IEM-Report 11/2001

A Head Position Related Binaural Sound Reproduction System

Piotr Majdak, Markus Noisternig, Robert Höldrich

Zusammenfassung:

Bei Kopfhörerwiedergabe erfordert die originalgetreue Reproduktion eines aufgezeichneten Schallfeldes, sowie die Spatialisation von Monoquellen, eine Filterung der virtuellen Quellen mittels Außenohrübertragungsfunktionen (HRTFs), welche den Unterschied der Signale an den beiden Trommelfellen in Abhängigkeit vom Einfallswinkel der Schallwelle beschreiben. Die menschliche Hörwahrnehmung erlaubt eine Verbesserung der Lokalisation von Schallereignissen mittels kleiner Kopfdrehungen. Um dieses Phänomen in VR Applikationen zugänglich zu machen ist die Implementierung eines Headtrackers notwendig. Bei dynamischen Systemen entsteht die Problematik der zeitvarianten Interpolation zwischen den verschiedenen HRTF-Kurven. Eine Lösungsmöglichkeit bietet die Kodierung der Signale in Ambisonic-Signale mittels zeit-invarianter Filter, wobei die Kopfdrehung mittels einer zeitvarianten Rotationsmatrix berücksichtigt wird.

Ein Prototyp dieses Systems wurde mittels jMax und PD (zweier Echtzeit-Soundverarbeitungstools) auf einem UNIX-Rechner implementiert und auf Qualität und Effizienz überprüft.

Das Ziel dieses Projektes ist eine DSP-Implementierung der oben genannten Algorithmen.

Abstract:

Convincing sound reproduction via headphones requires filtering of virtual sources with head related transfer functions (HRTFs) which describe signal differences at the two ear drums as a function of the source angle. Regarding hearing in natural sound fields humans are able to improve source localization capabilities due to small head movements . To benefit from this phenomenon in VR applications, head tracking has to be incorporated in the time-varying binaural sound reproduction system. This leads to the problem of high-quality time-varying interpolation between different HRTFs. The proposed method solves this problem with a virtual Ambisonic approach which results in time-invariant filters. The influence of the head position is taken into account with a cheap time-variant rotation matrix.

A prototype version of this binaural sound reproduction system has been implemented on a Unix machine running jMax and pd (two real-time sound processing tools originiating in the computer music community) and has been proven to yield high quality binaural signals with reasonable computational effort.

The next step is the hand coded implementation on a DSP.

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Theoretische Grundlagen

1.1.1 Ambisonic Begriffsdefinition

Grundgleichungen für den zweidimensionalen Fall [BAMF]:

Bei Ambisonic ist es üblich das Koordinatensystem um 90° gedreht darzustellen. Eine ebene Welle die aus einer Richtung ψ in Bezug auf die x-Achse eintrifft, soll durch das Ambisonic-System im Zentrum der Hörzone reproduziert werden.

Die ursprüngliche Welle kann nur bei einer unendlichen Anzahl von Übertragungskanälen zu 100% rekonstruiert werden. Bei der Implementierung beschränkt man sich auf ein System m-ter Ordnung, für welches im zweidimensionalen Fall $(2m+1)$ Übertragungskanäle benötigt werden, wobei jeder Ordnung m eine sphärische Harmonische entspricht.

1.1.2 Verwendeter Algorithmus

Der grundlegende Verarbeitungsalgorithmus wird aus folgenden Teilalgorithmen zusammen-gesetzt:

- Kodierung der Audiosignale zu Ambisonic unter Berücksichtigung des Azimuthwinkels der Kopfposition
- Dekodierung und Filterung mit HRTF-Kurven
- Rekonstruktion der Audiosignale mittels gewichteter Summation

Somit ergibt sich der in Abbildung 1.1 dargestellte Algorithmus :

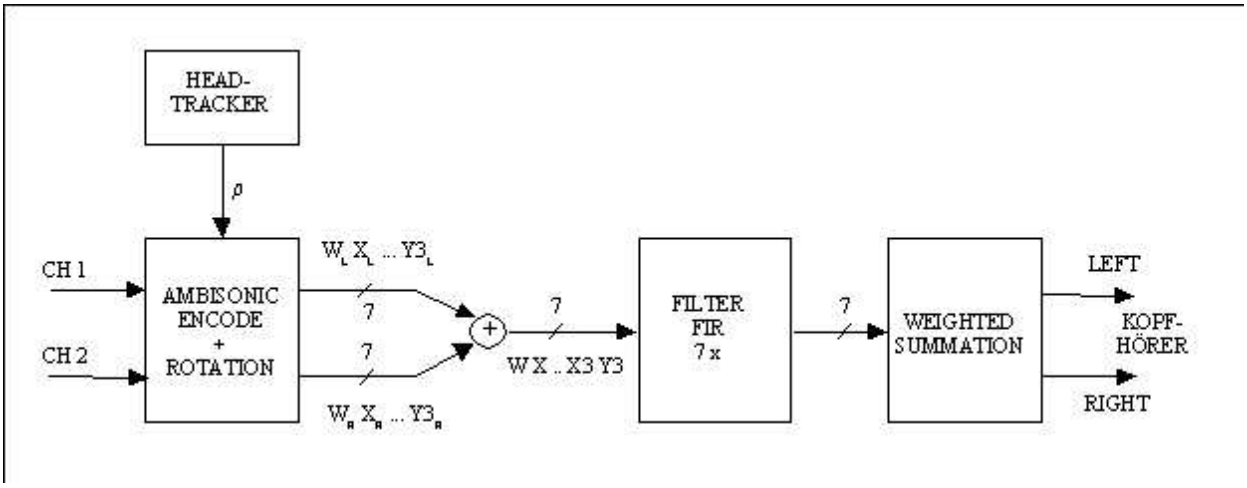


Abbildung 1.1: Grundlegende Umsetzung des Algorithmus

2 Auswahl der Hardware

Bei der Auswahl der Hardware sollten sowohl die technischen wie auch die wirtschaftlichen Aspekte berücksichtigt werden. Weiters sollte die ausgewählte Plattform den zukünftigen Entwicklungen am Institut genügend Spielraum lassen.
Da die technischen Anforderungen durch den vorgegebenen Algorithmus starr vorliegen, gilt es jene Hardware zu finden, bei der sich der zeitliche Entwicklungsaufwand auf ein Minimum reduzieren läßt.

2.1 Anforderungen an die Hardware

Die gesamte Hardware wird in folgende, autonom arbeitende Module unterteilt:

- **Audio-Codec:** Übernimmt die Digitalisierung der analogen Audiosignale am Eingang und die Konvertierung der digitalen Audiodaten vom DSP.
- **Digital Signal Processor (DSP):** Hier findet die gesamte Verarbeitung der Audiodaten statt.
- **Headtracker:** Ermittelt den momentanen Azimuthwinkel der Kopfposition.
- **PC:** Koordiniert die Kommunikation zwischen Headtracker und DSP, und fungiert als Benutzerschnittstelle (Parametereingabe, Datenvisualisierung)

2.2 Abschätzung des Leistungs- und Speicherbedarfes

Bevor die detailierten Vergleiche der jeweiligen DSP-Plattformen verschiedener Hersteller durchgeführt werden, sollte der verwendete Algorithmus auf die benötigte Verarbeitungs-geschwindigkeit und Speicherbedarf untersucht werden.

Wie schon im vorigen Kapitel beschrieben, kann der gesamte Algorithmus in 3 Teile zerlegt werden:

1. Kodierung der Stereo- zu Ambisonicsignalen
2. Filterung der Ambisonic-Signale mit den HRTF-Kurven
3. Dekodierung der Ambisonic-Signale zu Stereosignalen

Folgende Überlegungen gehen von HRTF-Kurven mit einer Länge von 256 bins und einer Bearbeitung der Audiodaten Frame-für-Frame mit einer Frame-Länge von 256 Samples aus. Weiters wird mit einer Samplingrate von 48kHz gerechnet.

2.2.1 Kodierung der Stereo- zu Ambisonicsignalen

Zur Kodierung eines Signals in die Ambisonic-Ebene werden folgende Resourcen benötigt:

Verarbeitungsgeschwindigkeit:

1. 1 Speicherplatz für das Audiosample – 1 LOAD[1];
2. Cosinus-Berechnung, realisiert mit einer Look-Up-Table – 3 LOADs;
3. Sinus-Berechnung, Verwendung der Cosinus-Tabelle (Winkelverschiebung von 90°)
   – 3 LOADs;
4. Speicherplätze für die jeweiligen Ambisonic-Koeffizienten – 7 LOADs;
5. Multiplikation der Audiosamples mit den jeweligen Faktoren – 7 MPYs[2];

Operation	***LOADs***	***MPYs***	***ADDs***[3]
Anzahl pro Sample	14	7	Keine
Anzahl pro Sekunde[4]	672k	336k	0

Insgesamt werden also ca. *1Mio.* Operationen pro Sekunde für die Kodierung der Daten benötigt.

Speicherbedarf: Die Auflösung der cosinus-Tabelle ist direkt abhängig von der Lokalisationsauflösung des menschlichen Gehörs. Die kleinste, dedektierbare Winkeländerung beträgt im günstigsten Fall, d.h. bei Einfall der Schallwelle von vorne ($\varphi = 0°$) ca. 2° bis 3°.

Beschränkt man die Cosinustabelle auf 256 Werte, ergibt sich eine Auflösung von

$$\frac{360°}{256} = 1.4° < 2°$$

Die Ambisonic Kanäle werden in 7 verschiedenen Buffern mit der Länge eines Frames abgespeichert.

Buffer	***Länge***
Cosinustabelle	256
Ambisonic-Kanäle	7 mal 256 = 1792

Es werden also gesamt *2048 Worte* benötigt.

2.2.2 Filterung der Daten

Die HRTF-Filter werden mit einer Länge von 256 bins realisiert ([SONT] , [LEIT]).

Beide Realisierungsmöglichkeiten von HRTF-Filtern sollen auf ihren Leistungsbedarf untersucht werden:

1. im Zeitbereich – direkte FIR-Form, Tap-Filter;
2. im Frequenzbereich – komplexe Multiplikation mit dem Frequenzspektrum des Filters -

   FFT-Form

Zeitbereich:

Beim Tap-Filter wird die Impulsantwort des Filters direkt mit dem jeweiligen Audiosample gefaltet:

$$y(n)=x(n)*h(n)=\sum_{i=0}^{N-1} h(n-i)\cdot x(i) \qquad mit\ N=256 \quad (2.1)$$

Leistungsbedarf:

Folgende Resourcen werden benötigt:

1. Speicherplatz für die 256 letzten Audiosamples – 256 LOADs;
2. Speicherplatz für 256 Punkte der Impulsantwort – 256 LOADs;
3. Multiplikationen der Operanden – 256 MPYs;
4. Summation des Multiplikationsergebnisses – 256 ADDs

Operation	*LOADs*	*MPYs*	*ADDs*
Anzahl pro Sample	512	256	256
Anzahl pro Sekunde	24576k ≅ 25M	12288k ≅ 12.3M	12.3M

Das entspricht in Summe ca. *50Mio* Operationen pro Sekunde, allerdings pro Filter.

Da zur Berechnung 7 Filter simultan benötigt werden, beträgt der tatsächliche Leistungsbedarf bei der direkten Methode: *350Mio.* Operationen pro Sekunde!

Speicherbedarf:

Für die Filterung müssen immer die dem momentanen Sample vorangegangenen 256 Worte gespeichert werden. Da die Länge der Impulsantwort mit der Frame-Länge übereinstimmt, verdoppelt sich die Länge der Ambisonic-Buffer.

Buffer	*Länge*
Impulsantworten	7 mal 256 = 1792
Ambisonicbuffer, zusätzlich	7 mal 256 = 1792
Gefilterte Daten	7 mal 256 = 1792

Es werden also insgesamt 5376 Worte benötigt.

