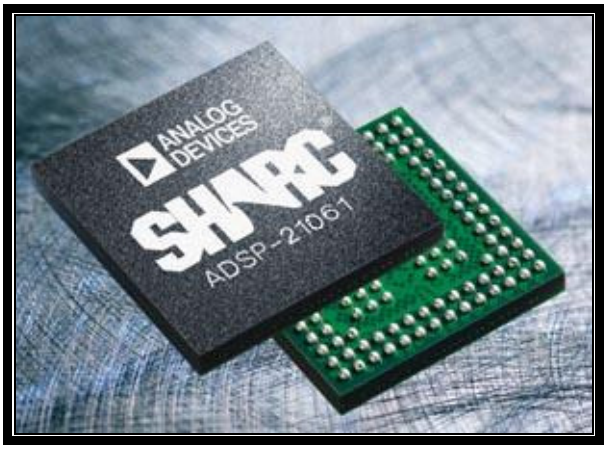
Dokumentation zum Komplexpraktikum

Realisierung von Audio-Modulen für das ADSP21061 EZ-Kit lite



Bearbeitet von: Stefan Müller Matrikel 936717
Thomas Pilgrim Matrikel 936720

Zeitraum: Oktober 2005 – Januar 2006

Betreuer: Prof. Dr. rer. nat. habil. H. Kahnt
Prof. Dipl.-Ing. H. Wagner

FACH HOCH SCHULE JENA
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Jena
Fachbereich Elektrotechnik / Informationstechnik
Studiengang „Kommunikations- und Medientechnik“

Inhaltsverzeichnis

1. Einleitung

Die digitale Signalverarbeitung nimmt in der Audiotechnik eine immer größere Rolle ein. Im Mittelpunkt stehen hierbei Digitale Signalprozessoren, um Signalverarbeitungsalgorithmen auf digitalisierte Audiosignale anzuwenden.

Aufgabe im Rahmen des Komplexpraktikums war die Auseinandersetzung mit dem digitalen Signalprozessor *SHARC ADSP – 21061* der Firma *Analog Devices*, die Realisierung verschiedener Module (Mittelwertberechnung, Audiofilterberechnung etc.) und die Dokumentation der gesammelten Erkenntnisse und Schwierigkeiten. Hierzu stand die Entwicklungsplatine (englisch: evaluation board) *EZ kit lite* des Prozessors zur Verfügung. Der SHARC Prozessor ist für dieses Thema prädestiniert, da er durch seine Struktur (*SHARC = Super Harvard Architektur*) schnell Daten einlesen, verarbeiten und ausgeben kann und somit ideal für die digitale Signalbearbeitung (speziell Audiosignalbearbeitung) geeignet ist.

Wichtige Meilensteine dieses Themas waren die Inbetriebnahme des Boardes, die Konfiguration des Audio Codec zum Empfang von Daten, die Programmierung der Module in „C“ sowie die Kommunikation des Evaluationsboardes via RS232 nach erfolgreichem Laden und Abspielen der Module. In den folgenden Abschnitten werden die einzelnen Etappen ausführlich erläutert und auf die möglicherweise entstehenden Probleme hingewiesen.

Diese Anleitung soll eine Hilfestellung sein, um schneller die grundlegende Arbeitsweise dieser Platine zu erfahren. Für eventuelle weiterführende Projekte sollten ebenfalls die Betriebsanleitungen des *SHARC ADSP 21061* und der Entwicklungsplatine mitbenutzt werden.

Hinweis:
Dieser Dokumentation liegt eine CD bei, auf der sich neben den Datenblättern der verschiedenen Komponenten des Entwicklungsboardes unter anderem das Demonstrationsprogramm inklusive aller Quelltexte und verschiedene MatLab – Dateien zur Filterberechnung befinden. Alle in dieser Dokumentation verwendeten Dateien und die betreffenden Pfadangaben beziehen sich auf diese beiligende CD.

2. Installation und Inbetriebnahme der Entwicklungsplatine

In diesem Punkt wird erklärt, wie man zu einer fehlerfreien Kommunikation zwischen PC und Prozessor gelangt, welche Software man nutzen sollte und wie man eigens geschriebene Programme im Prozessor zum Starten bekommt.

2.1 Vorraussetzungen

Um erfolgreich mit der Entwicklungsplatine arbeiten benötigt man:

- eine Stromversorgung (Ausgang des Netzteils: 8VDC und 1.5A),
- PC und korrekte Software (siehe Punkt 2.3)
- ein Kabel zur seriellen Kommunikation via RS232 zum PC (SUB D9 Stecker),
- eventuell diverse Stereoaudiokabel mit 3,5 mm Klinkenstecker, um Eingangssignale zum bzw. Ausgangssignale von der Platine mit entsprechender Peripherie zu verbinden (zum Testen oder Ausführen von Programmen)

2.2 Anschlüsse und Peripherie der Entwicklungsplatine des ADSP 21061

In Abb. 1 sind die wichtigsten Anschlüsse (wie bereits unter 2.1 erwähnt) und für uns interessanten Peripheriebausteine schematisch dargestellt.

- U2 → Digitaler Signalprozessor *(SHARC ADSP 21061)*
- U4 → UART Baustein PC16550D, wichtig für die Kommunikation über RS232 (siehe Abschnitt 6)
- U9 → Audio CODEC Baustein AD1847, wichtig für das Abtasten und Einlesen des Audioeingangssignal über den SPORT 0 des ADSP (siehe Abschnitt 4.1)
- U7 → EPROM Baustein ST9819F, beinhaltet das geladene Programm, welches der DSP über externe Speicherverwaltung in den Programmspeicher lädt.
- S1 & S2 → Taster, welche im DSP einen externen Interrupt auslösen (können)
- S3 → RESET Taster (Master Reset)
- D1 → POWER LED, leuchtet, wenn VDC angelegt wurde
- D2 – D4, D6 → FLAG LED, leuchten, wenn ein Interrupt ausgelöst wurde

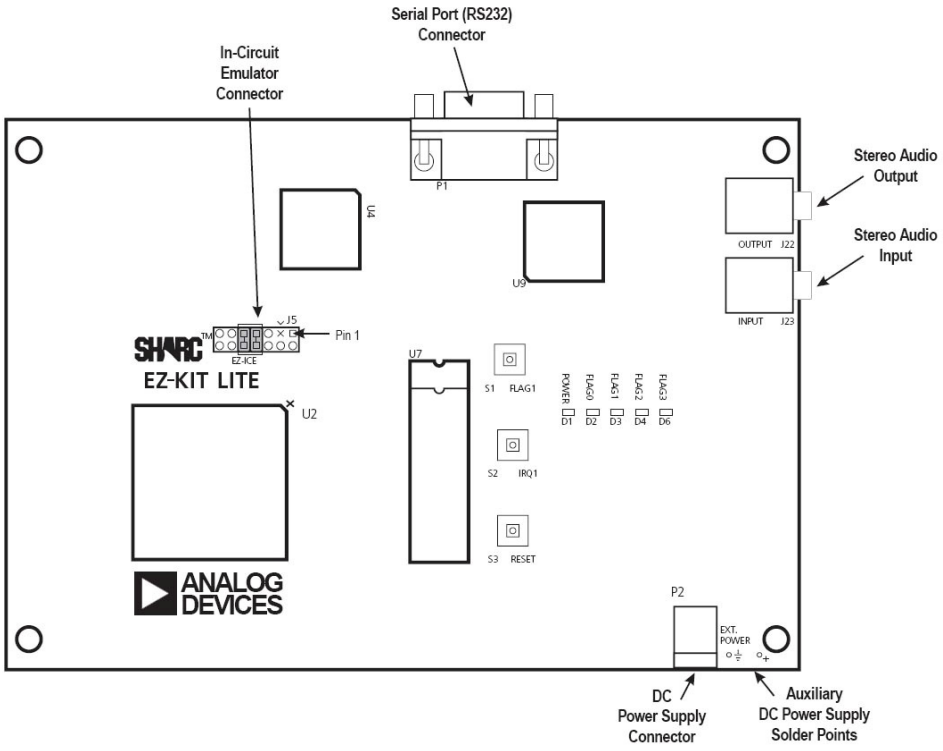


Abbildung 1 – schematischer Aufbau der ADSP 21061 Entwicklungsplatine

D2 - D4 & D6 können nützliche Programmierhilfen sein, man kann sie an kritischen Programmpunkten als Indikatoren verwenden (z.B. Schleife erfolgreich durchlaufen, Abtastwert korrekt übernommen etc.).

Bei Betätigen von S3 erfährt der DSP einen Master Reset und lädt das Bootprogramm aus dem EPROM (U7).

2.3 Inbetriebnahme der Entwicklungsplatine – der erste Sound

Zum ersten Funktionstest der Platine bedarf es zweier Schritte:

- Anschließen von Kopfhörern oder Lautsprechern an die Klinkenbuchse J22
- Anschließen des Spannungsversorgungsgerätes an die DC POWER Buchse

Bei einem funktionsfähigen Board leuchtet D1 ständig, D4 & D6 blinken und es wird nun eine Melodie hörbar. Es wird keine Melodie abgespielt, wenn sich nicht das Original EPROM im Sockel U7 befindet oder keine korrekte Versorgungsspannung anliegt (D1 leuchtet dann nicht).

Im ausgelieferten Zustand enthält das Bootprogramm des EPROMs die Konfiguration der UART Schnittstelle, die Konfiguration des SPORT0 (ein schneller serieller PORT des DSP),

die Konfiguration des *AD1847* (Audio CODEC), sowie eine Routine, welche die Melodie erzeugt. Für eine genauere Betrachtung sollte das Handbuch (reference manual) unter dem Punkt 2 studiert werden.

2.4 Installation der richtigen Version von Visual DSP

Um das im Rahmen dieses Komplexpraktikums entstandene Programm „DSP Audio Demo“ oder eigene Programme zu starten, ist es stets von Nöten, diese in den externen Speicher (EPROM) des DSP zu laden. Das geschieht mit einem „Loader“ Programm. Den elegantesten Weg bietet hier die Entwicklungsumgebung *VisualDSP++*. Da es von Seiten der Firma *Analog Devices* einige Kompatibilitätsprobleme bezüglich dieser Software und dem verwendeten Evaluation Board gibt, werden an dieser Stelle einige nützliche Hinweise gegeben.

Es sollte unbedingt die *VisualDSP++ 3.0* - Version installiert werden. Höhere Versionen unterstützen die Entwicklungsplatine nicht. Mit Doppelklick auf **SHARC3.0.exe** wird die Installation gestartet (Abbildung 2 erscheint).

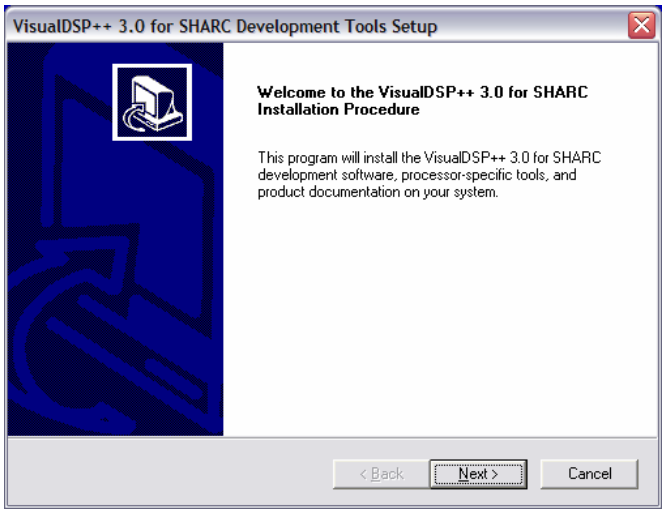


Abbildung 2 – Installation von VisualDSP++ 3.0

Nach diversem Klicken (Lizenz-Zustimmung etc.) muss später die richtige Entwicklungsplatine ausgewählt werden (Abbildung 3). Gemäß der Aufforderung, muss der Computer nun neu gestartet werden.

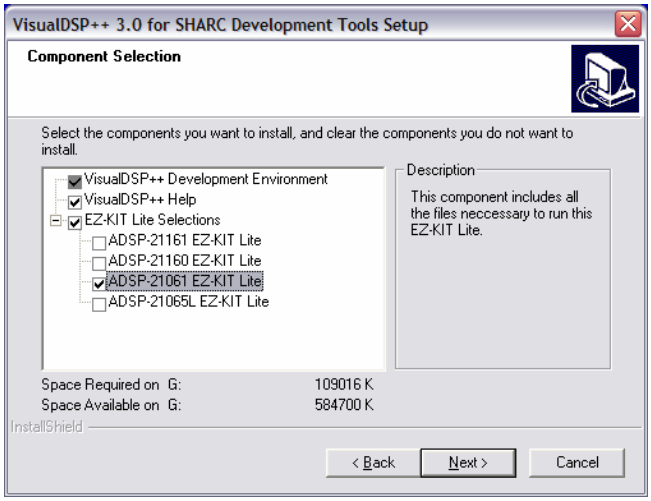


Abbildung 3 – Auswahl der richtigen Entwicklungsplatine

Nach dem Neustart des Rechners muss eine Datei ausgetauscht werden, um spätere Probleme bei der Unterstützung des Boardes durch *VisualDSP++* zu vermeiden. Hierzu befindet sich auf der CD die Datei **ArchDef.xml**, mit der die bestehende Version dieser Datei im Installationsordner (**C:\Programme\Analog Devices\VisualDSP\System\ArchDef**) ersetzt werden muss.

Nach Starten des Programms (**idde.exe** im Installationsverzeichnis/System) muss man sich entscheiden, ob man mit dem Simulator des Programms oder der Platine arbeiten möchte (Abb. 4). Bei Wahl des *EZ-kit Lite* (so die Bezeichnung der Platine in *VisualDSP++*) durch „Activate“ muss sichergestellt sein, dass das Board an den PC angeschlossen (über das RS232 Kabel mit SUB D9 Stecker) ist und mit Strom versorgt wird.

