Entwicklung einer DSP-basierten PCI-Karte zur Implementierung von Tracking-Algorithmen

II. Physikalisches Institut
Justus-Liebig-Universität
Gießen

November 1998

Diplomarbeit von Ingo Fröhlich
aus Echzell, OT Bisses

Zusammenfassung

Die Analyse von Spektrometerdaten mit mehreren Teilchenspuren erfordert eine hohe Rechenleistung. Um diese zu Verfügung zu stellen und damit den Zeitverbrauch zu reduzieren, wurde für Experimente des DISTO-Spektrometers (Saturne, Frankreich) eine PCI-Karte für die Offline-Analyse entwickelt und mit Hilfe mehrerer Beispielprogramme getestet. Diese Karte ist mit sechs **d**igitalen **S**ignal**p**rozessoren (DSPs der Firma Analog Devices) und zusätzlichen lokalen Speicher-Bänken bestückt, wobei jede Speicher-DSP-Gruppe einzeln über den PCI-Bus angesteuert werden kann. Die Datenweitergabe in dem parallelisierten Konzept erfolgt durch schnelle Link-Ports, wodurch der Algorithmus für den »Track-Fit« beschleunigt werden kann. Dieser rechenintensive Teil der Datenanalyse berechnet aus Detektorortsinformationen die Größen der Spurparameter eines Teilchens. Durch vielseitige Programmier- und Konfigurationsmöglichkeiten kann die Karte an viele Aufgaben angepaßt werden. Eine Weiterentwicklung dieses Konzeptes ist wegen der zu erwartenden Rechenleistungsprobleme im Rahmen der Analyse der HADES-Daten (**H**igh **A**cceptance **D**i **E**lectron **S**pectrometer) nötig und kann auch mit DSPs der neuen Generation mit vertretbarem Aufwand erfolgen.

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

Der Grundgedanke eines Magnetspektrometers ist die Bestimmung des Impulses geladener Teilchen durch die Verfolgung ihrer Spur mit Hilfe von ortsempfindlichen Detektoren. Auf diese Weise kann man auch kurzlebige Teilchen durch ihre Zerfallsprodukte rekonstruieren.

Ein Beispiel für ein solches Experiment ist HADES (**H**igh-**A**cceptance **D**i-**E**lektron **S**pectrometer), welches zur Zeit am Schwerionensynchrotron (SIS) der GSI (**G**esellschaft für **S**chwer**i**onenforschung) aufgebaut und 1999 in Betrieb gehen wird. Das physikalische Programm dieses Spektrometers erfaßt sowohl »In-Medium-Modifikationen« der leichten Vektormesonen (ρ, ω, ϕ) als auch die Messung elektromagnetischer Formfaktoren [1].

Diese »In-Medium-Modifikationen« werden experimentell durch Stöße von Schwerionen untersucht, wobei kurzzeitig eine heiße, vedichtete Zone von Kernmaterie produziert wird. Für diesen Fall wird eine Veränderung der Masse von Hadronen durch die sog. »Restauration der chiralen Symmetrie« vorhergesagt [2]. Da in diesem Modell alle Hadronen aufgrund des Vorhandenseins eines »chiralen Kondensates« eine vergrößerte Masse haben soll, müßte in verdichteter Materie dieses Kondensat dünner werden (sog. »Restauration«). Die daraus resultierende Massenveränderung der Hadronen soll mit dem Spektrometer HADES durch den elektromagnetischen Zerfall der leichten Vektormesonen untersucht werden. Bei einem zentalen Stoß von schweren Kernen entstehen jedoch eine Vielzahl von geladenen Teilchen, welche die Auswertung erschweren.

Das fertiggestellte HADES-Spektrometer soll eine Massenauflösung von weniger als einem Prozent haben. Damit wird auch eine gute Ortsauflösung benötigt, d.h. die Position der Reaktionsprodukte, die durch den Detektor fliegen, muß an mehreren Stellen mit einer Auflösung von $\sigma = 100\mu m$ bestimmt werden. Die Aufgabe, aus den gewonnen Ortsdaten (also den Punkten, wo sich die Teilchenspur mit

geeigneten Detektoren kreuzt) die physikalisch interessanten Informationen zu gewinnen, wie z.B. den Impuls der Teilchen kurz nach einer Reaktion, bzw. einem anschließenden Zerfall, wird im folgenden »Track-Fitting« genannt. Ein weiterer Punkt ist das kombinatorische Problem. Bei der Datenanalyse muß geklärt werden, welches Detektorsignal zu welcher Teilchenspur gehört. Beides zusammen bildet ein numerisch aufwendiges Verfahren, daß heutzutage auf Workstations oder 80x86-kompatiblen Rechnern (sog. **P**ersonal **C**omputers)durchgeführt wird. Dies erscheint zunächst als einfache und preiswerte Variante. Andererseits wurden diese Maschinen für andere Aufgaben konzipiert und sind von ihrer Architektur her zwar extrem flexibel, was die darauf laufenden Programme betrifft, bieten aber wenig Möglichkeiten, Aufgaben zu parallelisieren[1]. So stellt sich die Frage, ob nicht andere Hardware-Topologien für dieses spezielle Problem geeigneter sein könnten.

Gerade bei den hohen Teilchenzahlen[2], die für HADES zu erwarten sind, ist mit einem enormen Anstieg der Analysedauer zu rechnen, der auf konventionellem Weg nur durch eine Erhöhung der Rechnerzahl begegnet werden kann.

Die vorliegende Arbeit soll daher mit der Entwicklung einer speziellen Hardware ein Alternativkonzept verfolgen, um die benötigte Rechenleistung zur Verfügung zu stellen. Da sich HADES noch im Aufbau befindet, aber reale Daten, bzw. ein Vorbild für einen Algorithmus für einen kritischen Test dieser Hardware benötigt werden, wurden die Daten eines anderen Magnetspektrometers herangezogen. Es handelt sich dabei um das Experiment der DISTO-Kollaboration (**D**ubna-**I**ndiana-**S**aclay-**TO**rino-GSI-Krakow-Gießen-Frankfurt), dessen Ergebnisse zur Zeit ausgewertet werden und welches deshalb als realitätsnahes Vergleichsobjekt der Leistung beider Systeme - der heutigen Rechnergeneration und der speziell entwickelten Tracking-Hardware - dienen kann.

Zunächst soll daher das DISTO-Spektrometer und seine Komponenten kurz beschrieben werden. Danach ist eine Untersuchung des Analyseverfahrens notwendig, damit ein Hardware-Konzept erstellt werden kann. Bestandteil der Aufgabenstellung ist, daß die Hardware so konfigurierbar bleibt, daß sie auch für ähnliche numerische Verfahren, wie die HADES-Analyse, zur Verfügung steht. Darauffolgend werden die verwendeten Bauteile, die technische Realisierung und die durchgeführten Tests vorgestellt. Details, wie z.B. die Quelltexte finden sich im Anhang.

[1] Für den Pentium Pro und den Nachfolger Pentium II gibt es die Möglichkeit, mehrere Prozessoren in einem System zu betreiben. Diese müssen sich jedoch den Arbeitsspeicher teilen, so daß die Rechenleistung i.A. nicht mit der Anzahl der Prozessoren skaliert.

[2] Z.B. 200 geladene Teilchen bei Au+Au bei 1AGeV.

Kapitel 2

Das zugrundeliegende Experiment: DISTO

Die Entwicklung einer speziellen Hardware zur Implementierung von »Tracking-Algorithmen« läßt sich nur auf der Basis eines konkreten Experimentes durchführen, da jeder Algorithmus speziell auf den Aufbau des Experimentes angepaßt werden muß. Als Basis für diese Arbeit diente dafür das DISTO-Spektrometer. Bevor eine Hardware entwickelt und bewertet werden kann, muß der Algorithmus untersucht werden, und dieser kann nur mit dem Wissen über den Aufbau des zugrundeliegenden Experimentes verstanden werden. Daher widmet sich dieses Kapitel den wesentlichen Elementen des DISTO-Spektrometers.

2.1 Vorbemerkung

Das Disto-Spektrometer war im Laboratoire National Saturne (LNS) in Saclay, Frankreich, aufgebaut. Dieses Labor lieferte einen polarisierten Protonenstrahl, womit die Produktionen von

- Λ- und Σ^0-Hyperonen (Y), $\vec{p}p \rightarrow pK^+Y$
- η-, ω-, $\eta\prime$ und ϕ-Mesonen (X), $\vec{p}p \rightarrow ppX$

bis zu einer Strahlenergie von 2.85 GeV untersucht werden konnte. Als Ende 1997 das LNS aufgelöst und die Beschleunigeranlagen entgültig abgeschaltet wurden, wurde auch das DISTO-Spektrometer abgebaut. Die noch andauernden Analysen produzieren nun Ergebnisse, die auch für die Interpretation der HADES-Dileptonenspektren erforderlich sein werden, insbesondere die Untersuchungen

zur ϕ-Mesonen-Produktion nahe der Schwelle [3]. Erste Ergebnisse der Untersuchungen - sowohl zu Hyperonen, als auch zu Mesonen - sind nun zur veröffentlicht [5, 6].

2.2 Elemente des Spektrometers

Details über den Aufbau des DISTO-Spektrometers finden sich an anderer Stelle [4]. Im folgenden wird ein Überblick gegeben, tiefergehende Informationen finden sich nur, falls es für das Verständnis anschließender Abschnitte erforderlich ist.

2.2.1 Übersicht

Das DISTO-Experiment war ein Magnetspektrometer mit einem Flüssig-Wasserstoff-Target. Sämtliche Detektoren waren in zwei Armen angeordnet, zwischen welchen der Protonenstrahl den Aufbau verließ. Der Dipol-Magnet überdeckte eine kreisrunde Fläche, innerhalb derer sich pro Arm zwei Kammern mit szintillierenden Fasern befanden (vgl. Abb 2.1). Jede dieser Kammern beinhaltete drei Ebenen, in welcher die Fasern in verschiedene Richtungen liefen. Außerhalb des vom Magnetfeld abgedeckten Bereiches schlossen sich zwei Ebenenen mit Vieldrahtproportionalkammern an (MWPC = **M**ulti **W**ire **P**roportional **C**hamber), auch jeweils aus drei Ebenen mit unterschiedlichen Drahtrichtungen bestehend. Weiterhin gab es ein Hodoskop aus szintillierenden Streifen, Cherenkov-Detektoren und zwei Szintillatoren, die als Polarimeter dienten.

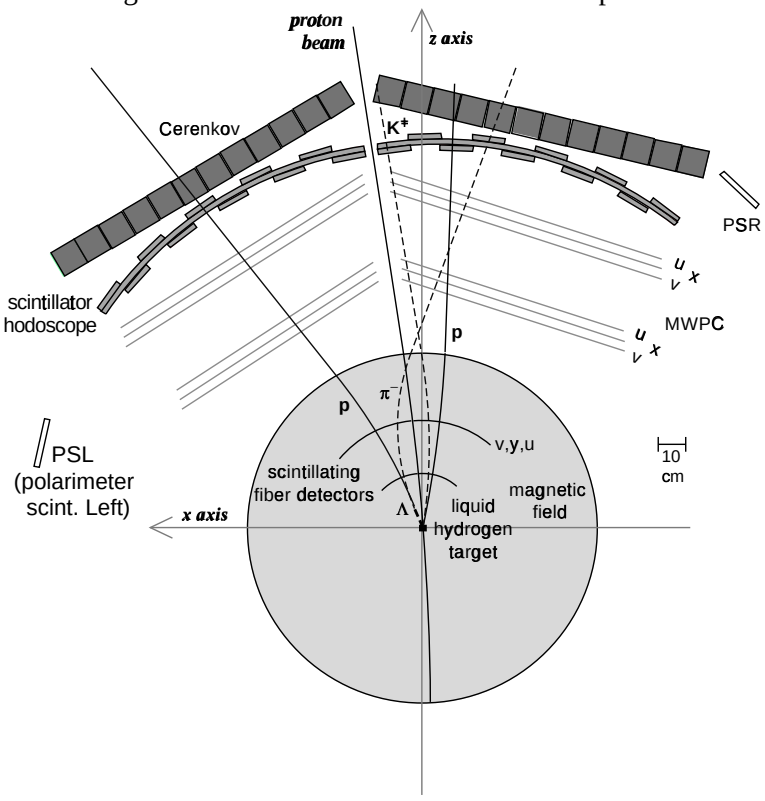
2.2.2 Die Komponenten

Strahl Das Spektrometer wurde mit einem Strahl polarisierter Protonen bis zu einer kinetischen Energie von 2,85 GeV betrieben. Der Polarisationsgrad erreichte dabei ca. 70% [4]. Wie bei jedem Synchrotron gab es auch beim LNS eine Beschleunigungs- und eine Extraktionsphase. Hier betrugen die »Spill-On-Zeiten« - also die Zeit, in welcher der Strahl für das Experiment zur Verfügung stand - 400ms, dies wiederholte sich bei einer Energie von 2,85 GeV alle 4,2 Sekunden.

Der ausgehende Strahl wurde durch ein Kohlefaserrohr zwischen den beiden Spektrometerarmen herausgeführt. Um Streuungen zu unterbinden, war dieses Rohr evakuiert. Der Ablenkwinkel des Strahls - verursacht durch das Magnetfeld - betrug $8{,}7^{o}$.

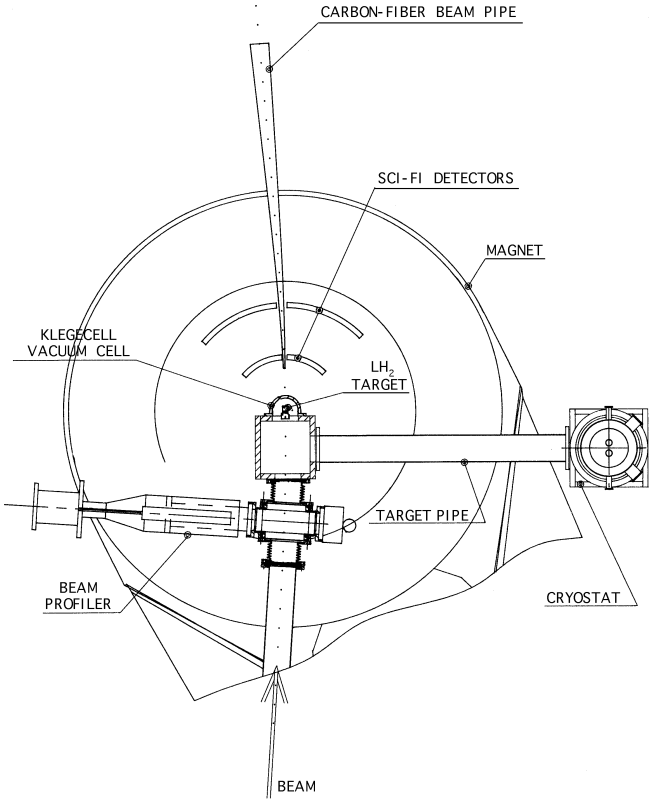
Magnet Der Dipolmagnet wurde bei einer Strahlenergie von 2,85GeV mit einer Feldstärke von 1,49 Tesla betrieben. Die genaue Feldstärke mußte bei jeder

Abbildung 2.1: Schematische Ansicht des Disto-Spektrometers



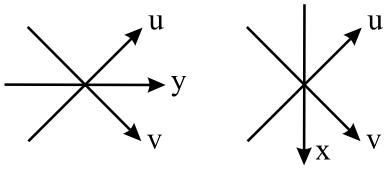
Blick von oben auf das DISTO-Spektrometer, entnommen aus [4]. Der Protonenstrahl durchquert den Detektor im Bild von unten nach oben. Die graue Fläche zeigt den Bereich des Dipol-Magnetfeldes. Von der Strahlrichtung aus folgen: Ebenen mit szintillierenden Fasern (fibers), Vieldraht-Proportionalkammern (MWPC), ein Hodoskop und Cherenkov-Detektoren, sowie das Polarimeter (PSR/PSL). Die Trajektorien stellen ein Ereignis der Reaktion $pp \to pK^+\Lambda$ dar.

Abbildung 2.2: Targetregion



Details der Target-Region, entnommen aus [4]. Man erkennt das eingehende Strahlrohr, die Kühlvorrichtung für den flüssigen Wasserstoff, die szintillierenden Fasern und das Rohr für den ausgehenden Strahl.

Abbildung 2.3: Richtung der Fasern und Drähte



Zur Verdeutlichung der Tripletts: Links die Richtungsbezeichnungen der Fasern, rechts die der MWPCs.

Energie erneut eingestellt werden, um den ausgehenden Protonenstrahl auf dem geforderten Ablenkwinkel zu halten.

Szintillierende Fasern Durchquert ein ionisierendes Partikel eine szintillierende Faser, wird diese zum Leuchten angeregt.

Wie schon erwähnt, befanden sich noch innerhalb des Magnetfeldes vier Kammern mit szintillierenden Fasern - jeweils zwei pro Spektrometerarm. Beide hatten zylinderförmige Geometrie, so daß sie einen festen Radius um das Target hatten. Davon hatte die eine einen Radius von 20 cm und die andere von 40 cm. In der mittlersten Ebene waren die Fasern jeweils horizontal angeordnet (y-Koordinate). Die beiden anderen Ebenen waren um $\pm 45^o$ verdreht (u, v-Koordinaten, vgl. Abb. 2.3). Diese Kombination soll im folgenden als »Faser-Triplett« bezeichnet werden.

Die Konvertierung des Faserlichtes in ein elektrisches Signal erfolgte durch positionsempfindliche Photomultiplier[1] (PSPM = **P**osition-**S**ensitive **P**hoto**M**ultiplier). Das elektrische Signal wurde dann verstärkt und mit einem LeCroy PCOS III-System ausgelesen (s.u.). Um die gewünschten Signale vom Untergrundrauschen zu trennen, muß dabei am Verstärker eine gewisse Schwelle eingestellt sein, unterhalb derer ein Signal unterdrückt wird. Die Höhe dieser Schwelle ist ein Kompromiß zwischen Rauschunterdrückung und der Wahrscheinlichkeit, daß ein Signal von einem Teilchen die Schwelle überschreitet. Diese Wahrscheinlichkeit wird Effizienz genannt und hängt von der Energie und der Teilchensorte ab; hier betrugen die Effezienzen für die einzelnen Ebenen um die 70% [4].

Vieldraht-Proportionalkammern Wie schon die Faser-Tripletts dienten auch die Drahtkammerm zur Verfolgung der Trajektorien, waren aber planar. Da auch sie drei verschiedene Ebenen mit unterschiedlichen Drahtrichtungen beinhalteten, sollen sie »Drahtkammer-Triplett« genannt werden.

Die mit einem Argon-Isobutan-Gemisch gefüllten Kammern wurden ebenfalls mit dem PCOS III-System ausgelesen.

[1] Typ Hamamatsu H5828, 80 Kanäle, 19 Stufen, Verstärkung insgesamt $5 \cdot 10^6$.

Hodoskop Das Hodoskop bestand aus 32 Streifen aus szintillierendem Plastik, welche in x- und y-Richtung angeordnet waren. Die gesamte Hodoskopebene folgte - wie schon die Faserebenen - einer zylindrischen Geometrie. Der Radius zum Target betrug hier 143cm. Eine Aufgabe dieses Detektorteils war es, Informationen über die Multiplizität für den Hardeware-Trigger zur Verfügung zu stellen. Bei der späteren Analyse diente die deponierte Energie eines Teilchens und die Flugzeit zur Teilchenidentifikation.

Die Auslese der Plastikstreifen erfolgte auf einer Seite; Pulshöhe und Zeitinformation wurden mit LeCroy FERA ADCs (**A**nalog to **D**igital **C**onverter) digitalisiert. Nach Software-Korrekturen betrug hier die Zeitauflösung $\sigma = 800$ ps.