Achtung: Diese Überlegung geht nicht auf implementierungspezifische Probleme wie Quantisierung und Skalierung ein!

Frequenzbereich:

Diese Realisierungsform ist etwas komplexer, verspricht aber eine resourcenschonendere Implementierung.

Um die Filterung im Frequenzbereich durchführen zu können, ist folgende Vorgehensweise notwendig:

1. Speichern der Daten in 2 Buffer. Bei einer Länge der Frequenzantwort von 256 bins, müssen die Buffer jeweils 256 Samples lang sein. Die Daten werden um 128 Samples versetzt abgespeichert, sodaß alle 128 Samples ein Buffer voll wird – dieser wird dann auch bearbeitet. Dadurch entsteht eine Überlappung der Daten, welche die Kontinuität der Information im Frequenzbereich gewährleistet.

2. Multiplikation der Audiodaten mit einer Fensterfunktion:

$$x_w(n) = x(n) \cdot w(n) \quad für \;\; n = 0 \; bis \; 255 \;\; (Länge \, des \, Buffers)$$

3. Transformation der Daten in den Frequenzbereich. Dies geschieht mit Hilfe der diskreten Fouriertransformation (DFT):

$$X_p(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{(2\pi/N)(-nk)} \quad mit \;\; k = -127, -126, \ldots, +127$$

Implementiert wird die Transformation mittels eines FFT-Algorithmus, der den Leistungsbedarf auf $N \cdot \log_2(N)$ MPYs und ADDs reduziert.

4. Filterung der Koeffizienten mit der Frequenzantwort:

$$Y_p(k) = X_p(k) \cdot H(k) \quad für \;\; k = 0 \; bis \; 255$$

Achtung: es handelt sich hier um eine komplexe Multiplikation! Nach der Implementierung ergeben sich 4 Multiplikationen und 2 Additionen!

5. Rücktransformation der Koeffizienten in den Zeitbereich mittels der inversen Fourier-transformation:

$$y_w(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} Y_p(k) \cdot e^{j(2\pi/N)(nk)}, \quad n = 0 \; bis \; 255$$

Hier wird ein IFFT-Algorithmus angewendet, der den Leistungsbedarf auf $N \cdot \log_2(N)$ reduziert.

1. Überlappung der neuen Daten mit dem vorherigen Buffer – Overlap-and-Add:

$$y(n) = y_w(n) + y'(n) \quad mit \;\; n = 0 \; bis \; 255$$

Leistungsbedarf:

Operationen	*LOADs*	*MPYs*	*ADDs*
Fensterfunktion	256	256	Keine
FFT	256 mal 256 = 65k	2048	2048
Filterung	3 mal 256 = 768	4 mal 256 = 1024	2 mal 256 = 512
IFFT	256 mal 256 = 65k	2048	2048
Overlap-and-Add	256	Keine	256

All diese Operationen sind für jeden Ambisonic-Kanal durchzuführen. Findet allerdings die Dekodierung der Ambisonicsignal zu Stereosignalen im Frequenzbereich statt (d.h. bereits vor der IFFT), ist die inverse Fouriertransformation nur mehr pro Audiokanal, also 2 mal, notwendig.

Gesamt ergibt sich also folgender Leistungsbedarf:

Operationen	*Ambisonic-Bereich*	*Zeitbereich*	*Gesamt*
Anzahl pro Kanal	72k	70k	-
Anzahl pro Frame	507k	140k	650k
Anzahl pro Sekunde[5]	190M	52.5M	*ca. 244M*

Verglichen mit der direkten Methode kann man feststellen, daß es uns gelungen ist ca. 60% der benötigten Leistung einzusparen. Allerdings ist damit ein großer Implementierungsaufwand verbunden.

Speicherbedarf:

Folgende Buffer werden benötigt:

Buffer	*Länge*
Doppelte Bufferung der Ambisonic-Kanäle	7 mal 256 = 1792
Fensterfunktion	256
Frequenzantworten – Realteil	7 mal 256 = 1792
Frequenzantworten – Imaginärteil	7 mal 256 = 1792

Buffer	*Länge*
Gefilterte Daten – Realteil	7 mal 256 = 1792
Gefilterte Daten – Imaginärteil	7 mal 256 = 1792
Rücktransformation	7 mal 256 = 1792
Overlap-and-Add	nicht benötigt

Der gesamte Speicherbedarf beträgt also ca. *11000 Worte*.

2.2.3 Dekodierung der Ambisonic-Signale

Die zu realisierenden Algorithmen [SONT] ermöglichen es, bei Verwendung spezieller Filter, die Dekodierung der Daten auf eine einfache Summation mit den Gewichtungsfaktoren zu reduzieren.

Leistungsbedarf:

Operationen	*LOADs*	*MPYs*	*ADDs*
Operationen pro Kanal	7	Keine	7
Operationen pro Sekunde	672k	Keine	672k

Es werden also ca. *1.3 Mio.* Operationen pro Sekunde für die Dekodierung benötigt.

Speicherbedarf:

Die neuen Audiodaten werden direkt in den aktuellen Audiobuffer kopiert, es entsteht kein zusätzlicher Speicherbedarf.

2.2.4 Treiber und Datenverwaltung.

Die Datenverwaltung setzt sich aus dem Audiotreiber und der Initialisierung des Algorithmus zusammen.

Die Initialisierung des Programms findet nur beim Programmstart statt, und unterliegt bezüglich der Verarbeitungsgeschwindigkeit, bzw. des Speicherbedarfes keinen besonderen Vorschriften.

Der Leistungsbedarf des Audiotreibers läßt sich vorab schwer abschätzen, zu diesem Zweck wären detailierte Angaben über die Hardware notwendig. Da die Audiodaten auf jeden Fall in Eingangs-/Ausgangsbuffer kopiert werden müssen, beträgt das Minimum an Operationen 2 LOADs pro Sample. Mit einem Sicherheitsfaktor von 2 und der Shadow-Bufferung (wiederum Faktor 2) ergeben sich ca. *400000 Operationen* pro Sekunde.

Der Speicherbedarf des Audiotreibers ergibt sich aus der Größe eines Frames, welche hier mit 256 Worten festgelegt wurde. Pro Audiokanal werden inkl. Shadow-Bufferung 512 Worte, ingesamt *1024 Worte,* benötigt.

2.2.5 Zusammenfassung.

Es stehen uns also 2 Möglichkeiten der Implementierung zur Auswahl: FIR und FFT/IFFT.

Folgende Tabelle veranschaulicht den Leistungs- und Speicherbedarf der beiden Techniken:

Operation	***Leistungsbedarf***		***Speicherbedarf***	
Kodierung zu Ambisonicsignalen	1Mio		2048	
Filterung	***FIR***	***FFT***	***FIR***	***FFT***
	350Mio	244Mio	5376	11000
Dekodierung	1.3Mio		kein	

Operation	***Leistungsbedarf***		***Speicherbedarf***	
Verwaltung	0.4Mio		1024	
Summe	***FIR***	***FFT***	***FIR***	***FFT***
	ca. 353Mio	*ca. 247Mio*	*ca. 9k*	*ca. 14k*

Daraus folgt, daß die Implementierung des gewünschten Algorithmus sehr hohe Ansprüche auf die Rechenleistung des Prozessors stellt, der Speicherbedarf dagegen eine eher untergeordnete Rolle spielt.

Aus dem Vergleich FIR vs. FFT/IFFT sieht man, daß der FIR-Algorithmus zwar eine höhere Rechenleistung benötigt, im Verhältnis zum FFT/IFFT-Algorithmus der Unterschied nur 60% beträgt – der Grund dafür liegt an der hohen Anzahl von Fouriertransformationen. Deswegen sollte bei der Auswahl der DSP-Plattform auch der Implementierungsaufwand berücksichtigt werden.

Der abgeschätzte Leistungs- und Speicherbedarf sollte mit einem Sicherheitsfaktor von mind. 2 berücksichtigt werden.

In Summe werden also folgende Resourcen benötigt:

Implementierungstechnik	***Leistungsbedarf***	***Speicherbedarf***
FIR	700 Mio Operationen	18k Worte
FFT/IFFT	500 Mio Operationen	28k Worte

2.3 Digitaler Signalprozessor

Entscheidungskriterien und deren Betrachtung:

1. Datenbreite der CPU:

   Da die Audiodaten mit 16 bit digitalisiert werden, wird eine CPU mit einer Mindestdatenbreite von 16 bit benötigt. Die Multiplikation zweier 16 bit Zahlen ergibt eine 32 bit Zahl, wiederholt man diese Multiplikation 256 mal (entspricht der Frame-Größe), so erreicht das Ergebnis eine maximale Größe von 40 bit. Die CPU sollte also eine „16x16=32“-bit-Multiplikation und eine 40-bit-Addition unterstützen um eine weitere Skalierung zu vermeiden, welche wiederum zu einem erhöhten Leistungsbedarf führen würde. Eine optimale Lösung würde eine 32-bit-Fließkomma-CPU bieten, auch hinsichtlich etwaiger zukünftiger Entwicklungen.

2. Verarbeitungsgeschwindigkeit der Daten:

   Die nötige Verarbeitungsgeschwindigkeit beträgt *500 bzw. 700 Mio.* Operationen pro Sekunde (siehe vorheriges Kapitel). Bei einem Fließkommaprozessor spricht man von *500 bzw. 700 MFLOPS* (Mega Flotaing Point Operations Per Second).

3. Speichergröße:

   Die meisten DSPs besitzen einen schnellen und einen langsamen Speicher. Der schnelle Speicher wird meist im Prozessor integriert (on-die), dadurch ist er in seiner Größe sehr beschränkt. Die hohe Zugriffsgeschwindigkeit erreicht man durch eine hohe Anzahl der Datenleitungen, mit der die CPU an den Speicher angebunden wird. Manchmal fungiert dieser Speicher auch nur als Cache-Speicher. Der Langsame Speicher wird extern ausgeführt, dadurch ist man in der Anzahl der Datenleitungen sehr beschränkt, was die Zugriffs-geschwindigkeit herabsetzt, jedoch besteht kaum eine Einschränkung bezüglich der Größe des Speichers.

   Der benötigte Speicher muß folgende Bedingung erfüllen: Die CPU muß die vorgegebene Verarbeitungsgeschwindigeit bei Zugriff auf diesen Speicher beibehalten können – der errechnete Speicherbedarf von *18 bzw. 28 kWorte* bezieht sich also auf den internen, schnellen Speicher. Bei der Datenbreite von 16 bit entspricht der Speicherbedarf *36 bzw. 52kByte*.

4. max. Speicherbereich:

   Sollte für zukünftige Entwicklungen zur Verfügung stehen, spielt hier eine untergeordnete Rolle.

5. Anbindung der Peripherie (Audio-Codec):

   Eine Anbindung eines Codecs mit Audio-Qualität (48kHz sampling rate und mind. 16 bit Wortbreite) sollte möglich sein.

6. Entwicklungssoftware:

   Es sollte ein Assembler und Debugger zur Vefügung stehen, wenn möglich sollte auch ein C-Compiler vorhanden sein.

7. Qualität der Dokumentation:

   Hängt direkt mit dem Zeitaufwand der Entwicklung zusammen – je besser die Dokumentation und der Support des Herstellers, desto schneller und reibungsloser verläuft die Entwicklung.

8. Wirtschaftlichkeit:

   Der Entwicklungsaufwand sollte proportional zum Anschaffungspreis der DSP-Plattform stehen.

2.3.1 Elite-Programm von Texas Instruments

Die Firma Texas Instruments (TI) bietet den Universitäten bei DSP-bezogenen Projektarbeiten kostenlose DSP-Plattformen sowie technische Unterstützung an. Im Gegenzug dafür erhält TI die gesamten Unterlagen nach Abschluß des Projektes.