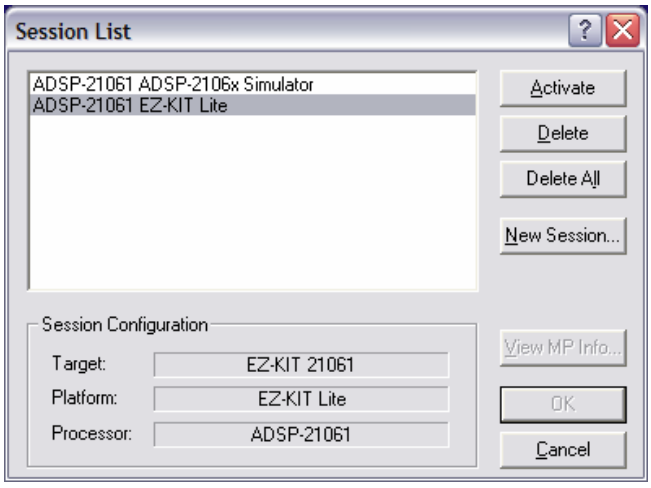


Abbildung 4 – Auswahl zwischen Simulator und Board

Nach der Aufforderung „**Reset Target**“ ist Taster S3 zu betätigen und anschließend mit „OK“ zu quittieren. Danach öffnet sich die Arbeitsoberfläche von *VisualDSP++*. Im Output-Fenster (unteres Querfenster) erscheint im Reiter „Console“ die Statusmeldung „Communication Success!“.

Falls folgende Fehlermeldung (Abb. 5) beim Start erscheint, wurde die Datei **ArchDef.xml** fehlerhaft kopiert. Schließen Sie das Programm und kopieren sie die Datei erneut.

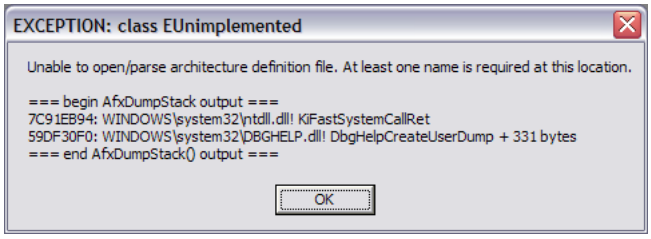


Abbildung 5 – Fehlermeldung bei inkorrektem Ersetzen der ArchDef.xml

Zur Kontrolle könnte man nun beispielsweise einen Kommunikationstest durchführen, indem man im Menüpunkt „Settings“ – „Test Communications“ anwählt. Im Output Fenster erscheint ein weiteres „Communications Success!”. Hierbei wird nur die Verbindung über RS232 überprüft und angesagt, dass die richtige Platine mit den richtigen Treibern installiert ist. Diese Verbindung ist im Debug - Modus von großer Bedeutung, da in diesem Fall sämtliche Registerstände an das Programm übermittelt werden.

Falls das Menü „Settings“ nicht existiert, wurde entweder eine falsche Session ausgewählt oder das Programm fehlerhaft installiert. Im ersten Fall kann man im Menüpunkt „Session“ → „Select Session“ die entsprechende Auswahl („ADSP 21061 EZ-KIT lite“) treffen. Möglicherweise muss die Entwicklungsplatine erneut über den Taster S3 in den RESET gebracht werden.

2.5 Andere mögliche oder nötige Software (EZ Lite vs. Visual DSP)

Wenn bis zum Punkt 2.4 alles reibungslos verlief, ist man nun in der Lage über *VisualDSP++* eigene oder fertige Programme zu schreiben, auf den DSP zu laden und zu starten. Natürlich gibt es zum Laden von bereits kompilierten und gelinkten Programmen auch andere Möglichkeiten. Eine davon ist das für die Platine vorgesehene *SHARC EZ-KIT Lite*, welches standardmäßig zum Evaluationsboard mitgeliefert wird. Der Vorteil dieses Programms ist das kennen lernen des DSP mit bereits vorgefertigten Demo - Programmen.

Hierzu zählen:

- Peter Gunn Demo
- Blink Demo
- Talkthrough Demo
- FFT Demo

Zu empfehlen ist an dieser Stelle das Handbuch „*SHARC ADSP 21061 – Reference Manual* – Chapter 3: EZ-KIT Lite“, wo detailliert beschrieben wird, wie man die einzelnen Demo Programme zu bedienen und interpretieren hat. Aufgrund der grafischen Oberfläche werden die Programme sehr leicht bedienbar und sind sehr anschaulich.
Leider ist das Programm darüber hinaus sehr unkomfortabel. Wenn man sich beispielsweise entschließt, selbst Programme zu schreiben oder außerhalb dieser Umgebung entstandene Programme auf den DSP zu laden, stößt man schnell an die Grenzen von *SHARC EZ–KIT lite*. Für die weitergehende Arbeit und Betrachtung des DSP ist deshalb von diesem Programm abzuraten. Wurde das Programm installiert, wurde auch die **autoexec.bat** umgeschrieben. Für eine spätere Arbeit mit *VisualDSP++*, sollte das Programm deinstalliert und die Einträge in der **autoexec.bat** gelöscht werden. (Start → Ausführen → SYSEDIT eingeben → mit OK bestätigen: Alle Einträge, das EZ-KIT – Lite betreffend sind zu entfernen).

2.6 Laden von Programmen mit Visual DSP

Wie unter Punkt 2.5 bereits erwähnt, sind bei der Installation von *SHARC EZ KIT Lite* einige Demoprogramme enthalten. Das gleiche gilt für *VisualDSP++*. Letztere dienen jedoch in erster Linie als Programmieranregungen und sollten vorrangig mit dem Software Simulator bearbeitet werden. Als Beispiel soll an dieser Stelle ein selbst geschriebenes Tiefpass – Programm dienen: Dieses befindet sich auf der beigefügten CD im Ordner **Programme\Butterworth**. Hierzu muss zunächst das Programm geöffnet werden. Da Programme in *VisualDSP++* in der Regel aus mehreren Teilen bestehen, wurde es als „Projekt“ abgespeichert. Hierdurch sind alle dazugehörigen Dateien vereint und alle Einstellungen bezüglich des Projektes gespeichert. Zum Öffnen eines Projektes wählt man im geöffneten *VisualDSP++* (und angeschlossener Entwicklungsplatine) im Menüpunkt „Project“ → „Open“ und wählt den Ordner „Butterworth“. Ein Projekt unter *VisualDSP++* hat die Endung **.**dpj, es muss also die Datei **butterworth_lowpass.dpj** ausgewählt werden. Im Project Fenster erscheinen nun die dem Projekt zugehörigen Dateien. Mit F7 (oder über das Menü „Project“ → „Build Pro-

ject“) wird der C - Quellcode kompiliert und danach mit der bereits existierenden Linker-Datei verlinkt (**butterworth.ldf**). Wird im Output Fenster der Reiter „Build“ auswählt, erfährt man den Zustand des Kompilierungsprozesses. Mit der Meldung „build successful“ bestätigt der Compiler die Richtigkeit des Quellcodes.

Wurde im Vorfeld kein anderes Projekt bearbeitet, so wird die entstehende **butterworth_lowpass.dxe** - Datei automatisch via RS232 auf die Platine übertragen und im EPROM der Platine (U7 in Abbildung 1) zwischengespeichert. Im Output Fenster erscheint die Meldung „load complete“.

Zum Start des Programms muss nun lediglich im Menü „Debug“ → „Run“ ausgewählt werden. Dies lädt das Programm vom EPROM in den Programmspeicher des DSP und startet es. Zur Laufzeitkontrolle blinkt nun die LED D2. Dies ist ein Zeichen, dass der DSP Abtastwerte des Audio Codec einlädt, sie bearbeitet und wieder ausgibt.

Schließt man nun eine Audioquelle an die Buchse J23 an (Beachten der Jumperstellung, ob Mikrofon oder Line IN Quelle) und verbindet die Buchse J22 mit Lautsprechern, hört man ein Ausgangssignal, welches ab der Frequenz 500 Hz stark bedämpft ist (Tiefpassfilter 4. Ordnung).

Zum Anhalten eines Programms wählt man im Menüpunkt „Debug“ → „Halt“, um das laufende Programm zu stoppen.

3. Arbeiten mit Visual DSP

Dieser Punkt soll als Anleitung dienen, um zu einem selbst geschriebenen Projekt zu gelangen und eigene Programme und Algorithmen auf dem DSP zu verwirklichen. Auch hier kann lediglich ein kurzer Einblick erfolgen und unterstützend zum ersten eigenen DSP Programm verholfen werden. Zum weiterführenden Programmieren bzw. Arbeiten mit *VisualDSP++* ist es unbedingt empfehlenswert, sich mit den auf der CD befindlichen Manuals zu beschäftigen. Um zu einem lauffähigen Programm auf der Entwicklungsplatine zu gelangen sind folgende Punkte zu durchlaufen:

- Erstellen eines neuen Projektes,
- Erstellen des Quellcodes (C,C++,Assembler)
- Einbinden einer LINKER Datei
- Kompilieren des Programms
- gegebenenfalls Simulieren und Debuggen des Programms
- Laden und Starten eines lauffähigen Programms

Die folgenden Abschnitte erläutern die aufgeführten Punkte schrittweise.

3.1 Erstellen eines neuen Projektes

Wie bereits im Punkt 2.5 angedeutet, wird ein Programm für *VisualDSP++* stets aus mehreren Dateien bestehen (Headerdateien, Quellcodedateien, Linkerdateien). Um diese gemeinsam zu einem lauffähigen Programm zusammenzufassen, werden sie in einem Projekt vereint.

Zum Erstellen eines neuen Projektes:

- Anwählen des Menüpunktes Project → NEW
- Name (und ggf. Speicherort) des Programms wählen mit OK bestätigen
- Einstellen der „Project Options“ wie in Abbildung 6 und mit OK bestätigen

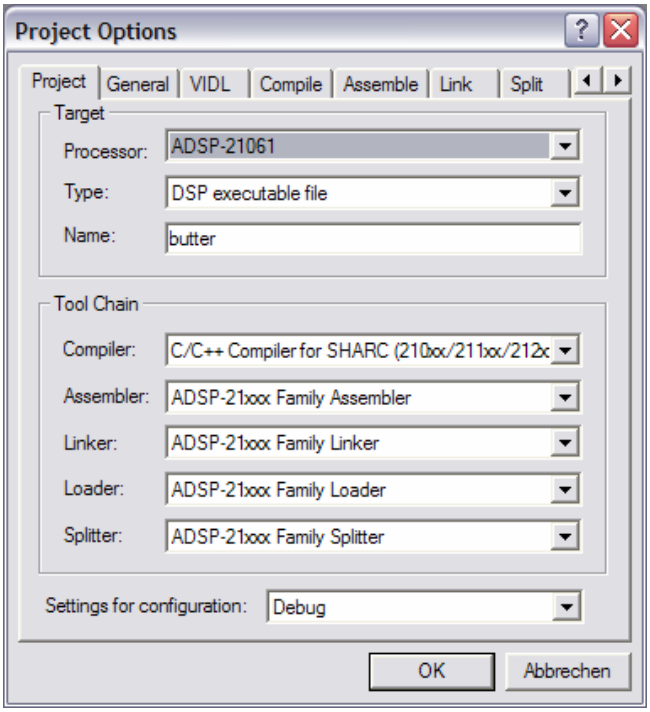


Abbildung 6 – Einstellen der Projektoptionen

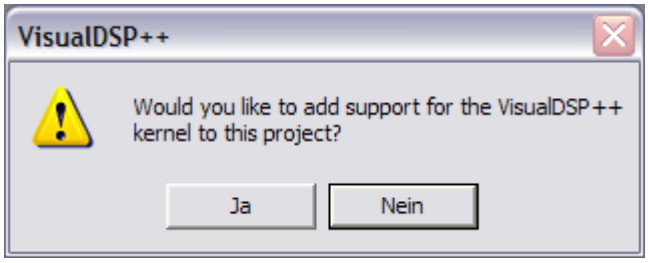


Abbildung 7 – Kernel Support entsagen

Das erscheinende Auswahlfenster (Abb. 7) mit „Nein“ beantworten. Dadurch wird die Verwendung des *VisualDSP++* Kernels untersagt.

Nun existiert ein neues (und noch leeres) Projekt. Mit „File“ → „New“ wird ein leeres Editorfenster geöffnet. Bevor man mit Programmieren beginnt, sollte das leere Fenster abgespeichert werden („File“ → „Save As“). Bei Wahl des Namens entscheidet man über die Programmiersprache, in der diese Datei geschrieben wird:

- Endung .asm → Assemblerdatei
- Endung .c → C – Code Datei
- Endung .h → Header – Datei
- Endung .cpp → C++ Datei

Durch „Project“ → „Add to Project“ → „File(s)“ werden die jeweiligen Dateien nun zum Projekt hinzugefügt. In der Projekthierarchie (linke Spalte) werden die Dateien in den entsprechenden Ordnern angezeigt.

3.2 Einbinden eines Linker – Entscheidung für die Programmiersprache

Der Linker wird benötigt, um das geschrieben Programm an die bestehende Architektur des aktuellen DSP anzupassen. Mehr Informationen dazu können im *„Linker Manual“* erhalten werden.

