**IEKP-KA/97-3**

# Ein schneller datenflußorientierter Präprozessor zum Zählen von Ionisations-Clustern in Driftkammern

*Markus Imhof*

Zur Erlangung des akademischen Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

von der Fakultät für Physik

der Universität (TH) Karlsruhe

genehmigte

DISSERTATION

von

Markus Imhof

aus Ludwigshafen am Rhein

Tag der mündlichen Prüfung: 2. 5. 1997
Referent: Prof. Dr. Th. Müller
Koreferent: Prof. Dr. H. Gemmeke

*The current state of knowledge can be summarized thus: In the beginning, there was nothing, which exploded.*

Terry Pratchett über die Kosmogenese, in *Lords and Ladies*, Corgi, 1992, ISBN 0-552-13891-6.

## Übersicht

In Experimenten der Hochenergiephysik wird zur Identifikation geladener Teilchen deren spezifischer Energieverlust pro Weglänge, $\frac{dE}{dx}$, im Inneren einer gasgefüllten Driftkammer gemessen. Die Technik des *Cluster Counting* (‘Clusterzählung’) kann im Unterschied zu dieser konventionellen $\frac{dE}{dx}$-Messung durch eine Statistik mit geringerer Varianz eine verbesserte Auflösung bei der Teilchenidentifikation erreichen. Um die Durchführbarkeit dieser Methode in der Praxis zu zeigen, wurde ein Demonstrationsmodell eines datenflußorientierten Prozessors zur Vorverarbeitung der Rohdaten (‘Präprozessor’) aufgebaut. Dieser Präprozessor hat die Aufgabe, die beim Einsatz von Cluster Counting anfallende extrem hohe Rohdatenrate ohne Verlust an physikalisch relevanter Information auf ein handhabbares Maß zu verringern. Zu diesem Präprozessor wurde ein neuer Algorithmus zur Clustersuche entwickelt, der die gesamte in den Meßwerten vorhanden Information ausnutzt. Der nötige Zeitaufwand zur Verarbeitung eines Meßdatensatzes ist jedoch noch so hoch, daß eine Anwendung an einem Hochraten-Experiment in der gegenwärtigen Konfiguration nicht sinnvoll ist. Durch eine Implementation dieses Algorithmus in einem speziellen integrierten Schaltkreis kann die benötigte Rechenzeit jedoch drastisch verringert werden, so daß der Einsatz von Cluster Counting zur Teilchenidentifikation auch bei hohen Ereignisraten möglich wird.



## Abstract

In high energy physics experiments, the specific energy loss per distance, $\frac{dE}{dx}$, is used for particle identification inside gas-filled drift chambers. *Cluster Counting* as a particle identification technique can yield a better resolution than this conventional $\frac{dE}{dx}$-method due to a decreased variance in the underlying statistics. To show the feasibility of this technique, a demonstration model of a dataflow oriented processor for preprocessing the raw data (‘preprocessor’) has been built. This preprocessor has the task of reducing the enormous raw data rate, resulting from the requirements of cluster counting, to a more manageable rate without the loss of physically significant data. In addition, a new algorithm for cluster counting to run on this preprocessor was developed, which makes use of all of the available information in the raw data. Unfortunately, the time required to process a set of data in this cofiguration is too high to use this setup in a high-rate experiment. By implementing the algorithm in a dedicated integrated circuit, processing time can be cut drastically, and therefore the use of cluster counting as a method for particle identification becomes feasible even for high event rates.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

In Experimenten der Hochenergiephysik werden die bei Stößen zwischen hochenergetischen Hadronen oder Leptonen auftretenden Wechselwirkungen untersucht. Typische Energien liegen hier im Bereich vom 0.5 GeV pro Strahl für $e^+/\ e^-$-Collider zur Untersuchung der $\Phi$-Resonanz und mehreren TeV in geplanten $p\bar{p}$-Collidern für die Suche nach neuen Teilchen im Rahmen von Erweiterungen des Standardmodells.
Bei einem solchen Stoß zwischen zwei hochenergetischen Teilchen werden bis zu mehreren hundert Sekundärteilchen erzeugt, die zur Analyse der Wechselwirkung nachgewiesen, vermessen und identifiziert werden müssen. Diese Sekundärteilchen können nach der Art ihrer Wechselwirkung mit dem Material des Detektors zunächst in drei Gruppen unterteilt werden:

- geladene Teilchen
- Photonen
- ungeladene Hadronen.

Geladene Teilchen können über die elektromagnetische Wechselwirkung mit dem Material des Detektors vermessen und identifiziert werden. Abhängig vom Ziel des Experiments und dem gewählten Aufbau des Detektors können hier verschiedene Meßmethoden zum Einsatz kommen.
Eine dieser Methoden bedient sich des spezifischen Energieverlustes pro Weglänge, $\frac{d\mathrm{E}}{d\mathrm{x}}$, dieser Teilchen im Inneren einer gasgefüllten Driftkammer, in der auch die Spuren der geladenen Teilchen vermessen werden. Bei Cluster Counting handelt es sich um eine Methode zur Auslese dieser Driftkammern, die gegenüber der Messung von $\frac{d\mathrm{E}}{d\mathrm{x}}$ ein besseres statistisches Verhalten aufweist und damit eine verbesserte Teilchenidentifikation ermöglichen sollte. In früheren Untersuchungen zu dieser Methode kam eine spezielle Driftkammergeometrie zum Einsatz [1]. Die technische Entwicklung erlaubt mittlerweile den Einsatz dieser Methode in konventionellen Driftkammern, wenn eine verbesserte Teilchenidentifikation, zum Beispiel bei Präzisionsexperimenten der Hochenergiephysik, benötigt wird. Eine weitergehende Diskussion des Vorteiles von Cluster Counting gegenüber der bisherigen Methode der Messung des Energieverlustes findet sich in Abschnitt 2.1.
Der Nachweis von Photonen erfolgt über deren elektromagnetische Wechselwirkung mit dem Material eines sogenannten Kalorimeters. Hierbei entsteht ein charakteristischer Schauer von Sekundärteilchen, über den sich Informationen über das auslösende Photon gewinnen lassen.
Auch die ungeladenen Hadronen werden mittels eines Kalorimeters nachgewiesen und vermessen. In diesem Fall wird die starke Wechselwirkung der Teilchen mit den Kernen des Kalorimetermate-

rials ausgenutzt. Hierdurch werden wieder charakteristische Sekundärschauer ausgelöst, über die die ungeladenen Hadronen nachgewiesen und vermessen werden können.

