

---

# Mehrkanalige Störgeräuschunterdrückung

auf einem Signalprozessor

---

## Diplomarbeit

Adrian Eggenberger  
Mischa Jud

Betreuer: A. Schaub



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>7</b>
<b>2</b>	<b>Aufgabenstellung</b>	<b>9</b>
<b>3</b>	<b>Planung und Projektablauf</b>	<b>11</b>
3.1	Zeitplan . . . . .	11
3.2	Projektablauf . . . . .	12
<b>4</b>	<b>Beschreibung des DSP</b>	<b>13</b>
<b>5</b>	<b>Entwicklungsumgebung des DSP</b>	<b>15</b>
5.1	Comptool . . . . .	15
5.2	Versuchsaufbau . . . . .	17
<b>6</b>	<b>Umsetzung auf den DSP</b>	<b>19</b>
6.1	Programm dem DSP anpassen . . . . .	19
6.1.1	Arbeiten in Echtzeit mit dem DSP . . . . .	19
6.1.2	Bearbeitungsvorgang im DSP-Programm integrieren . . . . .	21
6.2	DMA . . . . .	22
6.3	Laufzeitverzögerung . . . . .	24
6.4	Bearbeitungszeit . . . . .	26
<b>7</b>	<b>DSP Steuerung</b>	<b>27</b>
7.1	Systemvoraussetzungen . . . . .	27
7.2	Installation . . . . .	27
7.3	Bedienung des Programms . . . . .	28

7.3.1	Steuerung des DSP-Programms . . . . .	28
7.3.2	Überwachung des Signalprozessors . . . . .	29
7.4	Aufbau des Programms . . . . .	33
7.4.1	Klassendiagramm . . . . .	33
7.5	Kommunikation zwischen dem DSP und dem PC . . . . .	35
<b>8</b>	<b>Optimierung mit Assembler</b>	<b>37</b>
<b>9</b>	<b>Festkommaarithmetik</b>	<b>39</b>
9.1	Warum Java? . . . . .	39
9.2	Programmumstellung . . . . .	39
9.3	Umgang mit Festkommazahlen . . . . .	40
9.3.1	Festkommazahl mit n Kommastellen . . . . .	40
9.3.2	Logarithmische Operationen . . . . .	42
9.4	Genauigkeit . . . . .	45
<b>10</b>	<b>Diskussion der Ergebnisse</b>	<b>47</b>
<b>11</b>	<b>Schlusswort</b>	<b>49</b>
<b>12</b>	<b>Abbildungsverzeichnis</b>	<b>51</b>
<b>13</b>	<b>Literaturverzeichnis</b>	<b>53</b>
<b>14</b>	<b>Glossar</b>	<b>55</b>
<b>A</b>	<b>Listing : DSP Programm</b>	<b>57</b>
A.1	Version 1.2 Stereo . . . . .	57
<b>B</b>	<b>Listing : DSP Steuerung</b>	<b>71</b>
B.1	Problem Domain . . . . .	71
B.1.1	DSPCmd.h . . . . .	71
B.1.2	DSPCmd.cpp . . . . .	72

B.2	Observer Pattern . . . . .	76
B.2.1	Observer.h . . . . .	76
B.2.2	Observer.cpp . . . . .	77
B.2.3	SpektrumView.h . . . . .	79
B.2.4	SpektrumView.cpp . . . . .	80
B.2.5	HuellkurvenView.h . . . . .	84
B.2.6	HuellkurvenView.cpp . . . . .	85
B.2.7	BaenderView.h . . . . .	89
B.2.8	BaenderView.cpp . . . . .	90
B.3	Windows Dialog . . . . .	94
B.3.1	DSP_ControlDlg.h . . . . .	94
B.3.2	DSP_ControlDlg.cpp . . . . .	96
B.3.3	DSP_Control.h . . . . .	103
B.3.4	DSP_Control.cpp . . . . .	104
<b>C</b>	<b>Listing : Assembler</b>	<b>105</b>
<b>D</b>	<b>Listing : Fixkomma</b>	<b>107</b>
D.1	Noiseerase.java . . . . .	107
D.2	WaveKlasse.java . . . . .	111
D.3	FensterKlasse.java . . . . .	116
D.4	FFTKlasse.java . . . . .	118
D.5	BaenderKlasse.java . . . . .	120
D.6	defs.java . . . . .	124
D.7	WaveHeader.java . . . . .	125
D.8	WerteImBandKlasse.java . . . . .	126
D.9	BandWerteKlasse.java . . . . .	127
D.10	IntegerOps.java . . . . .	128
D.11	FixpointOps.java . . . . .	129

***über dieses Dokument***

Ausgabe 1.0 vom 7. Dezember 1999

Diese Dokument wurde mit  $\text{\LaTeX}$  geschrieben.

Es wurde der MikTeX 1.20 Compiler verwendet.

# 1 Einleitung

---

Auf Grund unserer positiven Erfahrung während der zweiten Semesterarbeit haben wir uns entschlossen, jenes Thema weiter zu verfolgen und als Diplomarbeit zu vertiefen. Unsere zweite Semesterarbeit beinhaltete die Entwicklung einer Störgeräuschunterdrückung auf dem PC. Sie sollte in Hörgeräten und Hörhilfen eingesetzt werden. Hierzu müsste das Programm aber auf einem portablen Prozessor ablaufen. Für solche Aufgaben eignen sich DSPs (Digitale Signalprozessoren) speziell gut.

Hauptbestandteil der Diplomarbeit war die Umsetzung der Störgeräuschunterdrückung auf den DSP Sharc 21060 von Analog Device. Zusätzlich wurde ein PC-Programm entwickelt, um den Ablauf des DSP-Programms zu überwachen. Ferner wurden die Problematik der Festkommaarithmetik untersucht. Hierzu programmierten wir unseren Algorithmus mit Festkommazahlen. Ein weiteres Thema war die Optimierung des Programmcodes in Assembler. Dieses Thema haben wir anhand eines kleinen Beispielprogramms studiert. Da diese Arbeit auf der Semesterarbeit aufbaut, ist das Studium der dazugehörigen Dokumentation [3] für das Verständnis dieser Arbeit Voraussetzung.

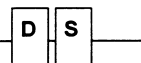


---

## 2 Aufgabenstellung

---

ITR – DIGITALE SIGNALVERARBEITUNG

**D-99-01****Diplomarbeit** für Herrn A. Eggenberger und Herrn M. Jud**Mehrkanalige Störgeräuschunterdrückung**

Aufbauend auf die Resultate der vorangegangenen Studienarbeit soll in dieser Diplomarbeit eine Implementation der mehrkanaligen Störgeräuschunterdrückung auf einer DSP-Karte erfolgen.

In einem ersten Schritt kann das vorhandene Programm mit relativ geringfügigen Anpassungen und mittels eines C-Compilers auf einem Signalprozessor mit Fließkomma-Arithmetik realisiert werden.

Für die Praxis ist eine Realisierung mit Festkomma-Arithmetik jedoch aus verschiedenen Gründen nach wie vor vorteilhaft. Für eine solche Implementation ist das bestehende Simulationsprogramm zunächst zu überarbeiten, und zwar im Hinblick auf die Verwendung von Festkommazahlen, in der Regel 16 Bit breite so genannte Short-Zahlen. Gewisse Größen erfordern gelegentlich auch den Einsatz von 32 Bit breiten Integer-Zahlen.

Eine weitere Schwierigkeit bietet die Behandlung mathematischer Funktionen, wie z.B. die Logarithmusfunktion zur Berechnung von Signalleistungen auf einer dB-Skala. Zu diesem Zweck ist oft der Einsatz von Tabellen sinnvoll.

Alle diese Probleme lassen sich jedoch am besten anhand eines in einer höheren Programmiersprache wie C/C++ oder Java geschriebenen Simulationsprogramms lösen und ausführlich erproben.

Für die Realisierung auf dem Signalprozessor kann auch im Fall der Festkomma-Arithmetik zunächst wieder auf einen C-Compiler zurückgegriffen werden. Wie aus der Erfahrung bekannt ist, ergeben sich dabei aber stets Programme, die wesentlich mehr Ressourcen erfordern, als von der Aufgabe her eigentlich notwendig wäre.

In einem letzten Optimierungsschritt kommt man deshalb nicht darum herum, mindestens die wesentlichen Ausführungsteile des Algorithmus selbst in Assembler auszuformulieren.

**Aufgabe**

Die Diplomarbeit lässt sich in folgende Teilaufgaben gliedern:

- Wahl einer geeigneten DSP-Karte und Einarbeitung in die Entwicklungsumgebung mittels einfacher einführender Beispiele.
- Implementation einer Variante mit Fließkomma-Arithmetik (optional). Der Vorteil beim Realisieren einer solchen Variante liegt darin, dass Sie die Akquisition und Wiedergabe der Audiosignale, die Kommunikation zwischen PC und DSP-Karte sowie die Benutzeroberfläche auf dem PC schon frühzeitig und separat von der weiteren Programmentwicklung auf realistische Weise mit der tatsächlichen DSP-Applikation erarbeiten können.
- Überarbeitung des bestehenden Simulationsprogramms im Hinblick auf Festkomma-Arithmetik.

**ITR – DIGITALE SIGNALVERARBEITUNG**

- Implementation des überarbeiteten Simulationsprogramms, optional mit Übersetzung durch C-Compiler oder allenfalls schon mit Programmierung in Assemblersprache von Beginn an.
- Optimierung der Implementation mit Festkomma-Arithmetik durch Einfügen von Anweisungen in Assemblersprache, wo es sinnvoll erscheint und durch den vorhergehenden Aufgabenteil nicht schon vorweggenommen wurde.
- Erstellen einer übersichtlichen Dokumentation im Bericht zur Diplomarbeit, mit Schwergewicht auf den eigenen Überlegungen und Arbeitsergebnissen.

**Literatur**

- [1] Brent W. Edwards, Zezhang Hou, Christopher J. Struck, and Priya Dharan: *Signal-processing algorithms for a new software-based, digital hearing device*, The Hearing Journal, September 1998, Vol. 51, No. 9.
- [2] ANSI S3.5-1997: *Methods for Calculation of the Speech Intelligibility Index*, American National Standard Institute, 1997.

**Bericht**

Über die Arbeit ist ein Bericht zu verfassen. Dabei sind die Richtlinien der Abteilung für Elektrotechnik für das Erstellen von Berichten einzuhalten. Alle verwendeten Quellen sind im Literaturverzeichnis des Berichts anzugeben (Hinweis im Text). Der Bericht ist im Doppel abzugeben; ein Exemplar verbleibt am Labor für Digitale Signalverarbeitung des ITR. Die erstellten Programme und der Text des Berichts sind auf einer beschrifteten Diskette (mit Nummer der Studienarbeit) beizulegen.

**Termine und Bedingungen**

Gemäss Vorgaben und Terminplan des Vorstandes der Abteilung für Elektrotechnik

Ausgabe der Aufgabenstellung:	18.10.1999
Abgabe der Zusammenfassung der Diplomarbeit	7.12.1999, 17.00 Uhr
Abgabe des Berichts zur Diplomarbeit:	7.12.1999, 17.00 Uhr

Arbeitsplatz: DS-Labor 1219 (zuständig: P. Roffler, Laborassistent)

**Assistenz**

Th. Hugentobler

**Kontaktadresse**

Bernafon AG  
A. Schaub  
Eichtalstr. 55  
8634 Hombrechtikon  
Tel. 055 - 264 13 51  
Fax. 055 - 264 13 54

Rapperswil, 11. Oktober 1999

A. Schaub

### 3 Planung und Projektlauf

#### 3.1 Zeitplan

Während der ersten Woche der Diplomarbeit versuchten wir einen straffen Zeitplan zu entwerfen. Speziell der Dokumentation wollten wir in der Anfangsphase der Arbeit mehr Zeit widmen.

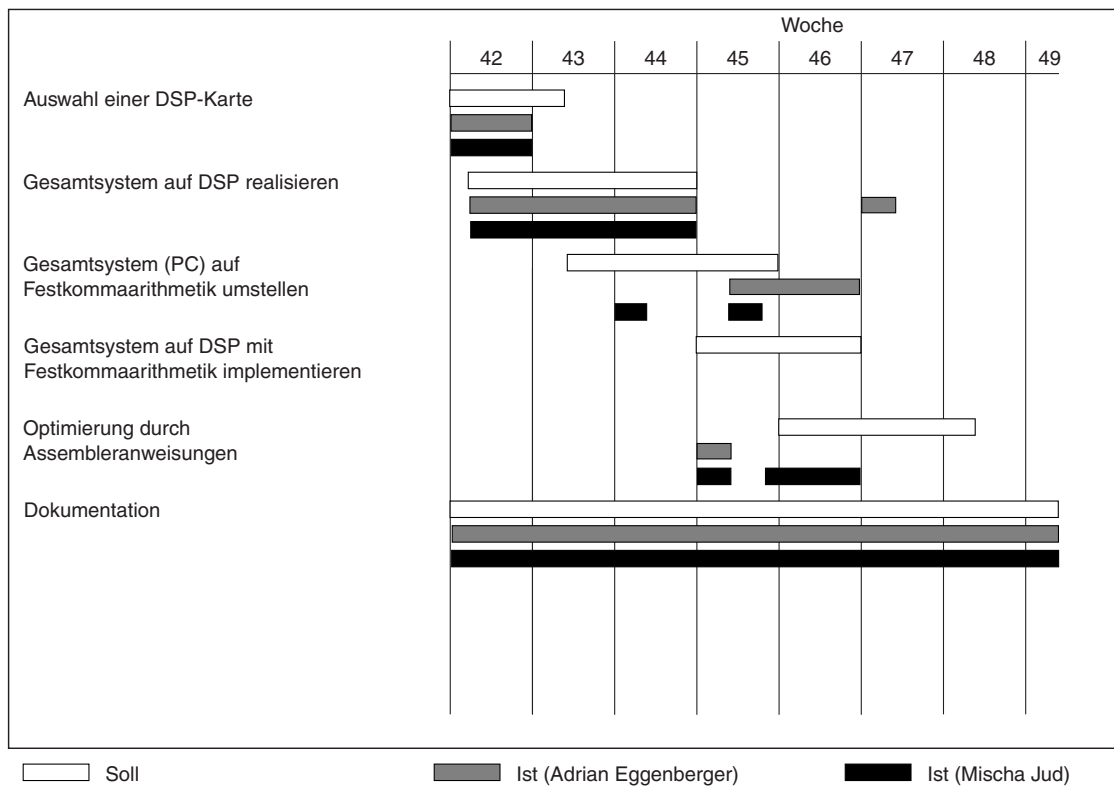


Abbildung 3.1: Zeitplan

Auf eine Umsetzung der Festkommaversion auf den Signalprozessor wurde nach Rücksprache mit Herrn A. Schaub verzichtet. Grund für diese Änderung ist, dass die DSP-Architektur des Sharc auf Fließkomma ausgelegt ist. Eine Implementierung mit Festkommazahlen ist daher nicht zweckmässig.

## 3.2 Projektablauf

Wir waren die ersten Studenten, die das eben neu erbaute DS-Labor beziehen durften. Alle notwendigen Arbeitsmittel standen uns zur Verfügung. Zu Beginn stand die Auswahl des Signalprozessors im Vordergrund. Die Wahl fiel uns nicht schwer, denn der Sharc ADSP-21060, welcher zum Laborbestand gehört, bietet den neusten Stand der Technik. Diesen Signalprozessor hatten wir zudem bereits in unserem DS-Vertiefungspraktikum des zweiten Studienjahres kennengelernt. Dies vereinfachte uns den Einstieg in die DSP-Programmierung. Dank dem Matlab-Programm "Comptool" war es uns nach kurzer Zeit möglich, erste Testprogramme auf dem Sharc zu realisieren. Das Programm "Comptool" stellt eine komfortable Bedienung des Compilers zur Verfügung und übermittelt den Maschinencode an den Signalprozessor.

Nach dem kurzen Kennenlernen des Signalprozessors und der Entwicklungsumgebung haben wir uns entschlossen die Arbeit aufzuteilen. Auf dem Signalprozessor wurde der Datenaustausch über DMA realisiert. Weiter haben wir Variablen zur Verfügung gestellt, über die unser Steuerprogramm später auf Zwischenergebnisse zugreifen kann. Parallel dazu schrieben wir das C++ Programm "NoiseErase" in ein C Programm um. Anschliessend konnten diese beiden Elemente mühelos zusammengefügt werden. Die ersten Testläufe waren deshalb schon frühzeitig möglich.

Da wir vor dem Zeitplan lagen, konnten wir eine vielfältige grafische Oberfläche zur Steuerung des DSP-Programms erstellen. Sie beinhaltet auch eine Darstellung der Zwischenergebnisse, welche die Funktionsweise des Bearbeitungsvorganges zusätzlich illustriert.

Um die Problematik von Festkommaprozessoren genauer kennenzulernen haben wir unser Programm in eine Festkommaversion umgeschrieben. Hierfür kam die Programmiersprache Java zum Einsatz.

Signalprozessoren werden nahe an der Hardware programmiert, um möglichst kurze Rechenzeiten zu erreichen. Assembler eignet sich hierfür besonders gut. Da die Prozessorleistung für unser Programm ausreichte, mussten wir jedoch keine Assembleroptimierung vornehmen. Stattdessen haben wir die Optimierung mit Assembler anhand eines kleinen Beispiels studiert.

Zwischen den erwähnten Arbeitsgängen wurde die Dokumentation aktualisiert.

## **4 Beschreibung des DSP**

---

Der ADSP-21060 SHARC (Super Harvard ARchitecture Computer) ist ein Hochleistungs 32 Bit Signalprozessor für Sprach-, Musik-, Grafik- und Bildbearbeitung. Der Prozessor besitzt ein Dual Port SRAM und integrierte Ein- und Ausgabeeinheiten. Dies ermöglicht das unabhängige Lesen und Beschreiben des Speichers durch verschiedene Zugreifer. Mit seinem Instruktionsspeicher kann der Prozessor jede Instruktion in nur einem Taktzyklus ausführen. Er ist fähig, bis zu 40 MIPS auszuführen. Der Sharc kann in einem Zyklus zwei Speicherstellen lesen, sofern diese in verschiedenen Memory Spaces liegen (Program Memory pm, Daten Memory dm). Die richtige Verteilung der Variablen in die verschiedenen Memory Spaces kann eine Geschwindigkeitszunahme bewirken. Der ADSP-21060 Sharc kombiniert ein Fließkommarechenwerk mit einem DMA-Kontroller, serielle Ports und Multiprozessorfähigkeit.



## 5 Entwicklungsumgebung des DSP

---

### 5.1 Comptool

Das Programm "Comptool" stellt eine komfortable Bedienung des Compilers zur Verfügung. Die Funktionsweise von Comptool ist anschliessend durch ein Blockdiagramm illustriert. In der Abbildung 5.2 ist die Bedienoberfläche zu sehen.

Zuerst muss der Pfad des zu compilierenden C-Files angegeben werden. Nun stellt man die gewünschte Compilerfunktion ein. Meistens benutzen wir die Funktion, welche aus dem C-File direkt einen ausführbaren Maschinencode erzeugt. Anschliessend wählt man das verwendete DSP-Board aus. In unserem Fall war es das "Blacktip PCI" Board. Danach folgt die Entscheidung, ob und welche Compileroptimierung man einschaltet. Wir haben gute Erfahrungen mit der Optimierungsstufe "02" gemacht. Wird nun die Schaltfläche "File erzeugen" gedrückt, wird der Code kompiliert, an das DSP-Board übermittelt und darauf der Start des Signalprozessors ausgelöst.

Um den Assemblercode optimieren zu können, hatten wir die Funktion "Assembler File aus C-File" eingeschaltet. Der so generierte Code liess sich nun optimieren. Danach generierten wir mit Hilfe der Funktion "Ausführbares File aus Assembler File" den Maschinencode.

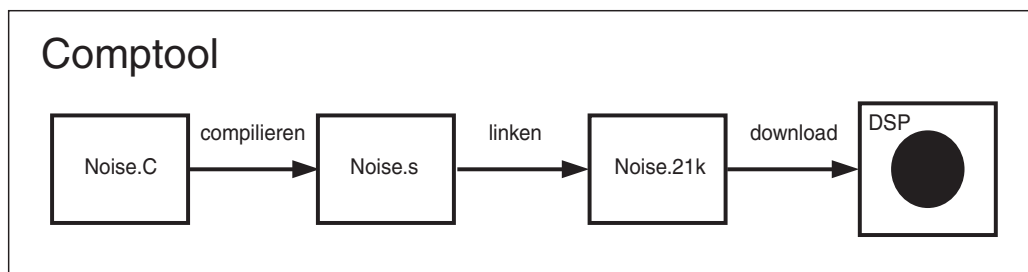


Abbildung 5.1: Arbeiten mit Comptool

5. ENTWICKLUNGSUMGEBUNG DES DSP Mehrkanalige Störgeräuschunterdrückung

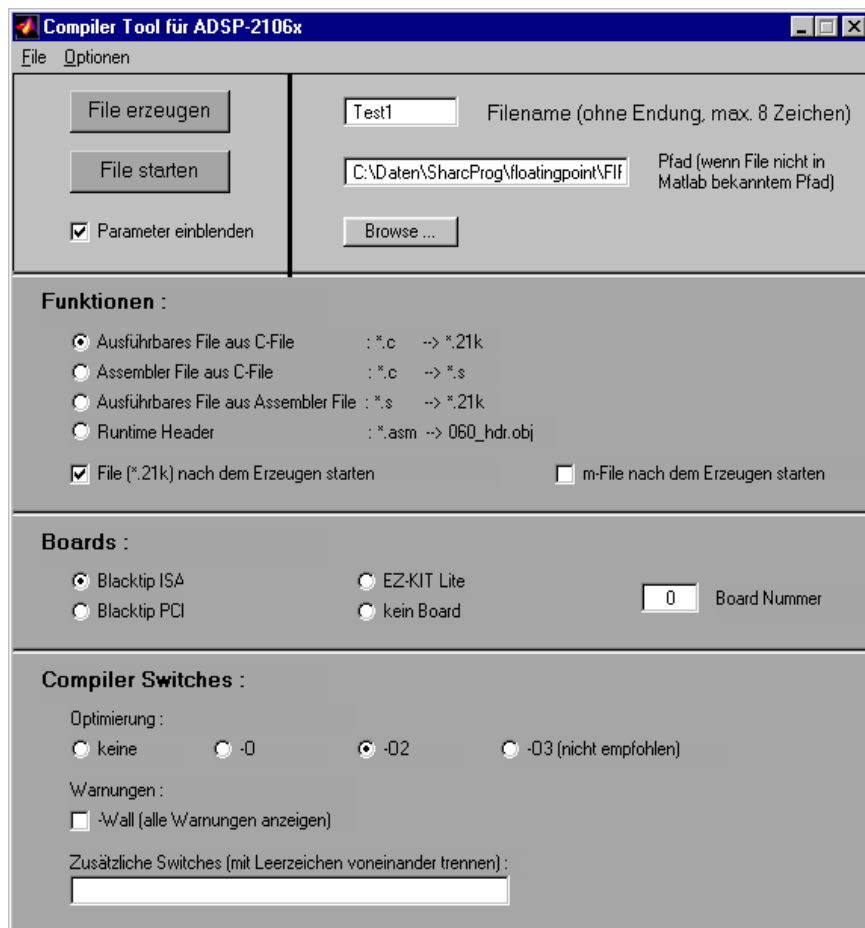


Abbildung 5.2: Bedienoberfläche von Comptool

## 5.2 Versuchsaufbau

Die Abbildung 5.3 zeigt den Versuchsaufbau unserer Arbeit. Das DSP-Board steckt in einem PCI-Slot im PC. Es stellt zwei analoge Ein- und Ausgänge zur Verfügung. An den Eingängen können verstärkte Mikrofone oder auch andere Signalquellen angeschlossen werden. Die Ausgänge sind mit einem Verstärker verbunden, welcher die Signale an die Lautsprecher weitergibt. Zur Überwachung wurden die Ein- und Ausgangssignale auf einem Oszilloskop dargestellt.

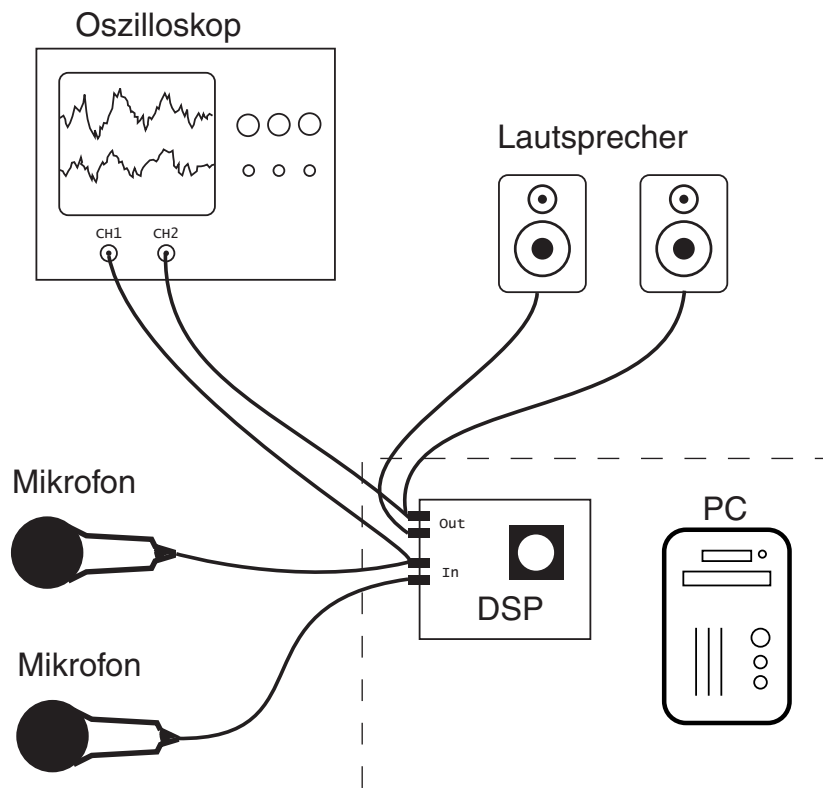


Abbildung 5.3: Versuchsaufbau

5. *ENTWICKLUNGSUMGEBUNG DES DSP Mehrkanalige Störgeräuschunterdrückung*

## 6 Umsetzung auf den DSP

---

In diesem Kapitel gehen wir zunächst auf den genauen Ablauf unseres Programms ein. Es wird erklärt, wie die Ein- und Ausgabe der Signaldaten vor sich geht. Der dazu verwendete Mechanismus (DMA-Transfer) wird im Kapitel 6.2 erläutert. Des Weiteren wird das Vorgehen beim Umsetzen des Programms auf den Signalprozessor beschrieben. Zudem kommen wir auf die Laufzeitverzögerung zu sprechen. Ihr muss bei Echtzeitanwendungen spezielle Beachtung geschenkt werden. Zum Schluss des Kapitels wird die Bearbeitungszeit der einzelnen Programmteile betrachtet.

### 6.1 Programm dem DSP anpassen

Die Aufgabe, unser Programm aus der Semesterarbeit [3] auf einem Signalprozessor in Echtzeit einzusetzen, konnte in zwei Teilaufgaben aufgeteilt werden. Als erstes entwickelten wir ein DSP-Programm, das Signaldaten in Echtzeit einlesen und wieder ausgeben kann. Dann integrierten wir den Code des Bearbeitungsvorgangs in das entstandene DSP-Programm.

#### 6.1.1 Arbeiten in Echtzeit mit dem DSP

Der Signalprozessor bietet mehrere Arten, Daten einzulesen bzw. auszugeben. Für unsere Anwendung eignet sich der Einsatz von DMA, um ganze Datenblöcke einzulesen oder auszugeben (siehe Kapitel 6.2).

Da in unserem Programm jeweils zwei Einleseblöcke für die Bearbeitung zusammengehängt werden, muss gleichzeitig auf zwei DMA-Buffer zugegriffen werden können. Zusätzlich soll während der Bearbeitung ein weiterer Block durch DMA beschrieben werden. Somit sind also drei DMA-Eingangsbuffer notwendig. Die Ausgabe der Signale wird ebenfalls über drei DMA-Buffer abgewickelt.

Die Abbildung 6.1 zeigt den genauen Aufbau des Einlesens sowie des Ausgebens von Signaldaten. Die DMA-Eingangsbuffer haben eine Breite von 32 Bit. In diesen Buffer werden die beiden 16 Bit breiten Eingangskanäle abgelegt. Der Kanal 1 (CH 1) kommt in den oberen 16 Bits zu liegen, der Kanal 2 (CH 2) in den unteren. Die drei DMA-Eingangsbuffer werden der Reihe nach mit den Eingangssignalen gefüllt. Ein DMA-Buffer bietet Platz für 256 Werte. Sobald ein DMA-Eingangsbuffer gefüllt ist, wird in den nächsten Buffer geschrieben und per Interrupt die Bearbeitung der Signaldaten ermöglicht. Auf der Abbildung 6.1 sind zwei solche zeitlich aufeinanderfolgende Abläufe illustriert. Der erste ist mit grauen, der nachfolgende mit schwarzen Pfeilen dargestellt. Der Ablauf wird an Kanal 1 gezeigt. Der Kanal 2 wird analog zu Kanal 1 behandelt.

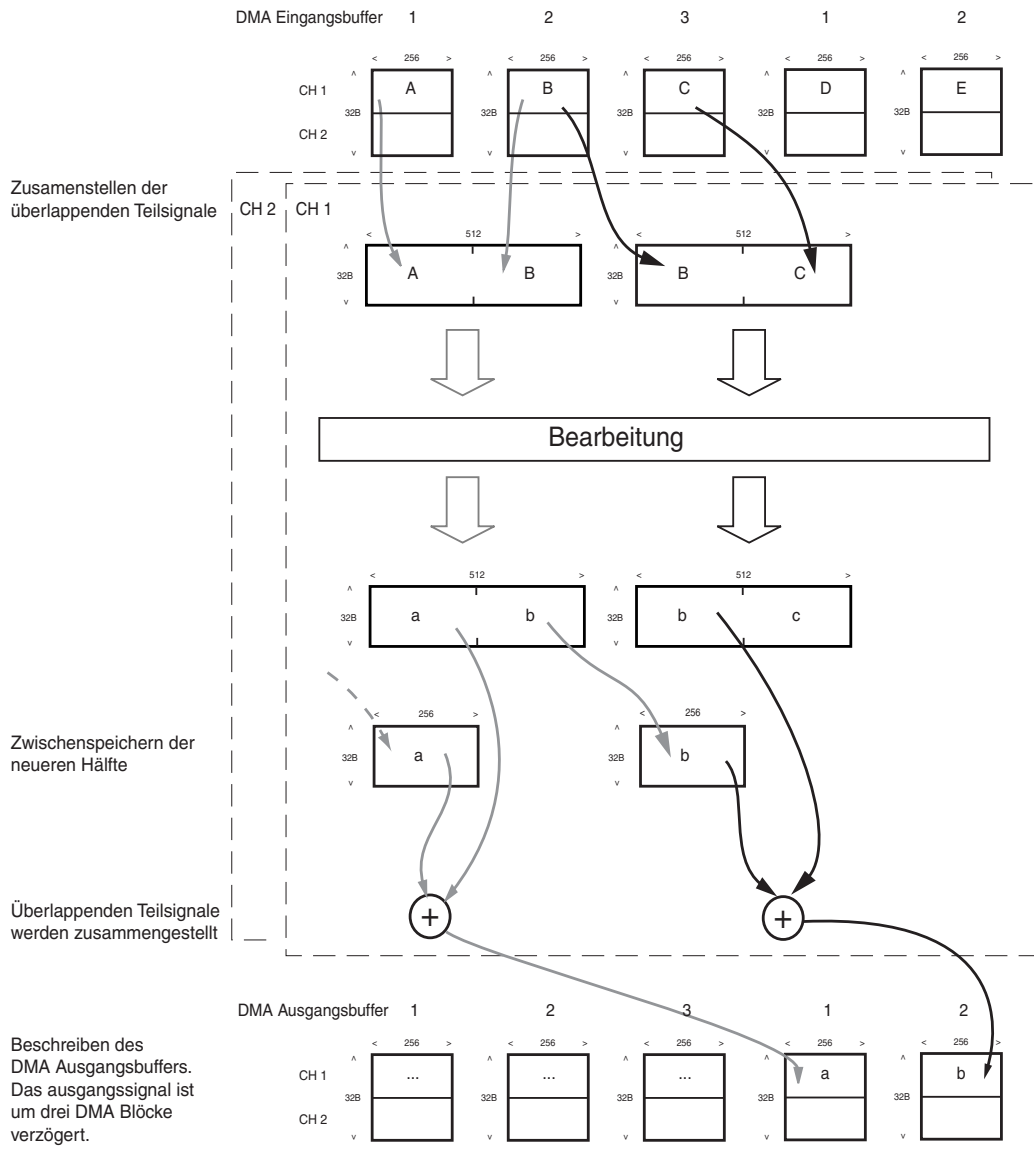


Abbildung 6.1: Ablauf der Ein- und Ausgabe

In einem ersten Schritt werden die 16 Bit breiten Abtastwerte des DMA-Eingangsbuffers in eine 32 Bit breite Fließkommazahl gewandelt. Dies erhöht die Genauigkeit der anschließenden Berechnungen. Die beiden DMA-Eingangsbuffer, welche nicht vom Signalprozessor mit neuen Werten gefüllt werden, können nun zusammen als Block mit 512 Werten bearbeitet werden.