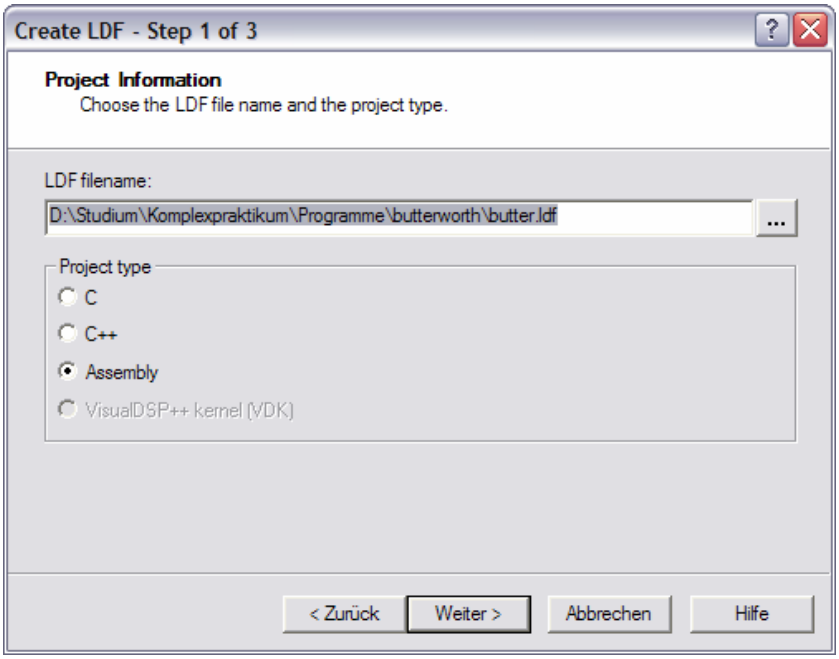


Abbildung 8 – Auswahl des Projekttyps

Erstellen einer Linker (LDF) - Datei:

- Menü Tools → Create LDF, Bestätigen mit OK
- Name (und ggf. Speicherort) der Linker-Datei festlegen
- Auswählen des Projekttyps (Programmiersprache des Projekts) (Abb. 8)
- Auswahl der Systemkonfiguration (Abb. 9)
- Bestätigen der Einstellungen mit Fertigstellen
- Linker Datei erscheint in der Programmhierarchie (linke Spalte)

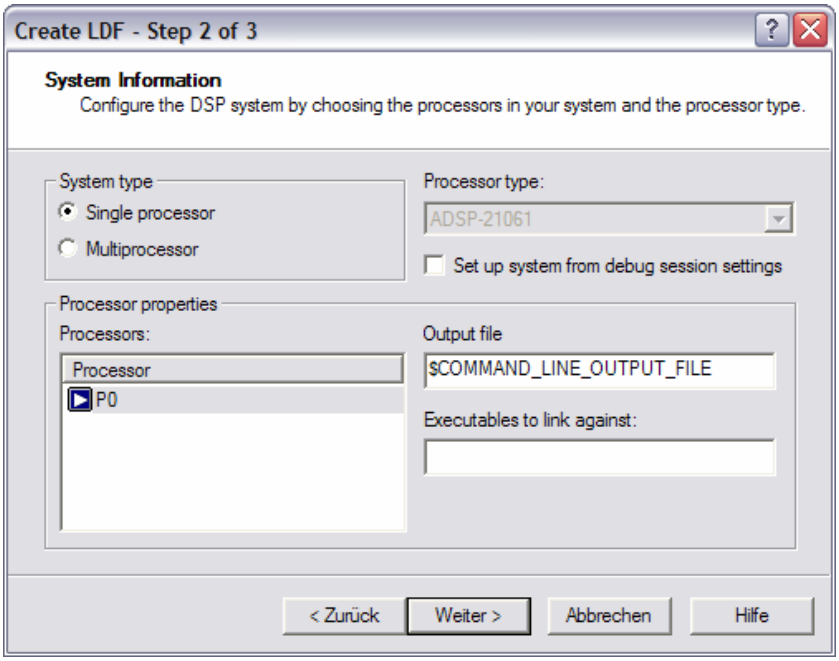


Abbildung 9 – Systemkonfiguration

Für die Entwicklungsplatine existiert ein vorgefertigtes LDF. Da es manchmal zu ungeklärten Komplikationen beim Starten eigener Projekte kommen kann, wird empfohlen, diese LDF-Datei (**Visual_DSP_3_0/ADSP21061.ldf**) einzubinden (in den jeweiligen Projektordner kopieren und mit „Project“ → „Add to Project“ → „File(s)“ zum Projekt hinzufügen).

3.3 Kompilieren, Simulieren und Debuggen des Projektes

Zur Übersetzung des geschriebenen Quellcodes ist ein Compiler zuständig. Mit dem Menüpunkt „Project“ → „Build Project“ oder F7 wird zum einen kompiliert, dann gelinkt und schließlich ein lauffähiges Programm erstellt (build = erstellen). Die Meldung „build successful“ im Output Fenster (unterer Rand) bestätigt das Erstellen dieses lauffähigen Programms. Es entsteht eine Datei mit der Endung .dxe. Bei eventuellen syntaktischen Fehlermeldungen kann der angegebene Fehlercode in der Programmhilfe nachgeschlagen werden.
Nun ist zu empfehlen, das Programm zu simulieren. Dadurch werden logische Fehler erkannt, die ein Compiler naturgemäß nicht finden kann und wird. Hierzu wird zunächst die richtige Session ausgewählt (Menü „Session“ → „Select Session“ → „ADSP 21061 ADSP 2106x-Simulator“). Um Teile des Kernprozessors zu überwachen, kann im Menü „Register“ → „Core“ ausgewählt werden. Zur Auswahl stehen: die 16 Hilfsregister, der Programmspeicher (DAG – PM), der Datenspeicher (DAG – DM), Zählregister, Interruptregister etc.

Zum Starten der Simulation kann im Menüpunkt „Debug“ zwischen schrittweiser Überprüfung („Step into“) oder einem kompletten Durchlauf („Run“) des Programms ausgewählt werden. Durch Setzen von Pausemarken („Breakpoints“) wird das Programm automatisch an diesen Stellen gestoppt und die aktuellen Registerstände angezeigt. Bei einem Komplettdurchlauf des Programms muss vor dem Einsehen dieser Stände „Halt“ (Menü „Debug“) ausgewählt werden.

3.4 Laden und Starten des Programms auf der Entwicklungsplatine

Nach der logischen Funktionsprüfung des Programms muss die Session wieder auf „ADSP 21061 - EZ-KIT“ umgestellt werden. Nach dem Reset der Platine (Aufforderung beachten) kann nun mit „File“ → „Load Program“ das lauffähige Programm (.dxe) in den EPROM der Platine geladen werden.
Wie bereits unter 2.5 beschrieben wird mit „load complete“ im Output Fenster angezeigt, dass das Programm auf dem EPROM gespeichert wurde. Zum Start des Programms muss nun lediglich „Debug“ → „Run“ ausgewählt werden

4. Das Audiosignal auf dem EZ-Kit lite

Grundlage für alle Signalverarbeitungsalgorithmen auf dem Evaluationboard ist das digitalisierte Audiosignal. Dieses Kapitel beschreibt den Weg des analogen Eingangssignals über die Digitalisierung im Codec, das Dateninterface des DSP bis in den Programmspeicher.

4.1. Der Codec

Das Evaluationboard verfügt über den Codec *AD1827* von *Analog Devices*. Grundlage der Digitalisierung des Audiosignals ist ein integrierter 16 Bit Stereo Sigma-Delta Wandler mit 64fachem Oversampling. Die maximale Abtastrate beträgt 48 kHz, liegt also über der einer Audio-CD (44,1 kHz). Weiterhin übernimmt der Codec die Zeitkontrolle für den DSP.
Der Codec verfügt über 16 Kontroll- und Statusregister (siehe Datenblatt), die in erster Linie zur Definition des Datenformats und –interfaces dienen, aber auch Kontrollmöglichkeiten bieten, um den aktuellen Status des Codec während des Programmablaufs durch den DSP zu überwachen.
Vor der Arbeit mit dem Codec muss dieser initialisiert werden. Am Beispiel des „Data Format Registers“ soll das Einstellen eines Kontrollregisters veranschaulicht werden.

IA3:0	Data 7	Data 6	Data 5	Data 4	Data 3	Data 2	Data 1	Data 0
1000	res	FMT	C/L	S/M	CFS2	CFS1	CFS0	CSL

Abbildung 10 – Data Format Register

Die Bedeutung der einzelnen Bits sind dem Datenblatt zu entnehmen. Im Falle des Data Format Registers gilt:

- CSL Taktquelle
- CFS2:0 Takt-Teilerverhältnis
- S/M Stereo / Mono Select
- C/L Companded (µ/A – Law) / Linear
- FMT Format Select (8 Bit / 16 Bit)

Die entsprechende Bitkombination für eine Abtastrate von 48 kHz (CSL, CFS2:0), 16 Bit Stereo (S/M, FMT) bei linearer Quantisierung (C/L) ist `0101 1100`, also `0x5c`. Die Register des Codec sind intern in den Speicherbereich des DSP gespiegelt. Um während der Program-

mierung einen leichten Schreib- / Lesezugriff auf die Register zu erhalten, werden die Adressen in einem Array hinterlegt:

```
#define SZ_regs_1847 16
int regs_1847[SZ_regs_1847] = {
      0xc000,                  // index 0 - left input control
      0xc100,                  // index 1 - right input control
      0xc280,                  // index 2 - left aux 1 input control
      0xc380,                  // index 3 - right aux 1 input control
      0xc480,                  // index 4 - left aux 2 input control
      0xc580,                  // index 5 - right aux 2 input control
      0xc600,                  // index 6 - left dac control
      0xc700,                  // index 7 - right dac control
      0xc850,                  // index 8 - data format
      0xc909,                  // index 9 - interface configuration
      0xca00,                  // index 10 - pin control
      0xcb00,                  // index 11 - no register
      0xcc40,                  // index 12 - miscellaneous information
      0xcd00,                  // index 13 - digital mix control
      0xce00,                  // index 14 - no register
      0x8f00};                 // index 15 - no register
```

Somit ist das Data Format Register über **`regs_1847[8]`** wie eine Variable ansprechbar und kann auch entsprechend beschrieben und ausgelesen werden. In diesem Fall genügt es also, mit **`regs_1847[8]=0x5c;`** das Register zu beschreiben. Die endgültige Übertragung dieser Information vom DSP zum Codec hängt vom verwendeten Dateninterface ab (siehe Abschnitt 4.2.).

4.2. Das serielle Interface des DSP

Um die Abtastwerte vom Codec in den DSP zu übernehmen, arbeiten beide Komponenten im Time Division Multiplex (TDM). Grundlage des TDM sind definierte Rahmen („Frames“), die in je 32 16-Bit Zeitschlitze („time slots“) unterteilt sind. Ein Abtastwert des Codec besteht aus 3 Words (16 Bit), für die folgende Reihenfolge festgelegt ist:

1. Kontroll- bzw. Statusword
2. Abtastwert des linken Kanals
3. Abtastwert des rechten Kanals

Pro gesendeten Rahmen werden zwei Stereo Abtastwerte übertragen, so dass lediglich 6 der vorhandenen 32 Zeitschlitze verwendet werden. In der Grundeinstellung sind dies die Schlitze 0,1 und 2 für den ersten sowie 16,17 und 18 für den zweiten Abtastwert.

Wurde ein solcher Rahmen vom Codec an den DSP gesendet, muss dieser im Speicher abgelegt werden. Hierbei kommt das DMA-Verfahren (Direct Memory Access) zum Einsatz. Der Vorteil hierbei liegt in der Übernahme der Speicherverwaltung vom integrierten DMA-Controllers des DSP und somit der vollständigen Entlastung des Prozessors bei der Übernahme der Abtastwerte in den Speicher.
Da die empfangenen 16-Bit Werte zeitlich versetzt im DSP ankommen, ist es wichtig, dass diesen definierte Speicherstellen zugewiesen werden, um später darauf zugreifen zu können. Da ein Sample aus drei relevanten Zeitschlitzen besteht, wird hierzu ein Empfangspuffer mit der Länge 3 definiert. Um später die bearbeiteten Abtastwerte zum Codec zurücksenden zu können, und für das Senden die gleiche Rahmendefinition verwendet wird, wird analog mit dem Sendepuffer verfahren:

```
unsigned rx_buf[3];
unsigned tx_buf[3];
```

rx_buf[0] ist die Start – Empfangsadresse für den DMA-Controller. Erreicht ein Rahmen den Controller, wird der Inhalt des ersten Zeitschlitzes (hier also das control word) in **rx_buf[0]** geschrieben. Anschließend wird der Zeiger auf die nächste Speicheradresse (**rx_buf[1]**) inkrementiert, so dass an dieser Stelle der Wert des linken Abtastwertes hinterlegt werden kann. Das gleiche geschieht mit dem rechten Abtastwert. Im Anschluss wird der Pointer wieder auf **rx_buf[0]** zurückgesetzt, um das nächste Sample zu empfangen. Als Ergebnis stehen nun in **rx_buf[1]** und **rx_buf[2]** die Samples des Stereo-Signals als 16 Bit signed Integer – Werte (-32768 … 32767) zur Weiterverarbeitung zur Verfügung.
Das Senden erfolgt analog, hierbei muss lediglich **tx_buf** gemäß den Vereinbarungen für die Zeitschlitze beschrieben werden.

5. Kommunikation zwischen PC und EZ-Kit lite

Während der Entwicklung der verschiedenen Audio-Module entstand das Ziel, diese in einem Rahmenprogramm zu vereinen und mittels externer Eingaben zu steuern. Da die Entwicklungsplatine bereits über eine serielle Schnittstelle verfügt, bietet es sich an, diese für eigene Zwecke nutzbar zu machen, zumal so eine einfache Kommunikation ohne zusätzliche Hardware oder Modifikationen am Board möglich wird.

Mittels dieses Kommunikationsweges sollen folgende Aufgaben erfüllt werden:

- Auswahl der einzelnen Audio-Module
- Ausgabe von Messergebnissen
- Eingabe von Parametern (Zeitintervalle, Filterkoeffizienten)

Dieser Abschnitt erläutert die Implementierung, Initialisierung und Nutzung der Schnittstelle.

5.1. Implementierung der Schnittstelle

Die Kommunikation erfolgt vom DSP über den bereits implementierten UART (Universal Asynchronous Receiver & Transmitter) – Chip, der in Verbindung mit einem RS-232 Treiber eine Verbindung mit einer seriellen Schnittstelle ermöglicht. Der verwendete UART-Schaltkreis *PC16550D* bietet die Möglichkeit der freien Baudraten-Wahl und ein programmierbares serielles Interface zur Einstellung des verwendeten Datenformats über mehrere Status- und Kontrollregister und verfügt somit über alle notwendigen Funktionen, um die Schnittstelle für eigene Anwendungen nutzbar zu machen.

