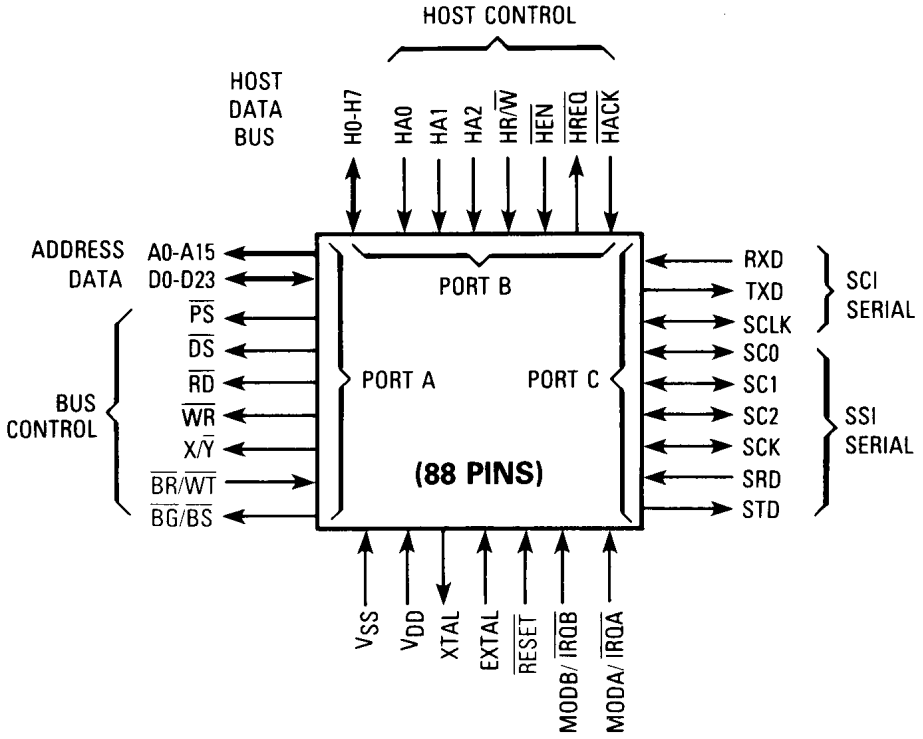
Praktikum Signalverarbeitung

mit dem Signalprozessor DSP56002
- Arbeitsunterlagen -

im Sommersemester 1998



Institut für Elektronik, TU Berlin

Prof. Dr.-Ing. Reinhold Orglmeister EN-537, ☎ 23362

Dipl.-Ing. Lutz Kahl EN-534, ☎ 24389, email: **L.Kahl@ ee.tu-berlin.de**

Dr.-Ing. Anselm Fabig EN-565, ☎ 24510, email: **anselm.fabig@t-online.de**

1 Einleitung 1
1.1 Der Ablauf der Laborveranstaltung 1
1.1.1 Zeitplan 1
1.1.2 Referatsthemen 2
1.1.3 Abschlußtermin 2
1.1.4 Abschlußbericht 3
1.1.5 Bewertung der persönlichen Leistung im Labor 3
1.2 Projektplanung 3
1.3 Alte Projektthemen 4
1.4 Tips für die Vorträge 5
2 Digitale Signalverarbeitung 7
2.1 Signalverarbeitung durch ein synchrones System 9
3 Der Signalprozessor DSP56002 11
3.1 Aufbau und Architektur 11
3.2 Registersatz 13
3.3 Adreßrechnung 14
3.4 Zahlenformat 17
3.5 Interrupt- und Exception-Verarbeitung 18
4 Das Prozessor-Board 20
4.1 Speicher 20
4.2 AD/DA-Umsetzer 21
4.3 Jumper und Steckerbelegungen 21
4.4 Spannungsversorgung 24
5 Die Entwicklungsumgebung 25
5.1 Ablauf bei der Programmentwicklung für den DSP 25
5.1.1 Assembler 25
5.1.2 Linker 26
5.1.3 Simulator 26
5.1.4 Debugger 27
5.2 Verzeichnis-Struktur 27
5.3 Bibliotheken 28
6 Tips zur Programmierung in Assembler 30
6.1 Hinweise für die SSI / SCI-Programmierung 31
6.2 Digitale Filter 32
6.2.1 FIR-Filter 32
6.2.2 IIR-Filter I 34
6.2.3 IIR-Filter II 35
6.2.4 IIR-Filter II 36
6.3 Programmaufteilung in Module 37
6.3.1 Sections 38
6.3.2 Speicherverwaltung 38
6.3.3 ORG-Anweisung 39
6.3.4 Benutzung der Adreßzähler 39
6.3.5 Projektgerüst 39
7 Beispielprogramme 41
7.1 Gleichrichten einer Folge von Abtastwerten 41
7.2 Daten über die SSI-Schnittstelle einlesen 43

7.3 Text über die SCI-Schnittstelle ausgeben 47
7.4 Berechnung des Effektivwerts einer Folge von Abtastwerten 50
7.5 Beispielprojekt Effektivwertmeßgerät 52
8 Literaturverzeichnis 56

Anhang A: Schaltpläne des DSP56002 EVM

Anhang B: Fractional and Integer Arithmetic

1 Einleitung

Der wachsenden Bedeutung der "Digitalen Signalverarbeitung" (DSV) entsprechend, wurde vor etwa zehn Jahren mit der Entwicklung von "Digitalen Signalprozessoren" (DSPs) begonnen. Diese Mikroprozessoren weisen Architekturen auf, die sich besonders gut zum schnellen Abarbeiten von Algorithmen der digitalen Nachrichtentechnik eignen. Dank ihrer guten Eigenschaften zur Realzeitverarbeitung werden DSPs zunehmend auch in anderen Bereichen der Prozeßtechnik (z.B. Robotersteuerung, Adaptive Regelung, etc.) eingesetzt.

Als einführende Literatur in die Digitale Signalverarbeitung sind die Werke von Oppenheim [17] , "Ludeman´s simple book" [10] und [21] sehr zu empfehlen. Weiterführend sind die Werke von Noll [15] und von Lüke [9] für das weite Gebiet der Signalübertragung hervorzuheben.

1.1 Der Ablauf der Laborveranstaltung

Im Projektlabor Signalverarbeitung (DSP-Labor) wird von den Laborteilnehmern gemeinsam ein selbst definiertes Projekt aus der Signalverarbeitung realisiert. Hauptziel ist das Kennenlernen des Signalprozessors DSP56002 von Motorola und der zugehörigen Entwicklungsumgebung. Zusätzliche soll die Herangehensweise an ein Gruppenprojekt[1] erlernt werden. Von besonderer Bedeutung hierbei ist die Aufteilung des Projektes in einzelne Teilaufgaben und die Definition der Schnittstellen zwischen den Teilaufgaben.

Zur Einarbeitung in das Signalprozessorsystem wird von jedem Teilnehmer ein Kurzreferat gehalten sowie eine Übungsaufgabe programmiert.

1.1.1 Zeitplan

Der folgende Zeitplan dient als Anhaltspunkt zum Ablauf der Labortermine. Zusätzlich sollte sich jede Gruppe regelmäßig außerhalb des Labors treffen, um organisatorische Fragen (Abstimmung der Module, Besprechung der Berichte, usw.) zu klären. Jeder Labortermin muß ausreichend vorbereitete werden (z.B. durch Struktogramme, Programmroutinen), um die Zeit beim Arbeiten mit dem Signalprozessor optimal auszunutzen.

VL-Woche	Inhalte
1	Vorstellung, Organisatorische Fragen, **Projektdiskussion,** Verteilung der Kurzreferate, Kennenlernen des Systems anhand von Beispielprogrammen.
2	Vier Kurzreferate, **Projektdiskussion.**
3	Drei Kurzreferate, Projektdefinition (Festlegung der Schnittstellen und Aufteilung in Module),
4 - 6	Arbeiten (intensiv).

[1] Siehe auch Absatz 1.2 *Projektplanung*.

7	Abgabe einer ausführlichen Projektbeschreibung (**Zwischenbericht**).
8	Arbeiten (intensiver):
9	**Lauffähige Minimalversion.**
10 - 14	Ausbau der Minimalversion.
15 (Letzte Woche)	Abschlußvortrag, Vorführung, Abgabe des Abschlußberichtes, Labordiskussion.

Tabelle 1

1.1.2 Referatsthemen

Ein Kurzreferat wird jeweils von einem Studenten der Gruppe ausgearbeitet und vorgetragen. Die Dauer des Vortrags sollte 10 min. nicht übersteigen. Es ist ein ein- bzw. zweiseitiges Papier anzufertigen.

a) Der DSP56002 (1)

- Besonderheiten des DSP gegenüber herkömmlichen Mikroprozessoren.

b) Erläutern Sie den Befehlssatz (Instruction Set) des DSP56002 anhand der verschiedenen Befehlsgruppen und ausgewählter Beispiele. (1)

c) Der grundlegende Aufbau eines Assemblerprogramms für den DSP56002 ist zu beschreiben. Insbesondere soll auf die nachfolgenden Eigenschaften und Befehle eingegangen werden: (1)

- Struktur eines Assembler-Files
- Was sind Pseudo-OpCodes?
- Labels
- EQU, DS, DC

d) Makroprogrammierung (1)

e) Linker und Modularisierung (2)

f) Der Simulator für den DSP56002 (2)

g) Die SSI- und SCI-Schnittstellen (2)

h) Programmieren in Matlab (2)

1.1.3 Abschlußtermin

Zwei Tage am Semesterende (in der Regel Montag und Dienstag in der letzten Vorlesungswoche) sind für die Vorstellung der Projektergebnisse vorgesehen. Hierbei nehmen alle **Teilnehmer** (auch vom Mikrocontroller- und Parallelverarbeitungslabor), die Betreuer sowie der zuständige Hochschullehrer teil. Jede Gruppe erhält 30 Minuten Redezeit, um ihr Projekt vorzustellen, an die sich eine praktische Vorführung anschließt. Am Ende des zweiten Tages erfolgt eine Abschlußdiskussion zur Art und Durchführung der Lehre.

1.1.4 Abschlußbericht

Der Abschlußbericht soll die von jeder Gruppe geleistete Arbeit dokumentieren, so daß das Projekt später von dritter Seite wieder in Betrieb genommen werden kann. Dazu gehören folgende Punkte:

- Zielsetzung,
- Projektdefinition,
- Modulaufteilung und Schnittstellenbeschreibung,
- Verwendete Hardware und Modelle,
- Beschreibung der Module (z.B. durch Struktogramme),
- Fazit,
- Abgabe aller kommentierten Programme auf einer Diskette (inkl. einer lauffähigen Gesamtversion

1.1.5 Bewertung der persönlichen Leistung im Labor

In die Bewertung des Gruppenteilnehmers gehen mehrere Faktoren ein:

- die *persönliche Leistung* des Studenten (Kurzreferat, Realisierung der Teilaufgabe, Abschlußvortrag etc.) mit einem Gewicht von ca. 2/3 und
- die *Leistung der Gruppe* (Funktionstüchtigkeit des gesamten Projekts, Abschlußvortrag, Vorführung etc.) mit einem Gewicht von ca. 1/3.

1.2 Projektplanung

Das Projekt soll im Team bearbeitet werden. Dazu gehört zum einen die Aufteilung der Aufgaben, zum anderen aber auch die ständige Kommunikation der Mitwirkenden, um eine problemlose Zusammenfügung der Ergebnisse zu erreichen.

Die Planung des Projekts ist entscheidend für den späteren Erfolg. Sie wird in der schriftlichen Spezifikation festgehalten. Zur Lösung der Aufgabe soll zunächst eine Minimalversion beschrieben werden, die durch eine spätere Ausbauversion erweitert werden kann, soweit die Zeit es zuläßt. Der angestrebte Projektfortschritt wird zu Beginn in einem Arbeitsplan festgehalten.

Ein größeres Assemblerprogramm läßt sich nur dann fehlerfrei erstellen, wenn es in kleine Einheiten zerteilt wird. Diese Module sollen eine inhaltlich geschlossene Einheit bilden und von jeweils einer Person bearbeitet werden. Für jedes Modul ist die Schnittstelle zur Außenwelt, d.h. zu einem Hauptprogramm festzulegen. Die Schnittstellen müssen so universell sein, daß sie im Laufe des Projekts nicht geändert werden müssen oder wenigstens leicht zu ändern sind. Ein Teilnehmer sollte bestimmt werden, um die Kompatibilität der Schnittstellen zu überwachen.

Zum Testen des eigenen Moduls ist es unerläßlich Testroutinen zu erstellen, die das Verhalten der anderen Module simulieren. Bei diesen Tests ist darauf zu achten, daß sowohl der korrekte Programmablauf als auch das Fehlverhalten überprüft wird. Nur so kann gewährleistet werden, daß das Zusammenfügen aller Module ohne größere Schwierigkeiten erfolgt.

Ein Projekt im Rahmen des DSP Labors muß immer folgende Elemente enthalten :

1) Signaltheoretische Betrachtungen zur Lösung der gewählten Aufgabenstellung.
2) Aufbau einer geeigneten Hardware und Anschluß an das verwendete DSP Board.
3) Entwurf und Programmierung einer Schnittstelle zwischen der selbstentworfenen Hardware und dem Prozessor.
4) Programmierung des unter 1) gefundenen Algorithmus' in DSP Assembler.

1.3 Alte Projektthemen

In den bisherigen Semestern wurden folgende Themen bearbeitet:

- **Ein System zur analogen Übertragung digital verschlüsselter Sprache.** Auf einer herkömmlichen, analogen Telefonleitung mit der gegebenen Bandbreite von 300 Hz bis 3,5 kHz sollte ein Sprachsignal derart übertragen werden, daß es für Lauscher unverständlich wird. Für einen berechtigten Empfänger sollte es regenerierbar sein.
- **Ein Analyzer / Equalizer für Audiosignale in CD-Qualität.** Ein digitalisiertes Audiosignal (Mono) wurde mit einer 8-kanaligen Filterbank selektiv manipuliert. Dabei wurde auf einer 8x11 LED-Matrix das (logarithmierte) Spektrum des Signals in Realzeit dargestellt.
- **Ein Wetterbildempfänger.** Das von einem Langwellenempfänger empfangene Signal einer Wetterfaxstation des deutschen Wetterdienstes sollte digital demoduliert werden und auf einer VGA-Grafikkarte dargestellt werden.
- **Eine Telefon Sprachmailbox.** Ein digitaler Anrufbeantworter mit DTMF Zielwahl für mehrere Empfänger und digitaler Aufzeichnung der Sprachen wurde realisiert. Dabei kam der GSM Speech-Codec für die Datenreduktion zur Anwendung, um Speicherplatz zu sparen.
- **Ein digitaler Funktionsgnerator.** Der DSP wurde zur Realzeitberechnung von Funktionswerten zur Erzeugung verschiedener Kurvenformen eingesetzt. Der Frequenzbereich reichte von einigen milli-Hz bis 50 kHz. Der Benutzer wurde über ein LC-Display interaktiv geführt.
- **Eine schwebende Kugel.** Der DSP wurde zur digitalen Regelung des Abstandes einer in einem Magnetfeld ‘hängenden’ Kugel eingesetzt. Im Rahmen des Laborprojektes wurden auch verschiedene alternative Abstandsmeßverfahren untersucht, um von der üblichen Lichtschranke in Verbindung mit einem Zweipunktregler wegzukommen.
- **Dolby-Surround.** Der DSP wurde als Matrixdekoder für das Dolby-Surround Verfahren eingesetzt. Dabei wird in nur zwei Kanälen (links und rechts) einer herkömmlichen Stereo-Aufzeichnung (wie sie bei allen neueren Kinofilmen auf VHS-Kassetten und Bildplatten zu finden ist), ein „Rundum“-Effekt codiert. die Wiedergabe geschieht mit vier Kanälen: Links-vorne, Mitte-vorne, Rechts-vorne und Hinten-links/rechts.
- **VOR-Empfang.** VOR ist ein Flugnavigationsverfahren der kommerziellen Luftfahrt im UKW-Bereich (108 - 118 MHz). Der DSP nimmt die FM-Demodulation eines Hilfsträgers und zwei Phasenwinkelmessungen vor. Anschließend stellt er das Ergebnis auf einem LC-Display dar.

- **Entfernungsmessung mittels Ultraschall.** Ein Meßsignal wird vom DSP über eine Ultraschallsender abgestrahlt. Anschließend wird das Echo empfangen und mit dem Meßsignal über eine Kreuzkorrelation verglichen. Die Zeitspanne zwischen dem Absenden des Meßsignals und der besten Übereinstimmung mit dem Empfangssignal ist ein Entfernungsmaß.
- **EEG-Meßgerät.** Es wurde ein EEG-Meßverstärker entwickelt, mit dem über zwei Sonden ein EEG gemessen und AD-konvertiert werden kann. Der DSP übernimmt anschließend die Auswertung des EEG-Signals. Eine anschließende Darstellung der Meßergebnisse erfolgt über einen PC.
- **Stereo Effektgerät.** Die Minimalversion erzeugt als Audioeffekt einen Hall-Effekt, wie man ihn in großen Räumen erleben kann. Aufbauend auf dieser Version wurde das Projekt durch einen digitalen Equalizer und weitere Effekte wie Stereo-Wide, Pseudostereo, reines Echo und Mono aus Stereo ausgebaut. Gesteuert wird der DSP über eine externe Hardware, die über ein LCD-Display mit Menüführung verfügt. Dort wird über Tasten der jeweilige Effekt und seine Intensität eingestellt.
- **Fernsehlogo Erkenner.** In der Regel blenden die Fernsehsender ihr Logo mit in das laufende Programm ein. Meistens erscheint es in der oberen linken Ecke des Bildes. Ausgenommen hiervon sind Werbesendungen. Hierbei wird das Senderlogo nicht mit übertragen.
  Das Projekt **Fernsehlogo Erkenner** nutzt diese Eigenschaft, um einen Videorekorder während der Aufnahme zu steuern. Solange ein Senderlogo vorhanden ist, zeichnet der Videorekorder auf. Wird der Spielfilm durch einen Werbeblock unterbrochen, verschwindet das Senderlogo und der Videorekorder wird gestoppt. Endet der Werbeblock wird der Aufnahmevorgang fortgesetzt.
- **MPEG Audio Dekoder.** Auf einer PCMCIA SRAM Karte kodiert gespeicherte Audioinformation wird in realzeit vom DSP dekodiert und über eine Stereoanlage ausgegeben.
- **Außerkopflokalisation**. Außerkopflokalisation und Richtungssteuerung einer Phantomschallquelle bei Kopfhörerwiedergabe von CD-Audiosignalen durch Modellierung der Außenohr-Übertragungsfunktion in der Horizontalebene.
- **Ultraschallmaus.** Die Ultraschallmaus dient als Ersatz für eine herkömmliche Maus. Angeschlossen wird sie an die serielle Schnittstelle eines Computer. Der Benutzer bewegt seine Hand in einem flachen, senkrechten Bereich, der von zwei Ultraschallsendern "ausgeleuchtet" wird. Vom DSP wird die Position der Hand innerhalb der Bereiches ermittelt und zu dem PC mit einem normalen Mausprotokoll übertragen.

1.4 Tips für die Vorträge

- Am Anfang des Vortrages sollt man sich unbedingt vorstellen und das Thema nennen.
- Nur wenn laut und deutlich gesprochen wird, kann jeder einen verstehen.
- Nicht 'gegen' die Tafel sprechen sondern in Richtung der Zuhörer.
- Nach Möglichkeit frei sprechen und nicht ablesen.
- Möglichst den Vortrag zu Hause vor dem Spiegel oder besser noch vor der/dem Freund/Freundin einmal vorsprechen.