Nach der Bearbeitung wird die neuere Hälfte des Blocks zwischengespeichert. Diese Hälfte wird im nachfolgenden Ablauf für die Überlappung benötigt. Sie erfolgt indem der Inhalt des Zwischenspeichers und der Inhalt der älteren Hälfte des bearbeiteten Blocks addiert werden. Das Resultat wird nun wieder in ein 16 Bit Wert gewandelt und im DMA-Ausgangsbuffer gespeichert.

Der vollständige Bearbeitungsablauf hat eine Zeitverzögerung von drei Einlesezyklen zur Folge (siehe auch Kapitel 6.3).

### **6.1.2 Bearbeitungsvorgang im DSP-Programm integrieren**

Die Implementierung des Bearbeitungsvorgangs lag uns aus der Semesterarbeit in der Form von C++ Code vor. Um den Code auf dem Signalprozessor einsetzen zu können, musste er auf C Code portiert werden. Da das DSP-Programm die Daten in der vom Bearbeitungsvorgang benötigten Form liefert, waren keine weiteren schwerwiegenden Anpassungen notwendig.

Probleme ergaben sich im Zeitablauf, da der Bearbeitungsvorgang länger dauerte als das Einlesen eines DMA-Blocks. Grund hierfür war unsere Implementation der FFT (sowie der IFFT) welche zuviel Rechenzeit beanspruchte. Deshalb ersetzten wir unsere Implementationen durch die Funktionen der "C Runtime Library", welche im Handbuch [5] beschrieben sind. Die Bearbeitung konnte nun in der vorgegebenen Zeit durchgeführt werden.

## 6.2 DMA

Direct Memory Access (DMA) stellt einen Mechanismus zum Transfer von ganzen Datenblöcken zur Verfügung. Der im ADSP-21060 integrierte DMA-Kontroller verschiebt Daten zwischen dem internen Speicher und externen Datenquellen. Er befreit damit den Kernprozessor von dieser Last. Der DMA-Kontroller erlaubt dem Kernprozessor einen Transferauftrag aufzugeben und danach mit den normalen Berechnungen fortzufahren. Der DMA-Kontroller überträgt währenddessen die Daten, unabhängig und unsichtbar für den Kernprozessor. In unserem Bearbeitungsvorgang wird ein Block mit den Signalwerten des Analog/Digital-Wandlers gefüllt. Sobald der Block voll ist, wird ein Interrupt ausgelöst. Die aufgerufene Interruptroutine wird dann durch den Kernprozessor ausgeführt. Gleichzeitig wird der nächste Block gefüllt.

Die Eigenschaften des DMA werden im Transfer Control Block (TCB) festgelegt. Im TCB wird beschrieben wie die einzelnen Datenblöcke miteinander verkettet sind. Die Abbildung 6.2 zeigt den Aufbau eines DMA mit drei Blöcken. Die drei Blöcke werden über Zeiger miteinander verkettet. Der Chain Pointer (cp) im ersten Feld des TCB zeigt auf den Eintrag "ii" im zweiten Feld. Der "cp" im zweiten Feld zeigt auf den Eintrag "ii" im Dritten Feld. Und der "cp" im dritten Feld zeigt wiederum auf den Eintrag "ii" im ersten Feld. Auf diese Weise wird der verkettete Ablauf festgelegt. Der Eintrag "c" im TCB bestimmt die Anzahl Werte, die in den entsprechenden Buffer eingelesen werden. Der Eintrag "im" bestimmt die Schrittweite des Bufferzeigers beim Inkrementieren. Der Eintrag "ii" zeigt auf die Startadresse des jeweiligen Buffers.

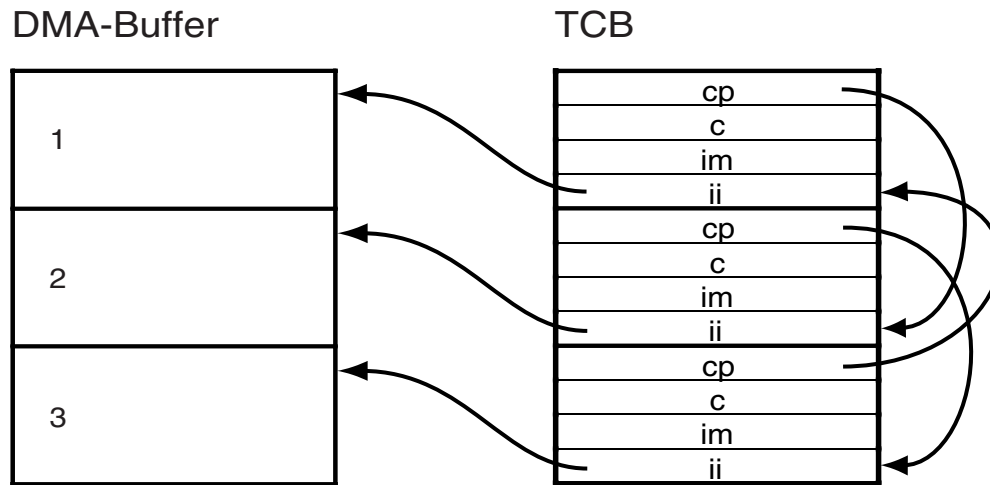


Abbildung 6.2: DMA Struktur

- cp Startadresse des Buffers (Chain Pointer)
- c Anzahl Wörter im Buffer (Count)
- im Schrittweite (Internal modifier)
- ii Startadresse des Datenbuffers (Internal Index)

### 6.3 Laufzeitverzögerung

Ein wichtiges Merkmal von Echtzeitanwendungen ist die Laufzeitverzögerung. Sie zeigt auf, um wieviel das Ausgangssignal im Bezug zum Eingangssignal verzögert ist.

Unser Programm liest die Signaldaten blockweise ein. Mit direktem Speicherzugriff (DMA-Transfer) werden die Daten in einen der drei DMA-Eingangsbuffer geschrieben. Sobald dieser Transfer abgeschlossen ist, wird via Interrupt die Verarbeitungsroutine aufgerufen. Während der nächste halbe Bearbeitungsblock vom Signalprozessor eingelesen wird, können die letzten zwei halben Bearbeitungsblöcke verarbeitet werden. Zusätzlich muss in dieser Zeit der Ausgangsbuffer mit den nächsten auszugebenden Signaldaten gefüllt werden. In der Abbildung 6.3 sind die Teilsignale und deren Zusammensetzung sowie der zeitliche Ablauf der Verarbeitung illustriert.

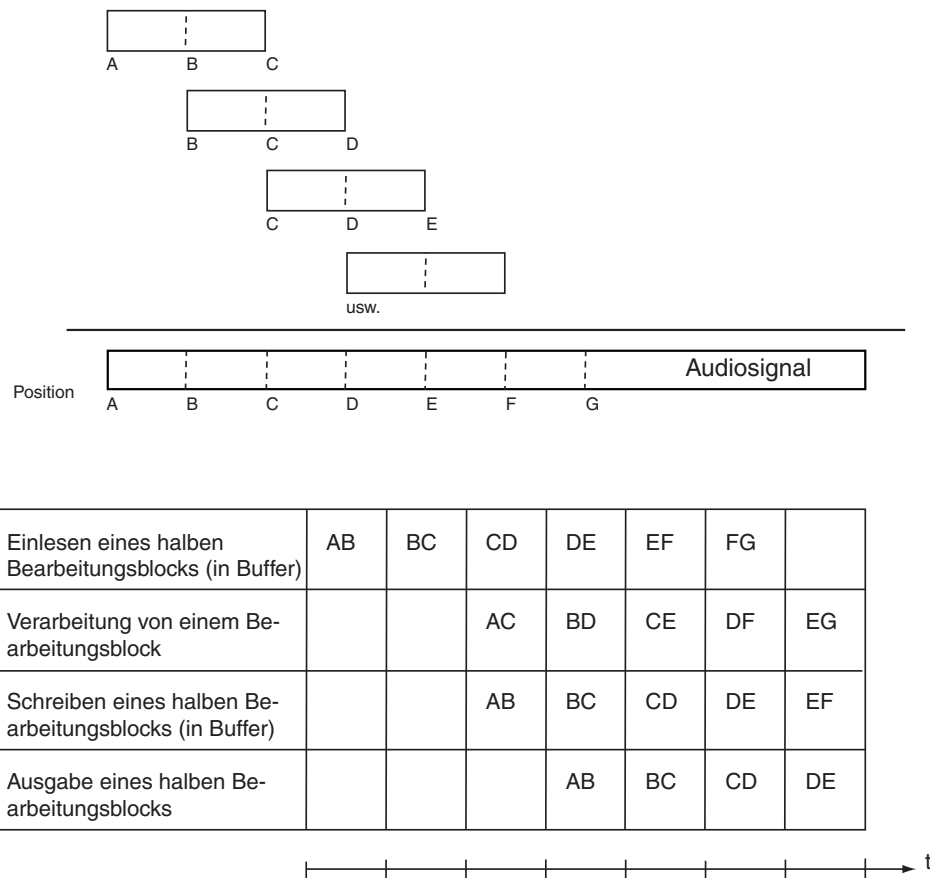


Abbildung 6.3: Laufzeitverzögerung

Aus der Abbildung 6.3 kann entnommen werden, dass die Verzögerung drei halben Bearbeitungsblöcken entspricht. Die Zeitdauer eines solchen Halbblocks ist von der Anzahl Werte eines Blocks und der Abtastfrequenz abhängig.

$$t_{hb} = \frac{1}{f_c} * \frac{N}{2} \quad (6.1)$$

$$t_v = 3 * t_{hb} \quad (6.2)$$

- $t_{hb}$  Zeit zum Einlesen eines halben Bearbeitungsblocks
- $t_v$  Laufzeitverzögerung
- $f_c$  Abtastfrequenz
- N Anzahl Werte in einem Bearbeitungsblock

Bei der von uns benutzten Länge der Bearbeitungsblöcke (512 Werte) und einer Abtastfrequenz von 44.1 kHz ergibt dies eine Laufzeitverzögerung von 17.4 ms.

## 6.4 Bearbeitungszeit

Die maximal zulässige Bearbeitungszeit wird durch die Länge des Bearbeitungsblocks, sowie durch die Abtastfrequenz bestimmt. Während ein halber Bearbeitungsblock eingelesen wird, muss ein ganzer Bearbeitungsblock bearbeitet werden. Während der Zeit  $t_{hb}$  (siehe Kapitel 6.3) muss folglich ein ganzer Bearbeitungsblock bearbeitet werden können. Die Tabelle 6.4 zeigt die Zusammensetzung der Bearbeitungszeit des DSP-Programms. Die Zeiten können durch den Compiler weiter optimiert werden.

Die angegebenen Bearbeitungszeiten beziehen sich auf ein Monosignal. Bei einem Stereosignal verdoppelt sich der Rechenaufwand und somit auch die Bearbeitungszeit.

Funktion	Zeit in ms
Eingangsbuffer auslesen	0.44
Fensterfunktion anwenden	0.47
FFT	0.98
Bandmittelwerte berechnen	0.52
Banddynamik bestimmen	0.13
Bandfaktoren ermitteln	0.14
Frequenzspektrum anpassen	1.28
IFFT	1.10
Fensterfunktion anwenden	0.47
Ausgangsbuffer beschreiben	0.48
Total	6.01
Total (Compiler optimiert)	3.00

Tabelle 6.1: Zusammensetzung der Bearbeitungszeit

---

## 7 DSP Steuerung

---

Um die Arbeit des Signalprozessors überwachen zu können, haben wir ein Kontrollprogramm entwickelt. Dieses Programm ermöglicht uns, auf den Bearbeitungsvorgang Einfluss zu nehmen. Zusätzlich kann der Vorgang durch das Anzeigen von Zwischenergebnissen veranschaulicht werden.

Nach den guten Erfahrungen mit der Entwicklungsoberfläche VisualC++ bei der Semesterarbeit lag es nahe, das selbe Entwicklungswerkzeug ein weiteres mal einzusetzen.

### 7.1 Systemvoraussetzungen

Das Programm "DSP\_Control" ist unter allen Win32 Systemen (Win95 / Win98 / WinNT) lauffähig. Ebenfalls muss im Computer eine DSP-Karte (Sharc ADSP-21060) eingebaut sein.

### 7.2 Installation

Das Programm muss nicht installiert werden. Folgende Dateien müssen jedoch im selben Verzeichnis zu liegen kommen.

<i>DSP_Control.exe</i>	PC-Programm
<i>DSP_Control.hlp</i>	Hilfe des PC-Programms
<i>hil32.dll</i>	Softwarebibliothek des DSP-Boards
<i>Noise.21k</i>	DSP-Programm

### 7.3 Bedienung des Programms

Das Programm präsentiert sich dem Benutzer in Form eines Win32-Dialogs. Es ermöglicht das Steuern des DSP-Programms sowie das Betrachten von Zwischenergebnissen. Dieses Kapitel erklärt die Elemente des Dialogs (siehe Abbildung 7.1) und deren Funktion.



Abbildung 7.1: Bedienoberfläche von "DSP\_Control"

#### 7.3.1 Steuerung des DSP-Programms

##### DSP Start

Mit der Schaltfläche "DSP Start" kann ein (Neu-)Start des Programms bewirkt werden. Die Einstellungen werden auf die Startwerte zurückgesetzt.

##### DSP Reset

"DSP Reset" bewirkt einen Reset des Signalprozessors. Die DSP-Karte wird in den Grundzustand versetzt.

##### Bypass

Die Checkbox "Bypass" dient dazu, das Eingangssignal am Signalprozessor direkt (ohne Verarbeitung) an den Ausgang zu geben. Somit wird ein Vergleich des bearbeiteten Signals mit dem Originalsignal ermöglicht.

### **Maximale Dämpfung**

Die "Maximale Dämpfung" ist im Kapitel 5.4 unserer Semesterarbeit [3] genauer beschrieben.

### **Maximale Dynamik**

Die "Maximale Dynamik" ist im Kapitel 5.4 unserer Semesterarbeit [3] genauer beschrieben.

### **Anstiegsfaktor**

Der "Anstiegsfaktor" ist im Kapitel 5.3 unserer Semesterarbeit [3] genauer beschrieben.

### **Absinkfaktor**

Der "Absinkfaktor" ist im Kapitel 5.3 unserer Semesterarbeit [3] genauer beschrieben.

## **7.3.2 Überwachung des Signalprozessors**

Neben der Steuerung des DSP-Programms kann auch die Störgeräuschunterdrückung überwacht werden. Einzelne Zwischenergebnisse können auf dem Bildschirm dargestellt werden. Die Darstellungen werden kontinuierlich auf den neusten Stand gebracht.

Aus einer Liste wird ausgewählt, welche Daten dargestellt werden sollen. Die Schaltfläche "Werte Anzeigen" aktiviert die gewünschte Darstellung. Die genaue Bedeutung dieser Darstellungen kann der Dokumentation unserer Semesterarbeit [3] entnommen werden. Nachfolgend werden die drei verschiedenen Darstellungsarten erklärt.

### 7.3.2.1 Amplitudenspektrum

Das Ein- sowie Ausgangsspektrum des Signals wird wie folgt dargestellt. Die Frequenzachse wird bis zur halben Abtastfrequenz angezeigt. In Y-Richtung wird die Amplitude angezeigt.

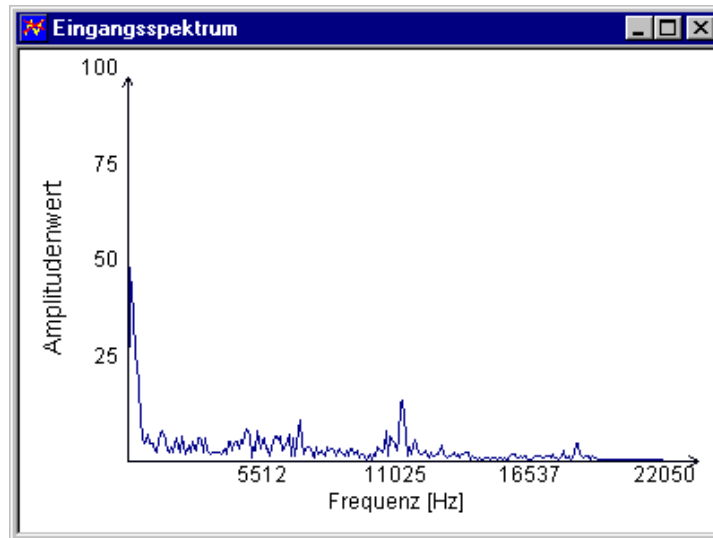


Abbildung 7.2: Darstellung eines Amplitudenspektrums

### 7.3.2.2 Hüllkurve

Für jedes Band kann die Hüllkurve angezeigt werden. Das anzuzeigende Band wird aus einer Liste ausgewählt. Die Hüllkurve zeigt den Verlauf der Maximal- und Minimalhüllkurve. Ebenfalls wird der Ist-Wert dargestellt. Die Hüllkurve kann jeweils nur für ein Band dargestellt werden.

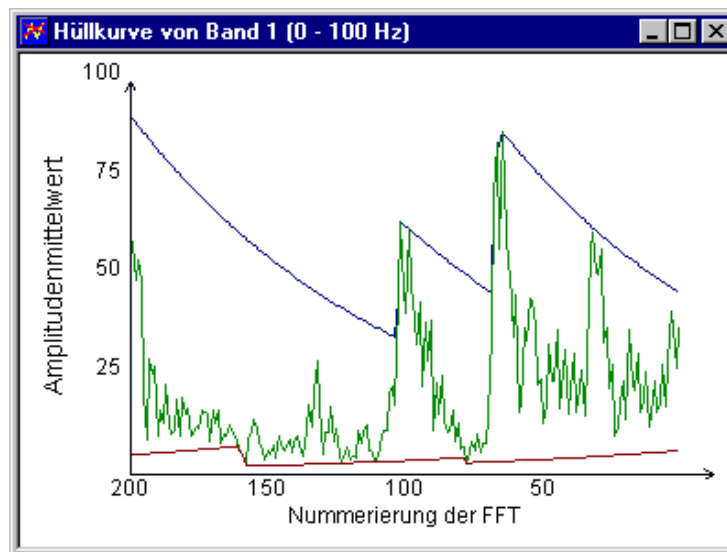


Abbildung 7.3: Darstellung einer Hüllkurve

### 7.3.2.3 Balkengrafik

Einige Daten werden für jedes Band einzeln bestimmt. Beispielsweise werden die Banddynamik und die Bandfaktoren auf diese Weise dargestellt.

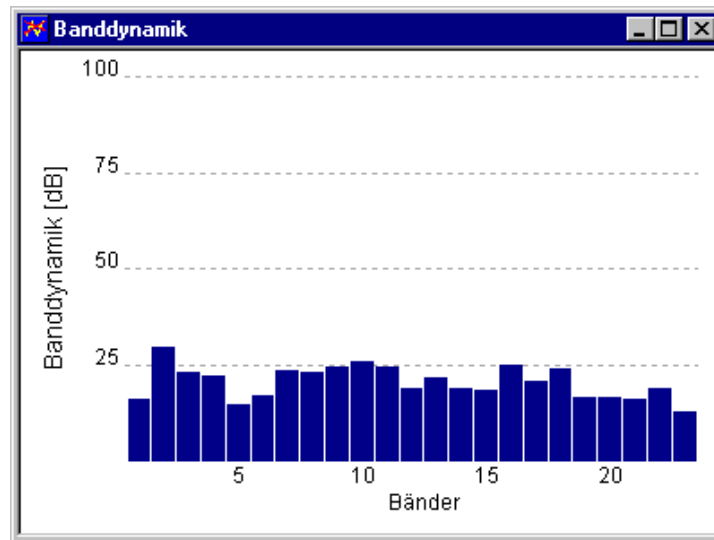


Abbildung 7.4: Darstellung einer Balkengrafik

## 7.4 Aufbau des Programms

Das Programm ist in Objekte aufgeteilt. Wir trennen zusätzlich die grafische Oberfläche (GUI) von der DSP-Steuerung (PD). Im Klassendiagramm (siehe Kapitel 7.4.1) ist die Aufteilung und Vernetzung der Objekte gut ersichtlich.

### 7.4.1 Klassendiagramm

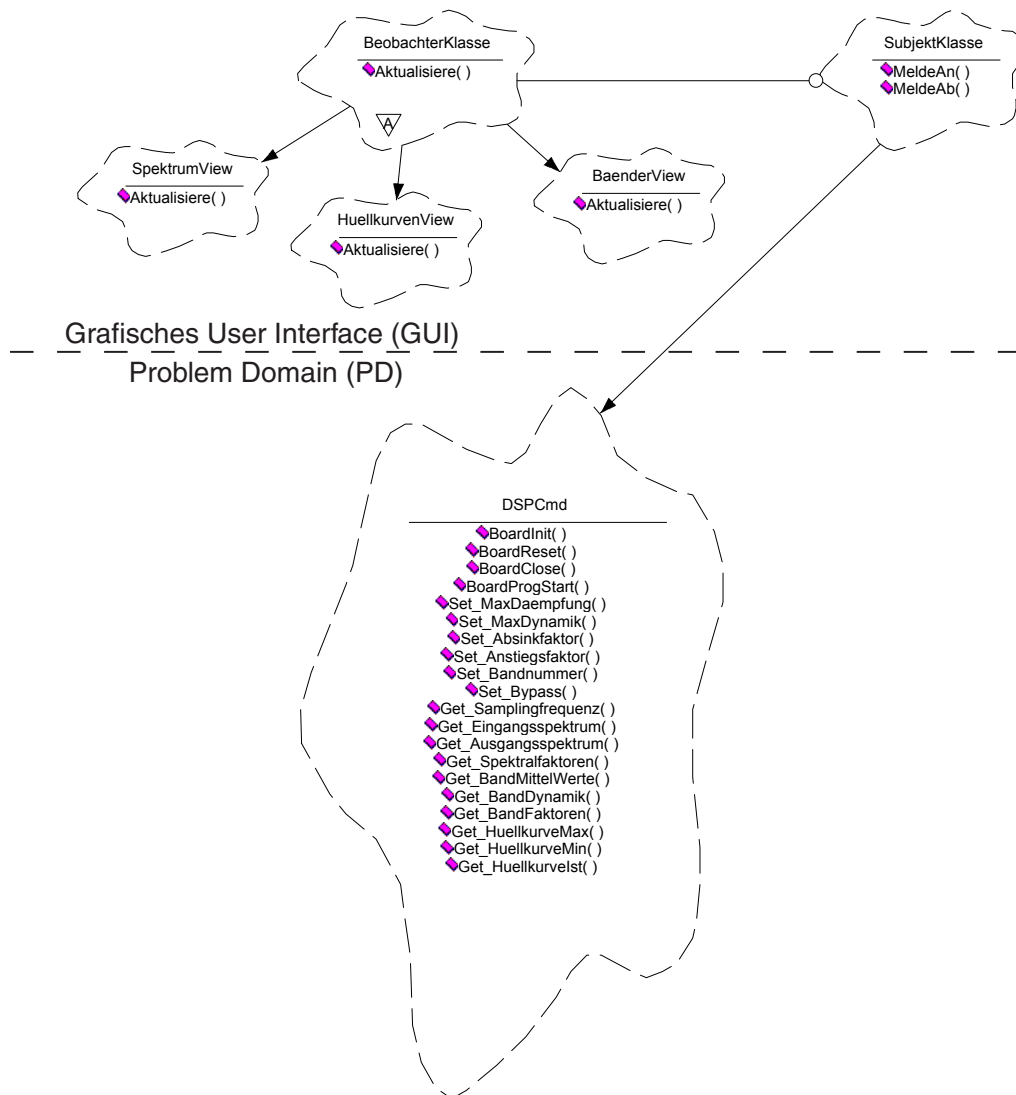


Abbildung 7.5: Klassendiagramm von "DSP\_Control"

**7.4.1.1 Observer-Pattern**

Wie bereits in der Semesterarbeit kam auch hier das Observer-Pattern zum Einsatz. Es ermöglicht die Trennung der Kommunikation mit dem Signalprozessor von der Systemoberfläche. Das Klassendiagramm (Abbildung 7.5) verdeutlicht diese Trennung.

Die Klasse "DSPCmd" ist die Subjektklasse. Beobachterklassen dieses Subjekts sind die Klassen welche für die Darstellung der Zwischenergebnisse zuständig sind.

**7.4.1.2 DSPCmd**

Die Klasse DSPCmd ist für die gesamte Kommunikation mit dem Signalprozessor verantwortlich (siehe Kapitel 7.5). Das Übergeben von Steuerwerten sowie das Anfordern von Zwischenergebnissen sind die Hauptaufgaben dieser Klasse. Ebenfalls wird die Initialisierung der DSP-Karte von dieser Klasse übernommen.

### 7.5 Kommunikation zwischen dem DSP und dem PC

Das DSP-Board kann über Subroutinen aus einer Softwarebibliothek direkt aus einem C-Programm angesprochen werden. Die verwendete Softwarebibliothek "hil32m.lib" gehört zum Softwarepaket des DSP-Boards. Eine ausführliche Beschreibung dieser Bibliothek kann der dazugehörigen Dokumentation [6] entnommen werden. Vor allem die Möglichkeit, Variablen des Signalprozessors zu beschreiben oder auch auszulesen, ist sehr nützlich. So können wir jederzeit die Eigenschaften des Bearbeitungsvorgangs verändern.

Etwas schwieriger gestaltete sich die Überwachung der Zwischenergebnisse. Um zu verhindern, dass Daten aus verschiedenen Bearbeitungszyklen dargestellt werden, haben wir einen Synchronisationsmechanismus implementiert. Die Abbildung 7.6 illustriert den Ablauf der Synchronisation. Das PC-Programm kündigt dem Signalprozessor an, dass es neue Zwischenergebnisse benötigt. Das DSP-Programm überträgt die neuen Daten des nächsten Zyklus in dafür vorgesehene Variablen. Nachdem der darzustellende Zyklus beendet ist, löst der Signalprozessor im PC-Programm einen Interrupt aus. In der Interrupt Service Routine (ISR) können die Zwischenergebnisse mit den Funktionen der Softwarebibliothek ausgelesen werden.

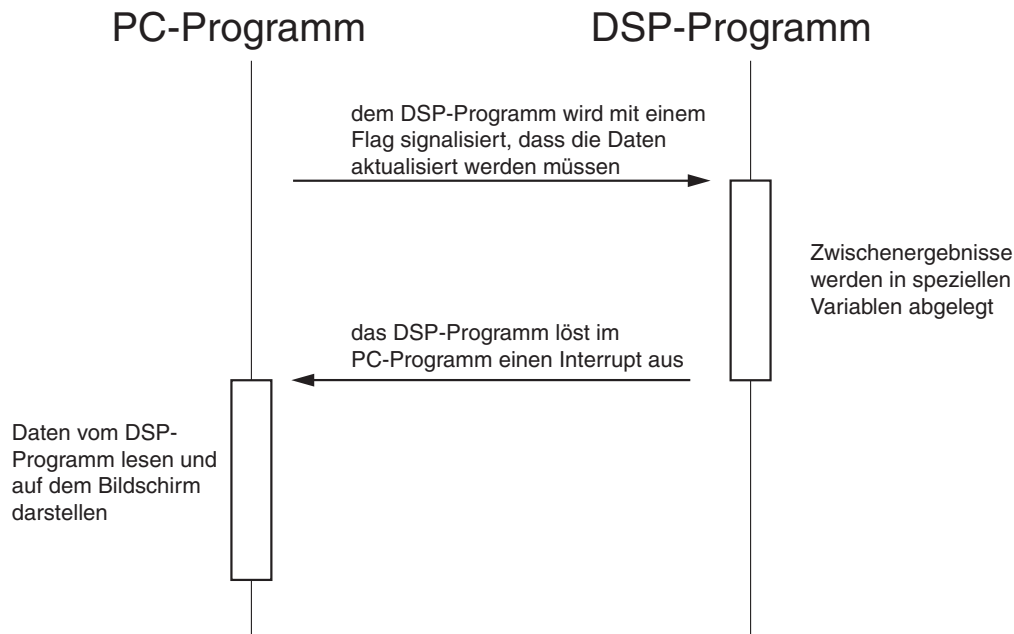


Abbildung 7.6: Synchronisation PC ↔ DSP



## 8 Optimierung mit Assembler

---

Das Programmieren in Assembler bringt grosse Vorteile mit sich. Zum einen kann direkt auf die Hardware zugegriffen werden. Der Programmcode kann auf diese Weise sehr genau auf die Fähigkeiten der Hardware abgestimmt werden. Gerade Signalprozessoren stellen verschiedenste Zähler, Register und parallele Recheneinheiten zur Verfügung. Diese Einheiten können mit der Hilfe von Assembler gezielt eingesetzt werden. Auf diese Weise wird die Prozessorleistung optimal genutzt. Diese Vorteile werden aber durch eine eher unübersichtliche Programmdarstellung erkauft. Eine Gliederung des Codes kann jedoch durch das Verwenden von Funktionen erfolgen.

Wir haben zwei Möglichkeiten der Assembleroptimierung angewendet. Die Assembleranweisungen können direkt in den C-Code eingesetzt werden. Der Compiler reiht diese Anweisungen dann selbständig ein. Diese Assemblerbefehle im C-Code kommen zur Anwendung, wenn keine entsprechende Funktion in C existiert oder wenn direkt auf die Hardware zugegriffen werden soll. Eine weitere Möglichkeit, den Code mit Assembler zu optimieren, bietet sich nach dem Compilieren des C-Codes an. Der durch den Compiler generierte Assemblercode kann von Hand weiter verbessert werden.