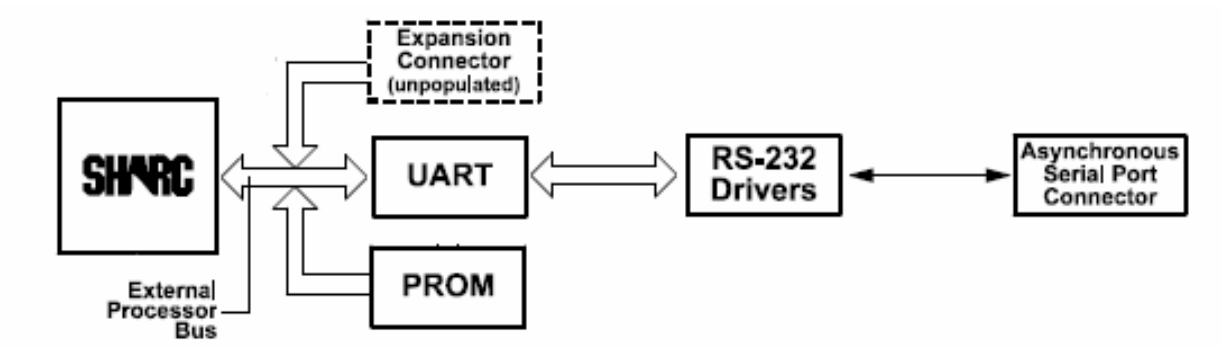


Abbildung 11 – Signalflussdiagramm DSP → Serieller Port

Das Problem hierbei besteht darin, dass sich der serielle Port nicht ohne größeren Aufwand für eigene Zwecke nutzen lässt, da die Schnittstelle durch die Entwicklungsumgebung *VisualDSP++* bereits belegt ist und hierbei für Debug- und Monitoring - Zwecke genutzt wird. Selbst nachdem ein Programm mittels „Debug“ → „Run“ gestartet wurde, erfolgt ein stetiger Datenaustausch zwischen PC und Board, um den Programmablauf steuern zu können und den Speicherinhalt zu überwachen. Da dieser Zugriff Interrupt - basiert erfolgt, besteht die einzige Möglichkeit darin, den betreffenden Interrupt (IRQ2) innerhalb des eigenen Programms zu deaktivieren:

```
interrupt (SIG_IRQ2, SIG_IGN);        // UART-Interrupt deaktivieren
```

Dies sollte bereits zu Beginn des Programmablaufs erfolgen, um im Anschluss die UART-Register nach den eigenen Bedürfnissen zu konfigurieren.

Hinweis:
Durch Deaktivieren des Interrupts gehen während des Programmablaufs alle Debug- und Monitoring Funktionen von *VisualDSP++* verloren.

5.2. Initialisierung der UART-Register

Nachdem der Interrupt deaktiviert wurde, ist es möglich, innerhalb des eigenen Programms auf die Register des UART-Chips zuzugreifen und diese zu modifizieren. Wie in Abb. 11 zu erkennen ist, erfolgt der Zugriff über den gleichen externen Bus, der auch für die Kommunikation mit dem EPROM und dem (optionalen) externen Speicher verwendet wird.
Die UART-Register werden durch den DSP in den externen Speicher gespiegelt („mapping“). Da die Register also intern als Speicherbereich behandelt werden, kann mit den gleichen Schreib- und Leseoperationen gearbeitet werden, wie es die Programmierung für jeden anderen Speicherbereich auch verlangt.
Der *PC16550D* verfügt intern über 12 Register (siehe Datenblatt), die separat beschrieben und/oder gelesen werden können. Im Speichermodell sind die einzelnen Register als Offset zur Basisadresse des UART-Chips zu verstehen. Diese Basisadresse ist intern fest als `0x0402000` definiert. Es erleichtert die Arbeit mit den Registern erheblich, wenn diese mit Pointern versehen werden, um sie im späteren Verlauf des Programms direkt über den Zeiger

ansprechen zu können, ohne jedes Mal die genaue Adresse angeben zu müssen. Diese Pointer-Definition sollte gleich zu Beginn der Initialisierung erfolgen:

```
#define base 0x0402000                    // Basisadresse

#define RBR ((volatile char*) base )      // Receiver Buffer Register
#define THR ((volatile char*) base )      // Transmitter Holding Register
#define IER ((volatile char*) (base+1))   // Interrupt Enable Register
#define IIR ((volatile char*) (base+2))   // Interrupt Ident. Register
#define FCR ((volatile char*) (base+2))   // FIFO Control Register
#define LCR ((volatile char*) (base+3))   // Line Control Register
#define MCR ((volatile char*) (base+4))   // Modem Control Register
#define LSR ((volatile char*) (base+5))   // Line Status Register
#define MSR ((volatile char*) (base+6))   // Modem Status Register
#define SRC ((volatile char*) (base+7))   // Scratch Register
#define DLL ((volatile char*) (base ))    // Divisor Latch (LS) (DLAB=1)
#define DLM ((volatile char*) (base+1))   // Divisor Latch (MS) (DLAB=1)
```

Nachdem die Register nun über ihre Kurzbezeichnung ansprechbar sind, können diese wie Variablen behandelt werden, indem sie beschrieben werden oder aus ihnen gelesen wird.
Am Beispiel des Line Control – Registers (LCR), mit dem das verwendete Datenformat konfiguriert wird, soll einer der Initialisierungsschritte veranschaulicht werden:

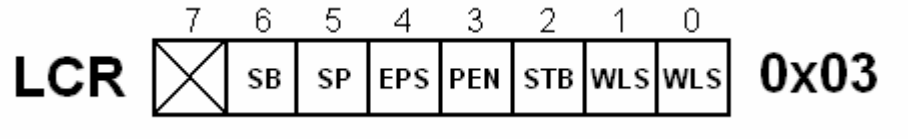


Abbildung 12 – Line Control Register

Im Falle des Demonstrationsprogramms soll mit der einfachsten Einstellung (8 Datenbits, 1 Stopp – Bit, keine Fehlerkontrolle (Parity Check)) gearbeitet werden. Die äquivalente Bitkombination hierfür lautet **0000 0011**, also **0x03**. Um diesen Wert in das LCR-Register zu schreiben, muss die Speicheradresse **0x0402003** (Basisadresse + Offset **0x03**) den Inhalt **0x03** erhalten. Da für alle Register bereits im Vorfeld Pointerdefinitionen erfolgt sind, lässt sich diese Operation sehr einfach mit

```
*LCR = 0x03; // 1 Stop Bit, 8 Daten Bits, Parity = 0
```

realisieren.

5.3 Senden und Empfangen

Für den Datenempfang unterstützt der *PC1550D* grundsätzlich zwei Betriebsmodi: Den Interrupt - basierten Betrieb und das Polling (Abfrage). Da der Interrupt deaktiviert wurde, bleibt nur die Polling – Variante. Hierzu ist es nötig, das Empfangsregister in geeigneten Zeitabständen manuell abzufragen. Für simple Tastaturabfragen, wie im Falle dieses Demoprogramms ist es völlig ausreichend, einen Timerinterrupt mit f = 4Hz zu definieren. Besteht eine Verbindung zum PC und wurde ein Zeichen empfangen, steht dies im Receive Register Buffer (***RBR**) zur Verfügung und kann nach Prüfung des Data Ready – Bits im Line Status Register ausgelesen werden:

```
char DR=0x01;                          // RX Checkmaske
if ((*LSR & DR)==DR) zeichen=*RBR;     // empfangenes Zeichen
```

Um Daten zu senden ist es lediglich notwendig, zu prüfen, ob das Transmitter Holding Register (***THR**) leer ist und im Anschluss das zu sendende Zeichen in das Register zu schreiben:

```
char TEMT=0x40;                        // TX Checkmaske
if((*LSR & TEMT)!=TEMT) *THR=zeichen  // zu sendendes Zeichen
```

Nachdem die Schnittstelle initialisiert wurde und (im einfachsten Fall) mittels eines Terminal-Programms (z.B. *HyperTerminal*) eine Verbindung zum Board hergestellt wurde, können Daten gesendet und empfangen werden. Vorteilhaft sind in diesem Zusammenhang die Sende- und Empfangsfunktionen in der Datei **uart.h** auf der beiliegenden CD!

6. Das Demonstrationsprogramm „DSP Audio Demo“

Um die Möglichkeiten der Audiosignalverarbeitung anschaulich demonstrieren zu können, wurden die einzelnen Modulprogramme (Änderung der Samplerate des Codec, Mittelwertbestimmung und verschiedene Filter) in einem Rahmenprogramm vereint.
Dieser Abschnitt erläutert das Laden und Steuern des Demonstrationsprogramms, erklärt die einzelnen Audiomodule und gibt Hinweise zur einfachen Implementierung eigener Signalverarbeitungsalgorithmen.

6.1. Initialisierung

Die Steuerung des Programms sowie die Ausgabe der Messwerte erfolgt vollständig durch ein Terminal-Programm. Im Folgenden werden die notwendigen Einstellungen zur Kommunikation mit dem Evaluation Board anhand des im Lieferumfang von *Microsoft Windows* enthaltenen Programms *HyperTerminal* erläutert:
Zunächst wird *HyperTerminal* gestartet (üblicherweise zu finden unter „Start“ → „Programme“ → „Zubehör“ → „Kommunikation“ → „HyperTerminal“). Nun fordert *HyperTerminal* zum Einrichten einer neuen Verbindung auf. Nach Eingabe eines Namens für die Verbindung und einem Klick auf OK erscheint ein neues Fenster. Im Feld „Verbinden über…“ muss der COM-Port gewählt werden, mit dem das Board mit dem PC verbunden ist. Im nächsten Fenster werden die Einstellungen zum verwendeten Datenformat getroffen. Diese müssen den Werten entsprechen, die bei Konfiguration der UART-Schnittstelle eingestellt wurden (vgl. Abschnitt 4.2.). Im Fall des Demonstrationsprogramms sind dies:

Baud Rate:	38400
Datenbits:	8
Parität:	keine
Stoppbits:	1

Mit einem Klick auf „OK“ werden die Eingaben bestätigt. Es ist empfehlenswert, die Verbindungseinstellungen mit Klick auf „Datei“ → „Speichern“ zu speichern, um später darauf zugreifen zu können.

Um das Demonstrationsprogramm zu laden, wird zunächst die Entwicklungsumgebung *VisualDSP++* gestartet und das Projekt **dsp_audio_demo.dpj** geöffnet, kompiliert und gestartet (vgl. Abschnitt 2.6.). Ob das geladene Programm läuft, signalisiert die blinkende LED für das Flag 2 auf dem Board. Nach erfolgreichem Start des Programms muss die Entwicklungsumgebung beendet werden, um den Zugriff auf den COM-Port freizugeben. Der Programmablauf auf dem Board bleibt hiervon unberührt.

Im Anschluss wird die oben erstellte *HyperTerminal* Konfiguration geöffnet. Sind alle Einstellungen korrekt, wird automatisch eine Verbindung zum Board aufgebaut. Nach Betätigung des Schalters S2 auf der Entwicklungsplatine erscheint das Hauptmenü:

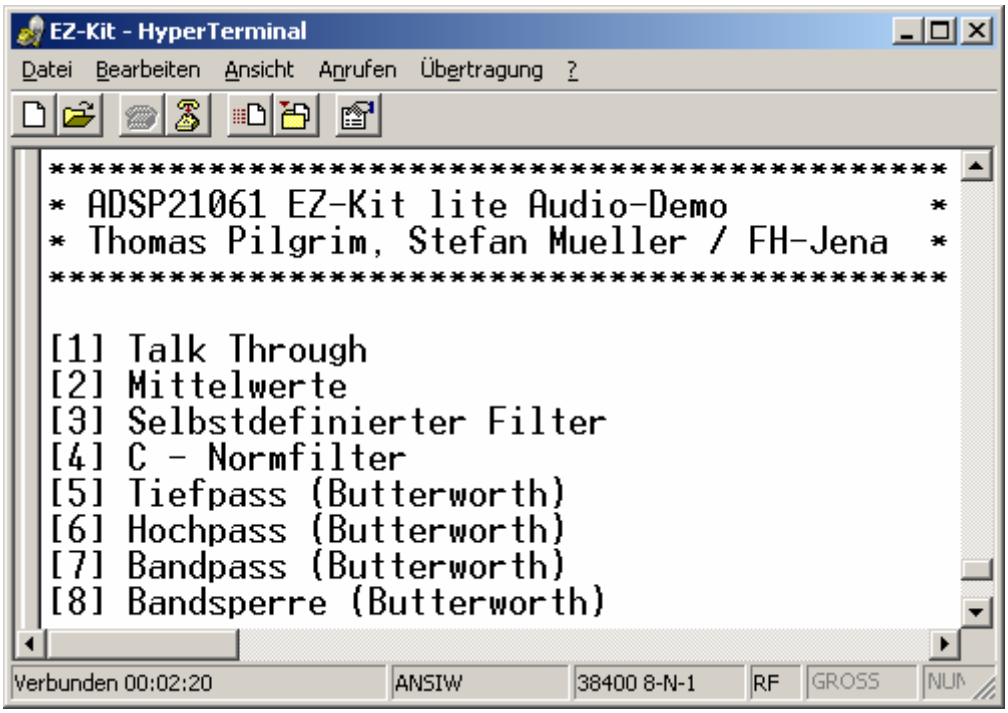


Abbildung 13 – Hauptmenü Demonstrationsprogramm

Um ein Audiomodul zu starten, ist die betreffende Ziffer im Terminal-Programm einzugeben. Innerhalb jeden Moduls führt [ESC] zurück ins Hauptmenü.

6.2. Die Audiomodule

6.2.1. Talk Through

Dieses Modul veranschaulicht den Weg vom analogen Eingang des Boardes über die Digitalisierung im Codec und anschließend die Übergabe des quantisierten Signals in den DSP. Hier

wird das Signal aus dem Empfangspuffer des seriellen Ports (SPORT1) direkt wieder an den Sendepuffer geschickt und steht somit (unverändert) wieder als analoges Signal am Ausgang des EZ-Kit zur Verfügung. Die Fähigkeit des Codec, das Signal mit verschiedenen Abtastraten zu digitalisieren, kann mit [+] oder [-] eingestellt und direkt im Audiosignal hörbar gemacht werden.