- Die Hände sollten nicht in den Hosentaschen oder verschränkt vor dem Bauch sein.
- Die Füße sollten nicht auf dem Tisch oder auf einem Stuhl abgestellt werden.
- Während des Vortragens die Zuhörer ansehen, nicht nur den Professor oder die Assistenten sondern auch andere Studenten.
- Wird etwas auf der Overheadfolie gezeigt, dann sollte das nicht an der Wand sondern am Projektor geschehen.
- Wenn man nervös ist und mit den Händen zittert (das kann jedem passieren), dann sollte der Stift während des Zeigens abgelegt werden.
- Folien möglichst quer benutzen.
- Die Folien sollten in einem einheitlichen Layout sein.
- Es sollte nicht zu viel auf eine Folie gepackt werden.
- Sichtbare Farben für die Folien verwenden (gelb und orange sind schwer zu erkennen).
- Keine Bits und Bytes auf der Folie erwähnen, auch mit Zahlenwerten sollte sparsam umgegangen werden.
- Ein Programmausdruck gehört nicht auf eine Folie.
- Der Vortrag soll wirklich nur fünf Minuten lang sein, lieber kürzer als zu lang !!!
- Am Ende des Vortrages ist es besser, wenn noch abschließend etwas gesagt wird. Nicht einfach nur weggehen.

2 Digitale Signalverarbeitung

Nachfolgend sind einige Vorteile der "Digitalen Signalverarbeitung" gegenüber herkömmlicher analoger Technik aufgeführt :

- Genauigkeit und Dynamikbereich werden durch die Auflösung bestimmbar.
- Volle Reproduzierbarkeit.
- Zeitmultiplex sowie Parallelverarbeitung sind möglich.
- Hohe Zuverlässigkeit.
- Fehlerschutz- und Korrekturmöglichkeiten bei Speicherung und Übertragung sind vorhanden.
- Geringere Übersprechprobleme als analoge Systeme.
- Keine Probleme durch Bauteilealterung, Bauteiletoleranzen und Temperaturabhängigkeiten.
- Ein Abgleich ist nicht mehr notwendig (geringere Fertigungs- und Wartungskosten).
- Niedrige Systemkosten bei komplexen Systemen.

Am Beispiel der Audiosignalverarbeitung (die sich im übrigen gut für Laborprojekte an der Universität eignet) wird besonders der Vorteil der Reproduzierbarkeit deutlich. So können beliebig viele Generationen von Kopien ohne jeglichen Informations- und somit Qualitäts-verlust erstellt werden.

Signalmanipulationen wie Echounterdrückung, Dynamikkompression bzw. -Expansion und Frequenztransponierung lassen sich analog nur schwer verwirklichen.

Die moderne Bildverarbeitung, die u.a. auch der Medizin ständig neue Möglichkeiten eröffnet, wäre ohne DSV undenkbar.

Einige Nachteile im Vergleich zur analogen Verarbeitung sollen allerdings nicht verschwiegen werden. So sind die Eigenschaften in bezug auf Verlustleistung, Platzbedarf und Störstrahlung oft ungünstiger. Auch die Kosten für vergleichbare einfache Systeme liegen meist höher als bei ihrem analogen Pendant.

Beispiele digitaler Signalverarbeitungssysteme :

- CD-Spieler und DAT-Recorder
- Modems
- Mobiltelefonsysteme (GSM, PCN, DECT, ...)
- Digitaloszilloskop und Spektralanalyse
- Datenkompression und Reduktion

Den prinzipiellen Ablauf der digitalen Verarbeitung analoger Signale zeigt Bild 2.1. Nach erfolgter Tiefpaßfilterung zur Vermeidung von Spiegeleffekten (Aliasing) muß das analoge Signal mit Hilfe eines AD-Umsetzers digitalisiert werden. Dabei werden aus dem wert- und zeitkontinuierlichen Analogsignal zu bestimmten Zeitpunkten Proben entnommen und quantisiert. Das auf diese Weise gewonnene (zeit- und wertdiskrete) Digitalsignal besteht aus einer Folge von Zahlen (Abtastwerten), die ein Prozessor verarbeiten kann. Der gezeichnete Kanal zum Informationstransport soll im folgenden als ideal angenommen werden.

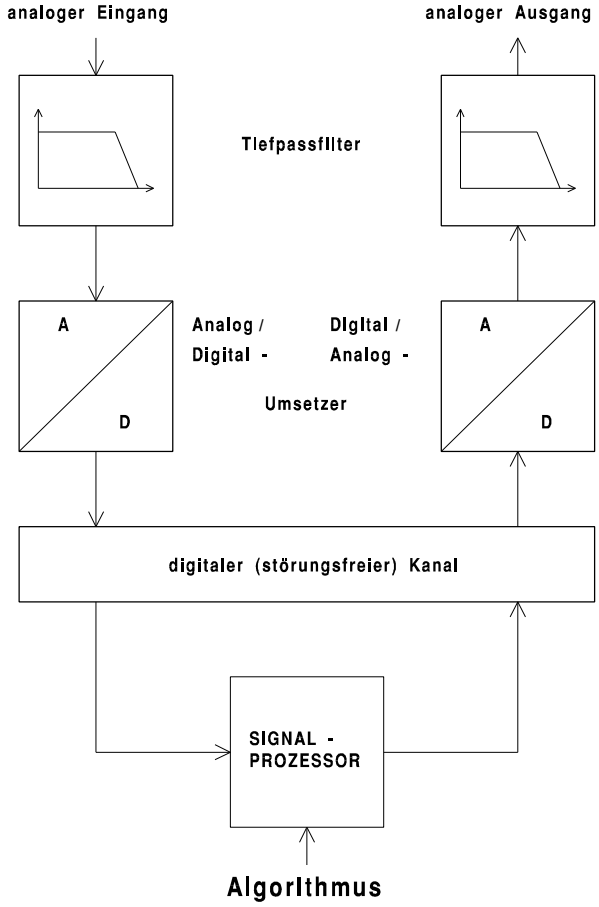


Bild 2.1

Falls ein analoges Ausgangssignal gewünscht ist, wird die vom DSP Errechnete Zahlenfolge mittels eines Digital/Analog-Umsetzers und anschließender Rekonstruktionsfilterung wieder in ein Analogsignal umgesetzt. Die Beschreibung von Abtastung und A/D- bzw. DA-Umsetzung soll hier auf das zum weiteren Verständnis unerläßliche Minimum beschränkt werden. Details zur AD/DA-Umsetzung sind z.B. in [17] zu finden.

Der AD-Umsetzungsprozeß beschreibt die ideale Umsetzung zeit- und wertekontinuierlicher Signale in zeit- und wertdiskrete Zahlenfolgen. Dieser Vorgang kann in Abtastung und Umsetzung unterteilt werden :

1.) Abtastung, ideal :

Das Zeitsignal wird mit einem Delta-Kamm III(t) multipliziert.

$$III(t) = \sum_{n=-\infty}^{\infty} \delta(t-n)$$

mit $$\int_{-\infty}^{\infty} \delta(t)dt = 1$$

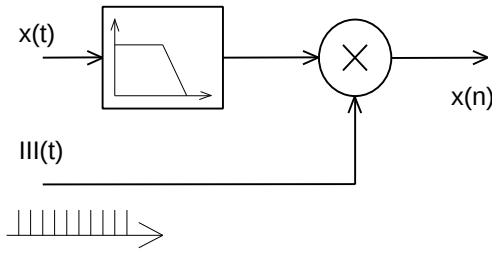


Bild 2.2, Abtastung des Analogsignals

In der Praxis wird der Abtastvorgang durch einen Sample & Hold-Baustein ausgeführt. Der Sample & Hold besteht aus einem schnellen Schalter, einem Kondensator und einem Impedanzwandler.

Da in der Praxis weder ideale Schalter noch Dirac-Impulse realisierbar sind, wird meist ein "Track & Hold" eingesetzt. Das bedeutet, der

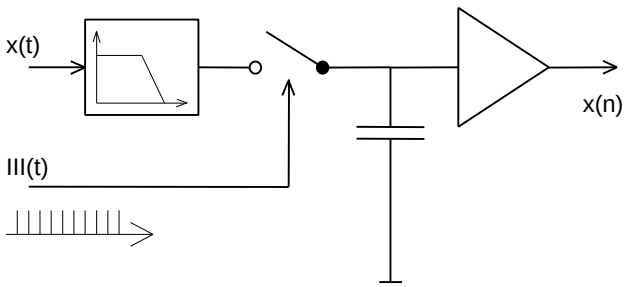
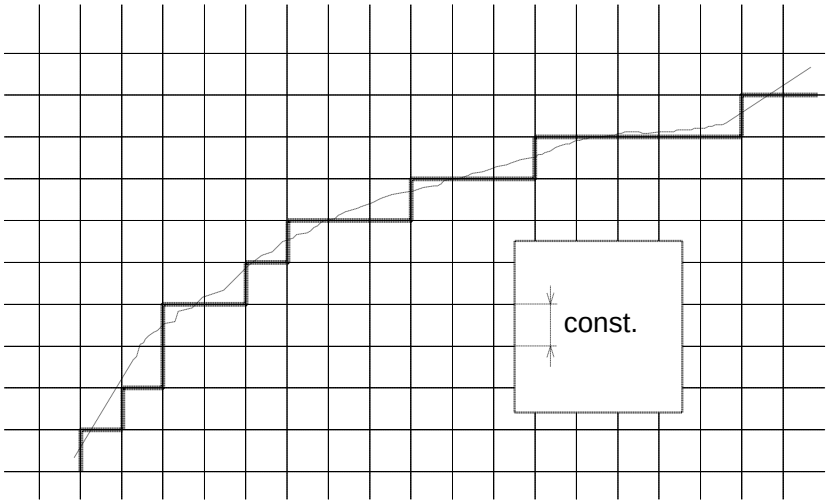


Bild 2.3, Prinzip des Sample & Hold

Kondensator folgt (Track) für die Dauer der High-Phase des Taktes dem Eingangssignal, um dann den Wert des Eingangssignals an der Stelle der fallenden Taktflanke des Taktes zu speichern.

2.) Umsetzung, ideal :



Das Eingangssignal muß gleichförmig quantisiert werden.

Bild 2.4, Gleichförmige Quantisierung

2.1 Signalverarbeitung durch ein synchrones System

Ein Beispiel für ein Signalverarbeitungssystem zeigt Bild 2.5.

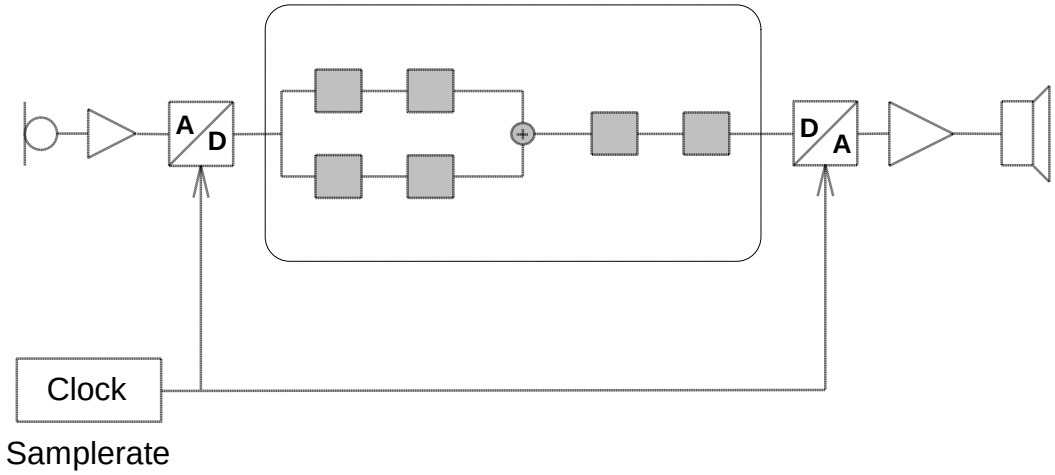


Bild 2.5

Beachtenswert ist der gemeinsame Takt für AD- und DA-Umsetzung. Es ist wichtig, daß der Takt für die AD- und DA-Umsetzung **phasenstarr** gekoppelt ist. Sie arbeiten **synchron**. AD- und DA-Umsetzrate müssen dagegen nicht zwingend gleich sein. Im allgemeinen wird aber ein gemeinsamer Taktgenerator verwendet, so daß diese Bedingung automatisch erfüllt ist.

Sehr wichtig in digitalen Signalverarbeitungssystemen ist eine hohe Konstanz und ein kleines Phasenjitter der Abtastrate. Da annähernd alle Algorithmen der DSV auf einem **äquidistant** abgetasteten Signal basieren.

Alle in Bild 2.5 grau unterlegten Blöcke stellen die eigentliche digitale Signalverarbeitung dar. Sie werden üblicherweise durch das Programm des DSP ausgeführt, nur selten realisiert man den digitalen Signalverarbeitungpfad in Hardware (Bzw. digitale Filterbausteine).

Die Darstellungsweise innerhalb des umrahmten Blocks in Bild 2.5 wird Signalflußdiagramm genannt.

3 Der Signalprozessor DSP56002

Für eine Entwicklung von Hard- und Software für den DSP56002 ist eine genau Kenntnis des Signalprozessors Grundvoraussetzung. Es sollen hier grundlegende Eigenschaften sowie einige Besonderheiten des DSP erklärt werden.

Die digitalen Signalprozessoren unterscheiden sich von herkömmliche Mikroprozessoren durch ihren internen Aufbau. Die Architektur des DSP56002 ist eine sogenannte 'Harvard-Architektur'. Sie ist gekennzeichnet durch die Trennung von Daten- und Programmspeicher. Herkömmliche Mikroprozessoren haben einen gemeinsamen Speicher für Daten und Programmcode, auch 'Von-Neumann-Architektur' genannt. Somit müssen Befehle und Daten nacheinander geladen werden. Bei der Signalverarbeitung ist aber ein hoher Datendurchsatz unerläßlich. Erst durch die Trennung von Befehls- und Datenspeicher wird ein paralleles Lesen von Befehlen und Daten ermöglicht und somit ein hoher Datendurchsatz erreicht. Durch die getrennten Speicher benötigt man allerdings auch separate Busse (Daten- und Programmbus). Die entsprechenden Adreßregister bzw. Adreßrechenwerke müssen getrennt aufgebaut sein.

Der DSP56002 läßt sich durch seinen Aufbau auch als Single-Chip Prozessor verwenden. An Peripherie sind dann nur noch ein Quarz, ein EPROM und 5V Spannungsversorgung notwendig. Der DSP bietet auf dem Chip diverse Schnittstellen, die die Integration in die Zielhardware wesentlich erleichtern. Im einzelnen sind dies :

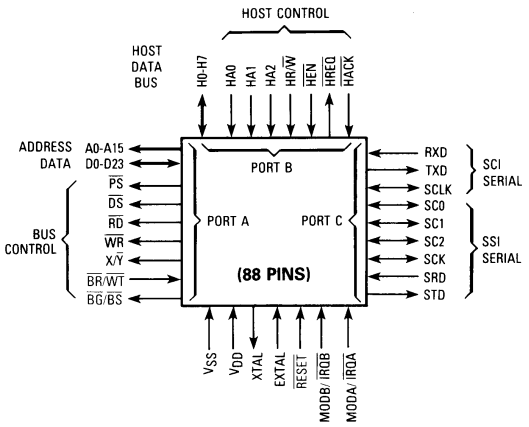


Bild 3.1

a) Eine synchrone im Timing für jedes Speichersegment einzeln programmierbare 24-Bit Busschnittstelle mit einfacher Bus-Arbitration-Logik und Erweiterungsmöglichkeit durch externen Speicher und Peripherie oder zum Anschluß an ein bestehendes MC-System. In der C-Maskenversion ist diese Busschnittstelle auch als asynchrone Schnittstelle programmierbar.

b) Eine 8-Bit Parallelschnittstelle (den sogenannten Host-Port) zur Verbindung mit einem Hostprozessor oder für schnellen parallelen Datentransfer (DMA).

c) Die Möglichkeit aus einem normalen 8 Bit EPROM nach dem Reset ein Programm Download (Boot) durchzuführen.

d) Eine synchrone und eine asynchrone Serienschnittstelle.

e) Zwei unabhängige Interrupteingänge.

3.1 Aufbau und Architektur

Der DSP56002 hat eine erweiterte 'Harvard-Architektur'. Neben dem Programm- und einem Datenbus hat er einen weiteren Datenbus und einen Steuerbus. Der DSP hat somit zwei Datenspeicher (X- und Y-Speicher). Daher können zwei Operanden, zum Beispiel für eine Multiplikation, gleichzeitig geladen werden.

Nach außen präsentiert sich der DSP56002 als CPU mit synchroner von-Neumann Busschnittstelle.

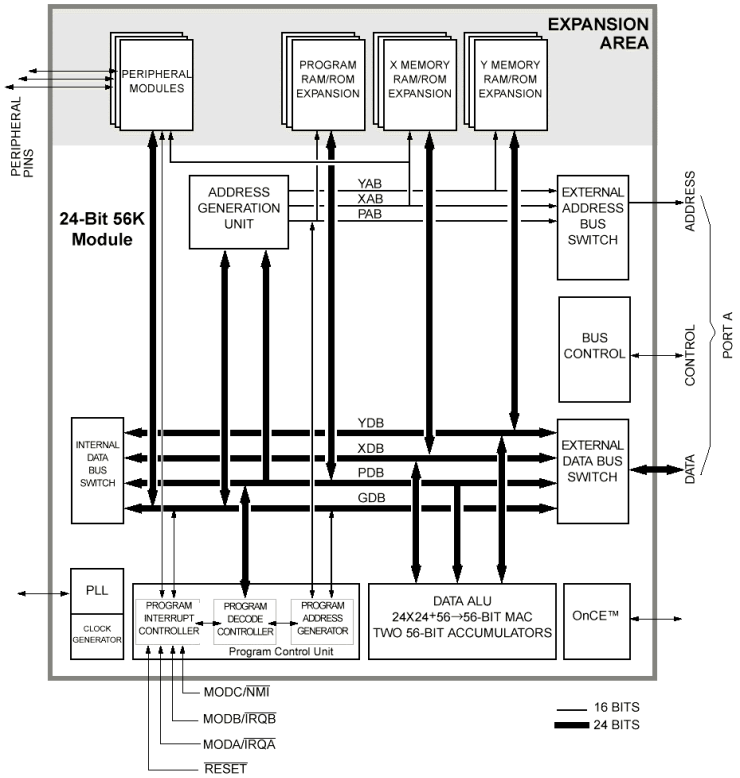


Bild 3.2

Aus dem DSP ist nur ein Bus herausgeführt. Auf Programm und Daten im externen Speicher kann daher nur sequentiell zugegriffen werden. Diese Lösung ist ein guter Kompromiß zwischen Geschwindigkeit und Gehäusegröße.

