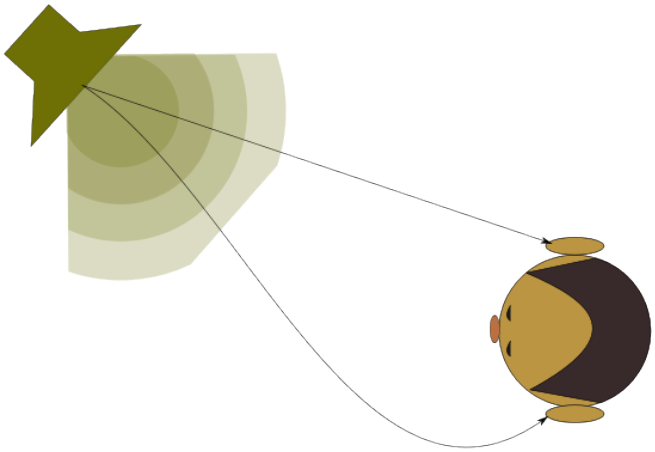


Abbildung 2: Head-Related Transfer Function Quelle: [Wik11]

Für die Messung der HRTF wird in einem reflektionsarmen Raum ein Testsignal auf einen 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 aufgenommenen Signale entsprechen annähernd den Schall-Signalen, welche wir Menschen von diesem Testsignal hören würden. Durch Dekonvolution (inverse Faltung) der aufgezeichneten Signale mit den Testsignalen im Frequenzbereich ergibt sich 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 sind direkt als Koeffizientensätze für eine FIR-Filterung 2.3 eines Audiosignals verwendbar. Die Schallquelle eines beliebigen auf diese Weise gefilterten Audiosignals ortet der Hörer an der Position, für die die Messung der HRIRs vorgenommen wurde. [Zöl02]

2.3 FIR-Filter

Um HRTF nachzubilden werden digitale Filter benötigt. 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. 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 Ausgangssignal 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 ein maximal positiver Puls zugeführt wird. Das Ausgangssignal zeigt, ob Überlaufe in der internen Addition der Multiplikationen auftreten. Die Sprungantwort ist das zeitliche Integral der Impulsantwort. 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]

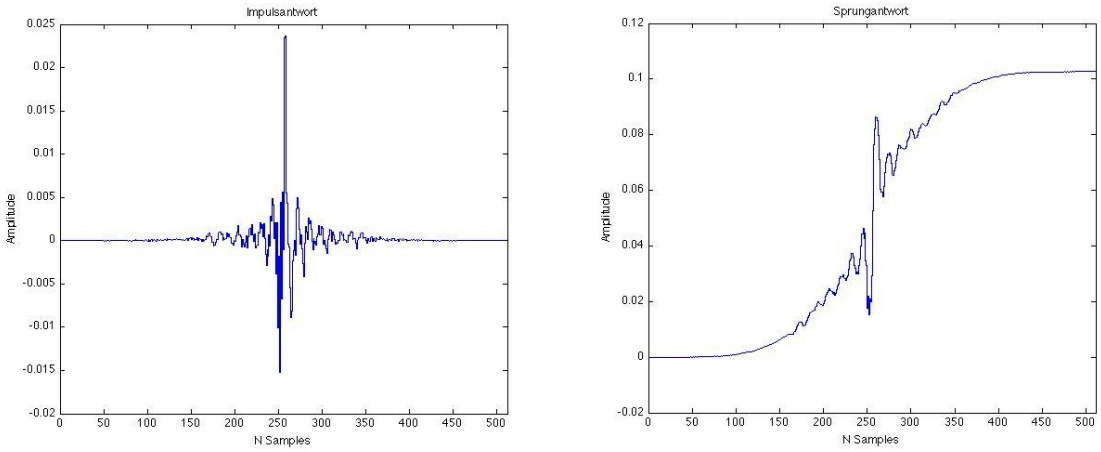


Abbildung 3: Impuls- und Sprungantwort eines FIR-Filters

2.3.1 Normierung von Filterkoeffizienten

Mit der Filterskalierung oder auch unter dem Begriff Normierung der Koeffizienten werden Überlaufeffekte bei der Addition der Produkte verhindert. Bei einem Überlauf im positiven Wertebereich ändert sich 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 begrenztem 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 werden keine Überläufe bei der Addition der Produkte verhindert. 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 Vergleichbare Arbeiten

In diesem Kapitel werden mehrere vergleichbare Arbeiten aus dem Bereich der mobilen Augmented Reality Audio vorgestellt. Dabei gliedern sich die vorgestellten Arbeiten in zwei Bereiche. Zu einem in Arbeiten mit theoretischen Ansätzen und zum anderen in Arbeiten, in denen vergleichbare Mobile Augmented Reality Audio Systeme vorgestellt werden.

3.1 Virtual acoustics and 3-D Sound in Multimedia signal processing

Diese Ausarbeitung ist die Dissertation von Jyri Huopaniemi von der Helsinki University of Technology vom 5. November 1999 [Huo99]. Schwerpunkt seiner Dissertation sind die Aspekte von der Echtzeit Modellierung und Synthese für 3D-Sound in Bezug auf digitales Audio, Multimedia und virtuelle Umgebungen.

In der Dissertation werden mehrere Ansätze für Augmented Reality Audio Systeme vorgestellt. Zwei dieser Vorschläge sind für Mobile ARA Systeme einsetzbar. Diese beiden Ansätze sind in Abbildung 4 dargestellt.

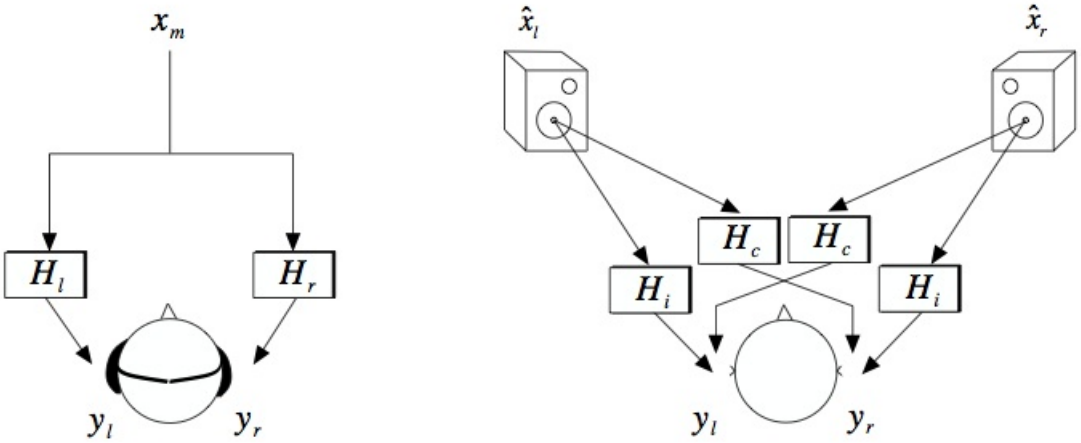


Abbildung 4: Signalflussgraph: links: binaural und rechts: crosstalk canceled binaural audio processing

Der binaurale Ansatz, auch Headphone Reproduction (links) genannt, arbeitet über Kopfhörer. Das Mono-Audiosignal x_m wird über die HRTF-Filter H_l und H_r der virtuellen Audioquelle gefiltert und über die entsprechende Kopfhörerseite abgespielt. Dieses Verfahren berücksichtigt die Akustik des Raumes, in dem sich der Zuhörer befindet, nicht. Ebenso wird die Kopfposition des Zuhörers nicht mit einbezogen. Mittels Head-Tracking wären Kopfbewegungen messbar und könnten somit in die Audiofilterung einbezogen werden.

Der zweite Ansatz ist die Loudspeaker Reproduction (rechts). Die crosstalk cancled binauralen Signale x_l und x_r werden über die Lautsprecher ausgestrahlt, um die Wunsch-Signale y_l und y_r zu erhalten. Die richtungsabhängigen Lautsprecher-zu-Ohr-Übertragungsfunktionen H_i und H_c sind zu beachtet, um den Effekt von virtuellen Schallquellen zu erfüllen. Die Filterung ist als kaskadierter Prozess zu verstehen. Zum einen die HRTF-Filterung und zum anderen die separate crosstalk canceling Filterung. Bei diesen Verfahren gibt es zwei wesentliche Grenzen: Die Zuhörerposition und die Raumposition des Zuhörers.

Die Dissertation von Jyri Huopaniemi gibt eine gute Übersicht vom Aufgabenfeld des Augmented Reality Audio. Des Weiteren zeigt er in seiner Arbeit, dass eine Vielzahl der erforderlichen Audio-Filterungen auf Digital Signal Processors (DSPs) berechenbar sind.

3.2 Augmented Reality Audio for Mobile and Wearable Appliances

Dieses Paper wurde in Zusammenarbeit von der Helsinki University of Technologie und dem Nokia Research Center (Media Technologies Lab) im Journal der Audio Engineering Society in der Ausgabe 52 (JAES-52) am 6. August 2003 vorgestellt [HJT+04]. Dabei unterteilt es sich in zwei verschiedene Bereiche. Im ersten Bereich werden die Anforderungen an ein mobiles Augmented Reality Audio (MARA) System festgelegt. Im anschließenden Bereich wird ein erstes Augmented Reality Audio System vorgestellt.

Zuerst wird das Konzept der Augmented Reality Audio Technik charakterisiert. Es handelt sich dabei um die Erweiterung der realen Umwelt mit virtuellen auditiven Umgebungen und Kommunikations-Szenarien. Dabei werden verschiedene Anforderungen an ein Augmented Reality Audio System festgestellt. Diese gelten ebenso für ein mobiles Augmented Reality System. Es werden folgende Punkte aufgeführt:

- Die virtuelle Audioumgebung verbindet sich mit der Realität
- Mobilität und Benutzerfreundlichkeit des Systems
- Echtzeitfähigkeit
- Raumklang

Mit dem ersten Punkt: “Die virtuelle Audioumgebung verbindet sich mit der Realität“ ist der Augmented Reality Ansatz gemeint. Ein ARA-System muss sicherstellen, dass der Zuhörer gleichzeitig die reale Umwelt und die virtuell erzeugte Umwelt wahrnimmt. In diesem Teilbereich von ARA gibt es viele verschiedene Ansätze. Der einfachste und zurzeit noch am meisten genutzte Ansatz sind die sogenannten Augmented Reality Audio Kopfhörer mit integrierten Mikrofonen und entsprechendem Audio-Mixer. Dabei wird die reale Audioumgebung mittels der Mikrofone aufgenommen und mit den virtuell erzeugten

Audiosignalen im AR-Mixer zusammengeführt und über die Kopfhörer abgespielt.

Ein weiterer wichtiger Punkt ist die Mobilität und Benutzerfreundlichkeit eines solchen ARA-Systems. Der Benutzer eines solchen Systems darf keinerlei Einschränkungen in Bezug auf die Bewegungsfreiheit und Mobilität haben. Ebenso ist eine intuitive Bedienung sehr wichtig. Im besten Fall ist dem Benutzer des Systems gar nicht bewusst, dass er ein ARA-System nutzt.

Die Echtzeitfähigkeit des Systems ist ebenso ein wichtiger Punkt. In jedem Fall ist sicherzustellen, dass das System auf jede Aktion sofort reagiert. Wenn der Benutzer sich zum Beispiel bewegt, ändert sich auch gleichzeitig die relative Position zu den virtuellen Audioquellen. Wenn die Positionsänderung nicht sofort wahrgenommen und die Audiofilterung nicht angepasst wird, erzeugt dieses Effekte, sodass die Illusion der virtuellen Schallquellen nicht mehr funktioniert. Hierbei gibt es zwei wichtige Faktoren, die zu berücksichtigen sind. Zum einen sind die Bewegungen des Benutzers umgehend und mit einer hohen Genauigkeit aufzunehmen. Zum anderen ist sicherzustellen, dass das System sofort auf diese Änderungen reagiert.

Der letzte Punkt, der aufgeführt wird, ist der Raumklang. Dabei gibt es zwei unterschiedliche Punkte, die zu beachten sind. Zum einen ist es wichtig, dass die virtuellen Audioquellen sich der Umgebung angepassen. Befindet sich zum Beispiel der Benutzer in einer großen Halle und die virtuelle Audioquelle weist nur die Eigenschaften eines kleinen Raumes auf, führt diese beim Benutzer zu einer nicht realen Einschätzung und die Audio-Quelle wird nicht als real wahrgenommen. Zum anderen ist die Position des Benutzers im Raum, um eine real wirkende virtuelle Audioumgebung zu erzeugen, wichtig.

Im zweiten Teil des Papers wurde ein Vergleich von realen und virtuellen Schallquellen gemacht. Der Aufbau des Vergleiches wird in Abbildung 5 dargestellt.

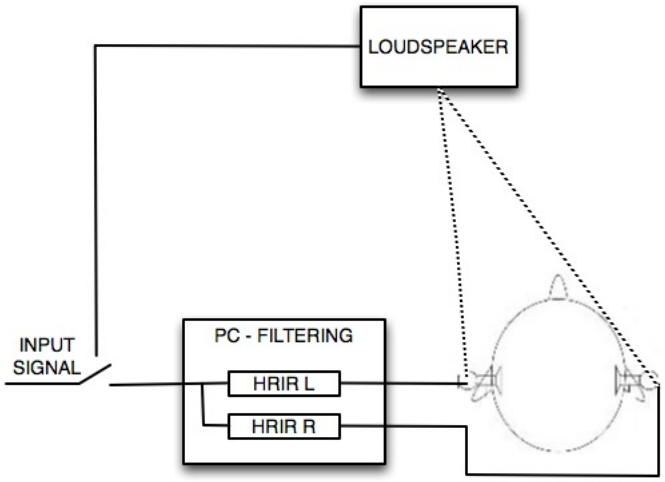


Abbildung 5: Zuhörer Test Aufbau. Vergleich virtueller Audioquellen und realer Audioquellen über AR-Audio Kopfhörer

Die Testpersonen nutzten Kopfhörer mit integrierten Mikrofonen und einem festen ARA-Mixer. Die Head Related Impulse Response (kopfbezogene Impulsantwort) wurde zwischen der Lautsprecherposition und der Kopfposition gemessen. Für den Vergleich wurden die Signale entweder über die Lautsprecher oder über die gemessenen HIRIs in einem PC gefiltert. Zu erwähnen ist, dass die Kopfposition nicht mit berücksichtigt wurde. Somit sind Änderungen der Kopfposition nur für die Lautsprecheridentifikation richtig, was zu einer Verfälschung der Testergebnisse führt. Der Vergleichstest zwischen realer Schallquelle und virtueller Schallquelle zeigt, dass für die meisten Zuhörer kein Unterscheid zwischen beiden Schallquellen hörbar war. Dadurch wurde theoretisch gezeigt, dass kein hörbarer Unterschied zwischen realen und virtuellen Schallquellen existiert.

3.3 Augmented Reality Audio Application in Outdoor Use

Das von Mikko Peltola in seiner Masterarbeit [Pel09] (23.02.2009) an der Helsinki University of Technology entwickelte mobile Augmented Reality Audio System ist für den Einsatz außerhalb von Gebäuden gedacht. Das System ist in Abbildung 6 dargestellt. Die Audioverarbeitung sowie die Koordinationsaufgaben werden auf einem handelsüblichen Laptop im Rucksack berechnet. Die Personenbestimmung arbeitet über GPS und mittels eines Head-Trackers, welcher auf einer Cappy befestigt ist. Die Benutzerinteraktion mit dem System erfolgt mittels einer Wii-Remote Fernbedienung.

Abbildung 6: Entwickeltes MARA-System

Der Augmented Reality-Ansatz wurde in dieser Arbeit vernachlässigt. Er verweist für dieses Themenfeld auf die zur gleichen Zeit geschriebene Dissertation “Development and evaluation of Augmented Reality Audio Systems“ von Mikka Tikander. [Tik09].

In seiner Arbeit stellt er verschiedene Ansätze vor, um Zusatzinformationen mit Audiosignalen zu verbinden. Solche Zusatzinformationen sind zum Beispiel die Positionen einer sich bewegenden Schallquelle. Um solche Informationen mit dem Audiosignal zu verbinden, nutzt er das ARA-Wave Format. Dieses Audioformat wurde bei der 35. international Conference der AES in London im Februar 2009 vorgestellt.

3.4 Erkenntnisse für das zu entwickelnde MARA-System

Die drei vorgestellten Arbeiten geben einen groben Überblick über das Themengebiet von Augmented Reality Audio Systemen. In diesem Unterkapitel werden die Erkenntnisse für meine Masterarbeit hervorgehoben.

Die Dissertation von Jyri Huopaniemi gibt eine gute Übersicht des Aufgabenfelds der Augmented Reality Audio. Dabei stellt er zwei verschiedene Ansätze vor, die für ein mobiles Augmented Reality Audio System denkbar wären. Im Verlauf meiner Arbeit wird nur der Ansatz mittels Kopfhörer verfolgt. Der zweite Ansatz ist trotzdem nicht zu vernachlässigen, da es ein erstes marktfähiges Produkt der Firma Texas Instruments gibt. Der von TI im Bereich Spatial Audio Technology vorgestellte Baustein erzeugt über kleine Lautsprecher ein räumliches Klangerlebnis. Das Einsatzgebiet sind mobile Geräte wie Smartphones und Tablets. Der Baustein arbeitet mit Verfahren Beamforming, Crosstalk Cancellation und Head Related Transfer Function (HRTF). Des Weiteren zeigt er in seiner Arbeit, dass eine Vielzahl der erforderlichen Audio-Filterungen auf Digital Signal Processors (DSPs) berechenbar sind. Diese und der Baustein der Firma Texas Instruments zeigt, dass die von uns gewählte Entwicklungsplattform eine FPGA basierten System on Chip (SoC) eine geeignete Entwicklungsplattform ist.

In dem Paper “Augmented Reality Audio for Mobile and Wearable Appliances“ sind die Anforderungen für ein MARA-System aufgeführt. Dazu zählen die Verbindung der realen Umwelt mit der virtuellen Umwelt, wobei die virtuelle Umwelt sich nicht von dem realen Raum in Bezug der Raumakustik unterscheiden darf. Ein solches System ist ein Echtzeitsystem, eine nicht sofort durchgeführte Berechnung würde eine Fehlanalyse zufolge haben. Ebenso darf das System die Mobilität des Benutzers nicht einschränken. Die aufgezählten Punkte sind somit auch für das zu entwickelnde System einzuhalten. Im zweiten Abschnitt des Papers wurde ein Vergleich zwischen virtuellen und realen Schallquellen durchgeführt. Dieser Vergleich hat gezeigt, dass der wahrgenommene Unterschied zwischen realen und virtuellen Audioquellen sehr gering ist.

In der letzten vorgestellten Arbeit von Mikko Peltola wurde ein funktionierendes MARA-System entwickelt. Dieses simuliert eine virtuelle Schallquelle. In seiner Arbeit stellt er verschiedene Ansätze vor, Zusatzinformationen wie die Position der Schallquelle mit dem Audiosignal zu verbinden. Dazu zählt auch das ARA-Wave Format (35. International Conference der AES, 2009). Ebenso wird auf die Dissertation von Mikka Tikander hingewiesen, welche sich mit dem Thema des AR-Ansatzes (Einbindung der realen Umwelt) beschäftigt.

4 Anforderungen und Konzept für das ARA-System

In diesem Kapitel werden die Vorstellungen und Voraussetzungen für das zu entwickelnde ARA-Systems konkretisiert. Diese unterteilen sich in zwei wesentliche Anforderungen: Zum einem die funktionalen Anforderungen eines ARA-Systems, zum anderen die Anforderungen für das zu entwickelnde Gesamtsystem. Zuletzt wird das Konzept des ARA-Systems vorgestellt, welches im Rahmen dieser zeitlich begrenzten Masterarbeit umgesetzt wurde.

4.1 Funktionale Anforderungen

Es wird ein System entwickelt, welches für einen Zuhörer eine virtuelle dreidimensionale Klangwelt erzeugt. Diese Klangwelt wird dem Zuhörer über Stereo-Kopfhörer simuliert. Das Audio-System simuliert mehrere virtuelle Schallquellen in virtuellen Raum an beliebigen Positionen. Dadurch wird eine realistische virtuelle Audioumgebung erzeugt.
Diese wird durch die Bewegungsfreiheit im Raum verbessert. Dieses impliziert eine kontinuierliche Anpassung der Kopfposition zu den Schallquellen. Die Berechnungen werden in Echtzeit bearbeitet, damit die gehörte Position der Audioquellen erhalten bleibt. Daraus ergibt sich eine geringe Signalverzögerung. Ist die Signalverzögerung zu groß, hätte diese zum Beispiel bei einer Kopfbewegung zufolge, dass die Audioquellen zeitverzögert eingespielt würden. Dieses führt somit zu einer Fehlinterpretation der Klangposition. Des Weiteren sollen die virtuellen Schallquellen einen räumlichen Eindruck beim Zuhörer hinterlassen, um ein gutes Klangerlebnis zu erzeugen.

Ein solches Audiosystem hat eine Vielzahl von Einsatzgebieten, darum ist auf eine einfache Bedien- und Benutzbarkeit und eine einfache Entwicklungsschnittstelle zu achten. Die Bedienbarkeit und Benutzbarkeit dient dem Hörer des 3D-Audio-Systems. Die Benutzung ist intuitiv zu gestalten, damit der Nutzer nicht beeinträchtigt wird. Im besten Fall, fällt dem Zuhörer nicht auf, dass er das ARA-System benutzt. Dieses bedeutet für das System auch, dass es die Anforderungen für ein mobiles Gerät erfüllt. Eine einfache Entwicklungsschnittstelle dient den Entwicklern, die das System einsetzen möchten. Ein Entwickler braucht außerdem keine Audiokenntnisse mitzubringen, um eine Entwicklung mit dem System durchführen zu können. Somit könnte die Technologie in einer Vielzahl von AR und VR Anwendungen ihren Einsatz finden.

Aus diesen Anforderungen an ein ARA-System ergeben sich die Anforderungen für die Hardware- und Software-Verarbeitung.

4.2 Anforderungen an Hard- und Software

Im folgenden Kapitel werden die Anforderungen an die Hardware und Software hervorgehoben und erläutert, welche sich aus den funktionalen Anforderungen ergeben.

Die Audioqualität hat den Anforderungen des menschlichen Gehörs zu entsprechen. Aktuell werden zwei Abtastfrequenzen verwendet. Zum einen die Abtastfrequenz von 44,1 kHz. Diese wird zum Beispiel auf Audio-CDs genutzt. Dann gibt es noch die 48 kHz Abtastfrequenz, welche von den meisten professionellen digitalen Geräten genutzt wird. Ein weiteres wichtiges Merkmal ist die Samplegröße, diese hat mindestens 16 Bit pro Kanal aufzuweisen. Es wurde sich für eine Abtastfrequenz von 48 kHz mit einer 16 Bit Auflösung entschieden. Diese Anforderung resultiert direkt in einen Leistungsanspruch an das ARA-System. Das zu entwickelnde System empfängt und verarbeitet Audiosamples mit einer Abtastfrequenz von 48 kHz.

Die Audioverarbeitung wird in Echtzeit bearbeitet. Wenn die Audiosamples mit 48 kHz vom System aufgenommen werden, müssen diese bis zur nächsten Aufnahme eines Audiosamples bearbeitet werden. Würde die Verarbeitung zu lange dauern, würde diese einem Verlust des Audiosamples entsprechen. Das ist aufgrund der funktionalen Anforderungen der Echtzeitfähigkeit an das System nicht erlaubt. Die Signalfilterung anhand der HRTF, welche in diesem ARA-System zum Einsatz kommt, ist nur mit leistungsstarken Prozessoren oder in Hardwaremodulen (IP-Cores) in Echtzeit berechenbar. Ein Prozessor mit einer Taktrate von 100 MHz hätte beispielsweise für die Signalfilterung eines Samples bei einer Abtastfrequenz von 48 kHz etwa 2083 Takte Zeit um 2 Kanäle zu berechnen. Da in dieser Arbeit mehrere Audioquellen parallel verarbeitet werden, sind die Takte noch durch die Anzahl der parallelen Audioquellen zu teilen. Der Overhead, der durch andere Aufgaben des Systems entsteht, ist bei dieser Berechnung nicht mit eingerechnet.

Eine minimale Signalverzögerung ist für das ARA-System sehr wichtig. Für den Menschen ist in einer virtuellen Realität bereits eine Latenz von 15-20 ms wahrnehmbar []. Diese Latenz bezieht sich auf die Dauer zwischen der Benutzereingabe, beispielsweise der Kopfbewegung des Hörenden, und ihrer audiovisuellen Auswirkung. Zum Beispiel auf die Kopfbewegung des Zuhörers. Eine Verzögerung von über 20 ms wird bereits als störend empfunden.

Für ein ARA-System ist die Positionsbestimmung im Raum wichtig. Zum einem ist die relative Position zwischen dem Hörenden und den Schallquellen wichtig. Diese lässt sich durch den Azimut und die Elevation bestimmen. Wenn die Entfernung zur Schallquelle mitberechnet wird, ist auch diese wichtig. Durch die Raumkoordinaten x, y, z lässt sich die Position der Person und der Schallquelle im Raum beschreiben und zur Berechnung der Entfernung benutzen. Um einen realistischen Raumklang zu erzeugen, sind Rauminformationen wichtig. Im einfachsten Fall genügt die Größe des Raumes und die davon

ausgehende Klangumgebung.

Zur Steuerung des Systems gibt es zwei Sichten. Die des Anwenders, welche intuitiv gehalten wird. Er soll ohne große Einweisung das System benutzen. Ebenso ist zu beachten, dass die Person nicht durch das System beeinträchtigt wird. Die zweite Sicht ist die eines Entwicklers. Für ihn ist ein einfaches Interface zu entwickeln, über welches er die Audiodateien und Informationen an das ARA-System weitergeben kann. Aus diesen Daten erzeugt das System die entsprechende virtuelle Klangwelt.

Bei Augmented Relativ Systemen ist auch die Bewegungsfreiheit des Benutzers wichtig. Deswegen steht auch bei diesem ARA-System die Mobilität des Systems im Vordergrund. Es ist zu beachten, dass eine mobile Datenverbindung zum System entwickelt wird. Ebenfalls ist auf die Größe und das Gewicht der Hardware zu achten. Dazu zählt auch, dass das System über Akkus betrieben werden kann.

4.3 Konzept zur Umsetzung des MARA-Systems

In diesem Unterkapitel werden die Anforderungen aus Kapitel 4.2 in die Systemarchitektur überführt. Die Abbildung 7 zeigt den genauen Aufbau des umgesetzten 3D-Audio-Systems.

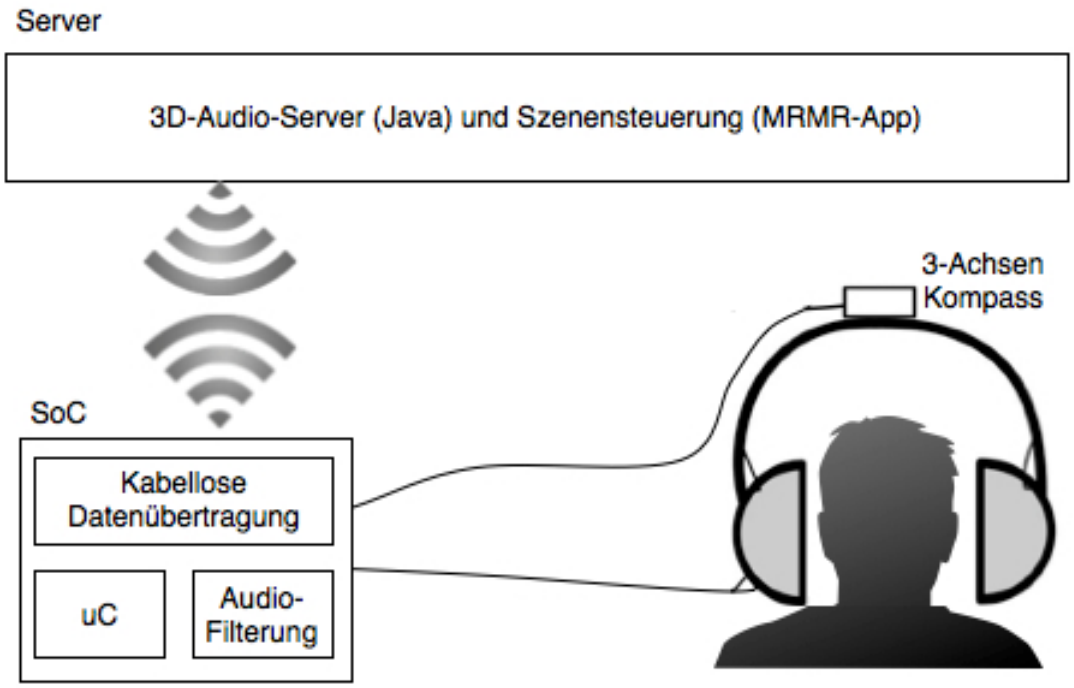


Abbildung 7: Umgesetzte Systemübersicht für das MARA-System

Die 3D-Audio-Serverseite besteht aus zwei Komponenten. Zum einen der 3D-Audio-Server, welcher die Hauptkomponente darstellt. Diese verbindet sich mit dem 3D-Audio-SoC und baut eine Verbindung zwischen beiden auf. Über diese Verbindung werden alle Daten ausgetauscht. Das SoC schickt seine Informationen an den Server. Die grafische Oberfläche des Servers stellt diese daraufhin dar. Ebenso werden über diese Verbindung alle Daten vom Server an das SoC verschickt. Dazu zählen die Audioquellen und deren Positionen. Die Positionen der Schallquellen werden durch die zweite Komponente bereitgestellt. Diese arbeitet auf einem Smartphone und dient als einfache Szenensteuerung. Bei der entwickelten Software wurde darauf geachtet, dass diese plattformunabhängig und leicht erweiterbar ist. Die Entwicklung des 3D-Audio-Servers ist in Kapitel 11 beschrieben.