## 1.1 Teilchenidentifizierung in Experimenten der Hochenergiephysik

Um verschiedene Arten von Sekundärteilchen voneinander zu unterscheiden, kommt in der Praxis zunächst einmal deren Ruhemasse als Unterscheidungsparameter in Betracht. Andere Parameter sind im Experiment zunächst nur schwer zugänglich (wie der Spin), oder auch zu vieldeutig (wie die Ladung). Zur Bestimmung der Ruhemasse eines sich bewegenden, geladenen Teilchens sind folgende vier Parameter zugänglich:

- Impuls
- Geschwindigkeit
- Gesamtenergie $E_{\mathrm{tot}}$
- Spezifischer Energieverlust $\frac{d\mathrm{E}}{d\mathrm{x}}$

Um aus diesen Meßwerten die Teilchensorte zu bestimmen, können die folgenden drei Beziehungen ausgenutzt werden:

- Relativistische Energie-Impuls-Beziehung [2] :

$$E_{\mathrm{tot}}^2 = p^2c^2 + m_0^2c^4 \tag{1.1}$$

- Relativistischer Geschwindigkeits-Impuls-Zusammenhang :

$$p = \gamma\beta mc \tag{1.2}$$

- Spezifischer Energieverlust $\frac{d\mathrm{E}}{d\mathrm{x}}$: nach der Bethe-Bloch-Formel ergibt sich der spezifische Energieverlust $\frac{d\mathrm{E}}{d\mathrm{x}}$ beim Durchgang eines geladenen Teilchens durch Materie zu [3] :

$$-\frac{d\mathrm{E}}{d\mathrm{x}}\bigg|_{T<T_{\mathrm{cut}}} = Kz^2\frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_ec^2\beta^2\gamma^2T_{\mathrm{upper}}}{I^2} - \frac{\beta^2}{2}\left(1+\frac{T_{\mathrm{upper}}}{T_{\mathrm{max}}}\right) - \frac{C}{Z} - \frac{\delta}{2}\right] \tag{1.3}$$

  mit folgenden Parametern:

  - $T$ : kinetische Energie in MeV
  - $T_{cut}$ : Abschneideenergieübertrag, bis zu dem $\frac{d\mathrm{E}}{d\mathrm{x}}$ ermittelt werden soll
  - $K$ : $4\pi N_A r_e^2 m_e c^2$, Konstante der Bethe-Bloch-Formel
  - $z$ : Ladung des betreffenden Teilchens, in Einheiten der Elementarladung $e$
  - $Z$ : Kernladungszahl des Mediums, durch das sich das Teilchen bewegt
  - $A$ : Atommasse des Mediums
  - $m_e c^2$ : Masse
  - $T_{\mathrm{upper}}$ : $\max(T_{\mathrm{cut}}, T_{\mathrm{max}})$

– $T_{\max}$ : maximaler Energieübertrag für ein punktartiges geladenes Teilchen mit Masse $m$ und Impuls $\beta\gamma$.

– $I$ : mittleres Ionisationspotential des Mediums, für Elemente schwerer als Sauerstoff ist $\frac{I}{Z} \approx 10\,eV$.

– $\frac{C}{Z}$ : Korrekturfaktor für den Schaleneffekt, Abhängig vom Medium und der Ladung des betreffenden Teilchens

– $\frac{\delta}{2}$ : Korrekturfaktor für den Dichteeffekt

Die Verknüpfung der ersten beiden dieser Beziehungen (1.1 und 1.2) ergibt einen Zusammenhang zwischen den Meßwerten für Energie und Impuls und der daraus ermittelbaren Masse des beobachteten Teilchens. Die zweite Beziehung zusammen mit der dritten (1.2 und 1.3) ergibt eine komplexe Funktion für die Abhängigkeit zwischen dem spezifischen Energieverlust $\frac{\mathrm{dE}}{d\mathrm{x}}$ und dem normierten Impuls $\beta\gamma = \frac{pc}{m}$ (Abbildung 1.1).

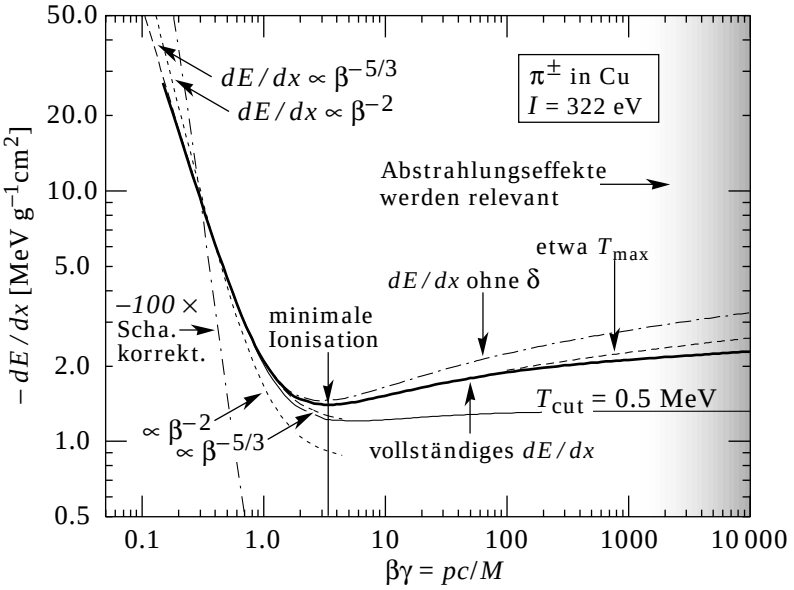



Abbildung 1.1: Spezifischer Energieverlust $\frac{\mathrm{dE}}{d\mathrm{x}}$ in Abhängigkeit vom normierten Impuls $\beta\gamma$ von Pionen in Kupfer, aus [4]

Abhängig vom Impuls der zu identifizierenden Teilchen sind unterschiedliche Meßmethoden praktikabel. Im Folgenden soll von der Identifizierung minimal ionisierender Teilchen ausgegangen werden, das heißt, von Teilchen mit normierten Impulsen $\beta\gamma$ im Bereich von etwa 0.3 bis 3.