Zwei weitere Plastik-Szintillatoren dienten als Polarimeter; mit ihnen wurde in 2 von 20 »Spills«[2] die elastische Streuung zwischen den Protonen gemessen.

Cherenkov-Detektor Den Abschluß des Aufbaus bildete ein Array von Wasser-Cherenkov-Detektoren, bestehend aus 12 Tanks pro Arm mit 90cm Höhe und einem Querschnitt von 10cm $\times$ 10cm. Zusammen mit der Pulshöhen- und relativen Zeitinformation der Plastikhodoskope dient der Cherenkov-Detektor bei der Analyse zur Identifizierung der Teilchensorte.

Die Auslese der Cherenkov-Lichtes erfolgte auf beiden Seiten, so daß die FERA-Elektronik insgesamt 48 Zeit- und 48 Pulshöhensignale zu verarbeiten hatte.

Vielteilchen-Trigger Während der normalen Datenaufnahme diente dieser Trigger dazu, die Rate bei ca. 3000 Ereignisse pro Spill zu halten, um die Totzeit möglichst gering zu halten. Ziel war es, möglichst nur Ereignisse mit vier geladenen Teilchen im Endkanal zu speichern. Der Vielteilchen-Trigger lieferte eine positive Entscheidung, wenn folgende wesentlichen Bedingungen erfüllt waren:

1. Mindestens sieben Hodoskopelemente haben ein Signal geliefert.

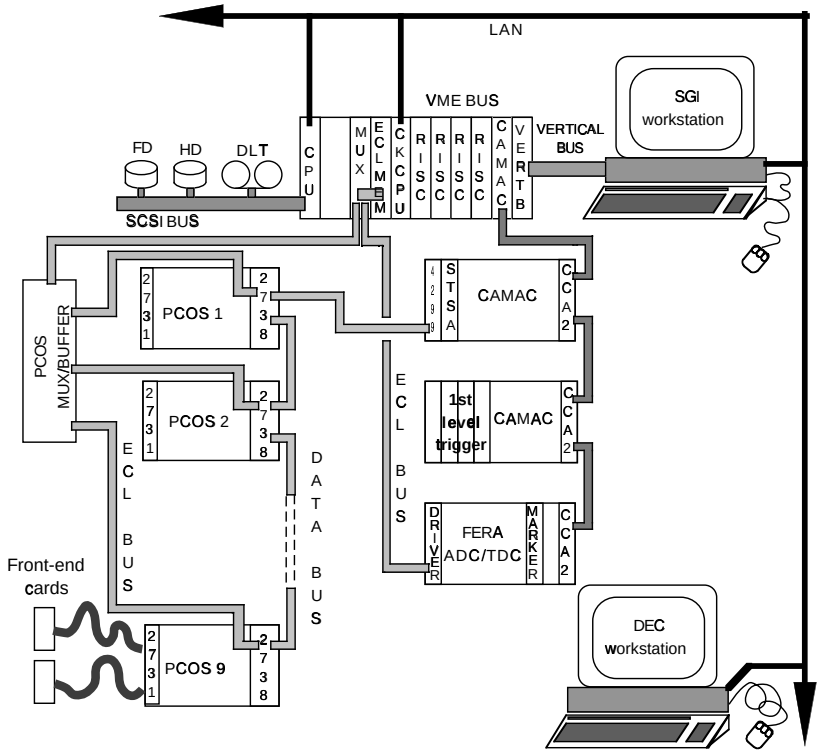
2. Mindestens drei Signale für das äußere Fasertriplett.

3. Mindestens zwei Signale für das innere Triplett.

Punkt 2 trägt der niedrigen Effizienz der Fasern Rechnung; Punkt 3 wurde so gewählt, da ein nach durchschnittlich 20 cm zerfallendes Λ auch noch eine positive Entscheidung erreichen sollte.

[2] Mit einem »Spill« bezeichnet man die Phase der Spill-On-Zeit.

Abbildung 2.4: Disto-Datenerfassung



Übersicht über das Datenerfassungssystem, entnommen aus [4].

Datenerfassung Eine Übersicht über die Datenerfassung liefert Abb. 2.4. Die Datenübermittlung von der FERA- und PCOS-Elektronik erfolgte über einen ECL-Bus. ECL (**E**mitter **C**oupled **L**ogic) zeichnet sich durch gute Übertragungsqualität bei hohen Frequenzen und Weglängen aus. Das Aufnahme-System selbst befand sich in einem VME-Crate (VME = **V**ersa **M**odule **E**urocard). Die dort untergebrachten Module hatten die Aufgabe, die Daten zunächst zwischenzuspeichern, um die gesamte »Spill-Off-Phase« zum Wegschreiben auf ein Magnetband zu nutzen. Eine weitere VME-Einheit gab die Daten auf das lokale Netz (LAN = **L**ocal **A**rea **N**etwork), um die Ereignisse auf zwei Rechnern direkt beobachten zu können.

Kapitel 3

Der Tracking-Algorithmus

Das folgende Kapitel geht auf den schon existierenden Algorithmus zum Track-Fitting für Ereignisse mit einem Vertex (Mesonenzerfälle) zur Auswertung der DISTO-Daten ein. Zum Verständnis des späteren Hardware-Konzeptes ist eine Analyse dieses Algorithmus nötig. Signifikante Beschleunigungen von Algorithmen sind nur dann möglich, wenn diese genau betrachtet werden und die numerischen Teile, die bisher am meisten Rechenzeit verbraucht haben, isoliert werden. Nur so können für diese Teile die Anforderungen an eine Hardware formuliert werden.

3.1 Einführung

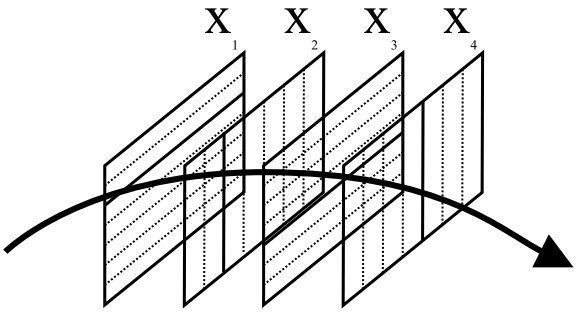
3.1.1 Tracking für ein Teilchen

Die Detektoren eines Magnetspektrometers liefern Ortsinformationen. Daraus müssen im wesentlichen die Impulse der beteiligten Teilchen berechnet werden. Da das Magnetfeld und die grundlegenden Bewegungsgleichungen bekannt sind, kann bei bekannten Anfangsbedingungen für jedes Teilchen die Spur berechnet werden und für ausgesuchte Werte in Tabellen gespeichert werden (»look-up-Tabelle«). Die experimentellen Gegebenheiten erfordern jedoch eine Invertierung dieser Funktion, denn diese Anfangsbedingungen werden ja gesucht. Dies geschieht durch Anwendung eines numerischen Ableitungsverfahren.

Ein Teilchen hat sechs Freiheitsgrade (drei des Ortes und drei des Impulses), eine Teilchenspur hat jedoch nur fünf Freiheitsgrade[1]. Hier wird i.A. der Schnittpunkt der Spur mit der Vertex-Ebene und der Impulsvektor an dieser Stelle gewählt.

[1] Ein Freiheitsgrad geht in den Längenparameter dieser Spur.

Abbildung 3.1: Spurverfolgung mit Detektoren



Mit Detektoren, die nur in einer Richtung empfindlich sind - wie z.B. Faserebenen, MWPCs oder MDCs - wird die Spur eines Teilchens verfolgt. Wichtig ist dabei, daß die ortempfindliche Richtung von Ebene zu Ebene immer wieder gedreht wird.

Um die Teilchenspur zu erfassen, werden bei Experimenten in der Kern- und Teilchenphysik in geeigneter Weise Vieldrahtkammern oder Faser-Detektoren in der zu erwartenden Flugrichtung montiert. Die Information, welche Drähte (oder szintillierende Fasern) von dem Teilchen auf seiner Spur angesprochen worden sind, wird im folgenden als *Koordinate* $\vec{x}$ bezeichnet, die Freiheitsgrade als *Parameter* $\vec{p}$. Abb. 3.1 zeigt ein Gedankenexperiment mit einer Trajektorie, die durch ein solchen Aufbau verläuft und dabei in jeder Detektorebene ein Signal erzeugt. In diesem Experiment sollen die Ebenen eine Effizienz von 100% haben (d.h. jedes Teilchen erzeugt ein Signal, ein Untergrundrauschen ist nicht vorhanden), was in der Praxis nicht erreicht wird. Idealerweise erhält man so für jede Spur so viele Koordinaten x_i wie Detektoren *hintereinander* plaziert worden sind.

Die gesamte experimentelle Konfiguration wird also gut durch eine Funktion $\vec{x} = F(\vec{p})$ beschrieben, welche die 5 Parameter auf die Koordinaten (im Falle von DISTO sind es 12 Stück) abbildet. Diese Funktion ist durch Simulationen darstellbar und wird in einer sogenannten »Look-up-Tabelle« gespeichert. Um den Speicherverbrauch zu reduzieren, wird die Granularität bewußt gering gehalten, dies erfordert eine anschließende Interpolation. Die Umkehrfunktion (Das Experiment liefert die Koordinaten, gesucht sind die Parameter) wird durch das in [7] beschriebene Verfahren implementiert, das im folgenden Abschnitt vorgestellt wird.

3.1.2 Handhabung des Vielteilchenproblems

Bei den modernen Experimenten (die sich durch hohe Akkzeptanz auszeichnen) ist die Zahl der registrierten Teilchen größer als eins. Das macht die Analyse schwieriger, da in den Detektorebenen dann mehrere Drähte oder Fasern angesprochen

wurden und i.A. nicht a priori klar wird, wie die Zuordnung ist. Die Suche nach der richtigen Kombination wird »Track-Finding« genannt. Dies geschieht durch Fitten aller in Frage kommenden Kombinationen und anschließender Analyse dieser Fits. Dies erfolgt durch Übergabe eines »Güte-Parameters« χ^2 für jeden Fit. Man kann sich leicht überlegen, daß die Rechenzeit so mit m^i steigt (m=Teilchenzahl, i=Anzahl der Detektorebenen).

3.2 Das Quasi-Newton-Verfahren für die DISTO-Analyse

3.2.1 Extraktion einzelner Prozeßschritte

Der Track-Fit selbst ist iterativ, man muß Startparameter $\vec{p}_0$ wählen; das Verfahren liefert dann in jedem Iterationsschritt Korrekturen $\Delta\vec{p}$. Ziel ist es, den Wert

$$\chi^2 = \sum_j \left(\frac{x_j^m - x_j(\vec{p})}{\sigma_j} \right)^2 \tag{3.1}$$

zu minimieren[2]. Dazu dient das »Quasi-Newton-Verfahren«, für das neben der Funktion $\vec{x}(\vec{p})$ auch die ersten Ableitungen in allen Richtungen benötigt werden. Die Korrekturen werden danach durch Aufstellen und Lösen eines 5×5-Gleichungssystem berechnet:

$$\hat{A} \cdot \Delta\vec{p} + \vec{b} = 0 \qquad \text{mit} \qquad a_{ij} = \sum_k \tilde{\sigma}_k \frac{\partial x_k}{\partial p_i} \frac{\partial x_k}{\partial p_j} \qquad \text{und} \qquad b_i = \frac{\partial \chi^2}{\partial p_i} \tag{3.2}$$

Mit $\tilde{\sigma}_k = 2 \cdot \sigma_k^{-2}$ geht die Meßgenauigkeit σ_k der Koordinate k ein. Da die Funktion $\vec{x}(\vec{p})$ in einer Tabelle gespeichert wird, können die benötigten Ableitungen aus den Differenzen in der Tabelle berechnet werden. Die Verwendung von »Tabelleneinheiten« mit abschließender Umrechnung spart bei jeder Ableitung (=32 mal!) eine Division[3]. Die Iteration wird abgebrochen, wenn die Korrektur Δp hinreichend klein ist.

Der Fittingprozess soll nun in Unterprozesse unterteilt werden[4]. Jeder Unterprozess soll dabei nur eine kleine Aufgabe wahrnehmen.

[2] Der Summenparameter läuft nicht immer über alle Koordinaten, da Ineffizienzen (»Draht hat nicht gefeuert«) berücksichtigt werden müssen. In einem solchen Fall muß diese Koordinate übersprungen werden

[3] Divisionen sind rechenintensiver als Multiplikationen. Sie sind daher - wenn möglich - zu umgehen.

[4] Dies soll geschehen um in Hinblick auf das spätere Hardware-Konzept die Parallelisierungs- oder Pipelining-Möglichkeiten zu ermitteln. Ersteres besteht aus unabhängig arbeitenden Teilalgorithmen, welche den gleichen Datensatz bearbeiten, während dies bei letzterem aufeinanderfolgend geschieht und dadurch mehrere Datensätze in dieser »Pipeline« stecken. Jedes Parallel-Konzept kann in ein Pipeline-Konzept umgewandelt werden.

Insgesamt werden folgende Unterprozesse vorgestellt:

Koeffizientenprozeß Dient zur Vorbereitung der Interpolation, Berechnung der betreffenden Tabellenzelle und der »Mischungskoeffizienten«

Interpolation Zur Berechnung der x_j mit $j = 1 \ldots 12$. Dieser Vorgang wird also 12mal ausgeführt, hier verbirgt sich die meiste Rechenzeit!

Quasi-Newton-Verfahren Aufstellen der Gleichung 3.2

Auflösen Lösen des Gleichungssystems

3.2.2 Interpolation in zwei Dimensionen

Da man sich eine fünfdimensionale Tabelle schlecht vorstellen kann, soll der Vorgang der Interpolation zunächst an zwei Dimensionen verdeutlicht werden. Man kann sich eine Funktion $x_j(p_1, p_2)$ als ein Gebirge über der p_1, p_2-Ebene vorstellen. Durch Diskretisierung in einer Tabelle wird das Gebirge kantig, die Kanten werden durch die Einträge der Tabelle »gestützt«, sie sollen daher im folgenden »Stützstellen« genannt werden (vgl. Abb. 3.2). Zu jedem Parameter $\vec{p}$ gibt es daher vier umliegene Stützstellen, die hier mit $\vec{p}^{(0,0)}$, $\vec{p}^{(1,0)}$, $\vec{p}^{(0,1)}$ und $\vec{p}^{(1,1)}$ bezeichnet werden.

Für die Angabe der Position der zu suchenden »Höhe« in dieser quadratischen Fläche benötigt man nun zwei Parameter: $n_0^{(1)}$ geht von der Stelle $(0,0)$ nach $(0,1)$ und $n_1^{(1)}$ geht nach $(1,0)$. Es gilt:

$$n_1^{(1)} + n_1^{(0)} = 1 \tag{3.3}$$

Die entstehende Formel hat 3 Additionen und 6 Multiplikationen:

$$\begin{aligned} x_j(\vec{p}) &= n_1^{(0)}[n_0^{(0)} x_j(\vec{p}^{(0,0)}) + n_0^{(1)} x_j \vec{p}^{(0,1)})] \\ &+ n_1^{(1)}[n_0^{(0)} x_j(\vec{p}^{(1,0)} + n_0^{(1)} x_j(\vec{p}^{(1,1)})] \end{aligned} \tag{3.4}$$

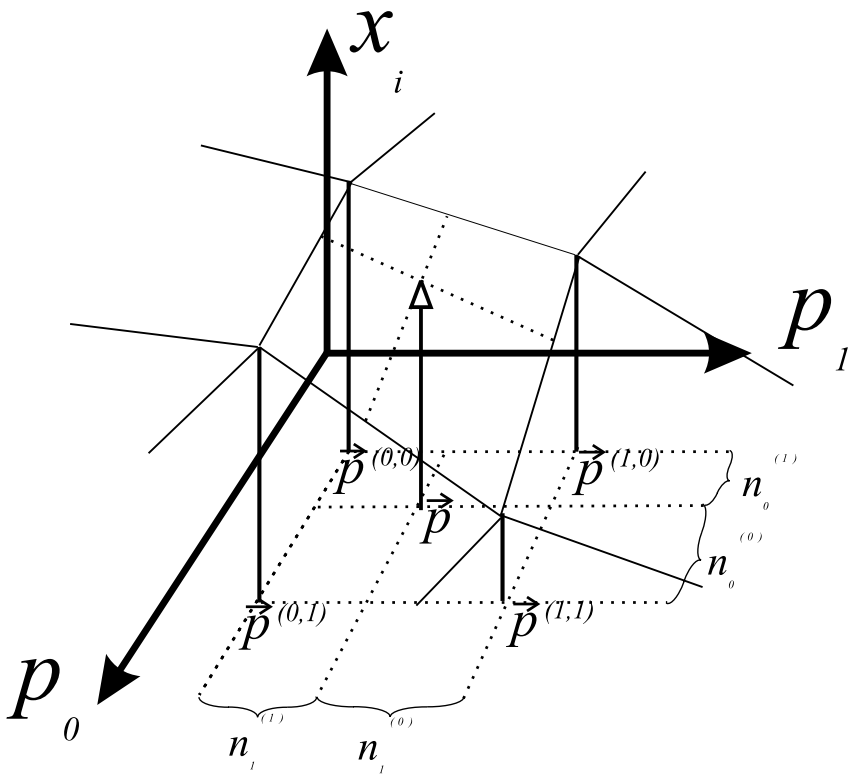
Wird dies ausmultipliziert, hat man mit 8 Multiplikationen und 4 Additionen

$$\begin{aligned} x_j &= n_1^{(0)} n_0^{(0)} \cdot x_j(\vec{p}^{(0,0)}) \\ &+ n_1^{(0)} n_0^{(1)} \cdot x_j(\vec{p}^{(0,1)}) \\ &+ n_1^{(1)} n_0^{(0)} \cdot x_j(\vec{p}^{(1,0)}) \\ &+ n_1^{(1)} n_0^{(1)} \cdot x_j(\vec{p}^{(1,1)}) \end{aligned} \tag{3.5}$$

eigentlich mehr, da für jede Koordinate x_i die selben Koeffizienten verwendet werden, sollte dies Grundlage des Designkonzepts sein. Die Koeffizienten lauten also:

$$n^{(b_0,b_1)} = n_1^{(b_1)} n_0^{(b_0)} \tag{3.6}$$

Abbildung 3.2: Der zweidimensionale Fall



Zur Verdeutlichung der Interpolation im zweidimensionalen Fall. Jede Koordinate ist ein »kantiges Gebirge« über der Fläche der Parameter. Die Position der Interpolationsstelle (Pfeil nach oben) wird durch Richtungsparameter n_i beschrieben.

Die Interpolation selbst reduziert sich dann auf eine einfache Summe:

$$x_j = \sum_{b_0,b_1=0}^{1} n^{(b_0,b_1)} x_j(\vec{p}^{(b_0,b_1)}) \tag{3.7}$$

3.2.3 Fünfdimensionaler Fall

Nun wird dieses Konzept auf fünfdimensionale Tabellen übertragen:

Koeffizientenprozess

$$n(b) = n_4^{(b_4)} n_3^{(b_3)} n_2^{(b_2)} n_1^{(b_1)} n_0^{(b_0)} \tag{3.8}$$

Die Nomenklatur der Richtungsparameter wurde beibehalten, $n_2^{(1)}$ ist der Anteil der Strecke von der Stelle $\vec{p}^{(0,0,0,0,0)}$ zu $\vec{p}^{(0,0,1,0,0)}$). Da diese Berechnung 32 mal gemacht werden muß, hat man insgesamt 128 Multiplikationen.

Interpolationsprozess

$$x_j(\vec{p}) = \sum_{b_0,..,b_4=0}^{1} n^{(b_0,..,b_4)} \cdot x_j(\vec{p}^{(b_0,..,b_4)}) \tag{3.9}$$

Hier hat man 32 Multiplikationen und 32 Additionen. Die 32 Tabellenzugriffe, die in sich in den Stützstellen $n(\vec{p}^{(b_0,..,b_4)})$ verbergen, sind nicht aufeinanderfolgend, sondern decken wegen der fünf Dimensionen einen sehr großen Bereich ab[5].