Da auch das Institut für Informatik der Technische Universität Graz mit TI kooperiert, haben wir eine Anfrage an Hr. *Richard Oed* von TI-München gestellt. Kurz darauf wurde uns das Projekt genehmigt – jetzt galt es nur noch einen geeigneten DSP von TI zu wählen.

Die DSP's von TI decken eine sehr breite Palette der Anwendungen ab [TEXI]– unsere Anforderungen wurden durch den Prozessor *TMS320C6711* gedeckt:

- 32-bit Floating Point CPU, 32-bit Fix Point,
- 16x16=32 MPYs und double long (40bit) Unterstützung
- 900 MFLOPS bei 167MHz Takt
- 64kByte interner Speicher, gemeinsam für Daten und Programm nutzbar, auch als Second-Level-Cache konfigurierbar
- 4kB First-Level-Cache für den Programmspeicher
- 4kB First-Level-Cache für die Daten
- 2 serielle Schnittstellen (Anbindung eines Audio-Codec's)

- EDMA-Controller
- 32-bit Timer

TI liefert für den C6711-DSP ein DSP Starter Kit (DSK) mit Anschluß für einen Debugger, 16MB externen RAM, einen Mono-Audio-Codec und Erweiterungssteckplatz. Weiters gibt es auch ein Evaluation Kit mit einem Audio-Codec, der unseren Anforderungen entspricht. Dieser wird über den Erweiterungssteckplatz mit dem DSK verbunden. Somit: ist der geeignete DSP für dieses Projekt: **TMS320C6711 von Texas Instrument.**

Bestellt wurde das 320C6711 Starter Kit, European Version mit der Bestellnummer TMDS320006711E.

2.4 Audio-Codec

Anforderungen an den Audio-Codec:

1. 44.1 bzw. 48kHz Samplingrate
2. mind. 16 bit Wortbreite
3. Anbindung an den DSP über eine serielle Schnittstelle
4. Evaluation Kit mit Erweiterungssteckplatz für das DSK vorhanden
5. Softwaretreiber vorhanden

Da Texas Instruments sich auf die Audioübertragung mit geringer Übertragungsrate spezialisiert hat (Anwendungsbereich in der Telekommunikation), ist bei TI nur ein Audio-Codec erhältlich, der den Anforderungen des Projektes entspricht: *TLV320AIC27*.

Folgende Kenndaten spezifizieren diesen Codec:

- 18 bit Stereo ADC/DAC
- SNR > 95dB
- Integrierte Lautstärkeregelung
- Line Level Outputs
- AC97 Rev. 2.1 kompatibel
- Lieferbar auf einem Evaluationsboard, passend für alle DSK's von TI

Da zu diesem Codec kein Audiotreiber von TI erhältlich ist, mußte dieser im Rahmen dieses Projektes erstellt werden.

Da auch andere Hersteller keine bessere Lösung anbieten konnten, wurde das Stereo-Audio-Codec-Evaluationsboard **TLV320AIC27 EVM von TI** mit der Bestellnummer TLV320AIC27 bestellt.

2.5 Headtracker

Der am Institut vorhandene Headtracker **Flock of Birds** von der **Firma Ascension Technology** stellt eine allgemeine Lösung in der Erfassung von Bewegung dar und kann für unsere Zwecke benutzt werden:

- Erfassung von 6 Freiheitsgraden
- 20 bis 144 Messungen pro Sekunde
- maximale Entfernung des Sensors von der Sendeeinheit: ca. 1.5 Meter
- Serielle Datenübertragung über RS-232C

2.6 PC

Da der DSP C6711 keine asynchrone serielle Schnittstellen (RS-232C) unterstützt, muß der Headtracker über einen Router angeschlossen werden. Weiters sind die Möglichkeiten der Benutzerschnittstelle (User Interace - UI) mit dem DSK sehr beschränkt – der Router muß also auch die gesamte Parametrierung und Visualisierung der Daten übernehmen.

Zwei Möglichkeiten wurden in Erwägung gezogen:

- eigene Entwicklung der Hardware: Mikrocontroller
- handelsüblicher PC

  Aus folgenden Gründen wurde ein handelsüblicher PC als Datenrouter und UI ausgewählt:

- Anbindung von DSK an den PC durch Treiber von Texas Intruments vorhanden

- Einschlägige Erfahrungen mit der Programmierung der seriellen Schnittstelle am PC
- Möglichkeit eine komfortable Benutzerschnittstelle zu implementieren

Gegen die Mikrocontroller-Lösung spricht die Notwendigkeit einer zusätzlichen Hardware-Entwicklung, welche einen enormen Mehraufwand darstellt.

Anforderungen an den PC:

- eine parallele Schnittstelle auf EPP konfiguriert
- eine serielle Schnittstelle (9600 baud)
- mind. 64MB Arbeitsspeicher
- Intel P3 350MHz bzw. AMD K6-II 350MHz oder schneller
- Betriebssystem: Microsoft Windows 32-API kompatibel (Windows 95, 98, NT4.0, 2000...)
- Bildschirm Auflösung von mind 800x600.

3 Programmierung

Die Implementierung des Ambisonic-Algorithmus stellt die Hauptaufgabe dar – ist jedoch nur ein kleiner Teil des Projektes. Die folgende Auflistung aller Arbeitsschritte stellt eine kurze Vorschau auf die in den folgenden Kapiteln beschriebenen Details dar:

- Erstellung einer Arbeitsumgebung am PC: Auf den Zusammenbau des PCs, Installation des Betriebssystems und Zusatzsoftware wird hier nicht näher eingegangen.
- Errichtung der Programmierumgebung: Die Installation der von Texas Instruments mitgelieferten Software verlief problemlos. Anzumerken ist jedoch, daß die Programmierumgebung „Code Composer Studio“ nicht sehr laufstabil ist. Die gesamte Programmierung erfolgte aus diesem Grund mittels Texteditor, der Kompiliervorgang mittels eigens dafür erstellten Make-Dateien.

- Verbindung des Audio-Codecs mit dem DSK: Dieser Punkt wird zusammen mit dem Konzept des Audiotreibers im Kapitel 3.1 ausführlich beschrieben.
- Technische Details der Implementierung des Ambisonic-Algorithmus: Werden im Kapitel 3.2beschrieben
- Datenübergabe DSK ↔ PC: Damit beschäftigt sich Kapitel 3.3
- Hostprogramm am PC: Ausführliche Beschreibung in Kapitel 3.4

3.1 Audio-CODEC

3.1.1 Allgemein

Folgende Hardwareeinstellungen sind am Codec-Board vorzunehmen (siehe [EVMU]):

Jumper	***Einstellung***	***Bedeutung***
J21	offen	EVM-eigenen Spannungsregler abschalten
J7	2-3 verbunden	DV_{dd} auf 3.3V setzen
J43	offen	AV_{dd} auf 12V unterbrechen
J44	verbunden	AV_{dd} auf 5V setzen
J17	offen	externen Takt benutzen
J18	verbunden	

Erst dann darf das Codec-Board mit dem DSP-Trägerboard verbunden werden.

3.1.2 AC'97 Protokoll

Die gesamte Kommunikation mit dem Audio-CODEC TLV320AIC27 erfolgt über das Protokoll AC'97 Rev.2.1. Dieses Protokoll (siehe [AC97]) wurde von Intel 1997 festgelegt und wurde zum Standardprotokoll bei der Anbindung an Audio-Codecs im PC. Folgende Merkmale spezifizieren AC'97:

- bidirektionale serielle Datenverbindung
- feste Datenrate: Takt mit 12.288 MHz
- 13 slots bilden ein Frame, die Frame-Frequenz ist mit 48kHz fixiert
- mehrere Konfigurationsmöglichkeiten (bis zu 8 Audiokanäle möglich)
- Simultane Übertragung von Audio- und Konfigurationsdaten

Da in unserem Fall nur eine Stereoverbindung notwendig ist, wird der Codec im Basic-Mode, auch Two-Channel-Mode genannt, betrieben. Die Abbildung 3.1 zeigt das Blockschaltbild, benutzt werden die Eingänge `LineIn` und Ausgänge `LnLvlOut`.

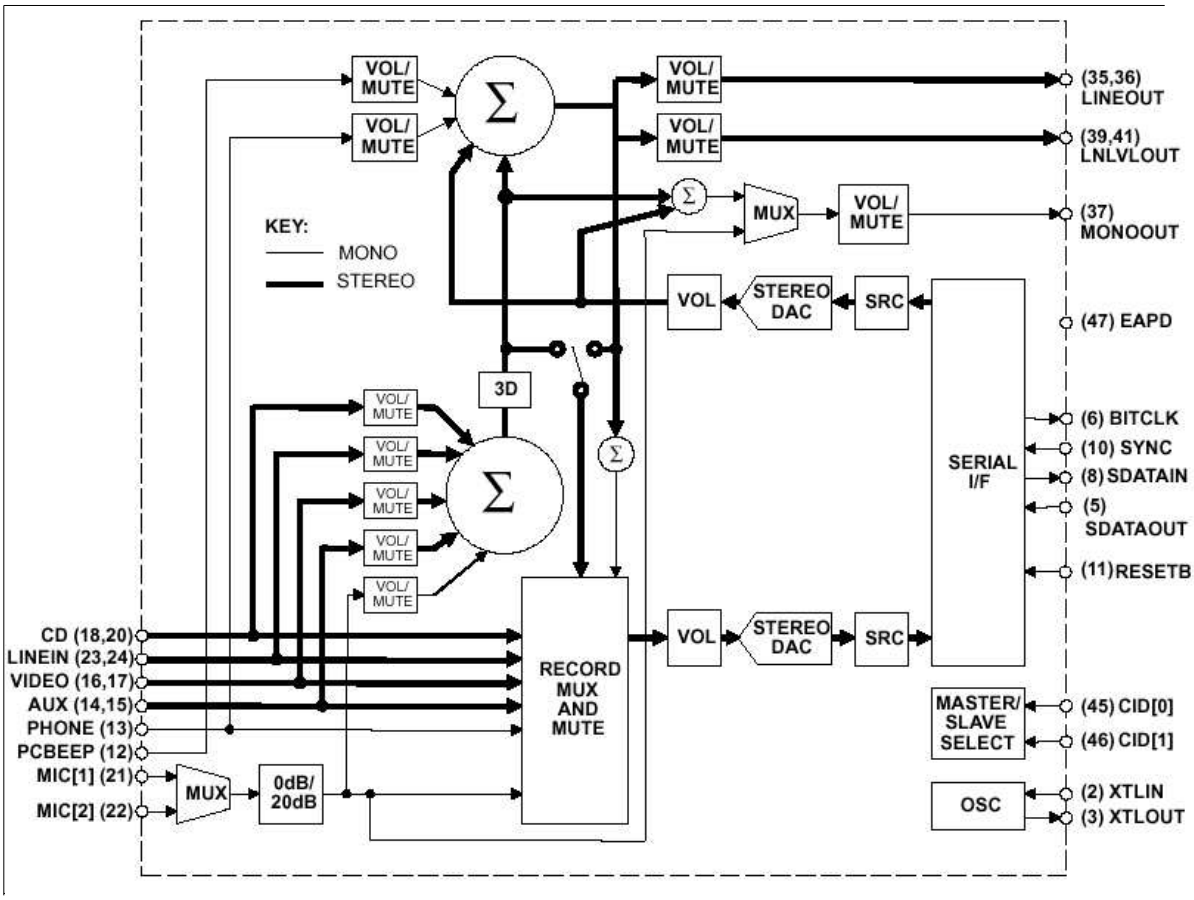


Abbildung 3.1: Blockschaltbild des Audio-Codec, Basic-Mode

Das AC'97-Protokoll im Basic-Mode wird wie folgt spezifiziert:

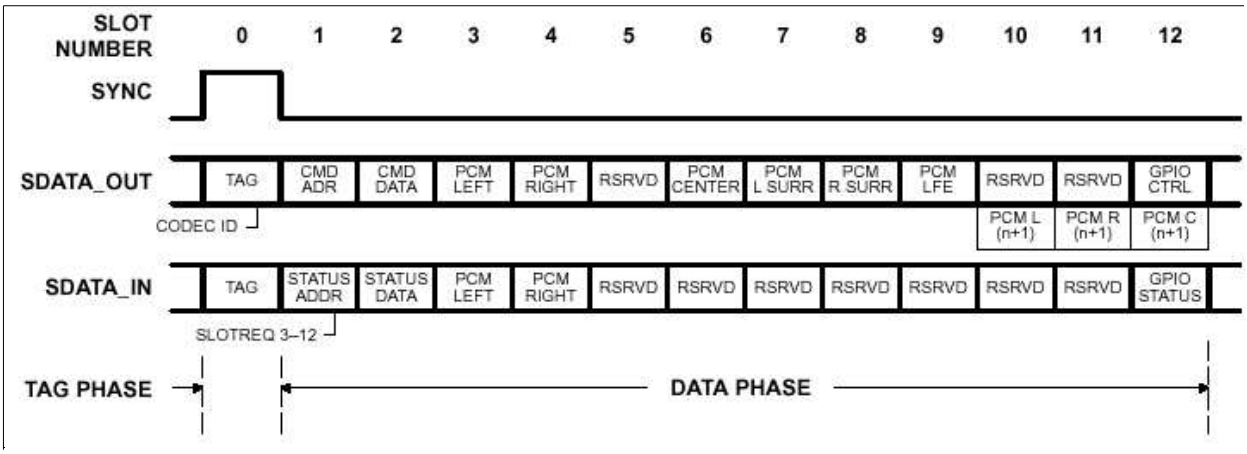


Abbildung 3.2: AC'97-Protokoll im Basic-Mode

Wichtig ist dabei die Nomenklatur:

- **Slot**: Beinhaltet einzelne Bits. Slot 0 ist 16bit, alle weiteren Slots 20bit lang
- **Phase**: Aneinanderreihung von Slots: Die Tag-Phase besteht aus einem einzigen Slot, dem Tag-Slot. Die Data-Phase besteht aus den restlichen Slots. Diese Unterscheidung besteht, um den Tag-Slot von den restlichen Slots logisch abtrennen zu können.
- Die Tag-Phase und Data-Phase bilden einen Audio **Frame**.

Diese Einteilung wird in der Abbildung 3.3 verdeutlicht. Weiters sind dort auch die für die Übertragung notwendigen Signale zu sehen.

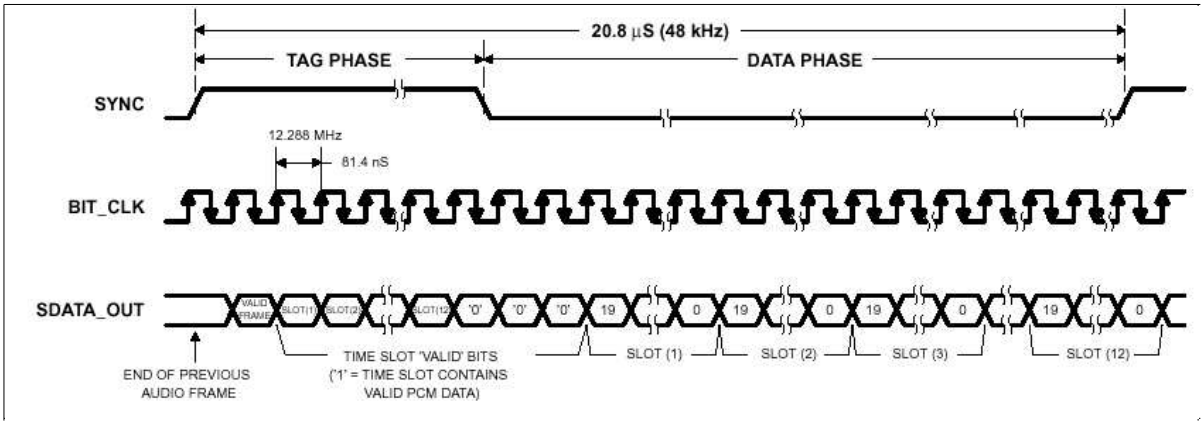


Abbildung 3.3: AC'97-Protokoll, Details eines Frames

Die zu übertragenden Daten wurden in Slots organisiert, wobei der Slot 0 in der Tag-Phase eine besondere Bedeutung hat – er beinhaltet Informationen über die Gültigkeit aller anderen Slots in der Data Phase:

Bits	***Bedeutung bei gesetzten Bits***
0	der gesamte Frame ist gültig
1 bis 12	die jeweiligen Slots 1 bis 12 sind gültig
13-15	keine Bedeutung

Wird ein Slot als ungültig markiert, muß er trotzdem übertragen werden, damit das gesamte Frame immer dieselbe Länge von $L = 12 \cdot 20 + 16 = 256 bits$ besitzt.

Bei der Übertragung DSP → Codec haben die Slots in der Data-Phase folgende Bedeutung:

Slot	***Bezeichnung***	***Bedeutung***	***Wird benutzt bei***
1	CMD ADR	Adresse des Registers für Befehlsübergabe	Initialisierung
2	CMD DATA	Daten des Registers	Initialisierung
3	PCM LEFT	Daten für DAC, links	Datenübertragung
4	PCM RIGHT	Daten für DAC, rechts	Datenübertragung
5	reserved	keine	-
6	PCM CENTER	Daten für Zusatz-DAC	-
7	PCM L SURR	Daten für Zusatz-DAC	-
8	PCM R SURR	Daten für Zusatz-DAC	-
9	PCM LFE	Daten für Zusatz-DAC	-
10	reserved	Daten für Zusatz-DAC	-
11	reserved	Daten für Zusatz-DAC	-
12	GPIO CTRL	General-I/O-Control	-

Die Bedeutung der Slots bei der Übertragung Codec → DSP:

Slot	*Bezeichnung*	*Bedeutung*	*Wird benutzt bei*
1	STATUS ADR	Bestätigung der Adresse des Befehlsregister	Initialisierung
2	STATUS DATA	ausgelesene Daten einer Registers	Initialisierung
3	PCM LEFT	Daten von ADC, links	Datenübertragung
4	PCM RIGHT	Daten von ADC, rechts	Datenübertragung
5 - 11	reserved	keine	-
12	GPIO CTRL	General-I/O-Control	-

Die Slots 1 und 2 werden für die Initialisierung des Codecs benutzt. Nach einem Reset muß der Codec wie folgt initialisiert werden:

- Record gain: 0dB, not muted
- Record select: line in
- Master volume: 0dB, not muted
- Line-level volume: 0dB, not muted
- PCM-out volume: 0dB, not muted

Nach dieser Initialisierung werden nur noch die Slots 3 und 4 (PCM LEFT, PCM RIGHT) benutzt. Für eine detailierte Beschreibung der einelnen Register des Codecs siehe [EVMD].

Sync-Impuls:

Jeder neue Frame wird mit einem Sync-Impuls begonnen (siehe Abb 3.3). Für die Generierung dieses Impulses ist das Steuergerät, in unserem Fall der DSP, zuständig. Der Sync-Impuls hat eine Periodendauer von 256bit, die ersten 16bit am Anfang sollten sich im Zustand HIGH befinden (Länge der Tag-Phase), wobei mit einem bit ein LOW-HIGH-Wechsel des BIT_CLK-Signals gemeint ist. Da das BIT_CLK Signal vom Codec selbst generiert wird, ist es mit einem 8bit Teiler und Zusatzgattern möglich den Sync-Impuls zu erzeugen. Diese Schaltlogik befindet sich bereits auf dem Evaluationboard, muss jedoch noch konfiguriert werden. Da der Teiler am Anfang auf 0 gesetzt werden soll, wird eine freikonfigurierbare Leitung des DSP's DB_TOUT0 als Teiler-Reset benutzt.

Folgende Konfiguration ergibt sich:

Jumper	*Einstellung*	*Bedeutung*
J31	verbunden	BIT_CLK zum Teiler
JP14	verbunden	AC97_SYNC auf SYNC des Codecs legen
JP23	verbunden	
J37	verbunden	SYNC auf DB_FSR0 des DSPs legen
J38-9 mit J33-unten	verbunden	DB_TOUT0 auf RESET des Teilers legen

Zur Datenübertragung müssen weitere Signale berücksichtigt werden:

- DB_DX0: der Datenausgang des DSPs (Data Transmit)
- DB_DR0: der Dateneingang des DSPs (Data Receive)
- DB_CLKX0: Takt für die Sendeeinheit (Clock Transmit)
- DB_CLKR0: Takt für die Empfangseinheit (Clock Receive)
- DB_FSX0: Frame-Signal für die Sendeeinheit (Frame Sync Transmit)
- DB_FSR0: Frame-Signal für die Empfangseinheit (Frame Sync Receive)

Die Bezeichnung DB_ bedeutet, daß die Signale zum Daughter Board führen, die nach-stehende 0 zeigt, daß es sich um die erste serielle Schnittstelle des DSPs handelt.

Um den Codec mit DSP verbinden zu können, muß folgende Konfiguration hergestellt werden:

DSK	*CODEC-EVM*
DB_DX0	SDATA_OUT
DB_DR0	AC97_SDATA_IN0
DB_CLKX0	X_CLKX0
DB_CLKR0	X_CLKR0

DSK	*CODEC-EVM*
DB_FSX0	SYNC
DB_FSR0	X_FSR0

Die Verbindungen mit Masse und Versorgungsspannungen wurden hier nicht berücksichtigt.

Weiters muß auch noch eine RESET-Leitung zum Codec geführt werden. Wie beim Sync-Teiler wird auch hier eine freikonfigurierbare Leitung des DSPs benutzt (DB_TOUT1).

Um die gesamte Anbindung herstellen zu können, müssen noch folgende Jumper gesetzt werden:

Jumper	*Einstellung*	*Bedeutung*
J40	verbunden	BIT_CLK auf DB_CLKX0 legen
J41	verbunden	BIT_CLK auf DB_CLKR0 legen

Folgende Reihenfolge bei der Initialisierung des Codecs sollte beibehalten werden:

1. TOUT1 auf LOW: Codec wird zurückgesetzt.
2. TOUT0 auf HIGH: Der Sync-Teiler wird zurückgesetzt.
3. Die serielle Schnittstelle des DSP's initialisieren.
4. TOU0 = LOW: Den Sync-Teiler freigeben.
5. Die SDATA_OUT-Leitung auf LOW halten (siehe [AC97]).
6. TOU1=HIGH: Den Codec einschalten. Ab jetzt liefert der Codec das BIT_CLK-Signal und der Teiler generiert alle 256bit ein Sync-Signal.

3.1.3 Serielle Schnittstelle des DSPs: McBSP

Der DSP C6711 besitzt eine synchrone serielle Schnittstelle, die über den Multiplexer

U34A und U35A am Erweiterungssteckplatz J3 ausgeführt wird (siehe [DSCH]). Zur Anbindung des Daugtherboards mit dem Codec stehen zwei 80pin Steckverbindungen zur Verfügung. Die Konfiguration des EVM-Boards des Codec wurde im vorherigen Kapitel erläutert. Die serielle Schnittstelle McBSP (Multi Channel Buffered Serial Port) des DSPs muß nun so konfiguriert werden, daß eine Datenübertragung zum und vom Codec stattfinden kann.

Die detailierte Erklärung der McBSP würde den Rahmen dieser Dokumentation überschreiten, die McBSP wird in [PERI] erklärt. An dieser Stelle sei nur die gewählte Konfiguration der Schnittstelle dargestellt.