### 1.1.1 Impulsmessung[1]

Zur Messung des Impulses eines geladenen Teilchens wird der Krümmungsradius seiner Bahn in einem äußeren Magnetfeld gemessen. Diese Bahn beschreibt im allgemeinen eine Helix im Raum

[1] Aus [3], Kapitel 24.8

mit Radius $R$ und Anstellwinkel $\lambda$. In der Ebene senkrecht zu $\vec{B}$ sind Impuls $p$ (in $\frac{\mathrm{GeV}}{\mathrm{c}}$) und Kurvenradius eines Teilchens mit Ladung $z \cdot e$ folgendermaßen miteinander verknüpft [3]:

$$p \cos \lambda = 0.3 \; z \; B \; R. \tag{1.4}$$

Diese Messung ist allerdings mit einem Fehler behaftet. Nach [5, 3] kann der Fehler der Krümmung $k \equiv \frac{1}{R}$ bei der Vermessung vieler Punkte auf der Bahn eines geladenen Teilchens in einem gleichförmigen Magnetfeld angenähert werden durch:

$$(\delta k)^2 = (\delta k_{\mathrm{res}})^2 + (\delta k_{\mathrm{ms}})^2 \tag{1.5}$$

Dabei sind

$\delta k =$ Fehler der Krümmung

$\delta k_{\mathrm{res}} =$ Fehler der Krümmung durch die endliche Auflösung der Messung

$\delta k_{\mathrm{ms}} =$ Fehler der Krümmung durch Mehrfachstreuung

Der Gesamtfehler folgt dabei annähernd einer Gaussverteilung.
Für eine ausreichend große Zahl gleichmäßig verteilter Messungen entlang einer Bahn in einem gleichförmigen Medium kann man für den Fehler $\delta k_{\mathrm{res}}$ folgendes ansetzen [5, 3]:

$$\delta k_{\mathrm{res}} = \frac{\epsilon}{L'^2} \sqrt{\frac{720}{N+4}} \tag{1.6}$$

Hierbei sind

$N =$ Anzahl der Meßpunkte längs der Bahn

$L' =$ Projektion der Bahnlänge auf die Krümmungsebene

$\epsilon =$ Meßfehler in jedem Punkt senkrecht zur Bahn

Falls auf den Anfangspunkt der Bahn ein *vertex constraint* angewandt wird, reduziert sich der Koeffizient unter der Wurzel in obiger Gleichung auf 320.
Der Beitrag durch Mehrfachstreuung kann durch

$$\delta k_{\mathrm{ms}} \approx \frac{(0.016)(\frac{\mathrm{GeV}}{\mathrm{c}})z}{Lp\beta \cos^2 \lambda} \sqrt{\frac{L}{X_0}} \tag{1.7}$$

angenähert werden [5, 3]. In dieser Gleichung beschreibt $L$ die gesamte Spurlänge innerhalb des Mediums und $X_0$ die Strahlungslänge diese Mediums in Meter.

### 1.1.2 Messung des spezifischen Energieverlustes $\frac{\mathrm{dE}}{\mathrm{dx}}$

In vielen Experimenten der Teilchenphysik wird die Messung des spezifischen Energieverlustes pro Weglänge $\frac{\mathrm{dE}}{\mathrm{dx}}$ in einer Driftkammer vorgenommen. Ein geladenes Teilchen erzeugt hierin auf dem Weg durch das Füllgas der Kammer durch Ionisation Elektron-/Ion-Paare (Bild 1.2). Die positiv geladenen Ionen wandern im elektrischen Feld einer Driftzelle langsam zu den Kathodendrähten, während die Elektronen zum Anodendraht driften. Bei verschiedenen hier gebräuchlichen

Gasmischungen stellt sich bei den in den Driftzellen vorliegenden Feldstärken ein Gleichgewicht zwischen der Beschleunigung der Elektronen durch das elektrische Feld und der Bremsung durch die Kollisionen mit den Gasatomen (oder Molekülen) ein [6], so daß die Driftgeschwindigkeit der Elektronen konstant wird[2].