6.2.2. Mittelwertbestimmung

In diesem Beispiel sollen einfache Signalverarbeitungsalgorithmen auf das Audiosignal angewendet werden. Hierzu werden aus dem digitalisierten Audiosignal wahlweise der lineare Mittelwert [M] oder der für die Audiotechnik wichtige Effektivwert (RMS – „Root Mean Square) [E] bestimmt.
Die Berechnung der Mittelwerte erfolgt formal nach folgenden Berechnungsvorschriften:

Mittelwert :

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i) \qquad (6.01)$$

Effektivwert:

$$x_{eff} = \frac{1}{n} \cdot \sum_{i=0}^{n} \sqrt{x_i^2} \qquad (6.02)$$

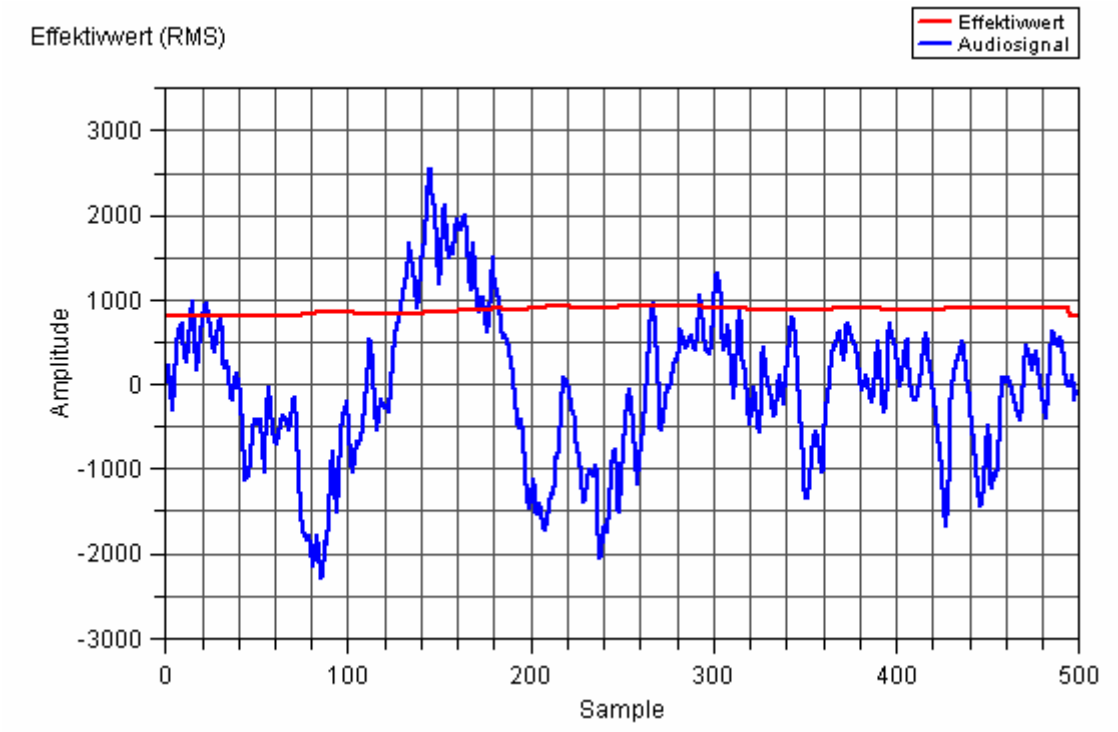


Abbildung 14 – Effektivwert eines Audiosignals

Die verschiedenen Mittelwerte werden im Programm über ein definiertes Zeitfenster T bestimmt. Die Länge dieses Zeitfensters bestimmt die Anzahl der Samples, die programmintern in einem Ringspeicher der Länge n abgelegt werden (wobei n die Anzahl der Samples angibt). In dieser Demonstration fließen 1000 Samples in die Berechnung ein, was bei einer Abtastrate von 48kHz einem Zeitfenster T = 20,8ms entspricht. Der Vorteil des Ringspeichers ist, dass jeder neue Abtastwert automatisch den ältesten Wert ersetzt. Da eine komplette Neuberechnung laut Formel für jeden Abtastwert aus Gründen der Programmgeschwindigkeit unmöglich ist, erfolgt die Berechnung iterativ (siehe Quelltext). Wegen des begrenzten Speicherplatzes des ADSP21061 beschränkt sich diese Demonstration bei der Berechnung auf den linken Kanal des Stereosignals. Abbildung 14 zeigt den Verlauf eines beliebigen Audiosignals und des zugehörigen Effektivwertes über einem Zeitfenster von ca. 10ms (500 Samples).

6.2.3. Filter-Module

Die Module [3] bis [8] veranschaulichen das Prinzip digitaler Filter für Audioanwendungen. Alle Demonstrationen beruhen auf dem Prinzip eines rekursiven IIR-Filters. Für die Berechnung des Ausgangssignals wurde auf bestehende Filterfunktionen für den ADSP21061 zurückgegriffen. Grundlage eines jeden digitalen Filters sind die Filterkoeffizienten, die - zusammen mit der Filterordnung- das Frequenzverhalten bestimmen. Der theoretische Hintergrund und Wege von der analogen Schaltung zu einem digitalen Filter sowie Hinweise zum Verständnis digitaler Filter gibt der Abschnitt *7. Berechnen von Filterschaltungen – Aufbereiten für einen DSP*.

Modul [3] erlaubt es, einen eigenen Filter zu definieren und die Wirkung direkt im Audiosignal hörbar zu machen. Zur Eingabe eigener Filterkoeffizienten innerhalb des Modules [c] wählen. Die Koeffizienten werden als Dezimalzahl eingegeben, wobei das Komma durch einen Punkt darzustellen ist. Bitte beachten, dass aus Gründen des Speicherplatzes im Codespeicher auf eine sichere Eingabe verzichtet wurde und somit falsche oder falsch eingegebene Koeffizienten unweigerlich zur Deaktivierung der Filterfunktion führen und diese erst nach erneutem Kompilieren wieder zur Verfügung steht.

Modul [4] wendet den C-Normfilter auf das digitalisierte Audiosignal an. Der C-Filter ist ein in der Audiotechnik ein sehr wichtiger Filter, da er die Lautstärke des Signals frequenzabhängig an das Lautstärkeempfinden des menschlichen Gehörs anpasst (vgl. Abschnitt 7.4)

Die Module [5] bis [8] demonstrieren einfache Filter nach dem Butterworth-Prinzip. Die Ergebnisse der Filter werden direkt im Ausgangssignal hörbar. Zur Auswahl stehen Hoch- und

Tiefpass sowie Bandpass und –sperre. Das spektrale Verhalten dieser Filter ist aus den Analysen des Audioanalysators zu entnehmen (Abb. 22)

6.3. Einbinden eigener Algorithmen

Um eigene Algorithmen zu programmieren und einbinden zu können, ohne sich um die Initialisierung von DSP, Codec und UART kümmern zu müssen, wurde die Menüstruktur erweiterbar gestaltet. Im Folgenden werden die einzelnen Schritte erläutert, um das vorgegebene Menüsystem um eigene Module zu ergänzen. Hierzu sind Modifikationen in den Dateien menu.h und main.c notwendig.

Modifikationen in menu.h:

- Variable **`menulength`** um 1 erhöhen
- Array **`*menudef`** um Nummer und Modulbezeichnung erweitern
- neues Modul erstellen (Vorlage: **`void modul_blank (void)`**)
  Hinweis: Ein Modul enthält nur die Bildschirmausgabe, nicht den eigentlichen Signalverarbeitungsalgorithmus!
- in **`void menu_control (void)`** das erstellte Modul in eine neue case - Anweisung einbinden
- falls Tastatureingaben behandelt werden sollen: **`void menu_action (void)`** um eine case – Anweisung erweitern

Modifikationen in main.c:

- in **`void spr0_asserted (int sig_num)`** eine neue case – Anweisung definieren, die die den eigentlichen Signalverarbeitungsalgorithmus aufruft

Nach diesen Veränderungen das Projekt neu kompilieren und laden. Im Anschluss steht im Terminal-Programm der neue Menüpunkt zur Verfügung.
Wie bereits einleitend erwähnt, funktioniert der ADSP 21061 nach der Harvard Architektur, dies sollte man stets für eine sinnvolle, effiziente und schnelle Programmierung beachten.

7. Berechnen von Filterschaltungen – Aufbereiten für einen DSP

Dieser Abschnitt befasst sich mit dem Errechnen von Filterkoeffizienten, welche in einem DSP verarbeitet werden können. Grundlage hierbei sollen real existierende (und funktionierende) analoge Filterschaltungen sein. Also Schaltungen, die mit diskreten Bauelementen (Widerstand, Kondensator, Spule) aufgebaut werden und ein Eingangssignal in Form eines Tief-, Hoch- oder Bandpasses bzw. einer Bandsperre behandeln.
Diese Berechnung durchläuft mehrere Etappen:

- Beschreibung der Schaltung mit Hilfe von Übertragungsfunktionen (A – Matrix, Laplace – Transformation)
- Umformen dieser in ein digitales System
- Umformen der Koeffizientenvektoren in richtige Reihenfolge

Die folgenden Punkte werden diese Berechnung genauer erläutern und am Beispiel eines einfachen RC Tiefpasses vollziehbar machen.

7.1. Mathematische Grundlagen

Als Ausgangspunkt für digitale Filter werden oft analoge Schaltungen zugrunde gelegt, da diese stabil sind und keine unerwünschten Nebeneffekte (Aufschwingung der Schaltung, Frequenzeinbrüche, ungewisses Verhalten bei f→ ∞ oder f → 0) aufweisen.
Zunächst wird von diesem Filter die Übertragungsfunktion ermittelt. Mit Hilfe der A - Matrix geht dies auf sehr einfachem Weg. Für die A - Matrix wird der analoge Filter in Längs – und Querzweig zerlegt.

Für den Längszweig gilt: $$A_L = \begin{bmatrix} 1 & Z \\ 0 & 1 \end{bmatrix} \tag{7.01}$$

Für den Querzweig gilt: $$A_Q = \begin{bmatrix} 1 & 0 \\ Y & 1 \end{bmatrix} \tag{7.02}$$

Die Gesamtmatrix A erhält man indem man die Längs- und Querglieder multipliziert (Achtung: Matrizenmultiplikation).

$$A_{gesamt} = \prod_{i=1}^{N} A_i \tag{7.03}$$

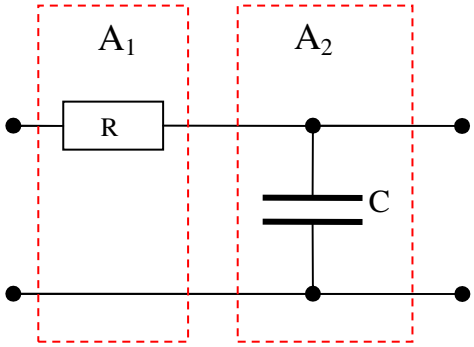
wobei: N ... Anzahl der Glieder

Um auf die Gesamtübertragungsfunktion G(p) bzw. G(jω) zu gelangen gilt:

$$G(j\omega) = \frac{U_2}{U_1} = \frac{1}{A_{11}} \qquad (7.04)$$

wobei: A_{11} ... A Parameter der 1.Spalte & 1.Zeile

Am Beispiel des RC – Tiefpasses wird dieser Zusammenhang nun verdeutlicht:



$$A_1 = \begin{bmatrix} 1 & R \\ 0 & 1 \end{bmatrix} \; ; \; A_2 = \begin{bmatrix} 1 & 0 \\ j\omega C & 1 \end{bmatrix}$$

A - Matrizen berechnet nach 7.01 und 7.02

Abbildung 15 – Beispielschaltung (RC - Tiefpass)

Mit Hilfe des Falkschemas (Abb.16) kann man nun die Gesamtmatrix bestimmen.

$A_{gesamt} = A_1 \cdot A_2$		$\begin{matrix} 1 & 0 \\ j\omega C & 1 \end{matrix}$
$\begin{matrix} 1 \\ 0 \end{matrix}$	$\begin{matrix} R \\ 1 \end{matrix}$	$\begin{matrix} 1 + j\omega RC & R \\ j\omega C & 1 \end{matrix}$

Abbildung 16 – Falkschema zum Erhalt von A_{Gesamt}

Wir erhalten also aus Abb. 16 und Formel (7.03)

$$A_{gesamt} = \begin{bmatrix} 1 + j\omega RC & R \\ j\omega C & 1 \end{bmatrix} \qquad (7.05)$$

Nach Formel (7.04) gilt nun für die Gesamtübertragungsfunktion des RC – Tiefpasses:

$$G(j\omega) = \frac{1}{1 + j\omega RC} \tag{7.06}$$

Sicherlich hätte man diese Übertragungsfunktion schneller über andere Wege erhalten können, jedoch dient dieses Beispiel der Verdeutlichung und soll prinzipielle Wege aufzeigen, um auch komplexere Schaltungen auf diesem Weg lösen zu können.

7.2. Vom analogen zum digitalen Filter

7.2.1. Klassifizierung der digitalen Filter

Grundlegend werden 2 Typen von digitalen Filtern unterschieden:

a) *IIR Filter* (= infinite impulse response) → Filter mit einer unendlichen Impulsantwort,
Die Berechnung des Ausgangswertes erfolgt aus den aktuellen und vergangenen Eingangs- und Ausgangswerten.

Für den IIR Filter gilt folgende allgemeine Form:

$$G(z) = \frac{\sum_{k=o}^{n} b_k \cdot z^{-k}}{a_0 + \sum_{k=1}^{n} a_k \cdot z^{-k}} \tag{7.07}$$

wobei: b_k – Koeffizienten des Zählerpolynoms (X – Werte)
a_K – Koeffizienten des Nennerpolynoms (Y – Werte)
z^{-n} – vergangener n – ter X oder Y – Wert
a_0 auf 1 normiert wird

b) *FIR Filter* (= finite impulse response) → Filter mit einer endlichen Impulsantwort,
Hier erfolgt die Berechnung des Ausgangswertes aus aktuellen und vergangenen Ein gangswerten.