Sehr effizient arbeitet, wer die Compileroptimierung einschaltet und danach den generierten Assemblercode von Hand weiter verbessert. So kann die Rechenzeit weiter reduziert werden, und der Programmieraufwand wird nicht zu hoch. Eine Optimierung von Hand ist speziell bei Schleifen wie sie bei FIR-Filtern vorkommen sinnvoll. Wenn die Filterkoeffizienten im Programmspeicher und die Signaldaten im Datenspeicher platziert sind, kann der Prozessor in einem Zyklus beide Speicherstellen einlesen und parallel dazu eine Multiplikation und eine Operation in der Alu ausführen. Im nachfolgenden Beispiel wird gezeigt, wie die vier Assemblerbefehle parallel zu liegen kommen.

Serielle Bearbeitung ( 4 Zyklen ):

f14 = f1*f6;	Fliesskomma Multiplikation
f7 = f8+f15;	Fliesskomma Addition
f2 = dm(i6,m6);	Zugriff auf Datenspeicher über DAG-Register
f3 = pm(i9,m9);	Zugriff auf Programmspeicher über DAG-Register

Parallele Bearbeitung ( 1 Zyklus ):

f14=f1\*f6,f7=f8+f15,f2=dm(i6,m6),f3=pm(i9,m9);

Pro Schleifendurchlauf werden somit nicht vier sondern bloss ein Zyklus für die Berechnungen benötigt. Dies zahlt sich speziell bei Filter mit langen Stossantworten aus. Die Zeitersparnis multipliziert sich mit der Anzahl Schleifendurchläufe.

Die Operanden müssen je nach bevorstehender Operation in speziell für diese Operation bereitgestellte Register geschrieben werden. Wie diese Register den Operationen zugeordnet sind, ist im User's Manual [4] unter dem Begriff "Multifunctional Computations" beschrieben. Auch die möglichen Kombinationen von den parallel ablaufenden Befehlen sind dargestellt. Auf die Speicherplätze wird über DAG-Register zugegriffen. Sie stellen unter anderem einen Speicherplatzzeiger zur Verfügung, welcher dekrementiert werden kann und dadurch auf die aktuelle Datenposition zeigt. Einzelheiten können dem User's Manual [4] unter dem Begriff "Data Addressing" entnommen werden.

Zu Testzwecken haben wir ein FIR-Filter implementiert (siehe Anhang). Ziel war es, die Rechenzeit bei verschiedenen Optimierungstufen zu bestimmen. Die Messungen ergaben folgende Resultate :

FIR-Filter ohne Compileroptimierung: 89.9 us.

FIR-Filter mit Compileroptimierung: 43.1 us.

FIR-Filter mit Compileroptimierung und Verbesserung von Hand: 8.6 us.

Diese Resultate zeigen, dass der Compiler die Rechendauer etwa halbieren kann. Wenn man nun aber die Zeilen innerhalb der Schleife von Hand weiter reduzieren kann, erreicht man Zeitersparnisse die sich mit der Anzahl Schleifendurchläufe multiplizieren. In diesem Beispiel wird die Schleife 146 mal durchlaufen.

---

## 9 Festkommaarithmetik

---

Fliesskommaprozessoren sind speziell auf naturwissenschaftliche Aufgabenstellungen zugeschnitten. Sie decken einen weiten Zahlenbereich ab und weisen auch eine hohe Genauigkeit auf. Die Arbeit mit Festkommazahlen ist im Gegensatz dazu auf einen kleineren Zahlenbereich begrenzt. Zudem kann das Fehlen von Kommastellen bei Programmen einige Schwierigkeiten in der Codierung bereiten. Trotzdem werden bei Produkten mit hohen Stückzahlen oft Festkommaprozessoren eingesetzt. Dies auf Grund der bedeutend tieferen Kosten. Ein weiterer bedeutender Vorteil der Festkommaprozessoren ist deren geringer Leistungsbedarf, was sich speziell bei netzunabhängigen Geräten auszahlt. Ebenfalls benötigen sie weniger Platz als Fliesskommaprozessoren. Aus diesen Gründen werden heute über 80% aller Signalprozessoren mit Festkommaarithmetik hergestellt. Sie sind unter anderem in Modems, Handys oder Motorensteuerungen zu finden.

Aufgrund der grossen Verbreitung von Festkommaprozessoren war es Teil unserer Aufgabe, das Programm auf Festkommaarithmetik umzustellen.

### 9.1 Warum Java?

Festkommaprozessoren arbeiten häufig mit 32 Bit Integerzahlen. Für Multiplikationen stehen ihnen doppeltsobreite Register zur Verfügung. Leider bietet C++ keinen Festkomma-Datentyp mit mehr als 32 Bit. Als Alternative bot sich Java mit dem 64 Bit Integerdatentyp "long" an. Des Weiteren erwarteten wir keine grossen Probleme bei der Umwandlung des Programms von C++ nach Java.

### 9.2 Programmumstellung

Als Erstes mussten wir die C++ Version in eine Java Version umschreiben. Alle Flieskommazahlen konnten dann durch Festkommazahlen mit n Kommastellen (siehe Kapitel 9.3.1.1) ersetzt werden. Als Folge davon mussten die arithmetischen Operationen diesen Zahlen angepasst werden (siehe Kapitel 9.3.1.2 und 9.3.2).

### 9.3 Umgang mit Festkommazahlen

Wir arbeiten mit zwei Formaten für Festkommazahlen. Zum Einen benutzen wir die ursprünglichen Integerwerte und zum Anderen unsere eigenen Festkommazahlen, welche n Kommastellen aufweisen.

#### 9.3.1 Festkommazahl mit n Kommastellen

##### 9.3.1.1 Interpretation

Um unser Programm umsetzen zu können, mussten wir die ursprünglichen Fließkommazahlen in Festkommazahlen ablegen.

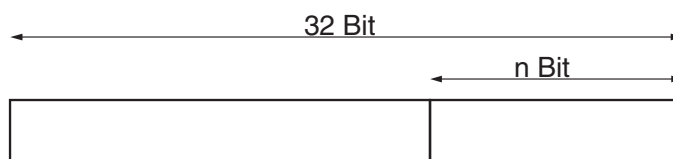


Abbildung 9.1: Interpretation einer 32 Bit Festkommazahl mit n Kommastellen

Fließkommazahlen lassen sich mit der Gleichung 9.1 in entsprechende Festkommazahlen mit n Kommastellen umrechnen.

$$y = 2^n * x \quad (9.1)$$

$x$  Fließkommazahl

$y$  Festkommazahl mit n Kommastellen

### 9.3.1.2 Multiplikation

Bei Rechenoperationen mit unseren Festkommazahlen gilt es folgendem Umstand Beachtung zu schenken. Bei Multiplikationen zweier solcher Werte entsteht ein Fehler um den Faktor  $2^n$ . Der Grund für diesen Fehler wird in der Gleichung 9.2 veranschaulicht.

$$u * v = (x * 2^n) * (y * 2^n) = x * y * 2^{2*n} = x * y * 2^n * 2^n \quad (9.2)$$

- $u$  erste Festkommazahl mit  $n$  Kommastellen
- $v$  zweite Festkommazahl mit  $n$  Kommastellen
- $x$  entsprechende Fliesskommazahl zu  $u$
- $y$  entsprechende Fliesskommazahl zu  $v$
- $n$  Anzahl Kommastellen der eingesetzten Festkommazahlen

Das korrekte Resultat der Multiplikation wäre  $x * y * 2^n$ . Durch die Verwendung der Integermultiplikation entsteht jedoch ein Fehler um den Faktor  $2^n$ . Dieser Fehler kann mit einem Bitshift um  $n$  Bit nach rechts korrigiert werden.

Bei anderen Operationen tritt das Problem in äquivalenter Weise auf. Die Lösung erfolgt wie bei der Multiplikation mit einem Bitshift-Befehl.

9.3.2 Logarithmische Operationen

9.3.2.1 Vom Hüllkurvenverhältnis zur Dynamik

Ein grösseres Problem war die Tatsache, das für Festkommazahlen kein Zehnerlogarithmus zur Verfügung steht. Die ursprüngliche Bestimmung der Dynamik (siehe Gleichung 9.3) musste deshalb ersetzt werden.

$$d(x) = 20 * \log \frac{A_{max}(x)}{A_{min}(x)} \tag{9.3}$$

- $d$         Dynamik
- $A_{max}$    Wert der Maximalhüllkurve
- $A_{min}$    Wert der Minimalhüllkurve

Die folgende Abbildung zeigt wie die Dynamik ohne den Zehnerlogarithmus bestimmt werden kann.

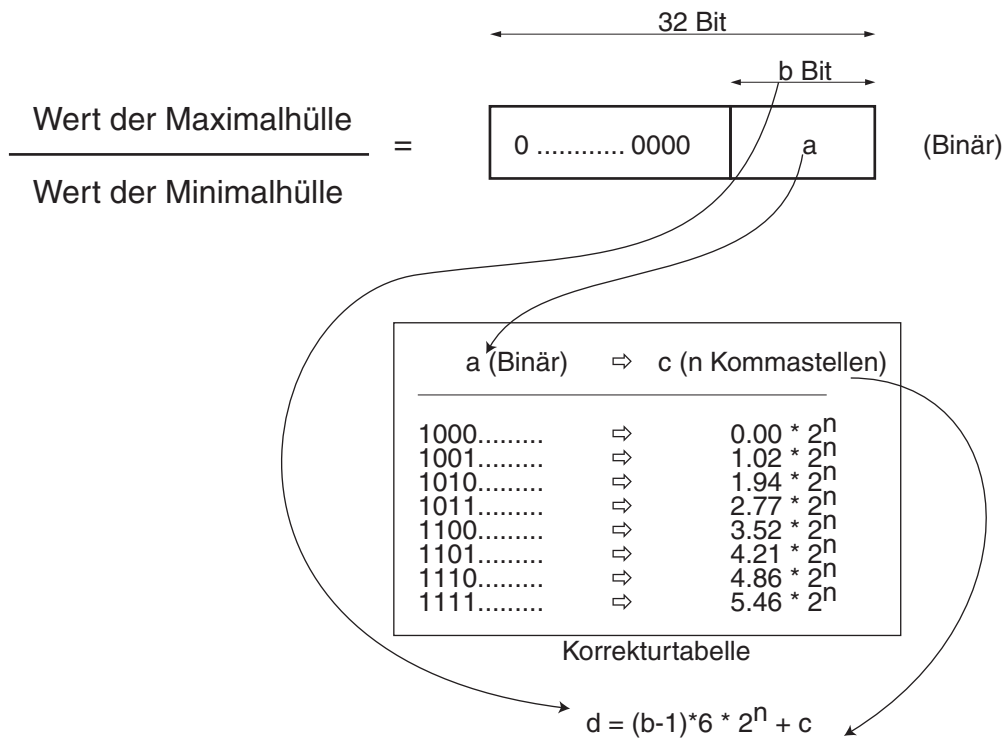


Abbildung 9.2: Berechnung der Dynamik aus dem Hüllkurvenverhältnis

Mit einer Intergerdivision wird das Hüllkurvenverhältnis ermittelt.

Für das weitere Vorgehen spielt die Anzahl relevanter Bits des Hüllkurvenverhältnisses ( $\Rightarrow b$ ) eine entscheidende Rolle. Das Verhältnis enthält folglich  $b - 1$  mal den Faktor 2. Es ergibt sich pro Faktor 2 eine Dynamik von 6dB.

Um bei der Dynamik nicht Werte mit 6dB Abstufung zu erhalten, sind Korrekturwerte erforderlich. Diese Werte werden aus einer Korrekturtabelle entnommen. Ein Korrekturwert ( $\Rightarrow c$ ) wird durch die vier vordersten relevanten Stellen des Hüllkurvenverhältnisses bestimmt. Der Korrekturwert ergibt sich aus folgender Gleichung.

$$c = 20 * (\log(x) - \log(8)) * 2^n \tag{9.4}$$

- $c$  Korrekturwert mit  $n$  Kommastellen aus Korrekturtabelle
- $x$  Wert der durch die vier vordersten Bits von  $a$  dargestellt wird

Die Dynamik lässt sich nun mit der Formel 9.5 bestimmen.

$$d = (b - 1) * 6 * 2^n + c \tag{9.5}$$

- $b$  Anzahl relevanter Bits des Hüllkurvenverhältnisses
- $c$  Korrekturwert aus Korrekturtabelle mit  $n$  Kommastellen
- $d$  Dynamik als Festkommazahl mit  $n$  Kommastellen
- $n$  Anzahl Kommastellen der eingesetzten Festkommazahlen

9.3.2.2 Dämpfung zu Bandfaktoren

Auch bei der Umrechnung der Dämpfung zum Bandfaktor kann nicht die Formel aus der Fließkommaversion (Gleichung 9.6) verwendet werden.

$$k_{Band} = \frac{1}{10^{\frac{A_{dB}}{20}}} \tag{9.6}$$

$k_{Band}$  Bandfaktor  
 $A_{dB}$  Dämpfung

Das Vorgehen um dieses Problem zu lösen ähnelt jenem aus dem vorangegangenen Kapitel. Die Abbildung 9.3 visualisiert den Ablauf der Umwandlung.

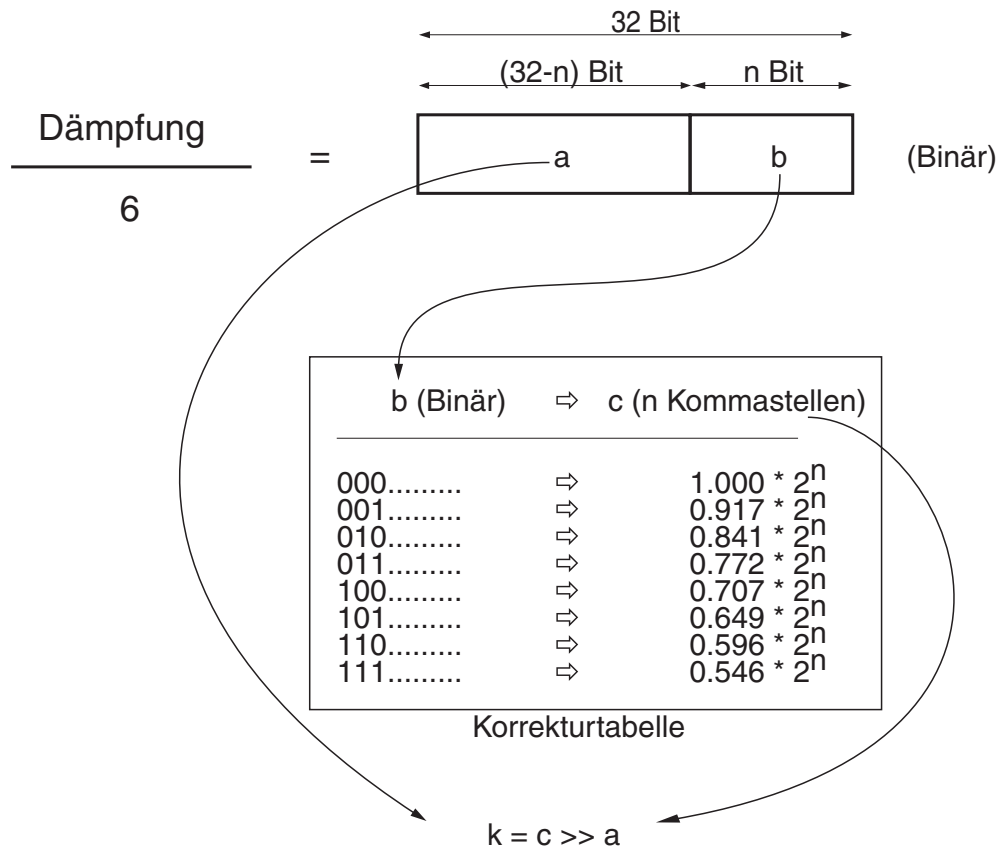


Abbildung 9.3: Berechnung des Bandfaktors aus der Dämpfung

Die anzuwendende Dämpfung wird mit einer Integerdivision durch 6 dividiert. Da die Dämpfung mit n Kommastellen vorliegt, gilt dies auch für das Ergebnis der Division.

Aus den Kommastellen ( $\Rightarrow b$ ) wird mit Hilfe einer Tabelle ein Grundfaktor ( $\Rightarrow c$ ) bestimmt. Dieser Grundfaktor entspricht dem Bandfaktor wenn die Dämpfung kleiner als 6dB ist. Er wird mit folgender Gleichung ermittelt.

$$c = 10^{\frac{8*20}{x*6}} \tag{9.7}$$

- $c$  Korrekturwert aus Korrekturtabelle (Grundfaktor)
- $x$  Wert der drei ersten Nachkommastellen

Wenn die Dämpfung grösser als 6dB ist, muss der Grundfaktor korrigiert werden. Aus der Division ist auch bekannt, wieviel mal die gewünschte Dämpfung 6dB enthält ( $\Rightarrow a$ ). Pro 6dB muss der Grundfaktor durch 2 dividiert werden. Der Bandfaktor errechnet sich folglich aus der folgenden Formel.

$$k_{Band} = c >> a \tag{9.8}$$

- $a$  Anzahl in der Dämpfung enthaltene ganze 6dB
- $c$  Korrekturwert aus Korrekturtabelle mit n Kommastellen
- $k_{Band}$  Bandfaktor mit n Kommastellen
- $n$  Anzahl Kommastellen der eingesetzten Festkommazahlen

### 9.4 Genauigkeit

Es stellt sich die Frage nach der Genauigkeit der Version mit Festkommaarithmetik. Sie wird bestimmt durch die Anzahl Kommastellen der Festkommazahlen ( $\Rightarrow n$ ). Diese Grösse ist in der Festkommaversion beliebig wählbar. Allerdings kann es bei zuvielen Kommastellen vorkommen, dass der Zahlenbereich zu klein wird und eine fehlerhafte Ausgabe entsteht.

Bei geeigneter Wahl der Anzahl Kommastellen (z.B. 12) ist das Resultat kaum vom Resultat der Fließkommaversion zu unterscheiden. Die Abstufungen, welche in den Bandfaktoren durch die eigene Implementation der logarithmischen Operationen entstehen, sind nicht hörbar.



---

## 10 Diskussion der Ergebnisse

---

Um die "Mehrkanalige Störgeräuschunterdrückung" auf dem Signalprozessor realisieren zu können, mussten wir zu Beginn verschiedene Mechanismen des Signalprozessors studieren. Beispiele hierfür sind der DMA-Transfer und das DAG-Register. Weiter haben wir das C++ Program "NoiseErase" in C Code umgeschrieben. Diesen Code konnten wir nach ein paar wenigen zusätzlichen Anpassungen auf dem Signalprozessor einsetzen.

Auch der Problematik der Festkommaarithmetik haben wir uns angenommen. Hierfür setzten wir die Programmiersprache Java ein. Wir entwickelten einen eigenen Festkomma-Datentypen mit dazugehörigen Funktionen. Speziell die Logarithmusfunktion hat uns einiges abverlangt. Mit einer Kombination von Berechnungsschritten und einer Tabelle konnte dieses Problem jedoch elegant gelöst werden.

Unser Programm "NoiseErase" eignet sich schlecht für eine Optimierung mit der Hilfe von Assembler. Denn die Komplexität unseres Programms lässt eine Optimierung nur mit viel Zeitaufwand zu. Nach einer Rücksprache mit Herrn Schaub haben wir uns entschieden, dieses Thema anhand eines einfacheren Beispiels zu behandeln. Unsere Wahl fiel auf ein FIR-Filter. Der schleifenförmige Aufbau dieses Filters kann durch Assembler gut optimiert werden.

Die Dokumentation haben wir parallel zur Arbeit stets auf den aktuellen Stand gebracht. So kamen wir auch gegen Ende der Diplomarbeit nicht vom Zeitplan ab.



## 11 Schlusswort

---

Die Diplomarbeit gab uns die Möglichkeit, unsere Semesterarbeit noch weiter zu vertiefen. Speziell die Problematik der Echtzeitprogrammierung förderte neue Erkenntnisse zu Tage. Bei der Umstellung auf Festkommaarithmetik erkannten wir, wie einfach doch die Arbeit mit Fließkommazahlen ist. Nur mit kniffligen mathematischen Lösungsansätzen konnten gewisse Operationen in Festkommaarithmetik nachgebildet werden.

Zum Schluss möchten wir allen Beteiligten recht herzlich für ihre Unterstützung danken. Spezieller Dank geht an Herrn A. Schaub für die vielen wertvollen Tips sowie sein Engagement. Ebenfalls bedanken wir uns bei Herrn A. Rüegg, der uns in seiner Funktion als Laborassistent jederzeit gerne unterstützte.

Rapperswil 7. Dezember 1999

Adrian Eggenberger

Mischa Jud



## 12 Abbildungsverzeichnis

3.1	Zeitplan . . . . .	11
5.1	Arbeiten mit Comptool . . . . .	15
5.2	Bedienoberfläche von Comptool . . . . .	16
5.3	Versuchsaufbau . . . . .	17
6.1	Ablauf der Ein- und Ausgabe . . . . .	20
6.2	DMA Struktur . . . . .	23
6.3	Laufzeitverzögerung . . . . .	24
7.1	Bedienoberfläche von "DSP_Control" . . . . .	28
7.2	Darstellung eines Amplitudenspektrums . . . . .	30
7.3	Darstellung einer Hüllkurve . . . . .	31
7.4	Darstellung einer Balkengrafik . . . . .	32
7.5	Klassendiagramm von "DSP_Control" . . . . .	33
7.6	Synchronisation PC ↔ DSP . . . . .	35
9.1	Interpretation einer 32 Bit Festkommazahl mit n Kommastellen . . . . .	40
9.2	Berechnung der Dynamik aus dem Hüllkurvenverhältnis . . . . .	42
9.3	Berechnung des Bandfaktors aus der Dämpfung . . . . .	44



## 13 Literaturverzeichnis

---

- [1] Bernt W. Edwards, Zezhang Hou, Christopher j. Struck, and Priya Dharan: Signal-processing algorithms for a new software-based, digital hearing device, *The Hearing Journal*, September 1998, Vol. 51, No 9.
- [2] ANSI S3.5-1997. Methods for Calculation of Speech Intelligibility Index, American National Standard Institute, 1997.
- [3] Semesterarbeit "Mehrkanalige Störgeräuschunterdrückung" von Mischa Jud und Adrian Eggenberger
- [4] "ADSP-2106x SHARC User's Manual" (Analog Devices)
- [5] "ADSP-21000 Family C Runtime Library Manual" (Analog Devices)
- [6] "DSP21k Toolkit User's Guide" Beschreibung der Software zum DSP-Board



---

## 14 Glossar

---

ALU	Arithmetic Logic Unit - Recheneinheit des Signalprozessors
Bitshift	ein Wert wird um eine Anzahl Bits nach Rechts oder nach Links geschoben
Comptool	Matlab Programm um das Umwandeln von C-Code in Assembler-Code zu vereinfachen und Programme auf dem Signalprozessor auszuführen
DAG-Register	Data Address Generator - Register für schnellen Speicherzugriff
DSP	Digitaler Signalprozessor
DMA	Direkter Speicherzugriff
ISR	Interrupt Service Request - Anfrage für die Abarbeitung eines Interrupts
MIPS	Million Instruction per Second
"NoiseErase"	Programm zur mehrkanaligen Störgeräuschunterdrückung aus unserer Semesterarbeit
VisualC++	Entwicklungsumgebung für C++

---

## Index

---

- ”Comptool”, 15
- Überwachung des Signalprozessors, 29
  
- Abbildungsverzeichnis, 51
- ADSP-21060, 13
- Amplitudenspektrum, 30
- Assembler, 37
  
- Balkengrafik, 32
- Bearbeitungszeit, 26
- Bypass, 28
  
- DAG-Register, 38
- Darstellungsarten, 29
- DMA, 22
- DSP\_Control, 27
  
- Echtzeitanwendung, 19, 24
  
- Festkomma, 39
- Fließkomma, 39
  
- Genauigkeit, 45
  
- Hüllkurve, 31
  
- Java, 39
  
- Klassendiagramm, 33
- Kommunikation PC ↔ DSP, 35
- Korrekturtabelle, 43, 45
  
- Laufzeitverzögerung, 24
- Literaturverzeichnis, 53
- Logarithmische Operationen, 42
  
- Optimierung, 37
  
- Sharc, 13
- Steuerung des DSP-Programms, 28
  
- Transfer Control Block, 22
  
- Versuchsaufbau, 17
- VisualC++, 27
  
- Zeitplan, 11
- Zyklus, 13

## Anhang A Listing : DSP Programm

### A.1 Version 1.2 Stereo

```

/*****\
* NOISEERASE *
*****
* Version : 1.2 Stereo *
*****
* Datum : 03.11.99 *
*****
* (c) 1999 by Mischa Jud & Adrian Eggenberger *
\*****/

/*-----*/

/* ADSP-21060 System Register bit definitions */
#include <def21060.h> // register definitions
#include <21060.h> // flag / timer definitions
#include <signal.h> // interrupt
#include <sport.h> // serial port
#include <trans.h> // rfft, ifft
#include <math.h> // fmod
#include <macros.h> // MIN, circular-buffer

#include "ctrlregs.h"
#include "bitsibb.h" // bitsi board
CIRCULAR_BUFFER(float,0,Huellkurven); // definition des Circularbuffers

#include "bitsi.h"

#define CP_PCI 0x20000 /* Program-Controlled Interrupts bit */
#define CP_MAF 0xiffff /* Valid memory address field bits */

/* macros */
#define SetIOP(addr, val) (* (int *) addr) = (val)
#define GetIOP(addr) (* (int *) addr)

/* Convert a 16 bit Integer value (the 16 left bits in a 32 bit word x)
to a 32 bit Integer value */
#define convertleft16to32(result, x) \
asm("%0=fext %1 by 16:16 (se);" : "d" (result) : "d" (x));

/* Convert a 16 bit Integer value (the 16 right bits in a 32 bit word x)
to a 32 bit Integer value */
#define convertright16to32(result, x) \
asm("%0=fext %1 by 0:16 (se);" : "d" (result) : "d" (x));

#define MAX_U_AD 2.75 /* maximum input voltage for A/D converter */
#define MAX_U_DA 3.0 /* maximum output voltage on D/A converter */

/*-----*/

/* DMA chaining Transfer Control Blocks */
typedef struct {
    unsigned** cp; /* Chain Pointer to next TCB */
    unsigned c; /* Count register */
    int im; /* Index modifier register */
    unsigned * ii; /* Index register */
} _tcb_;

#define NUM_BUFS 2 /* number of IFFT buffers needed */
#define TCB_BUFS 3 /* number of TCB buffers needed */

#define HALF_FFT_SIZE 256 /* half FFT length */
#define FFT_SIZE 512 /* FFT length */

#define rfft rfft512
#define ifft ifft512

#define fc 44100 /* sample frequency fc */

```

## ANHANG A. LISTING : DSP PROGRAMM Mehrkanalige Störgeräuschunterdrückung

```

/*-----*/

/* sport buffers */
int rx0_buf[TCB_BUFS][HALF_FFT_SIZE];      /* receive buffer */
int tx0_buf[TCB_BUFS][HALF_FFT_SIZE];      /* transmit buffer */

/* TCB's */
_tcb_rx0_tcb[TCB_BUFS] = { {0, HALF_FFT_SIZE, 1, 0}, /* receive tcb */
  {0, HALF_FFT_SIZE, 1, 0}, /* receive tcb */
  {0, HALF_FFT_SIZE, 1, 0} }; /* receive tcb */

_tcb_tx0_tcb[TCB_BUFS] = { {0, HALF_FFT_SIZE, 1, 0}, /* transmit tcb */
  {0, HALF_FFT_SIZE, 1, 0}, /* transmit tcb */
  {0, HALF_FFT_SIZE, 1, 0} }; /* transmit tcb */

/* set correction terms for input / output voltages */
float NORM = MAX_U_AD / 32767.0; /* normalization of sampled input values */
float CORR = 32767.0 / (MAX_U_DA*FFT_SIZE); /* denormalization for output values */

/* globals */
int *dms3;

/*****/

// Global variables.
volatile int rcvd_ints; /* Number of interrupts DSP has received.
volatile int interrupt_rcvd; /* We've been interrupted.
volatile int PC_Ready;

volatile int Test;

int *ms2_ptr; /* Points to beginning of PCI Chip control registers.

/*****/

/* FFT variables: rin -> real input, rout -> real output, iout -> imaginary output */
float FFT_rin[FFT_SIZE], FFT_rin[FFT_SIZE], FFT_rout[FFT_SIZE], FFT_iout[FFT_SIZE];

/* IFFT variables: rin -> real input, iin -> imag input, rout -> real output, iout -> imag output */
float IFFT_rin[FFT_SIZE], IFFT_iin[FFT_SIZE], IFFT_rout[FFT_SIZE], IFFT_iout[FFT_SIZE];

/* temporary buffers */
float ext_rec_buf[HALF_FFT_SIZE];

/* indices / pointers */
int tx_ptr; /* Pointer to the newer transmit DMA Buffer */
int tx_ptrb; /* Pointer to the oldest transmit DMA Buffer */
int rx_ptr = 0; /* Pointer to the newer receive DMA Buffer */
int rx_ptrb; /* Pointer to the oldest receive DMA Buffer */

// Einstellungen der zu kompilierenden Variante
#define PC_WATCH
#define TIMER_ON

// Variablen zur Ueberwachung des Prozesses
#ifndef PC_WATCH
#define Circ_Buf_Size 600 // Circular Buffersize
float CircularData[Circ_Buf_Size]; // Array das als Ringbuffer benutzt wird

int Watch_Bandnummer = 1;

float Watch_FFT_rout[FFT_SIZE];
float Watch_FFT_iout[FFT_SIZE];
float Watch_IFFT_rin[FFT_SIZE];
float Watch_IFFT_iin[FFT_SIZE];
float Watch_Bandmittel[23];
float Watch_Maxkurve [200];
float Watch_Minkurve [200];
float Watch_Istkurve [200];
float Maxkurve [200];
float Minkurve [200];
float Istkurve [200];
float Watch_Banddynamik[23];
float Watch_Bandfaktoren[23];
float Watch_SpektralFaktoren[FFT_SIZE];
const int Watch_fc = fc;

int Bypass = 0; // Bypass Variabel

#endif

```

```

#ifdef TIMER_ON
float et;
#endif

// Typ mit dem die Samples abgelegt werden
typedef float workdatatype;

// Typ der Faktoren der Fensterfunktion
typedef float windowfactortyp;

// Typ der Faktoren des Spektrums
typedef float spectrumfactortyp;

// buffers
workdatatype workvar[FFT_SIZE];
workdatatype outvar[2][FFT_SIZE];
workdatatype pastoutvar [2][HALF_FFT_SIZE];
workdatatype zeroarray [FFT_SIZE];

float Absinkfaktor;
float Anstiegsfaktor;
int MaxDaempfung;
int MaxDynamik;

#define SMOOTHFAKTOR 0

#define DSP_CHANNELS 2

// Windowfunctions:
#define RECHTECK 1
#define HALBSINUS 2

// Baender definition
#define ANZAHL_BAENDER 23

const int BAENDER [ANZAHL_BAENDER][2] =
{{0,100},{100,200},{200,300},{300,400},{400,510},{510,630},{630,770},
{770,920},{920,1080},{1080,1270},{1270,1480},{1480,1720},{1720,2000},
{2000,2320},{2320,2700},{2700,3150},{3150,3700},{3700,4400},{4400,5300},
{5300,6400},{6400,7700},{7700,9500},{9500,45000}};

// Variablen der Fensterfunktion
windowfactortyp Windowfactors [FFT_SIZE];

// Variablen der FFT Funktionen
short FFT_Potenz;

// Hilfsstrukturen der Baenderfunktionen
struct BandWerteStrukt
{
    spectrumfactortyp Wert [ANZAHL_BAENDER];
};

struct WerteImBandStrukt
{
    int Erster;
    int Letzter;
    int Anzahl;
};

// Variablen der Baenderfunktion
unsigned int Baender_Samplingfrequenz;
float Baender_Frequenzabstand;
struct WerteImBandStrukt WerteImBand [ANZAHL_BAENDER];

struct BandWerteStrukt BandMittelWerte;
struct BandWerteStrukt BandMinimalWerte [2];
struct BandWerteStrukt BandMaximalWerte [2];
struct BandWerteStrukt BandDynamikWerte;
struct BandWerteStrukt BandFaktorenWerte;
spectrumfactortyp SpektralFaktoren [FFT_SIZE];

//-----
// void Fenster_UseWindowfactors ( workdatatype* _InVector, workdatatype* _OutVector )
//
// * Fensterfunktion auf einen Vektor anwenden
//
//-----
void Fenster_UseWindowfactors ( workdatatype* _InVector, workdatatype* _OutVector )
{
    int z1;
    for ( z1=0 ; z1<FFT_SIZE ; z1++ )
    {
        _OutVector[z1] = _InVector[z1] * Windowfactors [z1];
    };
};