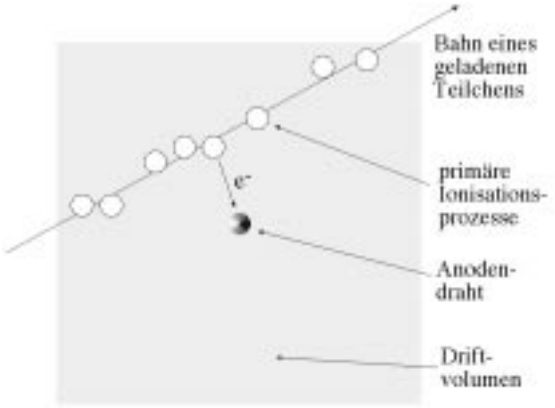



Abbildung 1.2: Schematische Darstellung einer Driftzelle

In der unmittelbaren Umgebung des Anodendrahtes werden die Elektronen durch das hohe elektrische Feld dann ausreichend beschleunigt, um selbst wieder durch Stoßionisation aus dem Kammergas Elektron-/Ion-Paare freisetzen zu können. Dies führt zu einem Lawineneffekt, durch den die ursprünglich durch das nachzuweisende Teilchen freigesetzte Ladung um einen Faktor $10^3$ bis $10^6$ verstärkt wird. Durch die Bewegung dieser Ladung im Feld der Driftzelle wird im Anodendraht ein elektrischer Strom induziert, welcher mit einer geeigneten Elektronik verstärkt und gemessen wird. Der Beginn dieses Signales ergibt ein Maß für den kleinsten Abstand der Teilchenspur zum Anodendraht, während der innerhalb eines gewissen Zeitfensters integrierte Strom eine Ladung ergibt, die proportional zu der Ladung ist, die durch den Durchgang des Teilchens durch die Driftzelle ursprünglich freigesetzt wurde. Unter der Annahme, daß diese ursprünglich freigesetzte Ladung proportional zur vom Teilchen innerhalb der Driftzelle deponierten Energie ist, erhält man somit ein Maß für diesen Energieverlust.
Die vom zu messenden Teilchen pro Wegstrecke in der Kammer deponierte Energie wird durch die Bethe-Bloch-Formel (1.3) gegeben. In der angegebenen Darstellung beschreibt diese Formel den Energieverlust pro Weglänge unterhalb einer maximal übertragenen Energie $T_{\mathrm{cut}}$. Die statistische Verteilung der Meßwerte für den spezifischen Energieverlust $\frac{\mathrm{dE}}{\mathrm{dx}}$ folgt einer sogenannten Landau-Kurve. Diese zeigt, wie auch andere Modelle [7], einen charakteristische Ausläufer hin zu hohen Energieverlusten ('Landau-Schwanz'). Dieser Ausläufer beschreibt eine kleine Anzahl von Stößen mit hohem Energieübertrag. Das in diesen Stößen freigesetzte Elektron ('$\delta$-Strahlung') kann zu den Meßfehlern beitragen, wenn es sich aus dem Volumen der Driftzelle entfernt. Wird dieser 'Schwanz' mit in der Auswertung berücksichtigt, so wird die Varianz der Verteilung drastisch erhöht.
In der Praxis wird der Energieverlust, wie in Gleichung 1.3 beschrieben, nur bis zu einem oberen Grenzwert berücksichtigt. Dazu wird für die Auswertung ein sogenannter 'Abgeschnittener Mittelwert' ('truncated mean') verwendet. Hierbei werden nur die 40…60 % der Meßwerte mit den kleinsten Amplituden in die Auswertung mit einbezogen, um die durch den langen 'Schwanz' der

[2]Die Driftgeschwindigkeit der Elektronen liegt hier im Bereich von wenigen cm/$\mu$s.

Landau-Verteilung verursachten Streuungen zu reduzieren. Unter diesen Umständen ergibt sich zum Beispiel für Teilchen der Ladung $e$ in einem mit Argon gefüllten Detektor eine Halbwertsbreite der Energieverlustverteilung von

$$\left.\frac{d\mathrm{E}}{d\mathrm{x}}\right|_{\mathrm{FWHM}} = 0.96 N^{-0.46} (xp)^{-0.32} \left.\frac{d\mathrm{E}}{d\mathrm{x}}\right|_{\mathrm{maximum}} . \tag{1.8}$$

Hierbei ist $N$ die Zahl der Zellen, $x$ die Dicke einer Zelle (in cm) und $p$ der Gasdruck (in Atmosphären). Diese Formel ergibt sich aus einer Anpassungsrechnung an experimentell ermittelte Werte [8, 7].

### 1.1.3 Gesamtenergiemessung

Zur Messung der Gesamtenergie eines Teilchens werden sogenannte Kalorimeter verwendet. Die Messung basiert darauf, daß ein hochenergetisches Primärteilchen beim Durchgang durch Materie entweder über elektromagnetische (Elektromagnetisches Kalorimeter) oder starke (Hadronisches Kalorimeter) Wechselwirkung niederenergetischere Sekundärteilchen erzeugt. Diese Sekundärteilchen erzeugen in weiteren Wechselwirkungen weitere Teilchen und bauen so eine elektromagnetische bzw. hadronische Kaskade auf.
Das Ziel eines Kalorimeters ist es, alle erzeugten Sekundärteilchen im empfindlichen Volumen aufzuhalten und nachzuweisen. Hierbei lassen sich zwei Ansätze unterscheiden: Sampling Kalorimeter und homogene (aktive) Kalorimeter.
In einem Sampling Kalorimeter wechseln sich das Absorbermedium und das Nachweismedium ab. Dies geschieht üblicherweise durch die Einbettung von Szintillatormaterial in Lagen ('Sandwich', 'Schaschlik') oder als Fibern ('Spaghetti') in ein Schwermetall (Eisen, Blei, Uran).
Ein homogenes Kalorimeter besteht aus einem transparenten Material hoher Dichte, das gleichzeitig als Absorber und als Nachweismedium dient. Hier kommen unter anderem Bleiglas oder Kristalle aus NaI(Tl) oder BGO zum Einsatz. Dieser Kalorimetertyp bietet eine höhere Energieauflösung als ein Sampling Kalorimeter, ist aber auch teurer.
Ein andere Nachteil homogener Kalorimeter ist, daß sie nicht als kompensierte Kalorimeter ausgeführt werden können. Das heißt, ein einfallendes stark wechselwirkendes Teilchen erzeugt – bei gleicher Energie – ein kleineres Signal als ein elektromagnetisch wechselwirkendes Primärteilchen. In einem Sampling Kalorimeter kann dieser Unterschied durch Variation des Verhältnisses zwischen Absorber und Nachweismedium ausgeglichen werden.
Im Prinzip könnte nun über Gleichung 1.1 aus der so gemessenen Gesamtenergie eines Teilchens und dessen Impuls die Masse dieses Teilchens bestimmt werden. In der Praxis scheitert dies daran, daß die Energieauflösung der Kalorimeter für eine zuverlässige Teilchenbestimmung auf diesem Wege nicht ausreichend ist. Durch die unterschiedliche Schauerentwicklung können aber zumindest Elektronen und Hadronen im Kalorimeter voneinander unterschieden werden. Indirekt können hier auch Myonen als diejenigen Teilchen identifiziert werden, die in der Lage sind, das Kalorimeter vollständig zu durchqueren.