Für den FIR Filter gilt folgende allgemeine Form:

$$G(z) = \sum_{k=o}^{n} b_k \cdot z^{-k} \tag{7.08}$$

Der große Vorteil dieses Filtertyps ist die Tatsache, dass er stets stabil ist, da er keine Polstellen besitzt. Dies ist aber in gewisser Weise auch ein Nachteil, da erheblich höhere Ordnungen (und dementsprechend mehr vergangene X - Werte) dieses Filtertyps benötigt werden, um ähnlich Übertragungscharakteristika (Flankensteilheit etc.) wie die der IIR Filter zu erhalten.

Da für Audiofilterberechnung sehr leistungsfähige Prozessoren entwickelt wurden und der IIR Algorithmus schneller zum gewünschten Ziel führt, favorisiert man hier oft den IIR Typ.

7.2.2. Die Bilineare Transformation

Durch die bilineare Transformation wird nun aus einem analogen Filter einen digitaler IIR-Filter gewonnen. Allgemein gilt für die Z – Transformation:

$$z = e^{j\omega T} \tag{7.09}$$

mit: T ... Abtastperiode

Aus (7.09) erhält man

$$j\omega = \frac{1}{T} \cdot \ln z \tag{7.10}$$

Da ln(z) eine transzendente Funktion ist (eine Funktion, die sich nicht komplett über Polynomfunktionen n – ten Grades beschreiben lässt), wird sie durch die Reihenentwicklung des Logarithmus Naturalis angenähert (Abbruch nach dem linearen Glied) und man erhält die Vorschrift der bilinearen Transformation:

$$j\omega = \frac{2}{T} \cdot \left(\frac{z-1}{z+1} \right) \tag{7.11}$$

Betrachten wir wieder das RC - Beispiel, so ist die Gleichung (7.11) nun als Approximation in Gleichung (7.06) einzusetzen. Nach zusätzlichem diversem Umformen erhält man:

$$G(z) = \frac{T \cdot z + T}{(T + 2 \cdot RC)z + T - 2 \cdot RC} \tag{7.12}$$

bzw. um es auf die allgemeine Form nach Gleichung (7.10) umzuformen:

$$G(z) = \frac{\frac{T}{T + 2 \cdot RC} + \left(\frac{T}{T + 2 \cdot RC}\right) \cdot z^{-1}}{1 + \left(\frac{T - 2 \cdot RC}{T + 2 \cdot RC}\right) \cdot z^{-1}} \tag{7.13}$$

Durch konkrete Werte für R,C & T erhält man nun die Koeffizienten der Übertragungsfunktion. Für eine einfachere Rechnung bietet es sich an, mit normierten Werten zu arbeiten:

T = 0,5 R = 1 C = 2

Diese Werte in (7.13) eingesetzt ergeben:

$$G(z) = \frac{0{,}11\overline{1} + 0{,}11\overline{1} \cdot z^{-1}}{1 - 0{,}77\overline{7} \cdot z^{-1}} \tag{7.14}$$

Zur Kontrolle kann hierfür das Matlab Programm **RC_Tiefpass.m** herangezogen werden.

7.2.3. Die Koeffizienten

Da die Struktur von 7.07 stets gleich ist und sich nur die Koeffizienten vor den z – Werten ändern, gibt man bei einem digitalen Filter oft nur die Koeffizienten an, da diese das System eindeutig beschreiben. Sie werden in Zählerpolynomkoeffizienten und Nennerpolynomkoeffizienten unterschieden. In diesem Beispiel wären dies in Zeilenform:

$$\begin{aligned} b &= \left(0{,}11\overline{1} \quad 0{,}11\overline{1}\right) \\ a &= \left(1{,}00 \quad -0{,}77\overline{7}\right) \end{aligned} \tag{7.15}$$

Es ist weiterhin von großer Bedeutung, die Rechengenauigkeit vollkommen auszunutzen.
Die Rechen-Genauigkeit von *MatLab* beispielsweise liegt bei ca. 16 Dezimalstellen. Diese Genauigkeit sollte, sofern sie der jeweilige Prozessor auch verarbeiten kann auch genutzt werden. Durch die Tatsache, dass es sich beim *ADSP - 21061* um einen 32Bit Gleitkommaprozessor handelt, ist die Genauigkeit extrem hoch und die Abweichungen aufgrund von Rundungsfehlern verschwindend gering.

Weiterhin ist darauf zu achten, welche Vereinbarungen das System für die a bzw. b – Koeffizienten verlangt. Im Fall von *MatLab* gilt:

b → Zählerkoeffizientenvektor (englisch: numerator)
a → Nennerkoeffizientenvektor (englisch: denumerator)

Es lässt sich schon an diesem einfachen Beispiel erkennen, dass eine komplexere Schaltung sehr schnell zu einem riesigen Rechenaufwand führt, der auf schriftlichem Weg kaum zu bewältigen ist.

7.3. Umgang mit größeren Schaltungen – der PC hilft

Für komplexere Schaltung ist der Gebrauch einer Berechnungssoftware unbedingt von Nöten. Im Ordner **Matlab** auf der beiligenden CD befinden sich einige m – Dateien, die zur Verdeutlichung und als Beispiel dienen sollen.
Es gibt natürlich auch unter *MatLab* mehrere Wege zum Erhalt der Koeffizienten. Die m – Datei **C_Filter.m** enthält einige unterschiedliche Berechnungen.

7.3.1. Die A- Matrix unter Matlab

In **C_Filter.m** ab Zeile 49 wird diese Variante vorgestellt. Sie ist erst ab der Version 7 von *MatLab* möglich. Für ältere Versionen empfiehlt sich die Variante unter 7.3.2.

Mit der Zeile 40:

```
s = tf('s');
```

(tf → transfer function)

erklärt man unter Matlab den Buchstaben „s“ als Laplace Operator ρ.

Nun gibt man die verschieden A – Matrizen ein (Zeile 55 – 58 unter *MatLab* ist das Semikolon das Zeichen für eine neue Spalte).
Zeile 61 zeigt den großen Vorteil von *MatLab*, denn durch Multiplikation der Teilmatrizen wird nun die Gesamtmatrix berechnet.
Wie Gleichung (7.04) aufzeigt wird mit Zeile 62 die Gesamtübertragungsfunktion aus der A - Matrix gewonnen. Zeile 63 stellt nur eine Anhebung von Ga(p) dar.

Mit Zeile 64:

```
[b,a]=tfdata(Ga1,'v')
```

erhält man die Koeffizienten des analogen Systems (äquivalent zu 7.06). Wenn man sich diese Koeffizienten ansieht, erkennt man, dass jeweils 2 Nullen den Vektor abschließen. Durch Weglassen dieser (Zeilen 65-66) wird mathematisch durch p^2 geteilt (Abhängig vom Filter möglich oder nicht). Wichtig ist, dass beide Vektoren die gleiche Anzahl von Nullen gekürzt bekommt.
Zur Darstellung der Funktion sind die Zeilen 67 – 69 mit den Funktionen **`freqs()`** und **`semilogx()`** zu benutzen.

7.3.2. Die Funktion analyse_kbs ()

Herr Professor Ostritz hat eigens eine Funktion für ältere Versionen von *MatLab* geschrieben. Ab Zeile 30 bis 47 wird diese Variante beschrieben. Die Matrix **`SCH[]`** sollte anfangs leer übergeben werden. Hierdurch wird man im *Matlab* „Command Window“ aufgefordert, die Längs- und Querzweige der Filterkoeffizienten einzugeben. Durch Öffnen der Funktion **`analyse_kbs()`** wird erklärt, wie man vorgehen sollte.

Hinweise:

- Das Ausgabeformat sollte auf „long“ umgestellt werden (**`format long;`**), um die interne Rechengenauigkeit von *Matlab* auch anzuzeigen (16 Stellen)
- Nach Beenden der Eingabe der Komponenten sollte man die erhaltene **`SCH[]`** Matrix in einer m – Datei abspeichern, um diese nicht jedes Mal neu eingeben zu müssen.

Die Rückgabeparameter der Funktion sind unter anderem G(p), welches dann wie unter 7.3.1 beschrieben, dargestellt werden kann.

Hinweis:

Neben der eigentlichen Funktion muss sich ebenfalls die Funktion **`mamul()`** im Arbeitsverzeichnis von *Matlab* befinden, welche die Matrizenmultiplikation durchführt.

7.3.3. Berechnung der digitalen Koeffizienten

Zur Berechnung der digitalen Koeffizienten stellt *Matlab* ebenfalls verschiedene Funktionen zur Verfügung. Wichtig ist im Allgemeinen die Angabe einer Abtastperiode T_S bzw. einer Abtastfrequenz f_S. Diese ist natürlich abhängig von der im Codec eingestellten Abtastfrequenz (siehe Abschnitt 4.1, bzw. bei Arbeit mit dem Demonstrationsprogramm auch 6.2.1). In dem Programm **C_Filter_digitalVGL.m** werden einige unterschiedliche Varianten zur Bestimmung der Koeffizienten verglichen, wobei die Abtastfrequenz auf f_{Smax} = 48000 Hz festgelegt ist.

a) Mit Zeile 70

```
Gz1 = c2d(tf(b,a),Ts)
```

(c2d → continuous to discrete)

wird die digitale Übertragungsfunktion berechnet. Voraussetzungen sind natürlich die analogen Koeffizientenvektoren a & b. Als zusätzlichen Parameter kann bei **`c2d()`** noch die Art der Approximation angegeben werden. Für den Fall, dass kein Parameter übergeben wird, wird automatisch „zero order hold“ (zoh) eingestellt (= klassisches Sample & Hold Verfahren). Mit Zeile 85 wird durch „first order hold“ (foh) approximiert (= lineare Approximation der analogen Abtastwerte).

b) In Zeile 56

```
[bz1,az1]=bilinear(b,a,Fs*2*pi)
```

werden die digitalen Koeffizienten nach Gleichung (7.11) berechnet. Zur Darstellung können die Funktionen **`freqz()`** bzw. -unter Version 7 verfügbar- die Funktion **`fvtool()`** verwendet werden.

7.3.4 Formung der Koeffizienten

Für einige Prozessoren muss die Reihenfolge der Vektorkomponenten vertauscht werden. Der SHARC ADSP21061 gehört zu diesen. Mit den Funktionen

`flipud ()` → flip up to down (Spaltenvektor) bzw.

`fliplr ()` → flip left to right (Zeilenvektor)

werden die Koeffizienten in der Reihenfolge gedreht.

Da sich die IIR – Berechnungsvorschrift der *C Runtime Library* von der Berechnungsvorschrift unter *Matlab* unterscheidet (siehe **Filterberechnung\IIR_Filter_fuer_C.pdf**), müssen die Vorzeichen der A – Koeffizienten negiert werden.

Weiterhin wird a_0 des A – Vektors weggelassen, um Rechenzeit zu sparen, da diese 1 ja den aktuellen Y-Wert bedeutet, was einem „Durchschleifen“ des aktuellen Samples gleichkommt.

Eine Umformung der Vektoren von Matlab für einen SHARC Prozessor sieht wie folgt aus:

```
bzout = flipud(bz')          → Drehen von b
azout = -flipud(az');        → Negieren und Drehen von a
azout = azout(1:(end-1));    → Weglassen von a0
```

Deshalb ist der A Vektor immer um eins kürzer als der B-Vektor. Auf das Tiefpassbeispiel angewandt würden die Koeffizienten lauten:

$$b = \left(0{,}11\overline{1} \quad 0{,}11\overline{1}\right)$$
$$a = \left(0{,}77\overline{7}\right) \tag{7.16}$$

7.4 Die Theorie zum C – Filter

Als etwas umfangreicheres Beispiel realisieren wir nun einen digitalen C – Filter. Die Schaltung, welche dem Filter zugrunde liegt ist in Abb. 17 dargestellt:

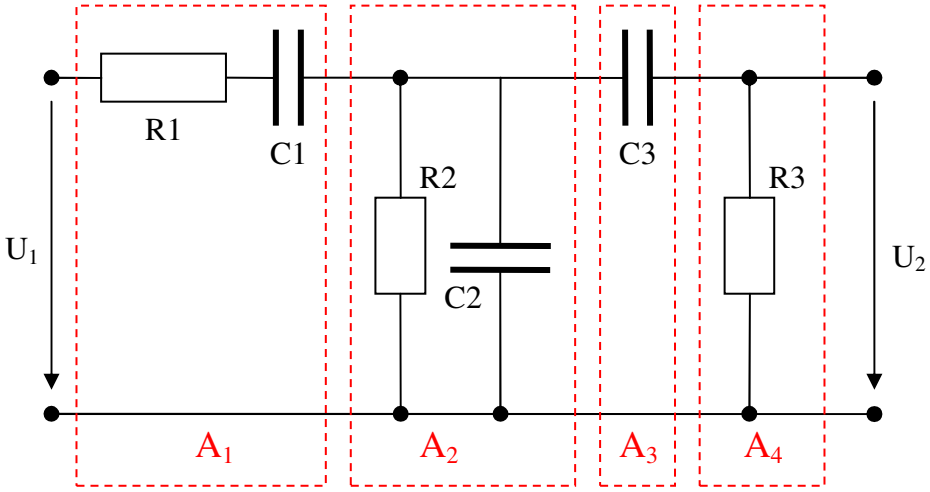


Abbildung 17 – analoges C – Filter mit Aufteilung in Teilmatrizen

Dieser C – Filter dient dazu, ein Eingangssignal dem Lautstärkeempfinden des menschlichen Ohres in einem bestimmten Schalldruckbereich anzupassen. Neben dem C-Filter gibt es noch drei weitere Gewichtungsfilter: A, B und D – Filter (siehe **Elliot_A_Filter.doc**).