1. 2 Phases
2. Phase 1: One slot, 16-bit/slot
3. Phase 2: 12 slots, 20-bit/slot
4. Transmitter/Receiver Enabled
5. Right-Justified and sign extension on MSB (Codec liefert Zahlen mit Vorzeichen)
6. No Loopback or test mode
7. Normally clocking
8. DX-Enabled disabled
9. xINTM driven by xRDY
10. Frame Sync Generator and Sample Rate Generator OFF

3.1.4 EDMA

Laut Spezifikation des AC'97-Protokolls wird alle 1.627µs ein Slot übertragen. Um den DSP von der Datenübertragung zu entlasten, wird die gesamte Übertragung in zwei Teile zerlegt:
- die Initialisierung des Codec's und
- die eigentliche Audio-Datenübertragung.

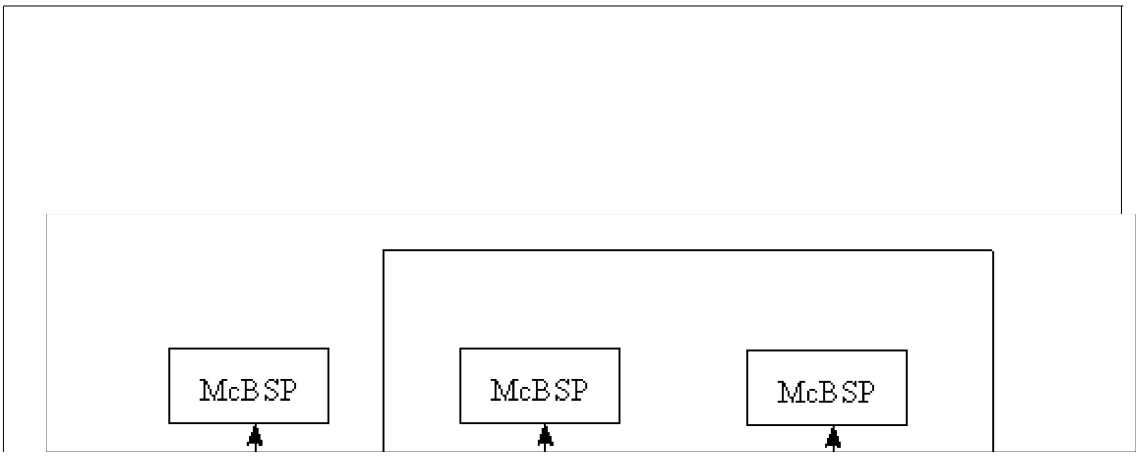
Für die Initialisierung des Codec's stehen folgende Funktionen zur Verfügung
- `McBSP0_Read()` und
- `McBSP0_Write()`,

welche jeweils einen Slot von der McBSP lesen bzw. auf die McBSP ausgeben. Da die Initialisierung nicht zeitkritisch abläuft (keine Berechnungen während der Initialisierung), ist

diese Methode ausreichend.

Für die Übertragung der Audio-Daten wird folgende Vorgehensweise angewendet:

- Die McBSP wird so konfiguriert, daß nur die Slots TAG, PCM LEFT und PCM RIGHT übertragen werden (Multichannel Funktionen werden ausgenutzt).
- Ein DMA-Kanal wird für die Übertragung der Audiodaten vom DSP-Speicher zur McBSP konfiguriert, ein weiterer Kanal für die Übertragung der Audiodaten vom McBSP in den DSP-Speicher. Somit übernimmt der EDMA-Controller (siehe [PERI]) die Datenübertragung.
- Die Länge der Buffer für die Audiodaten wurde auf 256 Frames (3 mal 256 slots) festgelegt, somit dauert ein EDMA-Durchlauf 5.333ms.
- Sobald die Audiobuffer übertragen wurden, wird ein Interrupt ausgelöst, die CPU verzweigt zur Funktion `EDMA_ISR()` - dort findet später die gesamte Berechnung der Audiodaten statt.
- Damit die im Speicher stehenden Daten während der gesamten 5.333ms für die Berechnungen zur Verfügung stehen, benötigt man eine doppelte Bufferung (shadow buffering). Zu diesem Zweck werden die zwei bereits konfigurierten EDMA-Kanäle automatisch (über link address, siehe [PERI], Linking EDMA Transfer) auf zwei zusätzliche EDMA-Kanäle verzweigt, welche die weiteren Daten in andere (shadow) Buffer kopieren. Die Abbildung 3.2 veranschaulicht diesen Prozeß (nur Datenfluß zum Codec dargestellt, Eventnummern eingekreist dargestellt):
- In der Interrupt-Service-Funktion `EDMA_ISR()` werden die aktuellen Audiodaten in weitere Buffer `sInputL` und `sInputR` kopiert, damit sie für weitere Berechnungen fortlaufend im Speicher stehen. Danach wird auch die Funktion `FrameServiceRoutine()` aufgerufen, in der die gesamten Berechnungen durchgeführt werden. Die berechneten Daten sollten in die Buffer `sOutputL` und `sOutputR` kopiert werden. Sie werden in weiterer Folge von `EDMA_ISR()` wieder in ein für den EDMA-Controller zugängliches Format konvertiert.



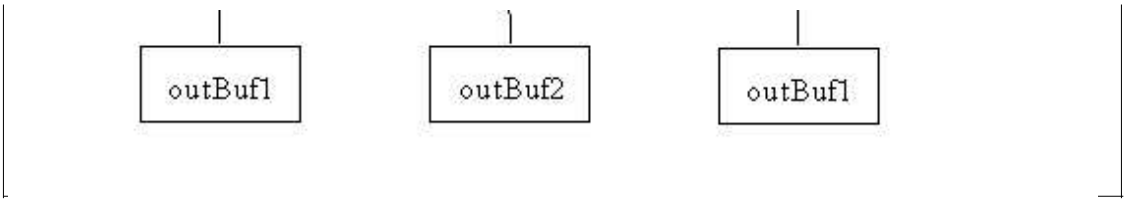


Abbildung 3.4: Ablauf von EDMA-Events für Ausgangsaudiodaten

3.2 Implementierung des Algorithmus

Zunächst werden die jeweligen Cosinus-Faktoren aus den Winkeln der Quellen (`HPI_DeltaN`) berechnet.

```
iX = (HPI_Phi1 + HPI_DeltaN) % 0x100;
sFactorX_1 = sCos[iX];
sFactorY_1 = sCos[(iX+64) & 0xFF];
sFactorU_1 = sCos[(iX*2) & 0xFF];
sFactorV_1 = sCos[(iX*2+64) & 0xFF];
sFactorX3_1 = sCos[(iX*3) & 0xFF];
sFactorY3_1 = sCos[(iX*3+64) & 0xFF];
iX = (HPI_Phi2 + HPI_DeltaN) % 0x100;
sFactorX_2 = sCos[iX];
sFactorY_2 = sCos[(iX+64) & 0xFF];
sFactorU_2 = sCos[(iX*2) & 0xFF];
sFactorV_2 = sCos[(iX*2+64) & 0xFF];
sFactorX3_2 = sCos[(iX*3) & 0xFF];
sFactorY3_2 = sCos[(iX*3+64) & 0xFF];
```

Danach werden in einer Schleife die Ambisonic-Kanäle aus den Audiodaten berechnet:

```
#pragma MUST_ITERATE (FRAMES_NUMBER, , 8);
for(iX=0;iX<FRAMES_NUMBER;iX++)
{
EncodeSampleL(iX);
EncodeSampleR(iX);
}
```

Die #pragma Anweisung vor der Schleife sorgt für Optimierungen beim Kompilieren – siehe [PROG].

Achtung: die Ambisonic-Buffer sind nicht 256 sondern 512 Worte lang, da die FIR-Funktion die vorhergegangenen 256 Worte zur Filterung benötigt. Deswegen werden die neuen Werte ab dem Index `FRAME_NUMBER` (=256) abgespeichert.

Die Filterung erfolgt lauf Formel 2.1, die jeweiligen Multiplikationsergebnisse müssen wegen der Überlaufgefahr skaliert werden: Eine Multiplikation zweier 16bit Zahlen ergibt eine 32bit Zahl. Werden 256 solcher Zahlen zusammengezählt ergibt das eine Zahl mit einer max. Länge von 40bit. Da das endgültige Ergebnis der Filterung nur 16bit lang sein darf, müßte man jede Zahl nach der Filterung um 24bit nach rechts skalieren. In der Praxis würde das so entstandene Signal äußerst leise ausfallen, weil weder das Eingangssignal noch die Impulsantworten die maximale Energie besitzen, die aber notwendig wäre um die 40bit Zahl zu füllen. Es kann also mit weniger als 24bit skaliert werden – unter der

Berücksichtigung der Energie der Impulsantworten. Zu diesem Zweck wird bei der Initialisierung des Programms jede Impulsantwort auf ihren Energiegehalt untersucht (es wird die Gesamtenergie berechnet) und mit dem maximal möglichen Energiegehalt der 16bit Zahlen verglichen. Der Unterschied, in bit ausgedrückt, ergibt die Anzahl der bits um die es *nicht* skaliert werden muß. Da die Pegelverhältnisse der einzelnen Ambisonic-Kanäle zueinander immer gleich bleiben müssen, wird bei allen Filterungen um dieselbe Zahl skaliert – entsprechend dem Skalierungsmaximum aller Impulsantworten.

Für die FIR-Berechnung steht von Texas Instruments eine Funktion namens `fir_r8()` zur Verfügung, bei der allerdings keine Skalierung vorgenommen werden kann, das Multiplikations-ergebnis wird außerdem nur in einem 32bit Register abgespeichert. Deswegen wurde eine eigene FIR-Funktion `myfir_r8()` geschrieben, die das Ergebnis in einem 40bit Register behält und jedes Sample um die Zahl `uiDivBits` skalieren kann:

```
  void myfir_r8 (short x[restrict], short h[restrict], short
r[restrict])
  {
  int i, j;
  long sum;
  #pragma MUST_ITERATE (FRAMES_NUMBER,,8);
  for (j = 0; j < FRAMES_NUMBER; j++)
  {
  sum = 0;
  #pragma MUST_ITERATE (FRAMES_NUMBER, , 8)
  for (i = 0; i < FRAMES_NUMBER; i++)
  sum += (int)x[i + j] * (int)h[FRAMES_NUMBER-i-1];
  r[j] = (int)(sum >> uiDivBits);
  }
  }
```

Diese Funktion wird für jeden Ambisonic-Kanal mit den dazugehörigen Impulsantworten aufgerufen.

Die neuen Werte werden in Buffern mit Endung `0` (zB.: `sWBuffer0`) gespeichert, die `FRAME_NUMBER` (=256) lang sind.

Dekodierung:

Zur Implementation eines Lautstärkereglers werden die Daten mit einem Verstärkungsfaktor `HPI_OutputGain` multipliziert.

```
  #pragma MUST_ITERATE (FRAMES_NUMBER, , 8);
  for(iX=0;iX<FRAMES_NUMBER;iX++)
  {
  // decode to audio
  DecodeSampleL(iX);
  DecodeSampleR(iX);
  // gain the output
  iY = sOutputL[iX] * (short)HPI_OutputGain;
  sOutputL[iX] = iY >> 16;
  iY = sOutputR[iX] * (short)HPI_OutputGain;
```

```
sOutputR[iX] = iY >> 16;
}
```

3.3 Kommunikation DSP – PC

3.3.1 Vorbereitungen am DSP

Die Kommunikation zwischen PC und DSP wird vom PC aus gesteuert – der PC holt vom Speicher des DSPs über ein Host Port Interface (HPI) die Daten ab und legt dort auch die notwendigen Parameter ab. Für den DSP ist dieser Vorgang (bis auf einige Verzögerungen) völlig transparent - der DSP greift wie gewohnt auf einen Speicherbereich zu und behandelt die Daten wie normale Variablen. Auf dem PC erfolgt dieser Zugriff über eine von Texas Instruments zur Verfügung gestellte Bibliothek namens `dsk6211.dll`.
Als Beispiel wird das Programm `dsk6xldr.exe` mit den Quellcodes mitgeliefert, dort wird der Umgang mit den jeweiligen Funktionen gezeigt.