```

```

//-----
// void Fenster_Create_Windowfactors ( unsigned char _Windowfunction )
//
// * Windowfaktoren in der entsprechenden Form und Laenge generieren
//
//-----
void Fenster_Create_Windowfactors ( unsigned char _Windowfunction )
{
    int z1 = 0;
    double pi = 3.14159265359;

    switch (_Windowfunction)
    {
        case RECHTECK :
        {
            for (z1 = 0; z1 < FFT_SIZE; z1++)
            {
                Windowfactors [z1] = 1;
            };
            break;
            case HALBSINUS :
            {
                for (z1 = 0; z1 < FFT_SIZE>>1; z1++)
                {
                    Windowfactors [z1] = sin (z1*pi/(FFT_SIZE>>1-1));
                };
                for (z1 = 0; z1 < FFT_SIZE>>1; z1++)
                {
                    Windowfactors [FFT_SIZE-1-z1] = Windowfactors [z1];
                };
            };
            break;
            // Defaultmaessig wird Rechteckfenster erzeugt
        default :
        {
            for (z1 = 0; z1 < FFT_SIZE; z1++)
            {
                Windowfactors [z1] = 1;
            };
        };
    };
};

//-----
// void FFT_Transformation ( workdatatype* _FFT_In_Real, workdatatype* _FFT_In_Imag,
// workdatatype* _FFT_Out_Real, workdatatype* _FFT_Out_Imag )
// * Fouriertransformation auf Source anwenden und Ergebnis im Target ablegen
//
//-----
void FFT_Transformation ( workdatatype* _FFT_In_Real, workdatatype* _FFT_In_Imag,
workdatatype* _FFT_Out_Real, workdatatype* _FFT_Out_Imag )
{
    workdatatype pi = 3.14159265359;
    workdatatype xdum, ydum, s, c, angle, argument;
    int l, j, k, i;
    int n1, n2 = FFT_SIZE;
    int z1, z2;

    // Falls Output nicht im Eingabevektor muss _Source kopoert werden
    if ( _FFT_In_Real != _FFT_Out_Real )
    {
        for ( z1 = 0; z1<FFT_SIZE; z1++) _FFT_Out_Real [z1] = _FFT_In_Real [z1];
    };
    if ( _FFT_In_Imag != _FFT_Out_Imag )
    {
        for ( z1 = 0; z1<FFT_SIZE; z1++) _FFT_Out_Imag [z1] = _FFT_In_Imag [z1];
    };

    // Transformation durchfuehren
    for( z1 = 1; z1<=FFT_Potenz; z1++ )
    {
        n1 = n2;
        n2 /= 2;
        angle = 0;
        argument = 2*pi/n1;
        for( j = 0; j<n2; j++ ) {
            c = cos(angle);
            s = -sin(angle);
            for( i = j; i<FFT_SIZE; i+=n1 )
            {
                l = i+n2;
                xdum = _FFT_Out_Real [i] - _FFT_Out_Real [l];
                _FFT_Out_Real [i] = _FFT_Out_Real [i] + _FFT_Out_Real [l];

                ydum = _FFT_Out_Imag [i] - _FFT_Out_Imag [l];
                _FFT_Out_Imag [i] = _FFT_Out_Imag [i] + _FFT_Out_Imag [l];
            }
        }
    }
};

```

```

        _FFT_Out_Real [1] = xdum*c - ydum*s;
        _FFT_Out_Imag [1] = ydum*c + xdum*s;
    }
    angle = (j+1)*argument;
}
};

j = 0;

for( z2 = 0; z2<(FFT_SIZE-1); z2++ )
{
    if( z2<j )
    {
        xdum = _FFT_Out_Real [j];
        _FFT_Out_Real [j] = _FFT_Out_Real [z2];
        _FFT_Out_Real [z2] = xdum;

        ydum = _FFT_Out_Imag [j];
        _FFT_Out_Imag [j] = _FFT_Out_Imag [z2];
        _FFT_Out_Imag [z2] = ydum;
    }
    k = FFT_SIZE >> 1;

    while( k < (j+1) )
    {
        j -= k;
        k >>= 1;
    };

    j += k;
};

//-----
// void FFT_InversTransformation ( workdatatype* _IFFT_In_Real, workdatatype* _IFFT_In_Imag,
// workdatatype* _IFFT_Out_Real, workdatatype* _IFFT_Out_Imag )
// * Inverse Fouriertransformation auf Source anwenden und Ergebnis im
// Target ablegen
//-----
void FFT_InversTransformation ( workdatatype* _IFFT_In_Real, workdatatype* _IFFT_In_Imag,
workdatatype* _IFFT_Out_Real, workdatatype* _IFFT_Out_Imag )
{
    int z1, z2;

    // Falls Output nicht im Eingabevektor erfolgen soll, muss _Source
    // kopiert werden
    if ( _IFFT_In_Real != _IFFT_Out_Real )
    {
        for ( z1 = 0; z1<FFT_SIZE; z1++) _IFFT_Out_Real [z1] = _IFFT_In_Real [z1];
    };
    if ( _IFFT_In_Imag != _IFFT_Out_Imag )
    {
        for ( z1 = 0; z1<FFT_SIZE; z1++) _IFFT_Out_Imag [z1] = _IFFT_In_Imag [z1];
    };

    // Konjugiertkomplexe Werte ermitteln
    for ( z1 = 0; z1 < FFT_SIZE; z1++)
    {
        _IFFT_Out_Imag [z1] = _IFFT_Out_Imag [z1]*(-1);
    };

    // Transformation anwenden
    FFT_Transformation ( _IFFT_Out_Real, _IFFT_Out_Imag, _IFFT_Out_Real, _IFFT_Out_Imag );

    // IFFT wieder Normieren
    /* for( z2 = 0; z2<FFT_SIZE; z2++ )
    {
        _IFFT_Out_Real[z2] /= FFT_SIZE;
        _IFFT_Out_Imag[z2] /= FFT_SIZE;
    };*/
};

//-----
// void FFT_Init ( )
//
// * Variablen der Klasse initialisieren
//
//-----
void FFT_Init ( )
{
    int z1;

    FFT_Potenz = 0;

    // search the lowest 1 bit in n. if only one bit is set, like it should be,
    // Diese Funktion entspricht log2()
    for( z1 = FFT_SIZE; z1 > 0; z1 >>= 1) FFT_Potenz++;
};

```

```

};

//-----
// void Baender_Calculate_WerteImBand ()
//
// * Aufteilen des Spektrums in Baender
//
//-----
void Baender_Calculate_WerteImBand ()
{
    int Band;
    int WertNr;

    // Bestimmung, welche Werte in welchem Band liegen
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        float Frequenz = 0;
        WerteImBand [Band].Erster = -1;
        WerteImBand [Band].Letzter = -1;
        for ( WertNr = 0 ; WertNr<(FFT_SIZE>>1) ; WertNr++)
        {
            // Erster Werte des Bandes bestimmen
            if (WerteImBand [Band].Erster == -1)
            {
                if (Frequenz >= BAENDER [Band][0]) WerteImBand [Band].Erster = WertNr;
            };

            // Letzter Wert des Bandes bestimmen
            if (Frequenz < BAENDER [Band][1]) WerteImBand [Band].Letzter = WertNr;

            // naechste Frquenz errechnen
            Frequenz += Baender_Frequenzabstand;
        };

        WerteImBand [Band].Anzahl =
        WerteImBand [Band].Letzter - WerteImBand [Band].Erster + 1;
    };
};

//-----
// void Baender_Init ( int _Samplingfrequenz );
//
// * BaenderKlasse initialisieren mit allen Vorgabewerten
//
//-----
void Baender_Init ( int _Samplingfrequenz )
{
    Baender_Samplingfrequenz = _Samplingfrequenz;
    Baender_Frequenzabstand = Baender_Samplingfrequenz / FFT_SIZE;

    // Bandwerte bestimmen
    Baender_Calculate_WerteImBand ();
};

//-----
// void Baender_Get_BandAmplitudenMittelwerte ( workdatatype* _Spektrum_Real, workdatatype* _Spektrum_Imag, BandWerteStrukt *_BandMittelwerte)
//
// * Aus einem Frequenzspektrum werden die Bandmittelwerte beechnet
//
//-----
void Baender_Get_BandAmplitudenMittelwerte ( workdatatype* _Spektrum_Real, workdatatype* _Spektrum_Imag, struct BandWerteStrukt *_BandMittelwerte)
{
    int Band;
    int WertNr;

    // Baender durchzaehlen
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        // Amplitudenwerte eines Bande quadratisch addieren
        for ( WertNr = WerteImBand [Band].Erster ; WertNr<=WerteImBand [Band].Letzter ; WertNr++)
        {
            (*_BandMittelwerte).Wert [Band] += (_Spektrum_Real [WertNr])*(_Spektrum_Real [WertNr])+(_Spektrum_Imag [WertNr])*(_Spektrum_Imag [WertNr]);
        };

        // Wurzel ziehen
        (*_BandMittelwerte).Wert [Band] = sqrt ((*_BandMittelwerte).Wert [Band]/WerteImBand [Band].Anzahl);
    };
};

//-----
// void Baender_Get_BandMinimalWerte (struct BandWerteStrukt *_BandMittelwerte,
// struct BandWerteStrukt *_BandMinimalWerte)
// * Bestimmen der neuen Bandmaximalwerte aus den Bandmittelwerten und früheren Bandmaximalwerten
//
//-----
void Baender_Get_BandMaximalWerte (struct BandWerteStrukt *_BandMittelwerte,
struct BandWerteStrukt *_BandMaximalWerte)
{
    int Band;

```

```

// Baender durchzaehlen
for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
{
    if ((*_BandMaximalWerte).Wert [Band] *Absinkfaktor<= (*_BandMittelwerte).Wert [Band])
    {
        (*_BandMaximalWerte).Wert [Band] = (*_BandMittelwerte).Wert [Band];
    }
    else
    {
        (*_BandMaximalWerte).Wert [Band] = (*_BandMaximalWerte).Wert [Band]*Absinkfaktor;
    }
};
};

//-----
// void Baender_Get_BandMinimalWerte (struct BandWerteStrukt *_BandMittelwerte,
// struct BandWerteStrukt *_BandMinimalWerte)
// * Bestimmen der neuen Bandminimalwerte aus den Bandmittelwerten und früheren Bandminimalwerten
//-----
void Baender_Get_BandMinimalWerte (struct BandWerteStrukt *_BandMittelwerte,
    struct BandWerteStrukt *_BandMinimalWerte)
{
    int Band;

    // Baender durchzaehlen
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        if ((*_BandMinimalWerte).Wert [Band] *Anstiegsfaktor > (*_BandMittelwerte).Wert [Band])
        {
            (*_BandMinimalWerte).Wert [Band] = (*_BandMittelwerte).Wert [Band];
        }
        else
        {
            (*_BandMinimalWerte).Wert [Band] = (*_BandMinimalWerte).Wert [Band]*Anstiegsfaktor;
        }
    };
};

//-----
// void Baender_Get_BandDynamik (struct BandWerteStrukt *_BandMaximalWerte,
// struct BandWerteStrukt *_BandMinimalWerte, struct BandWerteStrukt *_BandDynamikWerte)
// * Bestimmen der Banddynamik aus Maximal- und Minimalwerten
//-----
void Baender_Get_BandDynamik (struct BandWerteStrukt *_BandMaximalWerte,
    struct BandWerteStrukt *_BandMinimalWerte, struct BandWerteStrukt *_BandDynamikWerte)
{
    int Band;

    // Baender durchzaehlen
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        // Dynamik errechnen
        (*_BandDynamikWerte).Wert [Band] = 20 * log10((*_BandMaximalWerte).Wert [Band] / (*_BandMinimalWerte).Wert [Band]);
    };
};

//-----
// void Baender_Get_BandFaktoren (struct BandWerteStrukt *_BandDynamikWerte,
// struct BandWerteStrukt *_BandFaktorenWerte)
// * Bestimmen der Bandfaktoren aus der Dynamik des Bandes (Dynamik-Daempfung-Kennlinie)
//-----
void Baender_Get_BandFaktoren (struct BandWerteStrukt *_BandDynamikWerte,
    struct BandWerteStrukt *_BandFaktorenWerte)
{
    int Band;

    // Baender durchzaehlen
    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        // Bandfaktoren berechnen
        (*_BandFaktorenWerte).Wert [Band] = 1 / pow (10, 0.05* (MaxDaempfung - ((float)MaxDaempfung / (float)MaxDynamik)
            * MIN (MaxDynamik, (*_BandDynamikWerte).Wert [Band])));
    };
};

//-----
// void Baender_Get_SpektralFaktoren ( struct BandWerteStrukt *_BandFaktorenWerte, double *_SpektralFaktoren )
// * Fuer jeden Wert des Spektrums wird sein Spektralfaktor berechnet
//-----
void Baender_Get_SpektralFaktoren ( struct BandWerteStrukt *_BandFaktorenWerte, spectrumfactortyp *_SpektralFaktoren )
{
    int Band;

```

## ANHANG A. LISTING : DSP PROGRAMM *Mehrkanalige Störgeräuschunterdrückung*

```

int WertNr;

// Baender durchzaehlen
for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
{
    // Werte der ersten Haelfte des Spektrums
    for ( WertNr = WerteImBand [Band].Erster ; WertNr<=WerteImBand [Band].Letzter ; WertNr++)
    {
        _SpektralFaktoren [WertNr] = (*_BandFaktorenWerte).Wert [Band];
    };
    // Werte der zweiten Haelfte des Spektrums
    for ( WertNr = (FFT_SIZE - WerteImBand [Band].Letzter - 1) ; WertNr<=(FFT_SIZE - WerteImBand [Band].Erster - 1) ; WertNr++)
    {
        _SpektralFaktoren [WertNr] = (*_BandFaktorenWerte).Wert [Band];
    };
};

//-----
// void Baender_Smooth_SpektralFaktoren ( spectrumfactortyp* _SpektralFaktorenIn, spectrumfactortyp* _SpektralFaktorenOut, unsigned int _Glaettungsfaktor )
//
// * Der Verlauf der Spektralfaktoren wird abgerundet
//
//-----
void Baender_Smooth_SpektralFaktoren ( spectrumfactortyp* _SpektralFaktorenIn, spectrumfactortyp* _SpektralFaktorenOut, unsigned int _Glaettungsfaktor )
{
    double Summenwert;
    int WertNr;
    int WertAdd;

    // Eingabevektor kopieren
    for ( WertNr = 0 ; WertNr < FFT_SIZE ; WertNr++ )
    {
        _SpektralFaktorenOut [WertNr] = _SpektralFaktorenIn [WertNr];
    };

    // Formel des Glaettungsfaktors anwenden
    for ( WertNr = _Glaettungsfaktor ; WertNr < ( FFT_SIZE - _Glaettungsfaktor ) ; WertNr++ )
    {
        Summenwert = 0;
        for ( WertAdd = WertNr - _Glaettungsfaktor ; WertAdd <= ( WertNr + _Glaettungsfaktor ) ; WertAdd++)
        {
            Summenwert += _SpektralFaktorenIn [WertAdd];
        };
        _SpektralFaktorenOut [WertNr] = Summenwert/((_Glaettungsfaktor<<1)+1);
    };
};

//-----
// void Baender_Calculate_Spektrum ( spectrumfactortyp* _SpektralFaktoren, workdatatype* _Spektrum_In_Real,
// workdatatype* _Spektrum_In_Imag, workdatatype* _Spektrum_Out_Real, workdatatype* _Spektrum_Out_Imag)
// * Ein Frequenzspektrum wird mit Spektralfaktoren multipliziert
//
//-----
void Baender_Calculate_Spektrum ( spectrumfactortyp* _SpektralFaktoren, workdatatype* _Spektrum_In_Real,
workdatatype* _Spektrum_In_Imag, workdatatype* _Spektrum_Out_Real, workdatatype* _Spektrum_Out_Imag)
{
    int WertNr;

    // Spektralwerte durchzaehlen
    for ( WertNr = 0 ; WertNr < FFT_SIZE ; WertNr++ )
    {
        _Spektrum_Out_Real [WertNr] = _SpektralFaktoren [WertNr]*_Spektrum_In_Real [WertNr];
        _Spektrum_Out_Imag [WertNr] = _SpektralFaktoren [WertNr]*_Spektrum_In_Imag [WertNr];
    };
};

//-----
// void Baender_Preset_BandWerte (struct BandWerteStrukt *_BandWerte, double _Wert)
//
// * Daten im BandWerteStrukt koennen vorgesetzt werden
//
//-----
void Baender_Preset_BandWerte (struct BandWerteStrukt *_BandWerte, double _Wert)
{
    int Band;

    for ( Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
    {
        (*_BandWerte).Wert [Band] = _Wert;
    };
};

/*-----*/
/* serial port receive DMA complete */
void spr0_asserted( int sig_num )
{

```

```

int temp;
int index;
int Channel;

#ifdef TIMER_ON
// Timer initialisieren
timer_set (0xff, 0xffffffff);
timer_on ();
#endif

/* find pointer to actual receive and transmit buffer */
rx_ptr_a = fmodf(rx_ptr_a, 3.0); // Pointer to the newer DMA Read Buffer
rx_ptr_b = fmodf((rx_ptr_a+2), 3.0); // Pointer to the oldest DMA Read Buffer
tx_ptr_a = fmodf((rx_ptr_a+2), 3.0); // Pointer to the next DMA Write Buffer
tx_ptr_b = fmodf((rx_ptr_a), 3.0); // Pointer to the 2nd DMA Write Buffer

for ( Channel = 1; Channel<=DSP_CHANNELS; Channel++)
{

/* fill in samples and normalize to volts */
if ( Channel == 1)
{
for (index=0; index<HALF_FFT_SIZE; index++)
{
convertleft16to32 (temp,rx0_buf[rx_ptr_a][index]);
workvar[index + HALF_FFT_SIZE] = (NORM * ((float)(temp)));
convertleft16to32 (temp,rx0_buf[rx_ptr_b][index]);
workvar[index] = (NORM * ((float)(temp)));
}
}
else
{
for (index=0; index<HALF_FFT_SIZE; index++)
{
convertright16to32 (temp,rx0_buf[rx_ptr_a][index]);
workvar[index + HALF_FFT_SIZE] = (NORM * ((float)(temp)));
convertright16to32 (temp,rx0_buf[rx_ptr_b][index]);
workvar[index] = (NORM * ((float)(temp)));
}
}
};

Fenster_UseWindowfactors ( workvar , workvar );

// compute FFT
rfft(workvar, FFT_rout, FFT_iout);
//FFT_Transformation ( workvar, zeroarray, FFT_rout, FFT_iout );

// Berechnung der Bandmittelwerte
Baender_Get_BandAmplitudenMittelwerte (FFT_rout,FFT_iout,&BandMittelWerte);

// Berechnung der Bandmaximal -und Bandminimalwerte
Baender_Get_BandMaximalWerte (&BandMittelWerte, &BandMaximalWerte [Channel-1]);
Baender_Get_BandMinimalWerte (&BandMittelWerte, &BandMinimalWerte [Channel-1]);

// Berechnung der Dynamik
Baender_Get_BandDynamik ( &BandMaximalWerte [Channel-1], &BandMinimalWerte [Channel-1], &BandDynamikWerte);

// Berechnung der Bandfaktoren
Baender_Get_BandFaktoren ( &BandDynamikWerte, &BandFaktorenWerte);

// Spektralfaktoren berechnen
Baender_Get_Spektralfaktoren ( &BandFaktorenWerte, Spektralfaktoren );

// Spektralfaktoren abschwächen
//Baender_Smooth_Spektralfaktoren ( Spektralfaktoren, Spektralfaktoren, SMOOTHFAKTOR );

if (Bypass == 0)
{
// Neues manipuliertes Spektrum berechnen
Baender_Calculate_Spektrum ( Spektralfaktoren, FFT_rout, FFT_iout, IFFT_rin, IFFT_iin );
}
else
{
for( index = 0; index < FFT_SIZE; index++)
{
IFFT_rin[index] = FFT_rout[index];
IFFT_iin[index] = FFT_iout[index];
}
}

#endif PC_WATCH

```

## ANHANG A. LISTING : DSP PROGRAMM *Mehrkanalige Störgeräuschunterdrückung*

```

    if (Channel == 1)
    {
        CIRC_WRITE(Huellkurven,1,BandMinimalWerte [0].Wert [Watch_Bandnummer],dm);

        CIRC_WRITE(Huellkurven,1,BandMaximalWerte [0].Wert [Watch_Bandnummer],dm);

        CIRC_WRITE(Huellkurven,1,BandMittelWerte.Wert [Watch_Bandnummer],dm);
    };

#endif

    // compute IFFT
    ifft(IFFT_rin, IFFT_iin, IFFT_rout, IFFT_iout);
    //FFT_InversTransformation ( FFT_rout, FFT_iout, IFFT_rout, IFFT_iout );

    if (Channel == 1) Fenster_UseWindowfactors ( IFFT_rout, outvar [0] );
    else Fenster_UseWindowfactors ( IFFT_rout, outvar [1] );

}

/* fill chained DMA transmit buffer with result */
for (index=0; index<HALF_FFT_SIZE; index++)
{
    tx0_buf[tx_ptrn][index] = (((unsigned)(pastoutvar[1][index] + (outvar[1][index]*CORR) & 0x0000ffff) | ((unsigned)(pastoutvar[0][index]
    + (outvar[0][index]*CORR))<<16)&0xffff0000);
    pastoutvar[0][index] = (outvar[0][index + HALF_FFT_SIZE]*CORR);
    pastoutvar[1][index] = (outvar[1][index + HALF_FFT_SIZE]*CORR);
}

// übertragung zum PC per Interrupt
if( PC_Ready == 1)
{

#ifdef PC_WATCH
for (index=0; index<23; index++)
{
    Watch_Bandmittel [index] = BandMittelWerte.Wert [index];
    Watch_Banddynamik [index] = BandDynamikWerte.Wert [index];
    Watch_Bandfaktoren [index] = BandFaktorenWerte.Wert [index];
}

for (index=0; index<FFT_SIZE; index++)
{
    Watch_SpektralFaktoren [index] = SpektralFaktoren [index];
}

for (index=0; index<FFT_SIZE; index++)
{
    Watch_FFT_rout [index] = FFT_rout[index];
    Watch_FFT_iout [index] = FFT_iout[index];

    Watch_IFFT_rin [index] = IFFT_rin[index];
    Watch_IFFT_iin [index] = IFFT_iin[index];
}

for (index=0; index<200; index++)
{
    CIRC_READ(Huellkurven,1,Watch_Minkurve [index],dm);
    CIRC_READ(Huellkurven,1,Watch_Maxkurve [index],dm);
    CIRC_READ(Huellkurven,1,Watch_Istkurve [index],dm);
    //Watch_Maxkurve [index] = Tester;
}
#endif

// Clear local flag and respond.
generate_dsp_to_pc_interrupt( 0 );
PC_Ready = 0;
}

#ifdef TIMER_ON
// Timer stoppen und Dauer in Variable ablegen
et = 30.30e-9*(0xffffffff-timer_off());
#endif

/* increment pointer to next buffer address */
rx_ptrn++;

}

/*-----*/
void setup_sport0( void )
{

```

```

/* Configure SHARC serial port */

/* TRANSMIT CONTROL REGISTER */
sport0_iop.txc.mdf = 0; /* multichannel frame delay (MFD) */
sport0_iop.txc.schen = 1; /* Tx DMA chaining enable */
sport0_iop.txc.sden = 1; /* Tx DMA enable */
sport0_iop.txc.lafs = 0; /* Late TFS (alternate) */
sport0_iop.txc.ltfs = 1; /* Active low TFS */
sport0_iop.txc.ditfs = 0; /* Data independent TFS */
sport0_iop.txc.itfs = 0; /* Internally generated TFS */
sport0_iop.txc.tfsr = 1; /* TFS Required */

sport0_iop.txc.ckre = 0; /* Data and FS on clock rising edge */
sport0_iop.txc.gclk = 0; /* Enable clock only during transmission*/
sport0_iop.txc.iclk = 0; /* Internally generated Tx clock */
sport0_iop.txc.pack = 0; /* Unpack 32b words into two 16b tx's */

sport0_iop.txc.slen = 31; /* Data word length minus one */
sport0_iop.txc.sendn = 0; /* Data word endian 1 = LSB first */
sport0_iop.txc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
/* Data type specifier */
sport0_iop.txc.spen = 1; /* Enable (clear for MC operation) */

/* RECEIVE CONTROL REGISTER */
sport0_iop.rxc.nch = 31; /* multichannel number of channels - 1 */
sport0_iop.rxc.mce = 0; /* multichannel enable */
sport0_iop.rxc.spl = 0; /* Loop back configure (test) */
sport0_iop.rxc.d2dma = 0; /* Enable 2-dimensional DMA array */
sport0_iop.rxc.schen = 1; /* Rx DMA chaining enable */
sport0_iop.rxc.sden = 1; /* Rx DMA enable */
sport0_iop.rxc.lafs = 0; /* Late RFS (alternate) */
sport0_iop.rxc.ltfs = 1; /* Active low RFS */
sport0_iop.rxc.irfs = 0; /* Internally generated RFS */
sport0_iop.rxc.rfsr = 1; /* RFS Required */
sport0_iop.rxc.ckre = 0; /* Data and FS on clock rising edge */
sport0_iop.rxc.gclk = 0; /* Enable clock only during transmission*/
sport0_iop.rxc.iclk = 0; /* Internally generated Rx clock */
sport0_iop.rxc.pack = 0; /* Pack two 16b rx's into 32b word */

sport0_iop.rxc.slen = 31; /* Data word length minus one */
sport0_iop.rxc.sendn = 0; /* Data word endian 1 = LSB first */
sport0_iop.rxc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
/* Data type specifier */
sport0_iop.rxc.spen = 1; /* Enable (clear for MC operation) */

/* Enable sport0 xmit & rcv irq's */
interruptf(SIG_SPROI, spr0_asserted);

/* Set up Transmit Transfer Control Block for chained DMA */
tx0_tcb[0].ii = tx0_buf[0]; /* DMA source buffer address */
tx0_tcb[1].ii = tx0_buf[1]; /* DMA source buffer address */
tx0_tcb[2].ii = tx0_buf[2]; /* DMA source buffer address */

tx0_tcb[0].cp = &tx0_tcb[1].ii; /* define ptr to next TCB (point to self) */
tx0_tcb[1].cp = &tx0_tcb[2].ii; /* define ptr to next TCB (point to self) */
tx0_tcb[2].cp = &tx0_tcb[0].ii; /* define ptr to next TCB (point to self) */

SetIOP(CP2, (((int)&tx0_tcb[0].ii) & CP_MAF) | CP_PCI);
/* define ptr to current TCB (kick off DMA) */
/* (SPORT0 transmit uses DMA ch 2) */

/* Set up Receive Transfer Control Block for chained DMA */
rx0_tcb[0].ii = rx0_buf[0]; /* DMA destination buffer address */
rx0_tcb[1].ii = rx0_buf[1]; /* DMA destination buffer address */
rx0_tcb[2].ii = rx0_buf[2]; /* DMA destination buffer address */

rx0_tcb[0].cp = &rx0_tcb[1].ii; /* define ptr to next TCB (point to self) */
rx0_tcb[1].cp = &rx0_tcb[2].ii; /* define ptr to next TCB (point to self) */
rx0_tcb[2].cp = &rx0_tcb[0].ii; /* define ptr to next TCB (point to self) */

SetIOP(CP0, (((int)&rx0_tcb[0].ii) & CP_MAF) | CP_PCI);
/* define ptr to current TCB (kick off DMA) */
/* (SPORT0 receive uses DMA ch 0) */
}

/*-----*/

void setup_bitsibb( void )
{
    BITSIBB_TIMO = BB_TIM(fc);
    BITSIBB_CTL = 1; /* Enable only one D/A */
    BITSIBB_CLKSEL = 0x0844; /* Ch 1-2 Tim 0, Ch 3-4 Tim 1 */
}

/*****
//

```

## ANHANG A. LISTING : DSP PROGRAMM *Mehrkanalige Störgeräuschunterdrückung*

```

// input: external_bank_number is 0-4.
//         0-3 select memory regions 0-3, 4 selects unbanked memory.
//
/*****
#define MSIZE_BITS 0x0000F000
#define MSIZE_SHIFT 12
#define MSO_BASE 0x00400000
#define MSIZE0_SIZE 0x00002000
int *get_dms_base( int msnun )
{
    int msize;
    int *dms;

    // Get MSIZE.
    msize = ((GetIOP(SYSICON) & MSIZE_BITS) >> MSIZE_SHIFT);

    // Get offset to correct base memory.
    dms = (int *) (MSO_BASE + (msnun * (MSIZE0_SIZE << msize)));

    return( dms );
}

void setup_pc_to_dsp_interrupt( void )
{
    // Make IRQ-0 edge-sensitive.
    asm("BIT SET MODE2 0x00000001;");

    // Get base location of PCI control registers, in External Memory Bank 2.
    ms2_ptr = get_dms_base( 2 );

    // Clear all PCI to local doorbell interrupts.
    *(ms2_ptr + DSP_PLX_PCI_BELL) = 0xffffffffL;
}

void generate_dsp_to_pc_interrupt( int data )
{
    // Set local to PCI doorbell.
    *(ms2_ptr + DSP_PLX_L0C_BELL) = 1;
}

void clear_pc_to_dsp_interrupt( void )
{
    // Clear all PCI to local doorbell interrupts.
    *(ms2_ptr + DSP_PLX_PCI_BELL) = 0xffffffffL;
}

void irq0_interrupt_handler(int irq_num)
{
    // Clear the interrupt.
    clear_pc_to_dsp_interrupt();

    // Count number of interrupts received from PC.
    rcvd_ints++;

    // Indicate we were interrupted.
    interrupt_rcvd = TRUE;
}

/*****
void main ( void )
{
    int t, msize;

    MaxDaempfung = 30;
    MaxDynamik = 30;
    Absinkfaktor = 0.99;
    Anstiegsfaktor = 1.01;

    // Initialisierung der Read und Write - Buffer
    for (t = 0;t<HALF_FFT_SIZE;t++)
    {
        rx0_buf[0][t] = 0;
        rx0_buf[1][t] = 0;
        rx0_buf[2][t] = 0;
        tx0_buf[0][t] = 0;
        tx0_buf[1][t] = 0;
        tx0_buf[2][t] = 0;
    };

    // Erstellen eines leeren Arrays
    for (t = 0;t<FFT_SIZE;t++)
    {
        zeroarray [t]=0;
    }
}