### 1.1.4 Geschwindigkeitsmessung

Die Geschwindigkeit relativistischer Teilchen läßt sich auf mehrere Arten bestimmen. Zunächst einmal kann man direkt die Flugzeit ('TOF'[3]) des Teilchens vom Punkt der Entstehung bis zu

[3] **T**ime **o**f **F**light

einem geeigneten Detektor messen. Für reale Detektorgrößen im Bereich von wenigen Metern ergeben sich hier Flugzeiten in der Größenordnung weniger Nanosekunden. Die Auflösung aktueller Zeitmeßgeräte ('TDCs'[4]) liegt im Bereich einiger Zehn bis etwa Einhundert Picosekunden, womit die relative Zeitauflösung bei dieser Meßmethode im Bereich von etwa $10^{-2}$ liegt. Hiermit wird diese Meßmethode nur für 'langsame' Teilchen praktikabel. Bei hochrelativistischen Teilchen ist die erzielbare Auflösung in der Zeitmessung nicht ausreichend.
Weitere Fehlerquellen in diesem Bereich sind die Vermessung der im Magnetfeld gekrümmten Spur (s.o.) sowie die Bestimmung des Ursprungsortes des interessierenden Teilchens. In beiden Fällen hängt die erzielbare Genauigkeit vom jeweiligen Aufbau des Detektors ab.
Eine andere, häufig verwendete Methode der Geschwindigkeitsmessung bedient sich des sogenannten Čerenkov-Effektes.

**Čerenkov-Effekt und Übergangsstrahlung [3, 9, 10, 11]**

Bewegt sich ein geladenes Teilchen in einem Medium schneller als mit Lichtgeschwindigkeit, oder passiert ein solches Teilchen die Grenzfläche zwischen zwei Medien mit unterschiedlichem Brechungsindex, so strahlt dieses Teilchen Photonen ab. Im ersten Fall spricht man von Čerenkov-Licht, im zweiten von Übergangsstrahlung.
Die beiden wichtigsten Parameter für den Čerenkov-Effekt sind die Grenzgeschwindigkeit, ab der dieser Effekt stattfindet, sowie der Öffnungswinkel des Lichtkegels. In einem Medium mit Brechungsindex $n$ gelten für ein Teilchen der Geschwindigkeit $\beta c$ folgende Beziehungen für den Öffnungswinkel $\theta_C$ und die Schwellengeschwindigkeit $\beta_t$:

$$\begin{aligned} \theta_C &= \arccos \frac{1}{n\beta} \\ \beta_t &= \frac{1}{n} \end{aligned} \tag{1.9}$$

Für die Anwendung des Čerenkov-Effekts in einem Teilchendetektor ist die Zahl der detektierten Photoelektronen ebenfalls von Bedeutung. Sie wird gegeben durch [3]:

$$N_{\mathrm{p.e.}} = L \frac{\alpha^2 z^2}{r_e m_e c^2} \int \epsilon_{\mathrm{coll}}(E) \epsilon_{\mathrm{det}}(E) \sin^2 \theta_C(E) dE \tag{1.10}$$

mit

- L: Pfadlänge im Detektor
- $\epsilon_{\mathrm{coll}}$ : Wirkungsgrad der Sammlung der erzeugten Photonen auf dem Photodetektor ('collection efficiency')
- $\epsilon_{\mathrm{det}}$ : Wirkungsgrad des Photonennachweises im Photodetektor

Im allgemeinen sind sowohl die Wirkungsgrade $\epsilon_{\mathrm{coll}}$ und $\epsilon_{\mathrm{det}}$ als auch der Abstrahlwinkel $\theta_C$ (oder genauer gesagt der Brechungsindex $n$) von der Energie $E$ des abgestrahlten Photons abhängig.

**Schwellen-Čerenkov-Detektor** Hierbei handelt es sich im wesentlichen um einen Behälter, der mit einem – im relevanten Bereich – transparenten Medium gefüllt ist. Durch die Wahl des

[4] **T**ime to **D**igital **C**onverter, Zeit-Digital-Wandler

Mediums sowie (bei Gasen) des Drucks läßt der Brechungsindex $n$, und damit auch die Schwellengeschwindigkeit $\beta_t$, auf einen gewünschten Wert einstellen. Durch eine geeignete Optik werden dann die entstehenden Photonen auf einen Photodetektor gelenkt.

Ein solcher Detektor liefert eine einfache Ja/Nein-Entscheidung. Ein hindurchfliegendes Teilchen hat entweder eine Geschwindigkeit $\beta > \frac{1}{n}$ und emittiert Photonen, oder eben nicht. Durch den Einsatz mehrerer Detektoren hintereinander kann eine Einteilung in Geschwindigkeitsbereiche vorgenommen werden.

**Ring Imaging Čerenkov Detector** Ein RICH, oder Ring Imaging Čerenkov Detector[5] ist vom Aufbau her deutlich aufwendiger. Bei einem solchen Detektor werden die von einem Teilchen unter dem Winkel $\theta_C$ abgestrahlten Photonen als Ring auf einen ortsauflösenden Photodetektor abgebildet. Speziell bei den üblicherweise zylindersymmetrischen Detektoren von Colliderexperimenten erfordert dies einen erheblichen optischen Aufwand. Der Vorteil dieses Konzepts ist, daß der Winkel $\theta_C$ über Gleichung 1.9 direkt mit der Geschwindigkeit des Teilchens gekoppelt ist. Damit läßt sich der aus diesem Öffnungswinkel resultierende Durchmesser des abgebildeten Rings direkt einer Teilchengeschwindigkeit zuordnen.

**Übergangsstrahlungsdetektoren** Beim Übergang zwischen dem Vakuum und einem Medium mit der Plasmafrequenz $\omega_p$ wird von einem geladenen Teilchen die Energie

$$I = \alpha z^2 \gamma \hbar \omega_p / 3 \tag{1.11}$$

abgestrahlt. Hierbei ist

$$\hbar\omega_p = \sqrt{4\pi N_e r_e^3} \frac{m_e c^2}{\alpha} \tag{1.12}$$

wobei $N_e$ die Elektronendichte im Medium beschreibt.