Ableitungen

Für das Quasi-Newton-Verfahren werden ebenfalls die Ableitungen benötigt. Diese sind an den Stützstellen bekannt - es handelt sich um die Differenz der »Höhenpunkte«[6]. Für die Berechnung der Ableitungen sind mehrere Verfahren denkbar. Im einfachsten Falle werden die Ableitungen an den Stützstellen als Näherungen akzeptiert; dies ist möglich, wenn die Tabelle keine starken Sprünge aufweist. Es sollte für jede denkbare Variante sichergestellt werden, daß das Newton-Verfahren konvergiert. Im genausten Falle werden mit den Ableitungen an den Stützstellen und den schon berechneten Koeffizienten fünf *weitere* Interpolationen durchgeführt, was die Rechenzeit gegenüber den Tabellenzugriffen um den Faktor sechs erhöht.

[5] Im Falle der hier zugrundeliegenden Struktur der look-up-Tabelle springen die Zugriffe über einen zwei MegaByte großen Bereich der etwa doppelt so großen Tabelle.

[6] Ohne Division, da Tabellenkoordinaten gewählt wurden.

Es sollte jedoch nicht unerwähnt bleiben, daß es auch noch weitere Verfahren gibt, die ohne Ableitungen auskommen, die sog. »Schrittmethoden«, wie die Rosenbrock-Methode oder die Simplex-Methode [9].

3.2.4 Lösen des Gleichungssystems

Die so berechneten Funktionswerte und die Ableitungen in fünf Richtungen werden dann einem weiteren Prozess übergeben, welcher die Matrix $\hat{A}$, den Vektor $\vec{b}$ erstellt und das χ^2 berechnet. Ein abschließender Prozeß löst das Gleichungssystem dann, indem er es z.B. auf Dreiecksgestalt bringt. An dieser Stelle lassen sich Divisionen nicht vermeiden. Die gewonnenen Korrekturen $\Delta\vec{p}$ können benutzt werden, um den Startvektor $\vec{p}$ den realen Werten für die Spur $\vec{p}^m$ näherzubringen. Ist der Vektorbetrag $|\Delta\vec{p}|$ unterhalb einer Schwelle, so wird der Fit als hinreichend konvergent betrachtet.

Kapitel 4

Erstellung eines Hardware-Konzeptes

Bei der Analyse der DISTO-Daten, die durch das in Kapitel 3 beschriebene Verfahren auf »Linux-Rechnern« umgesetzt wurde, hat sich gezeigt, daß dafür eine große Rechenleistung benötigt wurde[1]. Es wird daher ein alternatives Konzept vorgestellt, welches auf »Digitalen Signal-Prozessoren« (DSPs) basiert. Dieses wurde mit Hilfe einer »PCI-Steckkarte« realisiert, auf der Teile des Tracking-Algorithmus ausgelagert wurden, um sie dort beschleunigt auszuführen.

4.1 Grundgedanke und Aufgabenstellung

4.1.1 Das existierende Analyseverfahren

Bisher wurden die numerischen Aufgaben zur Analye der DISTO-Daten durch ein von einem »Compiler« (wie z.B. **C** oder **C++**) übersetzten Programm auf einem Rechner implementiert. Neuerdings ist hier eine Ablösung der UNIX-Workstations durch 80x86-kompatible Personal Computern (PCs) zu beobachten, auf welchen Linux installiert ist[2].

Ein solcher PC besteht aus einer Hauptplatine, die einen Prozessor - heutzutage ein K6 der Firma AMD oder ein Pentium II von Intel - einen Kontroller zur Ansteuerung der Masserspeichergeräte, einen langsamen Arbeitsspeicher, sowie einen schnellen Zwischenspeicher (»second level cache«) beherbergen. Der

[1] Ein Pentium II 300 MHz analysiert in einer Stunde ca. 100 MegaByte DISTO-Rohdaten. Die Menge der auszuwertenden Rohdaten wird hingegen auf 600 GigaByte geschätzt [8].

[2] Das Betriebssystem Linux untersteht der GNU-Lizenz für freie Software. Seine Distributionen sind sehr preiswert und beinhalten diverse Compiler.

Arbeitsspeicher besteht bei dem heutigen Standard aus SDRAM (**S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory), der Zwischenspeicher aus schnellem SRAM (**S**ynchronous **R**andom **A**ccess **M**emory). Des weiteren besitzen sie mehrere PCI-Schnittstellen, in welche beliebige Karten gesteckt werden können, wie Grafikkarte, Netzwerkkarte, weitere Controller etc. Der PCI-Standard (PCI=**P**eripheral **C**omponents **I**nterconnect) sieht einen 32-Bit breiten Bus vor, der sowohl die Daten, als auch die Adressen transportiert (»gemultiplext«), er wird mit 33MHz getaktet.

Der orginale x86-Befehlssatz zeichnet sich durch eine Vielfalt an Befehlen aus (»CISC-Architektur«, **C**omplex **I**nstruction **S**et **C**omputer). Diese Komplexität hat sich bis heute noch verstärkt. Die Prozessoren der sechsten Generation (Pentium Pro, Pentium II, K6) beinhalten daher eine aufwenige Vorhersagetechnik, die den sequentiellen Programmablauf parallelisiert. Die interne Abarbeitung der Befehle ist für den Anwender nicht vorhersagbar, eine Geschwindigkeitsoptimierung von Programmen erfolgt teilweise nach der »Try and Error-Methode«. Ein weiterer Flaschenhals ist der langsame Arbeitsspeicher[3]. Erfolgt ein Zugriff auf diesen, lädt der Cache-Kontroller ganze Blöcke in den schnellen second-level-cache, was für normale Arbeitsprogramme, bei welchen weitere Zugriffe meist aufeinanderfolgend sind, zu einer Geschwindigkeitssteigerung führt. Bei ständig über große Adressbereiche[4] springenden Zugriffen - wie es z.B. auch die Arbeit mit der look-up-Tabelle wäre - arbeitet der Cache eher kontraproduktiv, da ein erneuter Zugriff häufig nicht mehr im gerade geladenen Block liegt.

Fazit: Ein PC ist zwar einfach zu beschaffen, er bietet jedoch nicht die ideale Umgebung für numerische Verfahren, die viele sprunghafte Speicherzugriffe benötigen und parallel arbeiten.

4.1.2 Die Tracking-Karte

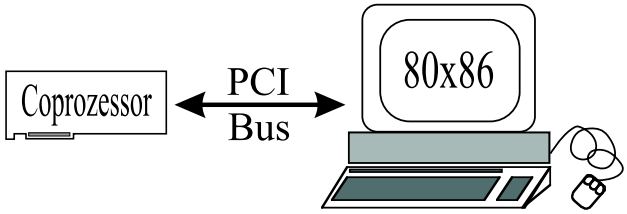
Ein signifikant Rechengewinn würde sich bieten, wenn man alle rechenintensiven Operationen auf eine PCI-Karte auslagern würde (Abb. 4.1). Diese Karte soll als »Coprozessor« arbeiten und den Hauptprozessor wesentlich entlasten. Über den PCI-Bus können große Datenmengen transportiert werden (bis 132 MegaByte pro Sekunde). Ferner soll die Karte die gesamte look-up-Tabelle enthalten, um den Hauptspeicher nicht zu belasten.

Bei der Entwicklung einer solchen PCI-Karte sind die Vorgaben des Algorith-

[3]Der heutige Standard des SDRAM ist das »100MHz-Modul«. Allerdings beziehen sich die 100MHz nur auf aufeinanderfolgende Zugriffe, ein Startzugriff benötigt vier Zyklen, bis die Daten gültig sind. Daher erfolgen Zugriffe, die nicht in Reihenfolge liegen, nur mit 25MHz [16].

[4]Wie schon in Abschnitt 3.2.3 erwähnt, springt der DISTO-Algorithmus über zwei MegaByte, während der Cache typischerweise 512 KiloByte groß ist.

Abbildung 4.1: Idee der Analyse mit der neuen Karte



Die grundlegende Idee des Tracking-Analyse: Ein Hauptprogramm auf einem PC bedient sich eines Coprozessors, welcher auf einer PCI-Karte untergebracht ist

mus zur DISTO-Datenanalyse in Kaiptel 3 genau zu betrachten. Daraus ergibt sich das numerische Grundgerüst für eine Tracking-Hardware. Für diese Aufgaben ist nach einer geeigneten Prozessorenfamilie zu suchen. Des weiteren nach einem Interface gesucht werden, welches die Kommunikation der Komponenten auf der Karte mit dem PCI-Bus übernimmt. Zudem muß geprüft werden, welche Art von Speicher am besten zur Speicherung der look-up-Tabelle geeignet ist.

4.1.3 Track-Fitting auf der Karte

Das Track-Fitting bietet ideale Möglichkeiten zur Auslagerung auf eine PCI-Karte:

- Die gemessenen Koordinaten und die Fit-Parameter können über den PCI-Bus transportiert werden.
- Auf der Karte müssen nur einfache Formeln implementiert werden.
- Durch möglichen parallelen Zugriff auf die look-up-Tabelle kann der Prozess der Interpolation beschleunigt werden.
- Da das Verfahren iterativ ist, verbleiben die Daten auf der Karte, bis das Abbruchkriterium erfüllt ist und $\vec{p}$ und χ^2 über den PCI-Bus dem Hauptprogramm zur weiteren Verarbeitung übergeben werden können. Die bearbeiteten Datensätze können dann durch neue ersetzt werden.

Durch diese Eigenschaften kann eine PCI-Karte zur Implementierung der Track-Fit-Routine problemlos in die Analyse integriert werden. Hier muß die bisher verwendete Tracking-Routine durch eine Software-Schnittstelle zur Kommunikation mit der Karte ersetzt werden.

4.1.4 Überlegungen für einen Prozessortypus

Zusammenfassend ergeben sich folgende Anforderungen für einen Prozessortypus, damit er die obigen Analysen mit hoher Geschwindigkeit durchführen kann:

- Ein 32 Bit breiter Busanschluß des Prozessors ist von Vorteil, da dies der PCI-Busbreite entspricht.
- Die Anzahl der Prozessoren sollte skalierbar sein.
- Der Transport des Datensatzes sollte *nicht* über einen gemeinsamen Bus erfolgen, da damit die Geschwindigkeit mit der Anzahl der Prozessoren stark verringert würde. Er sollte daher über geeignete Schnittstellen verfügen.
- Sein Speicher sollte erweiterbar sein, um auch größere look-up-Tabellen aufnehmen zu können. Der Zugriff auf den eigenen Speicher sollte ebenfalls *nicht* über den gemeinsamen Bus erfolgen.
- Um die Verarbeitung der Daten exakt planen zu können, sollte auf einfache Programmierbarkeit geachtet werden. Ein Betriebssystem ist nicht nötig.
- Er muß numerisch leistungsfähig sein. Hier ist insbesondere auf schnelle Vektor- und Matrixoperationen im Fließkommabereich zu achten.

Diese Eigenschaften werden von **D**igitalen **S**ignal**P**rozessoren (DSPs) erfüllt. Mit der zunehmenden Verbreitung digitaler Signale mußte auch die Verarbeitung dieser Signale immer wieder verbessert werden. Obgleich die grundlegenden Elemente der Prozessverarbeitung nur Multiplizieren, Addieren, Arrayhandhabung und Zwischenspeicherung beinhalten, bieten die DSPs der heutigen Generation eine Vielzahl von Möglichkeiten, wie Adressgenerierung, paralleles Addieren und Subtrahieren, gleichzeitige Speicherzugriffe und vieles mehr.

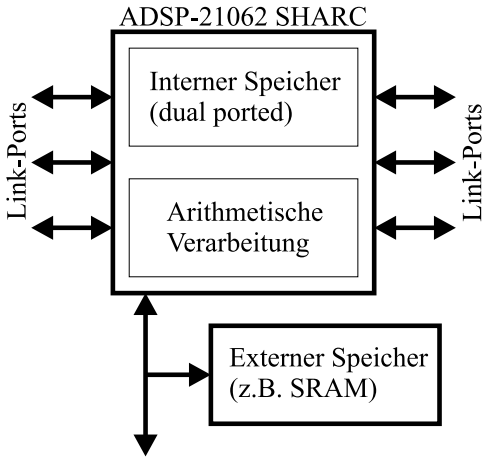
DSPs dienen der schnellen Verarbeitung von Bild- und Tonsignalen in der Multimediatechnik, der Medizin und bei Grafiksystemen. Auf Soundkarten und 3-D-Beschleunigern für PC-Spiele finden sie ihren Weg mittlerweile genauso in den Heimanwenderbereich wie in Faxgeräten und Telefonen. Da die Multimediabranche sehr stark expandiert, wird auch die Entwicklung der DSPs in den nächsten Jahren verstärkt weitergeführt werden.

4.2 Umsetzung des Konzeptes

4.2.1 Übersicht und Komponenten

Aus den im vorigen Abschnitt diskutierten Gesichtspunkten wurde folgendes Konzept erarbeitet (Abb. 4.3). Kern der Karte sind sechs DSPs des Typs ADSP-21062

Abbildung 4.2: Grobe Darstellung der ADSP21062 SHARC-Funktionen



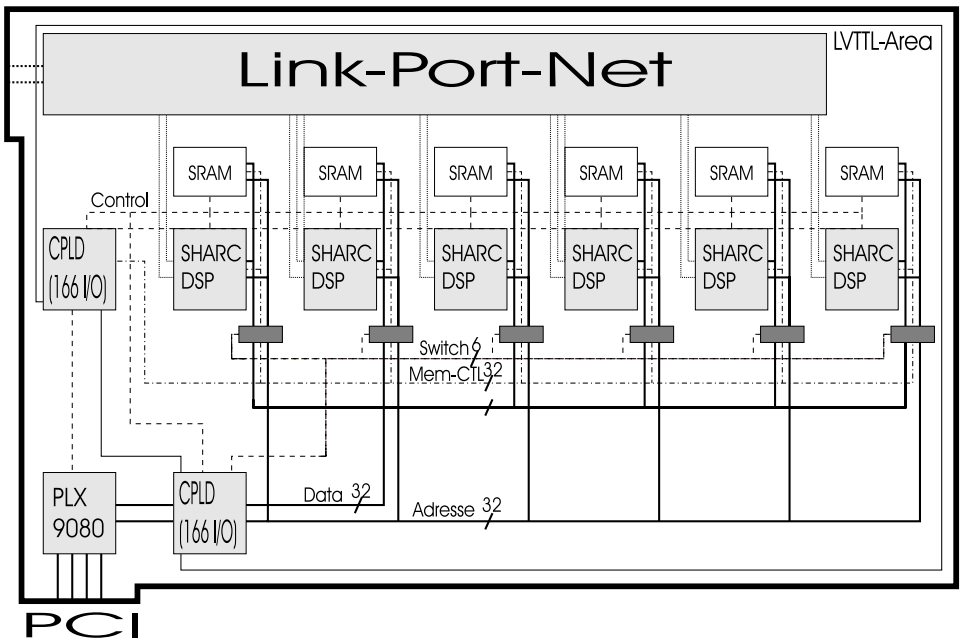
Der ADSP21062 SHARC von Analog Devices besteht aus internem Speicher und einer arithmetischen Einheit, die von einem Programmablauf gesteuert wird. Die Weitergabe von Datenpaketen kann über sechs jeweils unabhängig arbeitende »Link-Ports« mit einer Rate von jeweils 40 MegaByte/s erfolgen. Über einen Bus kann weiterer Speicher angeschlossen oder auf den internen Speicher zugegriffen werden.

SHARC von Analog Devices (SHARC= **S**uper **H**arvard **Ar**chitecture **C**omputer, siehe auch Abb. 4.2). Der Transport des Datensatzes erfolgt über »Link-Ports«, die Speicherung der look-up-Tabelle erfolgt über den »externen Port« der SHARCs in 16x256K SRAM. Da jeder SHARC nur über einen »externen Port« verfügt, aber ein globaler Bus benötigt wird, wurden »CMOS-Busswitche« eingesetzt (CMOS=**C**omplementary **M**etal-**O**xide **S**emiconductor), welche den lokalen SRAM-SHARC-Bus wie ein Schalter vom globalen Bus trennen können. Dadurch wird jede Baugruppe SHARC-SRAM abgeschottet (siehe Abschnitt 4.1.4). Zur Ansteuerung und Kontrolle des Boards dienen CPLDs (**C**omplex **P**rogrammable **L**ogic **D**evice) der Firma Xilinx, die Kontrolle des PCI-Busses wird durch den Baustein PCI9080 der Firma PLX übernommen. Im Folgenden sollen die verwendeten Komponenten näher beschrieben werden, besonderes Augenmerk wird dabei auf die SHARCs gelegt.

4.2.2 Der PCI-Bus

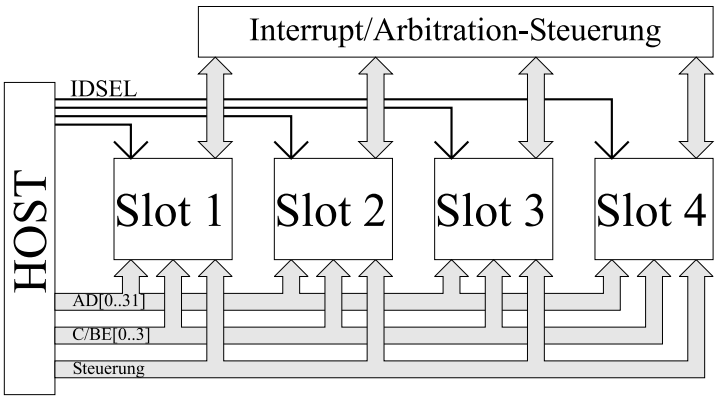
Da nicht nur jedes heuzutage produzierte PC-Mainboard mehrere PCI-Steckplätze (sog. »Slots«) beherbergt, sondern auch vermehrt PowerPCs (Macintosh) oder Alpha-Workstations (Digital), ist die Verwendung dieses Busses für eine Einsteckkarte die einzig sinnvolle Variante für schnelle Übertragungen. Nachstehende Abschnitte dienen einer Grundinformation über dieses Bussystem, im wesentlichen nach [10]. Die Beachtung der kompletten Spezifikation [11] war für dieses Projekt nicht erforderlich, da die Interfaceaufgaben von einem fertigen Produkt (siehe

Abbildung 4.3: Der Aufbau der Tracking-PCI-Karte



Hauptgerüst des Konzeptes sind sechs identische Baugruppen, jeweils bestehend aus einem ADSP-21062 SHARC, ausreichend schnellem Speicher (SRAM) und - zur Abschottung gegen einen globalen Bus - Bus-Switche, die hier als unbeschrifteter grauen Balken über den Bussen dargestellt werden. Der Transport der Datensätze läuft über ein »Link-Port-Net«, welches später erläutert wird (siehe Abschnitt 4.10). Jede der Baugruppen ist mit seinen Kontrolleitungen an einem »Kontroll-CPLD« angeschlossen. Die globalen Adress- und Datenleitungen sind mit einem »Register-CPLD« verbunden, welcher seine Daten und Adressen über einen speziellen Chip (PCI9080 von PLX) über den PCI-Bus erhält.
*Der größte Teil der Bauteile wird mit 3,3V betrieben (LVTTL = **L**ow Le**v**el **T**ransistor to **T**ransistor **L**ogic).*

Abbildung 4.4: Übersicht PCI-System



Ein PCI-System besteht in der Regel aus vier Geräten, die eingesteckt werden (»Slots«). Zu diesen Slots führen gemeinsame Adress- Daten- und Kontrolleitungen. Mit der IDSEL-Leitung wählt der Host die Karten während der Konfiguration aus. Im normalen Busbetrieb weist eine Interrupt- und Arbitrationssteuerung den Geräten den Bus zu.