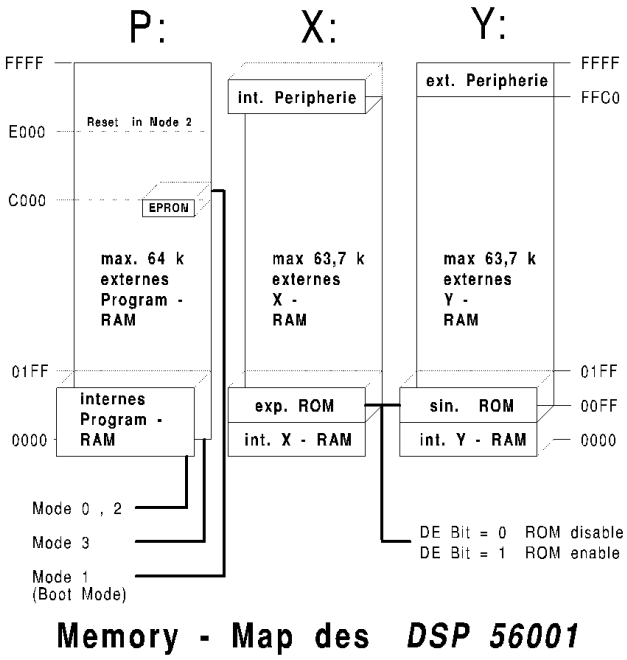
Der Buszugriff erfolgt bis zum Ablauf einer einstellbaren Anzahl von Wartezyklen (siehe BCR Register) synchron, danach kann ein Buszyklus durch ein externes Signal asynchron verlängert werden.

Der Speicher des DSP56002 ist unterteilt in drei Bereiche:

- Programmspeicher P:
- Datenspeicher X:
- Datenspeicher Y:

Auf alle drei Speichersegmente kann getrennt und unabhängig voneinander zugegriffen werden. Bei internen Zugriffen erfolgt der Zugriff während eines Taktzyklus'.

Auf dem Chip sind zwei ROMs vorhanden, in denen eine Sinus- und eine spezielle Logarithmus-Tabelle abgelegt sind.



Memory - Map des *DSP 56001*

Bild 3.3

Das Registermodell [12,13] des DSP56002 erinnert sehr an eine "normale" CPU, bis auf die zusätzlich vorhandenen Register für die Signalprozessorspezifischen Adressierungsarten und die hardware-loop Kontrollregister.

Der große Befehlssatz, [13] des DSP wird mit dem Nachteil einer fehlenden Orthogonalität erkauft. (Orthogonalität bedeutet, es können alle Operationen mit allen Registern ausgeführt werden). Es sind alle logischen und arithmetischen Operationen, wie man sie von einer modernen CPU erwartet (incl. DIVISION, MULTIPLIKATION, SHIFT, AND, OR, EOR, BITTEST und BITSET), vorhanden. Ihr Gebrauch ist aber an bestimmte Register (meist A1 und B1) gebunden.

3.2 Registersatz

Der DSP56002 verfügt über eine Vielzahl von unterschiedlichen Registern. Prinzipiell kann zwischen zwei Arten von Registern unterschieden werden.

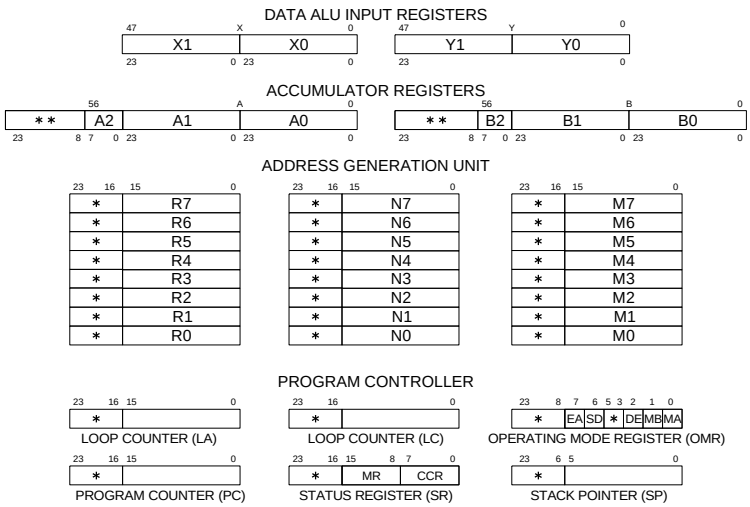


Bild 3.4: Der Registersatz des DSP56002

a) Die in Bild 3.4 dargestellten Register werden vom DSP zum Ausführen von Rechenoperationen und für die Adreßrechnung benötigt. Die Register werden durch ihre Namen angesprochen, es existieren keine expliziten Adressen.

   Die Akkumulator-Register A und B sowie die Daten-Eingaberegister X und Y werden für die arithmetischen und logischen Rechenoperationen des DSPs verwendet. Nach einer Rechenoperation kann das Ergebnis nur in eines der beiden Akkumulator-Register geschrieben werden. Die Daten für eine Rechenoperation können jedoch sowohl aus den Akkumulator-Registern als auch aus den Daten-Eingaberegistern stammen

   Die 24 Register der Address Generation Unit werden für die Adreßrechnung im DSP verwendet.

b) Die zweite Gruppe von Registern dient zum Konfigurieren und Programmieren der Schnittstellen des DSP56002. Diese Register werden über ihre Adresse angesprochen. Es werden vom DSP56002 z.B. Register für die Konfiguration der SSI- und SCI-Schnittstelle zur Verfügung gestellt.

3.3 Adreßrechnung

In der digitalen Signalverarbeitung können viele Probleme schon durch eine geschickte Adreßrechnung gelöst werden. Ein digitalisiertes "Signal" wird vom Prozessor als eine Folge von Zahlenwerten behandelt. Solche Folgen von Zahlenwerten werden im Speicher des DSP in einer bestimmten Reihenfolge abgelegt. Im Normalfall liegen zeitlich aufeinanderfolgende Werte (Samples) auch nacheinander im Speicher des Prozessors.

Nun benötigen aber bestimmte Algorithmen der digitalen Signalverarbeitung die Werte nicht sukzessive nacheinander, sondern in anderen Reihenfolgen.

Im DSP werden zur Bearbeitung der Daten nicht die Daten im Speicher verschoben, sondern es werden Adreßzeiger (pointer) auf die Datensätze verändert, um dann alle Quell- und Zieloperanden indirekt zu adressieren.

Von den Adressierungsarten des DSP56002 sind besonders erwähnenswert :

- Bitreverse (reverse carry) : diese Adressierungsart wird für die FFT benötigt. Man nennt sie auch reverse carry Arithmetik, da bei einer Adreßzähleraddition das carry Bit in der falschen Richtung, also von links nach rechts weitergegeben wird.
- Modulo-Arithmetik : mit dieser Adressierungsart können Ringzähler oder FIFOs verwaltet werden.
- Postinkrement und Postdekrement mit beliebigem Offset.

Der DSP56002 besitzt zur schnellen Adreßrechnung zwei voneinander unabhängige Adress-ALUs, denen je vier der acht ADR-Registersätze zugeordnet sind. Es können also zwei neue Adressen gleichzeitig berechnet werden. Die Adreßregister haben folgende Bedeutung :

R0 - R7 sind die Adreßzeiger (pointer) auf den Speicher.
N0 - N7 enthalten den Offset der Adreßoperation.
M0 - M7 bezeichnen die Adressierungsart, Normal, Modulo oder Bitreverse.

Die Registersätze 0 - 3 und 4 - 7 (Motorola nennt sie Register-Files) sind je einer der beiden Adress-ALUs zugeordnet. Bei allen Adreßoperationen wird zum verwendeten Adreßregister R0 bis R7 implizit immer der Inhalt der zugeordneten Offset- (N0 bis N7) und Modifier-Register (M0 bis M7) für die Berechnung der neuen Adresse mitverwendet.

In diesem Zusammenhang muß darauf hingewiesen werden, daß der Programmierer für eine gute Ausnutzung der vorhandenen Parallelität selbst verantwortlich ist.

Die Bedeutung der einzelnen Register läßt sich durch einige Beispiele gut erklären.

Beispiel 1: **`MOVE A0,X:(R0)`**

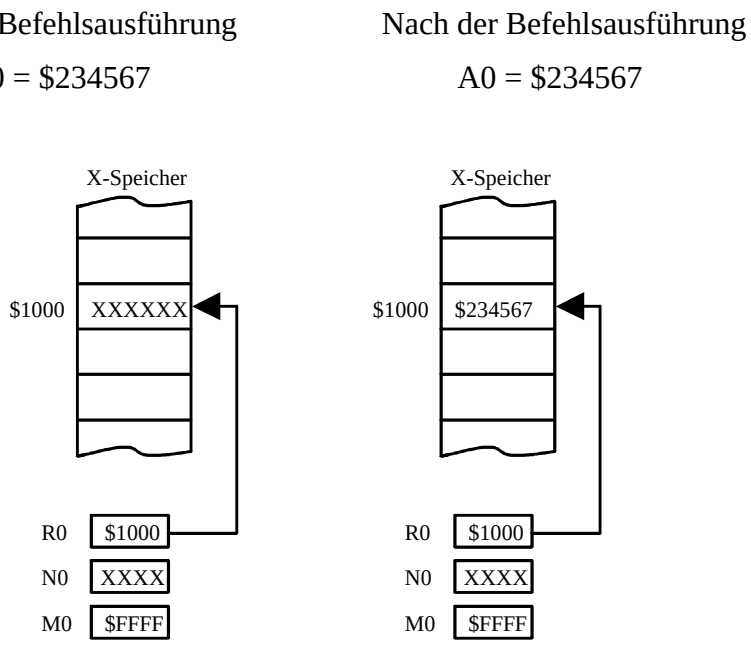


Bild 3.5

Beispiel 2: **`MOVE A0,X:(R0)+`**

Vor der Befehlsausführung
A0 = $234567

Nach der Befehlsausführung
A0 = $234567

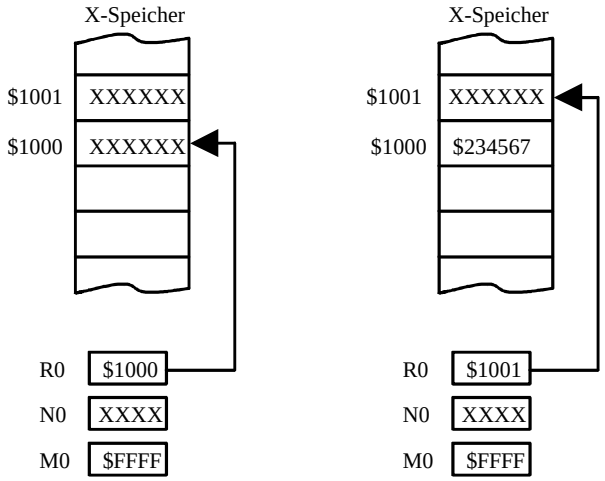


Bild 3.6

Beispiel 3: **`MOVE A0,X:(R0)+N0`**

Vor der Befehlsausführung
A0 = $234567

Nach der Befehlsausführung
A0 = $234567

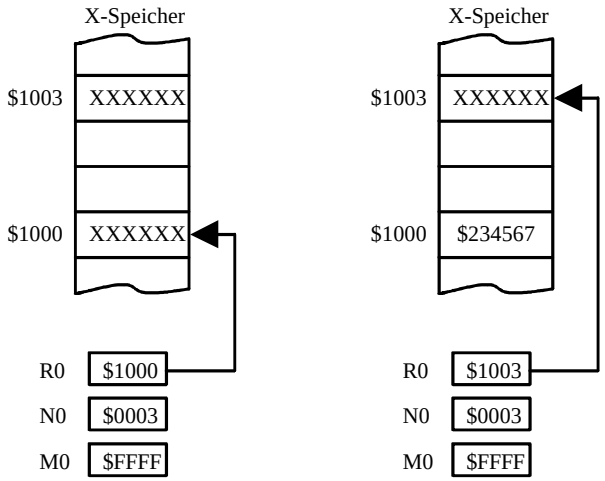


Bild 3.7

Das Moduloregister M0 wird für die Einrichtung von Ringzählern verwendet. Bei einem Ringzähler wird das Adreßregister R0 automatisch auf die Anfangsadresse zurückgesetzt, wenn das Ende des Ringzählers erreicht ist.

Beispiel 4:

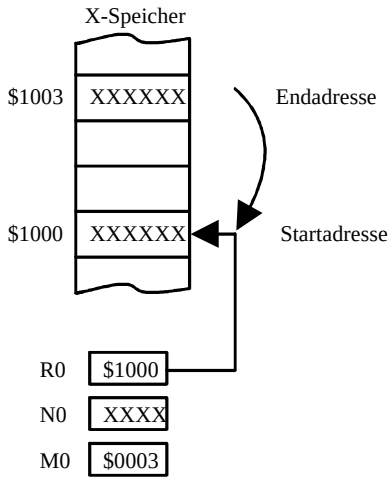


Bild 3.8

Das Adreßregister R0 soll als Startadresse den Wert $1000 enthalten. Das Moduloregister M0 ist mit $0003 geladen. wird nun der Adreßzeiger, zum Beispiel durch einen MOVE A0,(R0)+, erhöht, so wird bei der Berechnung der neuen Adresse das Moduloregister M0 mit berücksichtigt. Bei einem Stand von $1003 wird das Adreßregister R0 nicht auf $1004 erhöht, sondern es wird auf $1000 zurückgesetzt, wodurch das Adreßregister wieder auf den Anfang des Ringpuffers zeigt.

3.4 Zahlenformat

Der DSP56002 rechnet mit Zahlen in der sogenannten Fractional-Darstellung, dies ist ein 24 Bit Festkommaformat. Fractional bedeutet gebrochene Darstellung. Hierbei handelt es sich um eine andere Interpretation der Zweierkomplementdarstellung, die die Verwendung herkömmlicher Rechenwerke für Addition und Multiplikation erlaubt. Verglichen mit Zweierkomplementzahlen ist die Interpretation der Wertigkeit einer Zahl ist jedoch eine andere. In der Fractional-Darstellung haben die Bits folgende Bedeutung :

Ein gesetzte MSB in der Zahlendarstellung bedeutet ein negatives Vorzeichen der Zahl. Alle anderen Bits repräsentieren Brüche und zwar mit den folgenden Wertigkeiten (positive Zahl) :

Bit 23 = MSB ($0 \Rightarrow$ pos.),

Bit 22 = 1/2,

Bit 21 = 1/4,

Bit 20 = 1/8,

Bit 19 = 1/16, usw. bis

Bit 0 = $\frac{1}{2^{23}}$.

Daher kommt auch der Name gebrochene Darstellung. Eine negative Fractional-Zahl kann mit der bekannten Rechenregel für Vorzeichenumkehr von Zweierkomplementzahlen in eine positive Zahl umgewandelt werden. Dazu werden alle Bits negiert und ein LSB addiert. Da alle Zahlen in der Rechnung auf **± 1** normiert sind, muß Vorsorge für den Fall des Überlaufs in Richtung größerer oder kleinerer Zahlen getroffen werden. Im DSP56002 geschieht das derart, daß nach rechts einfach ein weiteres 24 Bit Register (A0) an den Akku angehängt wurde, so daß sich ein 48 Bit breiter Akku ergibt. Auf diese Weise wird die Auflösung auf $\frac{1}{2^{47}}$ gesteigert. Entsteht durch eine Operation ein Überlauf nach links, wird die Zahl also betragsmäßig größer als eins, so hat man weitere 8 Bit Akku-Extension-Register (A2) zur Verfügung, die dann als Summand von maximal 254 zum 48 Bit Resultat anzusehen sind. In

diesem Sinne entstehen die 56 Bit breiten (völlig gleichwertigen) Akkumulatoren A (A2:A1:A0) und B (B2:B1:B0).

Bei einem MOVE-Befehl in den Akku A oder B wird immer vorzeichenrichtig auf 56 Bit erweitert. Nach Rechenoperationen mit 48 oder 56 Bit Resultat muß das Ergebnis wieder auf 24 Bit normiert werden. Der DSP bietet hierzu schon eigene Befehle (z.B. MACR).

Über den Bus und zu peripheren Speichern oder Schnittstellen werden Ergebnisse nur mit 24 Bit Breite übertragen. Eine Ausnahme hiervon ist die L:memory reference, bei der mit einem MOVE Befehl je 24 Bit eines 48 Bit Resultates im externen X: und Y: Speicher getrennt abgelegt werden. Bild 9 zeigt die Wertigkeiten der Bits in den drei Genauigkeiten des DSP :

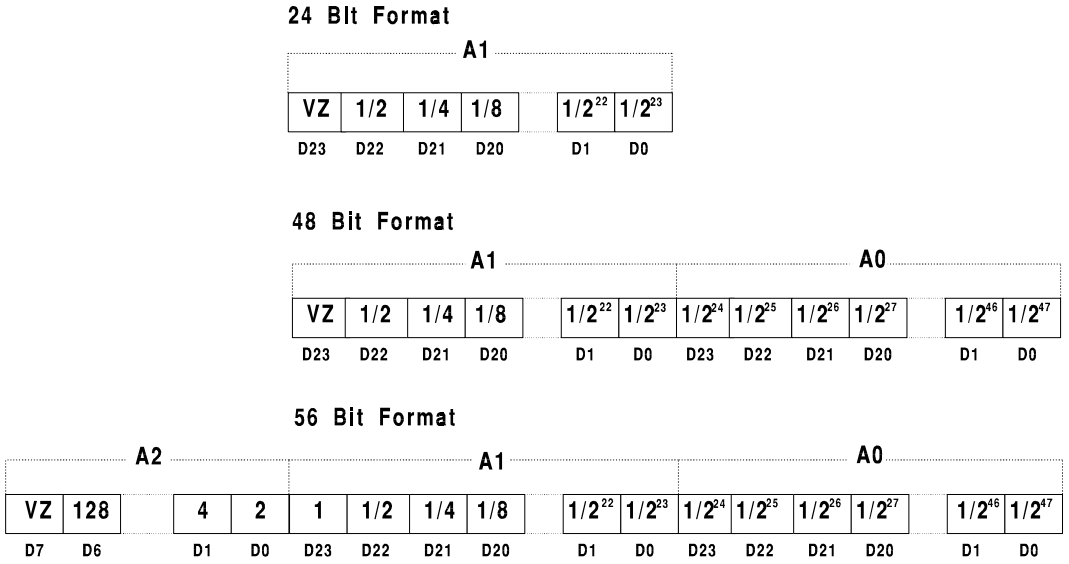


Bild 3.9

Mit 24 Bit lassen sich also :

- $1-\frac{1}{2^{23}}$ als größte positive Zahl und
- -1 (genau) als größte negative Zahl

darstellen.

3.5 Interrupt- und Exception-Verarbeitung

Der DSP56002 erlaubt eine sehr komfortable Verwendung und Verwaltung von Interrupts in drei Prioritätsebenen (die vom DSP unterstützten Exceptions sind im Tabelle 1 wiedergegeben).

Interrupt Start Adresse	IPL	Interrupt Quelle
P:$000 / P:$E000	3	Hardware Reset (ext.)
P:$0002	3	Stack Fehler
P:$0004	3	Trace
P:$0006	3	SWI
P:$0008	0-2	IRQA (ext.)
P:$000A	0-2	IRQB (ext.)
P:$000C-P:$0012	0-2	SSI
P:$0014-P:$001C	0-2	SCI
P:$001E	3	NMI (+10V)
P:$0020-P:$0024	0-2	HOST
P:$0026-P:$003C	0-2	Host-Command
P:$003E	0-2	Illegal Instruction

Tabelle 2

Die Ausnahmeverarbeitung ist, ähnlich der des MC68000, vektororientiert. Mit folgendem Unterschied: im MC68000 stehen an den ersten Adressen des Adreßbereiches nur die Zeiger (Vektoren), die auf den entsprechenden Exception- (Interrupt-) Handler zeigen. Im DSP56002 stehen auf den entsprechenden Adressen direkt die Befehle des Interrupt-Servers (Fast Interrupt).