Für die 3D-Audio-Clientseite wurde sich für einen auf FPGA basierenden System on Chip Ansatz entschieden. Dieses hat den Vorteil, dass die komplette Audioverarbeitung in Hardware berechenbar ist. Die Audioverarbeitung ist sehr gut parallelisierbar, somit wird der Vorteil eines FPGA gut ausgenutzt und es entsteht ein vorhersagbares Verhalten. Der Mikrocontroller des Systems hat nur Steuerungsaufgaben. Über die kabellose Schnittstelle empfängt das SoC die Daten vom 3D-Audio-Server und verarbeitet diese. Die Kopfposition des Benutzers wird über einen 3-Achsen-Kompass verfolgt. Die Entwicklung des Hardwarebeschleunigers für die Audioverarbeitung wird in Kapitel 7 beschrieben. Die Entwicklung des 3D-Audio-Clients ist in Kapitel 8 erläutert.

In dieser Arbeit werden nicht alle Aspekte aus den Anforderungen an ein ARA-System erfüllt. Der Augmented Reality Ansatz wird nur teilweise erfüllt. Durch die Nutzung von offenen Kopfhörern nimmt der Hörer zwar noch die reale Umgebung war, diese wird aber trotzdem verfälscht. Des Weiteren wird die Entfernung zur Schallquelle nicht simuliert. Dazu müsste die Position des Hörers im Raum bekannt sein. Ebenso wird der Raumklang, der durch die virtuellen Schallquellen erzeugt wird, nicht eingebunden. Um diesen zu berechnen, ist ein genaues Abbild des Raumes wichtig. Eine Entfernungs- und Raumklang-Nachbildung würden den Rahmen dieser Masterarbeit überschreiten.

5 Entwicklungsumgebungen des SoCs

5.1 SoC-Plattform Xilinx Virtex-5 ML507

Als Hardwareplattform wurde das Xilinx Virtex-5 ML507 ausgewählt, welches zuvor schon bei anderen Arbeiten zum Einsatz gekommen ist. Dieses Entwicklungsboard ist für System-On-Chip (SoC) Systeme ausgelegt. Es ermöglicht leistungsfähige parallelisierbare FPGA-Hardwaremodule einzubinden. Der integrierbare Softcore-Prozessor (MicroBlaze) kann diese FPGA-Hardwaremodule über Software ansprechen. Der Vorteil eines Softcore-Prozessors wie dem MicroBlaze ist, dass dieser im FPGA aufgebaut wird und damit für das entsprechende System optimierbar ist. Der MicroBlaze ist ein RISC Mikrocontroller 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 Softcore-Prozessor integriert. Somit übernimmt der Prozessor nur logiklastige Steueraufgaben.

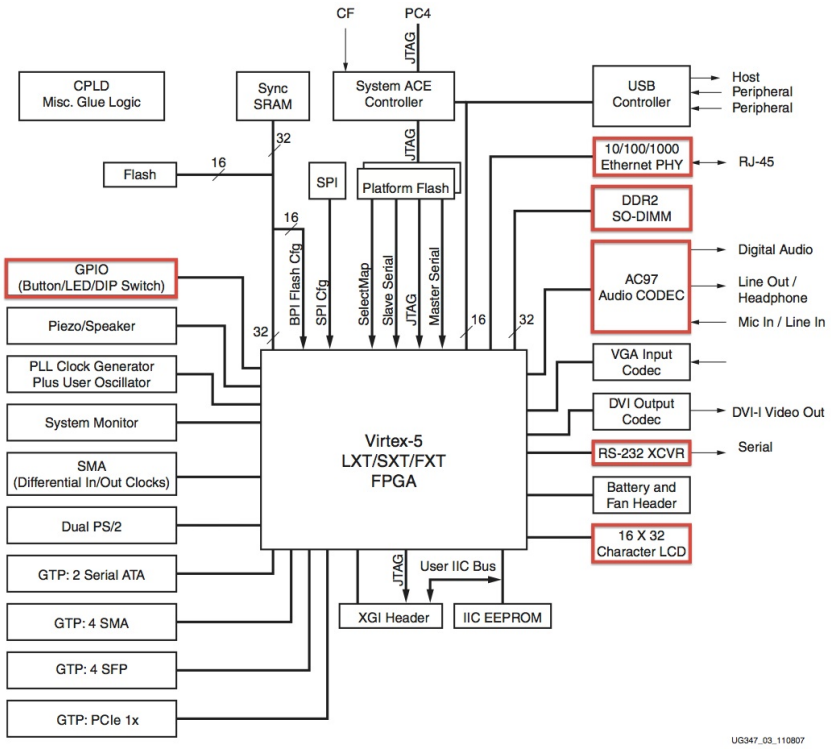


Abbildung 8: Blockdiagramm Virtex 5 ML507 [Xil11]

Wie die Grafik 8 zeigt, ist dieses System sehr flexibel einsetzbar und lässt sich für den Aufgabenbereich, in dem es eingesetzt wird, individuell konfigurieren. Rot eingefärbt sind die Module, die für das Mobile Augmented Reality Audio System eingebunden wurden.

Zum Aufbau des SoC gibt es die Xilinx Platform Studio (XPS) Entwicklungsumgebung. Sie ist ein Teil aus dem Xilinx-Embedded-Development-Kit 14.4 (EDK) und ermöglicht Hardware-Entwicklern ein sehr individuelles Embedded-System zu entwickeln. Die grafische Oberfläche bietet viele Assistenten, wie z.B. den Base System Builder, mit dessen Hilfe der Benutzer sehr bequem sein System entwickeln kann.

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 Package (BSP). Im SDK gibt es weitere Tools, wie zum Beispiel einen Linker-Script-Generator und einen komfortablen Debugmodus.

5.2 Xilinx System Generator für Embedded Systems

Xilinx bietet eine Erweiterung für die MATLAB/Simulink Umgebung [Mat]. Der Xilinx System Generator erlaubt es, aus der Modellierung von digitalen Fixed-Point Signalverarbeitungsalgorithmen, HDL-Codes für Xilinx FPGAs zu erzeugen. Die Modelle werden über den Simulink Ansatz, der grafischen Modellierung von Algorithmen, aus dem Xilinx DSP Blockset zusammengebaut. [Xil]

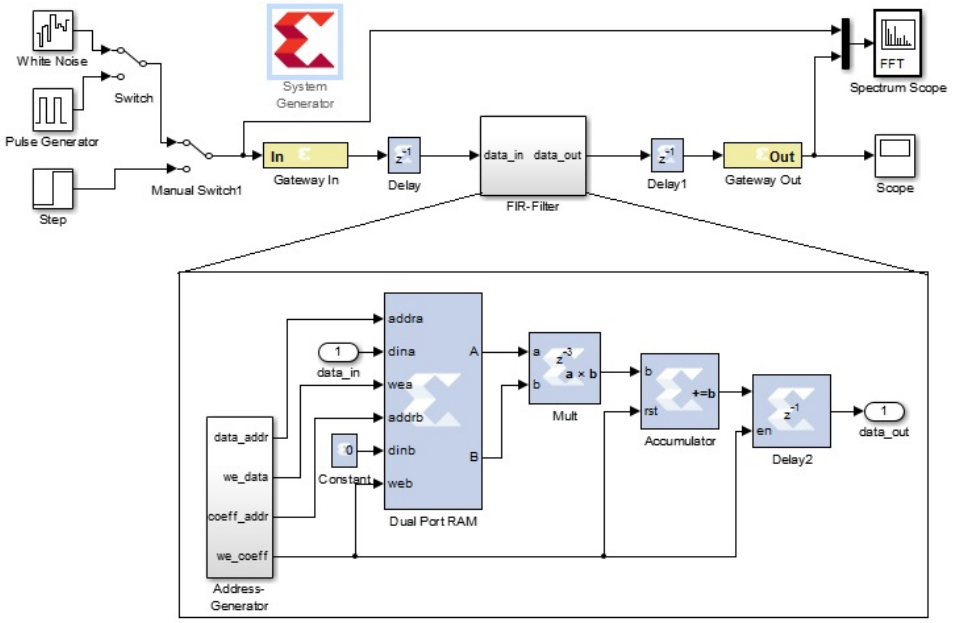


Abbildung 9: Modellierung eines Gesamtsystems mithilfe des System Generators am Beispiel eines FIR-Filters

Dazu stehen eine Vielzahl von Abstraktionselementen für mathematische-, logische-, Speicher- und DSP-Funktionen zur Verfügung. Die Blöcke sind sowohl bit-true als auch cycle-true. Dies beschreibt eine exakte, bitgenaue und zeitdiskrete Abbildung des Modells in die physikalische Hardwareplattform. Ein großer Vorteil ist, dass das Verhalten des Modells durch die Simulink Umgebung bereits bei der Entwicklung verifiziert wird. Ist diese nicht ausreichend, ist eine VHDL-Testbench-Generierung zur Simulation in ModelSim vorhanden. Die Erweiterung hat ebenfalls den vollständigen MATLAB Funktionsumfang, wodurch Systementwürfe vor der Umsetzung in System Generator Blöcke evaluierbar sind. Außerdem lässt sich auf alle MATLAB-Funktionen sowie auf MATLAB-Workspaces zugreifen.

Zur Modellierung stellt die Xilinx Blockset Library folgende Funktionsblöcke zur Verfügung:

- Grundblöcke (z.B. Concat, Up und Down Sampler, ...)
- Logikblöcke (z.B. Logical, MUX, Relational)
- Signalverarbeitungsblöcke (z.B. DSP48, FIR, FFT)
- Arithmetikblöcke (z.B. Mult, AddSub, Div)
- Speicherblöcke (z.B. Shift Register, FIFO, ROM, RAM)

Das Verhalten des Modells wird durch das Verbinden der Eingangs- und Ausgangsports der verschiedenen Funktionsblöcke beschrieben. Der durch Simulink vorgebend modulare Systementwurf erleichtert die Simulation, sowie die Wartbarkeit und Wiederverwendbarkeit der so entwickelten Module.

Um System Generator Module in ein Gesamtsystem einzubinden, werden die äußeren Systemschnittstellen durch Gateway in und Gateway out Module entkoppelt. Die Ein- bzw. Ausgangssignale müssen in den Modulen dabei dem Xilinx Fixed-Point Format genügen. Vorzeichenbehaftete Werte in Zweierkomplementdarstellung werden als FIX und vorzeichenlose Zahlen als UFIX bezeichnet. Das Fixed-Point-Format hat den Vorteil, dass die Berechnungen in Integer-Arithmetik durchgeführt werden. Gleichzeitig wird die Verteilung von Ganzzahl- und Fließkomma-Anteilen für Datentypen im FIX/UFIX-Format durch den Entwickler festgelegt, wodurch die Vektorbreiten, abhängig von der konkreten Verwendung des Datentyps, dimensionierbar wird [ASH08].

6 HRTF Messungen und Berechnung der Koeffizienten

Die Bachelorarbeit "HRTF Measurements and Filter Design for a Headphone-Based 3D-Audio System“ von S. Sima bietet ein funktionsfähiges MATLAB-System zur Berechnung von HRTFs aus Kunstkopf-HRIR-Messungen. 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. 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 Azimut von 0° bis 180° mit einer Schrittweite von 30°. Die zweite Messreihe hat für die Elevation +37° und 0° je die Messungen für die Azimut von 0° bis 180° mit einer Schrittweite von 15°. [Sim08]

In dieser Arbeit werden die Laufzeitdifferenz und die Pegeldifferenz getrennt voneinander verarbeitet. Dadurch ergeben sich für die Anwendung der SoC-Plattform zwei Filter-Typen. Ein FIR-Filter, der die Filterung für die Pegeldifferenz übernimmt und sowie ein Verzögerungsglied, das 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]

Um aus dieser Messreihe FIR-Filterkoeffizienten zu errechnen, wird das MATLAB-Skript “calculate_coeff_sima.m“ aus der Bachelorarbeit von S. Sima benutzt und erweitert. Auf eine genaue Erläuterung dieses MATLAB-Skripts wird verzichtet, weil dieses ausführlich in der Bachelorarbeit von S. Sima erklärt wird.

Bei der Messreihe Sima wurde der Phasengang und die Pegeldifferenz separat gemessen. Die Filterkoeffizienten für die Pegeldifferenz (ILD) werden wie in der Arbeit von Sima 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 Gruppenlaufzeit haben. Das ist für die interaurale Pegeldifferenz (ILD) gewünscht. Die Grafik 10 zeigt die Impulsantworten des rechten und linken Kanals auf der Elevationebene 0° mit einem Azimut von 60°.

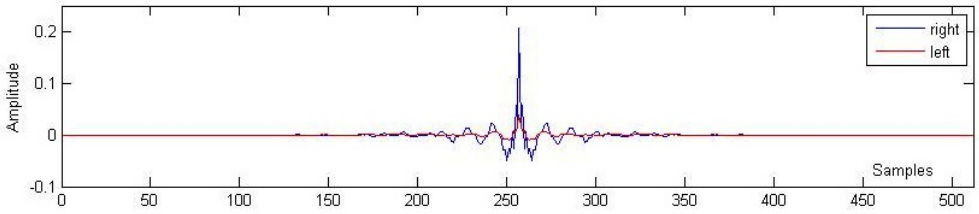


Abbildung 10: linke und rechte Impulsantwort der Sima-HRTF bei Elevation 0°, Azimuth 60° mit konstanter Verzögerung

Wie in der Grafik zu erkennen ist, sind die ersten und letzten Filterkoeffizienten sehr klein und haben somit keine Auswirkung auf das Filterergebnis. Um die Performanz zu erhören, wurde bei allen HRTF-Koeffizientensätzen geschaut, wo die ersten und letzten Auswirkungen auf das Signal vorkommen. Es hat sich gezeigt, dass eine Filterlängenverkürzung von 512 auf 256 Koeffizienten durchführbar ist. Alle HRTF-Koeffizientensätze wurden auf die Filterlänge von 256 gekürzt, dazu wurden jeweils die ersten und die letzten 128 Filterkoeffizienten entfernt.

Weiterhin wurde eine Anpassung der Filternormierung für den Einsatz in der SoC-Plattform durchgeführt. Es gibt mehrere Ansätze die Normierung von Filterkoeffizienten anzugehen. Zwei Methoden wurden in Kapitel 2.3.1 vorgestellt.

Die L1-Norm und L2-Norm wurden für alle HRTF-Koeffizientensätze berechnet. 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 der entsprechenden Norm. Beide Normierungsverfahren haben kein gutes Hörergebnis vermittelt. Im Fall der L1-Norm war trotz lautem Eingangssignal das gefilterte Ausgangssignal kaum noch hörbar. Bei der Normierung mit dem L2-Normfaktor gab es Übersteuerungseffekte. Da die Filter in Hardware selbst gebaut werden, wurde sich dazu entschlossen, den Normierungsfaktor dem maximalen absoluten Koeffizienten über alle HRTF-Koeffizientensätze gleichzusetzen. Durch dieses Verfahren ist der Wertebereich der Filterkoeffizienten von -1 bis +1 normiert. So ist sichergestellt, dass bei der Multiplikation mit dem Filterkoeffizienten keine Wertebereichsüberschreitung auftreten können. Bei der Akkumulation können Überlaufe auftreten, diese werden durch zusätzliche Guard-Bits im FIR-Filter aufgefangen. Die Anzahl der dafür zusätzlich erforderlichen Bits lässt sich durch die angegebene Formel beschreiben:

$$GuardBits = \lceil \log_2(\sum_{k=0}^{N} |c_k|) \rceil \tag{5}$$

Durch eine Sättigung des Ausgangssignals wird das Ergebnis wieder auf den Wertebereich von -1 bis +1 gebracht.

7 Entwicklung des Hardwarebeschleunigers zur Berechnung der virtuellen Audioumgebung

Da sich für jede Winkelposition (vertikal und horizontal) die HRTF-Filterung verändert, ist für jede Position die Berechnung für die entsprechende Winkelposition durchzuführen. Das hat eine große Anzahl von Filterkoeffizienten zur Folge. Bei der Annahme, dass der Evaluationswinkelbereich von +37° bis -41° und der Azimutbereich von 360° in 1°-Schritten abgedeckt wird, sind 56160 Filterkoeffizientensätze notwendig. Diese auf dem System permanent zu speichern ist mit der entwickelten SoC-Plattform nicht zu realisieren.

Ein anderer Ansatz ist es, dem System für bestimmte Winkelpositionen die HRTF-Filterkoeffizientensätze zur Verfügung zu stellen. Mit diesen Stützstellen errechnet das System durch eine Interpolation das Filterergebnis. Die als Grundlage vorliegende Bachelorarbeit von Silvia Sima stellt drei Elevationsebenen bereit. Zu jeder dieser Ebenen sind die horizontalen Ebenen mit einer Schrittweite von 30° gemessen worden. Durch das Vertauschen der rechten und linken Filterergebnisse ab einem Azimut größer 180° lässt sich die Anzahl der Filterkoeffizientensätze deutlich reduzieren. Es werden in diesem Fall nur noch 2 Kanäle (rechts, links) mit 3 Elevationsebenen und jeweils 7 Azimutstützstellen bereitgestellt. Dieses reduziert die erforderlichen Filter auf 42 Stützstellen. Die Abbildung 11 zeigt die Stützstellen grafisch.

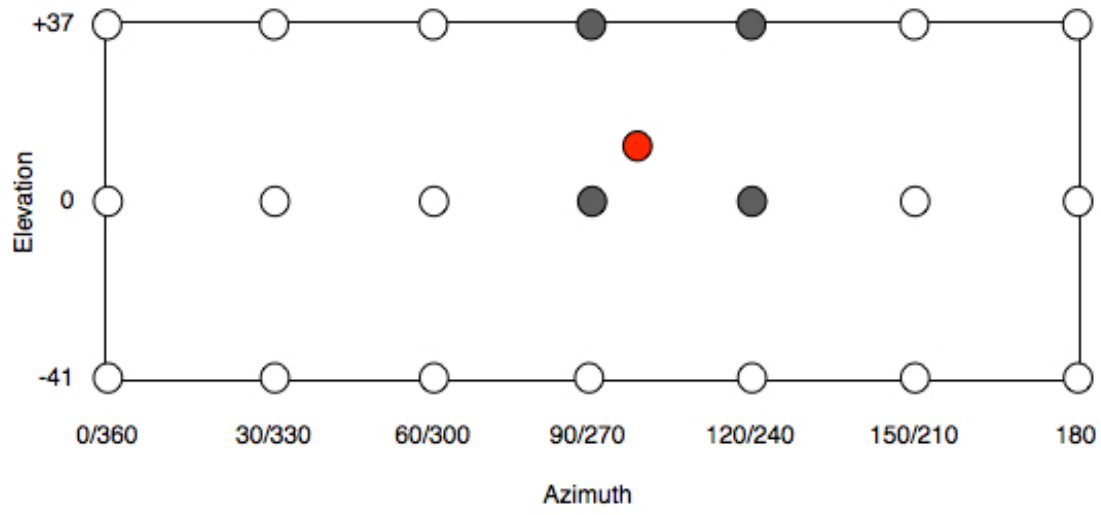


Abbildung 11: Anordnung der HRTF-Stützstellen-Filter für die SoC-Plattform. Ab einen Azimut größer 180° werden die Kanäle für das rechte und linke Ohr getauscht. Grau eingefärbt sind die benachbarten Stützstellen für die in rot eingezeichnete Position.

Grau in der Grafik 11 dargestellt sind die vier benachbarten Stützstellen, welche für die Berechnung der rot eingezeichneten Winkelposition herangezogen werden. Durch die Berechnung dieser Stützstellen und einer Interpolation zwischen diesen, unter Berücksichtigung des Abstandes zur gewünschten Filterposition, ergibt sich das gewünschte Filterergebnis. Da die Filter nur die HRTF-ILD berechnen, wird nach der Interpolation die HRTF-ITD Filterung durchgeführt.

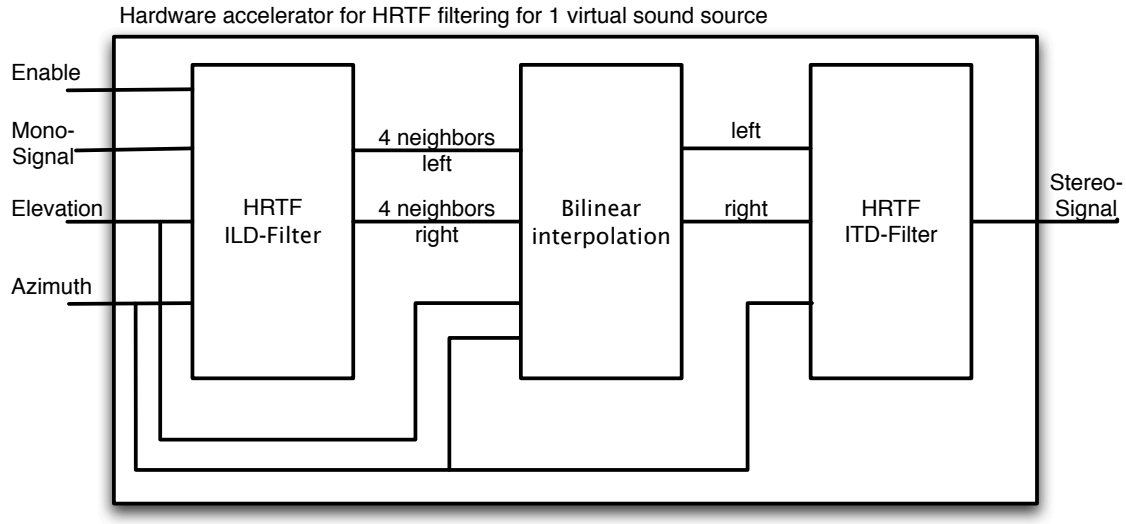


Abbildung 12: Hardwarebeschleuniger zur HRTF-Filterung für eine virtuelle Schallquelle mit getrennter HRTF-Verarbeitungskette und bilinearer Interpolation

In Abbildung 12 ist der Aufbau des entwickelten Hardwarebeschleunigers zur Berechnung einer Schallquelle mittels HRTF-Filterung dargestellt. Dieser unterteilt sich in drei Module. Zur Optimierung der HRTF-Filterung werden die interaurale Pegeldifferenz (ILD) und die interaurale Laufzeitdifferenz (ITD) getrennt und somit separat voneinander verarbeitet. Zwischen den interauralen Pegeldifferenz-Stützstellenfiltern wird eine bilineare Interpolation durchgeführt. Die Entwicklung dieses Hardwarebeschleunigers wurde mit dem Xilinx Systemgenerator [Xil12b] unter MATLAB/Simulink durchgeführt.

Der HRTF-ILD Filter ist ein FIR-Filter. Der Aufbau des Filters wird in Kapitel 7.1 vorgestellt. Die aus diesem Filter errechneten Ergebnisse werden durch eine bilineare Interpolation zur gewünschten Filterposition berechnet (Kapitel 7.2). Die daraus erzeugten Kanäle für das rechte und linke Ohr werden durch den HRTF-ITD Filter verarbeitet, Näheres dazu in Kapitel 7.3.

7.1 HRTF-ILD-Filter

Der HRTF-ILD Filter ist in Abbildung 13 schematisch dargestellt. Die Koeffizienten-RAM-Blöcke stellen die entsprechenden Filterkoeffizienten anhand des Azimuts und der Elevation zur Verfügung. Der Audio-RAM speichert die Audiosamples ab. Der Adress-Generator steuert die RAM-Blöcke, somit werden in den 8 MAC-Units die Filterergebnisse der benachbarten Stützstellen berechnet.

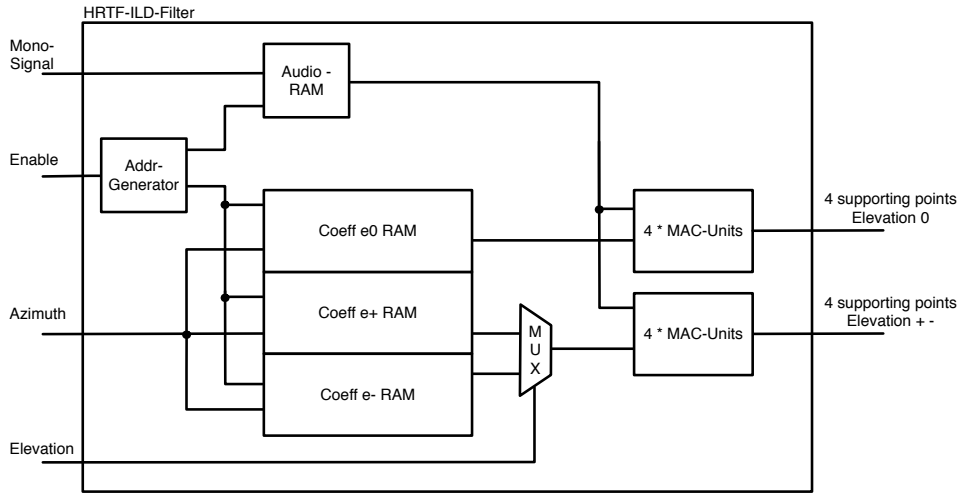


Abbildung 13: Aufbau des HRTF-ILD-Filters mit Koeffizienten-RAM-Blöcken und 8 MAC-Units zur Filterung einer virtuellen Schallquelle

Das Mono-Signal ist das Audiosignal der virtuellen Schallquelle und wird für die Filterung in einem Audio-RAM abgespeichert. Das Eingangs-Enable-Signal gibt einen Berechnungsdurchlauf des Filters frei. Durch eine steigende Flanke wird der Adressgenerator aktiviert und übernimmt die Ansteuerung der Einzelmodule.

Der Adressgenerator ist zugleich auch die Steuerungseinheit des Filters. Auf eine steigende Flanke erzeugt dieser eine Write Enable (WE) für den Audiospeicher, woraufhin dieser das anliegende Audiosample speichert. Im nächsten Takt werden die Adressen für die Filterkoeffizientenspeicher und den Audiospeicher erzeugt. Das Verhalten des Adressgenerators ist in Abbildung 14 aufgezeigt.

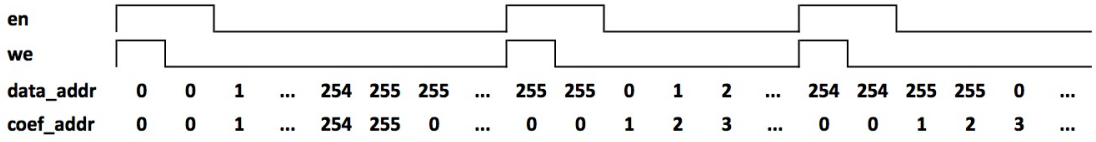


Abbildung 14: Ablauf des Adressgenerators für den HRTF-ILD-Filter

Die Datenadressfolge (data_addr) zählt von 0 bis N = 255 und setzt nach jeden N+1 Takten für ein Taktsignal aus. Das Data Write Enable Signal (we) wird aus einem Puls-Shorter aus dem Enable Signal generiert. Die Koeffizientenadresse (coef_addr) zählt durchgehend von 0 bis N = 255. Die Koeffizienten des rechten und linken Kanals liegen hintereinander in einem Dual-Port-RAM. Um beide zusammengehörenden Koeffizienten zu erhalten, wird die Koeffizientenadresse mit der Anzahl der Koeffizienten addiert und daraus die zweite Koeffizientenadresse bestimmt.