4.2.3) wahrgenommen wurden und für diese Zwecke Übersichtsliteratur ausreichte [12].

Aufbau

Der PCI-Bus ist ein 32 Bit breiter, nicht-terminierter Bus, der mit maximal 33 MHz und 5V-Signalen betrieben wird. Spezifikationen für 64 Bit und 66 MHz, bzw. für 3,3V-Signale sind vorhanden, werden aber von den Herstellern im Augenblick nicht genutzt. Die maximale Leistungsaufnahme über die Spannungsversorgung (Erde, 3,3V, 5V, $\pm$12V) ist pro Karte auf 25W beschränkt. Einen Überblick über den Aufbau liefert Abb. 4.4. In der Regel hat ein PCI-Host-System vier Slots. Jedes Gerät hat somit Zugriff auf die Busleitungen (AD), die Befehlsleitungen (C/BE), diverse Steuerleitungen und - für jedes Gerät extra - Interruptleitungen und ein Arbitrationsprotokoll. Letzteres ist nur für Geräte wichtig, die selbst den Bus kontrollieren können, um aktiv Daten zu transportieren (»Master-Funktion«, im Gegensatz zur interruptabhängigen »Slave-Funktion«).

Konfigurationsphase

Kurz nach dem Einschalten des Rechners (»Boot-Vorgang«) liest das PCI-Host-System die Register aus, die jedes PCI-Gerät haben muß. In diesen sind neben diversen Modieinstellungen auch die Hersteller- und Produktnummer untergebracht, außerdem die gewünschten Rechnerressourcen (Speicher und I/O-Bereiche). Die Auswahl der Karten während der Initialisierungsphase geschieht mit der IDSEL-Leitung (die aus diesem Grunde für jedes Gerät ausgeführt ist). Danach weist der Host in dem möglichen 32-Bit Raum (4 Gigabyte) die gewünschten Speichergrößen so zu, daß es keine Überschneidungen gibt. Nachdem die jeweiligen Adreßräume den Geräten zugewiesen wurden, sind diese für das Erkennen eines Zugriffs auf ihren Bereich verantwortlich, die IDSEL-Leitung bleibt im normalen Busbetrieb inaktiv. Die Karte muß weiterhin dafür sorgen, daß der lokale Adressraum von dem PCI-Bus abgekoppelt bleibt (sog. »um-mappen«).

Busbetrieb

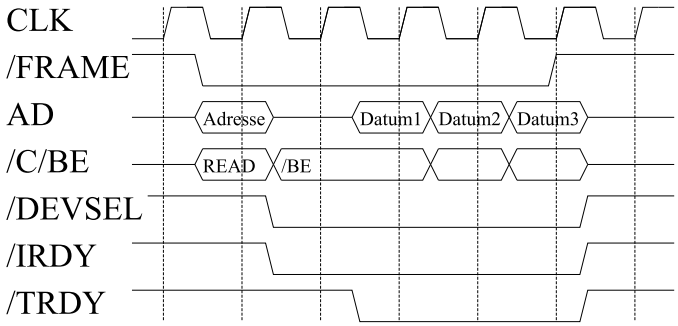
Die 32 Bit des PCI-Busses transportieren sowohl Adressen, als auch Daten (»Multiplexen«). Der durch die erforderliche Aufteilung in Daten- und Adressphasen verursachte Geschwindigkeitsverlust wird durch sogenannte »Bursts« teilweise wieder ausgeglichen. Dabei werden die Daten ohne weitere Adressen übermittelt, dies ist jedoch nur möglich, wenn es sich um aufeinanderfolgende Adressen handelt. Diese Arbeitsweise soll anhand eines Lesevorgangs (siehe Abb. 4.5) verdeutlicht werden. Ein Schreibvorgang würde analog aussehen.

Zunächst legt der Master die gewünschte Startadresse an und übermittelt über die 4 /C/BE-Leitungen »Lesen« (Read)[5]. Den richtig erkannten Zugriff auf sein zugewiesenen Speicherbereich bestätigt der Slave mit dem Herunterziehen der /DEVSEL-Leitung. Mit dem Ziehen der /IRDY-Leitung (**I**nitiator **R**ea**dy**) teilt der Master mit, daß er bereit ist, Daten zu empfangen. Mit /TRDY (**T**arget **R**ea**dy**) leitet der Slave die Datenübermittlung ein. Mit dem Zurücksetzen von /FRAME leitet der Master das Ende der Übermittlung ein. Die /C/BE-Leitungen dienen zunächst als Kommando-Bits (hier: »READ«, also Lesen); während der Datenübermittlung schalten sie die Bytes des 32-Bit-Datenbusses ein- oder aus (/BE=**B**yte **E**nable).

Dies ist jedoch nur ein Beispiel der komplexen Möglichkeiten des PCI-Busses. Zudem müssen die Signale sowohl strenge Zeit- wie kapazitive Kriterien erfüllen, so daß ein Design einer Schnittstellenlogik (z.B. in einem CPLD) mit sehr viel Arbeit verbunden ist. Es gibt jedoch schon einige fertige und preiswerte Lösungen, die eine PCI-Schnittstelle in einem Chip beherbergen. Für diese Arbeit wurde ein

[5]Mit dem Zeichen »/« (Slash) wird bei jedem Signal angedeutet, daß die gewünschte Funktion beim Herunterziehen von 5V auf 0V aktiv wird (»low-active«).

Abbildung 4.5: PCI-Lese-Burst



Das Verhalten grundlegender Steuerleitungen am Beispiel eines Lesezugriffs. Eine markante Eigenschaft des PCI-Busses ist das »Bursten«, also das einmalige Übertragen einer *Adresse und das mehrfache Übertragen* aufeinanderfolgender *Daten. Zum Verständnis der Steuerleitungen siehe Fließtext.*

Produkt der Firma PLX verwendet.

4.2.3 Der PCI9080 von PLX

PCI9080 erfüllt alle Funktionen, die der PCI-Bus benötigt. Die internen Register, die hauptsächlich während der Konfigurationsphase benutzt werden, können dabei über ein EEPROM (**E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory) beschrieben werden, so daß jede gewünschte Speichergröße angemeldet werden kann. Der gemultiplexte PCI-Bus wird auf einen lokalen Bus gemapped, so daß die Basisadresse der Speicherbereiche beliebig gewählt werden können. Dies ist insbesondere für den SHARC wichtig, da dieser die Speicherkonfiguration vorschreibt.

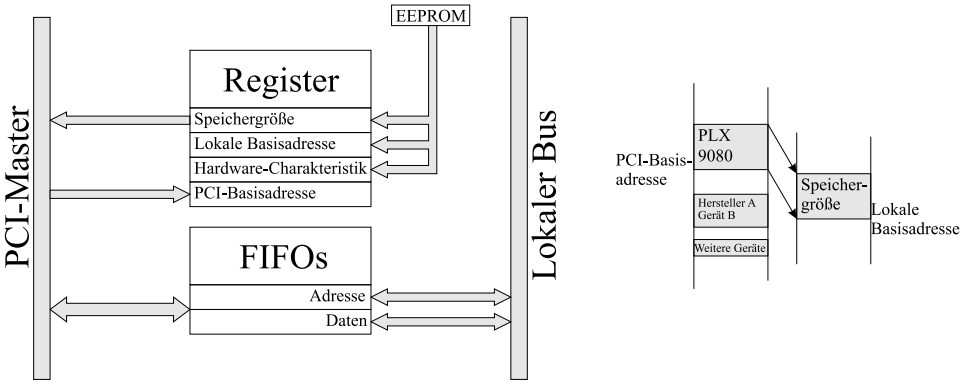
Der Anschluß des PCI9080 an den lokalen Bus erfolgt in dem hier gewählten Modus über einen 30 Bit breiten Adressbus und einen 32 Bit breiten Datenbus. Für die Anbindung an einen lokalen Prozessor sind eine Vielzahl von Steuerleitungen vorgesehen, von welchen hier einige zum späteren Verständnis kurz vorgestellt werden, eine detallierte Beschreibung findet sich unter [13]:

LA 30 Bit breiter Adreßbus. Die untersten zwei Bits sind bei einem 32-Bit-Zugriff nicht nötig und fehlen deshalb.

LD 32 Bit breiter Datenbus, Zugriffe mit 16 oder 8 Bit sind möglich.

/ADS Address-Strobe, zeigt eine gültige Adresse an.

Abbildung 4.6: Wesentliche Funktionen des PCI9080



Der PCI9080 von PLX beinhaltet FIFOs zum Puffern und Register, welche durch ein EEPROM beschrieben werden, z.B lokale Basisadresse und Speichergröße. Die PCI-Adresse wird vom PCI-Bus-Master anhand der Speichergöße ermittelt und bei der Konfiguration in ein Register geschrieben. Dieses »Remapping« ist rechts schematisch dargestellt.

/LBE Zeigen je nach gewählter Busbreite die Byte-Enable-Bits oder die untersten Adressbits für einen 16- oder 8-Bit-Zugriff.

LHOLD Wird gesetzt, wenn der PCI9080 den lokalen Bus beansprucht.

LHOLDA Wird von außen gesetzt, um das LHOLD-Signal zu bestätigen und dem PCI9080 den Bus zu übergeben.

READYO Zeigt einen gültigen Datensatz und das Ende des Transfers an.

/LRESETO Liefert ein »RESET«-Signal, um die lokalen Bauteile in einen definierten Anfangszustand zu versetzen.

Um die Vorteile des Burst-Verfahrens des PCI-Busses voll auszuspielen, sind die Datenwege mit »FIFOs« (**F**irst **I**n **F**irst **O**ut) versehen, welche die Daten zwischenspeichern und sie kurz darauf in der gleichen Reihenfolge ausgeben. Ein komplettes Schema der Funktionsweise kann Abb. 4.6 entnommen werden.

4.2.4 Der CPLD von Xilinx

Zur grundlegenden Boardkontrolle und zur Ermöglichung weitreichender Rekonfigurierbarkeit können programmierbare Logik-Bausteine eingesetzt werden, heutzutage häufig verwendete Familien sind die FPGAs und die CPLDs. Die Vor- und

Nachteile beider Familien sind an anderer Stelle beschrieben [17], so daß hier nur noch das konkret verwendete Modell, der XC95216 der Firma Xilinx vorgestellt wird. Diese CPLDs können mit einem speziellen Kabel (»X-Checker«) während des laufenden Betriebs programmiert werden (isp=**i**n-**s**ystem **p**rogrammable) und halten diese Programmierung auch nach Abschalten bis zu 20 Jahre. Eine Rekonfiguration ist bis zu 10000 mal möglich.

Architektur

Ein CPLD der Reihe XC9500 besteht aus Input/Output-Blöcken, die mit den externen Pins des Chips verbunden sind, einer Matrix zur internen Verbindungskonfiguration (»Switch-Matrix«) und mehreren Funktionseinheiten (»Function-Block«), die wiederum 18 »Makrozellen« beinhalten. Eine solche Makrozelle setzt die Eingänge, die von anderen Makrozellen oder der Switch-Matrix kommen, in einer kombinatorischen Einheit um (»product term«), und leitet dies den I/O-Blöcken und der Switch-Matrix zu. Pro Makrozelle ist ein Flip-Flop[6] vorhanden.

4.2.5 Der Bus-Switch von Fairchild

In dem FST16211 von Fairchild befinden sich 24 MOS-Feldeffekttransistoren, die zur bidirektionalen Kontrolle eines Signals dienen können. Im digitalen Sinne dient ein solcher Transistor als schneller Ein- und Ausschalter. Hauptmerkmale des FST16211 sind:

- Nur vier Ω Widerstand zwischen einer geschalteten Bus-Leitung.
- Minimale Zeitverzögerung der Bus-Signale (0,25ns).
- Minimaler Stromverbrauch.

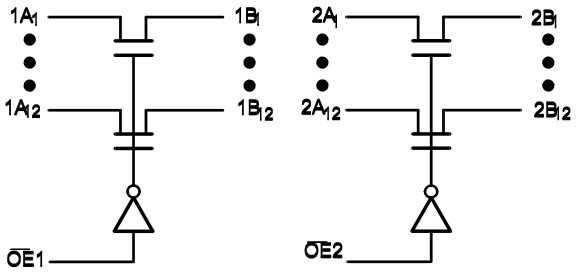
4.2.6 Der SHARC von Analog Devices

Der ADSP-21062 SHARC ist ein Digitaler Signalprozessor mit folgenden Hauptmerkmalen [14],[15]:

- Rechenwerke für 32-bit IEEE-Fließkommazahlen (»Multiplier«, »ALU«, »Shifter«).
- Zwei interne SRAM-Bänke mit jeweils 1MegaBit, »dual-ported«

[6] Ein Flip-Flop hält die Information über den Zustand, er dient also als »Ein-Bit-Speicher«.

Abbildung 4.7: Schaltbild des FST16211 Bus-Switch



- 40 MHz.
- Parallele Ausführung, dadurch bis 120 MFLOPS[7] Spitzenleistung.
- Übergabe des Operanden mit der Instruktion.
- Automatisches Modifizieren von Indices.
- DMA-Transfer bis 240 MB/s über den externen Port (32 Adressleitungen, bis zu 48 Datenleitungen) oder »Link-Ports«, jeweils 6 Stück (2 Steuerleitungen, 4 Datenleitungen, doppelte Taktfrequenz, asynchroner Transfer).
- Eingebaute symmetrische Multiprozessormöglichkeit mit bis zu sechs Prozessoren.

Rechenwerk

Die numerische Rechenleistung wird durch drei unabhängige Einheiten erreicht, die jeweils sowohl Festkomma- wie auch Fließkommaoperationen unterstützen. Die Einheiten bedienen sich dabei 16 Registern $Rx, x = 0 \ldots 15$ oder Fx[8].

Shifter Arbeitet nur mit Festkommazahlen. Er umfasst Verschiebungen, Rotationen und Bit-Operationen.

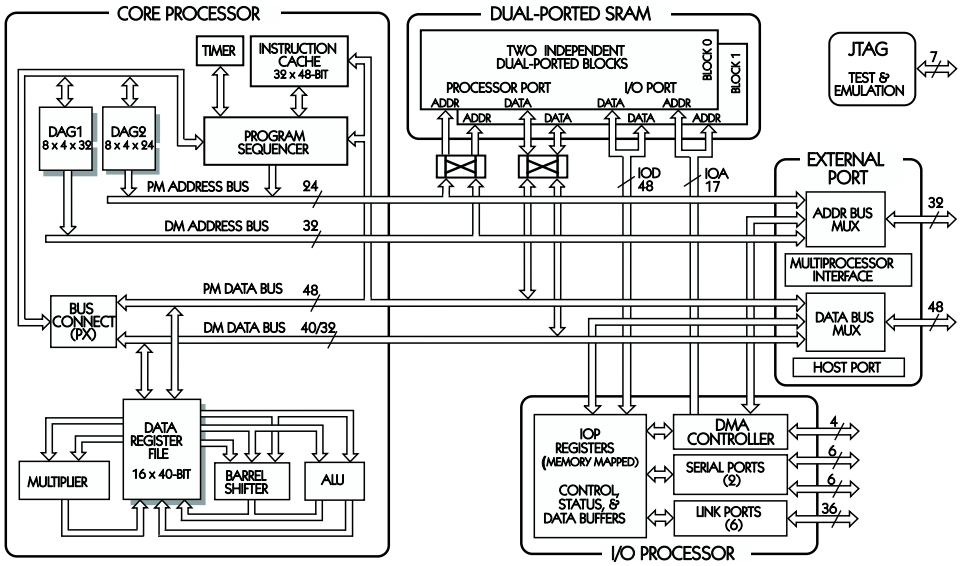
Arithmetic/Logic Unit (ALU) Diese Einheit umfasst zwei Grundfunktionen: Strichrechnungen wie

$$Fm = Fx + Fy \qquad \text{bzw.} \qquad Fm = Fx - Fy \tag{4.1}$$

[7] Die Einheit »MFLOPS« gibt an, wieviel Millionen Fließkommaoperationen pro Sekunde durchgeführt werden.

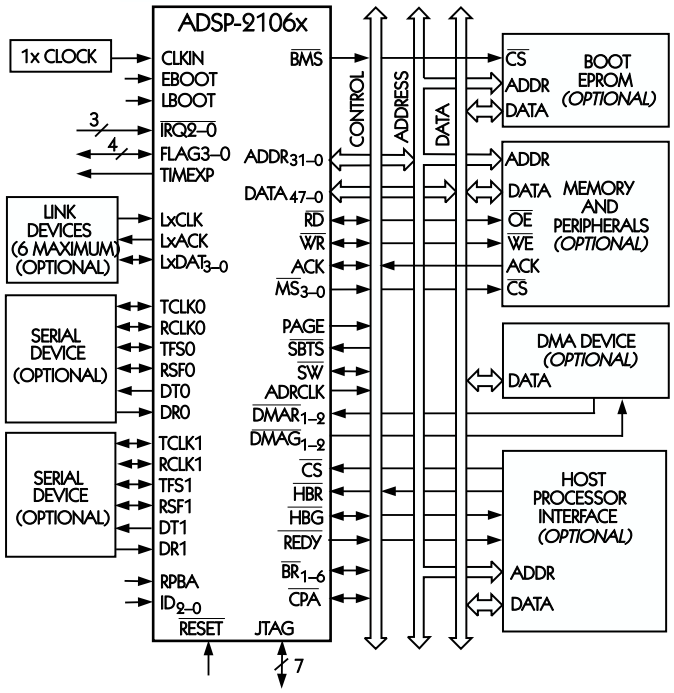
[8] Im Falle von Festkommazahlen ein »Rx«, für Fließkommazahlen ein »Fx«. Es handelt sich jedoch in Wirklichkeit um den gleichen Registersatz

Abbildung 4.8: Blockdiagramm ADSP21062 SHARC



*Der interne Aufbau des 2106x (entn. [15]) folgt der »Super-Harvard-Architektur«: Core, SRAM, I/O-Bereich und externer Port sind mit zwei unabhängigen Bussen verbunden, dem PM-Bus (**P**rogram **M**emory) und dem DM-Bus (**D**ata **M**emory). Die Adress-Generierung geschieht jeweils über einen DAG (**D**ata **A**ddress **G**enerator), welcher Aufgaben der indirekten Indizierung und Modifizierung übernimmt. Operanden werden in den 16 Registern des Data-Register-File gespeichert, mit Hilfe des Multipliers, des Shifters und der ALU kann dreifach parallel gerechnet werden.*
Das SRAM ist nicht nur in zwei Blöcke unterteilt, es ist zudem auch noch »dual-ported«, d.h. es können zwei Zugriffe gleichzeitig und unabhängig erfolgen. Die Punkt-zu-Punkt-Kommunikation erfolgt über sechs Link-Ports oder zwei seriellen Ports, der Anschluß an ein Bus erfolgt über des externen Port. Dies ermöglicht einem »Host« den uneingeschränkten Zugriff auf Speicher und Register.

Abbildung 4.9: Ein SHARC-System



Ein Single-Prozessor-System besteht neben dem ADSP21062 SHARC aus einem Host Interface und Memory, beides verbunden mit den externen Adreß- und Datenleitungen und diversen Steuerleitungen. Das Boot-EEPROM ist optional und wird in diesem Fall nicht benutzt. Über die sechs Link-Ports können Daten zu weiteren SHARCs transportiert werden.

oder logische Operationen wie AND, OR, XOR, NOT.

Multiplier Neben diversen Festkommaoperationen gibt es nur eine (aber dafür grundlegende) Fließkommaoperation:

$$Fn = Fx * Fy \tag{4.2}$$

Eine der wesentlichen Stärken des SHARCs liegt in der Kombination von Additionen und Multiplikationen, jedoch beschränkt sich ein solches Vorgehen auf:

$$Fn = F0 \ldots 3 * F4 \ldots 7, Fm = F8 \ldots 11 \pm F12 \ldots 15 \tag{4.3}$$

Programmablauf

Der Programmcode des SHARCs ist einfach zu lesen und orientiert sich am numerischen Verständnis. So soll der Befehl

$$Fn = DM(0x12345678) \tag{4.4}$$

bewirken, daß der Datenbus ein Datenwort von der Adresse 0x12345678 zum Register Fn bringen soll (wo es später von o.g. Rechenwerken bearbeitet werden kann). Die Übergabe der Operanden erfolgt *mit* dem Befehlswort, so daß ein solcher Befehl i.A. in einem Zyklus behandelt werden kann.