Mit den Gleichungen (7.01) & (7.02) ergeben sich folgende Teilmatrizen:

$$A_1 = \begin{bmatrix} 1 & R_1 + \dfrac{1}{p \cdot C_1} \\ 0 & 1 \end{bmatrix} \qquad A_3 = \begin{bmatrix} 1 & \dfrac{1}{p \cdot C_3} \\ 0 & 1 \end{bmatrix} \tag{7.17}$$

$$A_2 = \begin{bmatrix} 1 & 0 \\ \dfrac{1}{R_2} + p \cdot C_2 & 1 \end{bmatrix} \qquad A_4 = \begin{bmatrix} 1 & 0 \\ \dfrac{1}{R_3} & 1 \end{bmatrix} \tag{7.18}$$

Die Dimensionierung des Filters (abgeleitet aus **Elliot_A_Filter.pdf**) ergibt sich damit zu:

R1 =	300 Ω;	R2 =	1800 Ω;	R3 =	100 kΩ;
C1 =	220 nF;	C2 =	27pF;	C3=	47pF;

Mit den Gleichungen (7.04) & (7.05) erhält man (gerundet):

$$G(p) = \frac{6.72 \cdot 10^{-28} \cdot p^2}{4.05 \cdot 10^{-33} \cdot p^3 + 6.48 \cdot 10^{-28} \cdot p^2 + 1.42 \cdot 10^{-24} \cdot p + 2.69 \cdot 10^{-22}} \tag{7.19}$$

Wie in Unterpunkt 7.3 bereits ausführlich geschildert, erhält man somit folgende Koeffizientenvektoren (aufbereitet für den Prozessor):

$$a = \begin{pmatrix} 0.58875460429282 \\ -2.17451327307369 \\ 2.58575679226162 \end{pmatrix} \qquad b = \begin{pmatrix} 0.19444177702997 \\ -0.15696844654287 \\ -0.26938843800415 \\ 0.23191510751706 \end{pmatrix} \tag{7.20}$$

Die Normübertragungsfunktion für einen C – Filter sind in den Vorschriften *IEC / CD 1672*, bzw. *IEC / DIN 651* festgelegt und mit der Übertragungsfunktion:

$$G(p) = \frac{4\Pi^2 \cdot 12200^2 \cdot p^2}{(p + 2\Pi \cdot 20{,}6)^2 \cdot (p + 2\Pi \cdot 12200)^2} \qquad (7.21)$$

nach **http://www.cross-spectrum.com/audio/weighting.html** beschrieben. Hier lauten die digitalen Frequenzen (mit **`bilinear()`** erhalten):

$$a = \begin{pmatrix} -0.01253882314727, \\ 0.24849607388778 \\ -1.45513587894717 \\ 2.21917291405280 \end{pmatrix} \qquad b = \begin{pmatrix} 0.19788712002639 \\ 0.00000000000000 \\ -0.39577424005279 \\ -0.00000000000000 \\ 0.19788712002639 \end{pmatrix} \qquad (7.22)$$

Mit den *Matlab* Programmen **C_Filter.m** und **C_Filter_digitalVGL.m** lassen sich die Ergebnisse sehr gut nachvollziehen. Abb. 18 zeigt den Verlauf zwischen analogem und digitalem Filter.

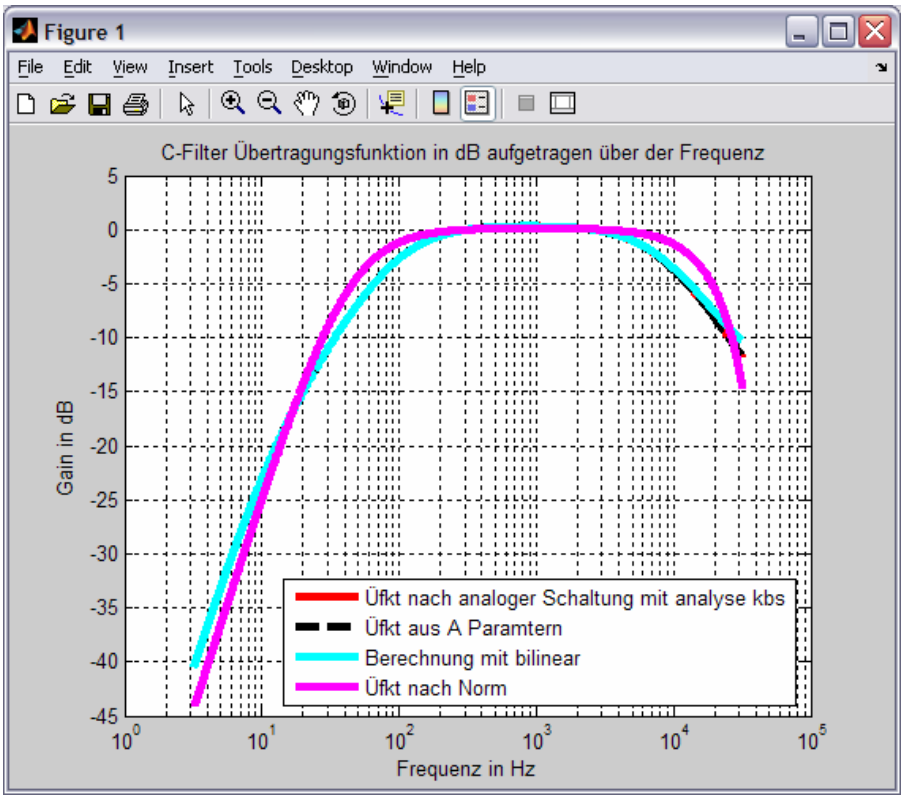


Abbildung 18 – Plot der Datei C_Filter.m (Vergleich zwischen Analog- & Digitalfilter)

Die Approximation der Schaltung (Abb. 17) ist über weite Bereiche der Norm entsprechend. Da die Schaltung jedoch lediglich aus einem Tief- und einem Hochpass besteht, sind die Abweichungen ab ca. 11 kHz im analogen Bereich etwas drastischer, als die Norm verlangt.
In Abb. 19 ist das Plotergebnis der Datei **C_Filter_digitalVGL.m** dargestellt. Hier werden nun die Unterschiede der verschiedenen Approximationen (siehe 7.3.3) deutlich. Auch hier wurde wieder die Normfunktion als Vergleich mit eingezeichnet. Aufgrund der besseren Approximation zur IEC – Vorgabe wurden die Koeffizienten der Funktion **`c2d()`** (first order hold) in das Programm implementier. Lediglich außerhalb des hörbaren Bereiches (f>18kHz) kommt es zu geringen Abweichungen. Im unteren Frequenzbereich stimmt sie hingegen ideal mit der analogen Übertragungsfunktion überein. (Der Unterschied zur Norm ist aus dem Filtergrad der Schaltung begründet). Laut *Matlab* entspricht die „foh“ Approximation der bilinearen Transformation. Deutlich zu erkennen ist auch die Wiederholung des Spektrums ab der Frequenz von 48 kHz, da diese Frequenz für die Simulation als Abtastfrequenz gewählt wurde. Die analoge Schaltung hingegen fällt weiterhin linear mit 20dB pro Dekade.

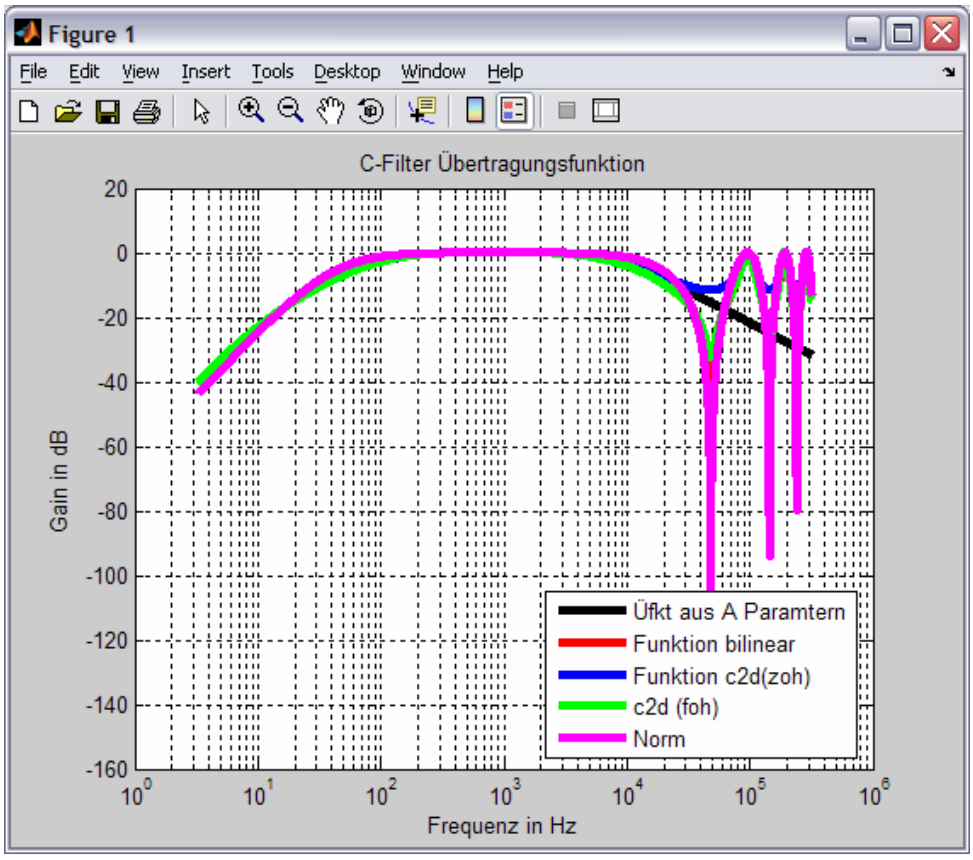


Abbildung 19 – Plot der Datei C_filter_digital.m (Vergleich. digitaler Approximation)

7.5 Analyse des C- Filters mittels Audioanalysators

In Abb. 20 wird die Analyse des C-Filters mit dem Audioanalysators *UDP* der Fa. *Rhode und Schwarz* dargestellt. Hierzu wurde mit Hilfe der Buchse J23 ein Testsignal mit folgenden Parametern eingespeist:

- Amplitude 1V effektiv (entspricht dem Normpegel für „Line In“)
- Frequenzen von 20 Hz bis 50 kHz,
- logarithmische Einteilung mit 100 Messungen pro Dekade.

Über die Buchse J22 wird das Ausgangssignal der Prozessorplatine ebenfalls an den Analysator angeschlossen. Der gemessene Spannungspegel wird in dB auf dem Display abgetragen (Referenzspannung 1V). Als Leitungen werden BNC Koaxialleiter mit entsprechenden Adaptern (auf 3.5 mm Klinkenstecker) verwendet. Das Ergebnis der Messung wird in Form von `.tcr` Dateien gespeichert und mit Hilfe von *Microsoft Excel* graphisch dargestellt.

Zunächst wird das Talkthrough – Modul (siehe 6.2.1) auf die Abtastfrequenz 11kHz eingestellt und das Eingangssignal durch den Prozessor „geschliffen“. Anschließend wird der Versuch mit einer Abtastfrequenz von 48kHz wiederholt. Durch den in Abb. 20 erhaltenen Frequenzgang wird somit der Frequenzgang der Entwicklungsplatine bestimmt. Wie in der Abbildung gut zu erkennen ist, wird das Signal ab der Hälfte der Abtastfrequenz sehr stark bedämpft. Weiterhin ist zu erkennen, dass die höhere Abtastfrequenz eine leichte Dämpfung (ca.2dB) des Signals bei Frequenzen bis ca. 100 Hz mit sich bringt.

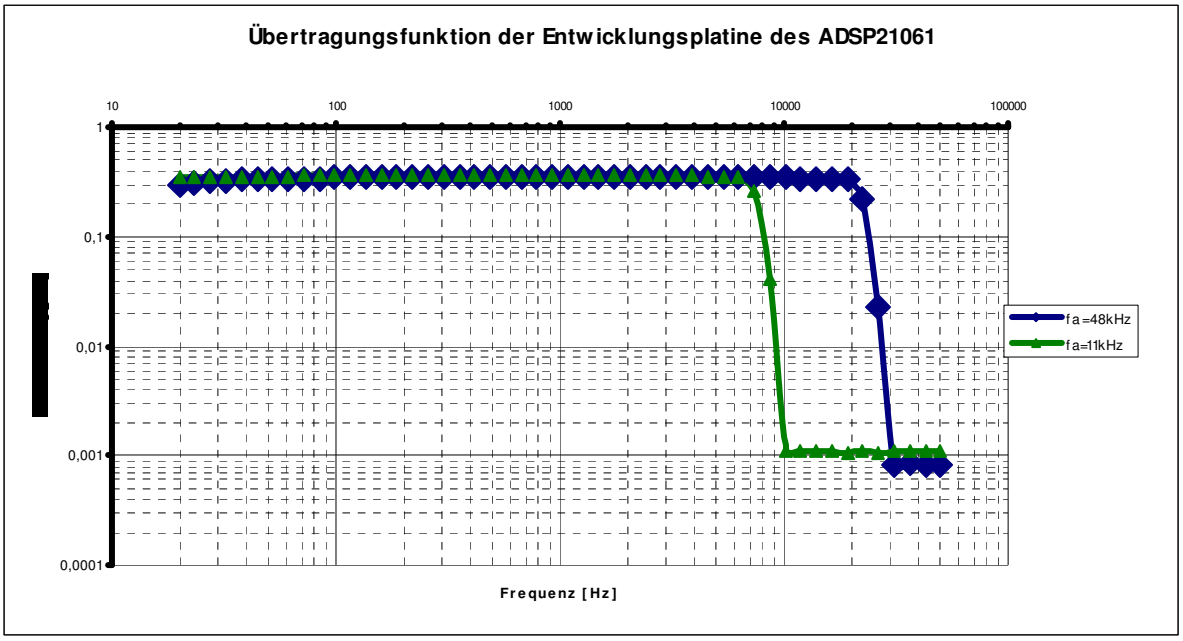


Abbildung 20 – Referenz - Übertragungsfunktion (ohne Filterberechnung)

Die Platine verhält sich also keineswegs neutral, sondern beeinflusst die in Abb. 21 dargestellten berechneten C – Filter.