```

## Mehrkanalige Störgeräuschunterdrückung ANHANG A. LISTING : DSP PROGRAMM

```
};

//-----Circular Buffer-----

BASE(Huellkurven) = CircularData; // das Array "CircularData" dem Circularbuffer "IstkurveC" zuweisen
LENGTH(Huellkurven) = Circ_Buf_Size;

//-----

/* GET BASE LOCATION OF DMS3 (BITS1) */
init_ms_bases(); /* Initialize memory select bases */

/* initialize hardware */
setup_bitsibb();
setup_sport0();

//-----PC-DSP & DSP-PC -Interrupts -----
// Initialize data.
rcvd_ints      = 0;
PC_Ready      = 0;
interrupt_rcvd = FALSE;

// Program SHARC to receive interrupts from the PC.
// (Must be done before interrupt handler set up).
setup_pc_to_dsp_interrupt();

// Set up interrupt handlers.
interrupt(SIG_IRQ0, irq0_interrupt_handler);

//-----

Fenster_Create_Windowfactors ( HALBSINUS );
FFT_Init ( );
Baender_Init ( fc );

// Startwerte setzten, so dass der Effekt von Beginn an zu wirken beginnt
Baender_Preset_BandWerte (&BandMinimalWerte [0],1000);
Baender_Preset_BandWerte (&BandMinimalWerte [1],1000);
Baender_Preset_BandWerte (&BandMaximalWerte [0],-1000);
Baender_Preset_BandWerte (&BandMaximalWerte [1],-1000);

while(1)
{
    idle();
}
}
```



## Anhang B Listing : DSP Steuerung

---

### B.1 Problem Domain

#### B.1.1 DSPCmd.h

```
// DSP Comandos

#ifndef __DSPCMD
#define __DSPCMD

#include <exception>
#include "dsp21k.h"

#include "observer.h"

// #define NO_DSP
#define BOARDNUMMER 0

// Exception der DSP Kommandoklasse
class DSP_Cmd_Exception : public exception
{
public :
    DSP_Cmd_Exception (int _DSP_Cmd_Error) throw ();
    virtual const char *what() const throw ();

private :
    int DSP_Cmd_Error;
};

// DSP Commandoklasse
class DSP_Cmd : public SubjektKlasse
{
public:
    DSP_Cmd ();
    ~DSP_Cmd ();

    void BoardInit ();
    void BoardClose ();
    void BoardReset ();
    void BoardProgStart ();

    static void __cdecl BoardISRFunc (ULONG data, ...);

    void ReadyForData ();

    void Set_MaxDaempfung (int _MaxDaempfung);
    void Set_MaxDynamik (int _MaxDynamik);
    void Set_Absinkfaktor (float _Absinkfaktor);
    void Set_Anstiegsfaktor (float _Anstiegsfaktor);
    void Set_Bandnummer (int _Bandnummer);
    void Set_Bypass (int _Bypass);

    void Get_Eingangsspektrum (float *Spec_Real, float *Spec_Imag);
    void Get_Ausgangsspektrum (float *Spec_Real, float *Spec_Imag);
    void Get_SpektralFaktoren (float *_SpektralFaktoren);

    void Get_BandMittelWerte (float *_BandWerte);
    void Get_BandDynamik (float *_BandWerte);
    void Get_BandFaktoren (float *_BandWerte);

    void Get_HuellkurveMax (float *_HuellkurveMax);
    void Get_HuellkurveIst (float *_HuellkurveIst);
    void Get_HuellkurveMin (float *_HuellkurveMin);

    int Get_Samplingfrequenz ();

private:
    bool Board_Initialized;
    bool Board_ProgStarted;
    PDSP21K Board;
};

#endif
```

## B.1.2 DSPCmd.cpp

```

// DSP Comandos

#include "stdafx.h"

#include <math.h>

#include "defs.h"
#include "dspcmd.h"

extern DSP_Cmd* DSP_Command;

DSP_Cmd_Exception::DSP_Cmd_Exception (int _DSP_Cmd_Error)
{
    DSP_Cmd_Error = _DSP_Cmd_Error;
};

const char* DSP_Cmd_Exception::what() const
{
    char* ErrorMessage;
    switch (DSP_Cmd_Error)
    {
        case 1 : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : Adresse auf DSP nicht gefunden";break;
        case 2 : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : Fehler beim Boardreset aufgetreten";break;
        case 3 : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : Programm kann nicht gestartet werden";break;
        case 4 : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : Board kann nicht konfiguriert werden";break;
        case 5 : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : Fehler beim Programmdownload";break;
        case 6 : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : Fehler beim Schliessen des Boards";break;

        default : ErrorMessage = "Ausnahmefehler der DSP_Cmd_Klasse : unbekannter Fehler";
    };
    return (ErrorMessage);
};

DSP_Cmd::DSP_Cmd ()
{
    Board_Initialized = false;
};

DSP_Cmd::~DSP_Cmd ()
{
    if (Board_Initialized)
    {
        BoardClose ();
    };
};

void DSP_Cmd::BoardInit ()
{
    {
        #ifndef NO_DSP
            if (Board_Initialized == false)
            {
                Board = dsp21k_open (BOARDNUMMER);
                Board_Initialized = true;
            };
        #endif
    };
};

void DSP_Cmd::BoardClose ()
{
    {
        if (Board_Initialized)
        {
            dsp21k_set_interrupt_function (Board, NULL);
            if (dsp21k_close (Board) != 0) throw (DSP_Cmd_Exception (6));
        };
    };
};

void DSP_Cmd::BoardReset ()
{
    {
        if (Board_Initialized)
        {
            dsp21k_set_interrupt_function (Board, NULL);
            if (dsp21k_reset_bd (Board) == false) throw (DSP_Cmd_Exception (2));
            if (dsp21k_cfg_proc(Board) == false) throw (DSP_Cmd_Exception (4));
        };
    };
};

void DSP_Cmd::BoardProgStart ()
{
    {
        if (Board_Initialized)
        {
            if (dsp21k_dl_exe (Board, "noise.21k") == false) throw (DSP_Cmd_Exception (5));
            if (dsp21k_start (Board) == false) throw (DSP_Cmd_Exception (3));
        };
    };
};

```

```

else ReadyForData ();
    dsp21k_set_interrupt_function (Board, BoardISRFunc);
};
};

void __cdecl DSP_Cmd::BoardISRFunc (ULONG data, ...)
{
#ifdef NO_DSP
    DSP_Command->Benachrichtige ();
    Sleep (100);
    DSP_Command->ReadyForData ();
#endif
};

void DSP_Cmd::ReadyForData ()
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_PC_Ready");
        if (Adresse != -1) dsp21k_dl_int( Board, Adresse, 1);
        else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Set_MaxDaempfung (int _MaxDaempfung)
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_MaxDaempfung");
        if (Adresse != -1) dsp21k_dl_int( Board, Adresse, _MaxDaempfung);
        else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Set_MaxDynamik (int _MaxDynamik)
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_MaxDynamik");
        if (Adresse != -1) dsp21k_dl_int( Board, Adresse, _MaxDynamik);
        else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Set_Anstiegsfaktor (float _Anstiegsfaktor)
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Anstiegsfaktor");
        if (Adresse != -1) dsp21k_dlflt( Board, Adresse, _Anstiegsfaktor);
        else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Set_Absinkfaktor (float _Absinkfaktor)
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Absinkfaktor");
        if (Adresse != -1) dsp21k_dlflt( Board, Adresse, _Absinkfaktor);
        else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Set_Bandnummer (int _Bandnummer)
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_Bandnummer");
        if (Adresse != -1) dsp21k_dl_int( Board, Adresse, _Bandnummer);
        else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Set_Bypass (int _Bypass)
{
    ULONG Adresse;
    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Bypass");
        if (Adresse != -1) dsp21k_dl_int( Board, Adresse, _Bypass);
        else throw (DSP_Cmd_Exception (1));
    };
};

```

```

};
};

void DSP_Cmd::Get_Eingangsspektrum (float *Spec_Real, float *Spec_Imag)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_FFT_rout");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, FFT_SIZE, Spec_Real);
    else throw (DSP_Cmd_Exception (1));

        Adresse = dsp21k_find_label(Board, "_Watch_FFT_iout");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, FFT_SIZE, Spec_Imag);
    else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Get_Ausgangsspektrum (float *Spec_Real, float *Spec_Imag)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_IFFT_rin");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, FFT_SIZE, Spec_Real);
    else throw (DSP_Cmd_Exception (1));

        Adresse = dsp21k_find_label(Board, "_Watch_IFFT_iin");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, FFT_SIZE, Spec_Imag);
    else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Get_SpektralFaktoren (float *_SpektralFaktoren)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_SpektralFaktoren");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, FFT_SIZE, _SpektralFaktoren);
    else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Get_BandMittelWerte (float *_BandWerte)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_Bandmittel");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, ANZAHL_BAENDER, _BandWerte);
    else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Get_BandDynamik (float *_BandWerte)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_Banddynamik");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, ANZAHL_BAENDER, _BandWerte);
    else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Get_BandFaktoren (float *_BandWerte)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_Bandfaktoren");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, ANZAHL_BAENDER, _BandWerte);
    else throw (DSP_Cmd_Exception (1));
    };
};

void DSP_Cmd::Get_HuellkurveMax (float *_HuellkurveMax)
{
    ULONG Adresse;

    if (Board_Initialized)

```

```
{
    Adresse = dsp21k_find_label(Board, "_Watch_Maxkurve");
    if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, ANZAHL_HUELLKURVENWERTE, _HuellkurveMax);
else throw (DSP_Cmd_Exception (1));
};
};

void DSP_Cmd::Get_HuellkurveIst (float *_HuellkurveIst)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_Istkurve");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, ANZAHL_HUELLKURVENWERTE, _HuellkurveIst);
else throw (DSP_Cmd_Exception (1));
};
};

void DSP_Cmd::Get_HuellkurveMin (float *_HuellkurveMin)
{
    ULONG Adresse;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_Minkurve");
        if (Adresse != -1) dsp21k_ul_flts( Board, Adresse, ANZAHL_HUELLKURVENWERTE, _HuellkurveMin);
else throw (DSP_Cmd_Exception (1));
};
};

int DSP_Cmd::Get_Samplingfrequenz ()
{
    ULONG Adresse;
    int returnvalue;

    if (Board_Initialized)
    {
        Adresse = dsp21k_find_label(Board, "_Watch_fc");
        if (Adresse != -1) returnvalue = dsp21k_ul_int( Board, Adresse);
else throw (DSP_Cmd_Exception (1));
};
    return (returnvalue);
};
```

## B.2 Observer Pattern

### B.2.1 Observer.h

```

/*
*****
* Observer.h *
*****
* Definitionen zum Observer-Pattern *
* * *
* * *
*****
* Geändert am : 02.07.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/
#ifndef __OBSERVER
#define __OBSERVER

class SubjektKlasse;

// Definitionen des Beobachter-Basisklasse
class BeobachterKlasse
{
protected:
    BeobachterKlasse ();

public:
    virtual ~BeobachterKlasse ();

    virtual void Aktualisiere (SubjektKlasse* _Subjekt) = 0;
};

// Definitionen des Subjekt-Basisklasse
class SubjektKlasse
{
protected:
    SubjektKlasse ();

public:
    virtual ~SubjektKlasse ();

    virtual void MeldeAn (BeobachterKlasse* _Beobachter);
    virtual void MeldeAb (BeobachterKlasse* _Beobachter);
    virtual void Benachrichtige ();

private:
    // Liste mit den Beobachtern (max 10)
    BeobachterKlasse* BeobachterListe [10];
    int AnzahlBeobachter;
};

#endif

```

## B.2.2 Observer.cpp

```

/*
*****
* Observer.cpp
*****
* Definitionen zum Observer-Pattern
*
*
*
*****
* Geändert am : 02.07.1999
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud
*****
*/
#include "stdafx.h"

#include <iostream.h>
#include "Observer.h"

// Definitionen des Beobachter-Basisklasse
BeobachterKlasse::BeobachterKlasse ()
{
};

BeobachterKlasse::~BeobachterKlasse ()
{
};

void BeobachterKlasse::Aktualisiere (SubjektKlasse* _Subjekt)
{
};

// Definitionen des Subjekt-Basisklasse
SubjektKlasse::SubjektKlasse ()
{
    // Observerliste reseten
    AnzahlBeobachter = 0;
};

SubjektKlasse::~SubjektKlasse ()
{
};

//-----
// MeldeAn (BeobachterKlasse* _Beobachter)
//
// * Anmelden eines Beobachters
//
//-----
void SubjektKlasse::MeldeAn (BeobachterKlasse* _Beobachter)
{
    // Neuen Beobachter beim Anmelden
    BeobachterListe [AnzahlBeobachter]=_Beobachter;
    AnzahlBeobachter++;
};

//-----
// MeldeAb (BeobachterKlasse* _Beobachter)
//
// * Abmelden eines Beobachters
//
//-----
void SubjektKlasse::MeldeAb (BeobachterKlasse* _Beobachter)
{
    if (AnzahlBeobachter > 0)
    {
        // Liste nach Observer durchsuchen und Loeschen
        int z1 = 0;
        while (z1 < AnzahlBeobachter)
        {
            if (BeobachterListe [z1]==_Beobachter)
            {
                // Observer aus Liste entfernen und leeren Platz mit letztem
                // Element fuehlen
                AnzahlBeobachter--;
                BeobachterListe [z1] = BeobachterListe[AnzahlBeobachter];
                // While-Schleife beenden
                z1 = AnzahlBeobachter;
            };
            z1++;
        };
    };
};

//-----
// Benachrichtige ()

```

```
//  
// * Alle Beobachter benachrichtigen  
//  
//-----  
void SubjektKlasse::Benachrichtige ()  
{  
    // Alle Observer benachrichtigen  
    for (int z1=0;z1<AnzahlBeobachter;z1++)  
        BeobachterListe [z1]->Aktualisiere (this);  
};
```

### B.2.3 SpektrumView.h

```

/*
*****
* SpektrumView.h *
*****
* Beobachter der NoiseEraseKlasse *
* Zeigt Werte des Spektrums an *
* *
*****
* Geändert am : 04.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/
#ifndef __SpektrumView
#define __SpektrumView

#include "Defs.h"
#include "Observer.h"
#include "dspcmd.h"

typedef float Spektrumtype;

// Definition der SpektrumViewKlasse
class SpektrumViewKlasse : public CFrameWnd, BeobachterKlasse
{
public:
    SpektrumViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _SpektrumViewActive, HICON _Icon);

    virtual ~SpektrumViewKlasse ();

    virtual void Aktualisiere (SubjektKlasse* _Subjekt);

private:
    void DrawSpektrum ();
    void PlotSpektrum (CDC* _SpektrumViewDC, int _x1, int _y1, int _x2, int _y2, Spektrumtype* _Spektrum, int _AnzWerte, COLORREF _Color );
    void GetLabelText (char* _Text, double _Value, Spektrumtype _MaxValue);

    // Ressourcenvariablen
    HICON WinIcon;

    // View variablen
    DSP_Cmd* View_DSP_Command;
    bool* SpektrumViewActive;
    int DataSource; // Kennziffer der Datenquelle :
    // 1 : Eingangsspektrum
    // 2 : Ausgangsspektrum
    // 3 : Spektralfaktoren
    CDC* Zeichnungsfeld;
    CBitmap* Zeichnungsfeld_Bitmap;

    // Daten variablen
    int Samplingfrequenz;
    Spektrumtype SpektrumReal [FFT_SIZE];
    Spektrumtype SpektrumImag [FFT_SIZE];
    Spektrumtype SpektrumAbs [FFT_SIZE];

    // Generated message map functions
    //{{AFX_MSG(SpektrumViewKlasse)
    afx_msg void OnPaint();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#endif

```

## B.2.4 SpektrumView.cpp

```

/*
*****
* SpektrumView.cpp
*****
* Beobachter der DSP_Command Klasse
* Zeigt nach dem Prozess die Ein- und Ausgabe
* an
*****
* Geändert am : 06.11.1999
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud
*****
*/
#include "stdafx.h"
#include <math.h>

#include "SpektrumView.h"

// Messagetabelle fuer Windows-Events
BEGIN_MESSAGE_MAP(SpektrumViewKlasse, CWnd)
//{{AFX_MSG_MAP(SpektrumViewKlasse)
ON_WM_PAINT()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//-----
// SpektrumViewKlasse::SpektrumViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _SpektrumViewActive,
// HICON _Icon)
//
// * Initialisieren der SpektrumViewKlasse
// * Anmelden des Views
//
//-----
SpektrumViewKlasse::SpektrumViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _SpektrumViewActive, HICON _Icon)
{
    View_DSP_Command = _DSP_Command;
    DataSource = _DataSource;
    SpektrumViewActive = _SpektrumViewActive;

    *SpektrumViewActive = true;

    View_DSP_Command->MeldeAn (this);
    RECT windowSize = {100,100,500,400};
    Create(NULL, _T(_Titelzeile), WS_OVERLAPPEDWINDOW, windowSize);
    SetIcon ( _Icon ,false );

    srand( (unsigned)time( NULL ) );
};

//-----
// ~SpektrumViewKlasse ()
//
// * Abmelden des Views
//
//-----
SpektrumViewKlasse::~SpektrumViewKlasse ()
{
    View_DSP_Command->MeldeAb (this);
    *SpektrumViewActive = false;
};

//-----
// void OnPaint ( )
//
// * Zeichnen des neuen Fensters (Ausgeloeset durch Window-Event)
//
//-----
void SpektrumViewKlasse::OnPaint ( )
{
    DrawSpektrum ();
    CFrameWnd::OnPaint ();
};

//-----
// void Aktualisiere (SubjektKlasse* _Subjekt)
//
// * Updaten des Views
//
//-----
void SpektrumViewKlasse::Aktualisiere (SubjektKlasse* _Subjekt)
{
    int z1;

#ifdef NO_DSP

```

```

Samplingfrequenz = 44400;
switch (DataSource)
{
case 1 : {
for (z1 = 0; z1 < FFT_SIZE; z1++)
{
SpektrumReal [z1] = (1+rand () % 50);
SpektrumImag [z1] = (1+rand () % 50);
SpektrumAbs [z1] = sqrt (SpektrumReal [z1]*SpektrumReal [z1]+SpektrumImag [z1]*SpektrumImag [z1]);
};
};break;
case 2 : {
for (z1 = 0; z1 < FFT_SIZE; z1++)
{
SpektrumReal [z1] = (1+rand () % 50);
SpektrumImag [z1] = (1+rand () % 50);
SpektrumAbs [z1] = sqrt (SpektrumReal [z1]*SpektrumReal [z1]+SpektrumImag [z1]*SpektrumImag [z1]);
};
};break;
case 3 : {
for (z1 = 0; z1 < FFT_SIZE; z1++)
{
SpektrumReal [z1] = (1+rand () % 50) / 100;
SpektrumImag [z1] = (1+rand () % 50) / 100;
SpektrumAbs [z1] = (1+rand () % 80) / (float)100;
};
};break;
};

#else
try
{
Samplingfrequenz = View_DSP_Command->Get_Samplingfrequenz ();
switch (DataSource)
{
case 1 : {
View_DSP_Command->Get_Eingangsspektrum (SpektrumReal,SpektrumImag);
for (z1 = 0; z1 < FFT_SIZE; z1++)
{
SpektrumAbs [z1] = sqrt (SpektrumReal [z1]*SpektrumReal [z1]+SpektrumImag [z1]*SpektrumImag [z1]);
};
};break;
case 2 : {
View_DSP_Command->Get_Ausgangsspektrum (SpektrumReal,SpektrumImag);
for (z1 = 0; z1 < FFT_SIZE; z1++)
{
SpektrumAbs [z1] = sqrt (SpektrumReal [z1]*SpektrumReal [z1]+SpektrumImag [z1]*SpektrumImag [z1]);
};
};break;
case 3 : View_DSP_Command->Get_SpektralFaktoren (SpektrumAbs);break;
};
}
catch (exception &e)
{
//AfxMessageBox (e.what(),MB_OK,0);
};
#endif

// Gewuenschte Kurven-View darstellen
Invalidate(false);
ShowWindow(SW_SHOW);
};

//-----
// void DrawSpektrum ()
//
// * Zeichnen der In und Outsamples
//
//-----
void SpektrumViewKlasse::DrawSpektrum ()
{
CDC* SpektrumViewDC = GetDC ();

RECT ClientRect;
GetClientRect (&ClientRect);
int WidthX = ClientRect.right - ClientRect.left;
int WidthY = ClientRect.bottom - ClientRect.top;

Zeichnungsfeld = new CDC ();
Zeichnungsfeld_Bitmap = new CBitmap ();

// Virtuellen Fensterkontext erstellen
Zeichnungsfeld->CreateCompatibleDC (SpektrumViewDC);
Zeichnungsfeld_Bitmap->CreateCompatibleBitmap (SpektrumViewDC,WidthX,WidthY);
Zeichnungsfeld->SelectObject (Zeichnungsfeld_Bitmap);

```

## ANHANG B. LISTING : DSP STEUERUNG Mehrkanalige Störgeräuschunterdrückung

```

// Background füllen
RECT Background = {0,0,WidthX, WidthY};
CBrush* BackgroundColor = new CBrush (0xFFFFFFFF);
Zeichnungsfeld->FillRect (&Background,BackgroundColor);
delete (BackgroundColor);

// Werte Aufzeichnen
PlotSpektrum (Zeichnungsfeld,10,15,WidthX-10,WidthY-10,SpektrumAbs,FFT_SIZE,0x00FF0000 );
//PlotSpektrum (SpektrumViewDC,10,10,WidthX-10,WidthY-10,SpektrumAbs,FFT_SIZE,0x00FF0000 );

// Bild auf Fensterkontext kopieren
SpektrumViewDC->BitBlt (0,0,WidthX,WidthY,Zeichnungsfeld,0,0,SRCCOPY);

// Bitmap wieder freigeben
Zeichnungsfeld_Bitmap->DeleteObject();

delete (Zeichnungsfeld_Bitmap);
delete (Zeichnungsfeld);

// Fensterkontext freigeben
ReleaseDC (SpektrumViewDC);
};

//-----
// void PlotSpektrum (CDC* _SpektrumViewDC, int _x1, int _y1, int _x2, int _y2, Spektrumtype* _Spektrum, int _AnzWerte, COLORREF _Color )
//
// * Zeichnen eines Datenbereichs
//
//-----
void SpektrumViewKlasse::PlotSpektrum (CDC* _SpektrumViewDC, int _x1, int _y1, int _x2, int _y2, Spektrumtype* _Spektrum, int _AnzWerte, COLORREF _Color )
{
int z1;
int WidthX = (_x2-_x1);
int WidthY = (_y2-_y1);

int UrsprungX = _x1+50;
int UrsprungY = _y2-30;

char text [10];

Spektrumtype MaxValue;
switch (DataSource)
{
case 1 : MaxValue = 100;break;
case 2 : MaxValue = 100;break;
case 3 : MaxValue = 1;break;
};
/*Spektrumtype MaxValue = _Spektrum [0];
for ( z1 = 1 ; z1 < _AnzWerte ; z1 ++ )
{
if (_Spektrum [z1]>MaxValue) MaxValue = _Spektrum [z1];
};*/

float FactorX = (float)(_x2-UrsprungX-WidthX*0.05)/(float)((_AnzWerte>>1)-1);
float FactorY = (float)(UrsprungY-_y1)/(float)MaxValue;

CPen* AchsenPen = new CPen ( PS_SOLID, 0, 0x110000 );
CPen* SamplePen = new CPen ( PS_SOLID, 0, 0x880000 );

// Kurve aufzeichnen
_SpektrumViewDC->SelectObject ( SamplePen );
_SpektrumViewDC->MoveTo ( UrsprungX, UrsprungY - _Spektrum [0]*FactorY );
for ( z1 = 1 ; z1 < (_AnzWerte)>>1;z1++)
{
_SpektrumViewDC->LineTo ( UrsprungX + floor (z1*FactorX), UrsprungY - _Spektrum [z1]*FactorY );
};

// Beschriftung der Anzeige
CFont Schrift_Horizontal;
Schrift_Horizontal.CreateFont (15,6,0,0,400,FALSE,FALSE,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,"Arial");
CFont Schrift_Vertikal;
Schrift_Vertikal.CreateFont (15,6,900,0,400,FALSE,FALSE,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,"Arial");

_SpektrumViewDC->SelectObject (&Schrift_Horizontal);
_SpektrumViewDC->SetTextAlign (TA_CENTER | TA_TOP);
itoa (Samplingfrequenz>>3,text,10);
_SpektrumViewDC->TextOut (UrsprungX+FactorX*64,UrsprungY,text);
itoa (Samplingfrequenz>>2,text,10);
_SpektrumViewDC->TextOut (UrsprungX+FactorX*128,UrsprungY,text);
itoa (3*Samplingfrequenz>>3,text,10);
_SpektrumViewDC->TextOut (UrsprungX+FactorX*192,UrsprungY,text);
itoa (Samplingfrequenz>>1,text,10);
_SpektrumViewDC->TextOut (UrsprungX+FactorX*256,UrsprungY,text);
_SpektrumViewDC->TextOut (UrsprungX+FactorX*128,UrsprungY+15,"Frequenz [Hz]");

```

```

// Y-Achse beschriften
_SpektrumViewDC->SetTextAlign (TA_RIGHT | TA_BASELINE);
GetLabelText (text,0.25, MaxValue);
_SpektrumViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.25*FactorY,text);
GetLabelText (text,0.50, MaxValue);
_SpektrumViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.50*FactorY,text);
GetLabelText (text,0.75, MaxValue);
_SpektrumViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.75*FactorY,text);
GetLabelText (text,1.00, MaxValue);
_SpektrumViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*1.00*FactorY,text);

_SpektrumViewDC->SetTextAlign (TA_CENTER | TA_TOP);
_SpektrumViewDC->SelectObject (&Schrift_Vertikal);
switch (DataSource)
{
    case 1 : _SpektrumViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Amplitudenwert");break;
    case 2 : _SpektrumViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Amplitudenwert");break;
    case 3 : _SpektrumViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Spektralfaktor");break;
};

// Achsen zeichnen
_SpektrumViewDC->SelectObject ( AchsenPen );
_SpektrumViewDC->MoveTo (_x2,UrsprungY);
_SpektrumViewDC->LineTo (UrsprungX,UrsprungY);
_SpektrumViewDC->LineTo (UrsprungX,_y1);

_SpektrumViewDC->MoveTo (_x2-WidthX/70,UrsprungY-WidthY/70);
_SpektrumViewDC->LineTo (_x2,UrsprungY);
_SpektrumViewDC->LineTo (_x2-WidthX/70,UrsprungY+WidthY/70);

_SpektrumViewDC->MoveTo (UrsprungX-WidthY/70,_y1+WidthX/70);
_SpektrumViewDC->LineTo (UrsprungX,_y1);
_SpektrumViewDC->LineTo (UrsprungX+WidthY/70,_y1+WidthX/70);

// Pens und Brushes freigeben
delete (SamplePen);
delete (AchsenPen);
};

//-----
// void SpektrumViewKlasse::GetLabelText (char* _Text, double _Value, Spektrumtype _MaxValue)
//
// * Text der Y-Labels bestimmen
//
//-----
void SpektrumViewKlasse::GetLabelText (char* _Text, double _Value, Spektrumtype _MaxValue)
{
    switch (DataSource)
    {
        case 1 : sprintf (_Text,"%u",(int)(_MaxValue*_Value));break;
        case 2 : sprintf (_Text,"%u",(int)(_MaxValue*_Value));break;
        case 3 : sprintf (_Text,"%0.2g",(float)(_MaxValue*_Value));break;
    };
};

```

## B.2.5 HuellkurvenView.h

```

/*
*****
* HuellkurvenView.h *
*****
* Beobachter der NoiseEraseKlasse *
* Zeigt Werte der Huellkurve an *
* *
*****
* Geändert am : 04.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/
#ifndef __HuellkurvenView
#define __HuellkurvenView

#include "Defs.h"
#include "Observer.h"
#include "dspcmd.h"

typedef float Huellkurventype;

// Definition der HuellkurvenViewKlasse
class HuellkurvenViewKlasse : public CFrameWnd, BeobachterKlasse
{
public:
    HuellkurvenViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _HuellkurvenViewActive, HICON _Icon);

    virtual ~HuellkurvenViewKlasse ();

    virtual void Aktualisiere (SubjektKlasse* _Subjekt);

private:
    void DrawHuellkurven ();
    void PlotHuellkurven (CDC* _HuellkurvenViewDC, int _x1, int _y1, int _x2, int _y2, Huellkurventype* _HuellkurveMax,
        Huellkurventype* _HuellkurveIst, Huellkurventype* _HuellkurveMin, int _AnzWerte, COLORREF _Color );
    void GetLabelText (char* _Text, double _Value, Huellkurventype _MaxValue);

    // Ressourcenvariablen
    HICON WinIcon;

    // View variablen
    DSP_Cmd* View_DSP_Command;
    bool* HuellkurvenViewActive;
    int DataSource; // Kennziffer der Datenquelle :
    // 1 : Max-Min-Ist Huellkurve

    CDC* Zeichnungsfeld;
    CBitmap* Zeichnungsfeld_Bitmap;

    // Daten variablen
    Huellkurventype HuellkurveMax [ANZAHL_HUELLKURVENWERTE];
    Huellkurventype HuellkurveIst [ANZAHL_HUELLKURVENWERTE];
    Huellkurventype HuellkurveMin [ANZAHL_HUELLKURVENWERTE];

    // Generated message map functions
    //{AFX_MSG(HuellkurvenViewKlasse)
    afx_msg void OnPaint();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#endif

```

## B.2.6 HuellkurvenView.cpp

```

/*
*****
* HuellkurvenView.cpp *
*****
* Beobachter der DSP_Command Klasse *
* Zeigt nach dem Prozess die Ein- und Ausgabe *
* an *
*****
* Geändert am : 02.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/
#include "stdafx.h"
#include <math.h>

#include "HuellkurvenView.h"

// Messagetabelle fuer Windows-Events
BEGIN_MESSAGE_MAP(HuellkurvenViewKlasse, CWnd)
//{{AFX_MSG_MAP(HuellkurvenViewKlasse)
ON_WM_PAINT()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//-----
// HuellkurvenViewKlasse::HuellkurvenViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _HuellkurvenViewActive,
// HICON _Icon)
//
// * Initialisieren der HuellkurvenViewKlasse
// * Anmelden des Views
//
//-----
HuellkurvenViewKlasse::HuellkurvenViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _HuellkurvenViewActive, HICON _Icon)
{
    View_DSP_Command = _DSP_Command;
    DataSource = _DataSource;
    HuellkurvenViewActive = _HuellkurvenViewActive;

    *HuellkurvenViewActive = true;

    View_DSP_Command->MeldeAn (this);
    RECT windowSize = {100,100,500,400};
    Create(NULL, _T(_Titelzeile), WS_OVERLAPPEDWINDOW, windowSize);
    SetIcon ( _Icon ,false );

    srand( (unsigned)time( NULL ) );
};

//-----
// ~HuellkurvenViewKlasse ()
//
// * Abmelden des Views
//
//-----
HuellkurvenViewKlasse::~HuellkurvenViewKlasse ()
{
    View_DSP_Command->MeldeAb (this);
    *HuellkurvenViewActive = false;
};

//-----
// void OnPaint ( )
//
// * Zeichnen des neuen Fensters (Ausgeloeset durch Window-Event)
//
//-----
void HuellkurvenViewKlasse::OnPaint ( )
{
    DrawHuellkurven ();
    CFrameWnd::OnPaint ();
};

//-----
// void Aktualisiere (SubjektKlasse* _Subjekt)
//
// * Updaten des Views
//
//-----
void HuellkurvenViewKlasse::Aktualisiere (SubjektKlasse* _Subjekt)
{
#ifdef NO_DSP
    int z1;