Zum Ablauf des Programmes selbst stehen die üblichen Befehle zur Verfügung, wie Schleifen, Sprünge, Unterroutinen. Das Abarbeiten der Befehle erfolgt in einer »Pipeline«, d.h. in mehreren Stufen. In der ersten Stufe wird ein Befehl aus dem Programm-Speicher geladen, in der zweiten dekodiert und in der dritten Stufe ausgeführt. Bei Sprüngen ist darauf zu achten, daß der Befehl zwei Zyklen vor dem eigentlichen Sprung übergeben wird, damit die Pipeline nicht leerläuft (delayed branch).

Adressierung

Die Adressgeneratoren (DAG=**D**ata **A**ddress **G**enerator) erlauben eine indirekte Adressierung des Speichers. Dazu besitzen sie jeweils 8 Indexregister und die gleiche Anzahl an Modify-Registern, mit welchen ein solcher Index entweder vor einem Zugriff (»pre-modify«) oder nach einem Zugriff (»post-modify«) verändert werden kann. Diese Adreßgenerierung ermöglicht sehr schnelle Vektor- und Matrixoperationen, da durch die Automatisierung dieser Prozedur der Programmablauf nicht aufgehalten wird. Diese Eigenschaft wird bei der Aufgabe des Track-Fittings intesiv genutzt.

Zusätzlich stehen noch Register für zirkulare Zugriffe zur Verfügung.

post-modify Mit dem Befehl

$$Fn = DM(Ix, My) \tag{4.5}$$

wird die Adresse diesmal nicht übergeben (vgl. (4.4)), sondern der Inhalt des Registers Ix als Adresse benutzt. Anschließend wird der Wert von Ix um den Wert My erhöht.

pre-modify

$$Fn = DM(My, Ix) \tag{4.6}$$

erhöht zunächst den Index Ix um den Wert My und greift dann auf den Speicher zu.

Analog ist auch der Zugriff auf den Programm-Speicher (PM) möglich.

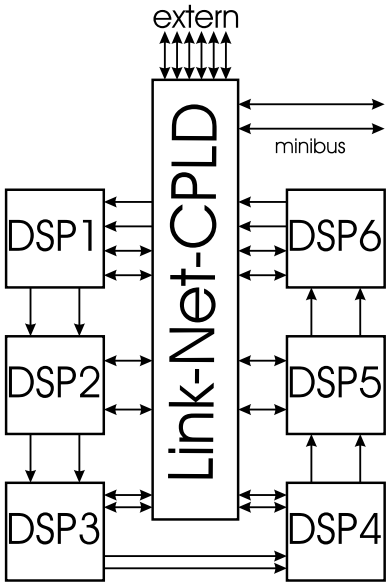
Link-Ports

Die Link-Ports (sechs an der Zahl für jeden SHARC) dienen der Punkt-Zu-Punkt-Verbindung zwischen den SHARCs. Auf diese Weise kann der Datentransport beliebig gestaltet werden, wie z.B. in 1-, 2- oder dreidimensionalen Arrays. Jeder Link-Port besteht aus vier Datenleitungen, einem Takt und einer Kontrolleitung (»Ack-Signal«). Der Takt kann mit doppelter Frequenz arbeiten, so daß pro Zyklus 8 Bit übertragen werden. Auf diese Weise können alle Link-Ports eines SHARCs zusammen eine Transferrate von 240MByte/s erreichen. Dies deckt sich mit den internen DMA-Pfaden (40 MHz $\times$ 48Bit); dadurch wird der Programmablauf durch die Link-Port-Aktivitäten nicht gestört. Die zu übertragende Block-Größe eines DMA-Transfers kann eingestellt werden. Nach erfolgter Übertragung wird ein Interrupt ausgelöst, der das laufende Programm unterbricht und eine vorher vereinbarte Service-Routine startet.

Der Hauptdatenstrang ist zyklisch, so daß vier Link-Ports benutzt wurden, die Daten zum nächsten SHARC weiterzugeben (siehe Abb. 4.10). Lediglich zwischen dem ersten und dem letzen SHARC wurde ein CPLD (»Link-Port-CPLD«) dazwischengeschaltet, der bei Bedarf die Kommunikation mit externen Geräten durchführen kann. Auf diese Weise können auch weitere Tracking-Karten in die Datenpfade eingeklinkt werden. Diese einfache Skalierbarkeit ist ein großer Vorteil dieses Konzeptes.

Die übrigen zwei Link-Ports wurden bei jedem SHARC ebenfalls mit dem Link-Port-CPLD verbunden, um weitere Rekonfigurationsmöglichkeiten zu erhalten. Dies sollte im Hinblick auf alternative Algorithmen sichergestellt werden (siehe Abschnitt 3.2.3). Zwei »Minibusse« (jeweils 6 Bit breit) sind sowohl mit den beiden anderen CPLDs verbunden, als auch mit einem Stecker (nur intern verfügbar), der während der Inbetriebnahme hauptsächlich zu Diagnosezwecken diente.

Abbildung 4.10: Das Link-Port-Netz



Die Verteilung der Link-Ports wird sowohl dem zyklischen Datenfluß gerecht, bietet jedoch auch Konfugurationsmöglichkeiten durch Verwendung eines CPLDs. Hervorzuheben ist der 36 Bit breite externe Stecker, mit dem weitere Geräte angeschlossen werden können, wie auch die beiden Minibusse, die eine Kommunikation mit den beiden anderen CPLDs ermöglichen.

Externer Bus und Host-Interface

Der SHARC besitzt einen 32 Bit breiten Adressbus und einen 48 Bit breiten Datenbus. Über diese Schnittstelle können Speicher, Ein-/Ausgabegeräte sowie ein »Host« betrieben werden. Dieser Host dient dazu, aktiv auf die SHARCs zuzugreifen. Auf diese Weise kann »von außen« der Speicher des SHARCs beschrieben und gelesen werden, die Zugriffe auf alle Register sind möglich, des weiteren können die SHARCs in einer Initalisierungsphase (»Booten«) mit dem nötigen Programmcode versehen werden.

Möchte ein Host den Bus benutzen, so muß er dies dem SHARC zunächst durch Setzen einer Leitung (»/HBR«=**H**ost **B**us **R**equest) mitteilen. Der SHARC beendet zunächst seine laufenden Aktionen und antwortet dann durch Setzen einer anderen Leitung (»/HBG«**H**ost **B**us **G**rant). Nun ist der Host dafür verantwortlich, die Busleitungen und die Steuerleitungen für den Speicher zu treiben, bis er den Bus wieder zurückgibt.

Zudem können bis zu sechs SHARC über diese Busleitungen miteinander verbunden werden. Diese Multiprozessorfähigkeit erfordert neben dem Verbinden weniger Steuerleitungen keinen Aufwand für den Anwender. Jeder SHARC erhält seinen eigenen Speicherraum, so daß ein Datentransport durch einfaches Schreiben in den entsprechenden Speicher erfolgen kann. Das nötige Protokoll wird durch der

Prozessor selbständig ausgeführt. Da diese Option in der vorliegenden Arbeit nicht benutzt wurde, soll hier nicht näher darauf eingegangen werden.

4.2.7 Das SRAM von Samsung

Um die nötigen Daten der »look-up-Tabelle« zu speichern, reicht der interne Speicher des SHARCs (2MBit=256KiloByte) nicht aus. Es ist daher erforderlich, die externen Busleitungen des SHARCs (vgl. Abb. 4.9) mit einem Speichermodul (»Memory«) zu verbinden. Die nötige Datenbreite beträgt für Fließkommazahlen 32 Bit; um Platz auf der Platine zu sparen, sind hier also 16 Bit breite Chips den 8 Bit breiten vorzuziehen. Des weiteren ist SRAM zu verwenden, um die nötige Geschwindigkeit zu gewährleisten, welches mindestens 15ns[9] schnelle Zugriffe erlaubt. Auf diese Weise kann der Zugriff in nur einem Takt erfolgen.

Da sämtliche SHARCs mit 3,3V betrieben werden, müssen auch die SRAM-Chips für diese Spannung geeignet sein (LVTTL). Die heutige maximale Speichergröße für solche Bauteile liegt bei 4MegaBit. Diese werden von einer Vielzahl von Herstellern angeboten, aufgrund guter Lieferbarkeit wurde der KM616V4002B von Samsung gewählt.

[9] Bei 40 MHz beträgt die Zykluszeit 25ns. Das SRAM muß jedoch schneller sein (vgl. [15]).

Kapitel 5

Inbetriebnahme der PCI-Karte

Die PCI-Karte wurde nach den Vorgaben in Kap. 4 entwickelt und in Betrieb genommen. Durch die Aufteilung der Funktionen auf der Karte in Baugruppen, die optimale Ausnutzung der programmierbaren Logik und die gute Ausstattung des Labors des II. Physikalischen Institutes mit Diagnosegeräten konnte ein Teil der Karte nach dem anderen in Betrieb genommen werden, ohne daß große Hindernisse auftraten.

In den folgenden Abschnitten soll erläutert werden, wie die Karte Stück für Stück in Betrieb genommen wurde. Im Abschnitt 5.2 wird gezeigt, wie ein Teil des Algorithmus als Geschwindigkeitstest implementiert wurde.

5.1 Konfiguration und Funktionstests der Schaltung

Die Bestückung der bei ILFA, Hannover hergestellten 8-Lagen-Karte wurde mit den Mitteln des Elektroniklabors des II. Physikalischen Institutes durchgeführt. Als Umgebung für die Inbetriebnahme und die anschließenden Tests diente ein Pentium90 unter Linux.

5.1.1 Der PCI-Bus

Linux-Treiber

Das Betriebssystem Linux - unter welchem die Karte letztlich ihre Funktion erfüllen sollte - gestattet es dem Anwender nicht, direkt auf Hardware-Ressourcen zuzugreifen. Um überhaupt die Kommunikation mit der Karte zu ermöglichen, ist das Schreiben eines »Treiber« notwendig. Dieser dient für den Anwender am PC als definierte Schnittstelle zur Ansteuerung der Karte, insbesondere für den Zugriff

auf ihren Speicherbereich. In diesem Falle wurde ein Treiber für eine andere Karte [19] als Vorbild genommen, um einen eigenen Treiber zu erstellen. Dieser wurde so geschrieben, daß er zur Laufzeit des Betriebssystems jederzeit implementiert werden konnte (sog. »modularer Treiber«). Näheres zum Linux-Treibermodell ist [18] zu entnehmen.

Überwachung der PCI-Signale

Um mögliche Fehlerquellen auf der PCI-Seite des PCI9080 zu finden, stand ein Diagnosesystem - bestehend aus einem »Extenderboard« und einem »Logikanalysator« - zur Verfügung. Das »Extenderboard«[1] wird zwischen den PCI-Slot und der zu untersuchenden Karte gesteckt und enthält Abgriffe der PCI-Leitungen. Diese können mit Hilfe des »Logikanalysators«[2] aufgezeichnet und dargestellt werden. Der Start der Aufzeichnung (sog. »Trigger«) läßt sich dabei vielfältig programmieren.

Konfiguration des PCI9080

Durch Überwachung der PCI-Signale konnte eine optimale Konfiguration des PCI9080 ermittelt werden. Diese erfolgt über ein serielles EEPROM (vgl. Abschnitt A.1), welches neben der Geräte- und Herstellernummer zur Identifikation der Karte weitere wichtige Informationen beinhaltet:

Speicherbedarf Wie in Abb. 4.6 schon erläutert, werden dem PCI-Host beim Booten die gewünschten Speichergrößen von bis zu vier Bereichen übermittelt. In diesem Falle wurde nur ein Speicherbereich von vier MegaByte benutzt. (vgl. Tabelle 5.1).

Optionales ROM Der PCI9080 bietet die Möglichkeit, auf einem ROM ein Programm zu speichern, welches beim Booten des PCs gestartet wird. Diese Funktion wird hier nicht benötigt und wurde abgeschaltet, da sonst der PC während der Boot-Phase stehenbleiben würde.

5.1.2 Schreiben und Lesen über den Host-Bus

Programmierung der CPLDs

Der PCI9080 übernimmt das PCI-Bus-Protokoll und stellt einen einfachen lokalen 32 bit breiten Bus zur Verfügung, zu dessen Benutzung nur wenige Kent-

[1] PI-PCI64 von Corelis.
[2] 16500B von Hewlett Packard.

Tabelle 5.1: Speicheraufteilung der Tracking-Karte

Startadresse (32bit)	Inhalt
0x0	IOP-Register SHARC*X*
0x20000	»Normal-Word«-Adressierung SHARC*X*
0x40000	»Short-Word«-Adressierung SHARC*X*
0x80000	SRAM Baugruppe *X*
0xC0000	Register **PG** zum Setzen von *X*
0xC0001-0xCFFFF	*Reserviert*

Aufteilung des Speicherbereiches der Tracking-Karte (es ist die PCI-Basisadresse zu addieren) für eine SHARC-SRAM-Gruppe. Die Ansteuerung der verschiedenen Baugruppen erfolgt durch Schreiben von $(1 \ldots 6)$ *in das Register* **PG** *(=***P***rimäre* **G***ruppe). Der reservierte obere Bereich steht noch für spätere Konfigurationen zur Verfügung.*

nisse ausreichend sind. Der Kontroll-CPLD wurde so programmiert, daß der SHARC-Host-Bus angesteuert wird. Da sämtliche Daten- und Adressleitungen durch den Register-CPLD laufen, können die Adressen auf die vorgeschriebene Aufteilung des SHARC-Speichers angepaßt werden. Der Inhalt des **PG**-Registers (vgl. Abb. 5.1) wird an den Kontroll-CPLD übermittelt, damit dieser das Host-Bus-Protokoll mit dem »richtigen« SHARC durchführt. Der Register-CPLD dient auch als Adreß-Dekoder, um nach den Vorgaben der Speicheraufteilung wahlweise die Selektions-Leitungen des SHARC oder der SRAM-Bank zu ziehen.

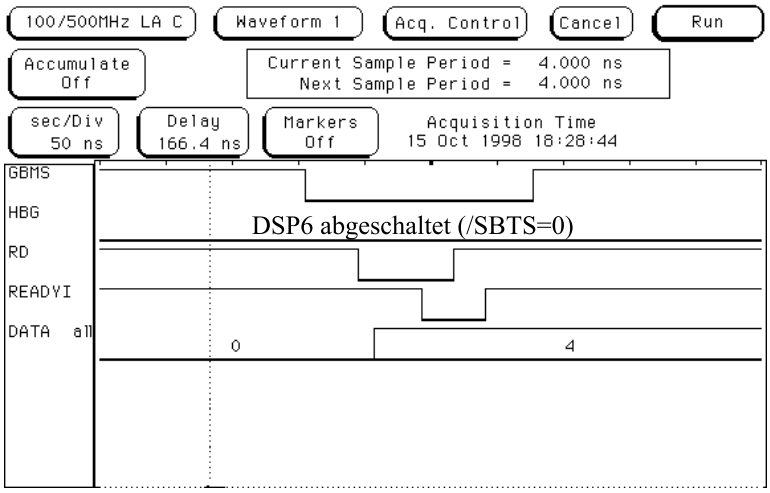
Die Quelltexte dieser beiden CPLDs können Anhang A.3, bzw. A.2 entnommen werden. Der Link-Port-CPLD wurde nicht programmiert, da er für erste Funktions-Tests noch nicht benötigt wurde.

Transport der ersten Daten

Die wichtigste Feststellung bei der anfänglichen Inbetriebnahme jeder Karte ist, daß diese die übermittelten Daten auch korrekt entgegennimmt. Da sechs Leitungen pro CPLD (die »Minibusse«) auch an einem internen Stecker zur Verfügung standen, konnte durch einfache Anpassung der Logikkonfiguration praktisch jedes Signal des lokalen Busses mit Hilfe des schon erwähnten Logikanalysators betrachtet werden.

Auf diese Weise gelang es sehr schnell, das **PG**-Register zu beschreiben. Da der Registerinhalt auch auf drei LEDs abzulesen war, konnte das korrekte Beschreiben dieses Registers auch visuell überprüft werden.

Abbildung 5.1: Auslesen einer SRAM-Bank



Der Lesezyklus aus dem SRAM, aufgenommen mit dem Logikanalysator: Zunächst zieht der Adress-Dekoder das Memory-Select des SRAMs (hier: »GBMS«), kurz darauf erfolgt das Ziehen der Lese-Leitung (»RD«). Die Daten liegen kurz darauf auf dem Bus (siehe die »4« auf »DATA all«, hier sind die letzten drei Bits der Daten abgebildet) und werden vom PCI9080 durch die fallenden Flanke des »READYi« übernommen.

Schreiben und Lesen aus dem SRAM

Von den sechs zur Verfügung stehenden Baugruppen wurden alle relevanten Funktionstests zunächst mit der letzten Baugruppe durchgeführt. Zunächst mußte geklärt werden, ob das Beschreiben und Wiederauslesen des SRAMs funktioniert. Damit mögliche Fehlerquellen besser isoliert werden konnten, wurde der SHARC durch Ziehen der /SBTS-Leitung zunächst abgeschaltet. Dabei zeigte sich, daß das SRAM problemlos beschrieben werden konnte und die Daten auch nach dem Lesen unverfälscht waren. Während der gesamten Arbeit mit der Karte trat keine Fehlfunktion der SRAM-Bauteile oder des PCI-Interfaces auf.

Zugriff auf einen SHARC

Nachdem durch die Zugriffe auf das SRAM die Grundfunktion des lokalen Busses sichergestellt war, wurde der SHARC der sechsten Baugruppe aktiviert. Nach Verfeinerung des Zeitverlaufs der Signale konnten die internen Register dieses SHARCs gelesen werden und mit den Soll-Werten nach [14] verglichen werden. Es war

zu beachten, daß der SHARC nach dem Reset immer auf einen 16 Bit breiten Bus eingestellt ist, d.h. zum Lesen der Register sind zwei Zugriffe auf die gleiche Adresse nötig, denn das 32-Bit-Wort wird dann in zwei Hälften übermittelt. Ebenfalls mußte darauf geachtet werden, daß das /REDY-Signals des SHARCs, durch dessen Ziehen dieser Wartezyklen fordern kann, noch auf »open-drain« eingestellt war[3]. Dadurch ist das REDY-Signal immer auf niedrigem Pegel. Dies machte den prophylaktischen Einschub von Wartezyklen nötig, um beim Zugriff auf die Systemregister und dem späteren Bootvorgang den SHARC nicht zu überfordern.

Durch Schreiben in das entsprechende Register (SYSCON, 0x0) konnte der Bus auf 32 Bit und das REDY-Signal auf Aktiv umgestellt werden. Dies konnte sowohl auf dem Logikanalysator (vgl. Abb. 5.2), als auch anhand der nunmehr 32 Bit breiten gelesenen Datenworte beobachtet werden.

5.1.3 Programmierung der SHARCs

Das Booten

Der letzte entscheidende Schritt bei der Inbetriebnahme einer Baugruppe war das Booten des SHARCs, d.h. das Laden und Starten eines Programms. Dies geschah durch Schreiben der 16-Bit-Datenworte eines »Bootblockes« in Register 0x4 des SHARCs. Die Erstellung dieses Bootblockes übernahm die Entwicklungssoftware von Analog, die auch einen Gnu-**C**-Compiler und einen Assembler enthält. Zum Booten unter Linux wurde ein Programm »boot.c« geschrieben, welches die Bootblöcke in die einzelnen SHARCs schrieb.