Ist die Ausnahmeverarbeitung länger als zwei Befehle, so muß an der entsprechenden Vektoradresse ein Unterprogrammaufruf (JSR) zum Exception-Handler stehen. Der DSP erkennt, ob innerhalb von zwei Zyklen nach Auftreten eines Interrupts ein Sprung ausgeführt wurde, um nur dann das Statusregister (SR) und den Programmzähler (PC) auf den Stack zu retten und die Exception wie eine normale CPU bis zum Auftreten des RTI-Befehls (Long Interrupt) durchzuführen.

Im ersten Fall, dem sogenannten Fast-Interrupt, wird das Statusregister nicht gerettet. Es darf deshalb von den zwei Befehlen des Exception-Handlers auch nicht verändert werden. Diese Möglichkeit gestattet es, sehr schnell auf Interrupts zu reagieren (z.B. einen neuen Abtastwert vom ADC in den Speicher zu übertragen), ohne den lästigen Overhead einer normalen Interruptverarbeitung.

4 Das Prozessor-Board

Die folgende Abbildung zeigt das Blockschaltbild der externen DSP-Karte EVM56k.

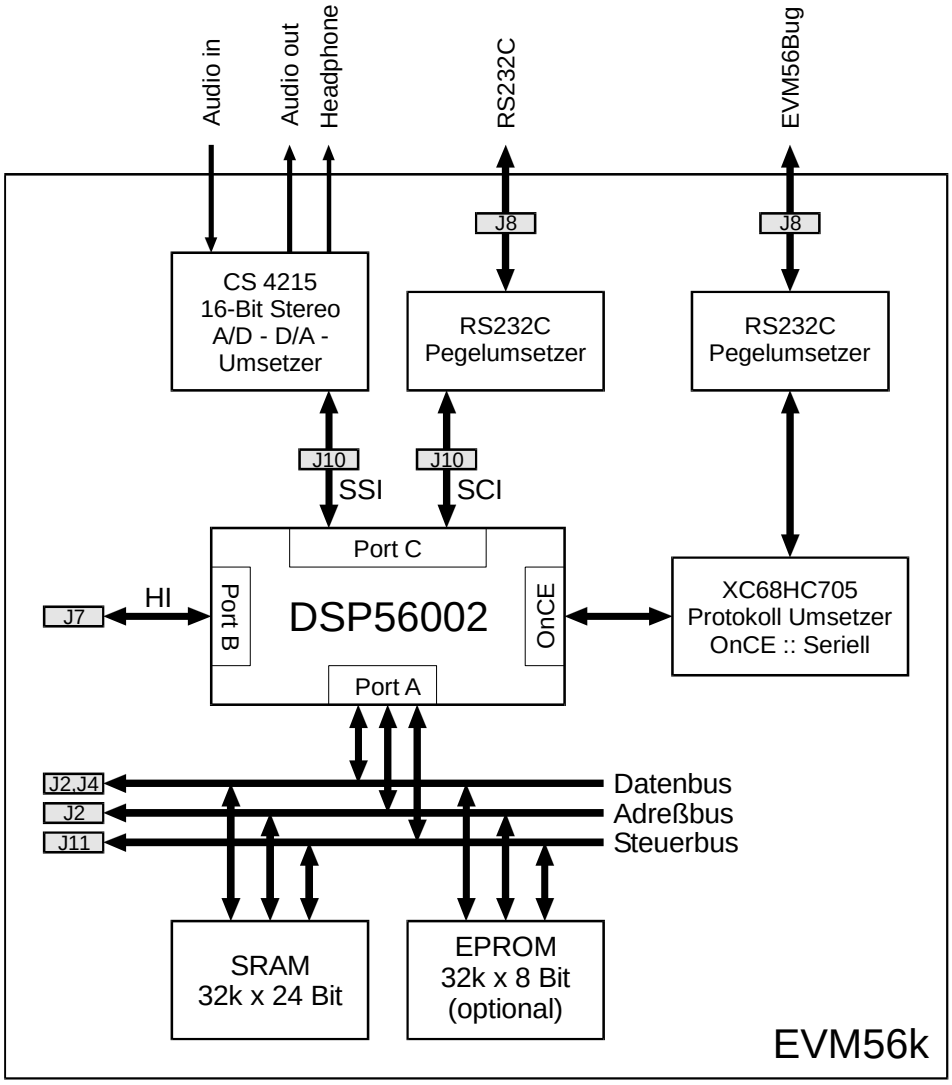


Bild 4.1

Den Kern bildet ein mit 40 MHz getakteter **DSP56002**. Weiterhin enthält die Karte :

4.1 Speicher

Das Board enthält 32k x 24Bit externes statische RAM ohne Wartezyklen zusätzlich zum internen Speicher des DSP56002. Weiterhin gibt es einen Sockel für ein EPROM mit maximal 32kByte. Der Speicher ist über den Port A mit dem DSP verbunden. Für die Aufteilung des externen Speichers auf die drei Adreßbereiche **`p:`**, **`x:`** und **`y:`** stehen zwei Konfigurationen zur Verfügung, die mit dem Jumper J12 ausgewählt werden können.

Das EPROM ist dem Adreßraum p: und x: unter den Adressen $8000..$FFFF zugeordnet.

4.2 AD/DA-Umsetzer

Auf dem Entwicklungsboard befindet sich ein hochwertiger integrierter 16Bit Stereo AD/DA-Umsetzer. Der Baustein CS4215 hat folgende Leistungsdaten:

- Stereo
- Abtastrate von 4kHz bis 50kHz
- 16 Bit linear, 8 Bit linear, µ-Law oder A-Law Kodierung der Audiodaten
- Integrierte Anti-Aliasing und Smoothing Filter
- Serielle synchrone Schnittstelle
- Programmierbare Verstärkung für die analogen Eingänge
- Programmierbare Dämpfung für den analogen Ausgang

4.3 Jumper und Steckerbelegungen

Das Entwicklungsboard EVM56k kann über verschiedene Jumper konfiguriert werden. Alle Signal- und Busleitungen sind über Steckerleisten zugänglich.

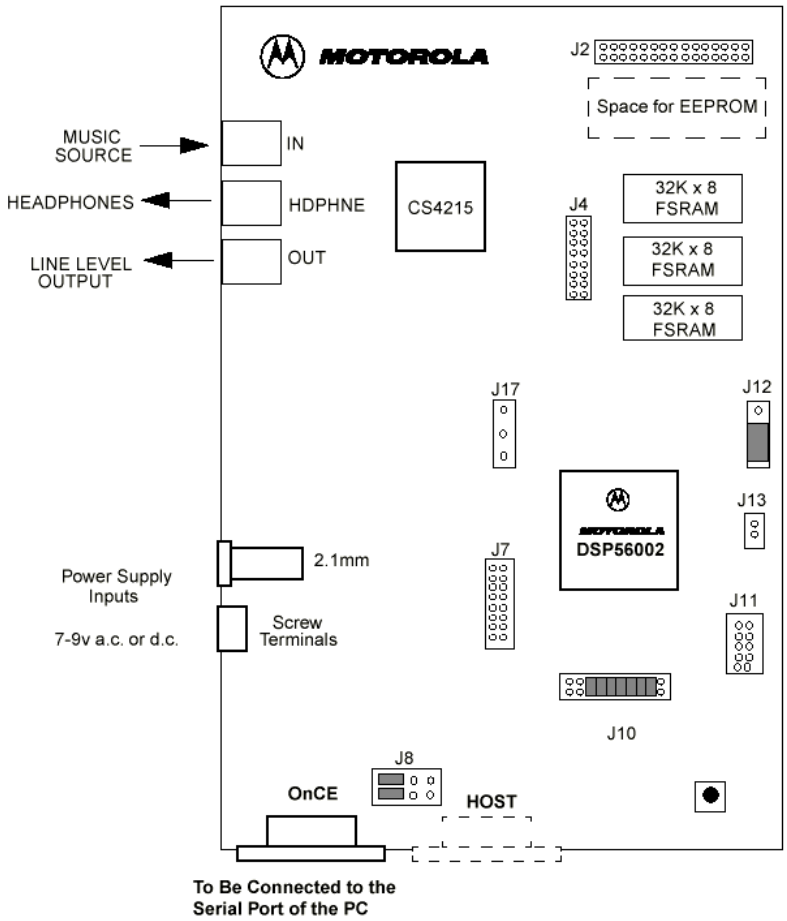


Bild 4.2: Bauteileanordnung auf dem EVM56k

Im folgenden wird die Pinbelegung der einzelnen Jumper erläutert:

Steckerleiste J2

Die folgenden Signale sind an die Steckerleiste J2 geführt:

- Adreßbus: DAB0..DAB15
- Datenbus: DBB0..DBB7
- Steuerbus: /WR, /RD
- Versorgungsspannung: Vcc, GND

/WR	1	2	Vcc
DAB15	3	4	DAB12
DAB7	5	6	DAB14
nc	7	8	DAB13
DAB6	9	10	DAB8
DAB5	11	12	DAB9
DAB4	13	14	DAB11
DAB3	15	16	/RD
nc	17	18	DAB10
DAB2	19	20	DBB7
DAB1	21	22	DBB6
DAB0	23	24	DBB5
DBB0	25	26	DBB4
DBB1	27	28	DBB3
DBB2	29	30	GND

Steckerleiste J4

An die Steckerleiste J4 sind die oberen 16 Bit des Datenbusses geführt.

DDB23	1	2	DDB22
DDB21	3	4	DDB20
DDB19	5	6	DDB18
DDB17	7	8	DDB16
DDB15	9	10	DDB14
DDB13	11	12	DDB12
DDB11	13	14	DDB10
DDB9	15	16	DDB8

Steckerleiste J7

An die Steckerleiste J7 sind die Signale des DSP Port B (Host Port) geführt.

HA0	1	2	/HACK
HA1	3	4	/HEN
HA2	5	6	HR//W
/HREO	7	8	H6
H7	9	10	H4
H5	11	12	H2
H3	13	14	H0
H1	15	16	GND

Steckerleiste J8

Die Steckerleiste dient zum Konfigurieren der Signalrichtungen der beiden seriellen Schnittstellen P4 (OnCE) und P5 (Terminal) des Entwicklungsboards.

P5: Pin 2	1	2	SCI Rx
SCI Tx	3	4	P5: Pin3
P4: Pin2	5	6	OnCE Rx
OnCE Tx	7	8	P4: Pin3

Steckerleiste J10

An die Steckerleiste J10 sind die Signale des DSP Port C (SCI- und SSI Schnittstellen), des AD/DA-Umsetzers und der 9-poligen Sub-D Buches geführt.

Die Steckerleiste kann so kofiguriert werden, daß der AD/DA-Umsetzer mit der SSI-Schnittstelle und die Sub-D Buchse mit der SCI-Schnittstelle des DSP verbunden wird. Das entspricht der Standardeinstellung des Entwicklungsmoduls.

SCI RX	1	2	PC0
SCI TX	3	4	PC1
D/C	5	6	PC2
nc	7	8	PC3
SCK	9	10	PC6
SSI FS	11	12	PC5
SSI TX	13	14	PC8
C RST	15	16	PC4
SSI RX	17	18	PC7
nc	19	20	GND

Steckerleiste J11

An die Steckerleiste J11 ist der Steuerbus des DSP geführt. Ausgenommen sind die beiden Steuersignale /RD und /WR, die an die Steckerleiste J2 geführt sind.

X//Y	1	2	/DS
/BS	3	4	/PS
/BG	5	6	/BR
/WT	7	8	/BN
TIO	9	10	GND

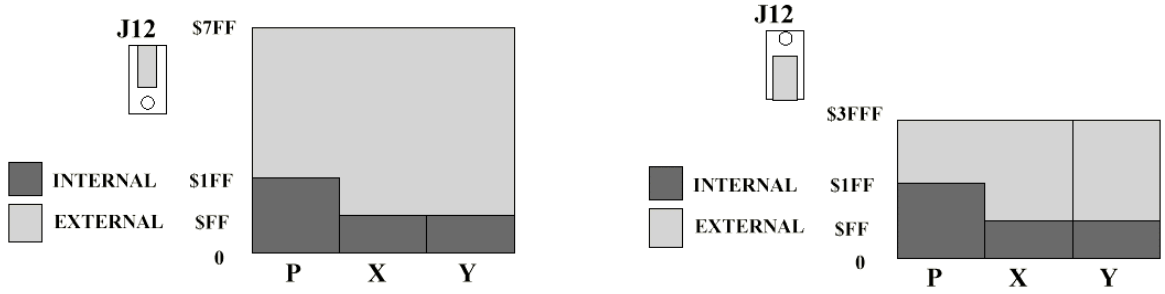
Steckerleiste J12

Die Steckerleiste dient zum Konfigurieren der Zuordnung von externem RAM und den drei Adreßräume des DSP.

In der Stellung 32K werden die gesamten 32kWorte des RAMs allen drei Adreßräumen zugeordnet. Der gesamte Speicherbereich des RAMs kann somit von den drei Adreßräumen aus adressiert werden. Für eine Trennung der Speicherbereiche ist der Programmierer verantwortlich. Die Adresse **`p:$0`** verweist z.B. auf die gleiche Speicherzelle wie die Adresse **`x:$0`** und **`y:$0`**.

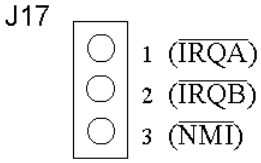
In der Stellung 16K werden die 32kWorte des RAMs in zwei gleiche Bereiche zu je 16k Worten aufgeteilt. Der untere Adreßbereiche ($0..$3FFF) des RAMs wird den

Adreßräumen **p:** und **x:** gemeinsam zugeordnet. Der obere Adreßbereich ($4000..$7FFF) wird dem Adreßraum **y:** zugeordnet. Die Adressen **p:$0** und **x:$0** verweisen z.B. beide auf die Speicheradresse $0 des RAMs. Im gegensatz dazu verweist die Adresse **y:$0** auf die Speicheradresse $4000 des RAMs.



Steckerleiste J17

Die drei Interrupteingänge des DSP sind an die Steckerleiste J17 geführt.



4.4 Spannungsversorgung

Auf dem Entwicklungsboard EVM56k befindet sich ein Gleichrichter sowie ein 5V Festspannunngsregler. Das EVM56k kann daher mit einer Wechselspannungsquelle von 7V bis 9V betrieben werden oder mit einer Gleichspannungsquelle von 7V bis 12V.

Die Spannungsversorgung kann entweder über die Buchs P1 oder die Klemmleiste J1 zugeführt werden.

5 Die Entwicklungsumgebung

Um eine Applikation für einen bestimmten Prozessor entwickeln zu können, wird eine leistungsfähige Entwicklungsunterstützung benötigt. Vom Hersteller wird deshalb die notwendige Firmware geliefert, mit der sich die Programmentwicklung wesentlich vereinfacht.

Das vollständige Entwicklungssystem besteht aus dem Assembler, dem Linker, der Bibliotheksverwaltung, dem Simulator und einer Prozessorkarte mit Debugger.

Weiterhin wird eine große Sammlung an Modulen und Routinen für die Signalverarbeitung mitgeliefert, aus denen sich Programme zusammenstellen lassen und mit deren Hilfe bestimmte Programmiertechniken der DSV gut zu verstehen sind.

Für zeitunkritische Anwendung ist ein **C-Compiler** vorhanden.

5.1 Ablauf bei der Programmentwicklung für den DSP

Jeder Anwender des Entwicklungssystems sollte ein eigenes Verzeichnis anlegen, in dem er seine Quellfiles bearbeitet. Verzeichnisübergreifend sollten nur gemeinsame Bibliotheken verwendet werden.

Alle mit dem DSP56002 in Zusammenhang stehenden Programme oder **`BATCH`**-Files beginnen mit den Ziffern 56. Alle **`BATCH`**-Files sind im **`s:\dsp_mc\dsp\bat`**-Directory zu finden. Dieses Verzeichnis steht -über den **`PATH`** Befehl- im Suchweg des PCs.

Handbücher mit ausführlichen Beschreibungen der verwendeten Programme liegen im Labor aus.

5.1.1 Assembler

Die Assembler Sprache des DSP erinnert immer wieder an den MC68000. Da der Prozessor einen großen Befehlssatz zur Verfügung stellt, sind Algorithmen der digitalen Signalverarbeitung verhältnismäßig leicht umzusetzen.

Die im Laborprojekt geschriebenen Programme (mit der Extension **`.ASM`**) werden mit dem Motorola Assembler in ein ladbares Objectfile mit der Extension **`.CLD`** übersetzt. Der Aufruf des Assemblers geschieht über ein Batchfile aus jedem beliebigen Unterverzeichnis (sinnvollerweise dem Arbeitsverzeichnis der Gruppe) folgendermaßen :

```
asm56k name <RETURN>
```

Der Assembler erzeugt ein lauffähiges Programm mit eingesetzten absoluten Adressen.

Zusätzlich wird ein List-File mit der Extension **`.LST`** im Directory des Quellfiles erzeugt. Es enthält kurz kommentierte Fehlermeldungen. List-Files können nach einer Sitzung am Rechner, bzw. nach der Programmentwicklung gelöscht werden.

Zwei besondere Anweisungen verdienen eine genauere Betrachtung. Die Anweisung
`REP n` wiederholt den nächsten Befehl n mal. Die Schleife

```
DO n
...
END
```

wiederholt ganze Routinen n mal. Der besondere Vorteil ist aber, daß die Überprüfung der Zähler (n) von spezieller Hardware auf dem Chip vorgenommen wird, so daß die Schleifen ohne zeitaufwendige Zählerabfragen ablaufen. Die **DO-**Schleifen können ineinander verschachtelt werden. Man hat damit ein sehr leistungsfähiges Werkzeug in der Hand, ohne, wie sonst üblich, Register für die Zähler "verschwenden" zu müssen.

Der Motorola Makro Assembler kann bedingt assemblieren und boolsche Ausdrücke direkt in bedingte Sprungkonstrukte umsetzen.

5.1.2 Linker

Im Labor wird der Linker nicht benutzt, da der Assembler bereits ausführbaren Code erzeugt. Der Linker gestattet es, vorübersetzte Module zu einem kompletten lauffähigen Gesamtprogramm zusammenzubinden.

Die Verwendung eines Linkers ist bei **einfachen** Programmen nicht erforderlich, was die Handhabung des Systems vereinfacht. Nur bei **größeren Projekten** ist die Verwendung des Linkers [2] sinnvoll. Einzelne Module werden getrennt übersetzt und dann mit Hilfe des Linkers verbunden. Ist ein Modul fehlerhaft, muß nur dieses korrigiert und übersetzt werden, wodurch die notwendigen Assemblierzeiten minimiert werden.

5.1.3 Simulator

Mit dem **Simulator** kann in Verbindung mit **Assembler** und **Linker** schon Software entwickelt und getestet werden, bevor der eigentliche DSP vorhanden ist. Der Simulator bildet das Verhalten des DSP56002 und dessen Hardwareumgebung auf einem PC per Software nach.

Mit einem Editor können ASCII Files erzeugt werden, die als "Testmuster" an einzelne Pins oder Pingruppen "angelegt" werden. Die Ergebnisse des (simulierten) DSPs können in Files mit Zeitmarken abgelegt werden und mit einem Texteditor bearbeitet werden. Man kann einem Port im Adreßraum des DSP ein Daten-File zuordnen, in dem z.B. die Abtastwerte eines Signals stehen. Damit kann simuliert werden, wie sich der DSP in der Realität verhalten würde und wie das Signal bearbeitet wird. Anhand der zu jedem Ergebnis errechneten Zeitmarken kann auf die spätere Ausführungsgeschwindigkeit geschlossen werden.

Im Simulatorpaket ist auch ein zeilenorientierter Assembler, ein Disassembler und eine Trace- bzw. Single-Step Funktion integriert. Der Simulator gestattet auch das Rechnen und Umrechnung von Zahlen in allen Zahlenformaten.