Das Strahlungsspektrum ist für kleine Energien logarithmisch divergent und nimmt für Energien $\hbar\omega > \gamma\hbar\omega_p$ schnell ab. Etwa die Hälfte der Energie wird im Bereich zwischen $0.1\gamma\hbar\omega_p$ und $\gamma\hbar\omega_p$ abgestrahlt. Genauere Rechnungen führen zu folgendem Ausdruck für die Zahl der Photonen, die mit einer Energie $\hbar\omega > \hbar\omega_0$ abgestrahlt werden:

$$N_\gamma(\hbar\omega > \hbar\omega_0) = \frac{\alpha z^2}{\pi} \left[ \left( \ln \frac{\gamma\hbar\omega_p}{\hbar\omega_0} - 1 \right)^2 + \frac{\pi^2}{12} \right]. \tag{1.13}$$

Für die Praxis bedeutet das, daß Übergangsstrahlungsdetektoren nur für Teilchen mit sehr hohem $\gamma$ ($\beta\gamma \gtrsim 1000$) sinnvoll werden. Verwendet werden sie bisher, um Elektronen von Pionen bei Impulsen $p \gtrsim 1$ GeV/c zu unterscheiden.

### 1.1.5 Vergleich der Methoden

Ein direkter Vergleich der verschiedenen Methoden zur Teilchenidentifikation ist nahezu unmöglich. Zunächst einmal ist jede dieser Methoden in der Praxis für einen begrenzten Bereich in $\gamma$ optimiert worden.
Ein Beispiel dafür zeigt die Abbildung 1.3. Hier sind die drei Teilchenidentifikationsmechanismen des Experiments DELPHI am LEP-Speicherring des CERN im Vergleich dargestellt (die Daten entstammen einer Simulation). Im unteren Abschnitt des dargestellten Energiebereichs kann

[5] wörtlich: Ringabbildender Čerenkov Detektor

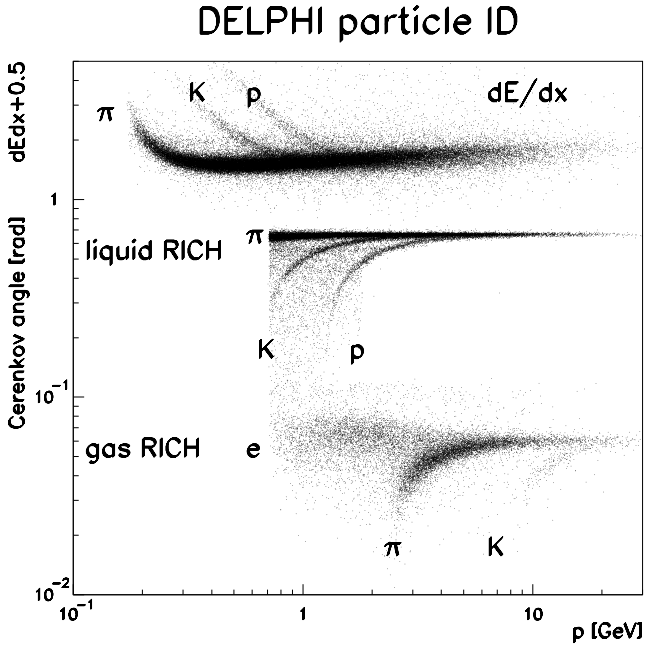



Abbildung 1.3: Teilchenidentifikation bei DELPHI [12]. Die y-Achse der $\frac{dE}{dx}$-Messung ist skaliert und verschoben, um den Einsatz der drei Meßmethoden in den unterschiedlichen Energiebereichen zu demonstrieren.

eine Teilchenidentifikation über die Messung des spezifischen Energieverlusts in einer Driftkammer vorgenommen werden. Im mittleren Energiebereich erlaubt diese Messung keine Trennung mehr zwischen den unterschiedlichen Teilchensorten, dafür kommt hier ein flüssigkeitsgefüllter Čerenkov-Detektor ('liquid RICH') zum Einsatz. Im oberen Energiebereich schließlich liefert ein gasgefüllter Čerenkov-Detektor ('gas RICH') die nötige Information zur Teilchenidentifizierung. Im unteren Impulsbereich kann zusätzlich noch eine Flugzeitmessung zum Einsatz kommen, bei großen Impulsen ein Übergangsstrahlungsdetektor.

## 1.2 Der KLOE-Detektor bei DAΦNE

Auf welche Weise die Teilchenidentifikation am sinnvollsten erfolgt, hängt also stark vom jeweiligen Experiment ab. Beim KLOE-Experiment, das für den Einsatz am Speicherring DAΦNE in Frascati vorgesehen ist [13, 14, 15, 16], wird für die Präzisionsmessung der CP-Verletzung im Kaonenzerfall eine sehr zuverlässige Unterscheidung zwischen Pionen und Myonen verlangt. Hierfür wurde Cluster Counting als eine mögliche Alternative zur herkömmlichen $\frac{dE}{dx}$-Messung untersucht. Nach den Ergebnissen der Testmessungen an den Prototypen des Kalorimeters und der Driftkammer wurde allerdings entschieden, daß auf eine Teilchenidentifizierung mittels einer Energieverlustmessung in der Driftkammer verzichtet werden kann, da die Teilchenidentifizierung anhand der Impulsmessung in der Driftkammer sowie der Energie- und Flugzeitmessung im ortsauflösenden Kalorimeter ausreichend sicher zu sein scheint.
Im Rahmen der ersten Untersuchungen zu Cluster Counting kristallisierte sich jedoch heraus, daß hiermit eine Methode zur Teilchenidentifizierung zur Verfügung steht, mit der sich eine gegen-

über der konventionellen $\frac{dE}{dx}$-Messung deutlich verbesserte Auflösung erreichen läßt. Aus diesem Grund wurden in einer Zusammenarbeit, zunächst zwischen der Universität Karlsruhe, dem INFN Lecce, dem INFN Rom und der Università di Roma II, bei weiteren Untersuchungen noch zwischen der Universität Karlsruhe und der Università di Roma II, Untersuchungen zum Potential und der Anwendbarkeit dieser Technik angestellt, in deren Rahmen auch die vorliegende Arbeit entstand.