```

## ANHANG B. LISTING : DSP STEUERUNG *Mehrkanalige Störgeräuschunterdrückung*

```

for (z1 = 0; z1 < ANZAHL_HUELLKURVENWERTE; z1++)
{
    HuellkurveMax [z1] = (1+rand () % 50);
    HuellkurveIst [z1] = (1+rand () % 50);
    HuellkurveMin [z1] = (1+rand () % 50);
};
#else
try
{
    switch (DataSource)
    {
    case 1 : {
        View_DSP_Command->Get_HuellkurveMax (HuellkurveMax);
        View_DSP_Command->Get_HuellkurveIst (HuellkurveIst);
        View_DSP_Command->Get_HuellkurveMin (HuellkurveMin);

    };break;
    };
}
catch (exception &e)
{
    //AfxMessageBox (e.what(),MB_OK,0);
};
#endif

// Gewuenschte Kurven-View darstellen
Invalidate(false);
ShowWindow(SW_SHOW);
};

//-----
// void DrawHuellkurven ()
//
// * Zeichnen der In und Outsamples
//
//-----
void HuellkurvenViewKlasse::DrawHuellkurven ()
{
    CDC* HuellkurvenViewDC = GetDC ();

    RECT ClientRect;
    GetClientRect (&ClientRect);
    int WidthX = ClientRect.right - ClientRect.left;
    int WidthY = ClientRect.bottom - ClientRect.top;

    Zeichnungsfeld = new CDC ();
    Zeichnungsfeld_Bitmap = new CBitmap ();

    // Virtuellen Fensterkontext erstellen
    Zeichnungsfeld->CreateCompatibleDC (HuellkurvenViewDC);
    Zeichnungsfeld_Bitmap->CreateCompatibleBitmap (HuellkurvenViewDC,WidthX,WidthY);
    Zeichnungsfeld->SelectObject (Zeichnungsfeld_Bitmap);

    // Background füllen
    RECT Background = {0,0,WidthX, WidthY};
    CBrush* BackgroundColor = new CBrush (0xFFFFFFFF);
    Zeichnungsfeld->FillRect (&Background,BackgroundColor);
    delete (BackgroundColor);

    // Werte Aufzeichnen
    PlotHuellkurven (Zeichnungsfeld,10,15,WidthX-10,WidthY-10,HuellkurveMax,HuellkurveIst,HuellkurveMin,ANZAHL_HUELLKURVENWERTE,0x00FF0000 );

    // Bild auf Fensterkontext kopieren
    HuellkurvenViewDC->BitBlt (0,0,WidthX,WidthY,Zeichnungsfeld,0,0,SRCCOPY);

    // Bitmap wieder freigeben
    Zeichnungsfeld_Bitmap->DeleteObject();

    delete (Zeichnungsfeld_Bitmap);
    delete (Zeichnungsfeld);

    // Fensterkontext freigeben
    ReleaseDC (HuellkurvenViewDC);
};

//-----
// void HuellkurvenViewKlasse::PlotHuellkurven (CDC* _HuellkurvenViewDC, int _x1, int _y1, int _x2, int _y2, Huellkurventype* _HuellkurveMax,
// Huellkurventype* _HuellkurveIst, Huellkurventype* _HuellkurveMin, int _AnzWerte, COLORREF _Color )
// * Zeichnen eines Datenbereichs
//
//-----
void HuellkurvenViewKlasse::PlotHuellkurven (CDC* _HuellkurvenViewDC, int _x1, int _y1, int _x2, int _y2, Huellkurventype* _HuellkurveMax,
Huellkurventype* _HuellkurveIst, Huellkurventype* _HuellkurveMin, int _AnzWerte, COLORREF _Color )
{
    int z1;
    int WidthX = (_x2-_x1);

```

```

int WidthY = (_y2-_y1);
int UrsprungX = _x1+50;
int UrsprungY = _y2-30;

char text [10];

Huellkurventype MaxValue;
switch (DataSource)
{
case 1 : MaxValue = 100;break;
};
/*Huellkurventype MaxValue = _Huellkurven [0];
for ( z1 = 1 ; z1 < _AnzWerte ; z1 ++ )
{
if (_Huellkurven [z1]>MaxValue) MaxValue = _Huellkurven [z1];
};*/

float FactorX = (float)(_x2-UrsprungX-WidthX*0.05)/(float)((_AnzWerte)-1);
float FactorY = (float)(UrsprungY-_y1)/(float)MaxValue;

CPen* AchsenPen = new CPen ( PS_SOLID, 0, 0x110000 );
CPen* MaxPen = new CPen ( PS_SOLID, 0, 0x880000 );
CPen* IstPen = new CPen ( PS_SOLID, 0, 0x008800 );
CPen* MinPen = new CPen ( PS_SOLID, 0, 0x000088 );

// MaxKurve aufzeichnen
_HuellkurvenViewDC->SelectObject ( MaxPen );
_HuellkurvenViewDC->MoveTo ( UrsprungX, UrsprungY - _HuellkurveMax [0]*FactorY );
for ( z1 = 1 ; z1 < (_AnzWerte);z1++)
{
_HuellkurvenViewDC->LineTo ( UrsprungX + floor (z1*FactorX), UrsprungY - _HuellkurveMax [z1]*FactorY );
};

// IstKurve aufzeichnen
_HuellkurvenViewDC->SelectObject ( IstPen );
_HuellkurvenViewDC->MoveTo ( UrsprungX, UrsprungY - _HuellkurveIst [0]*FactorY );
for ( z1 = 1 ; z1 < (_AnzWerte);z1++)
{
_HuellkurvenViewDC->LineTo ( UrsprungX + floor (z1*FactorX), UrsprungY - _HuellkurveIst [z1]*FactorY );
};

// MinKurve aufzeichnen
_HuellkurvenViewDC->SelectObject ( MinPen );
_HuellkurvenViewDC->MoveTo ( UrsprungX, UrsprungY - _HuellkurveMin [0]*FactorY );
for ( z1 = 1 ; z1 < (_AnzWerte);z1++)
{
_HuellkurvenViewDC->LineTo ( UrsprungX + floor (z1*FactorX), UrsprungY - _HuellkurveMin [z1]*FactorY );
};

// Beschriftung der Anzeige
CFont Schrift_Horizontal;
Schrift_Horizontal.CreateFont (15,6,0,0,400,FALSE,FALSE,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,"Arial");
CFont Schrift_Vertikal;
Schrift_Vertikal.CreateFont (15,6,900,0,400,FALSE,FALSE,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,"Arial");

// X-Achse beschriften
_HuellkurvenViewDC->SelectObject ( AchsenPen );
_HuellkurvenViewDC->SelectObject (&Schrift_Horizontal);
_HuellkurvenViewDC->SetTextAlign (TA_CENTER | TA_TOP);
_HuellkurvenViewDC->TextOut (UrsprungX,UrsprungY,"200");
_HuellkurvenViewDC->TextOut (UrsprungX+FactorX*50,UrsprungY,"150");
_HuellkurvenViewDC->TextOut (UrsprungX+FactorX*100,UrsprungY,"100");
_HuellkurvenViewDC->TextOut (UrsprungX+FactorX*150,UrsprungY,"50");
_HuellkurvenViewDC->TextOut (UrsprungX+FactorX*100,UrsprungY+15,"Nummerierung der FFT");

// Y-Achse beschriften
_HuellkurvenViewDC->SetTextAlign (TA_RIGHT | TA_BASELINE);
GetLabelText (text,0.25, MaxValue);
_HuellkurvenViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.25*FactorY,text);
GetLabelText (text,0.50, MaxValue);
_HuellkurvenViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.50*FactorY,text);
GetLabelText (text,0.75, MaxValue);
_HuellkurvenViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.75*FactorY,text);
GetLabelText (text,1.00, MaxValue);
_HuellkurvenViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*1.00*FactorY,text);

_HuellkurvenViewDC->SetTextAlign (TA_CENTER | TA_TOP);
_HuellkurvenViewDC->SelectObject (&Schrift_Vertikal);
_HuellkurvenViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Amplitudenmittelwert");

// Achsen zeichnen
_HuellkurvenViewDC->MoveTo (_x2,UrsprungY);
_HuellkurvenViewDC->LineTo (UrsprungX,UrsprungY);
_HuellkurvenViewDC->LineTo (UrsprungX,_y1);

```

## ANHANG B. LISTING : DSP STEUERUNG *Mehrkanalige Störgeräuschunterdrückung*

```
_HuellkurvenViewDC->MoveTo (_x2-WidthX/70,UrsprungY-WidthY/70);
_HuellkurvenViewDC->LineTo (_x2,UrsprungY);
_HuellkurvenViewDC->LineTo (_x2-WidthX/70,UrsprungY+WidthY/70);

_HuellkurvenViewDC->MoveTo (UrsprungX-WidthY/70,_y1+WidthX/70);
_HuellkurvenViewDC->LineTo (UrsprungX,_y1);
_HuellkurvenViewDC->LineTo (UrsprungX+WidthY/70,_y1+WidthX/70);

// Pens und Brushes freigeben
delete (MinPen);
delete (IstPen);
delete (MaxPen);
delete (AchsenPen);
};

//-----
// void HuellkurvenViewKlasse::GetLabelText (char* _Text, double _Value, Huellkurventype _MaxValue)
//
// * Text der Y-Labels bestimmen
//
//-----
void HuellkurvenViewKlasse::GetLabelText (char* _Text, double _Value, Huellkurventype _MaxValue)
{
    sprintf (_Text,"%u",(int)(_MaxValue*_Value));
};
```

## B.2.7 BaenderView.h

```

/*
*****
* BaenderView.h *
*****
* Beobachter der NoiseEraseKlasse *
* Zeigt nach dem Prozess die Ein- und Ausgabe *
* an *
*****
* Geändert am : 04.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/
#ifndef __BaenderView
#define __BaenderView

#include "Defs.h"
#include "Observer.h"
#include "dspcmd.h"

typedef float Baendertype;

// Definition der BaenderViewKlasse
class BaenderViewKlasse : public CFrameWnd, BeobachterKlasse
{
public:
    BaenderViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _BaenderViewActive, HICON _Icon);

    virtual ~BaenderViewKlasse ();

    virtual void Aktualisiere (SubjektKlasse* _Subjekt);

private:
    void DrawBaender ();
    void PlotBaender (CDC* _BaenderViewDC, int _x1, int _y1, int _x2, int _y2, Baendertype* _BandWerte, int _AnzWerte, COLORREF _Color );
    void GetLabelText (char* _Text, double _Value, Baendertype _MaxValue);

    // Ressourcenvariablen
    HICON WinIcon;

    // View variablen
    DSP_Cmd* View_DSP_Command;
    bool* BaenderViewActive;
    int DataSource; // Kennziffer der Datenquelle :
    // 1 : Amplitudenmittelwerte
    // 2 : Bandfaktoren
    // 3 : Banddynamik
    CDC* Zeichnungsfeld;
    CBitmap* Zeichnungsfeld_Bitmap;

    // Daten variablen
    Baendertype BandWerte [ANZAHL_BAENDER];

    // Generated message map functions
    //{{AFX_MSG(BaenderViewKlasse)
    afx_msg void OnPaint();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#endif

```

## B.2.8 BaenderView.cpp

```

/*
*****
* BaenderView.cpp
*****
* Beobachter der DSP_Command Klasse
* Anzeigen von bandabhaengigen Daten
*
*****
* Geändert am : 08.11.1999
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud
*****
*/
#include "stdafx.h"
#include <math.h>

#include "BaenderView.h"

// Messagetabelle fuer Windows-Events
BEGIN_MESSAGE_MAP(BaenderViewKlasse, CWnd)
//{{AFX_MSG_MAP(BaenderViewKlasse)
ON_WM_PAINT()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//-----
// BaenderViewKlasse::BaenderViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _BaenderViewActive,
// HICON _Icon)
//
// * Initialisieren der BaenderViewKlasse
// * Anmelden des Views
//
//-----
BaenderViewKlasse::BaenderViewKlasse ( DSP_Cmd *_DSP_Command, int _DataSource, char* _Titelzeile, bool* _BaenderViewActive, HICON _Icon)
{
    View_DSP_Command = _DSP_Command;
    DataSource = _DataSource;
    BaenderViewActive = _BaenderViewActive;

    *BaenderViewActive = true;

    View_DSP_Command->MeldeAn (this);
    RECT windowSize = {100,100,500,400};
    Create(NULL, _T(_Titelzeile), WS_OVERLAPPEDWINDOW, windowSize);
    SetIcon ( _Icon ,false );

    srand( (unsigned)time( NULL ) );
};

//-----
// ~BaenderViewKlasse ()
//
// * Abmelden des Views
//
//-----
BaenderViewKlasse::~BaenderViewKlasse ()
{
    View_DSP_Command->MeldeAb (this);
    *BaenderViewActive = false;
};

//-----
// void OnPaint ( )
//
// * Zeichnen des neuen Fensters (Ausgeloest durch Window-Event)
//
//-----
void BaenderViewKlasse::OnPaint ( )
{
    DrawBaender ();
    CFrameWnd::OnPaint ();
};

//-----
// void Aktualisiere (SubjektKlasse* _Subjekt)
//
// * Updaten des Views
//
//-----
void BaenderViewKlasse::Aktualisiere (SubjektKlasse* _Subjekt)
{
#ifdef NO_DSP
    int zi;

```

```

    switch (DataSource)
    {
    case 1 : {
        for (z1 = 0; z1 < ANZAHL_BAENDER; z1++)
        {
            BandWerte [z1] = (1+rand () % 50);
        };
        };break;
    case 2 : {
        for (z1 = 0; z1 < ANZAHL_BAENDER; z1++)
        {
            BandWerte [z1] = (float)(1+rand () % 100)/(float)100;
        };
        };break;
    case 3 : {
        for (z1 = 0; z1 < ANZAHL_BAENDER; z1++)
        {
            BandWerte [z1] = (1+rand () % 100);
        };
        };break;
    };
    #else
    try
    {
        switch (DataSource)
        {
        case 1 : {
            View_DSP_Command->Get_BandMittelwerte (BandWerte);
        };break;
        case 2 : {
            View_DSP_Command->Get_BandFaktoren (BandWerte);
        };break;
        case 3 : {
            View_DSP_Command->Get_BandDynamik (BandWerte);
        };break;
        };
    }
    catch (exception &e)
    {
        //AfxMessageBox (e.what(), MB_OK, 0);
    };
    #endif

    // Gewuenschte Kurven-View darstellen
    Invalidate(false);
    ShowWindow(SW_SHOW);
};

//-----
// void DrawBaender ()
//
// * Zeichnen der In und Outsamples
//
//-----
void BaenderViewKlasse::DrawBaender ()
{
    CDC* BaenderViewDC = GetDC ();

    RECT ClientRect;
    GetClientRect (&ClientRect);
    int WidthX = ClientRect.right - ClientRect.left;
    int WidthY = ClientRect.bottom - ClientRect.top;

    Zeichnungsfeld = new CDC ();
    Zeichnungsfeld_Bitmap = new CBitmap ();

    // Virtuellen Fensterkontext erstellen
    Zeichnungsfeld->CreateCompatibleDC (BaenderViewDC);
    Zeichnungsfeld_Bitmap->CreateCompatibleBitmap (BaenderViewDC, WidthX, WidthY);
    Zeichnungsfeld->SelectObject (Zeichnungsfeld_Bitmap);

    // Background füllen
    RECT Background = {0,0,WidthX, WidthY};
    CBrush* BackgroundColor = new CBrush (0xFFFFFFFF);
    Zeichnungsfeld->FillRect (&Background, BackgroundColor);
    delete (BackgroundColor);

    // Werte Aufzeichnen
    PlotBaender (Zeichnungsfeld, 10, 15, WidthX-10, WidthY-10, BandWerte, ANZAHL_BAENDER, 0x00FF0000 );

    // Bild auf Fensterkontext kopieren
    BaenderViewDC->BitBlt (0,0,WidthX,WidthY,Zeichnungsfeld,0,0,SRCCOPY);

    // Bitmap wieder freigeben
    Zeichnungsfeld_Bitmap->DeleteObject();
}

```

## ANHANG B. LISTING : DSP STEUERUNG *Mehrkanalige Störgeräuschunterdrückung*

```

delete (Zeichnungsfeld_Bitmap);
delete (Zeichnungsfeld);

// Fensterkontext freigeben
ReleaseDC (BaenderViewDC);
};

//-----
// void PlotBaender (CDC* _BaenderViewDC, int _x1, int _y1, int _x2, int _y2, Baendertype* _BandWerte, int _AnzWerte, COLORREF _Color )
//
// * Zeichnen eines Datenbereichs
//
//-----
void BaenderViewKlasse::PlotBaender (CDC* _BaenderViewDC, int _x1, int _y1, int _x2, int _y2, Baendertype* _BandWerte, int _AnzWerte, COLORREF _Color )
{
    int z1;
    int WidthX = (_x2-_x1);
    int WidthY = (_y2-_y1);

    int UrsprungX = _x1+50;
    int UrsprungY = _y2-30;

    char text [10];

    Baendertype MaxValue;
    switch (DataSource)
    {
        case 1 : MaxValue = 100;break;
        case 2 : MaxValue = 1;break;
        case 3 : MaxValue = 100;break;
    };
    /*MaxValue = _BandWerte [0];
    for ( z1 = 1 ; z1 < _AnzWerte ; z1 ++ )
    {
        if (_BandWerte [z1]>MaxValue) MaxValue = _BandWerte [z1];
    };*/

    float Balkenbreite = (float)(_x2-UrsprungX)/(float)_AnzWerte;
    float FactorY = (float)(UrsprungY-_y1)/(float)MaxValue;

    CBrush* BalkenBrush = new CBrush (0x880000);
    CPen* SkalaPen = new CPen (PS_DOT, 1, 0xAAAAAA);

    // Beschriftung der Anzeige
    CFont Schrift_Horizontal;
    Schrift_Horizontal.CreateFont (15,6,0,0,400,FALSE,FALSE,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
    CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,"Arial");
    CFont Schrift_Vertikal;
    Schrift_Vertikal.CreateFont (15,6,900,0,400,FALSE,FALSE,0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
    CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,"Arial");

    _BaenderViewDC->SelectObject (&Schrift_Horizontal);
    _BaenderViewDC->SetTextAlign (TA_CENTER | TA_TOP);
    _BaenderViewDC->TextOut (UrsprungX+Balkenbreite/2+4*Balkenbreite,UrsprungY,"5");
    _BaenderViewDC->TextOut (UrsprungX+Balkenbreite/2+9*Balkenbreite,UrsprungY,"10");
    _BaenderViewDC->TextOut (UrsprungX+Balkenbreite/2+14*Balkenbreite,UrsprungY,"15");
    _BaenderViewDC->TextOut (UrsprungX+Balkenbreite/2+19*Balkenbreite,UrsprungY,"20");
    _BaenderViewDC->TextOut (UrsprungX+12*Balkenbreite,UrsprungY+15,"Bänder");

    // Y-Achse beschriften
    _BaenderViewDC->SelectObject (SkalaPen);
    _BaenderViewDC->SetTextAlign (TA_RIGHT | TA_BASELINE);
    GetLabelText (text,0.25, MaxValue);
    _BaenderViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.25*FactorY,text);
    _BaenderViewDC->MoveTo (UrsprungX-2,UrsprungY-MaxValue*0.25*FactorY);
    _BaenderViewDC->LineTo (_x2,UrsprungY-MaxValue*0.25*FactorY);

    GetLabelText (text,0.50, MaxValue);
    _BaenderViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.50*FactorY,text);
    _BaenderViewDC->MoveTo (UrsprungX-2,UrsprungY-MaxValue*0.50*FactorY);
    _BaenderViewDC->LineTo (_x2,UrsprungY-MaxValue*0.50*FactorY);

    GetLabelText (text,0.75, MaxValue);
    _BaenderViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*0.75*FactorY,text);
    _BaenderViewDC->MoveTo (UrsprungX-2,UrsprungY-MaxValue*0.75*FactorY);
    _BaenderViewDC->LineTo (_x2,UrsprungY-MaxValue*0.75*FactorY);

    GetLabelText (text,1.00, MaxValue);
    _BaenderViewDC->TextOut (UrsprungX-5,UrsprungY-MaxValue*1.00*FactorY,text);
    _BaenderViewDC->MoveTo (UrsprungX-2,UrsprungY-MaxValue*1.00*FactorY);
    _BaenderViewDC->LineTo (_x2,UrsprungY-MaxValue*1.00*FactorY);

    _BaenderViewDC->SetTextAlign (TA_CENTER | TA_TOP);
    _BaenderViewDC->SelectObject (&Schrift_Vertikal);
    switch (DataSource)
    {

```

```

    case 1 : _BaenderViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Bandmittelwert");break;
    case 2 : _BaenderViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Bandfaktoren");break;
    case 3 : _BaenderViewDC->TextOut (_x1,UrsprungY-MaxValue*0.5*FactorY,"Baddynamik [dB]");break;
};

// Balken zeichnen
for ( z1 = 0 ; z1 < _AnzWerte;z1++)
{
    RECT Wertebalken = {UrsprungX+z1*Balkenbreite,UrsprungY - _BandWerte [z1]*FactorY,UrsprungX-1+(z1+1)*Balkenbreite,UrsprungY};
    _BaenderViewDC->FillRect (&Wertebalken , BalkenBrush);
};

// Pens und Brushes freigeben
delete (SkalaPen);
delete (BalkenBrush);
};

//-----
// void BaenderViewKlasse::GetLabelText (char* _Text, double _Value, Baendertype _MaxValue)
//
// * Text der Y-Labels bestimmen
//
//-----
void BaenderViewKlasse::GetLabelText (char* _Text, double _Value, Baendertype _MaxValue)
{
    switch (DataSource)
    {
        case 1 : sprintf (_Text,"%u", (int)(_MaxValue*_Value));break;
        case 2 : sprintf (_Text,"% .2g", (float)(_MaxValue*_Value));break;
        case 3 : sprintf (_Text,"%u", (int)(_MaxValue*_Value));break;
    };
};

```

## B.3 Windows Dialog

### B.3.1 DSP\_ControlDlg.h

```
// DSP_ControlDlg.h : header file
//

#include "dspcmd.h"
#include "Spektrumview.h"
#include "Huellkurvenview.h"
#include "Baenderview.h"

#if !defined(AFX_DSP_CONTROLDLG_H__35F8C8E8_8E0A_11D3_B254_00C04F7159B6__INCLUDED_)
#define AFX_DSP_CONTROLDLG_H__35F8C8E8_8E0A_11D3_B254_00C04F7159B6__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

//////////////////////////////////////
// CDSP_ControlDlg dialog

class CDSP_ControlDlg : public CDialog
{
// Construction
public:
    CDSP_ControlDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(CDSP_ControlDlg)
    enum { IDD = IDD_DSP_CONTROL_DIALOG };
    CButton m_Button_Bypass;
    CComboBox m_Combo_BandAuswahl;
    CComboBox m_Combo_ViewAuswahl;
    CSliderCtrl m_Slider_Absinkfaktor;
    CSliderCtrl m_Slider_Anstiegsfaktor;
    CEdit m_Edit_Anstiegsfaktor;
    CEdit m_Edit_Absinkfaktor;
    CEdit m_Edit_MaxDynamik;
    CSliderCtrl m_Slider_MaxDynamik;
    CSliderCtrl m_Slider_MaxDaempfung;
    CEdit m_Edit_MaxDaempfung;
    }}AFX_DATA

// ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CDSP_ControlDlg)
    public:
    virtual BOOL DestroyWindow();
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    virtual BOOL OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult);
    }}AFX_VIRTUAL

// Viewclasses
    SpektrumViewKlasse* EingangsSpektrumView;
    bool EingangsSpektrumViewActive;
    BaenderViewKlasse* BandMittelwerteView;
    bool BandMittelwerteViewActive;
    HuellkurvenViewKlasse* HuellkurvenView;
    bool HuellkurvenViewActive;
    BaenderViewKlasse* BandDynamikView;
    bool BandDynamikViewActive;
    BaenderViewKlasse* BandFaktorenView;
    bool BandFaktorenViewActive;
    SpektrumViewKlasse* AusgangsSpektrumView;
    bool AusgangsSpektrumViewActive;
    SpektrumViewKlasse* SpektralFaktorenView;
    bool SpektralFaktorenViewActive;

    int DSP_MaxDaempfung;
    int DSP_MaxDynamik;
    int DSP_Anstiegsfaktor;
    int DSP_Absinkfaktor;

// Implementation
protected:
    HICON m_hIcon;

// Generated message map functions
   //{{AFX_MSG(CDSP_ControlDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    }}AFX_MSG
};
```

```
afx_msg void OnDSPStart();
afx_msg void OnDSPReset();
afx_msg void OnAbout();
afx_msg void OnWerteanzeigen();
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnSelchangeComboViewauswahl();
afx_msg void OnBypass();
afx_msg void OnHilfe();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DSP_CONTROLDLG_H__35F8C8E8_8E0A_11D3_B254_00C04F7159B6__INCLUDED_)
```

### B.3.2 DSP\_ControlDlg.cpp

```
// DSP_ControlDlg.cpp : implementation file
//

#include "stdafx.h"
#include "DSP_Control.h"
#include "DSP_ControlDlg.h"

#include "dspcmd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern DSP_Cmd* DSP_Command;
////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

    // Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    }}AFX_DATA

    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }}AFX_VIRTUAL

    // Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
    }}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
    }}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    }}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDSP_ControlDlg dialog

CDSP_ControlDlg::CDSP_ControlDlg(CWnd* pParent /*=NULL*/)
: CDialog(CDSP_ControlDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CDSP_ControlDlg)
    }}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDSP_ControlDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CDSP_ControlDlg)
    DDX_Control(pDX, IDC_BYPASS, m_Button_Bypass);
    DDX_Control(pDX, IDC_BANDAUSWAHL, m_Combo_BandAuswahl);
    DDX_Control(pDX, IDC_COMBO_VIEWAUSWAHL, m_Combo_ViewAuswahl);
    DDX_Control(pDX, IDC_SLIDER_ABSINKFAKTOR, m_Slider_Absinkfaktor);
    DDX_Control(pDX, IDC_SLIDER_ANSTIEGSFAKTOR, m_Slider_Anstiegsfaktor);
    DDX_Control(pDX, IDC_EDIT_ANSTIEGSFAKTOR, m_Edit_Anstiegsfaktor);
    DDX_Control(pDX, IDC_EDIT_ABSINKFAKTOR, m_Edit_Absinkfaktor);
    DDX_Control(pDX, IDC_EDIT_MAXDYNAMIK, m_Edit_MaxDynamik);
    }}AFX_DATA_MAP
}

```

## Mehrkanalige Störgeräuschunterdrückung ANHANG B. LISTING : DSP STEUERUNG

```
DDX_Control(pDX, IDC_SLIDER_MAXDYNAMIK, m_Slider_MaxDynamik);
DDX_Control(pDX, IDC_SLIDER_MAXDAEMPfung, m_Slider_MaxDaempfung);
DDX_Control(pDX, IDC_EDIT_MAXDAEMPfung, m_Edit_MaxDaempfung);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDSP_ControlDlg, CDialog)
//{{AFX_MSG_MAP(CDSP_ControlDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_WM_CREATE()
ON_WM_CLOSE()
ON_BN_CLICKED(IDC_DSPstart, OnDSPstart)
ON_BN_CLICKED(IDC_DSPreset, OnDSPreset)
ON_BN_CLICKED(IDC_ABOUT, OnAbout)
ON_BN_CLICKED(IDC_WERTEANZEIGEN, OnWerteanzeigen)
ON_WM_TIMER()
ON_CBN_SELCHANGE(IDC_COMBO_VIEWAUSWAHL, OnSelchangeComboViewauswahl)
ON_BN_CLICKED(IDC_BYPASS, OnBypass)
ON_BN_CLICKED(IDC_HILFE, OnHilfe)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDSP_ControlDlg message handlers

BOOL CDSP_ControlDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE); // Set big icon
    SetIcon(m_hIcon, FALSE); // Set small icon

    // Initialisieren der Oberflaeche
    m_Slider_MaxDaempfung.SetRange (1, 50, false);
    m_Slider_MaxDaempfung.SetPos (30);
    m_Edit_MaxDaempfung.SetWindowText ("30");

    m_Slider_MaxDynamik.SetRange (1, 50, false);
    m_Slider_MaxDynamik.SetPos (30);
    m_Edit_MaxDynamik.SetWindowText ("30");

    m_Slider_Anstiegfaktor.SetRange (1001, 1100, false);
    m_Slider_Anstiegfaktor.SetPos (1010);
    m_Edit_Anstiegfaktor.SetWindowText ("1.010");

    m_Slider_Absinkfaktor.SetRange (900, 999, false);
    m_Slider_Absinkfaktor.SetPos (990);
    m_Edit_Absinkfaktor.SetWindowText ("0.990");

    DSP_MaxDaempfung = 30;
    DSP_MaxDynamik = 30;
    DSP_Anstiegfaktor = 1010;
    DSP_Absinkfaktor = 990;

    m_Combo_ViewAuswahl.SetCurSel (0);
    m_Combo_BandAuswahl.SetCurSel (0);

    return TRUE; // return TRUE unless you set the focus to a control
}

void CDSP_ControlDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFFF) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
}
```

## ANHANG B. LISTING : DSP STEUERUNG *Mehrkanalige Störgeräuschunterdrückung*

```
}
else
{
CDialog::OnSysCommand(nID, lParam);
}
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CDSP_ControlDlg::OnPaint()
{
if (IsIconic())
{
CPaintDC dc(this); // device context for painting

SendMessage(WM_ICONERASEBKGD, (LPARAM) dc.GetSafeHdc(), 0);

// Center icon in client rectangle
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;

// Draw the icon
dc.DrawIcon(x, y, m_hIcon);
}
else
{
CDialog::OnPaint();
}
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CDSP_ControlDlg::OnQueryDragIcon()
{
return (HCURSOR) m_hIcon;
}

int CDSP_ControlDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
if (CDialog::OnCreate(lpCreateStruct) == -1) return -1;

// Variablen preset
EingangSpektrumViewActive = false;
BandMittelwerteViewActive = false;
HuellkurvenViewActive = false;
BandDynamikViewActive = false;
BandFaktorenViewActive = false;
SpektralFaktorenViewActive = false;
AusgangSpektrumViewActive = false;

// Create DSP_Command-Object
DSP_Command = new DSP_Cmd ();

DSP_Command->BoardInit ();

SetTimer (0,100,NULL);
return 0;
}

void CDSP_ControlDlg::OnClose()
{
KillTimer (0);
CDialog::OnClose();
}

void CDSP_ControlDlg::OnDSPStart()
{
try
{
DSP_Command->BoardReset ();
DSP_Command->BoardProgStart ();

m_Slider_MaxDaempfung.SetPos (30);
m_Edit_MaxDaempfung.SetWindowText ("30 dB");
m_Slider_MaxDynamik.SetPos (30);
m_Edit_MaxDynamik.SetWindowText ("30 dB");
m_Slider_Anstiegsfaktor.SetPos (1010);
m_Edit_Anstiegsfaktor.SetWindowText ("1.010");
m_Slider_Absinkfaktor.SetPos (990);
m_Edit_Absinkfaktor.SetWindowText ("0.990");

DSP_MaxDaempfung = 30;