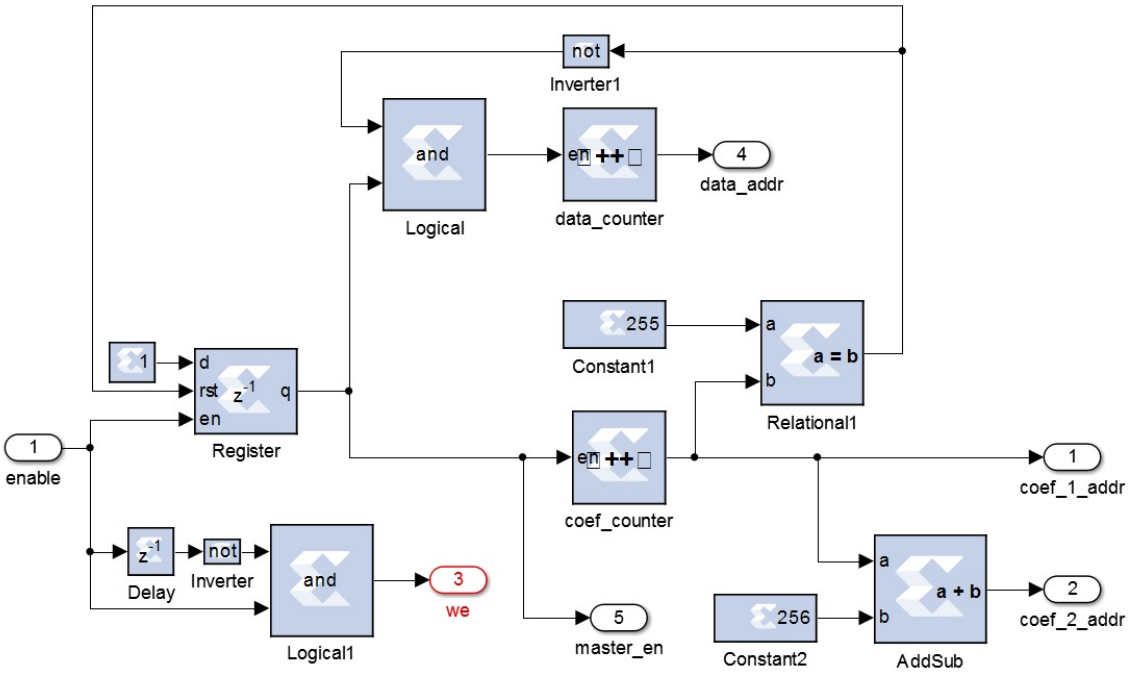


Abbildung 15: Aufbau des Adressgenerators für den HRTF-ILD-Filter

Die Abbildung 15 zeigt den Aufbau des Adressgenerators. Das Enable Signal erzeugt durch einen Puls-Shorter das Write Enable (we) für den Audiospeicher. Zugleich wird ein Enable-Bit für die Adressgenerator-Einheit erzeugt. Dieser wird durch ein weiteres Signal gelöscht, welches anzeigt, dass ein Filterdurchlauf durchgeführt wurde. Der Coef_Counter zählt von 0 bis 255. Durch das Addieren von 256 wird zugleich die zweite Koeffizientenadresse berechnet. Durch den Vergleich des Coef_Counter mit 255 wird ein Signal erzeugt. Dieses wird invertiert und mit dem Enable-Signal des Gesamtsystems verundet und dient als Enable für den Data_Counter.

Die Koeffizienten-RAM-Blöcke beinhalten jeweils für jede Ebene alle notwendigen Koeffizienten. Dabei handelt es sich um 7 Dual-Port-RAM-Blöcke (Azimut 0°, 30°, 60°, 90°,

120°, 150°, 180°). In jedem RAM liegen zuerst die Koeffizienten für den linken Kanal, danach die für den rechten Kanal. Die Adressen für die Ausgabe der jeweiligen Koeffizienten werden durch den Adressgenerator erzeugt. Write Enables sind nicht erforderlich, da die Koeffizienten sich nicht verändern. Durch entsprechende Multiplexer, welche der Azimut_MUX gesteuert, werden die entsprechenden Koeffizienten zur Verfügung gestellt. Der Azimut generiert das Azimut_MUX Signal, welches in einem vorgelagerten Modul berechnet wird. In dem vorgelagerten Modul wird der relative Winkel zwischen der Schallquelle und der Kopfposition berechnet.

$$Azimut := \begin{cases} Audioazimut - Compassazimut \text{ falls} & Compassazimut <= Audioazimut \\ Compassazimut - Audioazimut \text{ sonst} & \end{cases} \tag{6}$$

Wenn der errechnete Azimut größer als 180° ist, wird dieser umgerechnet.

$$Azimut = 360 - Azimut \tag{7}$$

Ein zusätzliches Signal gibt an, dass die Filterposition der Schallquelle größer ist und der rechte und linke Kanal vertauscht sind. Aus diesem so errechneten Azimut wird das Azimut_MUX-Signal wie folgt erzeugt.

$$Azimut\_MUX := \begin{cases} 0 \text{ falls} & Azimut > 0 \;\&\; Azimut < 30 \\ 1 \text{ falls} & Azimut > 30 \;\&\; Azimut < 60 \\ 2 \text{ falls} & Azimut > 60 \;\&\; Azimut < 90 \\ 3 \text{ falls} & Azimut > 90 \;\&\; Azimut < 120 \\ 4 \text{ falls} & Azimut > 120 \;\&\; Azimut < 150 \\ 5 \text{ sonst} & \end{cases} \tag{8}$$

Mithilfe des Elevation_MUX-Signals werden durch einen Multiplexer am Ausgang der Koeffizienten-RAM-Blöcke 'e+' und 'e-' die entsprechenden Koeffizienten weitergereicht. Dazu wird der relative Elevationswinkel zwischen der Kopfposition und der Schallquelle bestimmt.

$$Elevation = Audioelevation - Compasselevation \tag{9}$$

Das Signal Elevation_MUX wird wie folgt bestimmt:

$$Elevation\_MUX := \begin{cases} 1 \text{ falls} & Audioelevation <= Compasselevation \\ 0 \text{ sonst} & \end{cases} \tag{10}$$

Hiermit stehen alle Informationen zur Verfügung und die MAC-Einheiten des Filters berechnen die Filterergebnisse. Es sind 8 gleiche MAC-Einheiten erforderlich (2 Kanäle * 4 Stützstellen). Eine dieser MAC-Einheiten ist in Abbildung 16 dargestellt.

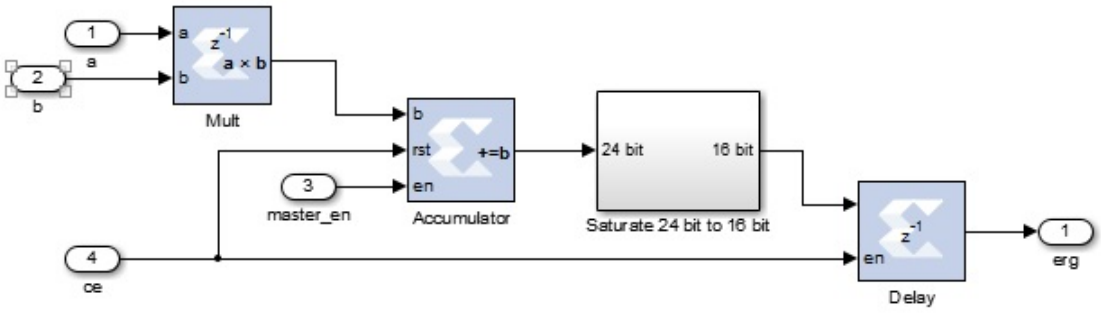


Abbildung 16: Aufbau einer MAC-Einheit mit Sättigungsmodul

Die Signale a und b stellen in jedem Takt den Filterkoeffizienten und das dazugehörige Audiosample zur Verfügung. Das Ergebnis aus der Multiplikation wird im Accumulator aufsummiert. Der Accumulator ist nur aktiv, wenn der Master-EN aus dem Adressgenerator gesetzt ist. Das Accumulator Ergebnis besitzt zusätzliche Guard-Bits, um Überläufe aus der Filternormierung aufzufangen. Durch die Sättigungseinheit wird das Ergebnis wieder in dem Wertebereich -1 bis +1 gebracht und auf 16 Bit gekürzt. Durch ein weiteres Signal, was das Ende der Berechnung anzeigt, wird das Accumulator-Ergebnis in ein Register gespeichert und aus der Accumulatoreinheit gelöscht.

Die Sättigungseinheit ist in Abbildung 17 dargestellt.

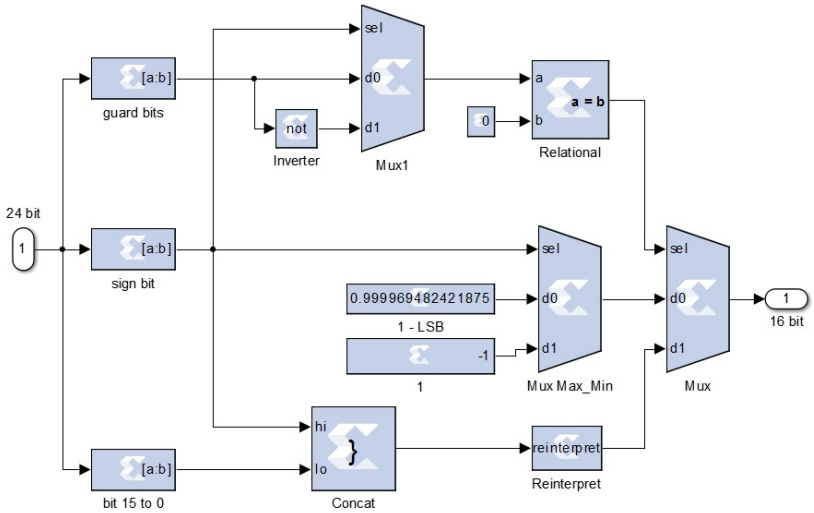


Abbildung 17: Aufbau des Sättigungsmoduls mit Verkürzung des Signals auf FIX16_15

In dem Modul wird das Accumulator Ergebnis in das Vorzeichenbit, die Guardbits und die Nachkommabits aufgeteilt. Durch die Kontrolle der Guard-Bits ist bekannt, ob es zu einer Überschreitung des Wertebreichs gekommen ist. Sind Guard-Bits im positiven Wertebereich vorhanden, bedeutet dies eine Überschreitung im positiven Wertebereich. Durch eine Negierung und die Eigenschaften des Zweierkompliments gilt dieses auch im negativen Wertebereich. Durch das so erzeugte Signal wird entweder -1 oder 1-LSB ausgegeben oder das aus Vorzeichenbit und Nachkommabits zusammengefügte Signal. Durch dieses Modul werden Überläufe im Accumulator abgefangen. Ist das Endergebnis größer als 1 oder -1 wird zugleich eine Sättigung durchgeführt.

Die Grafik 18 zeigt den Vergleich zwischen einer MAC-Einheit mit und ohne Sättigungsmodule.

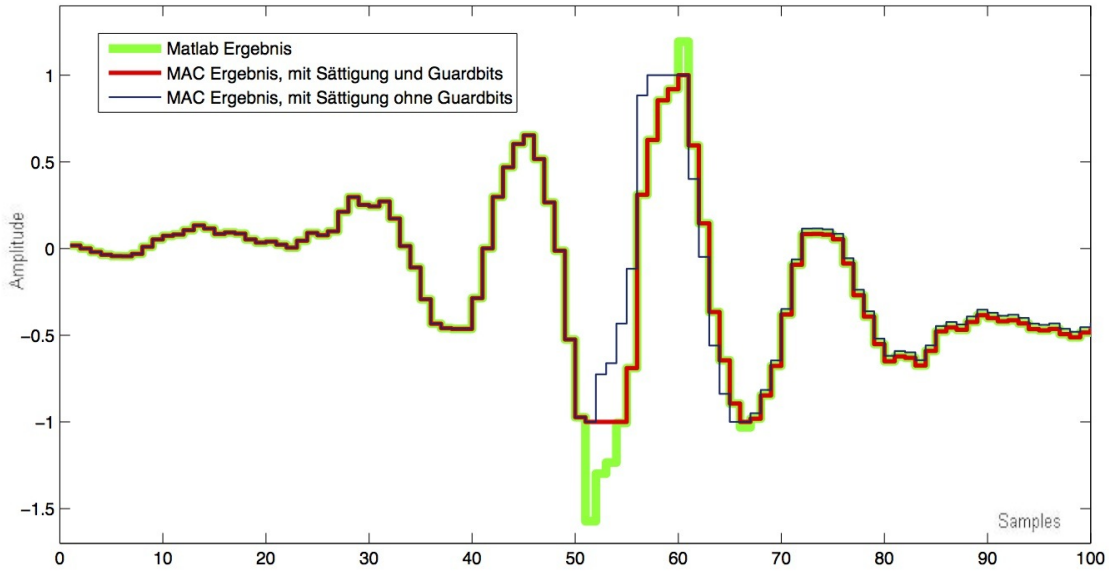


Abbildung 18: Wertebereichsüberschreitung bei einem Filterergebnis. Vergleich zwischen Sättigung und Sättigungsmodul mit Guard-Bits.

In Grün dargestellt ist das MATLAB-Filterergebnis, welches an mehreren Stellen den Wertebereich überschreitet. Die in rot eingezeichnete Linie ist das Ergebnis, welches aus der MAC-Einheit mit Sättigungsmodul und Guards-Bits berechnet wird. Gut zu erkennen ist, dass das Ergebnis denselben Kurvenverlauf im Wertebereich aufweist. Bei einer Überschreitung des Wertebreichs wird eine Sättigung des Signals durchgeführt. In blau eingezeichnet ist das Ergebnis eines MAC-Filters mit Sättigung ohne Guard-Bits. Dabei ist zu erkennen, dass nach der ersten Wertebereichsüberschreitung das Ausgangssignal verfälscht wird.

7.2 Bilineare Interpolation

Die bilineare Interpolation ist eine Erweiterung der linearen Interpolation um Zwischenwerte innerhalb eines zweidimensionalen, regulären Rasters zu bestimmen. In dieser Anwendung wird zwischen den Stützstellen der HRTF-ILD-Filterung interpoliert.

$$f(x,y) = \begin{bmatrix} 1-x & x \end{bmatrix} \times \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \times \begin{bmatrix} 1-y \\ y \end{bmatrix} \tag{11}$$

Die Gleichung geht davon aus, dass x und y im Wertbereich von 0 bis 1 liegen. Dieses ist bei dieser Interpolation nicht gegeben, da die Stützstellen auf der horizontalen Ebene nur alle 30° vorhanden sind. Auf der vertikalen Ebene hängen sie von der Messreihe ab (Kemar 30° und Sima -41°, +37°).

$$x = \frac{\text{Azimut mod } 30^\circ}{30^\circ} \tag{12}$$

$$y = \frac{\text{Elevation mod ElevationStep}^\circ}{\text{ElevationStep}^\circ} \tag{13}$$

Ausgeschrieben sieht die Gleichung wie folgt aus:

$$f(x,y) = (1-y)*((1-x)*ST1 + x*ST2) + y*((1-x)*ST3 + x*ST4) \tag{14}$$

Aus dieser Formel lässt sich ein ASM-Entwurf entwerfen. Ein ASM-Entwurf ist eine Methode zum Designen von Zustandsautomaten, wobei anstelle eines Zustandsdiagramms ein informelleres, aber leichter verständlicheres ASM-Chart genutzt wird, das die sequenziellen Operationen beschreibt [RS09]. Die ASM zur Berechnung der bilinearen Interpolation unterteilt sich in einen Steuer- und einen Datenpfad. Es wird auf die Nutzung von genau einem Multiplizierer und einem Subtrahierer/Addierer beschränkt.

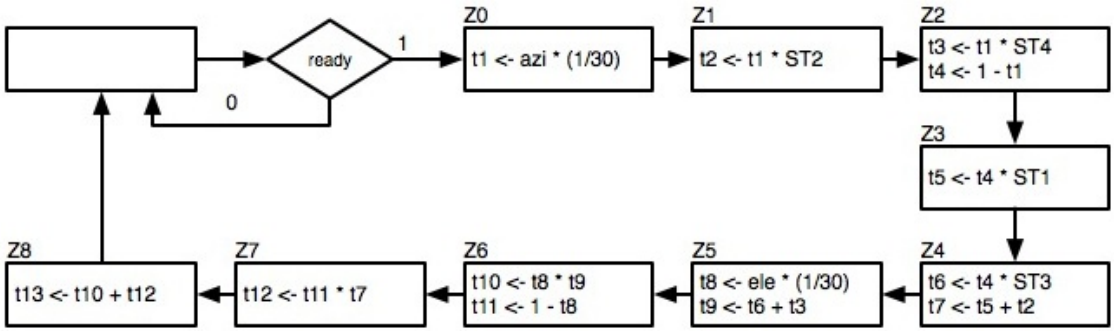


Abbildung 19: ASM-Chart der Bilinearen-Interpolation-ASM

Durch Nutzung einer parallelen Multiplizierer- und Subtrahierer/Addierereinheit lässt sich die bilineare Interpolation in 9 Zuständen abarbeiten. Aufgrund der Berechnung in Fixpoint-Arithmetik wurde eine Wertebereichsbetrachtung durchgeführt. Das Ergebnis ist in einer Tabelle im Anhang B beigefügt.

Neben der Einsparung von FPGA-Ressourcen durch Festlegung der Vektorbreiten wurden die Zwischenschritte auf ein Register Sharing analysiert. (Siehe Tabelle 1.)

	z0	z1	z2	z3	z4	z5	z6	z7	z8	Operation	Register
t1	e	x	x							*	R1
t2		e	x	x	x					*	R2
t3			e	x	x	x				*	R1
t4			e	x	x					-	R4
t5				e	x					*	R3
t6					e	x				*	R2
t7					e	x	x	x		+	R4
t8						e	x			*	R1
t9						e	x			+	R3
t10							e	x	x	*	R1
t11							e	x		-	R3
t12								e	x	*	R2
t13									e	+	R4

Tabelle 1: Register Sharing der Bilinearen-Interpolation-ASM.

Auf Basis des Register Sharings und der Signalvektorbreiten lässt sich die RTL-Modellierung des Datenpfades durchführen. Diese ist als MATLAB/Simulink Modell in Abbildung 20 dargestellt.

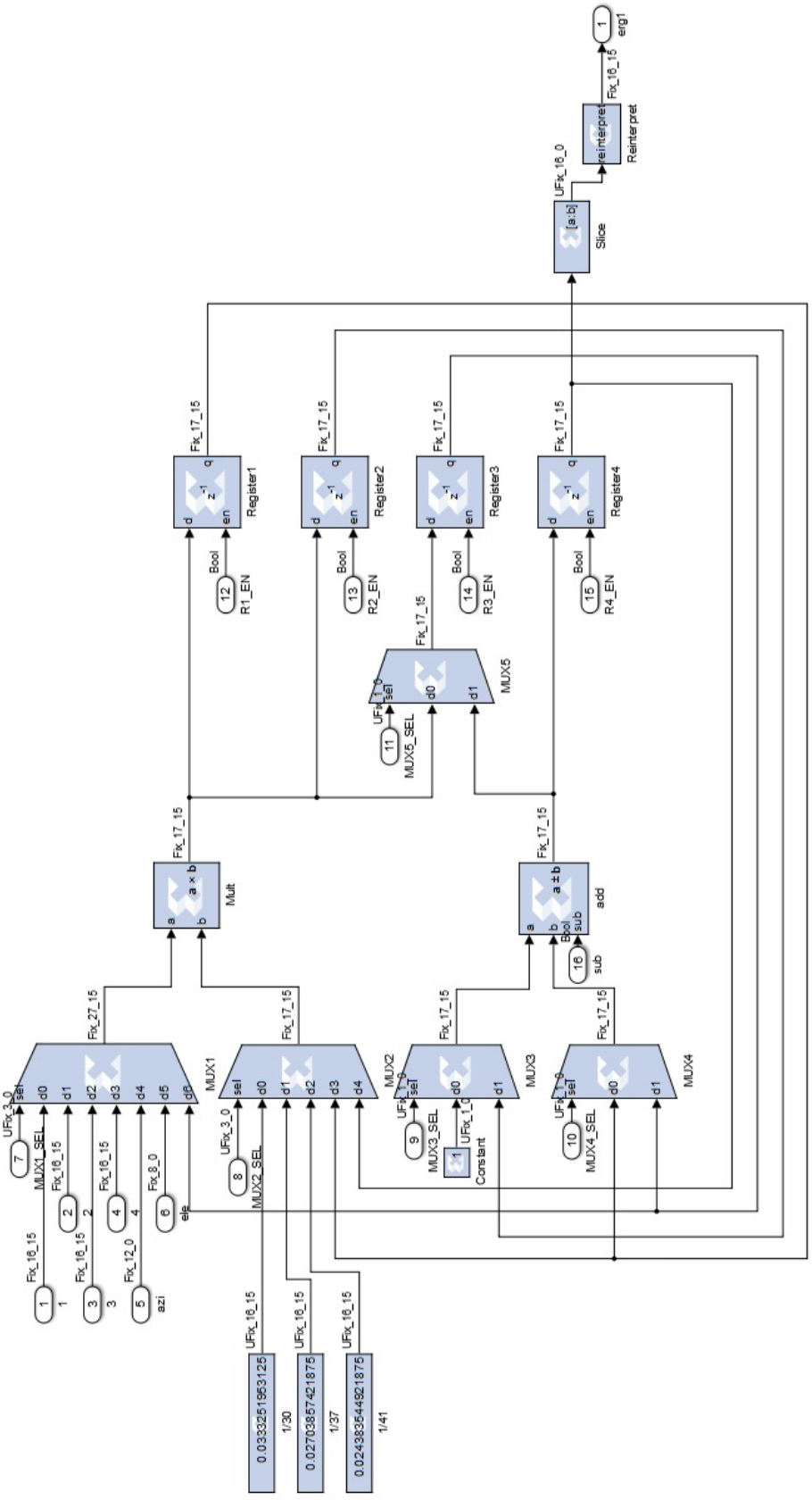


Abbildung 20: Datenpfad der Bilinearen-Interpolation-ASM. Das Ergebnis wird mithilfe jeweils einer Multiplizierer- und Subtrahierer/Addierereinheit berechnet und das Ergebnis auf den Wertebereich Fix_16_15 beschränkt.

Die bilineare Interpolation ist für beide Kanäle (rechts und links) durchzuführen. Dazu gibt es mehrere Ansätze um die bilineare Interpolation zu erweitern. Zum einem kann der Datenpfad zweimal instanziiert werden. Dieses hat zum Vorteil, dass beide Kanäle parallel verarbeitet werden, erfordert aber gleichzeitig doppelt so viel Ressourcen. Der andere Ansatz ist beide Kanäle nacheinander durch den gleichen Datenpfad zu verarbeiten. Dieses hat zur Folge, dass die Laufzeit zur Berechnung der Ergebnisse verlängert wird.

Der zweite Ansatz wurde in dieser Arbeit verfolgt. Dazu werden beide Audiokanäle nacheinander verarbeitet. Wie im ASM-Chart (Abbildung 8) zu sehen ist, wird im ersten Zustand Z0 eine Multiplikation und im letzten Zustand eine Addition durchgeführt. Da die beiden Ergebnisse in unterschiedliche Register gespeichert werden, lassen sich die beiden Zustände zusammenführen. Dieses verkürzt den ASM-Chart um einen Zustand. Somit werden beide Ergebnisse in 17 Takten berechnet.

Zur Steuerung des Datenpfades wurde eine FSM entworfen, welche die Steuerung mithilfe der Steuersignale übernimmt. Die Steuer-FSM wurde mittels eines MCode-Blocks realisiert und ist im Anhang B beigefügt. Die Abbildung 21 zeigt die Kopplung des Daten- und Steuerpfads der Bilinearen-Interpolations-ASM. Dabei steuert die Control_FSM den Datenpfad über die Register Enables und durch die Multiplexer. Ebenso übernimmt sie die Steuerung der Eingangssignale. Durch den Delay-Block am Ausgang wird das Signal um 8 Takte verzögert, sodass beide Signale gleichzeitig durch das DONE-Signal der FSM weitergegeben werden.

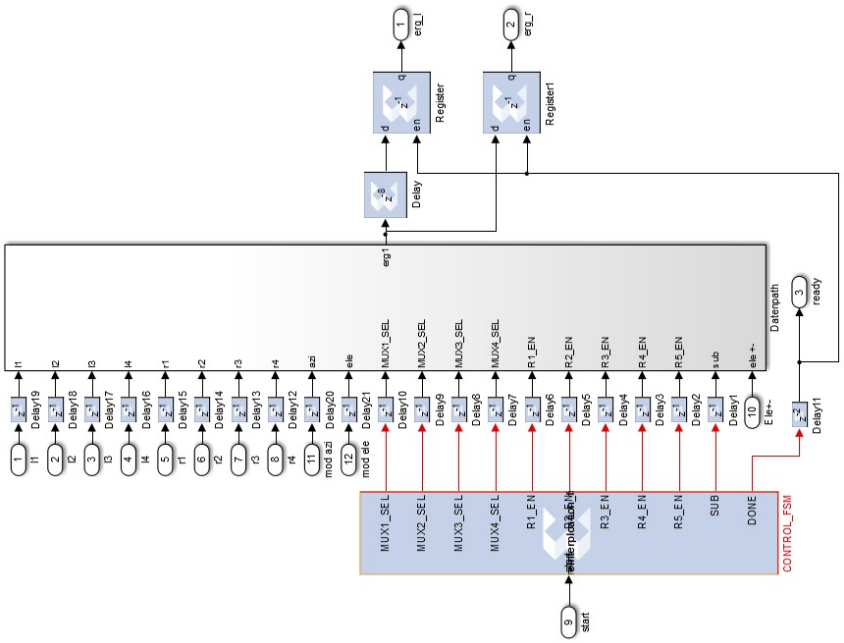


Abbildung 21: Kopplung des Daten- und Steuerpfads der Bilinearen-Interpolations-ASM

7.3 HRTF-ITD-Filter

Die Laufzeitdifferenzen (ITD) zwischen dem linken und rechten Ohr werden auf die Ergebnisse aus der bilinearen Interpolation angewendet. Dazu ist zuerst die Differenz zwischen beiden Ohren ermittelt worden. In meiner Bachelorarbeit habe ich die Differenz zum einen mittels des Kosinussatzes errechnet [Hem11]. Zum anderen wurde die Laufzeitdifferenz auch aus den HRTF-Messreihen errechnet. Beide Methoden zeigten die gleichen Ergebnisse. Die maximale interaurale Laufzeitdifferenz ITD beträgt 0,63 ms. Sie hängt von der Entfernung von Ohr zu Ohr ab. Da jeder Mensch eine andere Kopfform hat, ändert sich auch dieser Faktor.

Für die Umsetzung der ITD ist auf den Einsatz von Multiplizierern verzichtet worden, da diese nicht in ausreichender Anzahl auf dem FPGA des Entwicklungsboards zur Verfügung stehen. Aus diesem Grund wurde für jeden Kanal ein adressierbares Schieberegister gewählt. In diesem werden die Ergebnisse aus der bilinearen Interpolation zwischengespeichert. Da beide davorliegenden Stufen jeweils nur im Azimut-Wertebereich bis 180° arbeiten, sind bei einem Winkel größer 180° die Kanäle vertauscht. Dieses wurde durch einen Vergleich des Azimut mit 180° und zwei Multiplexern realisiert. Die Abbildung 22 zeigt den Aufbau des HRTF-ITD Moduls.