### 1.2.1 Der DAΦNE-Speicherring

Der DAΦNE-Speicherring besteht aus zwei getrennten Ringen für Elektronen und Positronen. Der Durchmesser auf der langen Achse beträgt 32.5 m, auf der kurzen Achse 23.3 m. Die beiden Ringe kreuzen sich an zwei gegenüberliegenden Punkten. Die eine dieser beiden Wechselwirkungszonen ist für das Experiment FINUDA vorgesehen, die zweite für KLOE.
Die gesamte Anlage besteht aus einem Linearbeschleuniger, einem Akkumulatorring sowie dem DAΦNE-Doppelspeicherring. Die Endenergie des Linearbeschleunigers beträgt 510 MeV, so daß bei den Kollisionen im Schwerpunktsystem 1.020 GeV zur Verfügung stehen. Die Designluminosität bei dieser Energie beträgt $\mathcal{L} = 5.2 \cdot 10^{32} \frac{1}{\mathrm{cm^2 s}}$ [17] bei 120 Teilchenpaketen. Die Hauptreaktionen bei dieser Energie sind $e^+e^- \rightarrow e^+e^-$ (Bhabha-Streuung, hauptsächlich unter kleinen Winkeln) sowie, mit einem Wirkungsquerschnitt von $4.4\ \mu b$, $e^+e^- \rightarrow \Phi$. Die wichtigsten Zerfallskanäle des Φ sind in der folgenden Tabelle aufgeführt.

| $\Phi \rightarrow$ | Anteil [%] | $p_{\max}\,[\frac{\mathrm{MeV}}{\mathrm{c}}]$ | $E_{\mathrm{kin,max}}$ [MeV] |
|---|---|---|---|
| $K^+K^-$ | $49.1 \pm 0.8$ | 127.9 | 16.3 |
| $K_SK_L$ | $34.4 \pm 0.7$ | 110 | 12.0 |
| $\rho\pi$ | $12.9 \pm 0.7$ | 183 | 21.5/90.6 |
| $\pi^+\pi^-\pi^0$ | $2.4 \pm 0.9$ | 462 | 346.3 |
| $\eta\gamma$ | $1.28 \pm 0.06$ | 362 | 108.9 |

Tabelle 1.1: Die wichtigsten Zerfallskanäle des Φ(1020) [16]

### 1.2.2 Der KLOE-Detektor

Das Hauptziel des KLOE-Detektors ist die Präzisionsmessung der direkten CP-Verletzung im Zerfall neutraler Kaonen. Wie in Tabelle 1.1 zu sehen ist, zerfallen die bei DAΦNE erzeugten Φs zu über 34 % in neutrale Kaonen.

#### Physik bei KLOE

Sollte die Natur unter Ladungs- und Paritätstransformation (kurz CP) symmetrisch sein, mischen sich die beiden *flavour*-Eigenzustände des Kaons, $K^0$ und $\bar{K}^0$, über die schwache Wechselwirkung zu den beiden Masse-Eigenzuständen $K_1$ und $K_2$:

$$|K_1\rangle \;=\; \frac{1}{\sqrt{2}}\left(\left|K^0\right\rangle + \left|\bar{K}^0\right\rangle\right) \approx |K_S\rangle \tag{1.14}$$

$$|K_2\rangle \;=\; \frac{1}{\sqrt{2}}\left(\left|K^0\right\rangle - \left|\bar{K}^0\right\rangle\right) \approx |K_L\rangle \tag{1.15}$$

Die CP-Erhaltung fordert dann, daß das kurzlebigere $K_1$ ($K_S$) in zwei Pionen, das langlebigere $K_2$ ($K_L$) in drei Pionen zerfällt. 1964 wurde aber zum ersten Mal der unter CP-Erhaltung verbotene Zerfall des $K_2$ in zwei Pionen beobachtet [18].
Dieses Verhalten kann dadurch erklärt werden, daß es sich bei den beobachteten Teilchen $K_S$ und $K_L$ nicht um die CP-Eigenzustände $K_1$ und $K_2$ des Kaonensystems handelt, sondern daß $K_L$ und $K_S$ aus Mischungen der beiden CP-Eigenzustände bestehen:

$$|K_L\rangle \;=\; (|K_2\rangle + \epsilon\,|K_1\rangle) \tag{1.16}$$
$$|K_S\rangle \;=\; (|K_1\rangle + \epsilon\,|K_2\rangle) \tag{1.17}$$

Die genauere Berechnung der Zerfälle der Kaonen führt schließlich zur Beschreibung zweier beobachtbarer Größen, der Verhältnisse der Amplituden der Zerfälle der Kaonen in geladene beziehungsweise neutrale Pionen. Diese beiden Verhältnisse werden mit zwei Parametern, $\epsilon$ und $\epsilon'$, beschrieben [19, 20, 21]:

$$\eta_{\pm} \equiv \frac{A\,(K_L \to \pi^+\pi^-)}{A\,(K_S \to \pi^+\pi^-)} \equiv \epsilon + \epsilon' \tag{1.18}$$

und

$$\eta_{00} \equiv \frac{A\,(K_L \to \pi^0\pi^0)}{A\,(K_S \to \pi^0\pi^0)} \equiv \epsilon - 2\epsilon' \tag{1.19}$$

wobei A die Amplitude des jeweiligen Zerfallsweges bezeichnet.
Der Parameter $\epsilon$ beschreibt hier die 'indirekte' CP-Verletzung durch nichtreziproke Übergänge zwischen den beiden neutralen Kaonen, $K^0 \not\Leftrightarrow \bar{K}^0$, während der Parameter $\epsilon'$ die 'direkte' CP-Verletzung im Zerfall der Kaonen beschreibt.
Für die Messung von $\epsilon$ und $\epsilon'$ wird unter anderem das sogenannte Doppelverhältnis