Mit dem Simulator können auch Multi-DSP-Projekte (bis zu 10 DSPs) simuliert werden. Hier ist Warten (auf die Simulationsergebnisse) sicherlich besser als Aufbauen und Probieren.

Soll das (fehlerfrei) übersetzte Programm in den Simulator geladen werden, so wird der folgende Aufruf verwendet :

```
sim56k <RETURN>
```

Im Simulator kann dann das Programm mit dem Befehl

```
load name <RETURN>
```

geladen werden.

---

[2] Ab Seite 37 beschrieben.

5.1.4 Debugger

Mit dem Debugger kann ein Programm auf dem DSP56002 des EVM-Moduls ausgeführt werden. Der Debugger wird durch Anklicken des Debugger Symbols in der Programmgruppe DSP gestartet.

Neben der Möglichkeit ein Programm auszuführen ermöglicht der Debugger auch eine Analyse des Prozessorzustandes und ein gezieltes Verändern diese Zustandes. Es lassen sich sämtliche Register des DSP56002 sowie die Speicherbereiche darstellen und verändern. Auch das gezielte Analysieren des Programmablaufes mit Hilfe des Einzelschrittmodus sowie durch Breakpoints wird unterstützt. Besonders erwähnenswert ist die Fähigkeit des Debuggers, einen Speicherbereich graphisch darzustellen. Eine qualitative Bewertung eines Verarbeitungsergebnisses wird damit wesentlich erleichtert.

5.2 Verzeichnis-Struktur

Alle zur Programmierung des DSPs notwendigen Files befinden sich in einem gut strukturierten Verzeichnisbaum.

Auf dem Netzlaufwerk **S:** befinden sich die Entwicklungssoftware für den DSP56002 und weitere Programme wie Matlab und Winword. Dieses Laufwerk kann nur gelesen werden.

Für jede DSP-Laborgruppe gibt es auf dem Netzwerklaufwerk **U:** unter dem Pfad **\DSP** ein Unterverzeichnis, in dem die Programme der Gruppen abgelegt werden können.

```
S:\DSP_MC                       ; Alles was zum Labor gehört
  DSP                           ; Alles was zum DSP gehört
     56K                        ; 56002
        DSPCBUG                 ; Debugger
        ASM                     ; Assembler
        CC                      ; C-Compiler
        MACROS                  ; Makrobibliotheken
             EVM                ; Initialisierungen und Konstanten
             MOTOROLA           ; Dr.BuB Public-Domain Prog
                  FLOAT         ; Floating-Point Arithmetik
                  FNTNS         ; Funktionen wie SQRT, Noise, etc.
                  MATRIX        ; Matrizenoperationen
                  SIOEQ         ; wichtige EQU Anweisungen
                  SORT          ; Sortieralgorithmen
                  DTMF          ; ein Demo DTMF-Dekoder
                  ...           ; ... und diverse andere
             IDEAL56            ; Alle Makros des IDEAL56 Systems.
                  MANAGE        ; Verwaltung und Initialisierung
                  DCT           ; Diskrete Cosinus-Transformation
                  CONV          ; Schnelle Faltung und Korrelation
                  FFT           ; FFT und iFFT
                  FILTER        ; FIR und IIR Filter
                  IO            ; div I/O-Treiber
                  WINDOW        ; Fensterfunktionen im Zeitbereich
        SIM                     ; Der 56001 Software Simulator
        LIBLNK                  ; Linker und Verwaltung (Motorola)

U:\                             ; Arbeitsverzeichnis
  DSP                           ; Alles was zum DSP gehört
     MoVo                       ; Arbeitsverzeichnis von Montag Vormittag
     DiVo                       ; Arbeitsverzeichnis von Dienstag Vormittag
     DiNa                       ; Arbeitsverzeichnis von Dienstag Nachmittag
```

```
FrVo                    ; Arbeitsverzeichnis von Freitag Vormittag
```

Die Quellfiles der Makros in den Bibliotheken sind vollständig dokumentiert, so daß bei Fragen zum Aufruf oder zur Funktion eines Makros nur das entsprechende File mit :

```
TYPE ...\PFADname\NAME.ASM
```

einzusehen ist.

5.3 Bibliotheken

Um die Programmierung des DSP zu erleichtern, ist die Verwendung fertiger Routinen möglich. Es stehen hierzu zwei Bibliotheken zur Verfügung.

a) Die Programmsammlung von **Motorola** ist eine Sammlung von Routinen signalverarbeitender Algorithmen. Diese Programme sind Teil einer Shareware-Sammlung, die Motorola über eine Mailbox ständig erweitert. Die Programme sind selbstdokumentierend (englisch) und nicht modular. In dieser Programmsammlung finden sich aber gute Anregungen. Siehe Kapitel Verzeichnis Struktur.

b) Die Makrobibliothek des **IDEAL56** Systems [3] ist vollständig modular und deckt viele Standardalgorithmen der DSV ab. Diese Bibliothek wurde im wesentlichen für den Einsatz in einem automatischen Codegenerator erstellt. Die enthaltenen Makros können jedoch auch einzeln verwendet werden. Diese Bibliothek enthält bereits die folgenden Routinen :

- FFT (Fast Fouriertransformation)
- IFFT (inverse FFT)
- DCT (Diskrete Cosinustransformation)
- IDCT
- Fensterfunktionen, darunter :
  - Blackman
  - Dreieck
  - Hamming
  - Hanning
  - Kaiser, mit einstellbarem α in der verwendeten Besselfunktion
  - Tapered ($\sin^2$), mit einstellbarem Übergangsbereich
  - Anwenderdefiniert
- Verzögerungsleitung
- AKF (direkte Autokorrelation im Zeitbereich)
- KKF (direkte Kreuzkorrelation im Zeitbereich)
  - für zwei zeitbegrenzte Eingangsfolgen
  - für eine zeitbegrenzte Eingangsfolge und eine zeitlich unbegrenzte Eingangsfolge (dynamisch)

[3] **I**ntuitive **D**esign **E**nvironment as an **a**lternative to established **l**anguages

  - direkt gerechnet oder
- Schnelle Faltung (über die FFT)
- Schnelle Kreuzkorrelation (FF, über die FFT)
- FIR Filter
- IIR Filter (nur Biquad Struktur)

Die Bibliothek enthält neben den o.a. Makros auch Funktionen zum Programmfluß-management und zur Bedienung einiger Schnittstellen. Hervorzuheben sind hier :

- settim (schnelle Programmierung der Interrupttimer auf dem Board)
- setintsw (Umschalten der Interruptquellen)
- setwait (schnelle Programmierung der Waitstates des DSPs)
- setstack (Einrichtung eines externen Stacks mit r7)
- push (Datum auf den Stack retten)
- pop (Datum vom Stack holen)

- div. Blockverwaltungsmakros
- etc.

Alle Makros der IDEAL56 Bibliothek müssen klein geschrieben werden, da der Assembler ´case-sensitive´ ist. Ein Makro wird zur sogenannten "Übersetzungszeit" (der Assembly-time) durch den Assembler expandiert und in das Programm eingefügt. Im Prinzip ist dies also nur eine textuelle Ersetzung. Die Makros werden durch Übergabe von Parametern konfiguriert.

6 Tips zur Programmierung in Assembler

Die folgenden Hinweise zur Programmierung in DSP5600x-Assembler ergänzen das DSP User's Manual, sollen zum weiteren Nachschlagen anregen und stellen die häufigsten Fehlerquellen dar.

- Zeichen in der **ersten** Spalte werden als Marke (engl. LABEL) interpretiert, sie müssen mit einem Buchstaben beginnen und dürfen kein reserviertes Schlüsselwort enthalten.
- Ein Label (oder auch Symbol) kann eine Adresse, eine Assemblierzeitvariable oder eine Konstante sein.
- Labels die mit _ beginnen, sind nur lokal verfügbar (wichtig beim Programmieren eigener Makros).
- Der Assembler ist Case-Sensitive, die Labels 'Hallo' und 'hallo' sind also verschieden. Mnemonics, Register und Schlüsselworte können aber beliebig geschrieben werden.
- Befehle dürfen ab der **zweiten** Spalte beginnen.
- Kommentar beginnt mit einem Semikolon (;). Kommentar nach einem doppelten Semikolon (;;) wird im **`.LST`**-File nicht gedruckt.
- Im Kommentar dürfen keine deutschen Umlaute verwendet werden, da der Assembler sie trotz der Kommentarzeichen falsch interpretiert.
- Sinnvollerweise verwendet man die TAB-Taste für 8-spaltiges Einrücken.
- Makronamen dürfen nur 8 Zeichen lang sein, wenn sie über die **MACLIB** Anweisung referenziert werden sollen.
- In das Programm sollte mit **include** das File **`LIBPATH.ASM`** eingebunden werden, es enthält Assembler-Anweisungen mit Zeigern auf die Bibliotheken, sowie wichtige Includes, z.B. I/O-Equates
- Der **`ORG`** Anweisung darf kein Label zugeordnet werden.
- Die **kleinste** Startadresse ist mit org P:$0090 zu wählen, da darunter die Interrupt-Vektoren und der Debugger liegen.
- Der Zeile nach der org Anweisung sollte ein Label z.B. 'anfang' zugeordnet werden und das Programm mit ' end anfang' (siehe Beispiel) beendet werden. Dies hat den Vorteil, daß die Startadresse im **`.CLD`**-File eingetragen wird und vom Debugger interpretiert werden kann.
- In der **`ORG`** Anweisung **muß** ein Speicherbereich (X, Y oder P) angegeben sein.
- Am Anfang des auszuführenden Codes sollte der interne Stackpointer (sp) mit Null geladen werden, damit bei mehrmaliger Benutzung des Debuggers (und Programmabbruch mit Neustart) der Stackpointer nicht überlaufen kann !
- Fast Interrupts werden nicht mit **`RTI`** abgeschlossen.
- Nicht initialisierte Interrupts (in P:0 - P:$40) sollten zwei NOPs enthalten. Bei Verwendung des Debuggers und nach einem Hardware-Reset werden automatisch NOPs eingesetzt.

- Die Anweisung **`EQU`** erzeugt Konstanten, die Anweisung **`SET`** erzeugt Variablen. Beide sind nur zur Assemblier-Zeit verfügbar, **nicht** jedoch zur Laufzeit !!!!!
- Die logischen oder arithmetischen Ausdrücke in Rechenanweisungen (z.B. @-Funktionen) für den Assembler mit sogen. Pseudo-Opcodes dürfen keine Blanks enthalten.
- Wenn während der Assemblierzeit mit Real-Konstanten gerechnet werden soll, so müssen sie durch einen Dezimalpunkt als Real-Zahlen gekennzeichnet werden. Bsp.: DC @cvf(**1.0/3.0**) erzeugt den Wert 1/3.
- Immediate Moves zu Speicherzellen (z.B. **`move #0,x:0`**) sind nicht zulässig (Gilt nicht für interne Peripherie).
- Memory-Memory moves (z.B. **`move x:0,y:0`**) sind nicht zulässig. Memory moves von oder zu **interner** Peripherie (z.B. Schnittstellenregister) sind jedoch zulässig.
- Quell- und Zielbezeichner einer move-Operation dürfen kein Leerzeichen enthalten
- Eine Kopie (move) einer 24 Bit Zahl in den Akku, erweitert A2 (oder B2) vorzeichenrichtig [4], so daß in A1 eventuell der falsche Wert steht. Diese Tatsache führt erfahrungsgemäß immer wieder zu Fehlern im Programmfluß.
- Bit Set, Test und Jump Anweisungen können - entgegen den Angaben im Handbuch - nicht auf Register angewendet werden. (erst ab Chip Maskenversion Rev. C, im Labor werden aber DSPs mit der B-Maske verwendet)
- Verschachtelte Do-Loops **müssen** unterschiedliche "End"-Adressen haben (**`NOP`** einfügen).
- **`REP`** Anweisungen sind nicht unterbrechbar, deshalb sollten in Realzeitanwendungen besser Do-Loops verwendet werden. Dies gilt auch für die MAC-Wiederholung im FIR Makro, wenn rekursive Interrupts verwendet werden.
- Vorsicht bei der Verwendung von **`RTS`** und **`ENDDO`** in einer **`Do`**-Loops.
- Daten, Konstanten und Koeffizienten **sollten** nur in X: und Y: abgelegt werden, da sonst zusätzliche P-Zugriffe den Instruction Prefetch verzögern.
- Das Register **R7** wird im Anwenderprogramm (analog zum A7 im 680xx) als Stackpointer verwendet, nach dieser Konvention ist R7 nicht anderweitig verfügbar !
- Wenn ein Breakpoint auf einen Interrupt gesetzt wird, so kann das Auftreten dieses Interrupts vom Debugger (im Moment) nur **einmal** erkannt werden !

6.1 Hinweise für die SSI / SCI-Programmierung

Die Programmierung der internen Peripherie des DSP56002 erfordert eine besondere Sorgfalt und ist infolge der Programmierung auf Bit-Ebene sehr fehleranfällig. Im folgenden werden einige Hinweise zur Port-C Programmierung gegeben.

Besser ist es, den aktuellen Wert des Registers zu lesen, zu maskieren und zurückzuschreiben. Diese Vorgehensweise erlaubt die getrennte Programmierung in einem Register zusammengefaßter Funktionen (Beispiel IPR) durch verschiedene Programmteile (und verschiedene Programmierer).

---

4 engl. sign extension

Es existieren bereits eine Reihe von Makros zur Programmierung der seriellen Schnittstellen, die genau diese Maskierung einzelner Bits leisten. Die Makros befinden sich in dem Verzeichnis **`L:\dsp\56k\macros\ideal56\manage`**. Es ist auf jeden Fall anzuraten diese Makros zu verwenden, da hierdurch bereits eine Reihe von möglichen Fehlern ausgeschlossen werden.

Für die folgenden Register existieren Makros zum modifizieren:

`PCC, CRA, CRB, IMR, IPR, SCR`

Zur Programmierung der SSI oder SCI bietet sich die folgende Reihenfolge an :

a) PCC zurücksetzen (z.B. mit **`move #0,x:m_pcc`**), wird dies unterlassen, so ist das Verhalten des Prozessors nur nach einem Reset determiniert.

b) CRA laden (SSI)

c) CRB laden (SCI)

d) PCDDR laden Richtung der Portpins einstellen.

e) IPR laden Hiermit wird die Priorität aller Interrupts festgelegt.

f) IMR laden Hiermit wird der Interruptlevel eingestellt und die Interrupts freigegeben.

g) PCC setzen Serielle Portfunktion freigeben.

Für den im Labor verwendeten DSP56002 gelten folgende Einschränkungen:

- **Für die SCI gilt**: In der verwendeten Maskenversion B des DSP56002 ist das Bit 3 des SCR-Registers nur mit **Null für LSB-first** programmierbar. Erst die später freigegebene Maskenversion C des DSP erlaubt das Schieben der SCI Daten in beide Richtungen. Dieser Punkt ist nur in neueren User's Manuals (Rev 2) erläutert.
- **Für die SSI gilt**: In der verwendeten Maskenversion B des DSP56002 ist das Bit 6 (SHFD) des CRB-Registers nur mit **Null für MSB-first** programmierbar. Erst die später freigegebene Maskenversion C des DSP erlaubt das Schieben der SSI Daten in beide Richtungen.

6.2 Digitale Filter

In diesem Kapitel soll auf die Problematik der digitalen Filterung eingegangen werden. Es werden verschiedene Filterstrukturen vorgestellt und deren Realisierung mit dem DSP gezeigt. Dabei wird deutlich, warum gerade der DSP56002 für diese Aufgabe so gut geeignet ist.

Bei den digitalen Filtern gibt es zwei Grundtypen: Filter mit nichtrekursiver Struktur und Filter mit rekursiver Struktur. Die beiden Filtertypen nennt man auch FIR bzw. IIR-Filter.

6.2.1 FIR-Filter

Bei einem FIR-Filter werden zur Berechnung des Ausgangssignals nur Eingangswerte und entsprechenden Filterkoeffizienten benötigt. Bei einem IIR-Filter hingegen werden auch die vorherigen Ausgangswerte zur Berechnung benötigt.

In Bild 3 ist ein FIR-Filter dritter Ordnung abgebildet.

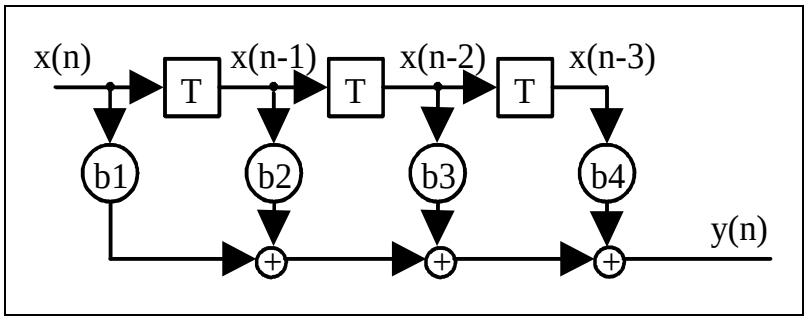


Bild 6.1: FIR-Filterstruktur 3. Ordnung

Aus der Filterstruktur läßt sich die mathematische Formel der Differenzengleichung ablesen:

$$y(n) = \sum_{k=1}^{N} b_k * x(n-k+1)$$

$$y(n) = b1 * x(n) + b2 * x(n-1) + b3 * x(n-2) + b4 * x(n-4)$$

In der Literatur findet man verschiedene Bezeichnungen für die Filterkoeffizienten b1 bis b4. Die hier gewählte Bezeichnung lehnt sich an das Programm MATLAB an, mit dessen Hilfe sich Filterkoeffizienten berechnen lassen.

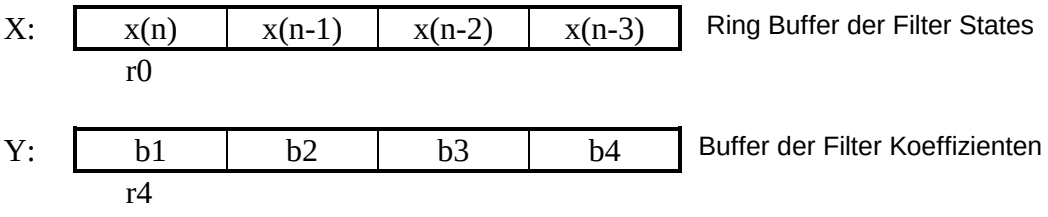
Für eine Filterroutine in der ein FIR-Filter realisiert werden soll benötigt man neben den Filterkoeffizienten auch noch die alten Eingangswerte. Beide, Filterkoeffizienten und Eingangswerte, müssen also im Speicher des DSP abgelegt sein. Da der DSP über zwei Datenspeicher verfügt (X- und Y-Speicher), ist es günstig, die Filterkoeffizienten in dem Y-Speicher und die Eingangswerte in dem X-Speicher abzulegen. Die Realisierung einer FIR-Filterroutine ist in dem folgenden Programmlisting zu sehen.

```
move        a1,x:(r0)                    ;;Eingangswert a1 in den Ringpuffer retten
clr    a    x:(r0)+,x0    y:(r4)+,y0     ;;loesche a, Eingangswert und Koeffizient
                                         ;; holen
rep    m0                                ;;repeat laenge -1
mac    x0,y0,a x:(r0)+,x0   y:(r4)+,y0   ;;a = a + x0 * y0, hole nächste Werte
macr   x0,y0,a                           ;;Letzten Wert berechnen
```

Damit diese Filterroutine korrekt arbeitet müssen einige Bedingungen vorher erfüllt sein.