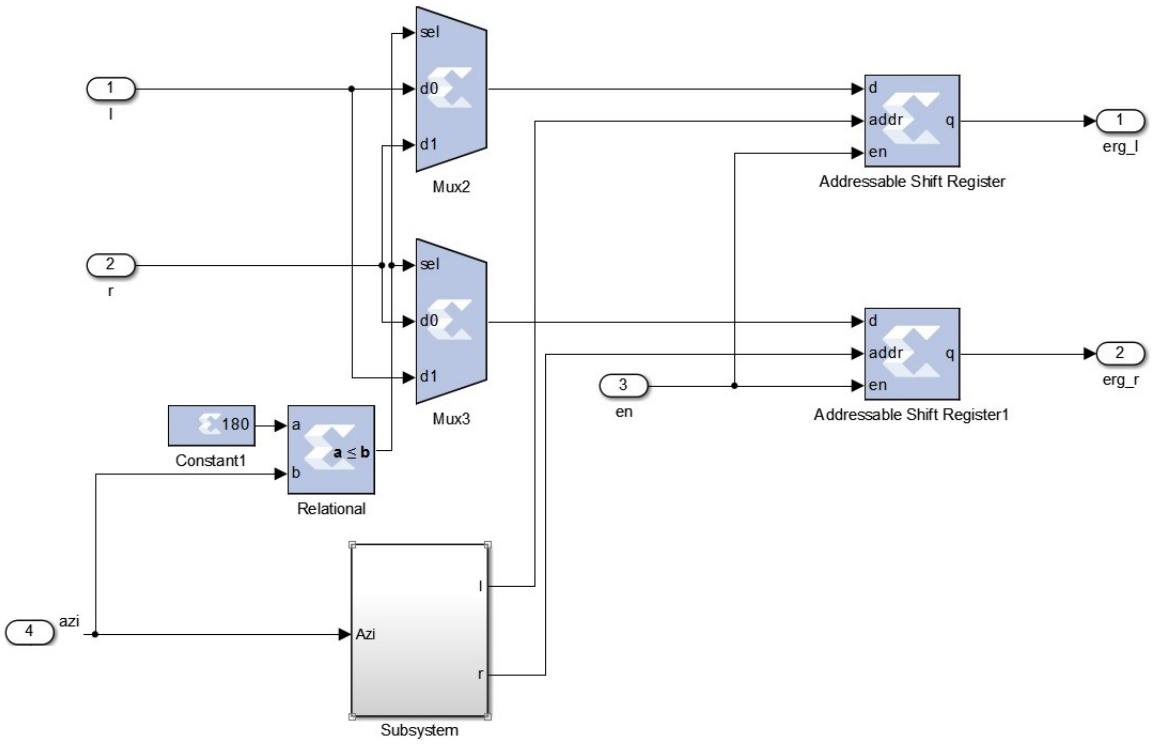


Abbildung 22: Aufbau des HRTF-ILD-Moduls mit Vertauschung der Kanäle und dem Laufzeitdifferenzen-Adressgenerators

Durch die vom Laufzeitdifferenzen-Adressgenerator (Abbildung 24) erzeugten Adressen werden die entsprechenden Audiosamples ausgegeben. Dazu wurden 45 Stützstellen festgelegt. Zu jeder dieser Stützstellen wurde die entsprechende Zeitdifferenz berechnet [Sen03]. Diese wird dem System durch einen RAM-Block zur Verfügung gestellt.

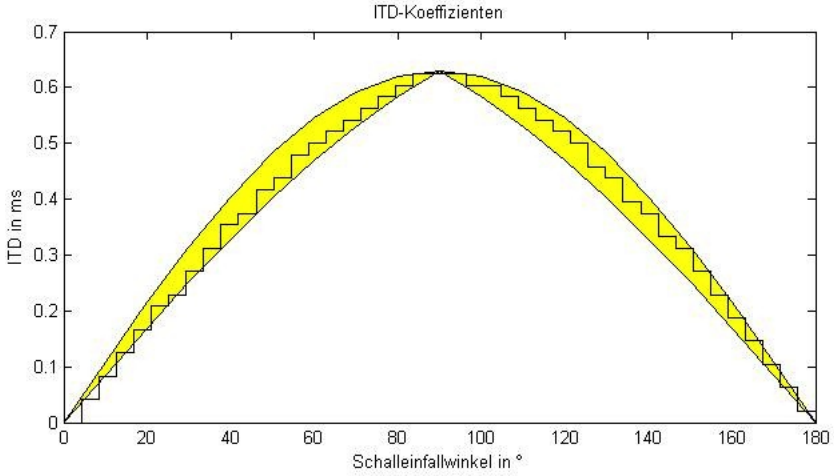


Abbildung 23: Gelb dargestellt berechneter ITD-Bereich in ms, schwarz die ITD-Koeffizienten

Der Zugriff auf den entsprechenden ITD-Koeffizienten geschieht durch eine Schiebeaktion auf dem Azimut. Dabei darf der Azimut nicht größer als 180° sein. Winkel größer 180° werden von 360° subtrahiert. Der so aus dem Speicher entnommene Koeffizient wird als Zeitverzögerung beim Zugriff auf das entsprechende Schieberegister verwendet.

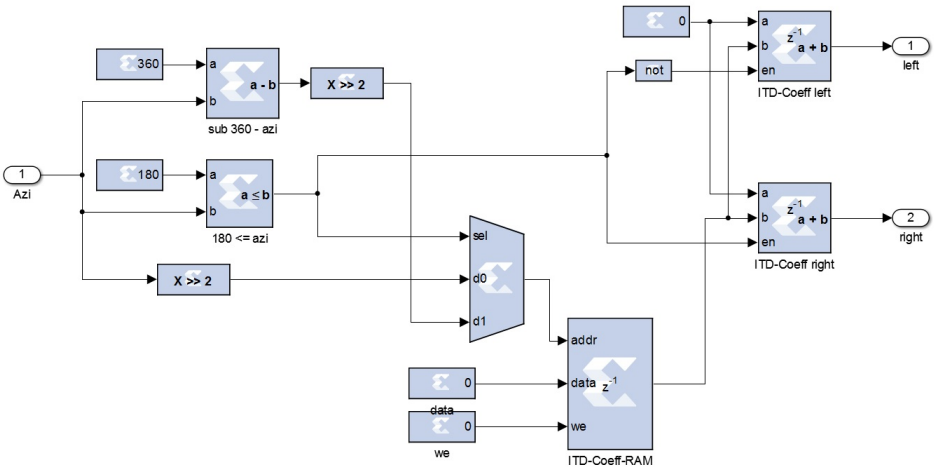


Abbildung 24: Aufbau des Laufzeitdifferenzen-Adressgenerators des ITD-Moduls

7.4 HRTF-Filter Testergebnis

In diesem Kapitel wird der Gesamttest des zuvor beschriebenen HRTF-Filters aufgezeigt. In der ersten Grafik ist der Azimut dargestellt. Dieser zählt in Schleife von 0° bis 360°.

Die untere Grafik aus Abbildung 25 zeigt die HRTF-ILD-Stützstellen-Ergebnisse an. In diesem Fall wurden die Filterkoeffizienten so ausgewählt, dass diese immer der Winkelposition entsprechen. Die Änderung der Filterkoeffizienten dient alleine der besseren Übersicht. Zudem sind die Filterkoeffizienten aller drei Elevationsebenen identisch. Je nach Winkelposition werden die beiden benachbarten HRTF-ILD-Stützstellen-Ergebnisse aus der entsprechen Elevationsebene weitergegeben. Bei einem Azimut größer 180° werden die Stützstellen rückläufig ausgegeben.

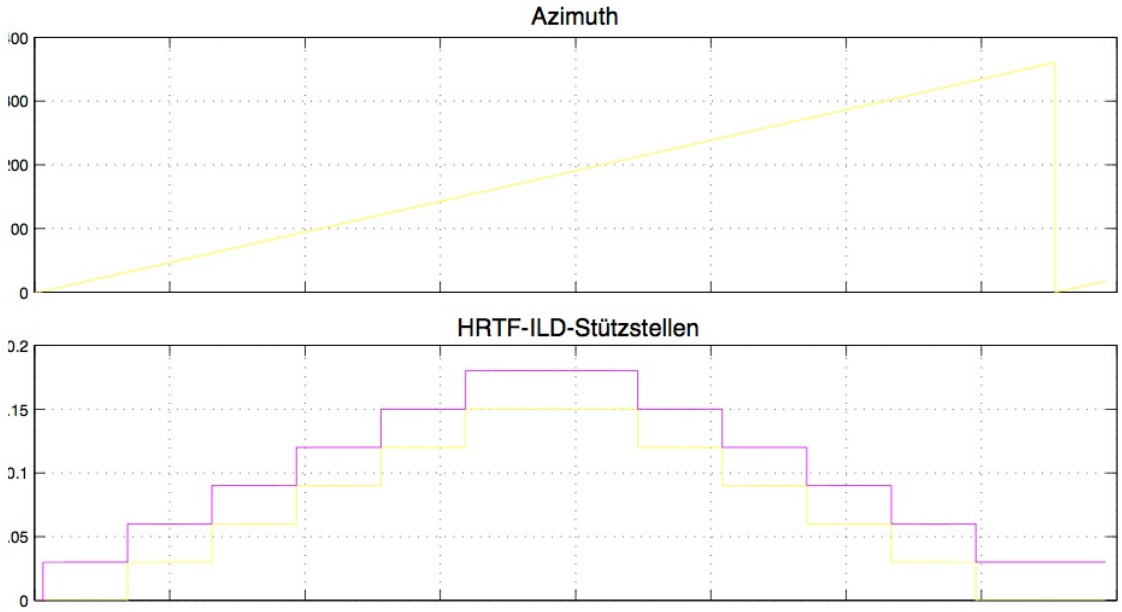


Abbildung 25: HRTF-ILD-Filterergebnis. Anhand des Azimut wird die HRTF-ILD-Filterung ausgegeben.

Die Abbildung 25 zeigt, dass der HRTF-ILD-Filter das gewünschte Ergebnis erzeugt. Die beiden benachbarten Filter-Ergebnisse werden ausgegeben. Bei einer Überschreitung des Stützstellen-Abstandes werden die nächsten beiden HRTF-ILD-Stützstellen-Ergebnisse ausgegeben.

Die Ergebnisse des HRTF-ILD-Filters werden in die Bilinearen-Interpolations-ASM weitergegeben und dort weiterverarbeitet. In der Abbildung 26 wird zuerst der Interpolationsfaktor dargestellt. Dieser gibt die prozentuale Verteilung zwischen beiden Stützstellen an. Dieser Interpolationsfaktor bewegt sich immer zwischen 0 und 29. Bei einem Winkel größer 180° passt er sich den HRTF-ILD Stützstellen an, indem er rückläufig ist.

In der zweiten Grafik aus Abbildung 26 ist das Endergebnis aus der Bilinearen-Interpolations-ASM dargestellt. Die Darstellung des zweiten Kanals wird durch den ersten Kanal überdeckt. Das Ergebnis stimmt mit den HRTF-ILD-Stützstellen überein und beschreibt das gewünschte Verhalten.

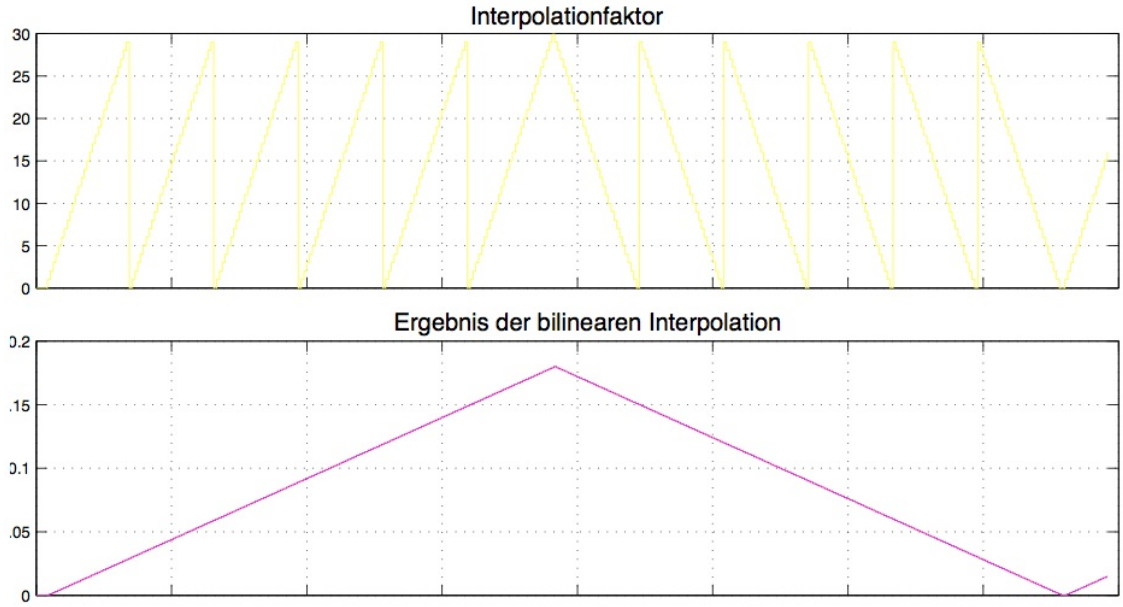


Abbildung 26: Bilinearer Interpolationsfaktor und Ergebnis von der Interpolations-ASM

Die Grafik 27 zeigt das Gesamtergebnis des HRTF-Filters. Dabei sind die Ergebnisse der Interpolation durch den ITD-Filter verarbeitet worden. Zuerst wird der linke Kanal (gelb) entsprechend der Laufzeitdifferenz verzögert. Ab einem Winkel von 180° wird der rechte Kanal (lila) verzögert.

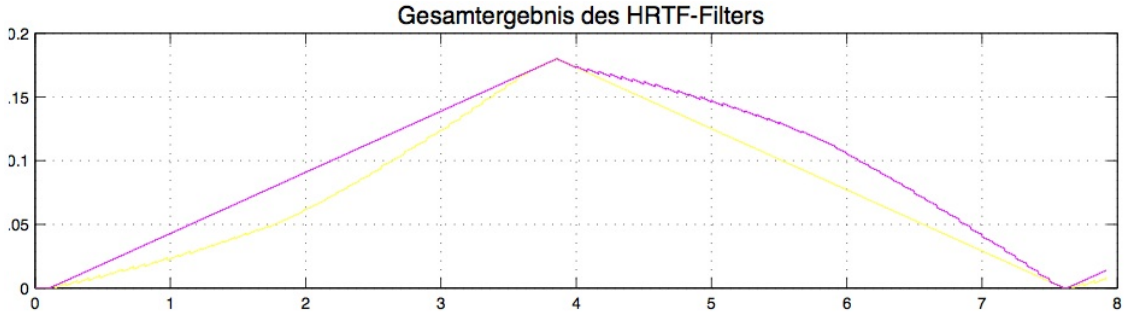


Abbildung 27: HRTF-Filterergebnis. Erkennbar ist die Verzögerung aus dem HRTF-ITD-Filter

7.5 Erweiterung zur Filterung mehrerer virtuelle Schallquellen

Eine Anforderung an das System ist es, mehrere Schallquellen parallel zu simulieren. Der einfachste Ansatz ist es, mehrere der beschriebenen HRTF-Filter in das SoC zu integrieren. Mit diesem Ansatz lassen sich rein theoretisch bis zu 16 Audioquellen verarbeiten. Die Begrenzung auf 16 Filter hängt mit der maximalen Anzahl von FSL-Schnittstellen zusammen [Xil12a]. Ebenso werden für jeden dieser eingebundenen Filter immer alle Koeffizienten-Filtersätze eingebunden. Es handelt sich immer um dieselben Koeffizientensätze. Aus diesem Grund wurde ein anderer Ansatz gewählt. Die HRTF-Filter werden zu einem großen Gesamtsystem zusammengeschlossen. Dadurch teilen sie sich die Filterkoeffizienten, welches eine Ressourcenreduktion zur Folge hat. Außerdem braucht das Gesamtsystem nur eine Schnittstelle, über welche alle Informationen weitergegeben werden. Der Aufbau des Gesamtsystems ist in Abbildung 28 dargestellt.

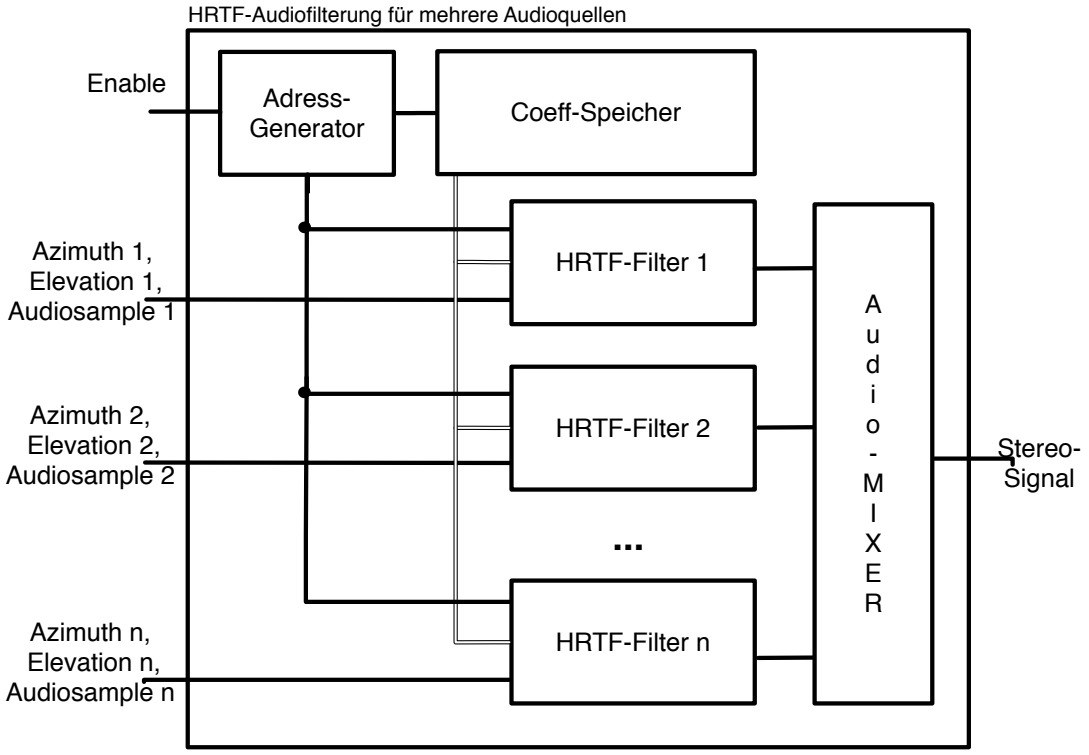


Abbildung 28: HRTF-Filteraufbau zur Filterung mehrerer Audioquellen mit nur einem Koeffizientenspeicher.

Die Steuerung übernimmt der Adressgenerator. Nach einem Enable erzeugt dieser die Koeffizientenadressen sowie die Adressen für den einzelnen Audiospeicher und weitere Steuersignale. Jeder HRTF-Filter ist mit allen Koeffizientenspeichern verbunden. In einem Modul werden anhand des Azimut und der Elevation die entsprechenden Filterkoeffizienten ausgewählt und für die FIR-Filter bereitgestellt.

In dieser Arbeit wurden 8 HRTF-Filter-Instanzen in das Gesamtsystem implementiert. Der Audio-Mixer zum Zusammenfügen der einzelnen Filterergebnisse zu einem Gesamtergebnis wird in Kapitel 7.6 beschrieben.

7.6 Zusammenmischen aller Audioquellen zum Gesamtergebnis

Wie im Kapitel 7.5 beschrieben, wurde für jede virtuelle Audioquelle eine eigene Filterinstanz eingebunden. Jede dieser Filterinstanzen erzeugt für beide Kopfhörerseiten die Signale der virtuellen Schallquelle. Um mehrere dieser virtuellen Schallquellen über die Kopfhörer einzuspielen, sind alle Signale zusammenzuführen. Die Abbildung 29 zeigt das Zusammenführen der Audiosignale eines Kanals.

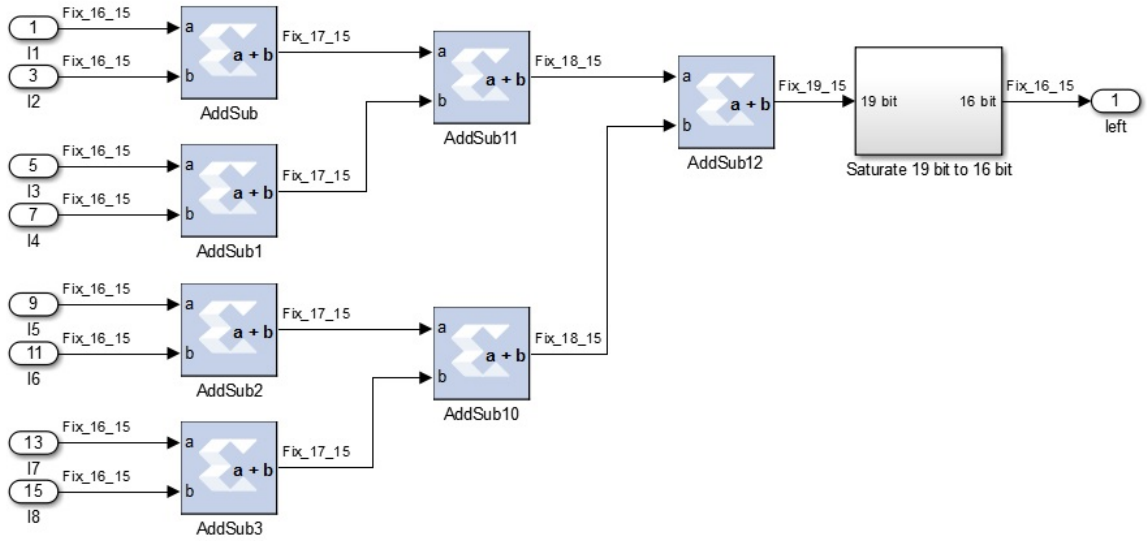


Abbildung 29: Audio-Mixer, um mehrere virtuelle Audioquellen zusammenzufügen, am Beispiel des linken Kanals, mit Sättigungsmodul

Bei 8 virtuellen Schallquellen sind 3 Additionsebenen erforderlich. In jeder dieser Additionsebenen wird das Ergebnis um 1 Guard-Bit verbreitert, um den vollen Wertebereich der Addition aufzunehmen. Das endgültige Signal wird durch eine Sättigungseinheit, die in Kapitel 7.1 beschrieben wurde, verarbeitet. Dabei wird der Wertebereich auf FIX16_15 begrenzt. Ist das Eingangssignal größer oder kleiner als der Wertebereich, wird eine Sättigung durchgeführt.

7.7 Anpassung der FSL-Schnittstelle für den HRTF-Filter

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-Schnittstelle 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. [Xil10b]

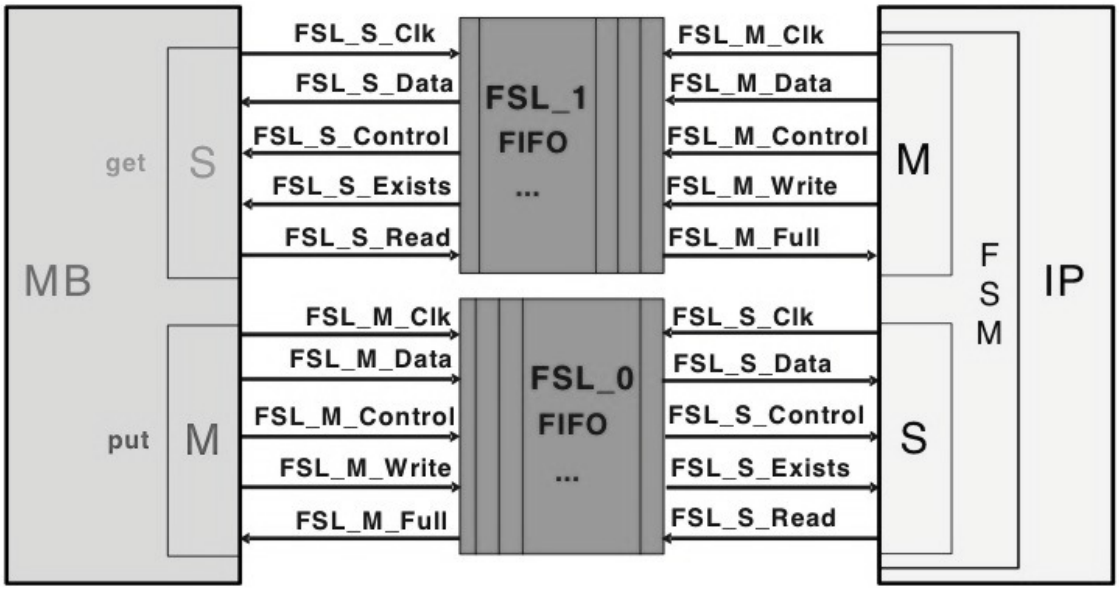


Abbildung 30: FSL-Schnittstelle zwischen MicroBlaze und IP-Core [**?**]

Mithilfe des "Create or Import Peripheral"-Assistenten wurde ein neuer IP-Core mit Fast Simplex Link (FSL) Interface erzeugt.

Das entwickelte Simulink Model wird mithilfe des System Generators als NGC Netlist für den Virtex5 FPGA erzeugt. Die FPGA clock period ist dabei auf 10 ns eingestellt. Die Multirate Implementation ist auf Expose Clock Ports eingestellt. Da das Modell ohne Multirate aufgebaut ist, wird nur eine CLK an das System angebunden. Die so erzeugten Dateien wurden in den zuvor erzeugten IP-Core abgelegt. Dabei werden alle NGC-Dateien im Unterordner netlist gespeichert. Die zwei erzeugten VHDL-Dateien werden im Ordner hdl/vhdl abgelegt. Daraufhin wurde die MPD-Datei und POA-Datei des IP-Cores angepasst.

Bei der MPD-Datei wurde eine Option hinzugefügt. Diese Option gibt an, dass das erzeugte Model aus VHDL- und NGC-Dateien besteht

```
OPTION STYLE = MIX
```

In der POA-Datei werden die beiden erzeugten VHDL-Dateien eingetragen. Dabei ist zu beachten, dass diese vor der Top-Entity aufgelistet werden.

```
lib hrtf_8filter_v1_00_c hrtf_8filter_v6_2_100mhz vhdl
lib hrtf_8filter_v1_00_c hrtf_8filter_v6_2_100mhz_mcw vhdl
lib hrtf_8filter_v1_00_c hrtf_8filter vhdl
```

Um alle NGC-Dateien bekannt zu machen, ist im Unterordner data eine BBD-Datei zu erzeugen. In dieser Datei sind alle erzeugten NGC-Dateien nur durch Kommas getrennt aufgelistet.

Als Nächstes wurde die Datei “hrtf_8filter.vhd“ aus dem erzeugten IP-Core angepasst. Dazu wird der HRTF-Filter als Komponente hinzugefügt.

```
component hrtf_8filter_v6_3_100mhz_mcw is
port (
audio_1_azimuth: in std_logic_vector(8 downto 0);
audio_1_elevation: in std_logic_vector(6 downto 0);
audio_1_input: in std_logic_vector(15 downto 0);
audio_2_azimuth: in std_logic_vector(8 downto 0);
audio_2_elevation: in std_logic_vector(6 downto 0);
audio_2_input: in std_logic_vector(15 downto 0);
...
audio_8_azimuth: in std_logic_vector(8 downto 0);
audio_8_elevation: in std_logic_vector(6 downto 0);
audio_8_input: in std_logic_vector(15 downto 0);
clk_1: in std_logic; -- clock period = 10.0 ns (100.0 Mhz)
compass_azi: in std_logic_vector(8 downto 0);
compass_elevation: in std_logic_vector(6 downto 0);
enable: in std_logic;
finish: out std_logic;
left: out std_logic_vector(15 downto 0);
right: out std_logic_vector(15 downto 0)
);
end component;
```

Nachdem die Komponente hinzugefügt wurde, wird diese mit der FSL-Schnittstelle verbunden. Dazu wurde in der Top-Entity eine Finite-State-Maschine entwickelt. Diese verbindet die FSL-Schnittstelle mit dem HRTF-Filter. Die Abbildung 31 zeigt den dafür entwickelten Automaten.