```

```

    DSP_MaxDynamik      = 30;
    DSP_Anstiegsfaktor = 1010;
    DSP_Absinkfaktor   = 990;
}
catch (exception &e)
{
    AfxMessageBox (e.what(),MB_OK,0);
};
}

void CDSP_ControlDlg::OnDSPReset()
{
    try
    {
        DSP_Command->BoardReset ();
    }
    catch (exception &e)
    {
        AfxMessageBox (e.what(),MB_OK,0);
    };
}

BOOL CDSP_ControlDlg::OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult)
{
    char txt [10];

    if (wParam == IDC_SLIDER_MAXDAEMPUNG)
    {
        itoa (m_Slider_MaxDaempfung.GetPos (),txt,10);
        m_Edit_MaxDaempfung.SetWindowText (strcat (txt," dB"));
    };

    if (wParam == IDC_SLIDER_MAXDYNAMIK)
    {
        itoa (m_Slider_MaxDynamik.GetPos (),txt,10);
        m_Edit_MaxDynamik.SetWindowText (strcat (txt," dB"));
    };

    if (wParam == IDC_SLIDER_ANSTIEGSFAKTOR)
    {
        sprintf (txt,"%f",m_Slider_Anstiegsfaktor.GetPos ()/(float)1000);
        m_Edit_Anstiegsfaktor.SetWindowText (txt);
    };

    if (wParam == IDC_SLIDER_ABSINKFAKTOR)
    {
        sprintf (txt,"%f",m_Slider_Absinkfaktor.GetPos ()/(float)1000);
        m_Edit_Absinkfaktor.SetWindowText (txt);
    };

    return CDialog::OnNotify(wParam, lParam, pResult);
}

void CDSP_ControlDlg::OnAbout()
{
    CAboutDlg dlgAbout;
    dlgAbout.DoModal ();
}

void CDSP_ControlDlg::OnWerteanzeigen()
{
    try
    {
        switch (m_Combo_ViewAuswahl.GetCurSel ())
        {
            // Eingangsspektrum
            case 0 : {
                if (EingangsspektrumViewActive == false)
                {
                    EingangsspektrumView = new SpektrumViewKlasse ( DSP_Command, 1, "Eingangsspektrum", &EingangsspektrumViewActive,m_hIcon);
                    DSP_Command->Benachrichtige ();
                }
                else AfxMessageBox ("Das Eingangsspektrum wird bereits angezeigt",MB_OK,0);
            };break;
            // Bandmittelwerte
            case 1 : {
                if (BandMittelwerteViewActive == false)
                {
                    BandMittelwerteView = new BaenderViewKlasse ( DSP_Command, 1, "Bandmittelwerte", &BandMittelwerteViewActive,m_hIcon);
                    DSP_Command->Benachrichtige ();
                }
                else AfxMessageBox ("Die Bandmittelwerte werden bereits angezeigt",MB_OK,0);
            };break;
            // Huellkurve
            case 2 : {
                if (HuellkurvenViewActive == false)
                {
                    DSP_Command->Set_Bandnummer (m_Combo_BandAuswahl.GetCurSel ());
                }
            };
        }
    }
}

```

## ANHANG B. LISTING : DSP STEUERUNG *Mehrkanalige Störgeräuscherdrückung*

```

char txt1 [100] = "Hüllkurve von ";
char txt2 [30];
m_Combo_BandAuswahl.GetLText (m_Combo_BandAuswahl.GetCurSel (),txt2);
HuellkurvenView = new HuellkurvenViewKlasse ( DSP_Command, 1, strcat (txt1,txt2), &HuellkurvenViewActive,m_hIcon);
    DSP_Command->Benachrichtige ();
}
else AfxMessageBox ("Es wird bereits eine Hüllkurve angezeigt",MB_OK,0);
};break;
// Bandedynamik
case 3: {
    if (BandDynamikViewActive == false)
    {
        BandDynamikView = new BaenderViewKlasse ( DSP_Command, 3, "Bandedynamik", &BandDynamikViewActive,m_hIcon);
        DSP_Command->Benachrichtige ();
    }
    else AfxMessageBox ("Die Bandedynamik wird bereits angezeigt",MB_OK,0);
};break;
// Bandfaktoren
case 4: {
    if (BandFaktorenViewActive == false)
    {
        BandFaktorenView = new BaenderViewKlasse ( DSP_Command, 2, "Bandfaktoren", &BandFaktorenViewActive,m_hIcon);
        DSP_Command->Benachrichtige ();
    }
    else AfxMessageBox ("Die Bandfaktoren werden bereits angezeigt",MB_OK,0);
};break;
// Spektralfaktoren
case 5: {
    if (SpektralFaktorenViewActive == false)
    {
        SpektralFaktorenView = new SpektrumViewKlasse ( DSP_Command, 3, "Spektralfaktoren",&SpektralFaktorenViewActive,m_hIcon);
        DSP_Command->Benachrichtige ();
    }
    else AfxMessageBox ("Das Ausgangsspektrum wird bereits angezeigt",MB_OK,0);
};break;
// Ausgangsspektrum
case 6: {
    if (AusgangsSpektrumViewActive == false)
    {
        AusgangsSpektrumView = new SpektrumViewKlasse ( DSP_Command, 2, "Ausgangsspektrum",&AusgangsSpektrumViewActive,m_hIcon);
        DSP_Command->Benachrichtige ();
    }
    else AfxMessageBox ("Das Ausgangsspektrum wird bereits angezeigt",MB_OK,0);
};break;
case 7 : {
    if (EingangsspektrumViewActive == false)
    {
        EingangsspektrumView = new SpektrumViewKlasse ( DSP_Command, 1, "Eingangsspektrum", &EingangsspektrumViewActive,m_hIcon);
    }
    else AfxMessageBox ("Das Eingangsspektrum wird bereits angezeigt",MB_OK,0);
    if (BandMittelwerteViewActive == false)
    {
        BandMittelwerteView = new BaenderViewKlasse ( DSP_Command, 1, "Bandmittelwerte", &BandMittelwerteViewActive,m_hIcon);
    }
    else AfxMessageBox ("Die Bandmittelwerte werden bereits angezeigt",MB_OK,0);
    if (HuellkurvenViewActive == false)
    {
        DSP_Command->Set_Bandnummer (m_Combo_BandAuswahl.GetCurSel ());
        char txt1 [100] = "Hüllkurve von ";
        char txt2 [30];
        m_Combo_BandAuswahl.GetLText (m_Combo_BandAuswahl.GetCurSel (),txt2);
        HuellkurvenView = new HuellkurvenViewKlasse ( DSP_Command, 1, strcat (txt1,txt2), &HuellkurvenViewActive,m_hIcon);
    }
    else AfxMessageBox ("Es wird bereits eine Hüllkurve angezeigt",MB_OK,0);
    if (BandDynamikViewActive == false)
    {
        BandDynamikView = new BaenderViewKlasse ( DSP_Command, 3, "Bandedynamik", &BandDynamikViewActive,m_hIcon);
    }
    else AfxMessageBox ("Die Bandedynamik wird bereits angezeigt",MB_OK,0);
    if (BandFaktorenViewActive == false)
    {
        BandFaktorenView = new BaenderViewKlasse ( DSP_Command, 2, "Bandfaktoren", &BandFaktorenViewActive,m_hIcon);
    }
    else AfxMessageBox ("Die Bandfaktoren werden bereits angezeigt",MB_OK,0);
    if (SpektralFaktorenViewActive == false)
    {
        SpektralFaktorenView = new SpektrumViewKlasse ( DSP_Command, 3, "Spektralfaktoren",&SpektralFaktorenViewActive,m_hIcon);
    }
    else AfxMessageBox ("Das Ausgangsspektrum wird bereits angezeigt",MB_OK,0);
    if (AusgangsSpektrumViewActive == false)
    {
        AusgangsSpektrumView = new SpektrumViewKlasse ( DSP_Command, 2, "Ausgangsspektrum",&AusgangsSpektrumViewActive,m_hIcon);
    }
    else AfxMessageBox ("Das Ausgangsspektrum wird bereits angezeigt",MB_OK,0);
        DSP_Command->Benachrichtige ();
};break;
};
};

```

```

    }
    catch (exception &e)
    {
        AfxMessageBox (e.what(),MB_OK,0);
    };
}

BOOL CDSP_ControlDlg::DestroyWindow()
{
    if (EingangsspektrumViewActive) EingangsspektrumView->DestroyWindow ();
    if (BandMittelwerteViewActive) BandMittelwerteView->DestroyWindow ();
    if (HuellkurvenViewActive) HuellkurvenView->DestroyWindow ();
    if (BandDynamikViewActive) BandDynamikView->DestroyWindow ();
    if (BandFaktorenViewActive) BandFaktorenView->DestroyWindow ();
    if (SpektralFaktorenViewActive) SpektralFaktorenView->DestroyWindow ();
    if (AusgangsspektrumViewActive) AusgangsspektrumView->DestroyWindow ();

    DSP_Command->BoardClose ();
    delete (DSP_Command);

    return CDialog::DestroyWindow();
}

void CDSP_ControlDlg::OnTimer(UINT nIDEvent)
{
#ifdef NO_DSP
    DSP_Command->Benachrichtige ();
#else
    if (m_Slider_MaxDaempfung.GetPos () != DSP_MaxDaempfung)
    {
        try
        {
            DSP_MaxDaempfung = m_Slider_MaxDaempfung.GetPos ();
            DSP_Command->Set_MaxDaempfung (DSP_MaxDaempfung);
        }
        catch (exception &e)
        {
            AfxMessageBox (e.what(),MB_OK,0);
        };
    };

    if (m_Slider_MaxDynamik.GetPos () != DSP_MaxDynamik)
    {
        try
        {
            DSP_MaxDynamik = m_Slider_MaxDynamik.GetPos ();
            DSP_Command->Set_MaxDynamik (DSP_MaxDynamik);
        }
        catch (exception &e)
        {
            AfxMessageBox (e.what(),MB_OK,0);
        };
    };

    if (m_Slider_Anstiegsfaktor.GetPos () != DSP_Anstiegsfaktor)
    {
        try
        {
            DSP_Anstiegsfaktor = m_Slider_Anstiegsfaktor.GetPos ();
            DSP_Command->Set_Anstiegsfaktor ((float)DSP_Anstiegsfaktor/(float)1000);
        }
        catch (exception &e)
        {
            AfxMessageBox (e.what(),MB_OK,0);
        };
    };

    if (m_Slider_Absinkfaktor.GetPos () != DSP_Absinkfaktor)
    {
        try
        {
            DSP_Absinkfaktor = m_Slider_Absinkfaktor.GetPos ();
            DSP_Command->Set_Absinkfaktor ((float)DSP_Absinkfaktor/(float)1000);
        }
        catch (exception &e)
        {
            AfxMessageBox (e.what(),MB_OK,0);
        };
    };

#endif

    CDialog::OnTimer(nIDEvent);
}

void CDSP_ControlDlg::OnSelchangeComboViewauswahl()
{
    if ((m_Combo_ViewAuswahl.GetCurSel () == 2) || (m_Combo_ViewAuswahl.GetCurSel () == 7))

```

ANHANG B. LISTING : DSP STEUERUNG    Mehrkanalige Störgeräuschunterdrückung

```
{
  m_Combo_BandAuswahl.EnableWindow (true);
}
else
{
  m_Combo_BandAuswahl.EnableWindow (false);
};
}

void CDSP_ControlDlg::OnBypass()
{
  try
  {
    DSP_Command->Set_Bypass (m_Button_Bypass.GetState () & 0x0003);
  }
  catch (exception &e)
  {
    AfxMessageBox (e.what(),MB_OK,0);
  };
};

void CDSP_ControlDlg::OnHilfe()
{
  // Aufrufen der Windows Hilfe
  WinHelp (1000, HELP_CONTEXT);
}
```

### B.3.3 DSP\_Control.h

```
// DSP_Control.h : main header file for the DSP_CONTROL application
//

#ifndef AFX_DSP_CONTROL_H__35F8C8E6_8E0A_11D3_B254_00C04F7159B6__INCLUDED_
#define AFX_DSP_CONTROL_H__35F8C8E6_8E0A_11D3_B254_00C04F7159B6__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h" // main symbols

////////////////////////////////////
// CDSP_ControlApp:
// See DSP_Control.cpp for the implementation of this class
//

class CDSP_ControlApp : public CWinApp
{
public:
    CDSP_ControlApp();

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDSP_ControlApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

    // Implementation
    //{{AFX_MSG(CDSP_ControlApp)
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DSP_CONTROL_H__35F8C8E6_8E0A_11D3_B254_00C04F7159B6__INCLUDED_)
```

### B.3.4 DSP\_Control.cpp

```
// DSP_Control.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "DSP_Control.h"
#include "DSP_ControlDlg.h"

#include "dspcmd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// Class to Control the DSP
DSP_Cmd* DSP_Command;

////////////////////////////////////
// CDSP_ControlApp

BEGIN_MESSAGE_MAP(CDSP_ControlApp, CWinApp)
//{{AFX_MSG_MAP(CDSP_ControlApp)
// NOTE - the ClassWizard will add and remove mapping macros here.
// DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

////////////////////////////////////
// CDSP_ControlApp construction

CDSP_ControlApp::CDSP_ControlApp()
{
// TODO: add construction code here,
// Place all significant initialization in InitInstance
}

////////////////////////////////////
// The one and only CDSP_ControlApp object

CDSP_ControlApp theApp;

////////////////////////////////////
// CDSP_ControlApp initialization

BOOL CDSP_ControlApp::InitInstance()
{
// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need.

#ifdef _AFXDLL
Enable3dControls(); // Call this when using MFC in a shared DLL
#else
Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

CDSP_ControlDlg dlg;
m_pMainWnd = &dlg;
int nResponse = dlg.DoModal();
if (nResponse == IDOK)
{
// TODO: Place code here to handle when the dialog is
// dismissed with OK
}
else if (nResponse == IDCANCEL)
{
// TODO: Place code here to handle when the dialog is
// dismissed with Cancel
}

// Since the dialog has been closed, return FALSE so that we exit the
// application, rather than start the application's message pump.
return FALSE;
}

```

---

## Anhang C Listing : Assembler

---

```

!*****
! Dies ist ein Ausschnitt aus dem Assemblercode 'FIRasm.S'
! Dieser Code wurde von Hand weiter optimiert.
! Interruptroutine zur Anwendung des FIR-Filters.
!*****

.global _spt0_asserted; _spt0_asserted: ! FUNCTION PROLOGUE:
spt0_asserted ! rtrts protocol, params in registers, DM stack,
doubles are floats
modify(i7,-3);
! saving registers:
dm(-2,i6)=r5;
dm(-3,i6)=r9;
dm(-4,i6)=r13;
.def end_prologue; .val .; .scl 109; .endif;
i4=227;
r13=255;
r9=m7;

TPERIOD=r13;
TCOUNT=r9;
r2=MODE2;
r2=FEXT r2 by 5:1;

BIT SET MODE2 32;
r2=TCOUNT;
r13=dm(i4,m5);
r2=fext r13 by 16:16 (se);
f4=float r2;
f1=dm(_NORM);
f0=f1*f4;
r2=fext r13 by 0:16 (se);
f4=float r2,f12=m5; !zero
i12=_hk+126;
dm(i1,1)=f0;
f5=f1*f4;
f0=dm(i1,m6),f4=pm(i12,m15);

!***** Schleife wird 126 mal durchlaufen *****

_L$40:
lcntr = 126, do _L$42-1 until lce; !until localcounter expired

f8=f0*f4,f12=f8+f12,f0=dm(i1,m6),f4=pm(i12,m15);

!*****

_L$42:
f12=f12+f8;
f8=f0*f4;
f12=f12+f8;

f2=dm(_CORR);
f4=f2*f12;
f2=f2*f5;
r4=TRUNC f4;
r2=TRUNC f2;
r8=65535;
r4=ashift r4 by 16;
r2=r2 and r8;
r4=r4 or r2;
dm(226)=r4;

BIT CLR MODE2 32;
r4=TCOUNT;
r2=m7;
r2=r2-r4;
r2=pass r2;
if ge jump (pc, _L$41) (DB);
f12=float r2;
f4= 0x33022334;

f8= 0x4f800000;
f12=f12+f8;
_L$41:

```

```
f4=f12*f4;
dm(_et)=f4;
! FUNCTION EPILOGUE:
  i12=dm(-1,i6);
  r5=dm(-2,i6);
  r9=dm(-3,i6);
  r13=dm(-4,i6);
  jump (m14,i12) (DB);!.return
  nop;
RFRAME;
```

---

## Anhang D Listing : Fixkomma

---

### D.1 Noiseerase.java

```

/*
*****
* Noiseerase.java *
*****
* Hauptprogramm des Algorithmus *
* *
* (Fixpunktversion) *
*****
* Geändert am : 22.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/

class Noiseerase
{
    public static void main(String[] args)
    {
        // Debugfile erstellen
        // DebugFile Debug = new DebugFile ("debugflt.xls");

        // Variablen erstellen
        int AnzFFTsTotal;
        int Samplerate;
        int FFTNumber;
        int AnzSamples;
        int AnzSamplesDone;
        short Channel;

        int WorkSamplesComplex_Real [] = new int [Defs.FFT_SIZE];
        int WorkSamplesComplex_Imag [] = new int [Defs.FFT_SIZE];
        int InputSamplesComplex1 [] = new int [Defs.FFT_SIZE];
        int InputSamplesComplex2 [] = new int [Defs.FFT_SIZE];
        int OutputSamplesComplex1 [] = new int [Defs.FFT_SIZE];
        int OutputSamplesComplex2 [] = new int [Defs.FFT_SIZE];

        BandWerteKlasse BandMinimalWerte [] = new BandWerteKlasse [2];
        for (int z1 = 0; z1 < 2; z1++) BandMinimalWerte [z1] = new BandWerteKlasse ();
        BandWerteKlasse BandMaximalWerte [] = new BandWerteKlasse [2];
        for (int z1 = 0; z1 < 2; z1++) BandMaximalWerte [z1] = new BandWerteKlasse ();
        BandWerteKlasse BandMittelWerte = new BandWerteKlasse ();
        BandWerteKlasse BandDynamikWerte = new BandWerteKlasse ();
        BandWerteKlasse BandFaktorenWerte = new BandWerteKlasse ();

        int SpektrumFaktoren [] = new int [Defs.FFT_SIZE];

        int Releasefaktor = (int)(0.99*(1<<Defs.FIXPOINT_SHIFT));
        int Raisefaktor = (int)(1.01*(1<<Defs.FIXPOINT_SHIFT));
        int Maxdaempfung = (int)(30*(1<<Defs.FIXPOINT_SHIFT));
        int Maxdynamik = (int)(30*(1<<Defs.FIXPOINT_SHIFT));

        //Titel und Daten anzeigen
        System.out.println("Mehrkanalige Störgeräuschunterdrückung");
        System.out.println("(c) 1999 by M.Jud & A. Eggenberger");
        System.out.println("");
        System.out.println("Releasefaktor : " + Releasefaktor);
        System.out.println("Raisefaktor : " + Raisefaktor);
        System.out.println("Maxdaempfung : " + Maxdaempfung);
        System.out.println("Maxdynamik : " + Maxdynamik);

        // Hilfsvariablen erstellen
        int z1;

        // Oeffnen des Source Waves
        try
        {
            WaveKlasse WaveSrc = new WaveKlasse ( "Streetprinzessin.wav" );
            WaveHeader WaveSrc_Header = WaveSrc.GetHeader ();

            // Oeffnen des Target Waves

```

```

WaveKlasse WaveTrg = new WaveKlasse ( "Output.wav", WaveSrc_Header.Channels, WaveSrc_Header.Samplerate, WaveSrc_Header.Bitaufloesung );
WaveHeader WaveTrg_Header = WaveTrg.GetHeader ();

// Infos über Header anzeigen
System.out.println( "Infos über Sourcefile");
System.out.println( "Samplingfrequenz : " + WaveSrc_Header.Samplerate);
System.out.println( "Channels : " + WaveSrc_Header.Channels);
System.out.println( "Bitaufloesung : " + WaveSrc_Header.Bitaufloesung);
System.out.println( "Datenbytes : " + WaveSrc_Header.AnzDatenbytes);

// Laenge bestimmen
AnzFFTsTotal = (int) Math.ceil( (double)WaveSrc_Header.AnzDatenbytes / (double)WaveSrc_Header.BytesPerSample / (double)(Defs.HALF_FFT_SIZE))-1;

// Samplerate übernehmen
Samplerate = WaveSrc_Header.Samplerate;

// Initialisierung des Algorhythmus
// Fensterfunktion erstellen
FensterKlasse Fenster = new FensterKlasse ( Defs.FFT_SIZE, FensterKlasse.HALBSINUS );

// FFT Klasse erstellen
FFTKlasse FourierTransformation = new FFTKlasse ( Defs.FFT_SIZE );

// Baender initialisieren
BaenderKlasse Baender = new BaenderKlasse ( WaveSrc_Header.Samplerate, Defs.FFT_SIZE, Releasefaktor, Raisefaktor, Maxdaempfung, Maxdynamik );

// Startwerte setzen, so dass der Effekt von Beginn an zu wirken beginnt
Baender.Preset_BandWerte (BandMinimalWerte [0],1000);
Baender.Preset_BandWerte (BandMinimalWerte [1],1000);
Baender.Preset_BandWerte (BandMaximalWerte [0],-1000);
Baender.Preset_BandWerte (BandMaximalWerte [1],-1000);

// Variablen des Prozesses initialisieren
FFTNumber = -1;
AnzSamples = Defs.FFT_SIZE;
AnzSamplesDone = 0;

while (AnzSamples != 0)
{
    // Anzahl FFT's zaehlen
    FFTNumber++;

// Stand des Prozesses ausgeben
System.out.println ("(" + FFTNumber + " / " + AnzFFTsTotal + ")");

// Einlesen des Eingabevektors
AnzSamples = WaveSrc.ReadSamples ();
AnzSamplesDone += AnzSamples;

for (Channel = 1;Channel <= WaveSrc_Header.Channels;Channel++)
{
    // InputSamplesComplex schieben
    if (Channel == 1) for (z1 = 0;z1<Defs.HALF_FFT_SIZE;z1++) InputSamplesComplex1 [z1] = InputSamplesComplex1 [z1+Defs.HALF_FFT_SIZE];
    else for (z1 = 0;z1<Defs.HALF_FFT_SIZE;z1++) InputSamplesComplex2 [z1] = InputSamplesComplex2 [z1+Defs.HALF_FFT_SIZE];

// Samples einlesen
if (Channel == 1) WaveSrc.GetSamples (InputSamplesComplex1,Channel,Defs.HALF_FFT_SIZE);
else WaveSrc.GetSamples (InputSamplesComplex2,Channel,Defs.HALF_FFT_SIZE);

// Debugdaten schreiben
/*for (int z2 = 0 ;z2 < Defs.FFT_SIZE;z2++)
{
    Debug.Write (""+InputSamplesComplex1 [z2]+ "\t");
};
Debug.Write ("\n");*/

// Fensterfunktion anwenden
if (Channel == 1) Fenster.UseWindowfactors ( InputSamplesComplex1, WorkSamplesComplex_Real );
else Fenster.UseWindowfactors ( InputSamplesComplex2, WorkSamplesComplex_Real );

// Signal ins Frequenzspektrum transformieren
FourierTransformation.Transformation (WorkSamplesComplex_Real,WorkSamplesComplex_Imag,WorkSamplesComplex_Real,WorkSamplesComplex_Imag);

// Debugdaten schreiben
/*for (int z2 = 0 ;z2 < Defs.FFT_SIZE;z2++)
{
    Debug.Write (""+WorkSamplesComplex_Real [z2]+ "\t");
};
Debug.Write ("\n");*/

// Berechnung der Bandmittelwerte
Baender.Get_BandAmplitudenMittelwerte (WorkSamplesComplex_Real,WorkSamplesComplex_Imag,BandMittelWerte);

// Debugdaten schreiben
/*for (int z2 = 0 ;z2 < BaenderKlasse.ANZAHL_BAENDER;z2++)
{
    Debug.Write (""+BandMittelWerte.Wert [z2]+ "\t");
};
}
}

```

```

};
Debug.Write ("\n");*/

// Berechnung der Bandmaximal -und Bandminimalwerte
Baender.Get_BandMaximalWerte (BandMittelWerte, BandMaximalWerte [Channel-1]);
Baender.Get_BandMinimalWerte (BandMittelWerte, BandMinimalWerte [Channel-1]);

// Debugdaten schreiben
/*for (int z2 = 0 ;z2 < BaenderKlasse.ANZAHL_BAENDER;z2++)
{
    Debug.Write (""+BandMinimalWerte[0].Wert [z2]+ "\t");
    Debug.Write (""+BandMittelWerte.Wert [z2]+ "\t");
    Debug.Write (""+BandMaximalWerte[0].Wert [z2]+ "\t");
};
Debug.Write ("\n");*/

// Berechnung der Dynamik
Baender.Get_BandDynamik ( BandMaximalWerte [Channel-1], BandMinimalWerte [Channel-1], BandDynamikWerte);

// Debugdaten schreiben
/*for (int z2 = 0 ; z2 < BaenderKlasse.ANZAHL_BAENDER;z2++)
{
    Debug.Write (""+BandDynamikWerte.Wert [z2]+ "\t");
};
Debug.Write ("\n");*/

// Berechnung der Bandfaktoren
Baender.Get_BandFaktoren ( BandDynamikWerte, BandFaktorenWerte);

// Debugdaten schreiben
/*for (int z2 = 0 ;z2 < BaenderKlasse.ANZAHL_BAENDER;z2++)
{
    Debug.Write (""+BandFaktorenWerte.Wert [z2]+ "\t");
};
Debug.Write ("\n");*/

// Spektralfaktoren berechnen
Baender.Get_SpektrumFaktoren ( BandFaktorenWerte, SpektrumFaktoren );

// Debugdaten schreiben
/*for (int z2 = 0 ;z2 < Defs.FFT_SIZE;z2++)
{
    Debug.Write (""+SpektrumFaktoren [z2]+ "\t");
};
Debug.Write ("\n");*/

// Neues manipuliertes Spektrum berechnen
Baender.Calculate_Spektrum ( SpektrumFaktoren, WorkSamplesComplex_Real, WorkSamplesComplex_Imag );

// Frequenzspektrum zuruecktransformieren
FourierTransformation.InversTransformation ( WorkSamplesComplex_Real,WorkSamplesComplex_Imag,WorkSamplesComplex_Real,WorkSamplesComplex_Imag );

// Fensterfunktion zum 2ten mal anwenden
Fenster.UseWindowfactors ( WorkSamplesComplex_Real, WorkSamplesComplex_Real );

// Zusammensetzen des Ausgangssignals
// Komplex addieren (1.Hälfte)
for (z1 = 0;z1<Defs.HALF_FFT_SIZE;z1++)
{
    if (Channel == 1) OutputSamplesComplex1 [z1] = OutputSamplesComplex1 [z1] + WorkSamplesComplex_Real [z1];
    else OutputSamplesComplex2 [z1] = OutputSamplesComplex2 [z1] + WorkSamplesComplex_Real [z1];
};

// Sample schreiben
if (Channel == 1) WaveTrg.SetSamples (OutputSamplesComplex1,Channel,0);
else WaveTrg.SetSamples (OutputSamplesComplex2,Channel,0);

// 2.Hälfte im Array ablegen
for (z1 = 0;z1<Defs.HALF_FFT_SIZE;z1++)
{
    if (Channel == 1) OutputSamplesComplex1 [z1] = WorkSamplesComplex_Real [Defs.HALF_FFT_SIZE+z1];
    else OutputSamplesComplex2 [z1] = WorkSamplesComplex_Real [Defs.HALF_FFT_SIZE+z1];
};
};

// Daten ins File schreiben
WaveTrg.WriteSamples (AnzSamples);
};

// Schliessen der Files
WaveTrg.CloseWave ();
WaveSrc.CloseWave ();

// Programmende anzeigen
System.out.println ("\n\nProgramm beendet !!!\n");
}
catch (java.io.IOException e)

```

```
{  
    System.out.println (e.getMessage ());  
}  
  
    // Debugfile schliessen  
    //Debug.Close ();  
}
```

## D.2 WaveKlasse.java

```

/*
*****
* WaveKlasse.java
*****
* Alles was mit Audiodateien zu tun hat, wird
* von diesem Modul erledigt
*
*****
* Geändert am : 22.11.1999
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud
*****
*/

import java.io.*;

class WaveKlasse
{
private boolean Readonly;
private WaveHeader Header;
private int SampleblockSize;
private RandomAccessFile Wavefile;

private byte Samples [];

private final byte MONO = 1;
private final byte STEREO = 2;

// Konstruktor der WaveKlasse
public WaveKlasse ()
{
Init ();
Readonly = true;
};

// Konstruktor der WaveKlasse (oeffnen eines Files)
public WaveKlasse ( String _Filename ) throws IOException
{
Init ();
Readonly = true;

OpenWave (_Filename, Readonly);

// Header einlesen
ReadHeader ();

SampleblockSize = Header.BytesPerSample;
};

// Konstruktor der WaveKlasse (erstellen eines neuen Files)
public WaveKlasse ( String _Filename, short _Channels, int _Samplerate, short _Bitaufloesung ) throws IOException
{
Init ();
Readonly = false;

OpenWave (_Filename, Readonly);

// neuer Header generieren
Header.Channels = _Channels;
Header.Samplerate = _Samplerate;
Header.Bitaufloesung = _Bitaufloesung;
Header.AnzDatenbytes = 0;

Header.BytesPerSample = (short)((Header.Bitaufloesung >> 3) * Header.Channels);
Header.BytesPerSecond = Header.Samplerate * Header.BytesPerSample;

SampleblockSize = Header.BytesPerSample;

Header.InfoLaenge = 0x10;
Header.Konstante = 0x01;

// "RIFF" + Filelaenge zahlen nicht (siehe Standard)
Header.Filelaenge = 32-8;//sizeof (Header)-8;

WriteHeader ();
};

// Destruktor
public void finalize () throws IOException
{
CloseWave ();
Done ();
};

// Block mit den naechsten Samples einlesen

```

```

public int ReadSamples () throws IOException
{
    int result = Defs.SAMPLES_PER_BLOCK;
    try
    {
        Wavefile.readFully ( Samples, 0, SampleblockSize * Defs.SAMPLES_PER_BLOCK );
    }
    catch (EOFException e)
    {
        result = 0;
    };
    return (result);
};

// Block mit den naechsten Samples schreiben
public int WriteSamples ( int _AnzahlSamples) throws IOException
{
    int result = _AnzahlSamples;
    Wavefile.write ( Samples, 0, SampleblockSize * _AnzahlSamples );
    Header.AnzDatenbytes += _AnzahlSamples*SampleblockSize;
    Header.Filelaenge += result*SampleblockSize;
    return (result);
};

// Header extern zugaenglich machen
public WaveHeader GetHeader ()
{
    return (Header);
}

// Samples in normalisierter Form holen
public void GetSamples ( int[] _Target, short _channel , int _offset)
{
    _channel--;
    if (Header.Channels == MONO)
    {
        if (Header.Bitaufloesung == 16)
        {
            for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
            {
                _Target[_offset+z1] = (int)((1<<Defs.FIXPOINT_SHIFT)*Normieren16Bit ((short)(
                    (((short) Samples [2*z1])) & 0x00FF)|
                    (((short) Samples [2*z1+1])<<8) & 0xFF00
                    ));
            };
        }
        else
        {
            for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
            {
                _Target[_offset+z1] = (int)((1<<Defs.FIXPOINT_SHIFT)*Normieren8Bit (Samples [z1]));
            };
        }
    }
    else // STEREO
    {
        if (Header.Bitaufloesung == 16)
        {
            for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
            {
                _Target[_offset+z1] = (int)((1<<Defs.FIXPOINT_SHIFT)*Normieren16Bit ((short)(
                    (((short) Samples [2*(z1*2+_channel)])) & 0x00FF)|
                    (((short) Samples [2*(z1*2+_channel)+1])<<8) & 0xFF00
                    ));
            };
        }
        else
        {
            for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
            {
                _Target[_offset+z1] = (int)((1<<Defs.FIXPOINT_SHIFT)*Normieren8Bit (Samples [z1*2+_channel]));
            };
        }
    };
};

// Samples in normalisierter Form setzen
public void SetSamples ( int[] _Source, short _channel , int _offset )
{
    _channel--;
    if (Header.Channels == MONO)
    {
        if (Header.Bitaufloesung == 16)
        {
            for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
            {
                short value = DeNormieren16Bit (_Source[_offset+z1]/(double)(1<<Defs.FIXPOINT_SHIFT));
                Samples [2*z1] = (byte) (value & 0x00FF);
            };
        }
    }
};