Der Eingangswert muß im Akku a stehen. Die Zeiger r0 und r4 müssen auf die alten Eingangswerte bzw. auf die Filterkoeffizienten zeigen. Die Moduloregister m0 und m4 müssen entsprechend eingerichtet sein.

Im Überblick sieht der Speicher des DSP dann folgendermaßen aus:



Der berechnete Ausgangswert des Filters steht ebenfalls im Akku a.

Von den digitalen Filtern stellt der FIR-Filter den Einfachsten dar. Er hat allerdings einige Nachteile. Durch seine Filterstruktur benötigt man für eine hohe Steilheit einen Filter mit sehr hoher Ordnung. Dies bedeutet aber wiederum eine lang dauernde Berechnung der Ausgangswerte.

6.2.2 IIR-Filter I

Im Gegensatz zu einem FIR-Filter hat ein IIR-Filter schon bei sehr geringer Ordnung eine beachtliche Steilheit. Dies kann allerdings dazu führen, daß das Filter instabil wird. Im Einzelfall muß das Filter auf seine Stabilität hin untersucht werden.

Wie bereits oben schon erwähnt, nutzt ein IIR-Filter nicht nur die letzten Eingangswerte, sondern auch die vergangenen Ausgangswerte. Die Struktur eines solchen Filters stellt somit eine Erweiterung der oben gezeigten FIR-Filterstruktur dar. Im Bild 2 ist eine IIR-Filterstruktur abgebildet.

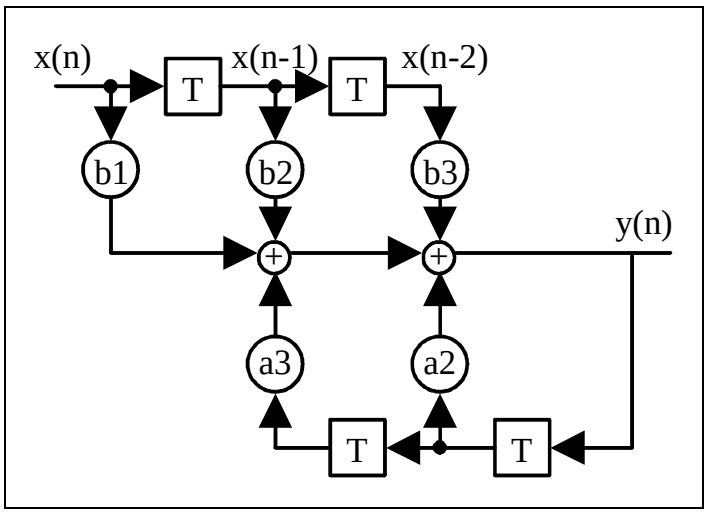


Bild 6.2: IIR-Filterstruktur zweiter Ordnung

Auch hier soll die mathematische Formel der Differenzengleichung angegeben werden:

$$y(n) = \sum_{k=1}^{N} b_k * x(n-k+1) - \sum_{i=2}^{N} a_i * y(n-i+1)$$

$$y(n) = b1 * x(n) + b2 * x(n-1) + b3 * x(n-2) - a2 * y(n-1) - a3 * y(n-2)$$

Die nötigen Filterkoeffizienten lassen sich wieder mit MATLAB berechnen.

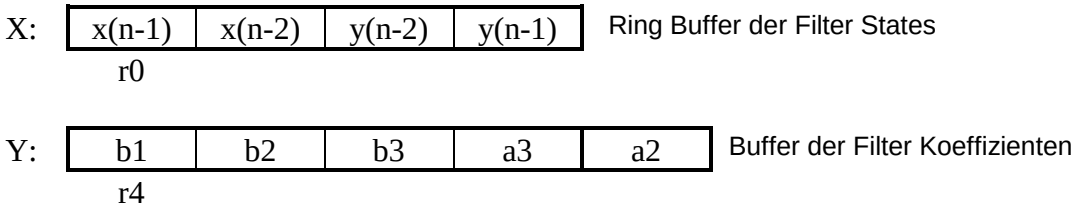
Bei der Realisierung des Filters mit dem DSP geht man ähnlich wie beim FIR-Filter vor. Auch hier müssen bestimmte Register und Speicherinhalte initialisiert werden.

Die Filter States werden im X-Speicher, die Koeffizienten wieder im Y-Speicher abgelegt. Adreßregister r0 und r4 sowie Moduloregister m0 und m4 müssen entsprechend eingerichtet werden. Der Eingangswert muß wieder im Akku a übergeben werden, der Ausgangswert steht ebenfalls im Akku a.

```
        do      n0,_EndeDo
        move    a,x1                            y:(r4)+,y0      ;x(n) -> x1, a1 -> y0
        mpy     x1,y0,a         x:(r0),x0       y:(r4)+,y0      ;x(n)*a1, x(n-1) -> x0, a2 -> y0
        mac     x0,y0,a         x1,x:(r0)+                      ;x(n-1)*a2, x1 -> X(n-1)
        move                    x:(r0),x1       y:(r4)+,y0      ;x(n-2) -> x1, a3 -> y0
        mac     x1,y0,a         x0,x:(r0)+                      ;x(n-2)*a3, x0 -> X(n-2)

        move                    x:(r0)+,x0      y:(r4)+,y0      ;Y(n-2) -> x0, b2 -> y0
        mac     -x0,y0,a        x:(r0)-,x0      y:(r4)+,y0      ;Y(n-2)*b2, Y(n-1) -> x0, b3 -> y0
        macr    -x0,y0,a        x0,x:(r0)+                      ;Y(n-1)*b3, x0 -> Y(n-2)
        asl     a
        rnd     a
        move                    a1,x:(r0)+                      ;A1 -> Y(n-1)
_EndeDo
```

Die Speicheraufteilung für den IIR-Filter unterscheidet sich kaum vom FIR-Filter



Um die Stabilität eines IIR-Filters zu gewährleisten, kann man einen IIR-Filter höherer Ordnung durch mehrere kaskadierte Filter mit niedriger Ordnung realisieren. Ein IIR-Filter vierter Ordnung läßt sich so mit zwei kaskadierten IIR-Filtern zweiter Ordnung realisieren. Soll ein Filter mit ungerader Ordnung realisiert werden, so muß nach der Filterkaskade ein Filter erste Ordnung folgen.

Mit der im Programmlisting gezeigten DO-LOOP werden n0 Teilsysteme zweiter Ordnung kaskadiert.

6.2.3 IIR-Filter II

Für die Realisierung eines IIR-Filter gibt es noch weitere Strukturen. Hier soll als Alternative zur Struktur I die sogenannte Canoesche Filterstruktur (auch Direktform I genannt) vorgestellt werden.

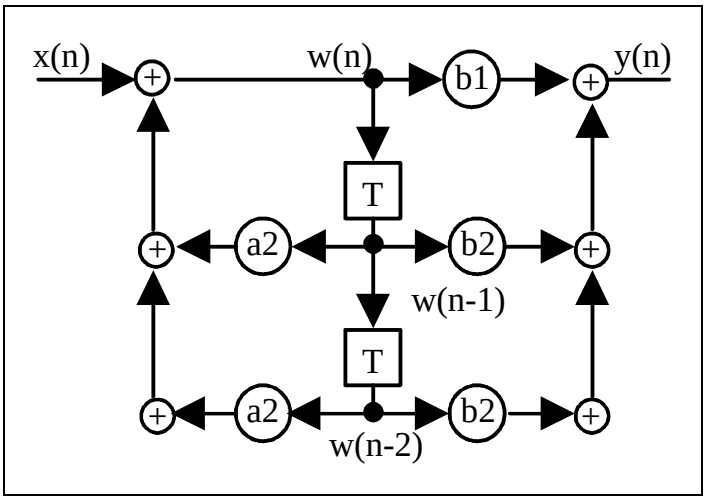


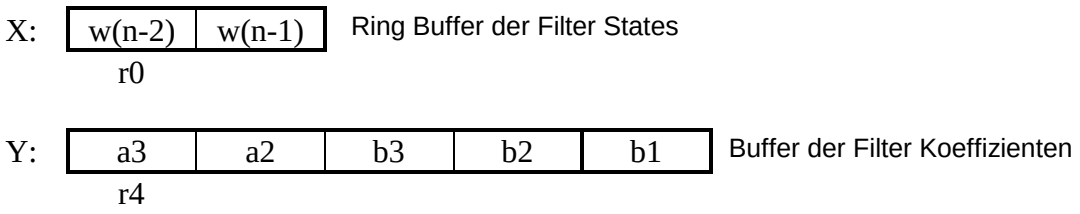
Bild 6.3: IIR-Filter II

Anhand des Strukturbildes ist zu erkennen, daß zur Berechnung nicht mehr Eingangs- und Ausgangswerte benötigt werden, sondern lediglich drei Zwischenergebnisse. Die Filterroutine wird dadurch schneller und ist einfacher zu programmieren.

```
        move                    x:(r0)+,x0      y:(r4)+,y0      ; load first state

        do      n0,_endiirloop                                  ; do each subsystem
        mac     -x0,y0,a        x:(r0)-,x1      y:(r4)+,y0      ; a := a + a(t)(3) * w(t)(n-2)
        macr    -x1,y0,a        x1,x:(r0)+      y:(r4)+,y0      ; a := a + a(t)(2) * w(t)(n-1)
        move    a1,y1                                           ; y1 := a
        mpy     x0,y0,a         a,x:(r0)+       y:(r4)+,y0      ; a :=     b(t)(3) * w(t)(n-2)
        mac     x1,y0,a                         y:(r4)+,y0      ; a := a + b(t)(2) * w(t)(n-1)
        mac     y0,y1,a         x:(r0)+,x0      y:(r4)+,y0      ; a := a + b(t)(1) * y1
_endiirloop
        rnd     a                                               ; round result
```

Wie schon bei den vorangegangenen Filterroutinen müssen auch hier der Speicher sowie die Adress- und Moduloregister initialisiert werden. Nachfolgend ist eine Übersicht der Speicherbelegung zu sehen.



Der Eingangswert muß sich im Akku a befinden, der Ausgangswert wird ebenfalls im Akku a zurückgegeben. Das realisierte Filter läßt sich über die DO-LOOP wieder kaskadieren, wobei im n0 die Anzahl der Teilsysteme zweiter Ordnung übergeben werden müssen.

6.2.4 IIR-Filter II

Die Filterstruktur aus dem vorherigen Kapitel läßt sich noch weiter vereinfachen. Die Koeffizienten bx lassen sich umrechnen, so daß der Koeffizient b1 zu 1.0 wird. Es kann dadurch die Rechenoperation mit b1 entfallen.

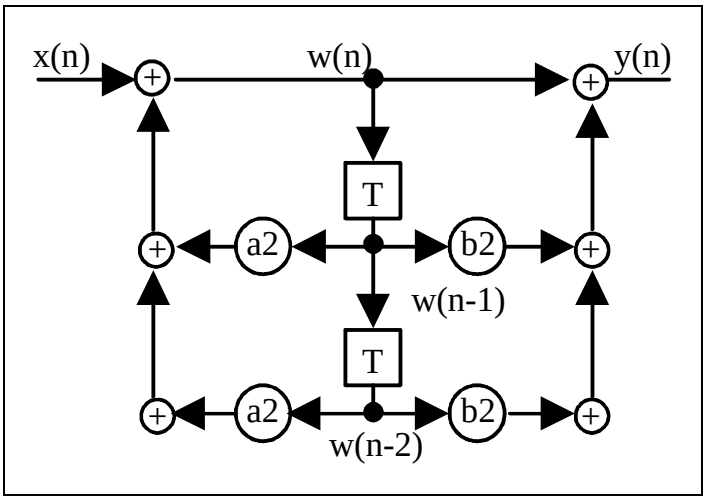
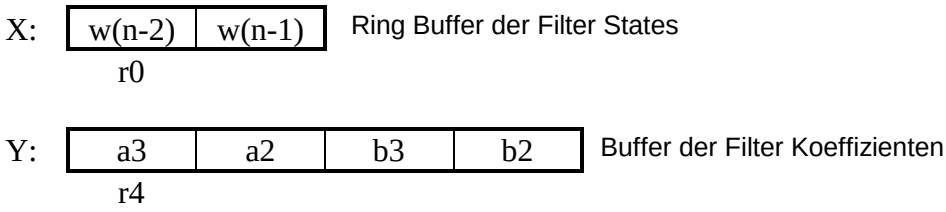


Bild 6.4: IIR-Filter III

Es ergibt sich somit eine leicht veränderte Speicheraufteilung:



Durch die weitere Vereinfachung der Filterstruktur wird das Programm ebenfalls kürzer.

```
        ori     #$08,mr                                 ;set scaling mode
        move                x:(r0)+,x0  y:(r4)+,y0      ;first state, a12
        do      n0,_ends                                ;do each section
        mac     -x0,y0,a    x:(r0)-,x1  y:(r4)+,y0      ;ax2
        macr    -x1,y0,a    x1,x:(r0)+  y:(r4)+,y0      ;ax1
        mac     x0,y0,a     a,x:(r0)+   y:(r4)+,y0      ;bx2
        mac     x1,y0,a     x:(r0)+,x0  y:(r4)+,y0      ;bx1
_ends
        rnd     a                                       ;round result
        andi    #$f7,mr                                 ;remove scaling mode
```

6.3 Programmaufteilung in Module

Es ist empfehlenswert, größere Projekte in mehrere Module aufzuteilen, insbesondere wenn ein Projekt von mehreren Programmierern bearbeitet wird. Wünschenswert ist auch eine getrennte Übersetzung dieser Module und ein individueller Test einzelner Module unabhängig vom Gesamtsystem.

Die Entwicklungsumgebung des DSP56002 bietet zwei Möglichkeiten der Modularisierung:

- Die Verwendung von Makros ermöglicht es einzelne Programmteile zu entwickeln und zu testen. Anschließend können die Teilprogramme sehr einfach mit der **`include`**-Anweisung in das Hauptprogramm eingefügt werden. Im Hauptprogramm stehen dann lediglich die einzelnen Makro-Aufrufe, wodurch die Übersichtlichkeit der Software unterstützt wird.

- Mit Hilfe des **Linkers** können einzelne assemblierte Module zusammengefügt werden.

Vorteile :

- Ein Gesamtprojekt wird in einzelne Module zerlegt, deren **Schnittstellen** genau festgelegt werden müssen.
- Stehen mehrere Rechner zur Verfügung, so können mehrere Programmierer parallel arbeiten. Es ist ersichtlich, welcher Teilnehmer einen bestimmten Abschnitt erstellt hat.
- Die Module haben definierte Schnittstellen. Es wird vom Assembler überprüft, ob das Zusammenwirken der Module ausschließlich über diese Schnittstellen abläuft. Die Software ist dadurch wartungsfreundlich und leichter testbar.
- Gleiche Symbolnamen (sofern es sich um lokale Symbole handelt) dürfen in mehreren Modulen verwendet werden, ohne daß dies zu Problemen führt.
- Ein Modul ist kürzer und damit übersichtlicher als das Gesamtprogramm.

6.3.1 Sections

Die gekapselten Programmteile (Module) heißen im DSP-56000-Assembler "sections". Jede Section ist folgendermaßen aufgebaut:

```
SECTION<Name>
[ XREF <Symbol 1a>,<Symbol 2a>,.... ]
[ XDEF <Symbol 1b>,<Symbol 2b>,.... ]
<ORG-Anweisung>
<Programmbefehle>
ENDSEC
```

- Die Anweisung **XREF** teilt dem Assembler mit, daß die angegebenen Symbole außerhalb der "section" definiert sind und innerhalb der "section" benutzt werden sollen.
- Die Anweisung **XDEF** macht die angegebenen Symbole der "section" global verfügbar. Alle übrigen Symbole der "section" sind lokal, d. h. in anderen "sections" nicht verfügbar.

Die Aufteilung eines Programms in "sections" ist jedoch nicht zwingend. Symbole außerhalb von "sections" sind global und können mittels **XREF** innerhalb von sections benutzt werden.

Symbole sind z.B. Sprungadressen, Unterprogrammadressen, Makronamen, **SET**-Variablen und **EQU**-Konstanten. Bei **SET**-Variablen gibt es Besonderheiten (siehe Handbuch), ebenso wie bei Makros.

Makros, die innerhalb von "sections" definiert sind, können nicht über **XDEF** global verfügbar gemacht werden.

6.3.2 Speicherverwaltung

Bei der Verwendung mehrerer Module ist es wichtig, die Belegung des Arbeitsspeichers zu koordinieren. Insbesondere sind Adressen von Variablen und Programmteilen festzulegen.

Man könnte jedem Programmierer Speicheradressen fest zuteilen. Es ist einleuchtend, daß diese Vorgehensweise unpraktisch ist. Nutzt beispielsweise ein Programmierer den für ihn reservierten schnellen internen Speicher nicht aus, läuft das Programm unnötig langsam ab. Außerdem ist die Verwendung von absoluten Adressen fehleranfällig und änderungsfeindlich. Viel sinnvoller ist es, die Adressen vom Linker festlegen zu lassen.

6.3.3 ORG-Anweisung

Die Startadresse eines Programms kann im absoluten Assemblermodus (-A Option) absolut im Programm angegeben werden, Beispiel:

```
ORG P:$40          ; Programm beginnt bei P:$0040
```

Wird das Programm in verschiedene Module aufgeteilt und mit dem Linker zusammengebunden, so darf keine ORG P Anweisung gefolgt von einer absoluten Adresse im Programm verwendet werden ! (Ausnahme : Startadresse einer Exception)

Für jeden Speicherbereich (X, Y und P) richtet der Linker drei Adreßzähler ein:

- einen Standardzähler
- einen L-Zähler und
- einen H-Zähler.

Die Zähler werden erst bei Aufruf des Linkers initialisiert. Mit Kommandozeilenparametern (siehe Handbuch des Linkers) werden die verschiedenen Zähler auf einen Startwert gesetzt. Der Linker erhöht die verwendeten Zähler dem Speicherbedarf entsprechend, so daß die Module automatisch lückenlos zusammengeführt werden. (Wichtig: P muß bei $40 beginnen, da von P:$0000 bis P:$0040 die Interruptvektoren abgelegt sind)

Beispiele:

```
ORG X          ;  Standardzähler für X-Speicher
```

oder

```
ORG PL         ;  L-Zähler für P-Speicher
```

6.3.4 Benutzung der Adreßzähler

Die Adreßzähler könnten z.B. dazu benutzt werden, den internen und den externen Speicher getrennt zu verwalten. Die L-Zähler könnten beispielsweise mit den Startadressen des internen Speichers und der H-Zähler mit den Startadressen des externen Speichers initialisiert werden.

```
ORG XL              ; Variablen im internen Speicher
ORG XH              ; Variablen im externen Speicher
```

6.3.5 Projektgerüst

Das folgende Programmgerüst zeigt schematisch, wie ein Projekt mehrere Programmierer aussehen könnte.

```
        SECTION         hauptp          ; von Programmierer 1
        XREF            inithandler
        XDEF            progstart       ; Die Startadresse hier angeben !
waitstates      equ 1                   ; eine Konstante
        INCLUDE ´...........´           ; Bibliotheken einbinden
        ORG P                           ; jetzt kommt Programmcode
progstart                               ; Startlabel
        setwait waitstates              ; Wartezyklen einstellen
        romoff                          ; ROM-Tabellen ausblenden
        jsr     inithandler             ; Initialisierungsroutine aufrufen
        ......                          ; Interrupt anschalten
        jmp     *                       ; Sprung auf sich selbst,
                                        ; ab jetzt nur noch Interrupts.
        ENDSEC
```

```
        SECTION         ivector         ; von Programmierer 2
        XREF            handler
ivectoraddr     equ ...         ; Adresse in der Vektorentabelle
        ORG     P:ivectoraddr           ; absoluter Adreßbezug
        jsr     handler                 ; Interrupthandler aufrufen
        nop
        ENDSEC
```

```
        SECTION         ihandler        ; von Programmierer 3
        XDEF            handler,inithandler
        ORG X
        .....                           ; Variablen im X-Bereich
        ORG Y
        .....                           ; Variablen im Y-Bereich
        ORG P                           ; jetzt kommt Programmcode
inithandler:
        .....                           ; initialisiere
        rts
handler:
        .....                           ; tu was
        rti
        ENDSEC
```

7 Beispielprogramme

7.1 Gleichrichten einer Folge von Abtastwerten

Das abgedruckte Beispielprogramm zeigt, wie ein Programm grundsätzlich für den DSP56002 aussehen kann. Es wird zunächst eine Sinustabelle erzeugt. Dieser Sinus wird anschließend durch den DSP 'gleichgerichtet'.