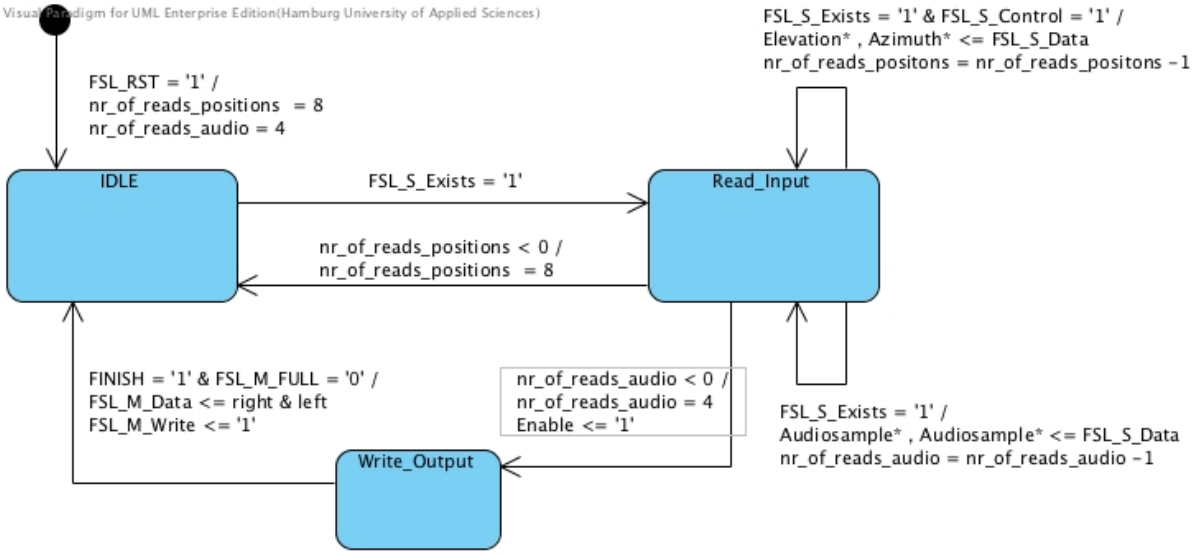


Abbildung 31: Finite-State-Maschine: Verbindung zwischen dem FSL-Interface und dem HRTF-Filter.

Nach einen FSL_RESET = ’1’ wird der Startzustand IDLE erreicht. Solange kein Datum im FIFO der FSL Slave Schnittstelle anliegt, bleibt dieser Zustand aktuell. Durch das Ereignis FSL_S_EXISTS = ’1’ signalisiert das FSL Interface, dass ein neues Datum anliegt und der Automat wechselt in den Zustand Read_Input. Im Zustand Read_Input wird das FSL_S_Control Signal überprüft. Ist das Signal gesetzt, handelt es sich um ein Positionssignal. Es werden immer alle Positionen nacheinander übertragen. Durch die Variable nr_of_reads_positions ist bekannt, um welches Datum es sich handelt. Nach jedem Lesen aus dem FIFO wird nr_of_reads_positions-1 gerechnet. Wenn alle Positionsdaten übermittelt wurden, wechselt der Automat in den Zustand IDLE und wartet auf neue Daten. Wenn im Read_Input kein FSL_S_Control Bit gesetzt ist, handelt es sich um Audioinformationen. Jedes Datum beinhaltet 2 Audiosamples, welche durch nr_of_reads_audio zugewiesen werden. Nach jedem Lesen aus dem FIFO wird nr_of_reads_audio runtergezählt. Wenn alle Audiosamples übertragen wurden, wird in den Zustand Write_Output gewechselt. Zudem wird die Audiofilterung durch das Setzen des Enable-Signals gestartet. Im Zustand Write_Output wird gewartet bis der HRTF-Filter durch das Setzen des Finish-Signals signalisiert, dass die Berechnung beendet ist. Nach der Überprüfung des Ausgangs-FIFOs wird das Ergebnis aus der Filterung in den FIFO geschrieben und steht somit dem MircoBlaze zur Verfügung.

8 Entwicklung des Embedded-Systems (SoC)

8.1 Zusammenstellung der Hardware-Module des SoCs

In diesem Kapitel wird der Aufbau der zu entwickelten SoC-Plattform erläutert. Dabei zeigt Abbildung 32 die SoC-Plattform.

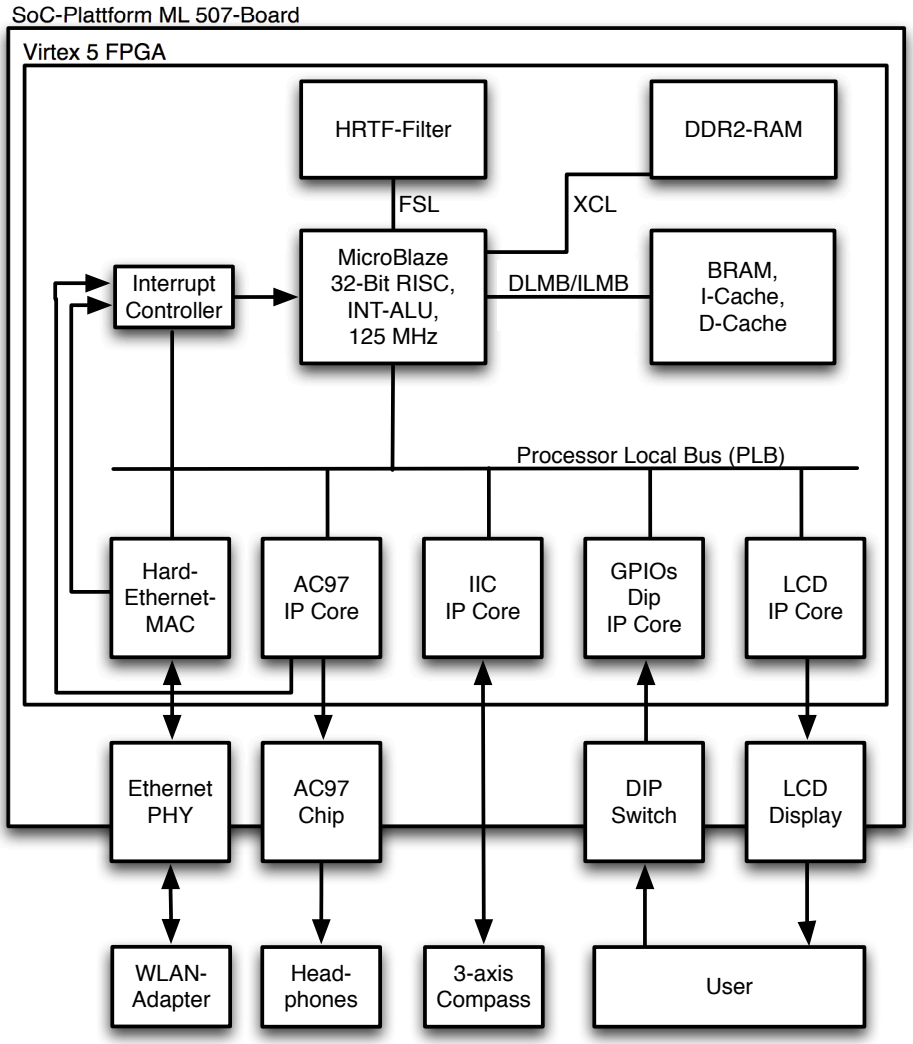


Abbildung 32: SoC-Architektur im Virtex5 FPGA mit MicroBlaze Softcore-Prozessor und allen Komponenten

Der MicroBlaze wird als Single Prozessor System eingestellt und mit einem Takt von 100MHz betrieben. Als Speicher wurde der 256 MB große DDR2-RAM ausgewählt, welcher zusätzlich mit einem Daten- und Instruktions-Cache erweitert wurde. Diese beiden

Caches sind jeweils ein 32kB großer BRAM-Block. Als Busschnittstelle wurde der Processor Local Bus (PLB) gewählt. Über die PLB Schnittstelle wird der MicroBlaze mit den restlichen Hardware-Komponenten verbunden. Dazu zählen:

- LCD IP-Core: Zur Statusausgabe des Systems über das LCD-Display [Mic09]
- GPIO IP-Cores: für LEDs, DIP für eine Benutzerinteration, sowie Pins zum Messen
- RS232 IP-Core: Anschluss für den 3-Achsen-Kompass
- Hard-Ethernet-MAC IP-Core: Kommunikationsschnittstelle mit dem Server
- AC97 IP-Core: Audioausgabe [DEV05]
- Interrupt Controller IP-Core: Weiterleitung der Interrupts von AC97 und der Ethernet-Schnittstelle an den MicroBlaze

Um den Status des Systems auszugeben, wird das auf dem Board vorhandene LCD-Display eingesetzt. Dazu wurde der in meiner Bachelorarbeit entwickelte IP-Core eingebunden. Wie auch bei den anderen IP-Cores wird dieser ebenfalls über den PLB-Bus angebunden. Der LCD-Port wird mit einer neuen externen Connection verbunden. Diese wird daraufhin mittels der UCF-Datei an das LCD-Display verbunden. (Siehe Anhang C)

Die drei Standard-GPIO-Module LEDs, DIP-Switches und Oszilloskop wurden hinzugefügt und über die PLB-Schnittstelle mit dem MicroBlaze verbunden. Die Push-Buttons und DIP-Switches dienen dem Benutzer der SoC-Platform zur Steuerung des Systems. Das Oszilloskop dient zur Messung von Systemfunktionen.

Die RS232 Schnittstelle wird mit einer Bitrate von 19200 bps und 8 Datenbits konfiguriert. Diese Bitrate resultiert aus der maximal angegebenen Bitrate des 3-Achsen Kompass der Firma Oceanserver. [Oce10]

Um die Ethernet-Schnittstelle einzubinden, wurden der Hard_Ethernet_MAC und der HARD_Ethernet_MAC_fifo hinzugefügt. Diese IP-Cores befinden sich im IP Catalog im Unterordner Communication High-Speed. Mittels Add IP werden sie dem MicroBlaze hinzugefügt. Beide werden über den PLB Bus angeschlossen. Über Configer IP werden beide für das System angepasst. Zu den Anpassungen gehört die Vergrößerung des Eingangsbuffers auf ca. 16 kB. Dadurch ist sichergestellt, dass mehrere Pakete zwischengespeichert werden, bevor diese durch die Software verarbeitet werden. Die genauen Anpassungen sind im Anhang C beigefügt.

Das Virtex5 ML507 Entwicklungsboard besitzt einen AC97-Codec-Baustein [DEV05].

Dieser wurde in meiner Bachelorarbeit ausführlich beschrieben [Hem11]. Der AC97 IP-Core wurde dem SoC hinzugefügt und über den PLB Bus mit dem MircoBlaze verbunden. Daraufhin wurde der IP-Core mittels Configer IP eingestellt. Der AC-97 Codec wird in diesem System nur Audio-Samples abspielen und keine Audio-Samples aufnehmen. Dazu wurde C_PlayBack = 1 und C_Record = 0 gesetzt. Das 3D-Audio-SoC ist ein Echtzeitaudioverarbeitungssystem, aus diesem Grund sind keine großen Zwischenspeicher erforderlich. Durch das Setzen einer 0 im C_USE_BRAM wird dem IP-Core mitgeteilt, dass er anstatt großer BRAM-Blöcke das System mit einem kleinen Schieberegister aufzubauen hat. Dieses Schieberegister speichert je nach Benutzung bis zu 16 Audiosamples zwischen, bevor diese ausgegeben werden. Die Benutzung dieses Zwischenspeichers wird durch das Interrupt-Level festgelegt. In diesem System wird das C_intr_level = 1 gesetzt. Bei dieser Einstellung erzeugt der IP-Core Interrupts, wenn der Zwischenspeicher zur Hälfte gefüllt ist. Somit ist sichergestellt, dass immer 8 Audiosamples als Buffer vorhanden sind. Daraufhin werden die Ports AC97Reset_n, Bit_Clk, SData_In, SData_Out und Sync jeweils mit einer neuen externen Connection verbunden und an die jeweiligen Pins des AC97-Bausteins angeschlossen. Die UCF-Datei wurde ebenfalls angepasst. Siehe Anhang C.

Der im Kapitel 7 entwickelte HRTF-Filter IP-Core wird über zwei unidirektionale FSL-Busse mit dem Embedded System verbunden. Zuvor wird der MicroBlaze mit entsprechenden FSL-Schnittstellen versehen. Diese werden mittels des XPS Core Config-Assistents im Unterpunkt Bus eingestellt. Dafür wird als Stream Interface die FSL-Schnitstelle ausgewählt und die Anzahl der Stream Links wird auf 1 gesetzt. Der HRTF-Filter IP-Core wurde mit einem weiteren Assistenten (Hardware/Configure Coprocessor) an das System angeschlossen. Dabei fügt der Assistent auch die FSL-Master und Slave Interfaces hinzu und verbindet diese sofort mit dem IP-Core.

8.2 Mobilität der Ethernet-Schnittstelle

Um das Ziel der mobilen SoC-Plattform zu erreichen, wurde die Ethernet-Schnittstelle zu einer mobilen Schnittstelle erweitert. Dazu wurde der Wireless Adapter WNCE3001 der Firma Netgear verwendet. Dieser Adapter ist in Abbildung 33 dargestellt. Er verbindet alle ethernetfähigen Geräte drahtlos mit einem vorhandenen WLAN. Durch die Dual-Band-Technik bietet dieser Adapter die zurzeit maximale Performance. [net]

Abbildung 33: Netgear WNCE3001 verbindet die Ethernetschnittstelle der SoC-Plattform über ein WLAN-Netz mit dem 3D-Audio-Server

Der Adapter unterstützt zwei verschiedene Installationsmöglichkeiten. Zum einen über das Wi-Fi Protected Setup kurz WPS. Dieses Protokoll unterstützt einen einfachen Verbindungsaufbau zwischen dem Adapter und dem WLAN-Router. Zum anderen ist der Adapter auch über eine einfache grafische Oberfläche zu konfigurieren. Die so eingestellten Informationen werden gespeichert. Bei jedem Start des Gerätes wird eine Verbindung mit dem eingestellten WLAN aufgebaut.

Durch ein Ethernetkabel wird die Ethernet-Schnittstelle des Entwicklungsboards mit dem Adapter verbunden. Die Stromversorgung des Adapters wird über das Entwicklungsboard bereitgestellt. Dazu wird der bisher nicht verwendete USB-Anschluss benutzt.

9 Kommunikationsinterface zwischen Server und Client

9.1 Open Sound Control zum Informationsaustausch

Zum beidseitigen Informationsaustausch zwischen Komponenten wurde sich für das Open Sound Control Protocoll entschieden. Open Sound Control (OSC) ist ein Protokoll zur Kommunikation zwischen Computern, Sound-Synthesizern und anderen Multimediageräten. Das Protokoll ist für den Einsatz in modernen Netzwerken optimiert, sodass einige der Vorteile des Protokolls hohe Kompatibilität, Genauigkeit und Flexibilität sind. [MATC]

OSC nutzt zur Übertragung sogenannte OSC-Pakete. Die Netzwerkschicht ist für die Übertragung der Pakete selbst verantwortlich. So ist ein OSC Paket als UDP-Paket oder auch über eine TCP-Verbindung zu verschicken. Eine OSC Nachricht besteht aus einer Adresse (OSC Address Pattern), gefolgt von beliebig vielen Datentypen (OSC Type Tag Strings). Diese Adresse ist ein URL-Datenpfad, sie beginnen mit einem "/" gefolgt von einem OSC-String. Nachfolgend einige Beispiele für OSC-Nachrichten:

```
/3DAudio/compass/elevation "i" 32
/3DAudio/debug "s" Test-String
```

Informationen, wie die Position einer Schallquelle oder die aktuelle Kompassposition, werden im gesamten 3D-Audio-System über OSC-Nachrichten verbreitet.

9.2 Audio Streaming Protokoll

Es wurde nach einer Schnittstelle gesucht, die die Audiosignale für die SoC-Plattform bereitstellt. Dabei sind die Audioinformationen über die Ethernet- bzw. WLAN-Verbindung zu verschicken. Normalerweise würde dazu das Real-Time Transport Protocol (RTP) infrage kommen. Dieses ist ein Protokoll zur kontinuierlichen Übertragung von audiovisuellen Daten (Streams) über IP-basierte Netzwerke. Das Protokoll wurde erstmals 1996 im RFC 1889 [rfca] standardisiert und 2003 durch das RFC 3550 [rfcb] abgelöst.

Das Real-Time Transport Protocol unterstützt kein Streaming von Mono-WAV-Dateien mit einer Abtastrate von 48kHz. Die Audiodateien mithilfe der Umrechnung in ein anderes Format zu verschicken wäre zwar denkbar, würde jedoch erheblich Rechenaufwand auf dem SoC führen. Ebenso unterstützt RTP eine Vielzahl von Funktionen, die nicht für das 3D-Audio-System benötigt werden. Aus diesen Gründen wurde sich gegen das RTP entschieden.

Aus diesen Gründen wurde ein eigenes Audio Streaming Protokoll entwickelt. Es basiert auf einem Client-Server Aufbau und ist in Abbildung 34 dargestellt.

Abbildung 34: ASP-Übersicht mit Kommunikationskanälen

Dabei wird eine Kommunikationsschnittstelle zwischen beiden Seiten aufgebaut, sowie für jeden Audiostream ein Datenstrom von Server zum Client. Der Vorteil der getrennten Audiostreams ist, dass die SoC-Seite die Streams anhand des Ports erkennt und nicht auf den Paketinhalt zurückgreifen muss. Durch die Portbindung wäre es denkbar, mithilfe eines DMA-Controllers die Daten an eine beliebige Speicheradresse zu schreiben. Im besten Fall ist diese die Speicheradresse des entsprechenden Buffers, in dem die Audiosamples abzuspeichern sind. Damit werden die Audiodaten im Hintergrund dem System zur Verfügung gestellt.

Über den Kommunikationskanal wird die Initialisierung durchgeführt. Bei diesem Prozess werden alle relevanten Daten ausgetauscht. Dazu zählt zum Beispiel die Anzahl der

virtuellen Schallquellen, die der Server anfordert, sowie die Antwort des SoC mit den zur Verfügung stehenden Audioquellen und den Ports der einzelnen Audiostreams. Ebenso werden Nachrichten zur Steuerung der Streams verschickt. Dazu zählt unter anderem das Starten und Stoppen sowie das neue Anfordern von Samples durch das SoC. Ein Standardablauf dieses Protokolls ist in Abbildung 35 dargestellt.

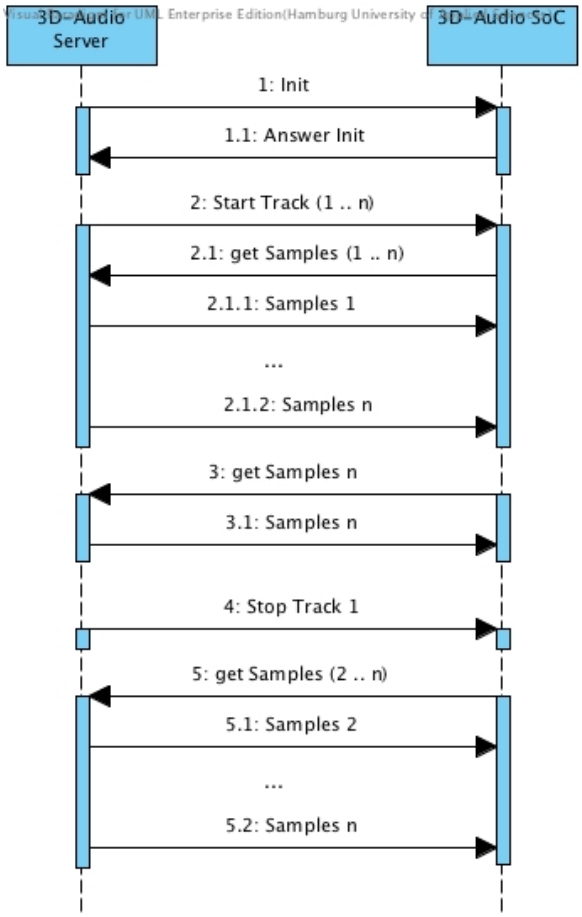


Abbildung 35: ASP Ablaufdiagramm

Zuerst wird eine Initialisierung durch den Server gestartet. Diese fordert beim SoC eine bestimmte Anzahl von Audioquellen an. Das SoC antwortet ihm mit der Anzahl der zur Verfügung stehenden Audioquellen und gibt den ersten Port für die Audiostreams an. Die Ports sind fortlaufend vergeben.

Durch die Start Track Nachricht mit entsprechender Bitmaske im Data Feld werden die entsprechenden Audiostreams gestartet. Daraufhin fordert das SoC mittels einer get Sample Nachricht die Audioinformationen an, welche durch den Server an die entsprechenden Ports

verschickt werden. Dass die Clientseite (SoC) die Audiodaten anfordert und diese nicht über eine einfache Push-Funktion vom Server verschickt werden, hat den Grund, dass auf dem SoC hierdurch keine großen Buffer erforderlich sind. Das SoC fordert immer wieder Daten an, wenn genügend freier Speicher im Audiobuffer der entsprechenden Streams vorhanden ist. Dadurch wird der Zwischenspeicher auf die performancestärkere Serverseite ausgelagert.

Wenn ein Audiostream zurzeit nicht gebraucht wird, lässt sich dieser durch eine Stop-Track Nachricht mit entsprechender Nummer abschalten. Somit werden keine neuen Daten mehr vom SoC für diesen Stream angefordert. Durch das erneute Senden einer Start Track Nachricht fordert das SoC wieder Daten für den Audiostream an. Durch diesen Mechanismus lässt sich eine unnötige Datenübertragung vermeiden.

Der ASP-Header ist in Abbildung 36 dargestellt. Er besteht aus einer 4-Bit großen Versionsnummer. Danach folgt eine 12-Bit lange Sequenz Nummer, welche bei jedem Verschicken einer Nachricht hochgezählt wird. Sowie eine 16-Bit große MSG ID, welche angibt um welche Art von Nachricht es sich handelt. Darauf folgen 4 Byte mit Daten. Siehe Abbildung 36.

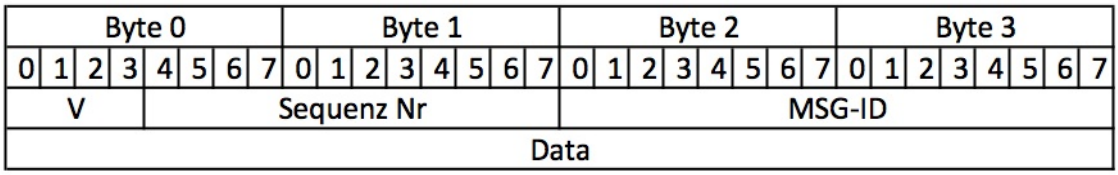


Abbildung 36: ASP Header

Tabelle 2 zeigt die zurzeit genutzten MSG-IDs und deren Bedeutung, sowie die Bedeutung der vier Datenbytes.

MSG-ID:	Beschreibung	Daten Byte
0	INIT	Anzahl von Audiostreams die angefordert werden
1	Antwort INIT	Anzahl von Audiostreams die verfügbar sind und der erste Port für Streams (Ports sind fortlaufend)
2	Start Track	ein gesetztes Bit startet den entsprechenden Audiostream
3	get Samples	ein gesetztes Bit fordert Daten für einen entsprechenden Audiostream an
4	Stop Track	ein gesetztes Bit stopt den Audiostream

Tabelle 2: MSG-IDs und deren Bedeutung

10 Software-Routinen für den MicroBlaze Prozessor

In diesem Kapitel wird die Entwicklung der C-Anwendung für das Embedded System beschrieben. Die Entwicklung wurde unter Zuhilfenahme des Xilinx Software Development Kit (SDK) durchgeführt.

10.1 3D-Audio-System Software-Application-Projekt

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

Für dieses wurde ein Board Support Package (BSP) 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, werden die entsprechenden Adressen der Hardware-Module in der Datei “xparameters.h“ abgelegt. Diese Datei enthält Defines und die Adressen jeder Hardware-Komponente.

Des Weiteren wurden Treiber für bestimmte Hardware-Komponenten erzeugt. In diesem Projekt wird ein zusätzlicher Treiber für das Ethernet-Interface eingebunden. Dabei handelt es sich um den Lightweight TCP/IP Stack (lwIP). lwIP ist eine leichtgewichtige Implementierung des TCP/IP Protokolls für 32-Bit Systeme, welche ursprünglich am Swedish Institute of Computer Science entwickelt wurde [Dun01]. Xilinx hat den Stack für ihre Systeme angepasst und stellt diesen über das EDK zur Verfügung. Auf diese Weise lässt sich lwIP vor dem Kompilieren, durch Änderungen an den Konfigurationsdateien (z.B. Puffergrößen und/oder unterstützte Protokolle), für die Anforderungen des Systems anpassen.

Der lwIP Stack stellt zwei verschiedene APIs zur Verfügung. Es lässt sich als RAW-API oder als Socket-API konfigurieren. Die RAW-API arbeitet nur mittels Callback-Funktionen, somit lassen sich Netzwerkanwendungen mit höherem Datendurchsatz entwickeln. Bei der Socket-API wird jede Netzwerkschicht separat behandelt, wodurch mehr Verwaltungsaufwand entsteht. Das führt zu einer besseren Plattformunabhängigkeit im Vergleich zur RAW-API.

Der Treiber wird in diesem Projekt als RAW-API in das BSP hinzugefügt. Der TCP-Treiber von lwIP wird deaktiviert, da dieser in dem entwickelten Projekt nicht eingesetzt wird. Die Anzahl der maximalen aktiven UDP-Verbindungen wird auf 32 Verbindungen eingestellt. Ansonsten bleiben alle Einstellungen auf den Default-Parametern.

Der Standard Input und Output wird mit dem MicroBlaze Debug Modul verbunden. Somit werden die Systemausgaben über die Schnittstelle an den Entwicklungscomputer

versendet. Eine weitere Einstellung des Projektes ist die Optimierungsstufe. Diese lässt sich über die Projekteigenschaften einstellen. Der im SDK eingebundene gcc-Compilier unterstützt verschiedene Optimierungsstufen. In diesem Projekt wurde die Optimierungsstufe -O2 eingeschaltet, um die maximale Performance zu erzielen.

10.2 Initialisierung der Hardware durch den MircoBlaze

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

In ihr werden zuerst die beiden Caches aktiviert. Dazu stellt das erzeugte Board Support Package die Funktionen Xil_ICacheEnable() und Xil_DCacheEnable() zur Verfügung. Durch das Aufrufen dieser Methoden werden die Caches für die Daten und Instruktionen des Embedded Systems aktiviert.

Durch die Methoden LCDInit() und LCDOn(), die in der Datei “lcd.c“ implementiert wurden, lässt sich das LCD-Display initialisieren und aktivieren. Das LCD-Display wird im System genutzt, um den aktuellen Betriebszustand auszugeben.

Durch die Methode ac97_init() wird der AC97-Baustein und der dazugehörige IP-Core aktiviert und eingestellt. Diese Methode resettet zuerst den AC97-Codec-Baustein und wartet bis dieser wieder bereit zum Arbeiten ist. Anschließend werden alle nicht erforderlichen Eingänge, Ausgänge sowie nicht gewünschte Funktionen ausgeschaltet. Der gewünschte Line-Out Ausgang wird freigeschaltet und die Abtastrate des Bausteins auf 48 kHz gesetzt. Die Einstellungen werden mittels der Funktion XAC97_WriteReg(baseaddr,register offset, data) an die entsprechenden Register des Bausteins weitergeben. Nach jeder Änderung wird durch die Funktion XAC97_AwaitCodecReady() gewartet bis der Baustein wieder aufnahmebereit ist. Beim Ändern der Abtastrate ist zu beachten, dass zuvor alle AD-Wandler bzw. DA-Wandler ausgeschaltet werden, bevor die Abtastrate geändert wird. Alle Register und sonstigen Einstellungen des AC97-Codec-Bausteins lassen sich dem Datenblatt entnehmen. [DEV05]

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

Die Ethernet-Schnittstelle wird durch die Initialisierung von lwIP mit der Methode lwIP_init() aus dem Treiber im BSP durchgeführt. Mittels der Funktion xemac_add() wird das Netzwerkinterface ausgewählt und eingestellt. Zu den Einstellungen gehört zum Beispiel die IP-Adresse, Netzmaske und die MAC-Adresse. lwIP unterstützt unter anderem auch einen DHCP. Auf diesen wird in dieser Arbeit verzichtet und das SoC erhält eine feste

IP-Adresse.

Zum Schluss der Initialisierung des SoCs werden die Interrupts freigegeben. Die Interrupt-Controller Baseadresse sowie die jeweilige Interrupt-ID werden aus der Header-Datei “xparameters.h“ entnommen. Die AC97-ISR wird im Kapitel 10.4 beschrieben. Diese ISR wird durch den Befehl XIntc_RegisterHandler() angebunden. Da der Interrupt des AC97-IP-Core standardmäßig ausgeschaltet ist, wird dieser mit der Methode ac97_in_interrupt_en() freigeschaltet. Diese Funktion gibt den Interrupt für das Playback-FIFO frei. Die Interrupts der Ethernetschnittstelle wurden bei der Initialisierung von lwIP angebunden. Dadurch sind alle entsprechenden ISR mit den Interrupts verbunden. Durch die Funktion XIntc_EnableIntr() werden die Interrupteingänge des Interruptcontrollers freigegeben. Der Interruptausgang des Interruptcontrollers wird durch die Methode XIntc_MasterEnable() aktiviert. Durch den Funktionsaufruf microblaze_enable_interrupts() wird der Interrupt des MicroBlaze aktiviert.