Da sowohl der DSP wie auch der PC auf denselben Speicherbereich zugreifen, sollte dieser besonders gekennzeichnet werden. Das wird erreicht, indem man in der Datei `command.6lx` einen gesonderten Speicherbereich angibt:

```
MEMORY
{
vecs: o = 00000000h l = 180h
I_HPI_MEM: o = 00000180h l = 00000080h
IRAM: o = 00000200h l = 00010000h
CE0: o = 80000000h l = 01000000h
}
SECTIONS
{
"vectors" > vecs
HPI_Section > I_HPI_MEM
.cinit > IRAM
.text > IRAM
.stack > IRAM
.bss > IRAM
.const > IRAM
.data > IRAM
.far > IRAM
.switch > IRAM
.sysmem > IRAM
.tables > IRAM
.cio > IRAM
}
```

Im Bereich `HPI_Section` sollten nun die gemeinsamen Variablen abgelegt werden. Dazu folgende Deklaration in der Datei `hpi.h`:

```
#pragma DATA_SECTION(uiHPIBuffer, "HPI_Section")
#define HPI_BUFFER_LEN 32
uint uiHPIBuffer[HPI_BUFFER_LEN];
```

Nun kann der PC auf den `uiHPIBuffer` zugreifen, wenn der Speicherbereich `0x180` bis `0x1FF` augelesen bzw. beschrieben wird.

Folgende Parameter werden vom DSP ausgewertet:

Parameter	***Index***	***Bedeutung***
`HPI_Phi1`	`(uiHPIBuffer[0])`	φ_1 , Winkel der ersten Quelle
`HPI_Phi2`	`(uiHPIBuffer[1])`	φ_2 , Winkel der zweiten Quelle
`HPI_DeltaD`	`(uiHPIBuffer[2])`	δ_D , Erwünschte Stellung des Kopfes
`HPI_Control`	`(uiHPIBuffer[3])`	Kontrollwort für Programmablauf
`HPI_OutputGain`	`(uiHPIBuffer[4])`	Verstärkungsfaktor für den Ausgang
`HPI_MaxInput`	`(uiHPIBuffer[5])`	Maximum des Signals am Eingang
`HPI_MaxOutput`	`(uiHPIBuffer[6])`	Maximum des Signals am Ausgang (vor dem Verstärkungsfaktor)
`HPI_MaxW`	`(uiHPIBuffer[7])`	Maximum des W-Signals
`HPI_MaxX`	`(uiHPIBuffer[8])`	Maximum des X-Signals
`HPI_MaxY`	`(uiHPIBuffer[9])`	Maximum des Y-Signals
`HPI_MaxU`	`(uiHPIBuffer[10])`	Maximum des U-Signals

Parameter	*Index*	*Bedeutung*
HPI_MaxV	(uiHPIBuffer[11])	Maximum des V-Signals
HPI_MaxX3	(uiHPIBuffer[12])	Maximum des X3-Signals
HPI_MaxY3	(uiHPIBuffer[13])	Maximum des Y3-Signals
HPI_CPULoad	(uiHPIBuffer[16])	Die momentane Auslastung des DSPs
HPI_ISRCount	(uiHPIBuffer[17])	Anzahl der berechneten Frames
HPI_DIPSwitches	(uiHPIBuffer[18])	Die momentane Stellung der DIP-Schalter
HPI_DeltaN	(uiHPIBuffer[19])	δ_N , die Stellung des Kopfes, momentan

Da bei den ersten Testläufen bei schnellen Kopfbewegungen Artefakte im Audiosignal festgestellt wurden (sehr hohe Winkelmodifikationsrate), wird für die Berechnung der Audiodaten ein gefilteter Wert δ_N des vom Headtracker übergebenen Winkels δ_D verwendet.

$$\delta_N = \delta_N + 0.5(\delta_D - \delta_N)$$

Da es sich dabei um ein zirkuläres Problem handelt, muß die Berechnung den Übergang von 0° bis 360° zusätzlich berücksichtigen.

3.3.2 Anbindung an den PC

Das DSK muß am PC angeschlossen und betriebsbereit sein. Die Verbindung wird folgendermaßen aufgebaut:

1. Verbindung zum DSK öffnen: `dsk6x_open()`

2. DSP Reset: `dsk6x_reset_dsp()`
3. HPI-Verbindung aufbauen: `dsk6x_hpi_open()`
4. Binäre Datei mit dem DSP-Programm laden: `dsk6x_coff_load()`
5. Programm starten: `dsk6x_hpi_generate_int()`
6. HPI-Verbindung trennen: `dsk6x_hpi_close()`
7. - Daten vom DSP lesen: dsk6x_hpi_read() und
   - Daten in den DSP schreiben: `dsk6x_hpi_write()`
8. Beim Programmende die Verbindung zum DSK schließen: `dsk6x_close()`

   All diese Funktionen bietet die Bibliothek `dsk6211.dll`.

3.3.3 Zusätzliche Bibliothek: dsk6xVB.dll

Leider hat es sich herausgestellt, daß die Parameterübergabe für die von Texas Instruments ausgelieferte Bibliothek `dsk6211.dll` nur C-kompatibel aber nicht Visual Basic-kompatibel ist. Deswegen mußte eine weitere Bibliothek, `dsk6xVB.dll`, geschrieben werden, die als Zwischenschicht die Aufrufe von VisualBasic in die Parameterübergabe von `dsk6211.dll` übersetzt. Folgende Funktionen wurden implementiert:

- Verbindung zu DSK öffnen: `DSK6xOpen()`
  Es wird direkt die Funktion `dsk6x_open()` aufgerufen.
- DSP-Programm laden: `DSK6xLoad()`
  Zusammenfassung der Funktionen `dsk6x_reset_dsp()`, `dsk6x_hpi_open()`, `dsk6x_coff_load()`, `dsk6x_hpi_generate_int()` und `dsk6x_hpi_close()`
- Speicher auslesen: `DSK6xRead()`
- Speicher beschreiben: `DSK6xWrite()`
- Verbindung zum DSK schließen: `DSK6xClose()`

3.3.4 Anbindung in Visual Basic

Zur Anbindung der Bibliotheken an Visual Basic wurde die Datei `DSK6xVB.bas` erstellt.

Sie muß in das VB-Projekt eingebunden sein, um die DSK6x-Funktionen nutzen zu können. Die Namen der Funktionen lauten wie folgt:

```
Public Declare Function dsk6x_open Lib "dsk6xvb.dll" Alias "_DSK6xOpen@8"
Public Declare Function dsk6x_load Lib "dsk6xvb.dll" Alias "_DSK6xLoad@8"
Public Declare Function dsk6x_read Lib "dsk6xvb.dll" Alias "_DSK6xRead@16"
Public Declare Function dsk6x_write Lib "dsk6xvb.dll" Alias "_DSK6xWrite@16"
Public Declare Function dsk6x_close Lib "dsk6xvb.dll" Alias "_DSK6xClose@4"
```

Weiters sollte sich die Datei `dsk6xVB.dll` im VB-Projekt-Verzeichnis befinden, damit sie zur Laufzeit eingebunden werden kann.

3.4 Host Programm am PC

Das Hostprogramm hat folgende Aufgaben:

- Abholung des Winkels der Kopfdrehung vom Headtracker und Weiterleitung an den DSP
- Laden des DSP-Programms beim Programmstart
- Parametrierung des Algorithmus
- Benutzerschnittstelle und Datenvisualisierung

3.4.1 Allgemein

Das Programm wurde in Visual Basic 6.0 realisiert und besteht aus folgenden Teilen:

Datei	***Beschreibung***
`vsps.vbp`	VisualBasic-Projekt-Datei
`vsps.exe`	Ausführbare Datei – Das Endergebnis
`vsps.ini`	Gespeicherte Einstellungen
`vsps.vbw`	wird von VB benötigt

Datei	*Beschreibung*
vsps.pdm	wird von VB benötigt
DSK6xVB.bas	Deklarationen für den Zugriff auf dsk6xvb.dll
frmAbout.frm	Fenster: „About“
frmAbout.frx	binaries
frmEinstellungen.frm	Fenster: „Einstellungen“
frmEinstellungen.frx	binaries
frmHeadTracker.frm	Fenster: „Einstellungen für den Headtracker“
frmHeadTracker.frx	binaries
frmMain.frm	Hauptfenster
frmMain.frx	binaries
frmParameter.frm	Fenster: „Parameter“
frmParameter.frx	binaries
HPI.bas	Deklarationen für den Host Port Interface
ini.bas	Routinen für den INI-Datei-Zugriff

Beim Programmstart wird das Hauptfenster `frmMain` geöffnet, von welchem alle anderen Fenster ein-/ bzw. ausgeblendet werden. Weiters befinden sich im Hauptfenster ein COM-Objekt, über das die Kommunikation mit dem Headtracker stattfindet, sowie eine Liste `lstStatus`, in der die Statusmeldungen eingetragen werden.

Beim Programmstart werden die globalen Variablen sowie die serielle Schnittstelle initialisiert. Die Verbindung zum Headtracker wird überprüft in dem der Winkel einmal abgeholt wird (wird von `SetReference()` erledigt). Sollte der Headtracker nicht antworten, wird das Programm im manuellen Modus ausgeführt, d.h. der Winkel der Kopfdrehung kann nur über einen Slider im Parameter-Fenster verändert werden.
Danach wird eine Verbindung zum DSK aufgebaut und das DSP-Programm in den Speicher des DSP geladen. Bei auftretenden Problemen bleibt das Programm nach der

Ausgabe der jeweiligen Fehlermeldungen stehen und sollte nach der Fehlerbehebung noch einmal gestartet werden.

3.4.2 Kommunikation mit dem Headtracker

Das VB-Programm kommuniziert mit dem Headtracker über das COM-Objekt namens `COM`, das sich in `frmMain` befindet. Der Timer `tmrHT` sorgt dafür, daß der Winkel vom Headtracker periodisch abgeholt wird – es wird am Anfang der Befehl `POINT` zum Headtracker geschickt (siehe [FLOC]), der in späterer Folge die 3D Position und Winkelstellung des am Kopfhörer montierten Empfängers liefert. Es wird nur der Azimuthwinkel ausgewertet. Diese Abfrage findet ca. 50 mal in der Sekunde statt.

Aus dem Azimuthwinkel wird durch die Subtraktion eines Referenzwertes `lDeltaReference` ein relativer Winkel `lDeltaHT` berechnet – dieser Wert wird zum DSP gesendet.

3.4.3 Kommunikation mit dem DSP

Nach dem die Funktion `LoadCOFF` beim Programmstart erfolgreich war (`DSP_Ready` wurde auf `true` gesetzt), kann die Kommunikation über das Menü Program/Start gestartet werden. Es wird die Form `frmParameter` eingeblendet und der Timer `tmrDSP` gestartet (`DSP_Running` auf `true` gesetzt). Dieser Timer startet die periodische Datenübertragung zum DSP. In der Funktion `tmrDSP_Timer()` werden alle Parameter zum DSP und alle Meßwerte vom DSP übertragen – die dazu notwendigen Variablen werden in `INI.bas` deklariert.

In der Form `frmParameter` werden die globalen Werte dargestellt – dort findet allerdings keine Kommunikation statt. Die `frmParameter` ist sozusagen ein Client der `frmMain` als Server.

Die Kommunikation mit dem DSP wird beendet durch das Setzen der Variable `DSP_Running` auf `False`. Das kann von der Form `frmParameter` (Form schließen oder Click auf „Halt program“), von der Form `frmMain` (Schließen der Form, Menü Program/Halt auswählen) wie auch vom System selbst (Aufruf von `Form_QueryUnload()`) getriggert

werden. In der Funktion `tmrDSP_Timer()` wird dann der DSP gemuted und die Kommunikation eingestellt.