Erläuterung zum Programm:

Kommentare werden in einem Assemblerprogramm mit einem Semikolon eingeleitet. Der Assembler ignoriert alle Zeichen, die nach einem Semikolon folgen. Kommentare nach einfachen Semikola werden jedoch unverändert in das **.LST**-File übernommen. Mit doppelten Semikola werden Kommentare eingeleitet, die nicht in das **.LST**-File übernommen werden sollen.

Die ersten fünf Zeilen nach den Kommentarzeilen sind Variablendeklarationen für den Assembler. Der Befehl **SET** ist ein Assembler Pseudo-Opcode. Die Pseudo-Opcodes werden vom Assembler zur Assemblierzeit bearbeitet. Sie haben nichts mit dem Befehlssatz des DSP56002 zu tun. Durch '**pstar SET $90**' wird zum Beispiel die Startadresse des Programms auf $90 festgelegt (P-Speicher ab der Adresse $90).

Die Kommentarzeilen nach der Variablendeklaration beginnen nur mit einem Semikolon (;), sie werden somit auch im **.LST**-File abgedruckt (s.o.).

Bei den folgenden acht Befehlszeilen handelt es sich ebenfalls wieder um Pseudo-OpCodes. An dieser Stelle wird die Sinustabelle erzeugt.

- Der **ORG**-Befehl ist für die Speicheraufteilung zuständig. Er sorgt dafür, daß die Sinustabelle im X-Speicher ab der Adresse '**sinstart**' ($0) abgelegt wird.
- Durch den Befehl **COBJ** wird in dem Object-File (**.LOD**) ein Kommentar eingefügt.
- Mit dem **MSG**-Befehl wird während des assemblierens ein Kommentar auf dem Bildschirm ausgegeben.
- Die **DUP**-Anweisung wiederholt die nächsten Zeilen (bis **ENDM**) '**points**' mal (512 mal). In dieser Schleife werden die einzelnen Sinuswerte berechnet und mit dem **DC**-Befehl im Speicher abgelegt.

Nach weiteren Kommentarzeilen beginnt das eigentliche Assemblerprogramm für den DSP56002.

Auf die einzelnen Befehle soll hier nicht näher eingegangen werden, da sie ausführlich im DSP-Manual erklärt werden.

```
;;
;;      Beispielprogramm fuer das DSP-Labor
;;
;;      Es wird durch den Assembler eine Sinustabelle erzeugt und im
;;      X-Speicher abgelegt. Diese Sinustabelle wird dann durch den
;;      DSP 'gleichgerichtet'.

pstart  SET     $90
sinstartSET     $200
points  SET     512
periods SET     4
pi      SET     3.141592654
```

```
freq    SET     2.0*pi/@CVF(points)*periods

;       Erzeugung einer Sinustabelle durch den Assembler
;       Diese Befehle werden von dem Assembler zur assemblierzeit
;       bearbeitet.

        ORG     X:sinstart
        COBJ    'Start der SIN Tabelle'
        MSG     'Berechnung der SINUS-Tabelle'
count   SET     0
        DUP     points
        DC      @SIN(@CVF(count)*freq)
count   SET     count+1
        ENDM                            ;; End dup

;       Hier beginnt das eigentliche Assemblerprogramm fuer den DSP

        ORG     P:pstart
anfang  MOVE    #sinstart,r1            ;Vorbereiten der Register
        MOVE    #0,x1
        MOVE    #>(points+sinstart),b

schleifeDO      #points,endloop ;Schleife über die Anzahl der Samples
                        ;
        MOVE    x:(r1),a        ;Lade eine Wert aus der Sinustabelle
                                ;in den Akku a.
        CMP     x1,a                    ;Vergleiche a mit x1 (a-x1)
        JPL     storeback               ;Ist a positiv, dann springe
                                        ;nach 'storback'.
        NEG     a                       ;negiere a
storeback       MOVE    a,y:(r1)+       ;Speichere a nach y
endloop                                 ;Ende der Schleife

ende    jmp     ende                    ;Endlosschleife
        END
```

7.2 Daten über die SSI-Schnittstelle einlesen

Das folgende Programm verwendet den Stereo Codec CS4215 auf dem Entwicklungsboard EVM. Ein analoges stereo Eingangssignal wird vom Codec digitalisiert und über die SSI-Schnittstelle zum DSP übertragen. Der DSP speichert die digitalisierten Abtastwerte in einem Ringpuffer. Anschließend werden die verzögerten Abtastwerte über die SSI-Schnittstelle zurück zum Codec übertragen und von diesem ausgegeben.

Das Programm ist in zwei Module aufgeteilt. Das Modul **`echo6.asm`** enthält das Hauptprogramm, das Modul **`ssiisr6.asm`** die Interruptverarbeitung.

Das Programm hat folgenden Programmflußplan:

Modul echo6.asm

```
        ident   1,1
        page    132,60

;*****************************************************************************
; Programm:     echo5.asm
; Zweck:        Das Programm liesst Daten vom AD/DA-Umsetzer CS4215
;               über die SSI-Schnittstelle in den DSP.
;               Der rechte Kanal wird in einem Ringpuffer im X-Speicher und
;               der linke Kanal wird in einem Ringpuffer im Y-Speicher
;               abgelegt.
;               In dem Modul 'ssiisr6.asm' befinden sich die Programmteile
;               zum Initialsieren der SSI-Schnittstelle und Abarbeiten deren Interrupts.
; Autors:       L. Kahl
; Version:      10.10.95/15.10.96
;*****************************************************************************

        maclib  's:\dsp_mc\dsp\56k\macros\evm'  ; Pfad zu den Makro- und Include-Dateien

        initevm main                    ; Das Makro initialisiert das EVM Modul und
                                        ; sollte im Hauptprogramm immer vor allen
                                        ; Befehlen stehen. Reset Vektor und Stack-
                                        ; pointer werden gesetzt. Anschliessend wird
                                        ; zur Marke 'main' gesprungen.

                                        ; Weiter Programm Module:
        include 'ssiisr6.asm'           ; SSI Interrupt Service Routine
                              ; SSI Initialisierung

;****** Anfang des Programm Moduls echo ************************************
        section echo
        xref    init_SSI
        xref    rx_data_12,rx_data_34
        xref    tx_data_34,tx_data_56,tx_data_78,tx_data_12
        xdef    ProcessData
        xdef    main

        include 'evm.inc'               ; Konstanten des EVM
        include 'cs4215.inc'            ; Konstanten des CS4215

;****** Konstanten ********************************************************
CTRL_12         equ     PREAMP_OFF+OUTAMP_HIGH+HI_PASS_ON+SAMP_RATE_48+STEREO+DATA_16
CTRL_34         equ     BITS_64+CODEC_MASTER

TONE_OUTPUT     equ     HEADPHONE_EN+LINEOUT_EN+(4*LEFT_ATTN)+(4*RIGHT_ATTN)
TONE_INPUT      equ     MIC_IN_SELECT+(15*MONITOR_ATTN)

BUF_START       equ     $4000           ; Startadresse der Ringpuffer
BUF_LEN         equ     $4000           ; Laenge der Ringpuffer

;****** Variablen *********************************************************
        org     xh:BUF_START
lBuf    ds      BUF_LEN                 ; Ringpuffer fuer den linken Kanal

        org     yh:BUF_START
rBuf    ds      BUF_LEN                 ; Ringpuffer fuer den rechten Kanal

;****** Programm **********************************************************
```

```
        org     p:
main
        jsr     init_cs4215
        jsr     init_SSI

        move    #BUF_START,r4           ; Zeiger auf den Ringpuffer initialisieren
        move    #(BUF_LEN-1),m4         ; Laenge des Ringpuffers initialisieren

        move    #TONE_OUTPUT,y0         ; headphones, line out, mute spkr, no attn.
        move            y0,x:tx_data_56
        move    #TONE_INPUT,y0          ; no input gain, monitor mute
        move            y0,x:tx_data_78

loop_1  jmp     loop_1                  ; Warte auf Interrupts

;****** Unterprogramm ProcessData ******************************************
;       Kopiert die zwei neu eingelesenen Abtastwerte aus dem Empfangspuffer in
;       den Ringpuffer und schreibt die zwei ältesten Abtastwerte aus dem
;       Ringpuffer in den Sendepuffer.
;***************************************************************************
ProcessData
        move    x:(r4),a        ; get old left sample
        move    y:(r4),b        ; get old rigth sample
        move    a,x:tx_data_12          ; Put value in left channel tx.
        move    b,x:tx_data_34          ; Put value in right channel tx.
        move    x:rx_data_12,a          ; get new left sample
        move    x:rx_data_34,b          ; get new right sample
        move    a,x:(r4)                ; store new left sample
        move    b,y:(r4)+               ; store new rigth sample
        rts

init_cs4215
        cs4215I #CTRL_12,#CTRL_34       ; Makroaufruf zum Initialisieren des CS4215
        rts                             ; Die Kontrollworte werden dem Makro als
                                        ; Parameter übergeben.
        endsec
;****** Ende des Programm Moduls echo ************************************
        end
```

Modul ssiisr6.asm

Das folgende Programmodul enthält die Initialisierungsroutine für die SSI-Schnittstelle und die Interrupt Service Routine für die SSI-Schnittstelle.

```
        ident   1,0
        page    132

;****** Anfang des Programm Moduls ssiisr **********************************
        section ssiisr

        xdef    rx_data_12,rx_data_34,rx_data_56,rx_data_78
        xdef    tx_data_34,tx_data_56,tx_data_78,tx_data_12
        xdef    init_SSI
        xref    ProcessData

        maclib  's:\dsp_mc\dsp\56k\macros\evm'
        include 'evm.inc'

;****** Konstanten Definition *********************************************
;
; CRA Register
;   15   14   13   12   11   10    9    8    7    6    5    4    3    2    1    0
; +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
; | PSR|   WL    |           DC           |                 PM                  |
; +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
; |  0 |  1    0 |  0    0    0    1    1 |  0    0    0    0    0    0    1    1 |
; +------------------+------------------+------------------+--------------------+
;           4                  3                  0                  3
;
```

```
; CRB Register
;   15   14   13   12   11   10    9    8    7    6    5    4    3    2    1    0
; +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
; | RIE| TIE| RE | TE | MOD| GCK| SYN|FSL1|FSL0|SHDF|SCKD|SCD2|SCD1|SCD0| OF1| OF0|
; +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
; |  1 |  0 |  1 |  1 |  1 |  0 |  1 |  1 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
; +------------------+------------------+------------------+------------------+
;           B                  B                  0                  0
;
; PCC Register
;     8      7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+------+
; |  STD |  SRD |  SCK |  SC2 |  SC1 |  SC0 | SCLK |  TXD |  RXD |
; +------+------+------+------+------+------+------+------+------+
; |   1  |   1  |   1  |   1  |   0  |   1  |   0  |   0  |   0  |
; +------+-------------------------+-------------------------+
;     1               E                         8

D_CRA   equ     $4303
D_CRB   equ     $BB00
D_PCC   equ     $01E8

;****** Variablen Definition ***************************************************
        org     x:

RX_BUFF_BASE    equ     *
rx_data_12      ds      1               ;data time slot 1/2 for RX ISR
rx_data_34      ds      1               ;data time slot 3/4 for RX ISR
rx_data_56      ds      1               ;data time slot 5/6 for RX ISR
rx_data_78      ds      1               ;data time slot 7/8 for RX ISR

TX_BUFF_BASE    equ     *
tx_data_56      ds      1               ;data time slot 5/6 for TX ISR
tx_data_78      ds      1               ;data time slot 7/8 for TX ISR
tx_data_12      ds      1               ;data time slot 1/2 for TX ISR
tx_data_34      ds      1               ;data time slot 3/4 for TX ISR


;****** Interrupt Vektoren Initialisieren **********************************
        org     PL:$000C
        jsr     ssi_rx_isr              ;SSI RX
        jsr     ssi_rx_isr              ;SSI RX w/Exception
        jsr     ssi_tx_isr              ;SSI TX
        jsr     ssi_tx_isr              ;SSI TX w/Exception

;****** Programm ************************************************************
        org     p:
        jmp     end_ssiisr              ; Sektion überspringen

;****** Unterprogramm init_SSI **********************************************
; Die SSI-Schnittstelle wird initialisiert
;*******************************************************************************
init_SSI
        ClrReg  x:PCC,#>D_PCC           ; Pins der SSI-Schnittstelle sperren
        movep   #>D_CRA,x:CRA           ; Initialisiere Register CRA
        movep   #>D_CRB,x:CRB           ; Initialisiere Register CRB
        andi    #$FC,mr                 ; Setze Interruput Mask auf 0
        SetReg  x:PCC,#>D_PCC           ; Pins der SSI-Schnittstelle freigeben

        move    #RX_BUFF_BASE,r0        ; Zeiger r0 initialisieren
        move    #$FFFF,m0
        move    #4,n0                   ; Offset Register n0 initialisieren
        rts

;****** Interrupt Service Routine ssi_rx_isr ********************************
; Über die SSI-Schnittstelle wird gleichzeitig ein Datenwort gesendet und
; empfangen. Nach jeder Übertragung tritt ein 'receive interrupt' auf, der
; von dem folgenden Unterprogramm abgearbeitet wird.
;*******************************************************************************
ssi_rx_isr
        jclr    #3,x:SSISR,next_rx      ; If not fr. syc, jump to receive data.
        move    #RX_BUFF_BASE,r0        ; If frame sync, reset base pointer.
        nop
next_rx
        movep   x:(r0+n0),x:SSIDR
        movep   x:SSIDR,x:(r0)+         ; Read out received data to buffer.
        jsset   #3,x:SSISR,ProcessData  ; Process a pair of stereo samples
        rti
```

```
ssi_tx_isr
        nop                             ; Dieser Interrupt sollte niemals auftreten.
        rti

end_ssiisr
        nop
        endsec
;****** Ende des Programm Moduls ssiisr ************************************
```

7.3 Text über die SCI-Schnittstelle ausgeben

Modul Terminal

```
        ident   1,2
        page    132

;***************************************************************************
; Programm:     terminal.asm
; Zweck:Das Programm gibt über die SCI-Schnittstelle Text auf einem
;               VT100 Terminal aus. Zum Initialisieren der Schnittstelle und
;               zum Bearbeiten der Interrupts wird das Modul 'scii11.asm' verwendet.
; Autor:L. Kahl
; Version:      10.10.95/18.10.96
;***************************************************************************

        maclib  's:\dsp_mc\dsp\56k\macros\evm'  ; Pfad zu den Makro- und Include-Dateien

        initevm main                    ; Das Makro initialisiert das EVM Modul und
                                        ; sollte im Hauptprogramm immer vor allen
                                        ; Befehlen stehen. Reset Vektor und Stack-
                                        ; pointer werden gesetzt. Anschließend wird
                                        ; zur Marke 'main' gesprungen.

        include 'scii11.asm'            ; Weitere Programm Module

;****** Anfang des Programm Moduls Terminal *********************************
        section terminal
        xref    puts,InitSCI            ; Extern definierte Marken
        xdef    main

;****** Constants ***********************************************************
CR              equ     $0A0D

;****** Output text *********************************************************
        org     x:
Text1   dc      $00001B,'=**hello world!',CR
Text1E
Text2   dc      $00001B,'=+*This is the second text!',CR
Text2E
Text3   dc      $00001B,'=,*And now we have a third text.',CR
Text3E

;****** Start of program ****************************************************
        org     p:
main    jsr     InitSCI                         ; Initialize the SCI interface
text1   PutStrg #>Text1,#>Text1E                ; Output first text
text2   PutStrg #>Text2,#>Text2E                ; Output second text
text3   PutStrg #>Text3,#>Text3E                ; Output third text
        jmp     *                               ; Endless loop

        endsec
;****** Ende des Programm Moduls terminal ***********************************
```

Modul scii11.asm

```
        ident   1,2
        page    120

;***************************************************************************
; Programm:     scii11.asm
; Zweck:Das Module stellt eine Initialisierungsroutine und eine
;               Interrupt Service Routine für die SCI-Schnittstelle zur
;               Verfügung.
; Autor:L. Kahl
; Version:      10.10.95/18.10.96
;***************************************************************************

;****** Macro PutStrg ********************************************************

PutStrg macro   StartOfString,EndOfString
        move    StartOfString,x0
```

```
        move    EndOfString,x1
        jsr     puts
        endm

;****** Anfang Programm Modul scii ***********************************************
        section scii                    ; start of section ex1scii
        xdef    puts,InitSCI            ; Exported references

        maclib  's:\dsp_mc\dsp\56k\macros\evm'
        include 'evm.inc'


;****** Constants ****************************************************************
pc_txd  equ     1                       ; TxD-Pin of PCC-register

;****** Variables ****************************************************************
        org     x:
EndAdr  ds      1                       ; End address of output string
CurrAdr ds      1                       ; Current string address
ByteCnt ds      1                       ; Buffer for r5

;****** SCI Transmit Interrupt Vector ********************************************

        org     PL:i_SCI_TD
        jsr     txd_int

;*********************************************************************************
;       Interrupt Service Routine for Transmit Interrupt                         *
;       This routine takes the string and outputs each byte successively.        *
;       The transmit interrupt will be disabled if the end of string is          *
;       is reached.                                                              *
;*********************************************************************************

        org     p:
txd_int
        move    y1,x:(r6)+              ; Save data registers that are
        move    b0,x:(r6)+              ; used in the interrupt routine
        move    b1,x:(r6)+
        move    b2,x:(r6)+
        move    r1,x:(r6)+
        move    r5,x:(r6)+
        move    m5,x:(r6)+
        move    x:CurrAdr,r1            ; Pointer to text
        move    x:ByteCnt,r5            ; Pointer to transmit register
        move    #2,m5                   ; Set ring counter to 3

        move    x:(r1),y1               ; Get next byte
        move    y1,x:(r5)-              ; Transmit next byte
        move    r5,x:ByteCnt            ; Save pointer to transmit register
        jclr    #1,x:ByteCnt,skip_inc   ; Jump if next byte to transmit is not a high byte
                                        ; (Only for a high byte the bit #1 in x:ByteCnt would
be set).
        move    (r1)+                   ; Increment text address
                                        ;  The  address  register  is  incremented  every  third
byte.
        move    x:EndAdr,b              ; Get last address of text
        move    r1,x:CurrAdr            ; Save pointer to text
        move    r1,y1
        cmp     y1,b                    ; Check if end of text is reached.
        jne     skip_inc
        bclr    #SCR_TIE,x:SCR          ; Disable transmit interrupt
skip_inc
        move    x:-(r6),m5              ; Restore data register
        move    x:-(r6),r5
        move    x:-(r6),r1
        move    x:-(r6),b2
        move    x:-(r6),b1
        move    x:-(r6),b0
        move    x:-(r6),y1
        rti                             ; Return from exception

;*********************************************************************************
;       Subroutine InitSCI                                                       *
;       This routine initializses the SCI interface.                             *
;       The transfer rate is set to 9600 baude with one start and one stop bit   *
;       in asynchronous mode. The transmitter will be enabled, but not the       *
;       interrupt.                                                               *
;*********************************************************************************
```

```
InitSCI                                         ; Start of sci initialization sequence
        setimr  3                               ; Set interrupt mask to 3
        setipr  4,2,0                           ; Set interrupt priority of sci to 2
                ;  111111
                ;  5432109876543210
        movep   #>%0000001000001010,x:SCR       ; Enable transmit interrupt,
                                                ; set word mode 2
        movep   #>$020,x:SCCR                   ; Initialize divider
        bset    #PCC_TXD,x:PCC                  ; Assign TxD-pin to SCI-interface
        setimr  2                               ; Set interrupt mask to 2
        rts

;*****************************************************************************
;       Subroutine puts
;       The routine initializes the memory locations x:ByteCnt,X:CurrAdr and x:EndAdr.
;       x:CurrAdr       Pointer to the text that will be output.
;                       The pointer is incremented every third byte
;       x:EndAdr        Address of end of text.
;       x:ByteCnt       Pointer to one of the three transmit register.
;*****************************************************************************
puts
        btst    #SCR_TIE,x:SCR          ; Test if transmit interrupt is disabled
        jcs     puts                    ; Wait for trie to be disabled
        move    x0,x:CurrAdr            ; Pointer to text
        move    x1,x:EndAdr             ; Pointer to end of text
        move    #>SCIDRH,x1             ; Pointer to transmit register (use x1 as temporary
        move    x1,x:ByteCnt            ; register only)
        bset    #SCR_TIE,x:SCR          ; Enable transmit interrupt
        rts

        endsec                          ; End of section scii
;****** Ende Programm Modul scii ********************************************
```