10.3 Main-Routine des MircoBlaze

Nach der Initialisierung der SoC-Plattform, wird die Software des MARA-Clients vorbereitet. Dazu wird die Initialisierung von Open Sound Control (OSC) durchgeführt. OSC ist ein Protokoll, welches den Informationsaustausch zwischen Server und Client durchführt. Das Protokoll wurde im Kapitel 9.1 geschrieben.

Außerdem werden Datenstrukturen zum Verwalten und Speichern der Audiodaten erzeugt. Für jede Audioquelle wird ein Ringbuffer erzeugt. Ebenso wird zu jeder Schallquelle die Position aufgrund des Azimuts und der Elevation abgespeichert. Des Weiteren werden mehrere Steuersignale angelegt.

Die Audioquellen werden über das entwickelte Audio Streaming Protokoll übermittelt. Dieses wurde in Kapitel 9.2 vorgestellt. Die Initialisierung wird durch die Methode init_AudioStreaming durchgeführt. Nach der Initialisierung der Hardware und Software arbeitet die SoC-Plattform die Main-Routine in einer Dauerschleife ab. Der Ablauf dieser Endlosschleife wird in Ablaufdiagramm 37 gezeigt.

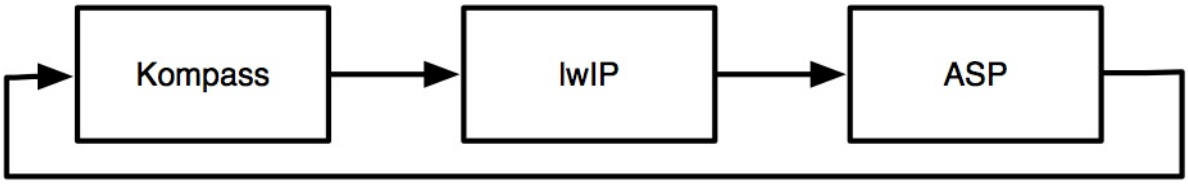


Abbildung 37: Ablaufdiagramm der Main-Routine

Bei jedem Durchlauf wird das Dip-Switch abgefragt. Die letzten beiden Dips sind für die Steuerung des Kompasses vorgesehen. Der achte Dip-Schalter aktiviert den Kompass. Wenn die Kompass-Steuerung aktiv ist, wird der Kompasswert abgefragt, ob eine neue Messung durchgeführt wurde. Ist dies der Fall, werden die Messergebnisse als neue Kopfposition übernommen. Durch das Setzen des siebten Dip-Schalters wird die aktuelle Winkelposition des Kompasses als neuer Nullpunkt festgelegt. Bei einer Änderung der Winkelposition des Kopfes wird diese an den HRTF-Filter weitergegeben und die Filterung sofort angepasst. Ebenso wird die neue Position mittels OSC an den Server geschickt.

Nach der Verarbeitung des Kompasses wird der lwIP Stack aufgerufen. Der Xilinx lwIP Treiber arbeitet im Interrupt Modus. Der Interrupt-Handler kopiert die empfangenen Pakete aus der MAC-Einheit in eine lwIP-Queue. Der Funktionsaufruf xemacif_input() überprüft die Queue auf empfangen Pakete und übergibt diese an die lwIP-Verarbeitung. Daraufhin verarbeitet lwIP das Paket und ruft die dazugehörige Receive-Methode auf.

Im Anschluss daran wird überprüft, ob ASP-Pakete empfangen wurden. Wenn dies der Fall ist, werden diese in den entsprechenden Audiozwischenspeicher gespeichert. Gleichzeitig wird überprüft, ob der Zwischenspeicher ein neues ASP-Paket aufnehmen kann. Ist dies der Fall, wird ein neues Paket angefordert.

10.4 Echtzeitaudioverarbeitung in der AC97-ISR

Der AC97-Codec arbeitet in diesem System mit der höchsten Interruptpriorität. Die Interrupteinstellung des AC97-IP-Cores wurde auf Level 1 eingestellt. Durch diese Einstellung erzeugt der IP-Core Interrupts, wenn der Output-FIFO nicht bis zur Hälfte gefüllt ist. Dadurch werden im IP-Core immer 8 Audiosamples vorgehalten und dienen als Zeitbuffer. Der Ablauf der AC97-ISR ist im Ablaufdiagramm 38 dargestellt.

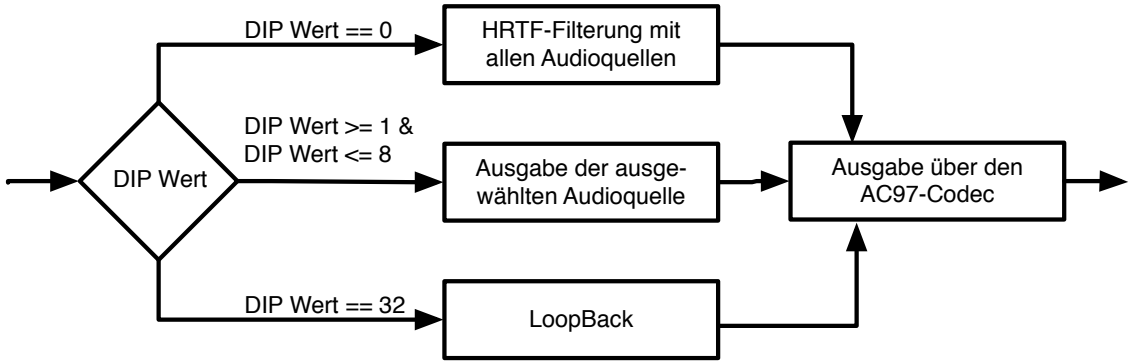


Abbildung 38: Ablaufdiagramm der AC97-ISR

Zuerst werden die ersten 6 Dips des Dip-Switchs des Boards abgefragt. Über diese kann der Benutzer verscheidene Arbeitsmodi auswählen. Die Tabelle 3 zeigt die Modi und deren Bedeutung für die ISR-Verarbeitung, sowie die dafür gemessenen Zeit.

DIP-Wert:	Bedeutung für die AC97-ISR	Zeitbedarf der ISR
0	HRTF-Filterung mit allen Audioquellen	6,2 μs
1-8	Audiospur x wird ausgegeben	6,2 μs
32	LoopBack: Das Eingangssignal wird ausgegeben	0,8 μs

Tabelle 3: Dip-Switch und dessen Bedeutung für die Audio-ISR

Bei der Filterung aller Audioquellen oder bei der Ausgabe eines Audiostreams wird immer aus jedem aktiven Audiobuffer ein Audiosample entnommen. Die Tabelle zeigt, dass die Verarbeitung in der Audio-ISR deutlich schneller ist als das ein neues Audiosample gefiltert werden muss. Somit steht für die Steuerung des Systems und das Empfangen von Audiodaten genügend Zeit zur Verfügung.

10.5 Audiostreaming auf der SoC-Seite

Durch die Initialisierung des Audio Streaming Protokolls wurde die Verbindung mit dem Server hergestellt. Über diese Verbindung werden alle erforderlichen Daten ausgetauscht. Die Main-Routine fordert immer neue Daten an, wenn genügend Speicherplatz im Zwischenspeicher vorhanden ist. Dabei achtet das System immer darauf, dass immer nur ein ASP-Paket für jeden Audiostream angefordert wird. Dieses ist notwendig, da es sonst vorkommen kann, dass die Reihenfolge der gesendeten und empfangenen Pakete verändert wird und somit die Audiodatei verfälscht wird.

Die Recieve-Methode, welche nach der Verarbeitung von lwIP aufgerufen wird, übergibt den Pointer des durch lwIP kopierten Paketes an die Main-Routine. Die Main-Routine verarbeitet im nächsten Durchlauf das Paket, indem es dieses in den entsprechenden Zwischenspeicher kopiert und den Speicher des Pakets bei lwIP freigibt.

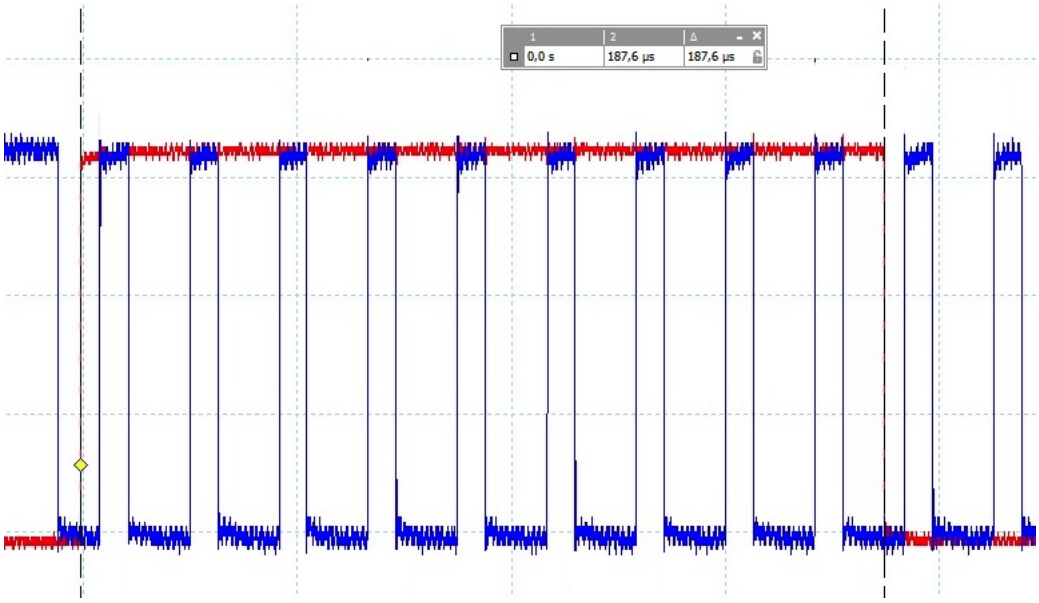


Abbildung 39: Zeitmessung: Audiosamples aus dem ASP-Paket in den Zwischenspeicher kopieren

Das Kopieren der Audiosamples wurde gemessen. Die Grafik 39 zeigt das Zeitverhalten. In Blau eingezeichnet ist die Audio-ISR, dabei arbeitet das System mit der aktiven HRTF-Filterung. Die steigende rote Flanke signalisiert das Starten des Kopiervorganges. Die fallende rote Flanke gibt das Ende des Kopierens an. Für einen Kopiervorgang werden im laufenden Betrieb ca. 187,6 μs gebraucht.

10.6 Messungen des Gesamtsystems

Das Gesamtsystem ist durch die vielen Schnittstellen und Aufgaben sehr komplex. Es ist sicherzustellen, dass die Audioverarbeitung immer rechtzeitig abgearbeitet ist. Wegen der Echtzeitanforderungen des Systems darf es nicht passieren, dass ein Audiosample verzögert berechnet wird.

Die meisten Ereignisse des Systems stellen kein Risiko dar. Da die Audio-ISR die höchste Interruptpriorität hat, wird die Main-Routine immer unterbrochen und stellt kein Risiko für die Audioverarbeitung dar. Die Audio-ISR arbeitet so schnell, das immer genügend Zeit für Main-Routine und deren Verarbeitung besteht. Somit stellen nur noch die Interrupts der Ethernet-Schittstelle und deren ISR Verarbeitung ein Risiko dar.

Die Messung 40 zeigt das Verhalten des Systems bei dem Empfangen eines großen ASP-Paketes.

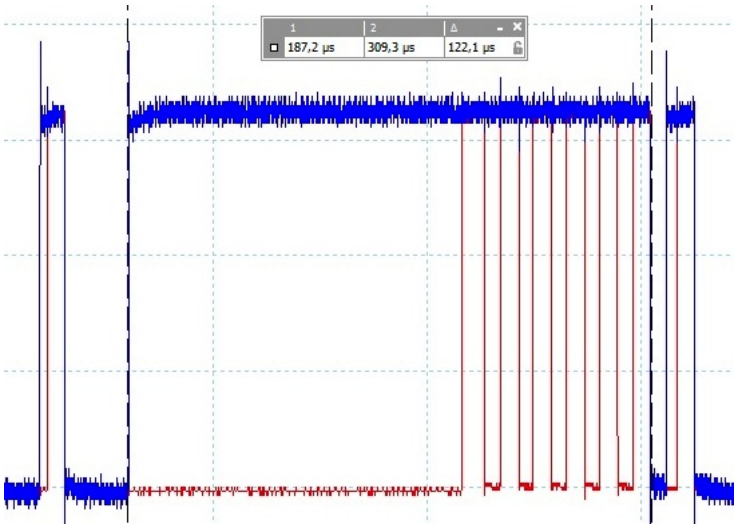


Abbildung 40: Messung des Verhaltens beim Empfangen eines großen ASP-Paketes unter Beobachtung der Audioverarbeitung.

In Blau dargestellt ist die Interruptleitung des AC97-IP-Cores. Die Audio-ISR besitzt im System die höchste Interrupt-Priorität. Der AC97-IP-Core erzeugt Interrupts bis der Ausgabespeicher zur Hälfte gefüllt ist. In Rot dargestellt ist die Audio-ISR. Zwischen dem Auftreten des Interrupts und der ersten Audio-ISR sind ca. 77,8 μs vergangen. In dieser Zeit müssten ca. 4 Audiosamples ausgegeben werden. Dieses Verhalten ist nur durch die Verarbeitung einer anderen ISR zu erklären. In vorliegendem System gibt es nur noch eine andere Interruptquelle. Dabei handelt es sich um die Ethernet-Schnittstelle. Eine Messung der Ethernet-ISR ist nicht einfach durchführbar. Die Zeitverzögerung zur Bearbeitung der Audio-ISR stellt kein Risiko dar, da der Audio-IP-Core 8 Audiosamples vorhält. Wie zu

erkennen ist, wird der Audiospeicher wieder aufgefüllt, bevor der Audio-Interrupt gelöscht wird. Die Zeit bis zum Löschen des Interrupts beträgt 122 μs, dieses entspricht zugleich 6 Audiosamples. Die Audio-ISR wurde bis zum Löschen des Interrupts 6-mal aufgerufen. Somit arbeitet das System ohne Störung weiter. Das Empfangen mehrerer Ethernet Pakete stellt keine Gefahr dar, da der Audio-ISR immer die höchste Priorität besitzt.

11 Entwicklung der Server-Seite

In diesem Kapitel wird die Serverarchitektur des 3D-Audio-Servers vorgestellt. Der 3D-Audio-Server besteht aus mehreren Komponenten und ist in Abbildung 41 dargestellt. Die Hauptkomponente ist in Java geschrieben und verbindet alle anderen Komponenten. Sie dient zur Steuerung der SoC-Plattform und gibt eine grafische Darstellung des aktuellen Gesamtsystems. Die IPhone-App MRMR dient der Szenensteuerung. Über das Jack-Interface werden die gewünschten Audiospuren abgegriffen. Zunächst werden die entsprechenden Kommunikationsschnittstellen vorgestellt. Danach wird der 3D-Audio-Server vorgestellt.

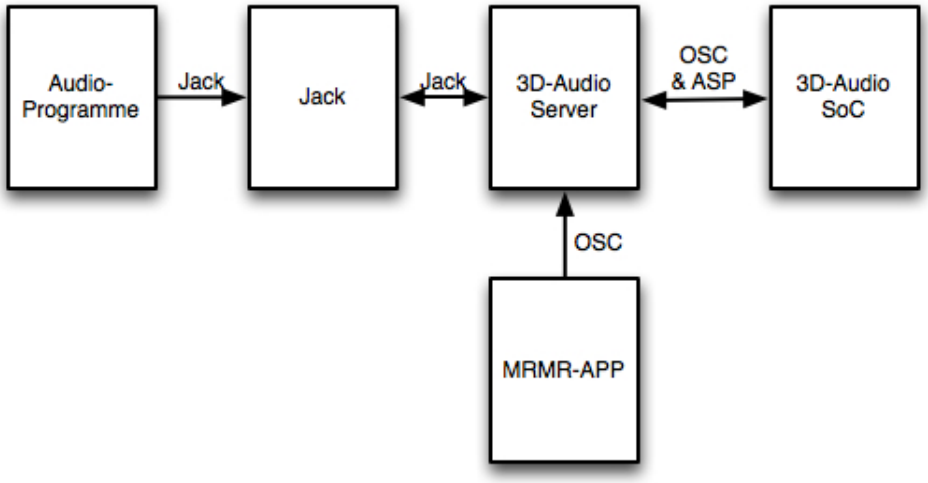


Abbildung 41: Komponentenübersicht mit Kommunikationsschnittstellen des 3D-Audio Server

11.1 JACK Audio Connection Kit

Das JACK Audio Connection Kit, in der Arbeit nur noch JACK genannt, ist eine Software-Schnittstelle für Audio-Anwendungen unter Unix-ähnlichen Systemen [jac]. In dieser Arbeit wurde aber die MAC-OX Version genutzt. Diese ist mit der Unix-Version voll kompatibel. JACK startet einen Daemon, welcher die Ein- und Ausgänge von Audioprogrammen verwaltet und es erlaubt die Audiosignale zwischen den Programmen zu routen. Der JACK-Server synchronisiert die Clients, indem er zu festen Zeiten Callback-Funktionen aufruft, die einen Block von Audiodaten lesen oder schreiben. Durch diese Funktion ist eine niedrige Latenzzeit zwischen den Anwendungen zum Datenaustausch erreichbar. Die minimale Latenzzeit, die bei JACK einstellbar ist, liegt bei 32 Audiosamples, welches unter Umständen eine Anpassung des Betriebssystems notwendig macht. Für das 3D-Audio-System wurde die Abtastrate bei JACK auf die 48kHz der SoC-Plattform eingestellt. Die Buffergröße wurde auf 512 Audiosamples eingestellt. Somit liegt die maximale Verzögerung bei 10,6 ms, was für dieses System ausreichend ist. Die Abbildung 42 zeigt eine grafische Darstellung des JACK-Routings, über welches die verschiedenen Audioprogramme verbunden werden.

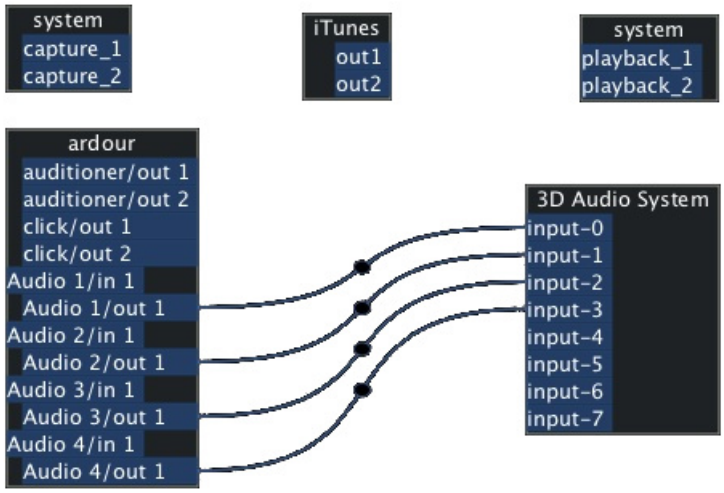


Abbildung 42: JACK-Routing Übersicht

Wie in der Jack-Routing Übersicht zu erkennen ist, kann auf jede Audioquelle des Systems zurückgegriffen werden. Auch eine Sprachaufnahme mittels Mikrofon lässt sich mittels JACK direkt an das 3D-Audio-System weiterleiten. Durch das Abgreifen jedes Signals ist die Flexibilität des Systems sehr groß. Das 3D-Audio System verfügt in dieser Arbeit über 8 Mono Signal Eingänge. JACK erlaubt es mehrere Audiospuren auf ein Eingangssignal zu mappen. Somit lassen sich beliebig viele Audiosignale verarbeiten, nur die Positionen der so zusammengeführten Signale sind identisch. Das Audioprogramm Ardour ist ein

Multikanal-Abspielprogramm, welches parallel mehrere Audiospuren abspielt und daher sehr gut als Abspielprogramm für das 3D-Audio System geeignet ist.

11.2 Szenensteuerung mittels MRMR-App

“MRMR - OPEN Mobile Touch Protocol“ [mrm] ist ein laufendes Open-Source-Forschungsprojekt. Diese IPhone-App ist zur Echtzeitsteuerung von Multimedia-Performances über mobile Geräte entwickelt worden. Sie erlaubt eine einfache Erstellung einer eigenen Benutzeroberfläche und verschickt die Benutzereingaben über OSC-Nachrichten.

Zur einfachen Steuerung der virtuellen Schallquelle wurde eine eigene Benutzeroberfläche erstellt. (Siehe Abbildung 43)

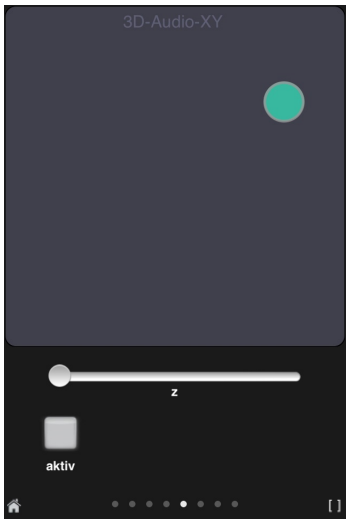


Abbildung 43: MRMR-Oberfläche zur Steuerung der virtuellen Schallquelle

Im 3D-Audio-XY Feld ist der grüne Punkt frei beweglich. Dieser symbolisiert die Position der virtuellen Schallquelle. Bei jeder dieser Positionsänderungen wird eine neue OSC-Nachricht mit der aktuellen Position verschickt. Der Z-Slider ist für die Höhe der Schallquelle gedacht.

Zur Steuerung des Audiostreams dient ein Push-Button. Dieser startet bzw. stopt den Audiostream. Zum Wechsel der verschiedenen Audioquellen ist im unteren Bereich eine Auswahlliste eingefügt.

Durch die MRMR-App lässt sich die Position der virtuellen Schallquelle leicht verändern. Ebenso sind die Audiostreams steuerbar. Eine Erweiterung oder Anpassung der Benutzeroberfläche ist schnell und einfach zu realisieren.

11.3 Hauptkomponente der Serverseite mit grafischer Darstellung

Die Hauptkomponente ist in Java implementiert. Es wurde darauf geachtet, dass alle eingebundenen Librarys betriebssystemunabhängig sind. Die Anwendung verbindet alle Komponenten wie die iPhone App MRMR oder Jack und das Audiostreaming und steuert die SoC-Plattform. Ebenso dient sie der grafischen Darstellung der aktuellen Systeminformationen. Abbildung 44 zeigt die grafische Oberfläche.

Abbildung 44: Entwickelte grafische Oberfläche der Hauptkomponente

Im oberen linken Bereich werden alle Informationen des Systems dargestellt. Dazu zählen PC-Informationen, wie die IP-Adresse und die entsprechenden Ports. Dieselben Informationen sind auch für das SoC aufgeführt. Des Weiteren werden daneben noch die Kompass-Werte dargestellt. Die Informationen der virtuellen Audioquellen werden ebenfalls dargestellt. Diese beinhalten eine Check-Box zum Aktivieren und Deaktivieren des Audiostreams. Ebenso werden die x,y,z Positionen, die über die MARA-App bereitgestellt werden, dargestellt. Zur Umrechnung in die Elevation, den Azimut und die Entfernung:

$$Azimut = atan2(x/y) \tag{15}$$

$$Elevation = atan2(x/z) \tag{16}$$

$$Entfernung = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{17}$$

Zudem ist angegeben, an welchen Input-Source von Jack die einzelnen Audioquellen angeschlossen sind.

Im linken unteren Bereich der Oberfläche werden alle aktuellen Informationen grafisch dargestellt. Die Kopfposition wird durch die SoC-Plattform festgestellt und mittels OSC-Nachrichten an den Server weitergegeben. Der Kopf bleibt immer zentriert, ist aber frei drehbar. Die Schallquellen sind durch entsprechende Einfärbung zu erkennen. Diese werden immer mit den durch die MRMR-App zur Verfügung gestellten Daten dargestellt. Die grafische Darstellung zeigt somit immer die aktuellen Systemeinstellungen.

Für die Serverseite wurde für die OSC-Verbindungen der OSC-Stack “Java OSC“ [jav] ausgewählt. Zum Senden von OSC-Nachrichten wird ein Objekt der Klasse OSCMessage erzeugt, welches über ein OSCPort (nur UDP) versendet wird. Zum Empfangen von OSC-Nachrichten wird ein OSCListener erzeugt. Dieser wird an ein OSCPortIn gebunden und gestartet. Wenn eine Nachricht empfangen wird, wird die acceptMessage-Methode des OSCListeners ausgeführt, in welche die entsprechenden Funktionen implementiert wurden. Durch diese Libary wird der Datenaustausch zwischen SoC-Plattform und Server in beiden Richtungen unterstützt.

Es gibt keine direkte Java-Schnittstelle, um sich mit dem Jack-Interface zu verbinden. Der Jack-Daemon ist in C++ geschrieben und bietet nur dafür ein entsprechendes Interface. Um mittels Java mit dem Jack-Daemon zu kommunizieren, gibt es mehrere Java-Wrapper. Es wurde sich für die JNAJACK Libary entschieden, weil diese auf allen Betriebssystemen, die JACK unterstützt, läuft. JNAJACK ist ein minimal objektorientierter Wrapper für das JACK-Audio-Connection KIT. Diese Libary arbeitet mittels Java Native Access [jnab], welche den Zugriff auf plattformspezifische dynamische Programmbibliotheken erlaubt. [jnaa]

Um mit Java auf Jack zuzugreifen, wurde eine JackAudioClient-Klasse entwickelt. Bei dem Erzeugen einer Instanz dieser Klasse verbindet sich diese mit dem Jack-Interface und erzeugt von jenem eine Instanz. Mittels dieser Instanz kann ein eigener JackClient erstellt werden. Dieser Client wird mit den gewünschten Input- und Outputports ausgestattet. In diesem Fall wurde ein 3D Audio System Client erzeugt, welcher 8 Inputports für die entsprechenden Audioquellen bereitstellt. Zum Aktivieren des Clients werden noch Jack-Informationen, wie die Samplerate und die Buffergröße, eingestellt. Ebenso werden die Callback-Funktionen erstellt und bekannt gemacht. Bei jedem Aufruf dieser Callback-Funktionen werden die Audiosamples zwischen Java und Jack ausgetauscht. Ebenso wird eine Methode einer weiteren Java-Klasse aufgerufen. Diese Java-Klasse genügt dem Interface JackAudioClient.Processor. Die aufgerufene Methode dieser Klasse ist process(FloatBuffer[] inputs, FloatBuffer[] outputs). In dieser wird beschrieben, wie die entsprechenden Audiodaten zu verarbeiten sind. In diesem Fall werden sie in einem synchronen Zwischenspeicher gesichert. Bei diesem Zwischenbuffer handelt es sich um

einen LIFO, der dem Audio Streaming Protokoll die zu verschickenden Audiodaten zur Verfügung stellt.

Somit sind alle Interfaces des 3D-Audio-Servers vorhanden. Durch die OSC Anbindung und die MRMR-App ist eine einfache Steuerung der Audioquellen implementiert worden. Ebenso ist durch OSC ein Informationsaustausch zwischen Client und Server durchführbar. Durch die Jack-Schnittstelle lassen sich alle im Betriebssystem vorhandenen Audioquellen vom 3D-Audio-Server abgreifen und mittels des entwickelten Audio Streaming Protokolls an die SoC-Plattform verschicken und verarbeiten.

11.4 Messungen Audio Streaming Protokoll

Die Abbildung 45 zeigt den Datenverkehr zwischen Client und Server nach dem Starten eines Audiostreams.

No.	Time	Source	Destination	Length	Protocol	Info
3	0.000144	192.168.1.10	192.168.1.55	50	ENIP	Source port: EtherN
4	0.000314	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
5	0.000536	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
6	0.001071	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
7	0.001223	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
8	0.001797	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
9	0.002059	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
10	0.002629	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
11	0.002836	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
12	0.003426	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
13	0.003659	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
14	0.016356	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
15	0.016593	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
16	0.031592	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
17	0.031725	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
18	0.046952	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
19	0.047095	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
20	0.062290	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
21	0.062609	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743
22	0.077558	192.168.1.55	192.168.1.10	60	ENIP	Source port: 49152
23	0.077769	192.168.1.10	192.168.1.55	1506	UDP	Source port: 59743

Abbildung 45: Datenverkehr zwischen Client und Server nach dem Starten eines Audiostreams, gemessen mit Wireshark

Die 3. Nachricht in Wireshark ist die Start-Nachricht des Servers an das SoC zum Starten eines Audiostreams. Die SoC-Seite fordert mit dem Versenden einer get-Sample Nachricht Audiodaten an. Daraufhin verschickt der Server ein Paket mit Audiodaten. Dieses Verhalten wiederholt sich immer wieder. Die ersten 5 Abläufe werden innerhalb von 3,7 ms verarbeitet. Dieses Verhalten zeigt das Befühlen des Zwischenspeichers auf der SoC-Seite. Ab Nachricht 13 wächst die Zeitdifferenz zwischen dem Anfordern von Daten auf ca. 15ms an. Der Server verschickt pro Paket 732 Samples, dies entspricht bei 48kHz einer Zeit von 15,25 ms.