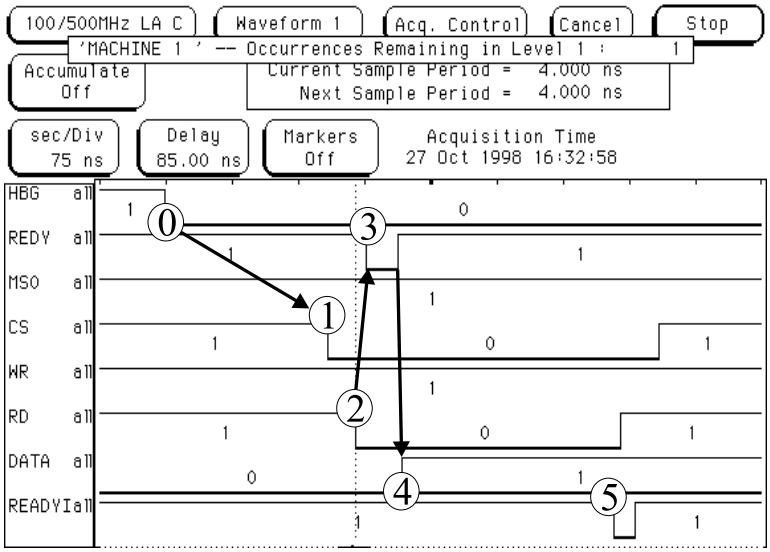
Um die erfolgreiche Bootoperation zu testen, wurden zunächst einfache Programme getestet, die z.B. eine LED ausschalteten oder interne Register beschrieben. Alle folgenden Tests wurden mit nur zwei von acht bestückten SHARCS durchgeführt.

SHARC-Funktionstests

Durch weitere Testprogramme wurden die zu verwendenden Funktionen des SHARCs getestet. Da der spätere Algorithmus keine komplexen Aufgaben erfüllen soll, sondern eher sich wiederholende kleine Teile beinhaltet, wurde von Anfang an auf die Programmierung in **C** verzichtet und alle Programme direkt in Assembler geschrieben. Diese Sprache ist leicht lesbar und nur so werden die großen Vorteile der SHARCs (Parallelisierung von Operationen, direkte Adressmodifizierungen usw.) ausgereizt. Es soll kurz erläutert werden, welche Tests durchgeführt wurden:

[3] Dadurch können mehrere SHARCs durch »wired-or«, also durch einfaches Verbinden der Leitungen, beobachtet werden.

Abbildung 5.2: Auslesen eines SHARC-Registers



Durch das Ziehen der /HBR-Leitung (hier nicht dargestellt) wird ein Zugriff angemeldet, der SHARC antwortet mit /HBG (siehe 0). Dadurch startet der Kontroll-CPLD die vorliegendene zeitliche Abfolge der Signale: Der Adress-Dekoder wird veranlaßt, entweder das /CS des SHARCs oder die Select-Leitung der SRAM-Bank (hier: /MS0) zu ziehen (1). Da der Zugriff in diesem Falle den SHARC betraf, antwortete dieser mit einem Ziehen der /REDY-Leitung, die hier schon auf »Aktiv« (siehe Fließtext) eingestellt worden ist (3). Mit dem Zurücksetzen der /REDY-Leitung sind die Daten auf dem Bus gültig (4). Diese werden vom PCI9080 bei (5) übernommen.

SRAM-Zugriff: Die Zahl der Wartezyklen für die SRAM-Bank wurde auf Null eingestellt, damit die höchste Geschwindigkeit erreicht werden konnte. Durch Lesebefehle im Programm des SHARCs und Ablegen in interne Speicherbereiche wurde bestätigt, daß das verwendete SRAM von Samsung für den SHARC bei 40MHz ausreichend schnell ist.

Rechenoperationen: Es wurden Zahlen vom Host-**C**-Programm an den SHARC übergeben, welcher einfache Additionen und Multiplikationen durchführte. Dabei zeigte sich, daß im Gegensatz zu den Festkommaoperationen (bei welchen keine Probleme auftauchten) bei den Fließkommazahlen die Ergebnisse falsch waren. Dies lag daran, daß der GNU-**C**-Compiler ein IEEE-Format mit einem genaueren 15-Bit-Exponenten benutzt, während der SHARC mit 7 Bit rechnet. Diese Inkompatibilität kann prinzipiell durch Umrechnung behoben oder durch Vermeidung von Fließkommaschnittstellen zwischen dem Host-Programm und der Tracking-Karte umgangen werden.

Link-Ports: Durch Programmierung eines weiteren SHARCs (Baugruppe 5) wurde die Kommunikation über die Link-Ports getestet. Der Transport von Daten von SHARC6 zu SHARC5 und zurück verlief erfolgreich.

DMA-Programmierung: Letztendlich sollten die Daten per Block im Hintergrund ohne Geschwindigkeitsbeeinträchtigung des laufenden Programms transportiert werden. Zu diesem Zweck wurden die DMA-Fähigkeiten des SHARC ausgenutzt. Es wurde ein Programm auf dem SHARC6 gestartet, welches ein DMA-Block mit 32*32 Bit zu SHARC5 sendet und gleichzeitig einen ebensogroßen von diesem erhält. Die SHARCs sollten vor einem Neustart des Programms warten, bis der alte Zyklus beendet war (»Selbstsynchronisation«). Dabei konnte beobachtet werden, daß der Zyklus (der auch den Start der DMA-Sequenz beinhaltete) insgesamt 4.2μs dauerte. Dies entspricht einer Übertragungsrate von 61 MegaByte pro Sekunde für zwei Kanäle (Ein Eingang- und ein Ausgangskanal) über die Links-Ports (gegenüber theoretischen 80MegaByte/s). Die Differenz läßt sich durch den Zeitverlust bei Programmschleifen und Reprogrammierung des DMA-Transfers verstehen, daher dürfte das Verhältnis bei zunehmender Blockgröße besser werden.

5.2 Teilimplementierung des Algorithmus

Nachdem alle zu verwendenden Funktionen der Karte getestet wurden, konnte ein SHARC dahingehend programmiert werden, Teile des Tracking-Algorithmus zu

übernehmen, insbesondere den Rechen- und Speicherintensiven Teil der Interpolation in fünf Koordinaten (vgl. Kapitel 3).

5.2.1 Ein realistischer Test des Gesamtkonzeptes

Die wesentlichen Schritte einer Stufe in dem Tracking-Algorithmus sind:

1. Die Aufstellung der Koeffizienten.
2. Die Interpolation und Berechnung der Ableitungen in der fünfdimensionalen Tabelle.
3. Die Aufstellung der Matrix.
4. Lösung des Gleichungssystems.

Dabei lassen sich die Punkte 2 und 3 für jede einzelne Koordinaten getrennt durchführen. Für den Test wurde der rechenintensivste Punkt implementiert, und zwar die Interpolation. Da die Karte mit sechs SHARCs bestückt ist, muß jeder zwei Interpolationen durchführen, denn es handelt sich um 12 Koordinaten im Falle von DISTO. Für einen realistischen Test reichte es aus, nur einen SHARC mit diesen Aufgaben zu programmieren, da die anderen Koordinaten analog behandelt werden und daher die gleiche Zeit brauchen.

Erläuterung des Programms

Das Programm (siehe Anhang C.3) für SHARC6 sollte folgende Funktionen erfüllen:

- DMA-Blocktransfer einer »Datenstruktur« über die Link-Ports, der z.B. die Koeffizienten oder die Matrixelemente enthält. Die letzendliche Anzahl der 32-Bit-Zahlen, die von einem Interpolationsschritt zum nächsten weitergegeben werden müssen, sind 82. Da im Endkonzept jeweils zwei Link-Port-Kanäle für die Annahme und Weitergabe der Datenvektoren zur Verfügung stehen, wurde die Blockgröße auf 41 eingestellt. Dabei war es für einen Test nicht nötig, diesen Vektor auch wirklich mit Daten zu füllen. Alle Datenvektoren sollen im internen SRAM des SHARCs abgelegt werden, um eine schnelle Weitergabe per DMA zu ermöglichen.
- Interpolation in der Tabelle, welche in der externen SRAM-Bank abgespeichert wurde. Da der Schritt 1 des Algorithmus noch nicht programmiert wurde, konnten die notwendigen Koeffizienten nicht durch einen vorhergehenden SHARC übermittelt werden, sondern von einem Host-Programm

auf dem Linux-PC. Die Sprünge bei den Speicherzugriffen entsprachen der Struktur der look-up-Tabelle für die DISTO-Analyse.

- Berechnung der fünf Ableitungen durch Subtraktion in den Stützstellen. Auch hier konnten die Stützstellen nicht durch einen vorhergehenden SHARC übermittelt werden, so daß feste Werte benutzt wurden, was aber für eine Geschwindigkeitsbewertung ohne Belang ist.
- Abspeicherung der Ergebnisse in einem Datenvektor, der im internen Speicher abgelegt wird.
- Wiederholung der letzten drei Schritte mit der zweiten Koordinate.
- Warten, bis der DMA-Transfer beendet ist und Neustart der Routine.

SHARC5 diente dabei nur zur Annahme und Weitergabe der DMA-Blöcke, wie schon in dem Test zuvor. Für die letztendliche Implementierung ist geplant, die Zeiger auf die Datenvektoren einfach umzusetzten, so daß gerade berechnete Ergebnisse beim nächsten Durchlauf der Routine zum nächsten SHARC geschickt werden, bzw. der vorher empfangene Vektor dann für die neue Berechnung benutzt wird. Durch die Selbstsynchronisation (vgl. vorhergehende Tests) kann eine Überschneidung in dem Daten- und Berechnungsablauf ausgeschlossen werden.

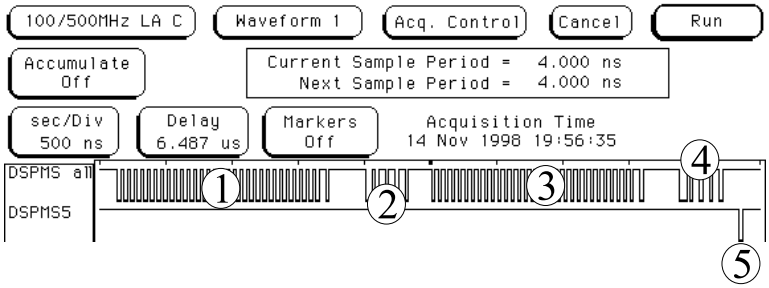
Zeitbedarf der Routine

Zur Überwachung der SHARC-Operationen wurde mit dem Logikanalysator die /MS0-Leitungen der SHARCs 5 und 6 aufgezeichnet. Dadurch konnte der Zugriff auf die look-up-Tabelle im SRAM beobachtet werden. Dies betraf sowohl die Interpolation, als auch die Differenzenbildung für die Ableitungen. Hier sei insbesondere auf das Makro »interpol« in Anhang C.1 verwiesen, welches die 32 Koeffizienten aus der Datenstruktur holt, um sie mit den 32 Stützstellen der Tabelle zu multiplizieren und die Summe zu bilden (vgl. (3.9)). Dabei wechselt der Zugriff auf das SRAM mit Operationen im internen Speicher ab, wie auch in Abb. 5.3 zu sehen ist. Man sieht, daß eine auf diese Weise durchgeführte Interpolation weniger als 2μs benötigte. SHARC5 quittierte den Abschluß aller DMA-Transfers mit einem SRAM-Zugriff, so daß zu erkennen ist, daß der gesamte oben beschriebene Zyklus in weniger als 5μs abgeschlossen war.

Das Host-Programm

Ein Host-Programm auf dem Linux-PC diente zur Generierung der Koeffizienten und dem Füllen der Datenstrukuren mit Zufallswerten. Sowohl die Ergebnisse,

Abbildung 5.3: SRAM-Operationen bei laufendem Betrieb



Die Zugriffe auf die beiden SRAM-Bänke von SHARC6 (oben) und SHARC5 (unten) bei laufendem Interpolations-Testprogramm mit zwei Link-Port-DMA-Kanälen. SHARC6 startet den DMA-Transfer (Ein Empfangs- und ein Sendekanal) mit SHARC5, der nur als Empfänger und Sender für den DMA-Transfer dient. Anschließend startet SHARC6 die erste Interpolation, indem die 32 Stützstellen aus dem SRAM gelesen werden (1). Für die gleiche Koordinate werden dann noch die Differenzen von sechs Stützstellen für die Ableitungen berechnet (2). Dieser Vorgang wird noch mit der zweiten Koordinate wiederholt (3+4). Der abgeschlossene DMA-Transfer wird von SHARC5 mit einem Speicherzugriff bestätigt (5). Ein Vergleich mit der Zeitskala zeigt, daß der gesamte Zyklus weniger als 5μs benötigt.

als auch die über die Link-Ports transportierten Daten wurden wieder entgegengenommen und überprüft. Allerdings traten dabei zunächst Fehler auf, was dadurch erklärt werden konnte, daß das Host-Programm die Ergebnisse las, bevor sie vom SHARC berechnet worden waren[4]. Nach einigen Modifikationen, die das Host-Programm ein wenig langsamer machten, lief es ohne Fehlermeldungen (Quelltext bitte Anhang B.1 entnehmen). Hier ist in späteren Versionen eine Verständigungsmöglichkeit zu implementieren, um dem Host-Programm die erfolgte Bearbeitung eines Fits anzuzeigen[5].

Dies zeigt, daß es weder bei den Zugriffen über den PCI-Bus auf die Komponenten zu Übertragungsfehlern kam, noch bei der Übergabe der Daten über die Link-Ports. Die SHARCs liefen stabil und liefen erwartungsgemäß.

5.2.2 Vergleich mit einem Intel-PC

Zur Bewertung wurde ein **C**-Programm geschrieben, das ebenfalls eine Interpolation mit 32 Stützstellen durchführte und die fünf Differenzen bildete (Quelltext siehe Anhang B.2). Dabei wurden die gleichen Sprünge in der Tabelle verwendet wie bei dem SHARC-Programm. Es zeigt sich, daß eine solche Routine auf einem Intel Pentium II (300 MHz) bei Einschaltung aller Optimierungen des Gnu-C-Compilers 9μs für die Berechnung von *12 Koordinaten* braucht. Jeder SHARC auf dem Tracking-Board benötigt 5μs für *2 Koordinaten,* jedoch mit komletter Datenübertragung via Link-Port, so daß auch der Zeitbedarf für 12 Koordinaten mit den komplett in Betrieb genommen SHARCs nicht steigen wird.

Der Geschwindigkeitsgewinn für den in Abschnitt 5.2.1 vorgestellten Algorithmus - der Interpolation mit 32 Stützstellen und dem Erhalt der Ableitungen durch fünf Differenzen - zwischen Tracking-Karte im Vergleich und einem Intel Pentium II (300 MHz) wird daher mit 1,8 angegeben.

[4] Dies betraf vor allem die Link-Ports.

[5] Also z.B. Auslösung eines Interrupts oder aktives Schreiben der Daten von der PCI-karte in den PC-Speicher durch Ausnutzung der Master-Fähigkeit des PCI9080.

Kapitel 6

Diskussion und Ausblick

6.1 Durchgeführte Test und mögliche Modifikationen

Die hier vorgestellte PCI-Karte erfüllt die Aufgabe als Coprozessor im Zusammenhang mit der Analyse der Daten des DISTO-Spektrometers. In der vorgestellten Version wurden die Burst-Möglichkeiten des PCI-Busses noch nicht genutzt und der Host-Zugriff auf die SHARCs noch nicht mit höchster Geschwindigkeit durchgeführt. Für die später zu übergebenden Datenstrukturen ist geplant, alle Vorteile des PCI-Busses zu nutzen. Dies ist erforderlich, um das Zeitfenster des Host-Zugriffes auf die SHARCs möglichst kurz zu halten. Durch Übergabe nicht nur einer Datenstruktur, sondern eines ganzen Speicherblockes, wird die Burst-Möglichkeit des PCI-Busses ausgenutzt werden. Auch eine Aktivierung der Master-Fähigkeit des PCI9080-Interfaces ist denkbar, um die gewonnenen Parameter und das χ^2 von einem SHARC in den Arbeitsspeicher des PCs zu schreiben.

Während der Tests mit der Karte kam es zu keinen sichtbaren Fehlfunktionen der SHARCs. Auch der Datendurchsatz mittels DMA über die Link-Ports lag mit ca. 30 MegaByte/s im Bereich der Erwartungen. Dadurch liegt der Zeitbedarf für die Weitergabe der Datenstrukturen unter der benötigten Interpolationszeit pro SHARC.

Durch die verwendete programmierbare Logik, das Link-Port-Netz und den externen Stecker sind viele weitere Anwendungsmöglichkeiten denkbar. So könnte man sowohl die Anzahl der Karten im System vergrößern, als auch externe Geräte anschließen. Durch diese einfache Skalierbarkeit kann der erreichte Geschwindigkeitsgewinn weiter gesteigert werden. Das Optimum wäre erreicht, wenn für die Interpolation jeder Koordinate ein gesonderter SHARC zur Verfügung stehen und vier weitere die anderen Stufen im Track-Fitting übernehmen würden.

Zusammenfassend zeigen die durchgeführten Tests, daß diese Karte Algorith-

men, die sehr viel und sprunghaft auf Tabellen zugreifen, gegenüber einer Programmierung auf einem PC beschleunigen kann. Für einen Teil der Track-Fit-Routine, die hier implementiert wurde, konnte eine Beschleunigung gegenüber einem Pentium II 300 um den Faktor 1.8 erreicht werden. Hier sollte aber klar vermerkt werden, daß dies nicht allgemein gilt. In dem hier implementierten Algorithmusteil haben die Speicherzugriffe und die Fließkommaoperationen in etwa das Verhältnis eins zu eins. Für andere Algorithmen, die u.U. weniger Speicherzugriffe benötigen, bietet der parallele Zugriff der SHARCs auf die lokalen Speicherbänke gegenüber dem globalen Speichermodell (wie bei Pentium-Prozessoren) keinen Vorteil mehr. Es muß also bei geänderten Algorithmen erneut beurteilt werden, welche Architektur besser geeignet sein könnte. Für die Auswertung von Spektrometerdaten, welchen ein ähnliches Konzept wie DISTO zugrunde liegt (mehrere ortsempfindliche Detektorebenen und große Akzeptanz) ist jedoch nicht mit einer grundlegenden Änderung des Konzeptes zu rechnen (Speicherung der Information $\vec{x}(\vec{p})$ in einer Tabelle, Interpolation und Minimasuche).

6.2 Entwicklungsmöglichkeiten im Hinblick auf HADES

Für die HADES-Analyse ist mit einem Anstieg der Analysedauer zu rechnen, da sowohl die Anzahl der Koordinaten (24 statt 12), als auch die Zahl der nachgewiesenen Teilchen größer als bei DISTO sein wird (bis zu 200 statt 4). Die gute Massenauflösung verlangt auch eine sehr viel größere look-up-Tabelle (ca. 100 MegaByte). Durch eine quadratische bzw. kubische Interpolation statt der hier vorgestellten linaren Interpolation ist zwar eine starke Reduktion der *insgesamt* benötigten Stützstellen (und damit eine Verringerung des Speicherbedarfs) möglich, dafür steigt die Zahl der *pro Interpolation* benötigten Stützstellen von $2^5 = 32$ auf $3^5 = 243$, bzw. $4^5 = 1024$.