7.4 Berechnung des Effektivwerts einer Folge von Abtastwerten

Modul rmstst.asm

```
        ident   1,2
        page    132

;*************************************************************************
; Programm:     rmstst.asm
; Zweck:Das Programm erzeugt im Y-Speicher eine Folge von Abtastwerten
;               mehrerer Sinusschwingungen. Anschließend wird deren Effektivwert
;               bestimmt. Der Effektivwert wird in der Speicherzelle x:RMSValue
;               abgelegt.
; Autor:L. Kahl
; Version:      10.10.95/18.10.96
;*************************************************************************

;****** Globale Konstanten ***************************************************
SampleNum       equ     $100
periods         equ     8
pi              equ     3.141592654
freq            equ     2.0*pi/@CVF(SampleNum)*periods
Peak            equ     1.0

        maclib  's:\dsp_mc\dsp\56k\macros\evm'
                                        ; Pfad zu den Makro- und Include-Dateien

        initevm main                    ; Das Makro initialisiert das EVM Modul und
                                        ; sollte im Hauptprogramm immer vor allen
                                        ; Befehlen stehen. Reset Vektor und Stack-
                                        ; pointer werden gesetzt. Anschließend wird
                                        ; zur Marke 'main' gesprungen.

        include 'rms.asm'               ; Weiter Programm Module

;****** Anfang des Programm Moduls rmstst ************************************
        section rmstst
        xref    ComputeRMS              ; Einsprungmarke
        xdef    RMSValue        ;
        xdef    main                    ; Einsprungmarke fuer 'initevm'

        org     y:$200
samples                                 ; Abtastwerte im Speicher erzeugen
count   set     0
        dup     SampleNum
        dc      Peak*(@SIN(@CVF(count)*freq))
count   set     count+1
        endm                        ;; End dup

;****** Variables ************************************************************
        org     x:
RMSValue
        ds      1

;****** Main program *********************************************************
        org     p:
main    move    #samples,r0                     ; Move buffer start address to r0
        move    #(SampleNum-1),n0               ; Create a ring buffer
        jsr     ComputeRMS                      ; Compute RMS of samples
end     jmp     *

        endsec
;****** Ende des Programm Moduls rmstst **************************************

        end
```

Modul rms.asm

```
        ident   1,2
        page    132

;*************************************************************************
; Programm:     ComputeRMS
; Zweck:Das Programm bestimmt den Effektivwert einer Folge von
```

```
;               Abtastwerten im Y-Speicher.
; Autor:L. Kahl
; Version:      10.10.95/18.10.96
;*************************************************************************
;
;                     ___________________
;                    /          ---
;                   /     1      \       2
; RMSValue   =  \  /     ---      |  x(n)
;                \/   SampleNum  /
;                             ---
;                          SampleNum
;*************************************************************************

        section rms

        maclib  's:\dsp_mc\dsp\56k\macros\motorola\fntns'

        xdef    ComputeRMS              ; Exported subroutine
        xref    RMSValue        ; External defined labels

factor  equ     1.0/@SQT(SampleNum)     ; Scaling factor is the reciprocal of the
                                        ; number of samples (global constant)
        org     p:                      ; Put subroutine in high P-memory

;****** Subroutine ComputeRMS ***************************************************
; Zweck:Die Routine bestimmt den Effektivwert einer Folge von Abtastwerten.
; Registers:    r0              Zeiger auf die Abtastwerte im Y-Speicher.
;               a,b,x,y1Werden verändert.
; Input:SampleNum       Anzahl der Abtastwerte, die zur Bestimmung des
;                               Effektivwerts herangezogen werden soll.
; Output:       RMSValueSpeicheradresse im X-Speicher unter welcher der
;                               errechnete Effektivwert gespeichert werden soll.
;*************************************************************************
        jmp     endComputeRMS           ; Skip code of subroutine
ComputeRMS
        move    #factor,x1              ; Get scaling factor
        clr     b       y:(r0)+,x0      ; Clear accumulator b, get first sample
        move    #SampleNum,y1
        do      y1,Loop                 ; loop
        mpy     x1,x0,a                 ; Scale sample
        move    a,x0                    ; Move sample back to x0
        mac     x0,x0,b y:(r0)+,x0      ; Square sample, get next sample
Loop    move    (r0)-                   ; Correct pointer to samples
        move    b1,y1                   ; Move squared sum of samples to b
        move    b0,y0
sqrt    sqrt3                           ; Get square root of y (macro)
Esqrt   move    b,x:RMSValue            ; Save RMS
        rts
endComputeRMS
        endsec
```

7.5 Beispielprojekt Effektivwertmeßgerät

Das folgend Programm kombiniert die letzten drei Beispiele zu einem Projekt. Mit dem DSP wird ein Effektivwertmeßgerät realisiert. Die Aufnahme der Meßwerte erfolgt mit der A/D-Umsetzerkarte DSP56ADC16 über die SSI-Schnittstelle. Die Meßwerte werden in einem Ringpuffer gespeichert. Sobald der Puffer halb gefüllt ist, wird von den Meßwerten der Effektivwert bestimmt. Anschließend erfolgt die Ausgabe des Effektivwertes auf ein Televideo 905 Terminal.

```
meter   ident   1,2
        page    120

;***************************************************************************
; Programm:     meter.asm
; Zweck:        Das Programm setzt analoge Spannungswerte in digitale Abtast-
;               werte um. Anschließend wird der Effektivwert der Folge von
;               Abtastwerten bestimmt und auf dem Televideo 905 Terminal ausgegeben.
; Autor:L. Kahl
; Version:      10.10.95/18.10.96
;***************************************************************************

;****** Globale Konstanten ********************************************

SampleNum  equ  $2000                   ; Number of Samples that are used to compute
BufStart   equ  $2000
BufLen     equ  2*SampleNum

LF         equ  $0A
CR         equ  $0D
NL         equ  $0D0A
ESC        equ  $1B

periods equ     8
pi      equ     3.141592654
freq    equ     2.0*pi/@CVF(SampleNum)*periods
Peak    equ     1.0

        maclib  's:\dsp_mc\dsp\56k\macros\evm'  ; Pfad zu den Makro- und Include-Dateien

        initevm main                    ; Das Makro initialisiert das EVM Modul und
                                        ; sollte im Hauptprogramm immer vor allen
                                        ; Befehlen stehen. Reset Vektor und Stack-
                                        ; pointer werden gesetzt. Anschließend wird
                                        ; zur Marke 'main' gesprungen.

        include '..\terminal\scii11.asm'
        include '..\codec\ssiisr6.asm'
        include 'io11.asm'
        include "..\rms\rms.asm"

;****** Anfang des Programm Moduls meter ***************************************
        section meter
        xref    puts,AddFrac,AddChar,InitSCI,IOScaler,BufPos,InitAddChar,ComputeRMS,init_SSI
        xdef    main,ProcessData
        xdef    RMSValue
        xref    tx_data_12,tx_data_56,tx_data_78,rx_data_12     ;ssiisr

        include 'evm.inc'
        include 'cs4215.inc'

;****** Konstanten ************************************************************
CTRL_12         equ     PREAMP_OFF+OUTAMP_HIGH+HI_PASS_ON+SAMP_RATE_48+STEREO+DATA_16
CTRL_34         equ     BITS_64+CODEC_MASTER

TONE_OUTPUT     equ     HEADPHONE_EN+LINEOUT_EN+(4*LEFT_ATTN)+(4*RIGHT_ATTN)
TONE_INPUT      equ     MIC_IN_SELECT+(15*MONITOR_ATTN)

;****** Variablen *************************************************************
                org     x:
RMSValuedc      0.75
StartMsgdc      $1B2A1B,$3D242A         ; Clear screen and set cursor to position (5,10)
                dc      'True RMS Meter   v1.1 by Lutz Kahl'
                dc      $1B,$3D2A2A             ; Set cursor to position (10,10)
```

```
                dc      'Input Voltage Vrms:          V'
StartMsg_end
rmsMsg_prefix   dc      $1B,($3D2020+10*256+31) ; Set cursor to position (10,31)
rmsMsg_value    dc      '0.000000'
rmsMsg_end

                org     y:BufStart
buffer          ds      BufLen                  ; Ringpuffer

;****** Start of main program ***************************************************
        org     p:
main
        jsr     InitMain
        jsr     InitSCI                         ; Initialize SCI
        PutStrg #>StartMsg,#>StartMsg_end       ; Output Header
        jsr     init_cs4215
        jsr     init_SSI                        ; Initialize the SSI interface
        move    #TONE_OUTPUT,y0                 ; headphones, line out, mute spkr, no attn.
        move            y0,x:tx_data_56
        move    #TONE_INPUT,y0                  ; no input gain, monitor mute
        move            y0,x:tx_data_78

mainloop
        jsr     WaitBufferFull
        jsr     ComputeRMS
        jsr     OutputResult
end     jmp     mainloop

;****** End of main program *****************************************************
        org     ph:
InitMain
        move    #buffer,r4              ; Initialize buffer base address for
                                        ; RMS-subroutine
        move    r4,r2                   ; Initialize buffer base address for
                                        ; SSI interrupt service routine
        move    r4,r3                   ; Use r3 as reference for WaitBufferFull
        move    #(BufLen-1),m4          ; Create a ring buffer for RMS-subroutine
        move    m4,m2                   ; Create a ring buffer for SSI-routine
        move    m4,m3                   ; Reference for buffer full
        move    #SampleNum,n3           ; Step width for reference pointer
        rts

OutputResult
        InitIO  #rmsMsg_value,1
        move    x:RMSValue,a1           ; Get rms value
        asl     a                       ; Multiply rms value by two
        jsr     AddFrac
OutStr  PutStrg #>rmsMsg_prefix,x:BufPos
        rts

WaitBufferFull
        move    (r3)+n3                 ; Get next full address
        move    r3,a                    ; Get reference address
DoWait  move    r2,x0                   ; Get current receiver buffer address
        cmp     x0,a                    ; Compare to rms address
        jne     DoWait                  ; Wait if not equal
        rts

;****** Unterprogramm ProcessData ***********************************************
ProcessData
        move    y:(r4),a        ; get old left sample
        move    a,x:tx_data_12          ; Put value in left channel tx.
        move    x:rx_data_12,a          ; get new left sample
        move    a,y:(r4)                ; store new left sample
        rts

init_cs4215
        cs4215I #CTRL_12,#CTRL_34       ; Makroaufruf zum Initialisieren des CS4215
        rts                             ; Die Kontrollworte werden dem Makro als
                                        ; Parameter übergeben.
        endsec
```

Modul io11.asm

```
        ident   1,1
        page    120
```

```
;****** Macro InitIO ************************************************************
; Zweck:Das Makro bereitet die Register r5,m5 und r4 für die Unterprogramme
;               AddFrac und AddChar vor.
; Input:BufAdr          Anfangsadresse des Zeichenpuffers
;               BytePos         Position des ersten anzufügenden Bytes im Zeichenpuffer:
;                               1 = high byte
;                               2 = medium byte
;                               0 = low byte
;*******************************************************************************
InitIO          macro   BufAdr,BytePos
                move    r5,y:(r6)+              ; Save registers
                move    r4,y:(r6)+
                move    #(IOScaler+BytePos),r5
                move    BufAdr,r4
                jsr     InitAddChar
                endm

        section io
        xdef    AddFrac,AddChar,IOScaler,InitAddChar,BufPos
        xref    puts

        org     xl:$3C
IOScaler
        dc      $000001         ; low byte
        dc      $010000         ; high byte
        dc      $000100         ; medium byte
BitSft  ds      1               ;
        org     x:
BufPos  ds      1

        org     p:
InitAddChar
        move    r4,x:BufPos
        move    r5,x:BitSft
        move    y:-(r6),r4
        move    y:-(r6),r5
        rts

;****** Subroutine AddFrac ******************************************************
; Zweck:Die Routine fügt an eine Zeichenkette im Speicher eine Fractional
;               Zahl an.
; Input:a1      Enthält die zu konvertierende Fractional Zahl.
; Output:       x:(r4)  Ab dieser Speicheradresse wird die Zahl abgespeichert.
;               r5      Byte Position innerhalb eines Wortes, ab welcher das Speichern
;                       der Zahl beginnen soll.
; Register:     b,a,x,y Werden verändert.
;*******************************************************************************
AddFrac
        move    a1,x1           ; Copy fractional number to x1
        move    #>'0',a
        jsr     AddChar
        move    #>'.',a
        jsr     AddChar
        move    #>10,x0
        do      #6,loop         ; Loop over six digits
        move    #>$30,a         ; Get Offset for digit zero
mpy10a  mac     x1,x0,a         ; Multiply register x1 times 10
        jsr     AddChar         ; Add current character to buffer
        move    #0,a1           ; Clear last char
        asr     a               ; Correct remainder
        move    a0,x1           ; Store remainder back in x1
loop    nop
        rts

;****** Subroutine AddChar ******************************************************
; Zweck:Die Routine fügt an eine Zeichenkette im Speicher ein weiteres
;               Byte an. Es werden immer drei Bytes zu einem Wort zusammengefaßt.
; Input:a1      Enthält in den niederwertigsten acht bit das anzufügende Byte.
; Output:       x:(r4)  Unter der Speicheradress wird das anzufügende Byte abgelegt.
; Register:     r5      Posititon des Bytes innerhalb eines 24 Bit Wortes.
;               b,y     Werden verändert.
;*******************************************************************************
AddChar
        move    r4,y:(r6)+      ; Save registers
        move    r5,y:(r6)+
        move    m5,y:(r6)+
        move    x:BufPos,r4     ; Get current position in text buffer
```

```
        move    x:BitSft,r5     ; Get r5 from memory
        move    #2,m5           ; Set ring buffer lenght to 3
        move    a1,y1           ; Get character from register a1
        move    x:(r5)+,y0      ; Get Scale factor
        mpy     y0,y1,b         ; Shifted character is in b0
        asr     b               ; correct shift for one bit
        move    b0,b1           ;
        jset    #0,x:BitSft,SkipAdd
        move    x:(r4),y1
        add     y1,b            ; Add current byte to the previously computed word
SkipAdd
        move    r5,x:BitSft     ; Save r5 to memory
        jset    #0,x:BitSft,SaveAndInc
        move    b1,x:(r4)       ; Save word
        jmp     restore
SaveAndInc
        move    b1,x:(r4)+      ; Save word and increment r4
        move    r4,x:BufPos
restore move    y:-(r6),m5      ; Restore registers
        move    y:-(r6),r5
        move    y:-(r6),r4
        rts

        endsec  ; io
```

8 Literaturverzeichnis

[1] Azizi, S.A.:
Enttwurf und Realisierung digitaler Filter,
4. Auflage, München, Odenbourg Verlag, 1988

[2] Brigham, E.O.:
FFT Schnelle Fourier Transformation,
3. Auflage, München, Oldenbourg Verlag, 1989

[3] Cooley, J.W., Tukey, J.W.;
An Algorithm for the Machine Calculation of Complex Fourier Series,
Math. Computation, Vol 19, pp. 297-301, April 1965

[4] Cooley, James W. :
How the FFT Gained Acceptance
IEEE Signal Processing Magazine, Jan. 1992.

[5] Fabig, A.: Analoge Welt A/D,
c't, Heinz Heise Verlag, März 1989, S.198 ff

[6] Gauss, Carl F. :
"Nachlass: Theoria interpolationis methodo novo tractata."
Carl Friedrich Gauss, Werke, Band 3,
Göttingen : Königliche Gesellschaft der Wissenschaften,
Seite 265-303, 1866.

[7] INMOS Manual IMS A100

[8] Jayant, S.N.; Noll, P.:
Digital Coding of Waveforms,
Englewood Cliffs, N.J., Prentice Hall Inc., 1984

[9] Lüke, Hans-Dieter: Signalübertragung,
5. Auflage, Berlin, Springer Verlag, 1992

[10] Ludeman, L.C.: Fundamentals of Digital Signal Prozessing,
Harper & Row, 1986

[11] Motorola : DSP5600/1 Digital Signal Processor User's
Manual, DSP56000UM/AD Rev. 2,
Phoenix, Arizona 85036, Motorola Inc., 1990

[12] Motorola : DSP56002 Advance Information, ADI1290 Rev. 1,
Motorola Inc., 1988

[13] Motorola : DSP56002 24-Bit General Purpose Digital Signal Processor,
DSP56002/D Rev. 2,
Motorola Inc., 1991

[14] Motorola :Advance Information DSP56200, ADI1257R1,
Digital Filter Chip, Motorola Inc., 1988

[15] Noll, Peter; Jayant, N.S.:Digital Coding of Waveforms,
Prentice-Hall, Inc., Englewood Cliffs, New Jersey,1984

[16] Oppenheim, Alan V., Schafer, Ronald W.:
Digital Signal Processing,
Prentice-Hall, Inc., 1975, ISBN 0-13-214635-5.

[17] Oppenheim, W.V.;Willsky, A.S.:
Signale und Systeme,
5. Auflage, Weinheim, VCH Verlagsgesellschaft, 1989

[18] Rabiner, L.R., Gold, B.:
Theorie and Application of Digital Signal Processing,
Englewood Cliffs, Prentice Hall, 1974

[19] Runge, C.; König, H. :
"Band XI, Vorlesung über Numerisches Rechnen."
Grundlehren Math. Wissenschaften,
Verlag Julius Springer, Berlin, 1924.

[20] Tietze, U.;Schenk, Ch.:
Halbleiterschaltungstechnik,
9. Auflage, Berlin, Springer Verlag 1990

[21] John G. Proakis, Dimitris G. Manolakis:
Digital Signal Processing,
Principles, Algorithms, and Applications
Third Edition, Prentice Hall, 1996

Anhang A: Schaltpläne des DSP56002 EVM