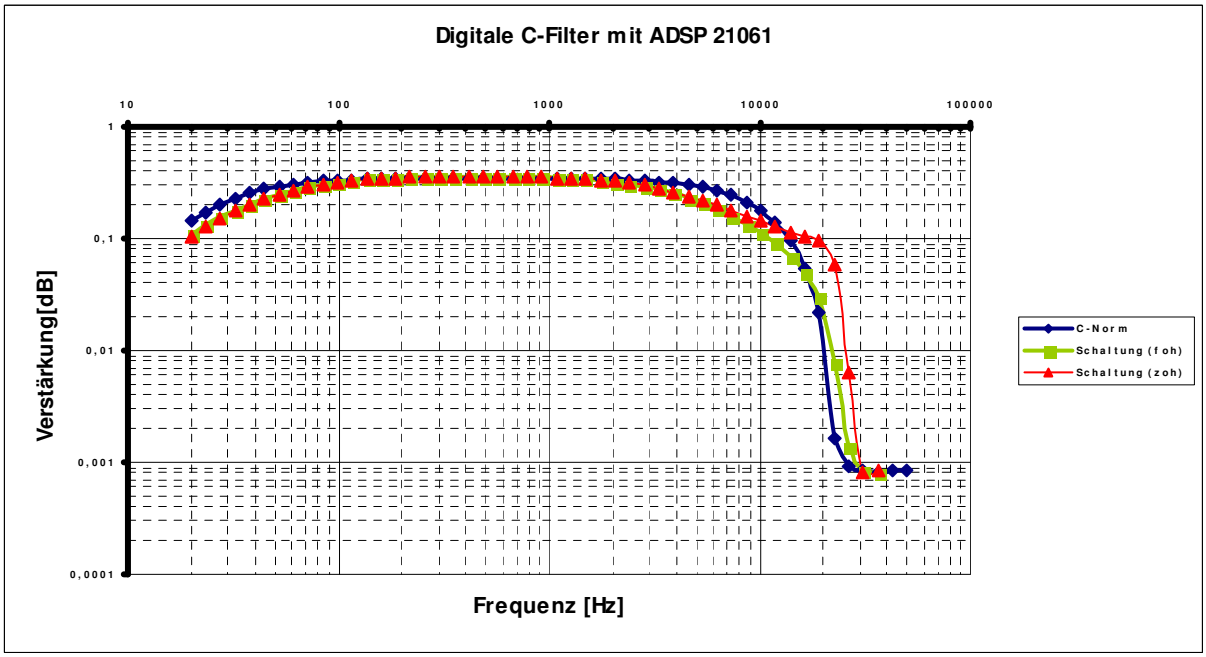


Abbildung 21 – gemessene Übertragungsfunktion (C - Filter)

Die Norm – C-Filter Messung (Kurve blau) erfolgt über Implementierung der Filterkoeffizienten nach Gleichung (7.22). Die Filterkoeffizienten aus (7.20) werden in einem zweiten Durchlauf ebenfalls in Abbildung 21 eingetragen (Kurve grün). Es ist, wie auch schon die

Simulation unter *MatLab* zeigt (Abb.19) eine leichte Abweichung im unteren und oberen Frequenzbereich ersichtlich, welche aus der Schlichtheit der Approximationsschaltung rührt. Ergänzend ist hier noch die Approximation „zero order hold“ eingezeichnet, welche ebenfalls das *MatLab* –Simulationsergebnis bestätigt (Kurve rot). Deutlich zu erkennen ist der radikale Abfall ab einer Frequenz von ca. 20 kHz. welche durch Übertragungsfunktion der Entwicklungsplatine zu erklären ist. Trotz der geringen Abweichungen im unteren und oberen Bereich sind die Übertragungsfunktionen nahe an der Normfunktion und können damit sicherlich als Gewichtungsfilter eingesetzt werden.

Als Ergänzung wurden ebenfalls die Butterworth – Filter (siehe 6.2.3) mit dem Analysator gemessen und in Abbildung 22 dargestellt. Die Welligkeit bis zu 100 Hz wird ebenfalls durch die Platinenfunktion verursacht.

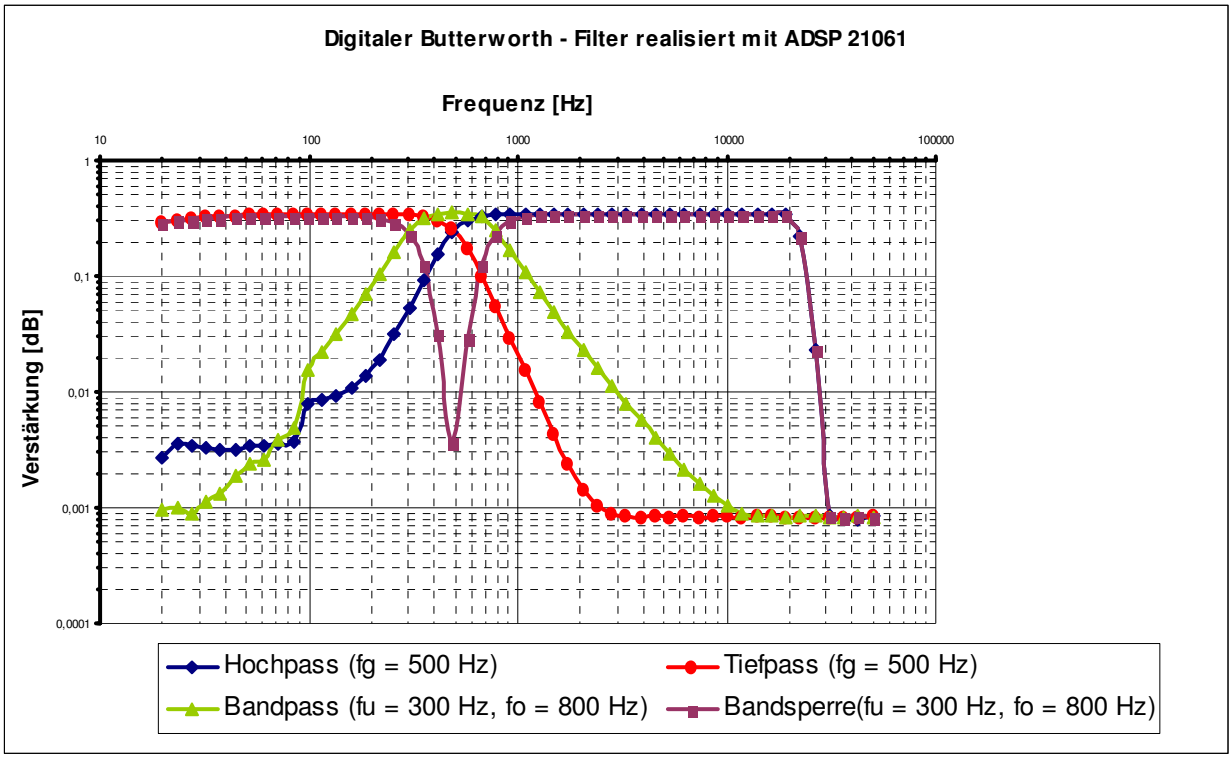


Abbildung 22 – gemessene Übertragungsfunktion (Butterworth – Filter)

Ansonsten sind auch diese Filter ihrer Simulation nach unter *MatLab* gleich. (Butterworth Filter → siehe **Matlab\butterworth.m**)

Für weiterführende Betrachtungen sind die Abbildungen 20 – 22 im Ordner „Analyse“ in der Datei **analyse09012006.xls** in einer besseren Auflösung zu finden.

8. Fazit und Ausblick

Mit Hilfe der Realisierung verschiedener Signalverarbeitungsalgorithmen in Form des Demonstrationsprogramms und dieser Dokumentation werden vor allem 2 Dinge erreicht:
Zum einen ist es nun möglich, einen kleinen Einblick in die digitale Signalverarbeitung zu erlangen. Dies bedeutet, dass es mit dem Studieren dieser Dokumentation möglich wird, den Weg von einer analogen Schaltung zu einem digitalen System aufzuzeigen mit Hilfe der hier dargestellten Anregungen in der Lage zu sein, tiefer in die Mathematik und Systemtheorie einzusteigen.
Der zweite Punkt, der hier erzielt wird, ist der praktische Umgang mit einem digitalen Signalprozessor. Sicherlich ersetzt diese Dokumentation in keiner Weise die Vorlesungen *„Signal- und Systemtheorie“*, *„Signalverarbeitung“* oder „*Spezielle Prozessoren (DSP)*“, jedoch ist es mit Hilfe der hier vorgestellten und programmierten Module möglich, eine Schnittstelle zwischen diesen beiden Bereichen zu erhalten. Mit den realisierten Algorithmen in Form der einzelnen Audio-Module des Demonstrationsprogramms kann nun Theorie in Praxis umgesetzt werden. Besonders hervorgehoben sei hier noch einmal die Funktion, mittels des Demonstrationsprogramms eigene errechnete Filterkoeffizienten einzugeben (siehe Abschnitt 6.2.3), das Resultat des Filters „live“ zu hören und mittels geeigneter Analysatoren näher untersuchen zu können (siehe Abschnitt 7.5). Für eventuelle Übungs- und Demonstrationszwecke enthält der Ordner „Matlab“ auf der beiliegenden CD außer dem Filtertyp „Butterworth“ noch die Filtertypen „Cauer“ und „Chebychev1“. Die Filterkoeffizienten dieser Berechnungen befinden sich im Anhang. Durch weiterführende Analyse könnte man diese Filtertypen ebenfalls auf ihre Vor – und Nachteile analysieren.

Im Moment sicherlich noch nachteilig ist der Umstand, das Demonstrationsprogramm stets via *VisualDSP++* in den Speicher laden zu müssen (siehe Abschnitt 2.6). Eine weitaus komfortablere Methode wäre das Brennen eines eigenen EPROM, welches unser Programm als Bootprogramm besitzt. Der immense zeitliche Mehraufwand bezüglich der Informationsbeschaffung über das Standard-Bootprogramm und der Entwicklung eines geeignetes Programms mit den Bootinitialisierungen (RS232 & SPORT – Konfigurationen, Speicherzugriff etc.), ermöglichte es jedoch nicht, dies im Rahmen dieses Komplexpraktikums zu realisieren, sollte aber ohne Zweifel ein Thema für nachfolgende Projekte bzw. Praktika sein.

Zusätzliche Erweiterungsmöglichkeiten bestehen ebenso in der Implementierung anderer Gewichtungsfilter (A, B und D - die dafür nötigen Filterkoeffizienten befinden sich ebenfalls im Anhang). Mit Hilfe des Moduls „Effektivwertberechnung“ und der vier Filter könnte weiterhin ein Schalldruckgerät entwickelt werden, welches sich ideal an das menschliche Hörverhalten anpasst. Anhand der RMS – Werte des angelegten Eingangssignals könnte der Prozessor selbstständig den zuständigen Filter wählen und das Ausgangssignal entsprechend formen.

Wir hoffen, mit dieser Dokumentation nicht nur die Grundlage für die hier angesprochenen Erweiterungsmöglichkeiten gelegt zu haben, sondern auch den allgemeinen Einstieg in die Signalverarbeitung mittels digitaler Signalprozessoren erheblich zu erleichtern.

9. Anhang

Die auf der CD befindlichen Programme für *MatLab* berechnen die unterschiedlichsten Koeffizienten. Hier eine kleine Auswahl.

- B – Filter:

$$a = \begin{pmatrix} 0.01228137868400 \\ -0.25593282823524 \\ 1.67375541091895 \\ -3.62874520437241 \\ 3.19864112568289 \end{pmatrix} \qquad b = \begin{pmatrix} -0.19830381262920 \\ 0.19830381262920 \\ 0.39660762525840 \\ -0.39660762525840 \\ -0.19830381262920 \\ 0.19830381262920 \end{pmatrix}$$

- D – Filter:

$$a = \begin{pmatrix} -0.90362132952962 \\ 3.70469962043607 \\ -5.69840423550622 \\ 3.89732536712193 \end{pmatrix} \qquad b = \begin{pmatrix} -0.14197990618294 \\ 0.28700351863463 \\ -0.00310835900484 \\ -0.28700351863463 \\ 0.14508826518779 \end{pmatrix}$$

- Chebychev1 – Tiefpass (f_g = 500 Hz):

$$a = \begin{pmatrix} -0.92462726420119 \\ 3.76694728475846 \\ -5.75974301163747 \\ 3.91741630040136 \end{pmatrix} \qquad b = \begin{pmatrix} 0.0000039477550 \\ 0.0000015791020 \\ 0.0000023686530 \\ 0.0000015791020 \\ 0.0000039477550 \end{pmatrix}$$

- Cauer - Tiefpass(f_g = 500 Hz):

$$a = \begin{pmatrix} -0.92466625057312 \\ 3.76705985749804 \\ -5.75985066477126 \\ 3.91745031678060 \end{pmatrix} \qquad b = \begin{pmatrix} 0.00001552698979 \\ -0.00003684816645 \\ 0.00004900632974 \\ -0.00003684816645 \\ 0.00001552698979 \end{pmatrix}$$

Äquivalent zu den Tiefpässen werden auch die Koeffizienten der Hoch- und Bandpässe, sowie der Bandsperren ausgegeben. Die Grenzfrequenzen sind (f_u = 300Hz, f_o = 800 Hz, f_g = 500 Hz)

10. Literatur

Die Anleitungen der Firma Analog Devices:

- ADSP-2106x SHARC Processor Users Manual, Revision 2.1
- VisualDSP++ 3.5 User's Guide for 16-Bit Processors
- Linker and Utilities Manual for 16-Bit Processors
- Loader Manual for 16-Bit Processors
- Component Software Engineering User's Guide for 16-Bit Processors
- C/C++ Compiler and Library Manual for SHARC® Processors
- SHARC EZ-KIT LITE Document Library

Websites :

- http://www.analog.com
- http://www.dsprelated.com
- http://www.cross-spectrum.com/audio/weighting.html
- http://www.sengpielaudio.com/Rechner-dba-spl.htm

Vorlesungen an der FH Jena:

- Spezielle Prozessoren
- Digitale Audiotechnik
- Analoger und Digitaler Filterentwurf
- Signal und Systemtheorie

11. Abbildungsverzeichnis