Die hier vorgestellte Karte ist nur als Test für eine Umsetzung für HADES zu verstehen, da der Speicher auf der Karte für diese Anwendung zu klein ist. Außerdem werden die hier benutzten SHARCs Mitte 1999 durch eine neue Generation abgelöst [20]. Dieser »ADSP21160« läuft mit 100 MHz und verdoppelt die arithmetischen Einheiten. Dadurch ist die Rechenleistung um den Faktor fünf größer. Weitere Verbesserungen sind die nunmehr acht Bit breiten Link-Ports und der 64 Bit breite externe Bus. Der ADSP21160 wird ausschließlich im »BGA«-Gehäuse (**B**all **G**rid **A**rray) geliefert, welches die Pins auf der Unterseite des Gehäuses trägt. Diese Gehäuseform spart Platz auf der Platine, wodurch eine bessere Ausnutzung einer PCI-Karte erfolgen kann, so daß statt der sechs Prozessoren bis zu 12 Platz auf der Karte finden werden. Eine Übertragung des Konzeptes auf den neuen Prozessor ist mit wenigen Modifikationen möglich. Tabelle 6.1 zeigt die möglichen

Tabelle 6.1: Geschwindigkeitsgewinn durch weiteren Ausbau

Ausbaustufe	Vorteil gegenüber Pentium II 300
Vorgestellte Karte (ADSP-21062)	Faktor 1,8
Mit 12 Prozessoren	Faktor 3,6
Mit ADSP-21160-Bestückung	Faktor 18

Geschwindigkeitsgewinne verschiedener Ausbaustufen des Konzeptes. Die Entwicklung eines weiteren Nachfolgers (der »TigerSHARC«) ist für das Jahr 2000 geplant [21].

Aus den genannten Gründen bietet dieses Konzept Möglichkeiten zur Weiterentwicklung. Gerade im Hinblick auf die Auswertung der HADES-Daten, welche den Rechenleistungsbedarf im Vergleich zur DISTO-Analyse erheblich vergrößern wird, sollte ein solches Konzept zum Einsatz kommen. Da der komplette Track-Fit in der hier vorgestellten Karte implementiert werden wird, führen die so gewonnenen Erfahrungen und Softwareentwicklungen zu einem Vorsprung bei der Weiterentwicklung des Konzeptes für die HADES-Analyse.

Literaturverzeichnis

[1] HADES-Collaboration; Proposal for a High-Acceptance Di-Electron Spectrometer, 1994.

[2] M. Lutz, S. Klimt and W. Weise; Nucl. Phys. A542(1992)621.

[3] Hans-Werner Pfaff for the DISTO Collaboration; Meson Production in pp-Collisions at 2.85 GeV; Proceedings of the 14^{th} Winter Workshop on Nuclear Dynamics; Snowbird, Utah (USA).

[4] F. Balestra et al; DISTO: A Large Acceptance Multiparticle Spectrometer; Nucl. Instrum. Methods, in print.

[5] F. Balestra et al; Production of ϕ and ω Mesons in Near Threshold pp Reactions; Phys. Rev. Lett. **81**, 4572 (1998).

[6] F. Balestra et al; Spin Transfer in Exclusive Λ Production from $\vec{p}p$ Collisions at 3.67GeV/c; submitted to Phys. Rev. Lett.

[7] Arndt Brenschede; Untersuchung der ω- und ϕ-Produktion in Proton-Proton-Stößen nahe der Reaktionsschwelle; Dissertation an der Universität Gießen, II. Phys. Inst. 1997.

[8] Hans-Werner Pfaff; Private Mitteilung.

[9] F. James, CERN; Function Minimization, Reprinted from the Proceedings of the 1972 CERN Computing and data Processing School.

[10] Oliver Rovini; Bus Basics, Technische Grundlagen des PCI-Bus; ELRAD 1997, Heft 3.

[11] PCISIG; PCI-Specifikation 2.1.

[12] Tom Shanley, Don Anderson; PCI System Architecture; Addison-Wesley Publ. Company, 1997.

[13] PLX Technology; PCI 9080 Data Sheet, V1.04.

[14] Analog Devices; ADSP-2106x SHARC User's Manual, 2nd edition.

[15] Analog Devices; ADSP-21062/ADSP21062L data sheet.

[16] René Becker; Private Mitteilung.

[17] Marc-André Pleier; Entwicklung eines Spill-Monitors zur Untersuchung der SIS-Strahlstruktur im Rahmen des HADES-Experimentes; Diplomarbeit an der Universität Gießen, II.Phys. Inst. 1998; und dort enthaltene Referenzen.

[18] Alessandro Rubini; Linux Device Drivers; O'Reilly & Associates Inc., 1998.

[19] Michael J. Wirthlin, Peter Bellows; Driver for 6216-based HotWorks board; Brigham Young University.

[20] Analog Devices; Priliminary Technical Data ADSP-21160.

[21] Presseinformation von Analog Devices; www.analog.com/publications/press/product/tiger_101498.html

Anhang A

Zur Konfiguration

A.1 Das EEPROM

Register	Inhalt
Device, Vendor-ID	DEAA BEEF
Class Code, Revision	0680 0001
Max. Latency, Min. Grant, Interrupt Pin + Line Routing	0000 0100
Mailbox 0	0000 0000
Mailbox 1	0000 0000
Adress space 0 (Range)	FFC0 0000
Adress space 0 (local Base address)	0000 0001
Local Arbitration	0020 0000
Big/Little Endian	0000 0000
ROM (Range)	0000 0000
ROM (local Base address)	0000 0000
Descriptor for PCI to local accesses	4800 0043
alle restlichen	0000 0000

A.2 Controller-CPLD

Typ: Abel-HDL

Name: ctrl.abl

---

module ctrl
Title `,ctrl,`

Declarations

```
"PLX-Steuerleitungen"
LBE0, LBE1, LBE2, LBE3 pin 57, 58, 60, 61;
READYo pin 51;
READYi pin 50 istype ,reg_d,;
LHOLDA pin 48;
LHOLD pin 47;
LLOCK pin 45;
LWR pin 43;
DTR pin 41;
DEN pin 40;
BTERM pin 39;
BLAST pin 38;
ADS pin 37;
WAITO pin 36;
USERI pin 35;
LRESETi pin 34;
LINTo pin 33;
LINTi pin 32;
DACK0 pin 31;
DREQ0 pin 30;
BTERMo pin 29;
USERO pin 28;
DACK1 pin 25;
DREQ1 pin 23;
LSERR pin 22;
BREQo pin 21;
PCHK pin 20;
DP0, DP1, DP2, DP3 pin 19,18,17,16;
DMPAF pin 15;
LLOCKo pin 14;
WAITI pin 12;
EOT0, EOT1 pin 8,10;
BREQ pin 6;
LDSHOLD pin 4;

"DSP-Steuerung"
SBTS1, SBTS2, SBTS3, SBTS4, SBTS5, SBTS6 pin 167, 147, 127, 118, 138, 158;
SBTS=[SBTS1..SBTS6];
CS1, CS2, CS3, CS4, CS5, CS6 pin 189, 191, 192, 193, 194, 195;
CS=[CS1..CS6];
DMAR1, DMAR2, DMAR3, DMAR4, DMAR5, DMAR6 pin 168, 148, 128, 119, 139, 159;
DMAR=[DMAR1..DMAR6];
HBR1, HBR2, HBR3, HBR4, HBR5, HBR6 pin 171, 150, 133, 121, 142, 161;
HBR=[HBR1..HBR6];
HBG1, HBG2, HBG3, HBG4, HBG5, HBG6 pin 196, 197, 198, 199, 200, 201;
HBG=[HBG1..HBG6];
IRQ1, IRQ2, IRQ3, IRQ4, IRQ5, IRQ6 pin 170, 151, 134, 122, 143, 162;
IRQ=[IRQ1..IRQ6];
```

```
"Baugruppen gesamt"
GBMS pin 106;
GBWR pin 110;
GBRD pin 111;
SW pin 113;
REDY pin 107;
RESET pin 101;

"Kommunikation"
mini0, mini1, mini2, mini3, mini4, mini5 pin 62, 63, 64, 67, 70, 71;
mini=[mini0. .mini5];
talk0, talk1, talk2, talk3, talk4, talk5 pin 89, 88, 87, 86, 85, 84;
talk_stat= [talk0. .talk2];
DD pin 78;
DA pin 80 istype ,reg_d,;
VALID pin 76 istype ,reg_d,;
GBHBG pin 109;

"LED"
LED0, LED1, LED2, LED3 pin 75, 74, 73,72;
LED_stat = [LED0, LED2, LED3]; "rote LEDs"

clock pin 44;
reset_input pin 206;

cycle0. .cycle5 node istype ,reg_d,;
cycle = [cycle0. .cycle5];
HBR_int node istype ,reg_d,;

Equations
"DSP-Steuerleitungen"
SBTS5 = 1;
SBTS6 = 1;
SBTS1=0;
SBTS2=0;
SBTS3=0;
SBTS4=0;
DMAR = ^B111111;
IRQ = ^B111111;
RESET = 1;
RESET.oe = reset_input;
SW=1;

"LED-Ansteuerung"
LED0 = !talk0;
LED1= 1;
LED2 = !talk1;
LED3 = !talk2;
```

```
"PLX-Steuerung"
LRESETi=1;
BREQ=0;
DREQ0=1;
DREQ1=1;
LINTi=1;
WAITI=1;
EOT0=1;
EOT1=1;
LLOCK=1;
BTERM = 1;

"Bus-Arbitrierung"
when (talk_stat == 1) then HBR1=HBR_int else HBR1=1;
when (talk_stat == 2) then HBR2=HBR_int else HBR2=1;
when (talk_stat == 3) then HBR3=HBR_int else HBR3=1;
when (talk_stat == 4) then HBR4=HBR_int else HBR4=1;
when (talk_stat == 5) then HBR5=HBR_int else HBR5=1;
when (talk_stat == 6) then HBR6=HBR_int else HBR6=1;

when (talk_stat == 1) then LHOLDA=!HBG1;
when (talk_stat == 2) then LHOLDA=!HBG2;
when (talk_stat == 3) then LHOLDA=!HBG3;
when (talk_stat == 4) then LHOLDA=!HBG4;
when (talk_stat == 5) then LHOLDA=!HBG5;
when (talk_stat == 6) then LHOLDA=!HBG6;
when ((talk_stat < 5) # (talk_stat == 7)) then LHOLDA=LHOLD;
GBHBG = HBR_int;

"Puffern der Ausgaenge"
VALID.clk = !clock;
READYi.clk = !clock;
DA.clk = !clock;
HBR_int.clk = !clock;

"Erzeugung der CYCLE-Daten fuer das Timing"
"cycle2 bei steigender Flanke des LHOLDA"
cycle.clk = clock;
when ((HBR_int == 0 ) & (ADS == 1)) then cycle.d = (cycle+1) else cycle.d =0;

"Erzeugung des VALID"
when ((cycle > 5) & (cycle < 18)) then VALID.d=0 else VALID.d=1;

"Erzeugung des READYi"
when (cycle == 18) then READYi.d = 0 else READYi.d=1;

"Erzeugung des DA"
when ((cycle > 3) & (cycle < 19)) then DA.d=1 else DA.d=0;
```

```
"Erzeugung des HBR_int"
when ((LHOLD == 1) # ((cycle > 2) & (cycle < 30))) then HBR_int.d=0 else HBR_int.d=1;

"Erzeugung des Schreib/Lesesignals"
GBWR = !(!VALID & LWR);
GBRD = !(!VALID & !LWR);
DD = LWR; "Beim Lesen ist DD=0"

mini=cycle;

end ctrl
```

A.3 Adress-Decoder-CPLD

```
module addr
Title ,addr,

Declarations

"PLX lokale Addressen"
LA2,LA3,LA4,LA5,LA6,LA7   PIN 49,48,47,45,43,41;
LA8,LA9,LA10,LA11,LA12,LA13 PIN 40,39,38,37,36,35;
LA14,LA15,LA16,LA17,LA18,LA19 PIN 34,33,32,31,30,29;
LA20,LA21,LA22,LA23,LA24,LA25 PIN 28,25,23,22,21,20;
LA26,LA27,LA28,LA29,LA30,LA31 PIN 19,18,17,16,15,14;
PLX_GA = [LA2. .LA31];
PLX_LA_A = [LA2. .LA18]; "Adressleitungen"
PLX_LA_S = [LA19. .LA20]; "S-Feld fuer DSP"
PLX_LA_E = [LA21. .LA22]; "Externes Feld (00=DSP, 01=MEM+REG)"
"Karte benoetigt 0x400000 Memory-Bereich = 4MByte"

"PLX lokale Daten"
LD31,LD30,LD29,LD28,LD27,LD26 PIN 10,8,6,4,208,205;
LD25,LD24,LD23,LD22,LD21,LD20 PIN 203,202,201,200,199,198;
LD19,LD18,LD17,LD16,LD15,LD14 PIN 197,196,195,194,193,192;
LD13,LD12,LD11,LD10,LD9,LD8 PIN 191,189,188,187,186,185;
LD7,LD6,LD5,LD4,LD3,LD2 PIN 183,182,180,179,178,175;
LD1,LD0 PIN 174,173;
PLX_LD = [LD0. .LD31];


"DSP globale Addressen"
GA0,GA1,GA2,GA3,GA4,GA5 PIN 135,136,137,138,139,140;
GA6,GA7,GA8,GA9,GA10,GA11 PIN 142,143,144,145,146,147;
GA12,GA13,GA14,GA15,GA16,GA17 PIN 148,149,150,151,152,154;
```

```
GA18,GA19,GA20,GA21,GA22,GA23 PIN 155,158,159,160,161,162;
GA24,GA25,GA26,GA27,GA28,GA29 PIN 164,165,166,167,168,169;
GA30,GA31 PIN 75,74;
GB_ADDR_A = [GA0..GA16]; "DSP interne Adressen"
GB_ADDR_S = [GA17..GA18]; "DSP S-Feld"
GB_ADDR_M = [GA19..GA21]; "DSP Multiprozessor-ID"
GB_ADDR_E = [GA23..GA31]; "Externes Memory"
GB_A=[GA0..GA31];

"DSP globale Daten"
GD16,GD17,GD18,GD19,GD20,GD21 PIN 91,95,97,99,100,101;
GD22,GD23,GD47,GD46,GD45,GD44 PIN 102,103,134,133,131,128;
GD43,GD42,GD41,GD40,GD39,GD38 PIN 127,126,125,123,122,121;
GD37,GD36,GD35,GD34,GD33,GD32 PIN 120,119,118,117,116,115;
GD31,GD30,GD29,GD28,GD27,GD26 PIN 114,113,112,111,110,109;
GD25,GD24 PIN 107,106;
GB_DATA = [GD16..GD47];

"Steuerleitungen"
DA PIN 171;
DD PIN 170;
VALID PIN 12;
SWOE1,SWOE2,SWOE3,SWOE4,SWOE5,SWOE6 PIN 67,64,63,62,61,60 istype ,reg_d,;
SWOE= [SWOE1..SWOE6];
plx_cs pin 71;
CS1, CS2, CS3 pin 83, 82, 80 istype ,reg_d,;
CS4, CS5 pin 78, 77 istype ,reg_d,;
CS6 pin 76 istype ,reg_d,;
CS=[CS1..CS6];
clock pin 44;
GB_MS pin 90 istype ,reg_d,;


"Minibus"
minia0, minia1, minia2, minia3, minia4, minia5 pin 58, 57, 56, 54, 51, 50;
minia = [minia0..minia5];
talk0, talk1, talk2, talk3, talk4, talk5 pin 89, 88, 87, 86, 85, 84;
talk_stat= [talk0..talk2];
talk_r = [talk3..talk5];

"Statusregister"
STATREG1,STATREG2,STATREG3 NODE istype ,reg_d,;
STATREG = [STATREG1..STATREG3];

reg_strobe node;
mem_select node;
da_output node;
switching node;
switching_d node istype ,reg_d,;
```

```
Equations

plx_cs=1; "kein Schreiben von lokaler Seite"


GB_ADDR_A=PLX_LA_A; "Globale Adressen sind unidirektional"
GB_ADDR_S=PLX_LA_S;
GB_ADDR_M=0;                                                                    90
GA22 = LA21;
GB_ADDR_E=^B111111111;


"Ansteuerung internes Register"
STATREG.d = [LD2..LD0]; "Bit 0-2 in Statusregister"
when ((LA20 == 1 ) & (VALID == 0) & (LA21==1)) then reg_strobe = 0 else reg_strobe = 1;
"Register in 300000"
STATREG.clk = reg_strobe;
"Uebernahme der Daten bei steigender Flanke von VALID"                          100
talk_stat=STATREG; "Inhalt des Registers an Controller"

"Schalten der Switches mit Statusregister (nur wenn DA=1)"
SWOE.clk=!clock;
WHEN ( (DA == 1) & ((LA20 == 0) # (LA21 == 0)))  then switching = 1 else switching = 0;
when (STATREG == 1) then SWOE1.d = !(switching) else SWOE1.d = 1;
when (STATREG == 2) then SWOE2.d = !(switching) else SWOE2.d = 1;
when (STATREG == 3) then SWOE3.d = !(switching) else SWOE3.d = 1;
when (STATREG == 4) then SWOE4.d = !(switching) else SWOE4.d = 1;
when (STATREG == 5) then SWOE5.d = !(switching) else SWOE5.d = 1;               110
when (STATREG == 6) then SWOE6.d = !(switching) else SWOE6.d = 1;

"Bidirektionale Logik fuer Daten"
"Durchschalten wenn VALID=0"
"Lesen (von PLX-Seite gesehen) wenn DD=0"
"Schreiben wenn DD=1"
"Ausgang schaltet, wenn .oe =1"
"Wichtig:Adressen werden nur aktiv, wenn DA=1"
GB_DATA=PLX_LD;
PLX_LD=GB_DATA;                                                                 120
GB_DATA.oe = (DD & DA);
PLX_LD.oe= (!DD & DA);

"SRAM selektieren"
mem_select = !(DA & !LA20 & LA21);
GB_MS.d = mem_select;
GB_MS.clk = !clock;

"DSP selektieren"
```

```
CS.clk=!clock;                                                                    130
when (STATREG == 1) then CS1.d = !(DA & !LA20 & !LA21) else CS1.d = 1;
when (STATREG == 2) then CS2.d = !(DA & !LA20 & !LA21) else CS2.d = 1;
when (STATREG == 3) then CS3.d = !(DA & !LA20 & !LA21) else CS3.d = 1;
when (STATREG == 4) then CS4.d = !(DA & !LA20 & !LA21) else CS4.d = 1;
when (STATREG == 5) then CS5.d = !(DA & !LA20 & !LA21) else CS5.d = 1;
when (STATREG == 6) then CS6.d = !(DA & !LA20 & !LA21) else CS6.d = 1;

end addr
```