4 Bedienungsanleitung

4.1 Hardwareanforderungen

4.1.1 DSP/Audio Codec

- **DSP Starter Kit TMS320C6711 von Texas Instrument** , European Version mit der Bestellnummer TMDS320006711E.
  Konfiguration:

Jumper	*Einstellung*
JP1	verbunden

- **AUDIO-CODEC:** Stereo-Audio-Codec-**Evaluationsboard TLV320AIC27 EVM** mit der Bestellnummer TLV320AIC27 bestellt.
  Konfiguration:

Jumper	*Einstellung*
J21	offen
J7	2-3 verbunden
J43	offen
J44	verbunden
J17	offen

Jumper	*Einstellung*
J18	verbunden
J31	verbunden
JP14	verbunden
JP23	verbunden
J37	verbunden
J38-9 mit J33-unten	verbunden
J40	verbunden
J41	verbunden

4.1.2 Anforderungen an den PC

- eine parallele Schnittstelle auf EPP konfiguriert
- eine serielle Schnittstelle (9600 baud)
- mind. 64MB Arbeitsspeicher
- Intel P3 350MHz bzw. AMD K6-II 350MHz oder schneller
- Betriebssystem: Microsoft Windows 32-API kompatibel (Windows 95, 98, NT4.0, 2000...)
- Bildschirm Auflösung von mind 800x600

4.1.3 Headtracker

Flock of Birds von der Firma Ascension Technology, siehe [FLOC]

Konfiguration:

DipSwitch	1	2	3	4	5	6	7	8
Stellung	OFF	ON	ON	OFF	OFF	OFF	OFF	OFF

4.1.4 Hardware Setup

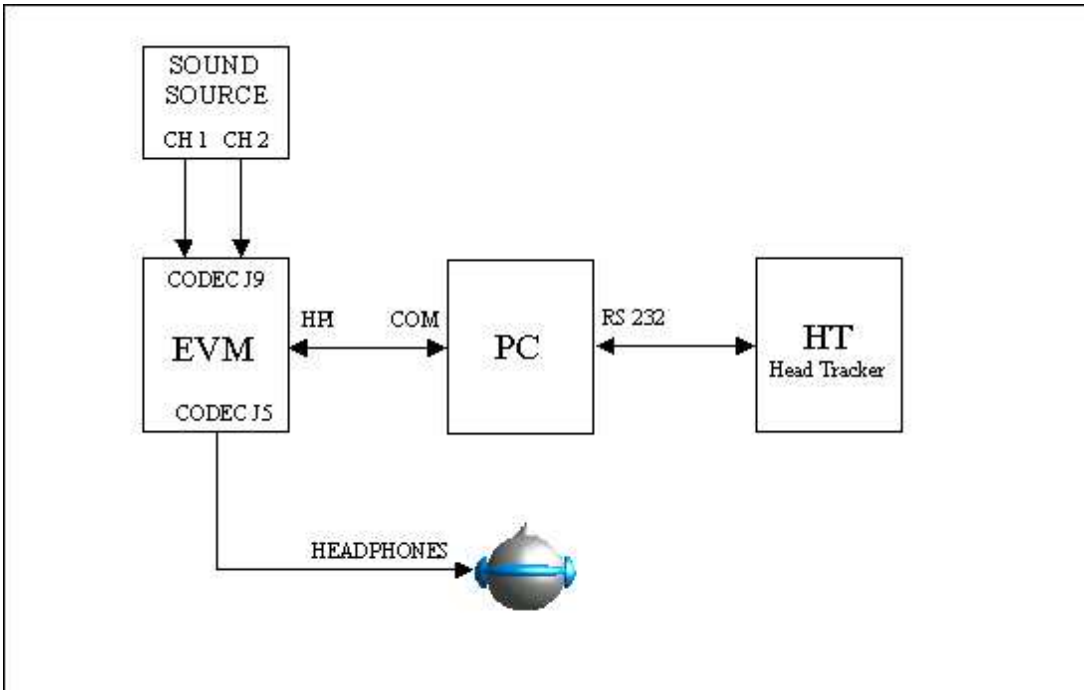


Abbildung 4.1: Hardware Setup

4.2 Installation der Software

- Voraussetzung ist die Installation des CCS Developement Studios von Texas Instruments, welches die benötigten DLLs zur Verfügung stellt
- Führen Sie die Datei `setup.exe` aus, welche im Paket Virtual Sound Positioning System enthalten ist.

4.3 Bedienung des Programms

Dieses Kapitel gibt einen kurzen Überblick über die Funktionen und Bedienelemente des Programms Virtual Sound Positioning System.

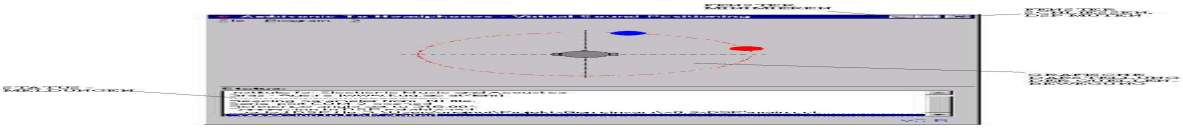
4.3.1 Starten von Virtual Sound Positioning System

- Starten der Datei `vsps.exe`
- Folgende Prozeduren werden während des Starts ausgeführt:
  - Lesen der Parameter aus der INI-Datei
  - Öffnen der in der INI-Datei definierten Seriellen Schnittstelle
  - Überprüfen der Kommunikation mit dem Headtracker (HT)
  - Setzen der aktuellen HT Position als Referenzwinkel
  - Öffnen der Verbindung zum DSP-Board
  - Laden der Datei `main.out` in den Speicher des DSP
  - Verbindung zum DSP öffnen, Kommunikation starten
- Beschreibung der Fehlermeldungen siehe 4.3.3

4.3.2 Virtual Sound Positioning System Desktop

Nach dem Starten des Programms erscheinen das Haupt- und das Parameterfenster mit allen Anzeige- und Bedienelementen, die Simulation wird automatisch gestartet.

Hauptfenster:



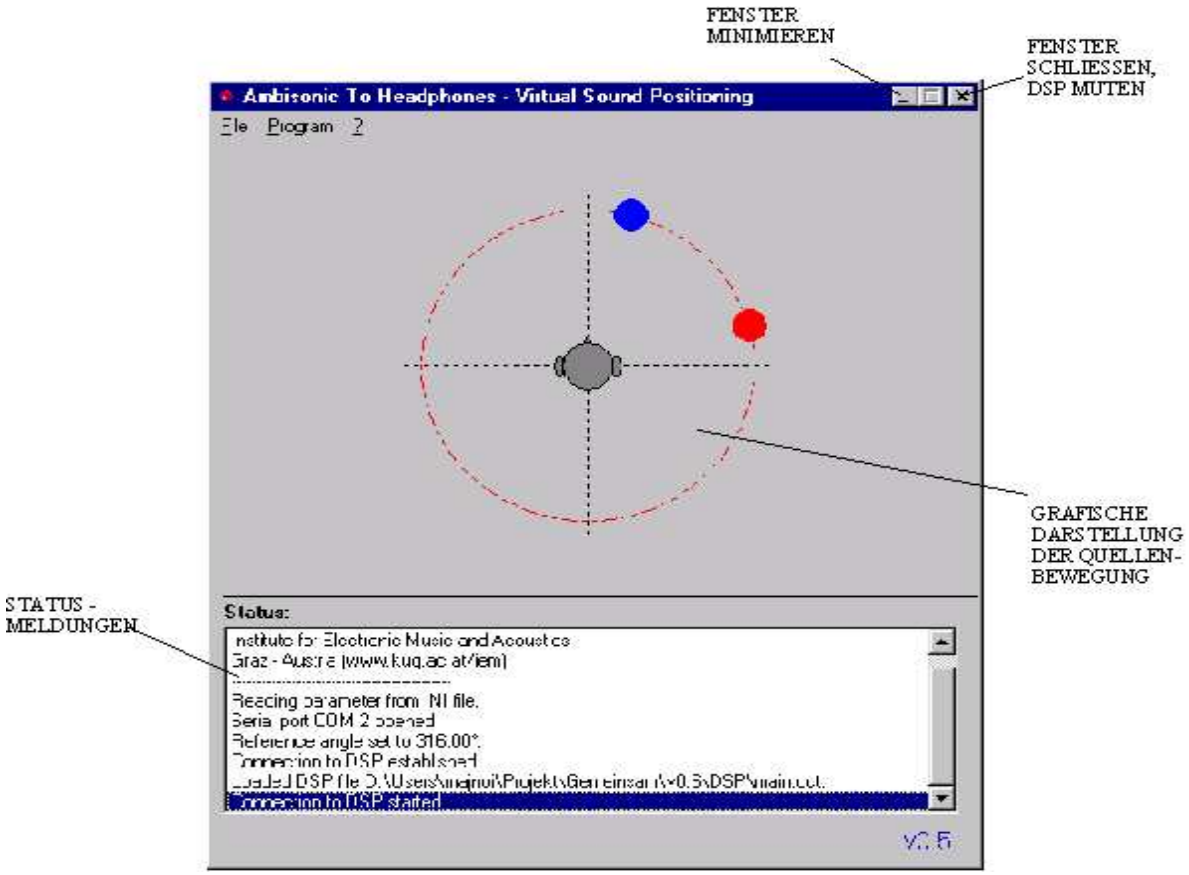


Abbildung 4.2: Hauptfenster

Das Hauptfenster umfaßt folgende Anzeigen:

- Grafische Simulation der virtuellen Quellenbewegung:
  Im HT-Modus wird die relative Quellenbewegung, nicht die Kopfdrehung dargestellt.
  Im Manuellen-Modus wird die Quellenbewegung direkt dargestellt.

- Statusmeldungen:
  Aktuelle Programmereignisse werden angezeigt.

Hauptfenster Menüpunkte:

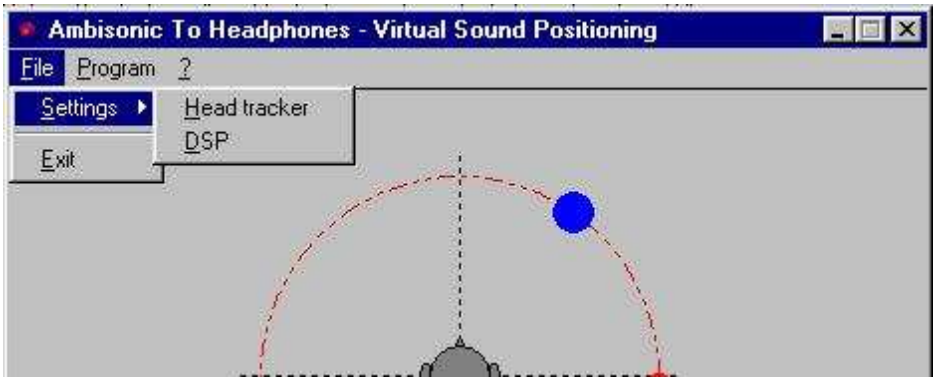


Abbildung 4.3: Menüpunkte Hauptfenster

Folgende Menüpunkte können nur bei angehaltener Simulation gewählt werden:

- File/Settings/HeadTracker:
  Hier kann die serielle Schnittstelle an der der Headtracker angeschlossen ist ausgewählt werden.

- File/Settings/DSP:
  Menü zum Laden einer COFF-Datei (`*.out`)in den Speicher des DSP.

- Program/Start:
  Bewegungssimulation starten (Parametermenü wird geöffnet)

Folgende Menüpunkte können auch während der Bewegungssimulation ausgeführt werden:

- Program/Halt:
  Stoppen der Bewegungssimulation, Freigabe obenstehender Menüpunkte.

- File/Exit:
  Programm beenden.