12 Usability Test

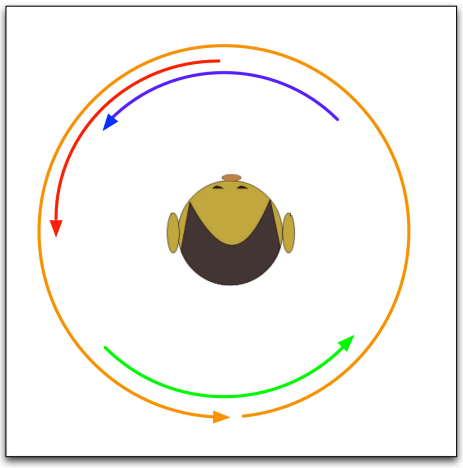
Durch die Durchführung des Usability Tests wurde festgestellt, wie Personen auf das entwickelte 3D-Audio-System reagieren. Dazu wurden verschiedene Testszenarien entwickelt. Diese werden im Kapitel 12.1 vorgestellt. Die Ergebnisse aus allen Testszenarien werden in Kapitel 12.2 vorgestellt und bewertet.

12.1 Testszenarien

Um das 3D-Audio-System zu testen, wurden vier verschiedene Testszenarien entwickelt und zu einem Usability Test zusammengeführt. Folgendes sind die Fragestellungen:

1. Bewegung einer Schallquelle
2. Ortung einer Schallquelle mittels Kopfbewegung
3. Gleichzeitige Ortung mehrerer Schallquellen mittels Kopfbewegung
4. Anwendungsbeispiele

Testszenario 1: Ortung einer Schallquelle
Im ersten Testszenario sollten die Testhörer eine sich bewegende Schallquelle lokalisieren. Es wurden vier Bewegungen vorgespielt. Diese sind in Abbildung 46 dargestellt. Bei der sich bewegenden Schallquelle handelt es sich um eine Frauenstimme. Die jeweilige Bewegung konnte beliebig oft angehört werden. Die Kopfbewegung der Testpersonen wurde in diesem Fall nicht berücksichtigt.



1. Bewegung = Blau
2. Bewegung = Rot
3. Bewegung = Orange
4. Bewegung = Grün

Abbildung 46: Usability Test, Testszenario 1 - Lokalisierung einer sich bewegenden Schallquelle

Testszenario 2: Ortung einer Schallquelle mittels Kopfbewegung
Beim zweiten Test wurde die Schallquelle fest im Raum positioniert. Die Testpersonen sollten durch Bewegungen des Kopfes die Position der Schallquelle ermitteln, dabei wurden die Kopfbewegungen des Zuhörers mittels des Kompasses gemessen und in die Audiofilterung einbezogen. Eine Zeitbegrenzung für diese Aufgabe gab es nicht. Die vier verschiedenen Positionen sind in Abbildung 47 dargestellt.

Testszenario 3: Gleichzeitige Ortung mehrerer Schallquellen mittels Kopfbewegung
Im dritten Testszenario wurden vier verschiedene Schallquellen fest im Raum positioniert und gleichzeitig eingespielt. Der Testaufbau ist in Abbildung 48 dargestellt. Bei den Schallquellen handelte es sich um 1. Frau, 2. Mann, 3. Geige und 4. Bass. Die Testhörer sollten wie in Testszenario 2 durch Kopfbewegung die Positionen der Schallquelle bestimmen. Der Kompass des 3D-Audio-Systems war dabei aktiviert und die Kopfposition wurde in für die Audiofilterung berücksichtigt. Die Anzahl der Schallquellen war dabei nicht bekannt. Es gab keine Zeitbegrenzung.

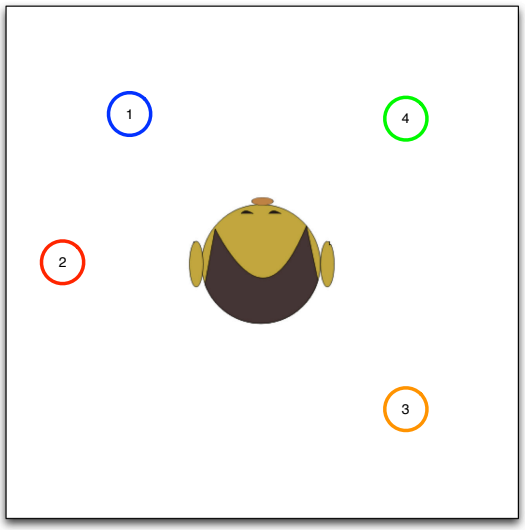


Abbildung 47: Usability Test 2 - Ortung einer Schallquelle mittels Kopfbewegungen

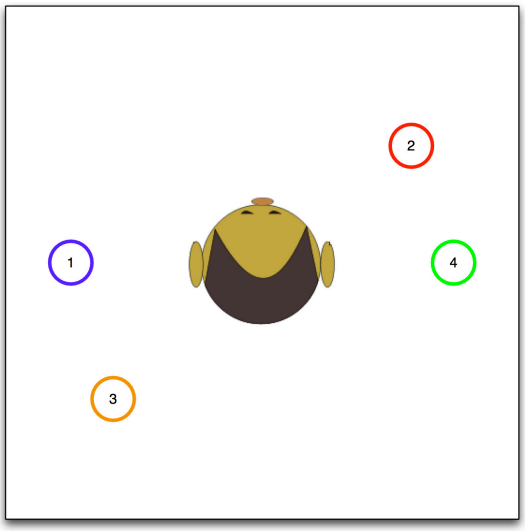


Abbildung 48: Usability Test 3 - Ortung mehrerer Schallquelle mittels Kopfbewegungen

Testszenario 4: Anwendungsbeispiele
Beim Anwendungstest wurden den Testhöreren 3 verschiedene Musik- beziehungsweise Audio-Stücke vorgespielt. Beim ersten Anwendungstest wurde den Testhörern ein Rocklied vorgespielt. Dabei wurden bei der 3D-Audio Version die 4 Schallquellen (Stimme, Bass, Schlagzeug, Gitarren) im Raum positioniert. In der Vergleichsversion wurden alle 4 Spuren zu einem Monosignal zusammengefügt, sodass auf beiden Kanälen dasselbe zu

hören war. Der zweite Anwendungsfall war klassische Musik. Dabei handelte es sich um ein Streichquartett, welches wie im ersten Anwendungsfall aufgebaut war. Beim dritten Anwendungsfall handelte es sich um Vögel in einem Wald. In diesen Fall wurden bei der 3D-Audio Version alle 8 zur Verfügung stehenden Schallquellen benutzt. Bei der Vergleichsversion (Stereo) wurden die verschiedenen Audiospuren gleichmäßig auf den rechten und linken Kanal verteilt. Die Aufgabe für die Testhörer war es, die beiden Varianten zu vergleichen. Zum einen auf die Lokalisierbarkeit und zum anderen auf die Natürlichkeit. Dazu wurden Punkte von 1 bis 5 vergeben. Wobei die 1 für schlecht und die 5 für sehr gut standen. Außerdem sollten die Testhörer für jedes Anwendungsbeispiel das ihnen besser gefallende angeben.

12.2 Durchführung und Ergebnisse des Usability Tests

Die Testreihe wurde an der HAW Hamburg durchgeführt. Die Testpersonen wurden jeweils einzeln getestet. Am Usability Test haben 10 Testpersonen teilgenommen. Hiervon waren 3 weiblich und 7 männlich. Drei Personen gaben an selbst Musik zu machen. Im Durchschnitt wurde angegeben, dass 4,3 Stunden Musik am Tag gehört wurden und dies überwiegend über Kopf-/Ohrhöhrer..

Auswertung Testszenarios 1: Bewegung einer Schallquelle
Bei der Testreihe wurde getestet, inwiefern die Testhörer die Bewegung einer Schallquelle wahrgenommen haben. Die Tabelle 4 zeigt das Testergebnis.

Bewegung:	Richtung	Vorne/ Hinten	Im Kopf
1:	6	2	3
2:	5	1	4
3:	9	3	4
4:	6	6	4
	6,5	3	3,75

Tabelle 4: Ergebnisse des Testszenarios 1 für alle 10 Testpersonen: Bewegung einer Schallquelle

Im Durchschnitt haben 65% die richtige Bewegung der Schallquelle wahrgenommen. Dabei war aber nur für 30% der Testpersonen eine Unterscheidung Vorne/Hinten auszumachen. Die meisten Angaben in Bezug auf die Vorne/Hinten-Lokalisierung wurden als hinter dem Kopf wahrgenommen. 37,5% der Testpersonen gaben an, dass sich die Schallquelle bei der Bewegung durch ihren Kopfmittelpunkt bewegt zu haben schien.

Auswertung Testszenarios 2: Ortung einer Schallquelle mittels Kopfbewegung
Bei dem Testszenario 2 wurde getestet, inwiefern die Testhörer die Position einer Schallquelle durch die Kopfbewegung wahrnehmen konnten. Dabei ist zu berücksichtigen, dass der im System befindliche Kompass nicht gut dafür geeignet ist. Er liefert im Gebäude nur sehr schlecht nutzbare Werte. Aufgefallene Effekte waren große Sprünge, sowie bei einer Nickbewegung eine Veränderung des Azimuts. Die Tabelle 5 zeigt das Testergebnis.

Position:	Position	Vorne/ Hinten	Im Kopf
1:	6	4	0
2:	3	6	1
3:	5	7	1
4:	3	3	1
	4,25	5	0,75

Tabelle 5: Ergebnisse des Testszenarios 2 für alle 10 Testpersonen: Ortung einer Schallquelle

Durch den nicht richtig funktionierenden Kompass fällt die Auswertung der Position und die Vorne/Hinten-Lokalisierung schwer. Die meisten Testpersonen gaben an, dass die Schallquelle sich auch ohne Kopfbewegung bewegt hat. Ebenso wurden große Sprünge wahrgenommen. Immerhin haben trotzdem fast die Hälfte aller Testpersonen die Position richtig geortet. Auffällig bei dieser Testreihe ist, dass die Position von Schallquellen nur im Durchschnitt von 7,5% im Kopfmittelpunkt geortet wurde.

Auswertung des Testszenarios 3: Ortung mehrerer Schallquellen
Bei der Testreihe wurde getestet, inwiefern die Testhörer die Position mehrerer Schallquellen gleichzeitig durch die Kopfbewegung wahrnehmen konnten. Wie bei dem zweiten Testszenario ist der vorhandene Kompass maßgeblich ein Problem. Die Tabelle 6 zeigt das gemessene Testergebnis.

Position:	Position	Vorne/ Hinten	Im Kopf
1:	5	7	0
2:	4	3	2
3:	3	2	0
4:	5	6	0
	4,25	4,5	0,5

Tabelle 6: Auswertung des Testszenarios 3 für alle 10 Testpersonen: Ortung mehrerer Schallquellen

Alle Testpersonen haben die eingespielten Audioquellen erkannt. Es bestätigt sich, wie in Testaufbau 2, dass die Ortung einer Schallquelle im Kopf kaum noch existiert. Eine genauere Auswertung der Position ist aufgrund des ungenauen Kompasses nicht durchführbar. Es zeigt sich aber, dass die Testergebnisse zwischen Test 2 und Test 3 nicht abweichen. Das lässt den Schluss zu, dass die Ortung mehrerer gleichzeitig eingespielter Schallquellen durch das 3D-Audio System wahrscheinlich ist.

Auswertung Test 4: Anwendungsbeispiele
Bei dieser Testreihe sollten die Testpersonen zwei Varianten in Bezug auf Lokalisierbarkeit und Natürlichkeit vergleichen. Dazu wurde den Testpersonen die gleiche Musik als 3D-Audio oder als Mono/Stereo eingespielt. Die Testhörer konnten beliebig zwischen beiden Versionen wechseln. Die Abbildung 49 zeigt die Auswertung aller drei Anwendungsbeispiele. Der schwarze Balken zeigt den Bereich der Antworten an. Der grüne Punkt das Mittel aus allen Bewertungen.

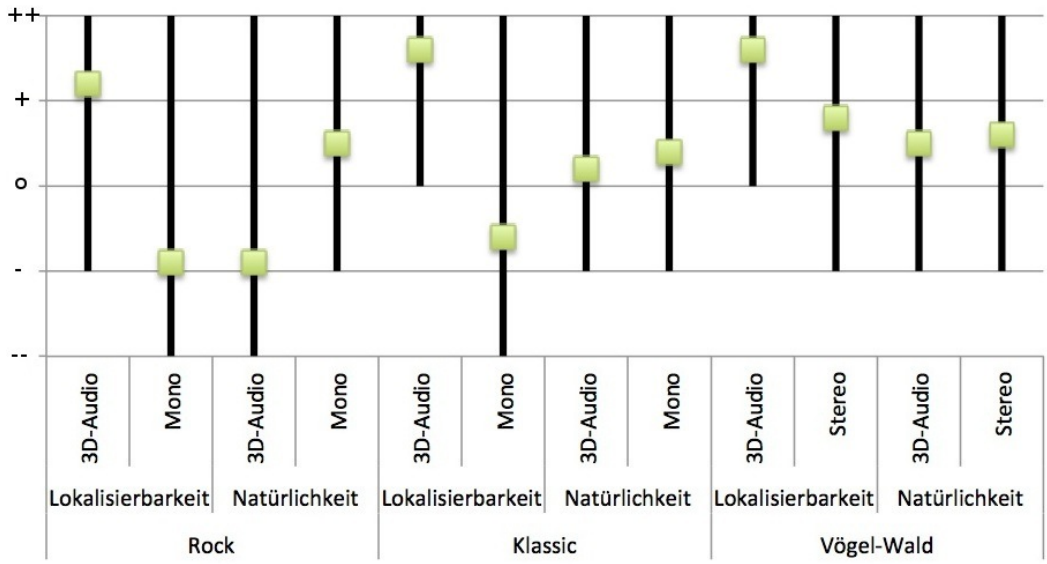


Abbildung 49: Auswertung des vierten Testszenarios

Beim Rock-Musik-Beispiel wurde die 3D-Audio Version in Bezug auf die Lokalisierbarkeit als deutlich besser empfunden als die Vergleichsversion. Eine Lokalisierung bei der Vergleichsversion ist nicht durchführbar, da beide Kanäle des Kopfhörers dasselbe Signal enthalten. Somit befindet sich die Lokalisierung der Schallquellen im Kopfmittelpunkt. Bei der Natürlichkeit wurde die Mono-Version bevorzugt. Es gibt bei der Frage nach der besseren Variante einen Gleichstand. Einige Testpersonen gaben genauere Gründe an:

- Schmerzen auf dem linken Ohr, da die Position der Stimme von links kommt.
- Die 3D-Audio-Version ist weniger basslastig.
- Die Mono-Version hört sich wie normale Musik an.

Die zweite Testreihe mit der klassischen Musik unterstreicht die Ergebnisse bei der Lokalisierbarkeit aus Testreihe Rock. Bei der Natürlichkeit wurde zwar die Mono-Version bevorzugt, ist aber nur minimal besser bewertet worden. Jedoch gaben 80% der Testhörer an, dass Sie die 3D-Audio-Version besser fanden. Auch bei dieser Testreihe wurden Gründe angegeben:

- Bei beiden Versionen hören sich die Instrumente metallisch an.
- Die 3D-Audio-Version gibt einen sehr räumlichen Eindruck wieder.
- Habe keine Erfahrung mit klassische Musik, aber die Variante 3D-Audio hört sich sehr natürlich an, vergleichbar mit dem Klang in einem Konzertsaal.

Bei dem letzten Testbeispiel handelte es sich um ein Naturbeispiel. Es wurde eine Vielzahl von Vögeln sowie ein Bach eingespielt. Dabei wurden alle 8 Spuren der 3D-Version benutzt und die Position der Audioquellen um den Kopf gleichmäßig verteilt. Bei der Vergleichsvariante wurden die Einzelspuren gleichmäßig auf den rechten und linken Kanal gelegt. Somit wurde eine Unterscheidung zwischen dem rechten und linken Ohr ermöglicht.
Die Lokalisierbarkeit wird bei der 3D-Version als besser empfunden. Durch die gleichmässige Aufteilung der Spuren auf den rechten und linken Kanal ist aber auch die Lokalisierung bei der Stereo-Variante als gut empfunden worden. Eine Unterscheidung zwischen beiden Varianten in Bezug auf die Natürlichkeit ist nicht festgestellt worden. Trotzdem bevorzugten 70% der Tester die 3D-Audio-Version. In dieser Testreihe wurden keine Gründe für die Entscheidungen genannt.

12.3 Fazit des Usability Tests

Durch den Hörtest wurde gezeigt, dass eine Lokalisierung von Schallquellen durch das ARA-System durchführbar ist. Eine aussagekräftige Auswertung ist bei der kleinen Anzahl an Testhörern und dem schlechten Kompass und dem daraus schlechten Headtracking nicht durchführbar. Durch das vierte Testszenario (Anwendungsfälle) wurde gezeigt, dass die 3D-Audio-Version als besser empfunden wurde. Bei der Durchführung und Auswertung ist aufgefallen, dass eine Vielzahl von Testhörern in einer Stereo-Denkweise hören, welche die normale Hörweise ist. Auffällig ist, dass eine Mehrheit der Testhörer sich nicht auf das 3D-Audio Verfahren einlassen wollte.

Interessant wäre eine weitere Studie, in der getestet würde, wie Testhörer sich an ein solches System anpassen. Da die HRTFs in dieser Arbeit mittels eines Kunstkopfes aufgenommen wurden, widersprechen sie den HRTFs der Testpersonen. Jede Person hat ihre eigene HRTF, welche von einer Vielzahl an Faktoren abhängig ist. Bei dieser Studie haben die Testpersonen mehrmals mit dem System gearbeitet. Eine identische Studie wurde in einem Paper der Firma Rockwell vorgestellt [SWC+03]. Erkenntnis aus dieser Veröffentlichung ist, dass der Mensch sich an ein solches System anpasst. Dadurch sind deutlich bessere Testergebnisse erzielbar. Dieses bestätigt sich aus eigener Erfahrung. Durch die lange Entwicklungszeit fällt es mir leicht, die Position bzw. Bewegungen von Schallquellen mittels des 3D-Audio-Systems festzustellen.

13 Diskussion von Optimierungsansätzen

Während der Entwicklungsphase des MARA-Systems und der anschließenden Dokumentationsphase wurden mehrere Optimierungsansätze entwickelt. Sie werden in diesem Kapitel vorgestellt.

13.1 Austausch des Kompasssensors

Der in dieser Arbeit eingesetzte 3-Achsen-Kompass ist für die Anforderungen des MARA-Systems nicht mehr ausreichend. Die Aktualisierungsfrequenz des Sensors von maximal 40 Hz ist deutlich zu langsam, um ein genaues Kopf-Tracking durchzuführen. Ebenso übermittelt der Sensor eine Vielzahl von Fehlmessungen, welche zusätzlich zu einer Fehlinterpretation führen.

Aus diesen Gründen ist der Kompasssensor durch einen anderen zu ersetzen. Das Faust-Team der HAW Hamburg hat von der Firma x-IO ein Sensorarray vom Typ x-BIMU zur Verfügung gestellt bekommen [IT13]. Mit diesem Sensor verfolgen sie die Bewegungen ihrer Modellfahrzeuge. Die Sensor-Eigenschaften sind:

- Kalibriertes Gyroskop ($\pm$ 2000°/s)
- Beschleunigungsmesser ($\pm$ 16 g)
- Magnetometer
- On-board AHRS Algorithmus
- Wählbare Datenraten von bis zu 256 Hz

Der Vorteil dieses Sensors liegt in der einfachen Nutzung. Er lässt sich an die entwickelte SoC-Plattform anschliessen und erfordert keine aufwendigen Berechnungen, da diese im Sensor durchgeführt werden. Des Weiteren lässt sich mittels dieses Sensors nicht nur die Kopfbewegung messen, sondern auch die Bewegung des Zuhörers. Diese wird wichtig, wenn eine Simulation der Entfernung von Schallquellen vorhanden ist.

13.2 Einbeziehung der Entfernung von Schallquellen

Das MARA-System ist auf die Positionierung mehrerer virtueller Schallquellen in einer festen Entfernung zum Kopfmittelpunkt beschränkt. Um eine vollständige dreidimensionale Positionierung der Schallquellen durchzuführen, ist zusätzlich die Entfernungssimulation im Signalverarbeitungsprozess zu berücksichtigen.

Hieraus ergibt sich zuerst die Problematik der Feststellung der Entfernung zur Klangquelle. Die Raumpositionen der Schallquellen sowie die des Benutzers sind zu bestimmen.

Außerdem sind die Bewegungen des Benutzers genauestens zu verfolgen. Mit dem derzeit verwendeten Sensor ist eine Bewegungsverfolgung der Person nicht denkbar. Durch den Einsatz des vorgeschlagenen Sensors ist durch die Beschleunigungs-, Gyroskop- und Kompass-Daten algorithmisch die Position im Raum berechenbar. Aus dieser und der Raumposition der Schallquellen lässt sich die Entfernung zur Schallquelle bestimmen.

Alternativ könnte über ein Indoor-Positionierungs-System nachgedacht werden, welches die Position bestimmt und über die kabellose Schnittstelle an das SoC übermittelt. Über einen GPS-Empfänger ist der Einsatz des MARA-Systems auch im Freien denkbar.

Die Audiofilterung ist mit der Entfernung zur Schallquelle zu berechnen. Dabei ist das Entfernungshören stark mit dem Räumlichkeitseindruck eines Raumes verbunden. Das menschliche Gehör wertet eine Vielzahl von Faktoren aus. Es lässt sich zwischen beiden keine klare Grenze ziehen. Die Faktoren sind:

- Direktschall zu reflektiertem Schall
- Anfangszeitlücke (ITDG)
- Lautheit
- Spektrale Verteilung (frequenzabhängige Dämpfung)

Eine genauere Beschreibung zum Entfernungshören und Räumlichkeitseindruck habe ich in meiner Seminararbeit [Hem13] ausgearbeitet.

Für eine erste Testreihe ist es auch denkbar, die Berechnungen auf dem PC durchzuführen und über die mobile Schnittstelle an das SoC zu verschicken, welches diese mit den gefilterten Audioquellen abspielt. Durch dieses Vorgehen lässt sich schnell ein Algorithmus zur Berechnung von Entfernungshören und Räumlichkeitseindruck testen. [BB08] [Sennt] [AT08]

13.3 Szenensteuerung für das MARA-System

Zur Steuerung der Schallquellen ist in dieser Arbeit eine einfache Szenensteuerung mittels der MRMR-App entwickelt worden. Diese erlaubt es mittels eines iPhones die Position einer Schallquelle zur gleichen Zeit zu bewegen bsw. zu positionieren. Die einfache Szenensteuerung ist sehr eingeschränkt. Aus diesem Grund wäre eine Neuentwicklung ratsam. Folgende Punkte sind zu beachten:

- Bewegung von Schallquellen: Es wird eine Position und eine Zeit vorgegeben. Durch die Angaben wird die Schallquelle in der gewünschten Zeit an die entsprechende Position bewegt.

- Mehrere Schallquellen gleichzeitig steuern: Zurzeit ist immer nur eine Schallquelle durch die Szenensteuerung steuerbar. Die erweiterte Szenensteuerung muss mehrere Bewegungen von Schallquellen zur gleichen Zeit unterstützen.
- Auswahl der Eingangssignale: Wenn das System zum Beispiel in einem großen virtuellen Raum eingesetzt wird, gibt es sicherlich mehrere Schallquellen als die zurzeit zur Verfügung stehenden 8 Audioquellen. Nicht alle Schallquellen sind aber gleichzeitig hörbar. Durch die Auswahl der Eingangssignale könnte das System immer Audioquellen in der näheren Umgebung weiterleiten. Wenn mehrere Schallquellen als Audiofilter auf der SoC-Plattform vorhanden sind, wäre es denkbar, dass mehrere Schallquellen zu einer gemeinsamen Schallquelle vereint werden und an das System weitergegeben werden.
- Simulation von Gebäuden: Wenn das System in einem Gebäude zum Einsatz kommt und Schallquellen in verschiedenen Räumen positioniert sind, ist zu beachten, welche Schallquellen durch den Benutzer und seine Position im Gebäude wahrnehmbar sind.

Die aufgeführten Szenarien sind nur einige Beispiele. Der Umfang der gewünschten Szenensteuerung hängt dabei stark vom Einsatzgebiet des MARA-Systems ab.

Ein wichtiger Punkt für die Steuerung des Systems ist, dass die Bedienung einfach aufgebaut ist. Denkbar wäre auch eine Szenensteuerung für das MARA-System und die Wellenfeldsynthese-Anlage der HAW Hamburg. Bei der Entwicklung des MARA-Systems wurde darauf geachtet, dass die gleichen Schnittstellen verwendet werden.

13.4 Augmented Reality Ansatz umsetzen

Der Augmented Reality Ansatz ist in diesem System nur teilweise umgesetzt. Durch die Nutzung von offenen Kopfhörern nimmt der Hörer zwar noch die reale Umgebung wahr, diese wird aber dennoch verfälscht. Um den AR-Ansatz umzusetzen, müssen andere Kopfhörer eingesetzt werden. Durch die Nutzung von geschlossenen Kopfhörern lässt sich die Umwelt weitestgehend ausblenden. Durch Mikrofone an beiden Ohren kann die Umwelt um den Zuhörer aufgenommen werden. Diese Signale sind durch das AR-System anzupassen und mit den virtuellen Schallquellen abzuspielen. Durch dieses Vorgehen ist der AR-Ansatz vollständig umzusetzen.

Erste Kopfhörer mit eingebauten Mikrofonen sind derzeit im Handel erhältlich. Diese nehmen die Umwelt auf und spielen diese umgehend über die Kopfhörer mit ab. Ob der Einsatz eines solchen Kopfhörers für das System relevant ist, kommt auf die Anforderungen an. Bei diesen Kopfhörern ist es meistens nicht einfach an die aufgenommenen Signale zu kommen. Wenn eine Anpassung der Signale erforderlich ist oder die Umwelt ganz ausgeblendet werden soll, ist der Einsatz von diesen Kopfhörern nicht denkbar.

14 Zusammenfassung

In dieser Masterarbeit wurde ein kopfhörerbasiertes MARA-System entwickelt. Dieses Echtzeit-Audiosignalverarbeitungssystem kann mehrere virtuelle Schallquellen an verschiedenen Positionen simulieren. Durch die Kopfverfolgung mittels eines 3-Achsen-Kompasses bleibt die virtuelle Schallquelle an Ort und Stelle.

Das System basiert auf einen schallquellenbasierten Ansatz. Die Positionierung der bis zu acht virtuellen Schallquellen beschränkt sich dabei auf den horizontalen 360° Bereich um dem Kopf, sowie im vertikalen Bereich von -41° bis +37° vom Kopfmittelpunkt. In beiden Ebenen kann die Schallquelle in 1°-Schritten positioniert werden. Die Entfernung der Schallquellen zum Hörer ist durch die zugrunde liegenden HRTF-Messungen mit einbezogen und ist 1,4 Meter vom Zuhörer entfernt.

Das entwickelte MARA-System baut auf einer Server-Client-Architektur auf. Bei der Serverseite wurde darauf geachtet, dass die entwickelte Software plattformunabhängig ist. Die Aufgabe des Servers ist es, alle relevanten Informationen zu sammeln und an den Client weiterzuschicken. Die Audioquellen werden mittels JACK vom System abgegriffen, somit ist das System sehr flexibel.

Die Client-Seite basiert auf einer SoC-Plattform. Diese empfängt die Daten vom Server über eine mobile Schnittstelle, dadurch ist die Mobilität des Zuhörenden gewährleistet. Die Audioberechnung für alle virtuellen Schallquellen wurde in einen IP-Core ausgelagert. Dieser HRTF-IP-Core wurde mittels dem Xilinx Systemgenerator und MATLAB/Simulink entwickelt und in das entwickelte SoC eingebunden. Durch diesen Hardwarebeschleuniger wird der Mikrocontroller entlastet und steht für logiklastige Steuerungsaufgaben zur Verfügung.