A.4 Bestückte Karte

Anhang B

C-Quelltexte

B.1 Host-Programm

Typ: **C**-Quelltext

Name: host.c

```
 /*************************************************************************
*
* Host-Program for the Tracking-Board
*
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>                                                    10
#include <sys/stat.h>
#include <linux/errno.h>
#include <fcntl.h>
#include <errno.h>
#include <asm/mman.h>
#include <asm/types.h>
#include "/home/ingo/driver/trck.h"
#include <math.h>
#include "test.h"
#include "dspboard.h"                                                     20

/*************************************************************************
*
* get_trck_board
*
```

```
*************************************************************************/
int get_trck_board(void)
  {
  int descriptor=-1;
  char dev_path[100];

  strcpy(dev_path,"/dev/trck");
  printf("trying to open:%s\n",dev_path);
  descriptor= open(dev_path,O_RDWR);
  return descriptor;
  }

/*************************************************************************
*
* get_memory_ptr
*
*************************************************************************/
unsigned long get_memory_ptr(int descriptor)
  {
  unsigned long memory_ptr;
  int temp;

  temp=mmap(0,TRCK_MEM_SIZE,(PROT_READ|PROT_WRITE),
            (MAP_FILE|MAP_SHARED),descriptor,0);
  if (temp== -1)
    return 0;
  else
    memory_ptr = temp;
  return memory_ptr;
  }

/*************************************************************************
*
* release_memory_ptr
*
*************************************************************************/
unsigned int release_memory_ptr(int descriptor, unsigned long memory_ptr)
  {
  return munmap(memory_ptr,TRCK_MEM_SIZE);
  }

/*************************************************************************
*
* main
*
*************************************************************************/
unsigned int main(void)
  {
  int descriptor=0;
```

```
FILE *fd;
unsigned int memory,i,number_coeff;
unsigned int value_dma[40];
float data,data_dsp, value_coeff[32];
unsigned int node[32],test;
volatile int waits;                                                    80

node[0]=0;node[1]=1;node[2]=48;node[3]=49;
node[4]=1536;node[5]=1537;node[6]=1584;node[7]=1585;
node[8]=9216;node[9]=9217;node[10]=9264;node[11]=9265;
node[12]=10752;node[13]=10753;node[14]=10800;node[15]=10801;
node[16]=18432;node[17]=18433;node[18]=18480;node[19]=18481;
node[20]=19968;node[21]=19969;node[22]=20016;node[23]=20017;
node[24]=27648;node[25]=27649;node[26]=27696;node[27]=27697;
node[28]=29184;node[29]=28185;node[30]=29232;node[31]=29233;
                                                                       90
printf("DSP Boot....\n");
descriptor = get_trck_board();
printf("Descriptor:%i\n",descriptor);
memory=get_memory_ptr(descriptor);
printf("Open the board memory at:%x\n",memory);

for(;;)
{
PLX_WRITEL(6,memory,0xc0000);
for(i=0;i<31;i++)                                                      100
        {
        value_coeff[i]=frand();
        PLX_WRITEL((flt_to_dsp(value_coeff[i])),memory, (0x2c001+i));
                /* set the coefficents */
        }

for(i=0; i<40; i++)
        {
        value_dma[i]=(unsigned int)(frand()*0xffffff);
        PLX_WRITEL(6,memory,0xc0000);                                  110
        PLX_WRITEL((value_dma[i]),memory, (0x2c100+i));
        PLX_WRITEL(5,memory,0xc0000);
PLX_WRITEL((value_dma[i]),memory, (0x2c200+i));
                /* set the dma-block */
        }
        PLX_WRITEL(6,memory,0xc0000);
        PLX_WRITEL(1,memory, 0x2cfff); /* start the dsp */

        for(waits=0;waits<10000;waits++){}
        data_dsp=dsp_to_flt(PLX_READL(memory,0x2c080));                120
        data=0;
        for(i=0;i<32;i++)
        {
```

```
        data=data+value_coeff[i]*dsp_to_flt(PLX_READL(memory,0x80000+(node[i]*2)));
        }

  if (((data / data_dsp) < 0.97) | ((data / data_dsp) > 1.03)){
    printf("error, value_dsp:%f must be:%f",data_dsp,data);
    return 0 ;}
                                                                              130
  PLX_WRITEL(5,memory,0xc0000);
  for(i=0; i<40; i++)
        {
        test=          PLX_READL(memory, (0x2c100+i));
        if (test != value_dma[i])
        {
        printf("error in DMA DSP5 no %u!!!",i);
        printf("mem: %x, link: %x",value_dma[i],test);
        }
        }                                                                     140

  PLX_WRITEL(6,memory,0xc0000);
  for(i=0; i<40; i++)
        {
        test=PLX_READL(memory, (0x2c200+i));
        if (test != value_dma[i])
        {
        printf("error in DMA DSP6 no %u!!!",i);
        printf("mem: %x, link: %x",value_dma[i],test);
        }                                                                     150
        }

        PLX_WRITEL(0,memory, 0x2cfff);
        }
  }
```

B.2 Vergleich für Pentium II

Typ: **C**-Quelltext

Name: bench3.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define frand() ((double) rand() /(RAND_MAX+1.0))

void main(void)
  {
```

```
volatile double data[1000000]; /* 1MWords of 32bit */
volatile unsigned int i,j, basis;
volatile double coeff[32],abl1=0,abl2=0,abl3=0,abl4=0,abl5=0;
volatile double stack=0;

/* filling the memory */
printf("Filling\n");
for (i=0;i<1000000;i++)
  {
  data[i]=frand()*100;
  }

printf("coeff\n");
/* filling the coefficents */
for (i=0; i<32; i++)
  {
  coeff[i]=frand();
  }
basis=0;
printf("starting...\n");
for (i=0; i<1000000;i++)
{
for(j=0;j<12;j++)
basis=frand()*250000;
{
basis=basis+2; /* the 12 coefficent are in order*/
stack=stack+data[basis]*coeff[0];basis=basis + 24;
stack=stack+data[basis]*coeff[1];basis=basis + 1128;
stack=stack+data[basis]*coeff[2];basis=basis + 24;
stack=stack+data[basis]*coeff[3];basis=basis + 35688;

stack=stack+data[basis]*coeff[4];basis=basis + 24;
stack=stack+data[basis]*coeff[5];basis=basis + 1128;
stack=stack+data[basis]*coeff[6];basis=basis + 24;
stack=stack+data[basis]*coeff[7];basis=basis + 183144;

stack=stack+data[basis]*coeff[8];basis=basis + 24;
stack=stack+data[basis]*coeff[9];basis=basis + 1128;
stack=stack+data[basis]*coeff[10];basis=basis + 24;
stack=stack+data[basis]*coeff[11];basis=basis + 35688;

stack=stack+data[basis]*coeff[12];basis=basis + 24;
stack=stack+data[basis]*coeff[13];basis=basis + 1128;
stack=stack+data[basis]*coeff[14];basis=basis + 24;
stack=stack+data[basis]*coeff[15];basis=basis + 183144;

stack=stack+data[basis]*coeff[16];basis=basis + 24;
stack=stack+data[basis]*coeff[17];basis=basis + 1128;
stack=stack+data[basis]*coeff[18];basis=basis + 24;
```

```
  stack=stack+data[basis]*coeff[19];basis=basis + 35688;

  stack=stack+data[basis]*coeff[20];basis=basis + 24;                    60
  stack=stack+data[basis]*coeff[21];basis=basis + 1128;
  stack=stack+data[basis]*coeff[22];basis=basis + 24;
  stack=stack+data[basis]*coeff[23];basis=basis + 183144;

  stack=stack+data[basis]*coeff[24];basis=basis + 24;
  stack=stack+data[basis]*coeff[25];basis=basis + 1128;
  stack=stack+data[basis]*coeff[26];basis=basis + 24;
  stack=stack+data[basis]*coeff[27];basis=basis + 35688;

  stack=stack+data[basis]*coeff[28];basis=basis + 24;                    70
  stack=stack+data[basis]*coeff[29];basis=basis + 1128;
  stack=stack+data[basis]*coeff[30];basis=basis + 24;
  stack=stack+data[basis]*coeff[31];basis=basis -701592;

  abl1=abl1+data[basis]-data[basis+24];
  abl2=abl1+data[basis]-data[basis+1152];
  abl3=abl1+data[basis]-data[basis+36888];
  abl4=abl1+data[basis]-data[basis+221184];
  abl5=abl1+data[basis]-data[basis+442368];
  }                                                                      80
  }
/* gcc: don't optimize my bench! */
printf("%f%f%f%f%f%f%f",stack,abl1,abl2,abl3,abl4,abl5);
  }
```

Anhang C

Assembler-Quelltexte SHARCs

C.1 Header-Datei

Typ: SHARC-Assembler

Name: trcks.h

```
/*************************************************************************
* name:          store(register, value)
* operation:     register = value
* register:      r0
*************************************************************************/
#define store(register, value) \
        r0 = value; \
        dm(register) = r0

/*************************************************************************
* name:          waitstate
* operation:     insert 4 waits (nop)
* register:      -
*************************************************************************/
#define waitstate nop;nop;nop;nop

/*************************************************************************
* name:          interpol(Ix, Iy, Fz)
* operation:     interpolate with coefficients (Ix) in
*                look-up-cell (Iy) -> Fz, Fz=(F13...F15)
*                Ix, Iy have the same value after the last step
* register:      r0, r1, r4, r5, r8, r9, r12
*************************************************************************/
#define interpol(basiscoeff, basislookup, retvalue) \
retvalue = 0;\
```

```
f0 = DM(basiscoeff,1);\
f4 = DM(0,basislookup);\
f8 = f0*f4, f1 = DM(basiscoeff,1);\
f5 = DM(2,basislookup);\
\                                                                    30
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(96,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(98,basislookup);\
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(3072,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(3074,basislookup);\
\                                                                    40
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(3168,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(3170,basislookup);\
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(18432,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(18434,basislookup);\
\                                                                    50
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(18528,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(18530,basislookup);\
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(21504,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(21506,basislookup);\
\                                                                    60
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(21600,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(21602,basislookup);\
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(36864,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(36866,basislookup);\
\                                                                    70
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(36960,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(36962,basislookup);\
```

```
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(39936,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(39938,basislookup);\
\                                                                    80
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(40032,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(40034,basislookup);\
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(55296,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(55298,basislookup);\
\                                                                    90
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(55392,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(55394,basislookup);\
\
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(58368,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue, f1 = DM(basiscoeff,1); \
f5 = DM(58370,basislookup);\
\                                                                   100
f9 = f1*f5, retvalue = f8+retvalue, f4 = DM(basiscoeff,1);\
f0 = DM(58464,basislookup);\
f8 = f0*f4, retvalue = f9+retvalue; f1 = DM(basiscoeff,-31);\
f5 = DM(58466,basislookup);\
f9 = f1*f5, retvalue = f8+retvalue;\
retvalue=retvalue+f9

/*************************************************************************
* name:          getdiff(Ix, Iy)
* operation:     difference for one node in                             110
*                five directions -> (Ix)..(Ix+4)
*                M0 points to the correct node, Iy to the basis of the
*                look-up-cell
*                M1...M5 are pointing to the directions
* register:      -
*************************************************************************/
#define getdiff(retindex, basislookup) \
f8=DM(M0,basislookup);\
f13=DM(M1,basislookup);\
f13=f13-f8;\                                                        120
f14=DM(M2,basislookup);\
f14=f14-f8; DM(retindex,1)=f13;\
f13=DM(M3,basislookup);\
```

```
f13=f14−f8; DM(retindex,1)=f14;\
f14=DM(M4,basislookup);\
f14=f14−f8; DM(retindex,1)=f13;\
f13=DM(M5,basislookup);\
f13=f14−f8; DM(retindex,1)=f14;\
DM(retindex,1)=f13
```

C.2 Programm für SHARC5

Typ: SHARC-Assembler

Name: loopback.asm

```
#include <def21060.h>
#include "trcks.h"

.segment /dm seg_dmda;
.endseg;
.segment /pm seg_pmco;

.global _main;
_main:
!       FUNCTION PROLOGUE: main                                          10
!       rtrts protocol, params in registers, DM stack, doubles are floats
.extern _exit;
        modify(i7,−6);
!       saving registers:
        dm(−5,i6)=r5;
        dm(−6,i6)=r9;
        r2=i0;
        dm(−7,i6)=r2;
        .def end_prologue;      .val    .;      .scl    109;    .endef;
                                                                         20
_init_dsp:
        store(0x0, 0xf400);     /* Host Bus = 32 bit, REDY = a/d */
        waitstate;
        store(0x2, 0x21ad6b41); /* no waitstates for SRAM */
        waitstate;
        bit set mode1 0x10000;  /* floating point = 32 bit */
        waitstate;
        store(LCOM, 0x3f000);   /* 80MHz for Link-Ports */
        waitstate;
        store(LAR, 0xb00);      /* assign Link4 to LB2, Link5 to LB3 */  30
        waitstate;
        store(IM5, 1);          /* store modify for DMA (LB3) */
        store(IM4, 1);          /* store modify for DMA (LB2) */
```

```
_main_loop:
                                /* store index of destination: */
        store(II5, 0xc100);     /* remember: offset of 0x20000 */
                                /* store index of source: */
        store(II4, 0xc200);                                          40
                                /* store count (length of the block!) */
        store(C4, 41);
        store(C5, 41);

                                /* START DMA SEQUENCE
                                this will be always the LAST step
                                stop DMA by clearing DEN before reprogramming */
        store(LCTL, 0x3b00);    /* 3=receive */
        dm(0x400000)=r1;        /* for diagnostic purpose */
                                                                     50
_wait_until_dma_finish:
        r0=DM(C5);
        r2=DM(C4);
        r0=r0+r2;
        r1=0;
        comp(r0, r1);
        if ne jump(PC, _wait_until_dma_finish);
        store(LCTL, 0x1900);    /* stopping DMA */

        jump (PC, _main_loop);                                       60

_end:
!       FUNCTION EPILOGUE:
        i12=dm(-1,i6);
        jump (PC, _exit) (DB);
        i7=i6;
        i6=dm(0,i6);
.endseg;
```

C.3 Programm für SHARC6

Typ: SHARC-Assembler

Name: dsp6.asm

```
#include <def21060.h>
#include "trcks.h"

.segment /dm seg_dmda;
```

```
.endseg;
.segment /pm seg_pmco;

.global  _main;
_main:
!        FUNCTION PROLOGUE: main                                          10
!        rtrts protocol, params in registers, DM stack, doubles are floats
.extern  _exit;
         modify(i7,-6);
!        saving registers:
         dm(-5,i6)=r5;
         dm(-6,i6)=r9;
         r2=i0;
         dm(-7,i6)=r2;
         .def end_prologue;      .val    .;      .scl    109;    .endef;
                                                                          20
_init_dsp:
         store(0x0, 0xf400);     /* Host Bus = 32 bit, REDY = a/d */
         waitstate;
         store(0x2, 0x21ad6b41); /* no waitstates for SRAM */
         waitstate;
         bit set mode1 0x10000;  /* floating point = 32 bit */
         waitstate;
         store(LCOM, 0x3f000);   /* 80MHz for Link-Ports */
         waitstate;
         store(LAR, 0xb00);      /* assign Link4 to LB2, Link5 to LB3 */      30
         waitstate;
         store(IM4, 1);          /* store modify for DMA (LB2) */
         store(IM5, 1);          /* store modify for DMA (LB3) */
         store(0x2cfff,0);       /* clear command bit */

_init_vector:
         store(0x20000,0x400000);
         store(0x2c021,0);
         store(0x2c022,1);
         store(0x2c023,1);                                                40
         store(0x2c024,1);
         store(0x2c025,1);
         store(0x2c026,1);

_main_loop:
_wait_until_command:
         r0=DM(0x2cfff);
         r1=0;
         comp(r0,r1);            /* wait until 0x2cfff != 0 */
                                 /* before the host programm starts the DSPs */   50
                                 /* the memory must be filled with correct values */
         if eq jump (PC,_wait_until_command);
```

```
store(II5, 0xc100);     /* store index of source (offset of 0x20000) */
store(II4, 0xc200);     /* store index of destination */

                        /* length of the block is:
                        32 coefficients, 1 basis cell
                        6 node parameters
                        the \chi^2                                    60
                        5 \chi^2/p_i = (b_i)
                        the 25 (A_{ij})
                        the measured coord (12)
                        in sum: 82 */

store(C4, 41);          /* store count (length of the block!) */
store(C5, 41);
                        /* with two link-port: count=41
                        2*2 link-ports cannot be testet
                        in loopback-modus!!!!! */                     70

                        /* START DMA SEQUENCE
                        this will be always the LAST step
                        stop DMA by clearing DEN before reprogramming */
store(LCTL, 0xb300);    b=transmit, 3=receive

                        /*******************************************
                        start of the interpolation p_1
                        *******************************************/
i1=0x400000;                                                          80
i2=0x2c001;             /* the coefficients at 0x2c001-0x2c020 */
interpol(i2, i1, f13);  /* start the interpolation, result in f13 */
DM(0x2c080) = f13;      /* insert the results at the CORRECT place */
                        /* remember: there are 12 results in summary! */

                        /*******************************************
                        get the differences
                        (the simple approach!!!!)
                        *******************************************/
M0=0;                                                                 90
M1=1;
M2=2;
M3=3;
M4=4;
M5=5;
i2=0x2c081;             /* target for the differences (0x2c081-0x2c085) */
getdiff(i2, i1);        /* store the differences */

                        /*******************************************
                        at this point, there must be calculated      100
                        the matrix (A_{ij})
                        this may cost ADDITIONAL 16 cycles/coordinate
```

```
                        ******************************************/

                        /******************************************
                        start of the interpolation p_2
                        ******************************************/
        i1=0x409000;    /* the 2nd SRAM-space */
        i2=0x2c001;     /* the coefficients at 0x2c001-0x2c021 */
        interpol(i2, i1, f13);  /* start the interpolation, result in f13 */        110
        DM(0x2c086) = f13;      /* insert the results at the CORRECT place */
                        /* remember: there are 12 results in summary! */

                        /******************************************
                        get the differences
                        ******************************************/
        M0=0;
        M1=1;
        M2=2;
        M3=3;                                                                   120
        M4=4;
        M5=5;

        i2=0x2c087;     /* target for the differences (0x2c087-0x2c08b) */
        getdiff(i2, i1);        /* store the differences */

                        /******************************************
                        at this point, there must be calculated
                        the matrix (A_{ij})
                        ******************************************/              130

_wait_until_dma_finish:
        r0=DM(C5);
        r2=DM(C4);
        r0=r0+r2;
        r1=0;
        comp(r0, r1);
        if ne jump(PC, _wait_until_dma_finish);
        store(LCTL, 0x9100);    /* stopping DMA */
        jump (PC, _main_loop);                                                  140

_end:
!       FUNCTION EPILOGUE:
        i12=dm(-1,i6);
        jump (PC, _exit) (DB);
        i7=i6;
        i6=dm(0,i6);
.endseg;
```

Anhang D

Danksagung

Das Erstellen einer Diplomarbeit ist weder eine Sache, die in wenigen Wochen oder Monaten geschehen kann, noch ist nur eine Person daran beteiligt. Vielmehr handelt es sich um einen Prozeß, an dem sehr viele Menschen über einen langen Zeitraum hinweg mitarbeiten oder diesen unterstützen. Daher möchte ich zunächst allen Menschen danken, die mit einem noch so kleinen Beitrag zum Gelingen dieser Arbeit beigetragen haben, auch wenn sie hier nicht namentlich erwähnt sind.

Zuerst möchte ich mich bei Prof. Dr. Wolfgang Kühn für die Aufnahme in die Arbeitsgruppe und die intensive Betreuung bedanken.

Den Mitgliedern der DISTO-Kollaboration möchte ich für die schöne Zeit in Saclay danken, auch wenn eine Woche viel zu kurz war.

Ein besonderes Danke geht natürlich an alle Mitarbeiter des II. Physikalischen Institutes, sowohl für die gute Athmosphäre, als auch für die vielen Hilfestellungen im Rahmen meiner Arbeit. Hier möchte ich vor allen erwähnen: Erik und Michael für das Verstehen der SHARC-Funktionen (und des Handbuches...), René für die Hilfe bei PCI-Fragen, Hans und Marc-André für das Korrekturlesen meiner Arbeit, Jim, daß er sich immer die Zeit genommen hat, meine Fragen ausführlich zu beantworten und Berengar, Carsten, Jörg und Markus für die immer gute Zusammenarbeit.

Ich danke auch denjenigen, die alles im Institut »am Laufen halten«: Anita (ohne die ich über fast alle Formalitäten gestolpert wäre), Jürgen, Marianne und Werner.

Meinen Freunden möchte ich einfach dafür danken, daß es sie gab, besonders den Leuten, die es vier Jahre geschafft haben, mit mir zusammenzuleben: Böhmi, Peter und Schmiddi.

Meinen Eltern und meinen Großmüttern möchte ich danken, daß sie mir mit ihrer Unterstützung (die keinesfalls nur materieller Natur waren) die Durchführung

meines Studiums ermöglicht haben.

Aber jetzt kommt noch das dickste Danke an meine Johanna, die nicht nur den Streß der letzten Monate ertragen, sondern immer ein gemütliches Zuhause geschaffen hat, wenn ich abends spät und erschöpft nach Hause kam. Der »Lucky-Streik-Zeit« möchte ich dafür danken, daß sie uns beide zusammengeführt hat.