Parameterfenster:

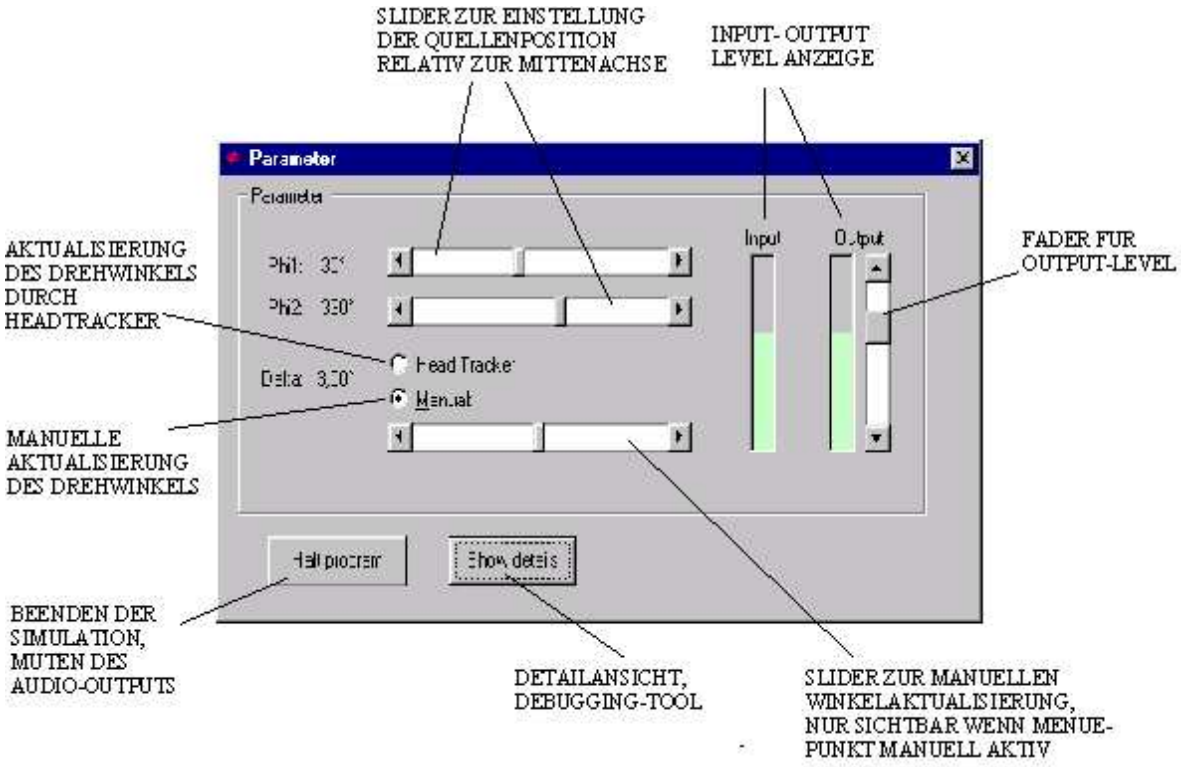


Abbildung 4.4: Parameterfenster

Das Parameterfenster umfaßt folgende Anzeige- und Bedienelemente:

- Quellenposition:
  Einstellung der jeweiligen Quellenposition relativ zur Hauptachse.

- Hauptachsenposition:
  Wird nur sichtbar, wenn der Menüpunkt 'Manual' gewählt ist.
  Einstellung des Drehwinkels der Hauptachse.

- Input- / Output-Level:
  Anzeige des jeweiligen Audiolevels.

- Output – Fader:
  Fader zur Einstellung des Output-Audiolevels.

- Halt program:
  Die Simulation wird beendet, der Menüpunkt 'File / Settings' im Hauptfenster wird freigegeben. Der DSP wird NICHT angehalten, lediglich der Audio-Ausgang gemutet.

- Set as Reference:

  Dieser Menüpunkt erscheint nur, wenn die Auswahl 'HeadTracker' aktiviert ist.
  Der Winkel der aktuellen Kopfposition wird als Referenzwinkel gesetzt.

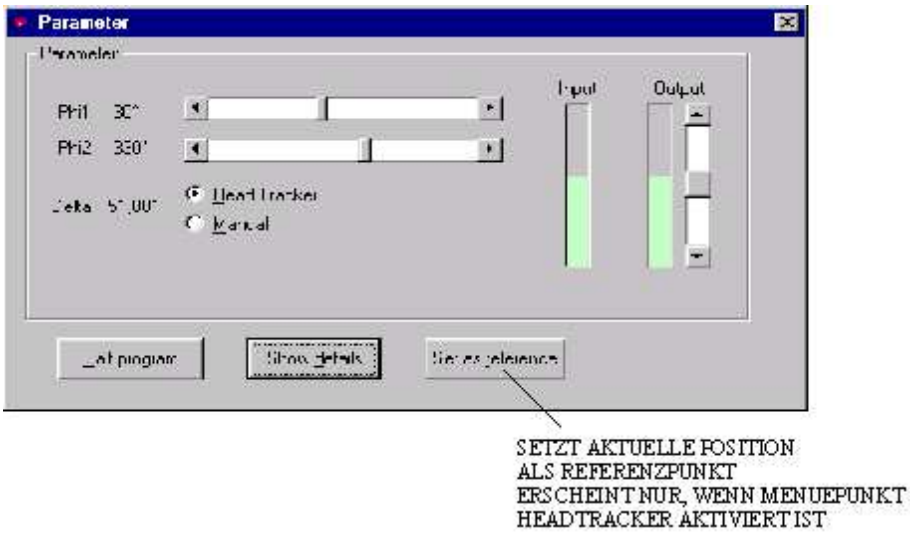
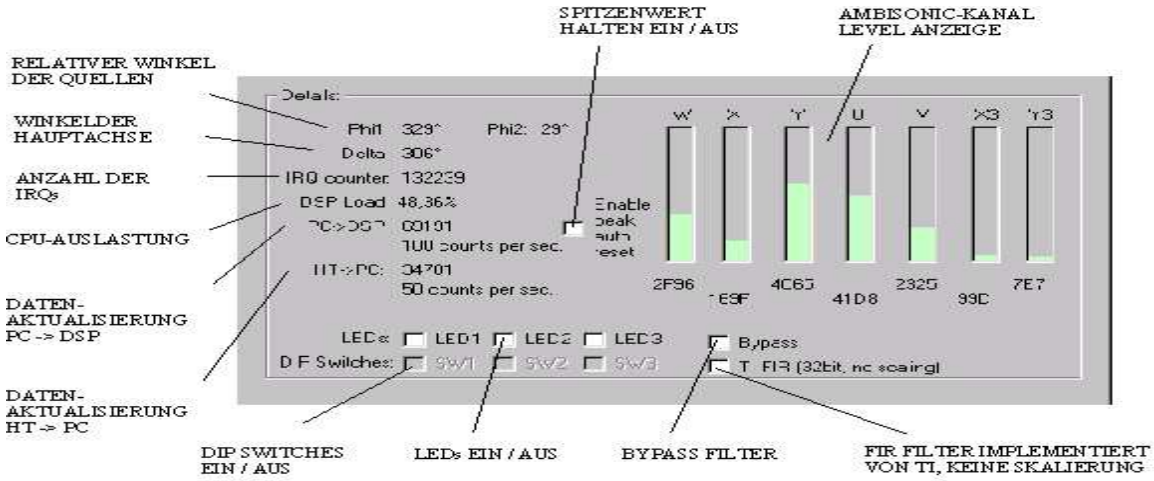


Abbildung 4.5: Referenzpunkt setzen

- Show / Hide Details:
  Zeigt / versteckt die Detailansicht zum Debuggen.



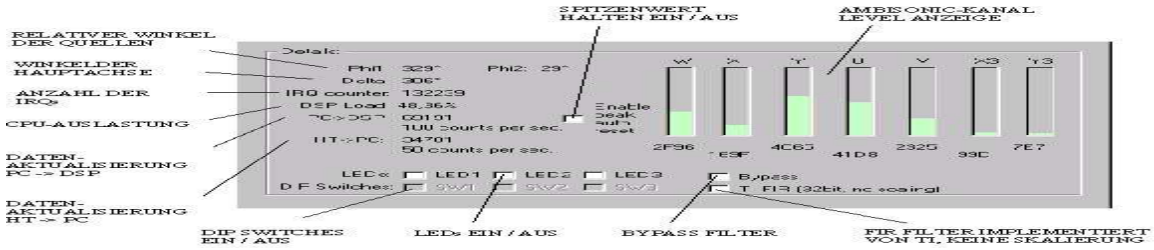


Abbildung 4.6: Detailansicht

- Phi 1, Phi 2, Delta:
  Aktueller Winkel der Quelle1, Quelle 2 relativ zur Hauptachse, sowie Winkel der Kopfdrehung.

- IRQ Counter:
  Zähler für die Anzahl der durchgeführten Interrupts.

- DSP Load:
  Momentane CPU Auslastung.

- PC -> DSP:
  Datenaktualisierungsrate der Kommunikation des PC mit dem DSP.

- HT -> PC
  Datenaktualisierungsrate der Kommunikation des HT mit dem PC.

- DIP Switches:
  Abfrage der Stellung der DIP-Schalter des Evaluationboards.

- LEDs:
  Die LEDs des Evaluationboards lassen sich setzen sowie rücksetzen.

- Bypass:
  Das Audiosignal wird ohne Filterung an den Ausgang weitergeleitet.

- TI FIR:

Die Filterung findet mit den der TI-Library entnommenen FIR-Algorithmen statt. Bei diesen Filtern wird der Additionsakkumulator nur mit 32 bit ohne jegliche Skalierung verwendet, somit erscheint das Signal lauter, es kann jedoch zu einer Übersteuerung der einzelnen Ambisonic-Kanäle kommen.

- Enable peak auto reset:
  Ist dieser Menüpunkt gewählt, werden die Spitzenwerte der jeweiligen Pegelanzeigen nicht eingefroren.

- Pegelanzeige:
  Die Pegel der einzelnen Ambisonic-Kanäle werden angezeigt.

Literaturverzeichnis

[BAMF]: J.S.Bamford, An Analysis of Ambisonic Sound System of First and Second Order; University of Waterloo, Waterloo, Ontario, Canada, Thesis; 1995

[SONT]: Alois Sontacchi, Dekodierung der Ambisonic Signale für binaurale Wiedergabe; Institut für Elektronische Musik und Akustik, Graz, IEM Report; 2001

[LEIT]: Stephan Leitner, Aufnahme- und Wiedergabesystem zur Berechnung eines dynamisch modifizierbaren binauralen Signalpaares; Institut für Elektronische Musik und Akustik, Graz, Diplomarbeit; 2000

[TEXI]: Texas Instruments Inc., Homepage; www.ti.com

[EVMU]: Texas Instrument Inc., TLV320AIC27 EVM User's Guide; slau051.pdf

[AC97]: Intel, Specification Audio Codec '97; ftp://download.intel.com/ial/scalableplatforms/ac97r22.pdf

[EVMD]: Texas Instruments Inc., TLV320AIC27 Data Sheet; slas253a.pdf

[DSCH]: Teaxas Instruments Inc., TMS320C6711 DSP Starter Kit, Schematics; 6711dsk_schematics.pdf

[PERI]: Texas Instruments Inc., TMS320C6000 Peripherals, Reference Guide; spru190c.pdf

[PROG]: Texas Instruments Inc., TMS320C6000 Programmer's Guide; spru198d.pdf

[FLOC]: Ascension Technology Corporation, The Flock Of Birds - Installation and Operation Guide;

ftp://ftp.ascension-tech.com/pub/ascension-tech.com/MANUALS/FlockOfBirds.pdf

---

1LOAD: Ladeoperation eines Operanden

2MPY: Multiplikation zweier Zahlen

3ADD: Addition zweier Zahlen

4Bei einer Sampling-Frequenz von 48kHz

5Bei 48kHz und Berechnung alle 128 Samples werden 48000/128 Frames pro Sekunde berechnet

---

Zum *Anfang*
Zu *Forschung und Entwicklung*