Der Hardwarebeschleuniger verarbeitet bis zu acht Audioquellen parallel. Die dafür zugrunde liegenden HRTFs wurden in die Pegeldifferenz (ILD) und in die Laufzeitdifferenz (ITD) aufgeteilt. Jede Audioquelle wird zuerst mittels ILD-Stützstellen-Filter verarbeitet. Die Ergebnisse der Stützstellen-Filter werden mittels einer Interpolation zur gewünschten ILD-Filterposition weiterverarbeitet. Das so erzeugte Ergebnis wird durch einen ITD-Filter zum Gesamtergebnis des HRTF-Filters verarbeitet. Alle Filterergebnisse der virtuellen Schallquellen werden zusammengemischt und über eine FSL-Schnittstelle an den Mikrokontroller weitergegeben, der die Ausgabe der Audiosamples übernimmt. Bei der Entwicklung des Hardwarebeschleunigers wurde auf den Ressourcenverbrauch geachtet. Ebenso war ein weiteres Kriterium, dass die Filterung schnellstens auf Änderung anpassbar ist. Wenn eine Positionsänderung festgestellt wird, werden die neuen Positionen an den Filter weitergegeben und die Filterung wird sofort auf die neue Filterposition angepasst.

Die Echtzeitleistung des Systems ist vollständig vorhanden. Das SoC empfängt alle Audio-

streams und verarbeitet diese durch den HRTF-IP-Core. Die Samplerate des Systems ist durch den verwendeten AC97-Audio-Codec auf 48 kHz begrenzt. Die geringe Aktualisierungsfrequenz und die große Anzahl von Fehlmessungen des eingesetzten Kompasssensors ist als kritisch zu beachten. Für eine Weiterentwicklung ist dieser Sensor auszutauschen, eine Anpassung von anderen Komponenten durch den Austausch des Sensors ist nicht notwendig.

Die Qualität des MARA-Systems bezüglich der gefilterten, virtuellen Schallquellen und dessen Positionserkennung ist als gut zu bewerten. Dieses wurde durch einen Usability Test ermittelt, an diesem Hörtest haben 10 Testpersonen teilgenommen. Vier verschiedene Testszenarien wurden getestet. Dabei handelte es sich um die Bewegung einer Schallquelle sowie die Ortung einer Schallquelle mittels Kopfbewegung. Ebenso wurde getestet, ob eine gleichzeitige Ortung mehrerer Schallquellen mittels Kopfbewegung durch das System ermöglicht wird. Im letzten Test hörten die Testhörer mehrere Anwendungsbeispiele und sollten einen Vergleich zwischen einer Vergleichsversion (Mono/Stereo) und der 3D-Audio-Version durchführen.

Das Embedded-System bietet ausreichende Ressourcen in Form von Prozessorleistung, Speicherkapazität und freiem Platz auf dem FPGA-Chip, sodass weitere Filterungen durchführbar sind. Bei diesen Filterungen könnte eine Umsetzung der Simulation von Entfernungen und Schallquellen und der zugleich damit umhergehenden Raumakustik implementiert werden. Diese und weitere Erweiterungen sind während der Entwicklung des MARA-Systems und der anschließenden Dokumentationsphase entstanden und im Kapitel 13 vorgestellt worden.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AC97	Audio Codec 97
AD-Wandler	Analog-Digital-Wandler
ARA	Augmented Reality Audio
ASM	Abstract State Machine
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
DSP	Digital Signal Processor
EDK	Embedded Development Kit
EN	Enable
FAT	File Allocation Table
FFT	Fast Fourier transform
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSL Bus	Fast Simplex Link Bus
FSM	Finite State Machine
FSM	Finite-State Machine
GPIO	General Purpose In Out
GUI	Graphical User Interface
HAW	Hochschule für Angewandte Wissenschaften
HDL	Hardware Description Language
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)
JEAS	Journal der Audio Engineering Society
JTAG	Joint Test Action Group
LCD	Liquid Cristal Display
LIFO	Last In First Out

LMB Bus	Local Memory Bus
LSB	Least Significant Bit
lwIP	Lightweight TCP/IP Stack
MAC	Multiply Accumulate
MARA	Mobile Augmented Reality Audio
MHS File	Microprocessor Hardware Specification File
MIT	Massachusetts Institute of Technology
MPD File	Microprocessor Peripheral Definition File
MSB	Most Significant Bit
MUX	Multiplexer
NGC File	File
OPB Bus	On Chip Peripheral Bus
OSC	Open Sound Control
PLB Bus	Processor Local Bus
POA File	Peripheral Analyze Order File
RAM	Random Access Memory
RFC	Request for Comments
RISC	Reduced Instruction Set Computer
RTL	Register transfer level
RTP	Real-Time Protocol
SDK	Software Development Kit
SLR	Schieberegister
SoC	System on Chip
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter
UCF File	User Constraints File File
UDP	User Datagram Protocol
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VR	Virtual Reality
XPS	Xilinx Platform Studio

Literatur

[ASH08] ASHENDEN, Peter: *Digital Design: An Embedded Systems Approach Using VHDL*. Frankfurt am Main : Morgan Kaufmann, 2008 (ISBN 9978-0-12-369528-4)

[AT08] AHNERT, Wolfgang ; TENNHARDT, Hans-Peter: *Handbuch der Audiotechnik: Raumakustik*. Springer Berlin Heidelberg, 2008 (ISBN 978-3-540-34301-1 (online))

[BB08] BLAUERT, Jens ; BRAASCH, Jonas: *Handbuch der Audiotechnik: Räumliches Hören*. Springer Berlin Heidelberg, 2008 (ISBN 978-3-540-34301-1 (online))

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

[Dun01] DUNKELS, Adam: *Design and Implementation of the lwIP TCP/IP Stack*. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.1795&rep=rep1&type=pdf`. Version: 2001

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

[Goo] GOOGLE: *Google Glass Webseite*. http://www.google.com/glass/start/, 22.08.2013

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

[Hem11] HEMMER, David: *Eine SoC-Plattform zur kontinuierlichen Interpolation von HRTF-Filtern für positionsveränderliche virtuelle Schallquellen*. Bachelorarbeit (HAW-Hamburg), 2011

[Hem13] HEMMER, David: *Mobiles Augmented Reality Audio System*. Seminar Ausarbeitung (HAW-Hamburg), 2013

[HJT+04] HÄRMÄ, Aki ; JAKKA, Julia ; TIKANDER, Miikka ; KARJALAINEN, Matti ; LOKKI, Tapio ; HIIPAKKA, Jarmo ; LORHO, Gaetan: *Augmented Reality Audio for Mobile and Wearable Appliances**. J. Audio Eng. Soc., Vol. 52, No. 6, 2004

[Huo99] HUOPANIEMI, Jyri: *Virtual Acoustics and 3-D Sound in Multimedia Signal Processing*. Dissertation, Helsinki University of Technology, 1999

[IIS] IIS, Fraunhofer: *Fraunhofer IIS "Cingo"Webseite*. http://www.iis.fraunhofer.de/de/bf/amm/produkte/audiocodec/audiocodecs/cingo.html, 22.08.2013

[IT13] IO TECHNOLOGIES x: *Internetseite x-BIMU*. http://www.x-io.co.uk/products/x-bimu/, 2013

[jac] *Jack Audio Webseite*. http://jackaudio.org, 22.08.2013

[jav] *Illposed Software — Java OSC*. `http://www.illposed.com/software/javaosc.html`

[jnaa] *JAudioLibs - JNAJack*. `http://code.google.com/p/java-audio-utils/`

[jnab] *Java Native Access (JNA) Webseite*. `https://github.com/twall/jna#readme`

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

[MATC] MUSIC, New ; AUDIO TECHNOLOGY (CNMAT), UC B.: *Introduction to OSC*. `http://opensoundcontrol.org/introduction-osc`

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

[mrm] *MRMR - OPEN Mobile Touch Protocol*. `http://mrmr.noisepages.com/`

[net] *Netgear WNCE3001*. `http://www.netgear.de/images/WNCE3001_DS_23Feb1222-35788.pdf`

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

[Pel09] PELTOLA, Mikko: *Augmented Reality Audio Applications in Outdoor Use*. Master's Thesis, Helsinki University of Technology, 2009

[rfca] *RFC 1889: RTP: A Transport Protocol for Real-Time Applications*. `http://www.ietf.org/rfc/rfc1889.txt`

[rfcb] *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*. `http://www.ietf.org/rfc/rfc3550.txt`

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

[Sen03] SENGPIEL, Eberhard: *Laufzeitdifferenzen beim natürlichen Hören. Interaurale Laufzeitdifferenz in Abhängigkeit von Schalleinfallswinkel bei Sprache und Musik*. http://www.sengpielaudio.com/, 2003

[Sennt] SENGPIEL, Eberhard: *Anfangszeitlücke ITDG und Pre-Delay*. http://www.sengpielaudio.com/AnfangszeitlueckeUndPredelay.pdf, Jahr unbekannt

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

[SWC+03] SUNDARESWARAN, V. ; WANG, Kenneth ; CHEN, Steven ; BEHRINGER, Reinhold ; MCGEE, Joshua ; TAM, Clement ; ZAHORIK, Pavel: *3D Audio Augmented Reality: Implementation and Experiments*. ACM International Symposium on Mixed and Augmented Realty (ISMAR '03), 2003

[Tec] TECHNOLOGY, New A.: *HAEDPHONE SURROUND 3D Webseite*. http://www.newaudiotechnology.de, 22.08.2013

[Tik09] TIKANDER, Miikka: *Development and Evaluation of Augmented Reality Audio Systems*. Dissertation, Helsinki University of Technology, 2009

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

[Xil] XILINX: *Xilinx System Generator for DSP Webseite*. http://www.xilinx.com/tools/sysgen.htm, 22.08.2013

[Xil08] XILINX: *ML50X schematics*. 2008

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

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

[Xil11] XILINX: *UserGuide ML505/ML506/ML507 Evaluation Platform*. 2011

[Xil12a] XILINX: *MicroBlaze Processor Reference Guide Embedded Development Kit EDK 14.1*. 2012

[Xil12b] XILINX: *System Generator DSP User Guide*. 2012

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

A Hardware

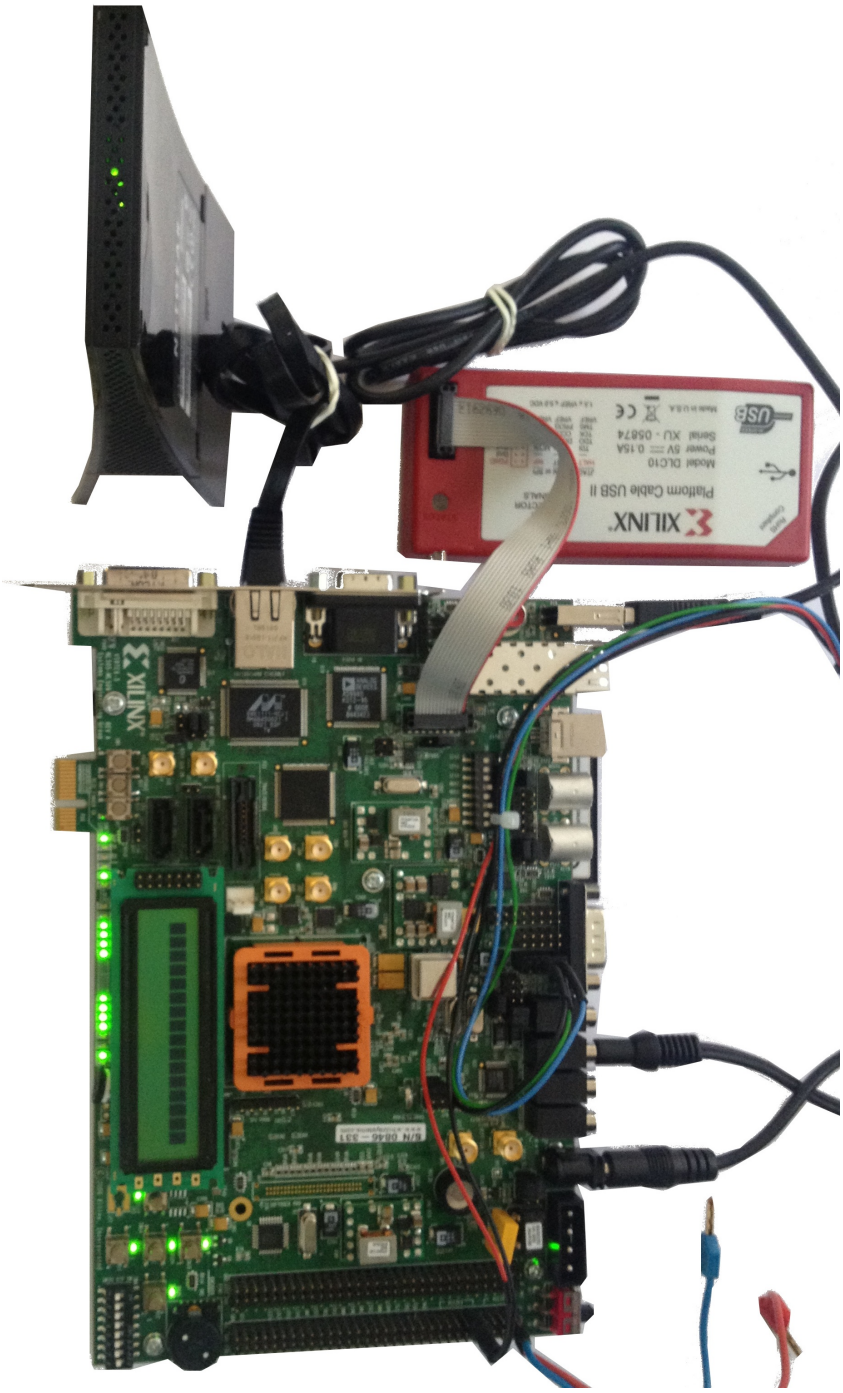


Abbildung 50: Virtex5 ML 507 + Wlan-Adapter

Abbildung 51: Kopfhörer Sennheiser HD 600 + Kompass

B HRTF-Filter

Variable	Precision OP1	Operator	Percision OP2	FULL Precision Result	User defined Precision
t1	UFix_12_0	*	Fix_16_15	Fix_28_16	Fix_17_15
t2	Fix_17_15	*	Fix_16_15	Fix_33_30	Fix_17_15
t3	Fix_17_15	*	Fix_16_15	Fix_33_30	Fix_17_15
t4	UFix_1_0	-	Fix_17_15	Fix_18_15	Fix_17_15
t5	Fix_17_15	*	Fix_16_15	Fix_33_30	Fix_17_15
t6	Fix_17_15	*	Fix_16_15	Fix_33_30	Fix_17_15
t7	Fix_17_15	+	Fix_17_15	Fix_18_15	Fix_17_15
t8	UFix_5_0	*	Fix_16_15	Fix_21_15	Fix_17_15
t9	Fix_17_15	+	Fix_17_15	Fix_18_15	Fix_17_15
t10	Fix_17_15	*	Fix_17_15	Fix_34_30	Fix_17_15
t11	UFix_1_0	-	Fix_17_15	Fix_18_15	Fix_17_15
t12	Fix_17_15	*	Fix_17_15	Fix_34_30	Fix_17_15
t13	Fix_17_15	+	Fix_17_15	Fix_18_15	Fix_17_15

Tabelle 7: Quantisierung der Zwischen- und Endergebnisse. Die Vektorbreiten der Zwischenergebnisse ergeben sich durch Zuschneiden der Full-Precision-Ergebnisse auf den tatsächlich genutzten Wertebereich der einzelnen Rechenschritte.

C Embedded System

Auszug UCF-File

```
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<0> LOC=U25 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<1> LOC=AG27 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<2> LOC=AF25 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<3> LOC=AF26 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<4> LOC=AE27 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<5> LOC=AE26 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<6> LOC=AC25 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<7> LOC=AC24 | IOSTANDARD=
    LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_Hard_Ethernet_MAC_TemacPhy_RST_n_pin LOC=J14 | IOSTANDARD =
    LVCMOS25 | TIG;
Net fpga_0_Hard_Ethernet_MAC_MII_TX_CLK_0_pin LOC = K17 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<0> LOC=AF11 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<1> LOC=AE11 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<2> LOC=AH9 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<3> LOC=AH10 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<4> LOC=AG8 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<5> LOC=AH8 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<6> LOC=AG10 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin<7> LOC=AG11 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TX_EN_0_pin LOC=AJ10 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TX_ER_0_pin LOC=AJ9 | IOSTANDARD =
    LVDCI_33;
Net fpga_0_Hard_Ethernet_MAC_GMII_TX_CLK_0_pin LOC=J16 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<0> LOC=A33 | IOSTANDARD =
    LVCMOS25;
```

```
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<1> LOC=B33 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<2> LOC=C33 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<3> LOC=C32 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<4> LOC=D32 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<5> LOC=C34 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<6> LOC=D34 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin<7> LOC=F33 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RX_DV_0_pin LOC=E32 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RX_ER_0_pin LOC=E33 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_GMII_RX_CLK_0_pin LOC=H17 | IOSTANDARD =
    LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MDC_0_pin LOC=H19 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MDIO_0_pin LOC=H13 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_PHY_MII_INT_pin LOC=H20 | IOSTANDARD =
    LVCMOS25 | TIG;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<0> LOC = AE24 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<1> LOC = AD24 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<2> LOC = AD25 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<3> LOC = G16 | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<4> LOC = AD26 | IOSTANDARD=LVCMOS18 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<5> LOC = G15 | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<6> LOC = L18 | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<7> LOC = H18 | IOSTANDARD=LVCMOS25 |
    PULLDOWN | SLEW=SLOW | DRIVE=2;
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 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 = LVCMOS25;

Net Oszi_4Bit_GPIO_IO_O_pin<0> LOC = H33 | IOSTANDARD=LVCMOS25 |
    PULLDOWN;
Net Oszi_4Bit_GPIO_IO_O_pin<1> LOC = F34 | IOSTANDARD=LVCMOS25 |
    PULLDOWN;
Net Oszi_4Bit_GPIO_IO_O_pin<2> LOC = H34 | IOSTANDARD=LVCMOS25 |
    PULLDOWN;
Net Oszi_4Bit_GPIO_IO_O_pin<3> LOC = G33 | IOSTANDARD=LVCMOS25 |
    PULLDOWN;

Net hrtf_8filter_0_HRTF_FINISH_pin LOC = H32 | IOSTANDARD = LVCMOS25;
Net hrtf_8filter_0_to_microblaze_0_FSL_Has_Data_pin LOC = J32 |
    IOSTANDARD = LVCMOS25;
```

Auszug MHS-File

```
BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0x50000000
 PARAMETER C_ICACHE_HIGHADDR = 0x5fffffff
 PARAMETER C_CACHE_BYTE_SIZE = 65536
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BASEADDR = 0x50000000
 PARAMETER C_DCACHE_HIGHADDR = 0x5fffffff
 PARAMETER C_DCACHE_BYTE_SIZE = 65536
 PARAMETER C_DCACHE_ALWAYS_USED = 1
 PARAMETER HW_VER = 8.40.b
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_FSL_LINKS = 2
 BUS_INTERFACE DPLB = mb_plb
```

```
 BUS_INTERFACE IPLB = mb_plb
 BUS_INTERFACE DXCL = microblaze_0_DXCL
 BUS_INTERFACE IXCL = microblaze_0_IXCL
 BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
 BUS_INTERFACE DLMB = dlmb
 BUS_INTERFACE ILMB = ilmb
 BUS_INTERFACE SFSL1 = fsl_fir_stereo_0_to_microblaze_0
 BUS_INTERFACE MFSL1 = microblaze_0_to_fsl_fir_stereo_0
 BUS_INTERFACE SFSL0 = hrtf_8filter_0_to_microblaze_0
 BUS_INTERFACE MFSL0 = microblaze_0_to_hrtf_8filter_0
 PORT MB_RESET = mb_reset
 PORT INTERRUPT = microblaze_0_Interrupt
END

BEGIN xps_gpio
 PARAMETER INSTANCE = DIP_Switches_8Bit
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_GPIO_WIDTH = 8
 PARAMETER C_INTERRUPT_PRESENT = 0
 PARAMETER C_IS_DUAL = 0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81440000
 PARAMETER C_HIGHADDR = 0x8144ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO = fpga_0_DIP_Switches_8Bit_GPIO_IO_pin
END

BEGIN xps_ll_temac
 PARAMETER INSTANCE = Hard_Ethernet_MAC
 PARAMETER C_NUM_IDELAYCTRL = 2
 PARAMETER C_IDELAYCTRL_LOC = IDELAYCTRL_X0Y4-IDELAYCTRL_X1Y5
 PARAMETER C_PHY_TYPE = 1
 PARAMETER C_TEMAC1_ENABLED = 0
 PARAMETER C_BUS2CORE_CLK_RATIO = 1
 PARAMETER C_TEMAC_TYPE = 0
 PARAMETER C_TEMAC0_PHYADDR = 0b00001
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_TEMAC0_RXFIFO = 16384
 PARAMETER C_BASEADDR = 0x87000000
 PARAMETER C_HIGHADDR = 0x8707ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE LLINK0 = Hard_Ethernet_MAC_llink0
 PORT TemacIntc0_Irpt = Hard_Ethernet_MAC_TemacIntc0_Irpt
 PORT TemacPhy_RST_n = fpga_0_Hard_Ethernet_MAC_TemacPhy_RST_n_pin
 PORT GTX_CLK_0 = clk_125_0000MHz
 PORT REFCLK = clk_200_0000MHzPLL0
 PORT LlinkTemac0_CLK = clk_100_0000MHzPLL0
 PORT MII_TX_CLK_0 = fpga_0_Hard_Ethernet_MAC_MII_TX_CLK_0_pin
 PORT GMII_TXD_0 = fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin
 PORT GMII_TX_EN_0 = fpga_0_Hard_Ethernet_MAC_GMII_TX_EN_0_pin
```

```
 PORT GMII_TX_ER_0 = fpga_0_Hard_Ethernet_MAC_GMII_TX_ER_0_pin
 PORT GMII_TX_CLK_0 = fpga_0_Hard_Ethernet_MAC_GMII_TX_CLK_0_pin
 PORT GMII_RXD_0 = fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin
 PORT GMII_RX_DV_0 = fpga_0_Hard_Ethernet_MAC_GMII_RX_DV_0_pin
 PORT GMII_RX_ER_0 = fpga_0_Hard_Ethernet_MAC_GMII_RX_ER_0_pin
 PORT GMII_RX_CLK_0 = fpga_0_Hard_Ethernet_MAC_GMII_RX_CLK_0_pin
 PORT MDC_0 = fpga_0_Hard_Ethernet_MAC_MDC_0_pin
 PORT MDIO_0 = fpga_0_Hard_Ethernet_MAC_MDIO_0_pin
END

BEGIN xps_gpio
 PARAMETER INSTANCE = LEDs_8Bit
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_GPIO_WIDTH = 8
 PARAMETER C_INTERRUPT_PRESENT = 0
 PARAMETER C_IS_DUAL = 0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81420000
 PARAMETER C_HIGHADDR = 0x8142ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO = fpga_0_LEDs_8Bit_GPIO_IO_pin
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart_2
 PARAMETER C_BAUDRATE = 19200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RX = fpga_0_RS232_Uart_2_RX_pin
 PORT TX = fpga_0_RS232_Uart_2_TX_pin
END

BEGIN xps_ll_fifo
 PARAMETER INSTANCE = Hard_Ethernet_MAC_fifo
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x81a00000
 PARAMETER C_HIGHADDR = 0x81a0ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE LLINK = Hard_Ethernet_MAC_llink0
 PORT IP2INTC_Irpt = Hard_Ethernet_MAC_fifo_IP2INTC_Irpt
END

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 = Hard_Ethernet_MAC_TemacIntc0_Irpt &
     Hard_Ethernet_MAC_fifo_IP2INTC_Irpt &
     fpga_0_Hard_Ethernet_MAC_PHY_MII_INT_pin & xps_ac97_0_Interrupt
 PORT Irq = microblaze_0_Interrupt
END

BEGIN lcd_ip
 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

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

BEGIN xps_gpio
 PARAMETER INSTANCE = Oszi_4Bit
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_GPIO_WIDTH = 4
 PARAMETER C_BASEADDR = 0x81400000
 PARAMETER C_HIGHADDR = 0x8140ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO_O = Oszi_4Bit_GPIO_IO_O
END

BEGIN fsl_v20
 PARAMETER INSTANCE = hrtf_8filter_0_to_microblaze_0
 PARAMETER HW_VER = 2.11.f
```

```
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT FSL_Clk = clk_100_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
 PORT FSL_Has_Data = hrtf_8filter_0_to_microblaze_0_FSL_Has_Data
END

BEGIN fsl_v20
 PARAMETER INSTANCE = microblaze_0_to_hrtf_8filter_0
 PARAMETER HW_VER = 2.11.f
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT FSL_Clk = clk_100_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN hrtf_8filter
 PARAMETER INSTANCE = hrtf_8filter_0
 PARAMETER HW_VER = 1.00.c
 BUS_INTERFACE SFSL = microblaze_0_to_hrtf_8filter_0
 BUS_INTERFACE MFSL = hrtf_8filter_0_to_microblaze_0
 PORT FSL_Clk = clk_100_0000MHzPLL0
 PORT HRTF_FINISH = hrtf_8filter_0_HRTF_FINISH
END
```

FPGA Ressourcenverbrauch

Art	verfügbar	verwendet	in %
Number of Slice Registers	44800	15649	35%
Number of Slice LUTs	44800	20354	45%
Number of occupied Slices	11200	8222	73%
Number of LUT Flip Flop pairs used		25590	
Number of bonded IOBs	640	182	28%
Number of BlockRAM/FIFO	148	96	64%
Number of BUFG/BUFGCTRLs	32	8	25%
Number of IDELAYCTRLs	22	5	22%
Number of BSCANs	4	1	25%
Number of BUFIOs	80	8	10%
Number of DSP48Es	128	77	60%
Number of PLL_ADVs	6	1	16%

Tabelle 8: FPGA Ressourcenverbrauch

D Software des Embedded Systems

Einstellungen lwIP

```
BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = lwip140
 PARAMETER LIBRARY_VER = 1.03.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER LWIP_TCP = false
 PARAMETER TCP_QUEUE_OOSEQ = 0
 PARAMETER MEMP_N_PBUF = 32
 PARAMETER MEMP_N_TCP_PCB = 0
 PARAMETER MEMP_N_TCP_PCB_LISTEN = 0
 PARAMETER MEMP_N_UDP_PCB = 32
END
```

Messungen Ethernetschnittstelle

```
64 bytes from 192.168.1.10: icmp_seq=96 ttl=255 time=0.433 ms
64 bytes from 192.168.1.10: icmp_seq=97 ttl=255 time=0.480 ms
64 bytes from 192.168.1.10: icmp_seq=98 ttl=255 time=0.480 ms
64 bytes from 192.168.1.10: icmp_seq=99 ttl=255 time=0.470 ms
--- 192.168.1.10 ping statistics ---
105 packets transmitted, 100 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.357/0.494/0.670/0.043 ms
```

Abbildung 52: Messung 1 mittels PING

```
64 bytes from 192.168.1.10: icmp_seq=96 ttl=255 time=0.489 ms
64 bytes from 192.168.1.10: icmp_seq=97 ttl=255 time=0.461 ms
64 bytes from 192.168.1.10: icmp_seq=98 ttl=255 time=0.427 ms
64 bytes from 192.168.1.10: icmp_seq=99 ttl=255 time=0.464 ms
--- 192.168.1.10 ping statistics ---
100 packets transmitted, 100 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.402/0.490/0.592/0.031 ms
```

Abbildung 53: Messung 2 mittels PING

E Inhalt der CD-Rom

- Masterarbeit “Eine kopfhörerbasierte, virtuelle Audioumgebung zur räumlichen Positionierung mehrerer Schallquellen mit einer mobilen SoC-Plattform“ (PDF)
- HRTF-Messungen von Sylvia Sima
- Kemar-Kunstkopf HRTF-Messungen des MIT
- MATLAB-Skripte zur Berechnung der Filterkoeffizienten
- MATLAB/Simulink-Projekt des entwickelten HRTF-Filters
- XPS-Projekt der SoC-Plattform
- SDK-Projekt der entwickelten Software
- Java-Projekt und Jar-File des 3d-Audio-Servers
- 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, 25. November 2013	
Ort, Datum	Unterschrift