```

```

        Samples [2*z1+1] = (byte)(((value)>>8) & 0x00FF));
    };
}
else
{
    for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
    {
        Samples [z1] = DeNormieren8Bit (_Source[_offset+z1]/(double)(1<<Defs.FIXPOINT_SHIFT));
    };
}
else // STEREO
{
    if (Header.Bitaufloesung == 16)
    {
        for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
        {
            short value = DeNormieren16Bit (_Source[_offset+z1]/(double)(1<<Defs.FIXPOINT_SHIFT));
            Samples [2*(2*z1+_channel)] = (byte) (value & 0x00FF);
            Samples [2*(2*z1+_channel)+1] = (byte)(((value)>>8) & 0x00FF));
        };
    }
    else
    {
        for ( int z1=0 ; z1<Defs.SAMPLES_PER_BLOCK ; z1++ )
        {
            Samples [z1*2+_channel] = DeNormieren8Bit (_Source[_offset+z1]/(double)(1<<Defs.FIXPOINT_SHIFT));
        };
    };
};

// Variablen erstellen
private void Init ()
{
    Header = new WaveHeader();
    Samples = new byte [4*Defs.SAMPLES_PER_BLOCK];
}

private void Done ()
{
};

// Samplewert auf -1 bis +1 normieren
private double Normieren16Bit (short _Value)
{
    return ((double)_Value / (double)0x7FFF );
};

private double Normieren8Bit (byte _Value)
{
    double test;
    if (_Value >= 0)
    {
        test = (-1)*(0x80-(short)_Value) / (double)0x7F;
    }
    else
    {
        test = (0x80+(short) _Value) / (double)0x7F;
    };
    return ( test);
};

// Samplewert von -1 bis +1 denormalisieren
private short DeNormieren16Bit (double _Value)
{
    // ungültige Werte korrigieren
    if (_Value > 1.0) _Value = 1.0;
    if (_Value < -1.0) _Value = -1.0;
    short value = (short)(_Value * 0x7FFF);
    //System.out.println ("D : "+value);

    return (value);
};

private byte DeNormieren8Bit (double _Value)
{
    // ungültige Werte korrigieren
    if (_Value > 1.0) _Value = 1.0;
    if (_Value < -1.0) _Value = -1.0;

    byte value = (byte)(0x80-_Value * 0x7F);
    return ( value );
};

// 16Bit Wert aus File lesen (Little Endian - Big Endian)
protected short read16() throws IOException

```

```

{
    short v = 0;
    v |= Wavefile.readUnsignedByte();
    v |= Wavefile.readUnsignedByte() << 8;
    return v;
}

// 16Bit Wert in File schreiben (Little Endian - Big Endian)
protected void write16(short x) throws IOException
{
    Wavefile.writeByte( x & 0xff );
    Wavefile.writeByte( (x >> 8) & 0xff );
}

// 32Bit Wert aus File lesen (Little Endian - Big Endian)
protected int read32() throws IOException
{
    int v = 0;
    v |= Wavefile.readUnsignedByte();
    v |= Wavefile.readUnsignedByte() << 8;
    v |= Wavefile.readUnsignedByte() << 16;
    v |= Wavefile.readUnsignedByte() << 24;
    return v;
}

// 32Bit Wert in File schreiben (Little Endian - Big Endian)
protected void write32(int x) throws IOException
{
    Wavefile.writeByte( x & 0xff );
    Wavefile.writeByte( (x >> 8) & 0xff );
    Wavefile.writeByte( (x >> 16) & 0xff );
    Wavefile.writeByte( (x >> 24) & 0xff );
}

// 4 Zeichen Tag aus File lesen
protected String readTag() throws IOException
{
    byte [] ba = new byte[4];
    ba[0] = Wavefile.readByte();
    ba[1] = Wavefile.readByte();
    ba[2] = Wavefile.readByte();
    ba[3] = Wavefile.readByte();
    return new String(ba);
}

// Oeffnen des Files
private void OpenWave ( String _Filename, boolean _Readonly )
{
    if (_Readonly)
    {
        try
        {
            Wavefile = new RandomAccessFile (_Filename,"r");
        } catch (java.io.IOException ex)
        {
            System.out.println ("Error loading WAV-File");
        }
    }
    else
    {
        File deleteFile = new File (_Filename);
        deleteFile.delete ();
        try
        {
            Wavefile = new RandomAccessFile (_Filename,"rw");
        } catch (java.io.IOException ex)
        {
            System.out.println ("Error loading WAV-File");
        }
    }
};

// Positionierung des Dateizeigers
private void SeekWave ( long _Offset )
{
    try
    {
        Wavefile.seek (_Offset);
    } catch (java.io.IOException ex)
    {
        System.out.println ("Error in Fileseek");
    }
};

// Positionierung des Dateizeigers auslesen
private long WavePos ()
{
    long pos = 0;

```

```

try
{
    pos = Wavefile.getFilePointer ();
} catch (java.io.IOException ex)
{
    System.out.println ("Error in Fileseek");
}

return (pos);
};

// Header einlesen (Dateiposition geht verloren)
private int ReadHeader () throws IOException
{
    int result = 0;
    SeekWave ( 0 );

    //RIFF Chunk
    if( !readTag().equals("RIFF") )
        throw new IOException("No WAVE file (RIFF missing)");
    Header.Filelaenge = read32(); //length of all following data
    if( !readTag().equals("WAVE") )
        throw new IOException("No WAVE file (WAVE missing)");
    //FORMAT Chunk
    if( !readTag().equals("fmt ") )
        throw new IOException("No WAVE file (fmt missing)");
    Header.InfoLaenge = read32(); //length of this chunk
    Header.Konstante = read16(); //Constant
    Header.Channels = read16();
    Header.Samplerate = read32();
    Header.BytesPerSecond = read32();
    Header.BytesPerSample = read16();
    Header.Bitauflösung = read16();
    //DATA Chunk
    if( !readTag().equals("data") )
        throw new IOException("No WAVE file (data missing)");
    Header.AnzDatenbytes = read32();

return (result);
};

// Header schreiben (Dateiposition geht verloren)
private int WriteHeader () throws IOException
{
    int result = 0;
    SeekWave ( 0 );

    //RIFF Chunk
    Wavefile.writeBytes("RIFF");
    //length of all following data:
    write32(0x4+0x18+Header.AnzDatenbytes );
    Wavefile.writeBytes("WAVE");
    //FORMAT Chunk
    Wavefile.writeBytes("fmt ");
    write32(0x10); //length of this chunk
    write16((short)0x01); //Constant
    write16(Header.Channels);
    write32(Header.Samplerate);
    write32(Header.BytesPerSecond);
    write16(Header.BytesPerSample);
    write16(Header.Bitauflösung);
    //DATA Chunk
    Wavefile.writeBytes("data");
    write32(Header.AnzDatenbytes);

return (result);
};

// Wavefile schliessen
public void CloseWave () throws IOException
{
    if (Readonly == false)
    {
        WriteHeader ();
    };

try
{
    Wavefile.close ();
} catch (java.io.IOException ex)
{
    System.out.println ("Error in Fileseek");
}
};
};

```

## D.3 FensterKlasse.java

```

/*
*****
* FensterKlasse.java *
*****
* Uebernimmt die Fensterfunktionen *
* (Fixpunktversion) *
* *
*****
* Geändert am : 22.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/

import java.lang.Math;

class FensterKlasse
{
    public static final char RECHTECK = 1;
    public static final char HALBSINUS = 2;

    public int Windowfactors [];
    private boolean Initialized;
    private short Size;

    // Konstruktor
    public FensterKlasse ()
    {
        Initialized = false;
    };

    // Konstruktor (mit Fensterlaenge und Fensterform)
    public FensterKlasse ( short _Size, char _Windowfunction )
    {
        Initialized = false;
        Create_Windowfactors ( _Size, _Windowfunction );
    };

    // Destruktor
    public void finalized ()
    {
        Destroy_Windowfactors ();
    }

    // Fensterfunktion anwenden
    public void UseWindowfactors ( int[] _InVector, int[] _OutVector )
    {
        if (Initialized == true)
        {
            for ( int z1=0 ; z1<Size ; z1++ )
            {
                _OutVector[z1] = FixpointOps.Mul ( _InVector[z1], Windowfactors [z1]);
            };
        };
    };

    // Windowfaktoren erstellen
    private void Create_Windowfactors ( short _Size, final char _Windowfunction )
    {
        int z1 = 0;
        int pi = (int)(3.14159265359*(1<<Defs.FIXPOINT_SHIFT));

        if (Initialized == false)
        {
            // Speicher reservieren
            Size = _Size;
            Windowfactors = new int [Size];

            switch (_Windowfunction)
            {
            case RECHTECK :
            {
                for (z1 = 0; z1 < Size; z1++)
                {
                    Windowfactors [z1] = (1<<Defs.FIXPOINT_SHIFT);
                };
            };break;
            case HALBSINUS :
            {
                for (z1 = 0; z1 < Size>>1; z1++)
                {
                    Windowfactors [z1] = FixpointOps.Sin (z1*pi/(Size>>1-1));
                };
            };
            for (z1 = 0; z1 < Size>>1; z1++)
            {

```

```
    Windowfactors [Size-1-z1] = Windowfactors [z1];
};
};break;
    // Defaultmaessig wird Rechteckfenster erzeugt
    default :
{
    for (z1 = 0;z1 < Size;z1++)
    {
        Windowfactors [z1] = 1;
    };
};
    Initialized = true;
};
};

// Klasse abschliessen
private void Destroy_Windowfactors ()
{
    if (Initialized)
    {
        Initialized = false;
    };
};
};
```

## D.4 FFTKlasse.java

```

/*
*****
* FFTKlasse.java
*****
* Transformation zwischen Zeitbereich und
* Frequenzbereich
* (Fixpunktversion)
*****
* Geändert am : 22.11.1999
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud
*****
*/

import java.lang.Math;

class FFTKlasse
{
    public boolean Initialized;
    private int FFTSize;
    private short Potenz;

    // Konstruktor
    public FFTKlasse ()
    {
        Initialized = false;
    };

    // Konstruktor (mit Groesse der FFT)
    public FFTKlasse ( short _FFTSize )
    {
        Initialized = false;
        Init (_FFTSize);
    };

    // FFT ausführen
    public void Transformation ( int[] _Source_Real,int[] _Source_Imag, int[] _Target_Real, int[] _Target_Imag )
    {
        if (Initialized)
        {
            int pi = (int)(3.14159265359*(1<<Defs.FIXPOINT_SHIFT));
            int xdum, ydum, s, c, angle, argument;
            int l, j, k;
            int n1, n2 = FFTSize;

            // Falls Output nicht im Eingabevektor muss _Source kopoert werden
            if ( _Source_Real != _Target_Real )
            {
                for ( int z1 = 0; z1<FFTSize; z1++) _Target_Real [z1] = _Source_Real [z1];
            };
            if ( _Source_Imag != _Target_Imag )
            {
                for ( int z1 = 0; z1<FFTSize; z1++) _Target_Imag [z1] = _Source_Imag [z1];
            };

            // Transformation durchfuehren
            for( int z1 = 1; z1<=Potenz; z1++ )
            {
                n1 = n2;
                n2 /= 2;
                angle = 0;
                argument = (int)(2*pi/n1);
                for( j = 0; j<n2; j++ ) {
                    c = FixpointOps.Cos(angle);
                    s = -FixpointOps.Sin(angle);
                    for( int i = j; i<FFTSize; i+=n1 )
                    {
                        l = i+n2;
                        xdum = _Target_Real[i] - _Target_Real[l];
                        _Target_Real [i] = _Target_Real[i] + _Target_Real[l];
                        ydum = _Target_Imag[i] - _Target_Imag[l];
                        _Target_Imag[i] = _Target_Imag[i] + _Target_Imag[l];

                        _Target_Real[l] = FixpointOps.Mul (xdum,c) - FixpointOps.Mul (ydum,s);
                        _Target_Imag[l] = FixpointOps.Mul (ydum,c) + FixpointOps.Mul (xdum,s);
                    }
                    angle = (j+1)*argument;
                }
            };

            j = 0;

            for( int z2 = 0; z2<(FFTSize-1); z2++ )
            {

```



## D.5 BaenderKlasse.java

```

/*
*****
* BaenderKlasse.java *
*****
* Alle Funktionen des Algorithmus im *
* Frequenzbereich *
* *
*****
* Geändert am : 22.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/

import java.lang.Math;

class BaenderKlasse
{
    public static final short ANZAHL_BAENDER = 23;
    public static final int BAENDER [][] =
        {{0,100},{100,200},{200,300},{300,400},{400,510},{510,630},{630,770},
        {770,920},{920,1080},{1080,1270},{1270,1480},{1480,1720},{1720,2000},
        {2000,2320},{2320,2700},{2700,3150},{3150,3700},{3700,4400},{4400,5300},
        {5300,6400},{6400,7700},{7700,9500},{9500,45000}};

    public static final int DYNAMIKKORREKTUR [][][] =
        {{{(int)(3.52 * (1<<Defs.FIXPOINT_SHIFT))},
        (int)(1.94 * (1<<Defs.FIXPOINT_SHIFT)),(int)(4.86 * (1<<Defs.FIXPOINT_SHIFT))),
        {(int)(1.02 * (1<<Defs.FIXPOINT_SHIFT)),(int)(4.21 * (1<<Defs.FIXPOINT_SHIFT))},
        {(int)(2.77 * (1<<Defs.FIXPOINT_SHIFT)),(int)(5.46 * (1<<Defs.FIXPOINT_SHIFT))}}};

    public static final int DAEMPFUNGSFAKTOR [][][] =
        {{{(int)(1 * (1<<Defs.FIXPOINT_SHIFT))},(int)(0.708 * (1<<Defs.FIXPOINT_SHIFT))),
        {(int)(0.841 * (1<<Defs.FIXPOINT_SHIFT)),(int)(0.595 * (1<<Defs.FIXPOINT_SHIFT))},
        {(int)(0.917 * (1<<Defs.FIXPOINT_SHIFT)),(int)(0.649 * (1<<Defs.FIXPOINT_SHIFT))},
        {(int)(0.772 * (1<<Defs.FIXPOINT_SHIFT)),(int)(0.546 * (1<<Defs.FIXPOINT_SHIFT))}}};

    private WerteImBandKlasse WerteImBand [];

    private boolean Initialized;
    private int Samplingfrequenz;
    private int Frequenzabstand;
    private int AnzahlSpektralwerte;
    private int AbsinkFaktor;
    private int AnstiegsFaktor;
    private int MaximaleDaempfung;
    private int Dynamikgrenze;
    private int Steigung;

    // Konstruktor
    public BaenderKlasse ()
    {
        Initialized = false;
        Samplingfrequenz = 0;
        Frequenzabstand = 0;
        AnzahlSpektralwerte = 0;

        WerteImBand = new WerteImBandKlasse [ANZAHL_BAENDER];
        for (int z1 = 0;z1 < ANZAHL_BAENDER;z1++) WerteImBand [z1] = new WerteImBandKlasse ();
    };

    // Konstruktor (mit Wertuebergabe)
    public BaenderKlasse ( int _Samplingfrequenz, int _AnzahlSpektralwerte, int _AbsinkFaktor, int _AnstiegsFaktor,
        int _MaximaleDaempfung, int _Dynamikgrenze )
    {
        Initialized = true;
        Samplingfrequenz = _Samplingfrequenz;
        AnzahlSpektralwerte = _AnzahlSpektralwerte;
        Frequenzabstand = Samplingfrequenz / AnzahlSpektralwerte;

        WerteImBand = new WerteImBandKlasse [ANZAHL_BAENDER];
        for (int z1 = 0;z1 < ANZAHL_BAENDER;z1++) WerteImBand [z1] = new WerteImBandKlasse ();

        // AbsinkFaktor und AnstiegsFaktor werden uebernommen
        AnstiegsFaktor = _AnstiegsFaktor;
        AbsinkFaktor = _AbsinkFaktor;

        // Daempfungskurve
        MaximaleDaempfung = _MaximaleDaempfung;
        Dynamikgrenze = _Dynamikgrenze;
        Steigung = IntegerOps.Div (MaximaleDaempfung, (Dynamikgrenze>>Defs.FIXPOINT_SHIFT));

        // Bandwerte bestimmen
        Calculate_WerteImBand ();
    }
}

```

```

};

// Destruktor
public void finalize ()
{
};

// Mittelwert eines Bandes bestimmen
public void Get_BandAmplitudenMittelwerte ( int[] _Spektrum_Real,int[] _Spektrum_Imag, BandWerteKlasse _BandMittelwerte)
{
    if (Initialized)
    {
        // Baender durchzaehlen
        for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
        {
            _BandMittelwerte.Wert [Band] = 0;
            // Amplitudenwerte eines Bande quadratisch addieren
            for ( int WertNr = WerteImBand [Band].Erster ;
                WertNr<=WerteImBand [Band].Letzter ;
                WertNr++)
            {
                _BandMittelwerte.Wert [Band] = _BandMittelwerte.Wert [Band] + FixpointOps.Mul (_Spektrum_Real [WertNr],_Spektrum_Real [WertNr])+
                    FixpointOps.Mul (_Spektrum_Imag [WertNr],_Spektrum_Imag [WertNr]);
            };
            // Wurzel ziehen
            _BandMittelwerte.Wert [Band] = IntegerOps.Sqrt (IntegerOps.Div (_BandMittelwerte.Wert [Band],WerteImBand [Band].Anzahl));
        };
    };
    //else throw (BaenderException (2));
};

// Maximalhulkkurve erweitern
public void Get_BandMaximalWerte (BandWerteKlasse _BandMittelwerte, BandWerteKlasse _BandMaximalWerte)
{
    if (Initialized)
    {
        // Baender durchzaehlen
        for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
        {
            if (_BandMaximalWerte.Wert [Band] <= _BandMittelwerte.Wert [Band])
            {
                _BandMaximalWerte.Wert [Band] = _BandMittelwerte.Wert [Band];
            }
            else
            {
                int Old_Value = _BandMaximalWerte.Wert [Band];
                _BandMaximalWerte.Wert [Band] = FixpointOps.Mul (_BandMaximalWerte.Wert [Band], AbsinkFaktor);
                if (Old_Value == _BandMaximalWerte.Wert [Band]) _BandMaximalWerte.Wert [Band]--;
            };
        };
    };
};

// Minimalhulkkurve erweitern
public void Get_BandMinimalWerte (BandWerteKlasse _BandMittelwerte, BandWerteKlasse _BandMinimalWerte)
{
    if (Initialized)
    {
        // Baender durchzaehlen
        for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
        {
            if (_BandMinimalWerte.Wert [Band] > _BandMittelwerte.Wert [Band])
            {
                _BandMinimalWerte.Wert [Band] = _BandMittelwerte.Wert [Band];
            }
            else
            {
                int Old_Value = _BandMinimalWerte.Wert [Band];
                _BandMinimalWerte.Wert [Band] = FixpointOps.Mul (_BandMinimalWerte.Wert [Band], AnstiegsFaktor);
                if (Old_Value == _BandMinimalWerte.Wert [Band]) _BandMinimalWerte.Wert [Band]++;
            };
        };
    };
};

// Bandedynamik ermitteln
public void Get_BandDynamik (BandWerteKlasse _BandMaximalWerte, BandWerteKlasse _BandMinimalWerte, BandWerteKlasse _BandDynamikWerte)
{
    if (Initialized)
    {
        // Baender durchzaehlen
        for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
        {
            // Dynamik errechnen
            int Dynamik = 0;
            int Huelkurvenverhaeltnis = IntegerOps.Div (_BandMaximalWerte.Wert [Band]<<Defs.FIXPOINT_SHIFT, _BandMinimalWerte.Wert [Band]);
            int Bitstellen [] = new int [3];
        };
    };
};

```

```

    while (Huelkurvenverhaeltnis > 1)
    {
        // letzten 3 Bitstellen werden im Schiebblock abgelegt
        Bitstellen [2] = Bitstellen [1];
        Bitstellen [1] = Bitstellen [0];
        Bitstellen [0] = Huelkurvenverhaeltnis & 0x1;

        // Wert durch 2 dividieren
        Huelkurvenverhaeltnis = Huelkurvenverhaeltnis>>1;
        Dynamik = Dynamik + 6*(1<<Defs.FIXPOINT_SHIFT);
    };

    // Korrektur des Wertes
    Dynamik = Dynamik + DYNAMIKKORREKTUR [Bitstellen [2]][Bitstellen [1]][Bitstellen [0]]- (1<<Defs.FIXPOINT_SHIFT)*Defs.FIXPOINT_SHIFT*6;

    // Wert ausgeben
    _BandDynamikWerte.Wert [Band] = Dynamik;
};
};

// Bandfaktoren aus Dynamik berechnen
public void Get_BandFaktoren (BandWerteKlasse _BandDynamikWerte, BandWerteKlasse _BandFaktorenWerte)
{
    if (Initialized)
    {
        // Baender durchzaehlen
        for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
        {
            // Bandfaktoren berechnen
            int Bitstellen [] = new int [3];
            int DivResult;
            int Faktor;
            int Daempfung = MaximaleDaempfung - FixpointOps.Mul (Steigung, Math.min (Dynamikgrenze, _BandDynamikWerte.Wert [Band]));

            DivResult = IntegerOps.Div (Daempfung,6);
            Bitstellen [0] = ((DivResult >> (Defs.FIXPOINT_SHIFT-1)) & 0x1);
            Bitstellen [1] = ((DivResult >> (Defs.FIXPOINT_SHIFT-2)) & 0x1);
            Bitstellen [2] = ((DivResult >> (Defs.FIXPOINT_SHIFT-3)) & 0x1);

            Faktor = DAEMPFUNGSFAKTOR [Bitstellen[2]][Bitstellen[1]][Bitstellen[0]];
            Faktor = Faktor >> (DivResult>>Defs.FIXPOINT_SHIFT);

            _BandFaktorenWerte.Wert [Band] = Faktor;
        };
    };
};

// Spektrumfaktoren berechnen
public void Get_SpektrumFaktoren ( BandWerteKlasse _BandFaktorenWerte, int[] _SpektrumFaktoren )
{
    if (Initialized)
    {
        // Baender durchzaehlen
        for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
        {
            int WertNr;
            // Werte der ersten Haelfte des Spektrums
            for ( WertNr = WerteImBand [Band].Erster ; WertNr<=WerteImBand [Band].Letzter ; WertNr++)
            {
                _SpektrumFaktoren [WertNr] = _BandFaktorenWerte.Wert [Band];
            };

            // Werte der zweiten Haelfte des Spektrums
            for ( WertNr = (AnzahlSpektralwerte - WerteImBand [Band].Letzter - 1) ; WertNr<=(AnzahlSpektralwerte - WerteImBand [Band].Erster - 1) ; WertNr++)
            {
                _SpektrumFaktoren [WertNr] = _BandFaktorenWerte.Wert [Band];
            };
        };
    };
};

// Spektrum mit Hilfe der Spektralfaktoren manipulieren
public void Calculate_Spektrum ( int[] _SpektrumFaktoren, int[] _Spektrum_Real, int[] _Spektrum_Imag)
{
    if (Initialized)
    {
        // Spektralwerte durchzaehlen
        for ( int WertNr = 0 ; WertNr < AnzahlSpektralwerte ; WertNr++ )
        {
            _Spektrum_Real [WertNr] = FixpointOps.Mul (_Spektrum_Real [WertNr], _SpektrumFaktoren [WertNr]);
            _Spektrum_Imag [WertNr] = FixpointOps.Mul (_Spektrum_Imag [WertNr], _SpektrumFaktoren [WertNr]);
        };
    };
};

// Min- und Maxwerte vorsetzen
public void Preset_BandWerte (BandWerteKlasse _BandWerte, int _Wert)

```

```

{
  for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
  {
    _BandWerte.Wert [Band] = _Wert;
  };
};

// Spektrum in Baender aufteilen
private void Calculate_WerteImBand ()
{
  // Bestimmung, welche Werte in welchem Band liegen
  for ( int Band = 0 ; Band<ANZAHL_BAENDER ; Band++ )
  {
    float Frequenz = 0;
    WerteImBand [Band].Erster = -1;
    WerteImBand [Band].Letzter = -1;

    for ( int WertNr = 0 ; WertNr<(AnzahlSpektralwerte>>1) ; WertNr++)
    {
      // Erster Werte des Bandes bestimmen
      if (WerteImBand [Band].Erster == -1)
      {
        if (Frequenz >= BAENDER [Band][0]) WerteImBand [Band].Erster = WertNr;
      };

      // Letzter Wert des Bandes bestimmen
      if (Frequenz < BAENDER [Band][1]) WerteImBand [Band].Letzter = WertNr;

      // naechste Frquenz errechnen
      Frequenz += Frequenzabstand;
    };

    WerteImBand [Band].Anzahl = WerteImBand [Band].Letzter - WerteImBand [Band].Erster + 1;
  };
};
}
}

```

## D.6 defs.java

```
/*
*****
* Defs.java *
*****
* Klasse die ausschliesslich die allgemeinen *
* Konstanten beinhaltet *
* *
*****
* Geändert am : 22.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/

class Defs
{
    public static final byte  FIXPOINT_SHIFT      = 12;

    public static final short SAMPLES_PER_BLOCK = 256;
    public static final short FFT_SIZE = SAMPLES_PER_BLOCK << 1;
    public static final short HALF_FFT_SIZE = SAMPLES_PER_BLOCK;
}
```

## D.7 WaveHeader.java

```
class WaveHeader
{
    public byte Text1 [] = {(byte)'R',(byte)'I',(byte)'F',(byte)'F'}; // "RIFF"
    public int Filelaenge; // Filelaenge in Bytes
    public byte Text2 [] = {(byte)'W',(byte)'A',(byte)'V',(byte)'E'}; // "WAVE"
    public byte Text3 [] = {(byte)'f',(byte)'m',(byte)'t',(byte)' '}; // "fmt "
    public int Infolaenge; // Laenge des Infoblocks = 10h
    public short Konstante; // immer 01h
    public short Channels;
    public int Samplerate; // Samples pro Sekunde
    public int BytesPerSecond;
    public short BytesPerSample; // 2 oder 1 Byte
    public short Bitaufloesung; // 16 oder 8 Bit
    public byte Text4 [] = {(byte)'d',(byte)'a','t',(byte)'a'}; // "data"
    public int AnzDatenbytes; // Anzahl Datenbytes

    public WaveHeader ()
    {
    };
    public void finalized ()
    {
    };
};
```

## D.8 WerteImBandKlasse.java

```
class WerteImBandKlasse
{
    public int Erster;
    public int Letzter;
    public int Anzahl;

    public WerteImBandKlasse ()
    {
    };

    public void finalized ()
    {
    };
}
```

## D.9 BandWerteKlasse.java

```
class BandWerteKlasse
{
    public int Wert[];

    public BandWerteKlasse ()
    {
        Wert = new int [BaenderKlasse.ANZAHL_BAENDER];
    };

    public void finalized ()
    {
    };
}
```

## D.10 IntegerOps.java

```

/*
*****
* IntegerOps.java *
*****
* Klasse fuer Integeroperationen *
* * *
* * *
*****
* Geändert am : 22.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/

import java.lang.Math;

class IntegerOps
{
public IntegerOps ()
{

};

// Division von zwei 32Bit zahlen mit Multiplikationen
// nachgebildet
static public int Div (int _Divident, int _Divisor)
{
int MinValue = 0;
int MaxValue = _Divident;
int MidValue = 1;
int Fehler = 10;
//System.out.println ("Division : "+_Divident + " / "+_Divisor);
while (Math.abs (Fehler) > 1)
{
MidValue = (MaxValue+MinValue)>>1;
//System.out.println ("Division Work : Min :"+MinValue+" Max"+MaxValue + " Mid :"+MidValue+" Soll " + (int)(_Divident/(float)_Divisor));
if (MidValue*_Divisor > _Divident) MaxValue = MidValue;
else MinValue = MidValue;
Fehler = MaxValue - MinValue;
};
//System.out.println ("Result: " + MinValue);
return (MinValue);
};

// Wurzel aus einem 32Bit Wert ziehen
static public int Sqrt (int _Square)
{
//System.out.println ("Wurzel von : "+_Square);
int OldResult = 0;
int Result;
int AnzShift = 100;
int NewSquare;

while (AnzShift > 0)
{
Result = 1;
AnzShift = 0;
NewSquare = (Result + OldResult)*(Result + OldResult);
while (NewSquare < _Square)
{
Result = Result<<1;
AnzShift++;
NewSquare = (Result + OldResult)*(Result + OldResult);
};
Result = Result>>1;
//System.out.println ("NewSquare : "+NewSquare + " Result :"+Result+" OldResult : "+OldResult);
OldResult = OldResult | Result;
//System.out.println ("Anzahl Shift : "+AnzShift + " OldResult : "+OldResult);
};

//System.out.println ("Resultat : "+ OldResult);
return ( OldResult );
};
};

```

## D.11 FixpointOps.java

```

/*
*****
* FixpointOps.java *
*****
* Klasse fuer Fixpointoperationen *
* * *
* * *
*****
* Geändert am : 22.11.1999 *
*****
* (c) 1999 by Adrian Eggenberger & Mischa Jud *
*****
*/

import java.lang.Math;

class FixpointOps
{
public FixpointOps ()
{

};

// Addition von zwei Fixpunktzahlen (32Bit) mit Hilfe
// eines 64Bit Registers
static public int Add (int _Value1, int _Value2)
{
long temp = (long)_Value1 + (long)_Value2;
return ((int)temp);
};

// Multiplikation von zwei 32Bit zahlen mit Hilfe von
// einem 64Bit Register (Shift notwendig)
static public int Mul (int _Value1, int _Value2)
{
long temp = ((long)_Value1 * (long)_Value2)>>Defs.FIXPOINT_SHIFT;
return ((int)temp);
};

// Sinuswert generieren. Diese Funktion kann durch eine Tabelle
// ersetzt werden
static public int Sin (int _Value)
{
// Spaeter mit Tabelle zu loesen
float temp = (float)_Value/(float)(1<<Defs.FIXPOINT_SHIFT);
return ((int)(Math.sin (temp)*(1<<Defs.FIXPOINT_SHIFT)));
};

// Cosinuswert generieren. Diese Funktion kann durch eine Tabelle
// ersetzt werden
static public int Cos (int _Value)
{
// Spaeter mit Tabelle zu loesen
float temp = (float)_Value/(float)(1<<Defs.FIXPOINT_SHIFT);
return ((int)(Math.cos (temp)*(1<<Defs.FIXPOINT_SHIFT)));
};
};
};

```