$$\mathcal{R} = \frac{\mathcal{R}^{\pm}}{\mathcal{R}^0} = \frac{\Gamma\,(K_L \to \pi^+\pi^-)\,\Gamma\,(K_S \to \pi^0\pi^0)}{\Gamma\,(K_S \to \pi^+\pi^-)\,\Gamma\,(K_L \to \pi^0\pi^0)} = \frac{N_L^{\pm}/N_S^{\pm}}{N_L^0/N_S^0} \approx 1 + 6\Re\frac{\epsilon'}{\epsilon} \tag{1.20}$$

benutzt. Hierbei sind die $\Gamma$ die Zerfallsbreiten (inverse Lebensdauern) der entsprechenden Kanäle, die N entsprechen der Anzahl der Zerfälle im gewählten Kanal.
Eine direkte Verletzung von CP sollte sich in einem von 1 abweichenden Doppelverhältnis äußern. Es gibt derzeit hierzu zwei experimentelle Ergebnisse :

$$\Re\frac{\epsilon'}{\epsilon} = (23 \pm 6.5)10^{-4} \tag{1.21}$$

von der Kollaboration NA31 am CERN sowie

$$\Re\frac{\epsilon'}{\epsilon} = (7.4 \pm 5.9)10^{-4} \tag{1.22}$$

von der Kollaboration E731 am FNAL [22].
Das Ziel von KLOE ist es, $\Re\frac{\epsilon'}{\epsilon}$ mit einer relativen Genauigkeit von $1 \cdot 10^{-4}$ zu bestimmen.
Um diese Genauigkeit erreichen zu können, müssen diejenigen Kaonenzerfälle, deren Signatur im Detektor derjenigen der Zerfälle in Gleichung 1.20 ähnelt, erkannt und bei der Auswertung zurückgewiesen werden können. Hierzu gehören, neben dem Zerfall des $K_L$ in drei neutrale Pionen,

vor allem die Zerfälle $K_L^0 \to \pi e \nu$ sowie $K_L^0 \to \pi \mu \nu$, bei denen das Elektron beziehungsweise das Myon als Pion fehlidentifiziert wird. Der erstere dieser beiden Zerfälle kann in einem hermetischen[6] Detektor durch die Kinematik des Zerfalls vom Zerfall des Kaons in zwei Pionen unterschieden werden. Beim zweiten dieser Zerfälle ist diese kinematische Unterscheidung nicht mehr ausreichend. Hier wird es notwendig, eine zuverlässige Möglichkeit zur Unterscheidung von Pionen und Myonen zur Verfügung zu haben [23].
Wie oben erwähnt soll diese Unterscheidung bei KLOE mittels der Impulsmessung in der Driftkammer sowie der Energie- und Flugzeitmessung im Kalorimeter vorgenommen werden. Bei den ersten Tests schien sich aber abzuzeichnen, daß damit keine ausreichende Unterscheidung zwischen Pionen und Myonen möglich sein würde. Aus diesem Grund wurde nach weiteren Möglichkeiten gesucht, die Teilchenidentifikation bei KLOE zu verbessern. Ein verfolgter Ansatz bestand darin, zu untersuchen, ob durch das Zählen der Ionisationsereignisse innerhalb einer Driftzelle ('Cluster Counting', CC) anstelle der Integration des induzierten Stromes eine Verbesserung der Auflösung der Energieverlustmessung, und damit eine bessere Teilchenidentifikation, möglich ist. In der Theorie läßt sich hierbei eine bessere Teilchenidentifikation erwarten, da durch die zugrundeliegenden Prozesse die Zahl der Ionisationsereignisse im Inneren einer Driftzelle einer Poisson-Verteilung folgt, während der Energieverlust einer Landau-Verteilung mit deutlich höherer Varianz folgt. Um diese Theorie zu überprüfen, sollte ein Testaufbau für Cluster Counting erstellt werden. Da es zum damaligen Zeitpunkt (1993) noch fraglich war, ob die erhältlichen elektronischen Bauelemente den Ansprüchen von Cluster Counting gewachsen waren, sollte dieser Testaufbau eine vollständige Auslesekette, von einer Testkammer bis zur Speicherung der aufgenommenen Daten, enthalten. Die vorliegende Arbeit befaßt sich mit dem digitalelektronischen Teil dieser Auslesekette.
In späteren Testaufbauten zeigte sich allerdings, daß die Teilchenidentifikation mittels des Kalorimeters zusammen mit der unerwartet guten Spurauflösung in der Driftkammer ausreichen sollte, um Pionen und Myonen im fraglichen Energiebereich ausreichend gut voneinander zu trennen, so daß im Rahmen von KLOE auf eine weitere Teilchenidentifikation mittels einer Energieverlustmessung in der Driftkammer – sei es $\frac{d\mathrm{E}}{d\mathrm{x}}$ oder Cluster Counting – verzichtet wurde. Da aber auch im Rahmen anderer Präzisionsexperimente der Hochenergiephysik ein grundsätzliches Interesse an einer verbesserten Teilchenidentifikation besteht, wurde die Arbeit an Cluster Counting im Rahmen einer Zusammenarbeit zwischen der Università di Roma II, dem INFN Lecce und dem Institut für Experimentelle Kernphysik der Universität Karlsruhe weiterverfolgt.

**Der KLOE-Detektor [14]**

Um das Ziel, die genaue Bestimmung des Verhältnisses $\Re \frac{\epsilon'}{\epsilon}$, erreichen zu können, muß der KLOE-Detektor möglichst viele Zerfälle der $K_L$ und $K_S$ analysieren. Hierbei gibt es unter anderem zwei Punkte zu beachten:

- Die Zerfallslänge der $K_L$ beträgt bei der Energie, mit der sie im KLOE-Detektor aus dem Zerfall des $\Phi$ entstehen, 3.44 Meter. Um möglichst viele der Zerfälle der $K_L$ beobachten zu können, muß der Detektor entsprechend groß sein.

- Die beim Zerfall des $K_S$ und $K_L$ vorwiegend entstehenden geladenen und neutralen Pionen müssen im Detektor nachgewiesen, identifiziert und vermessen werden.

[6] Ein Detektor, der den vollen Raumwinkel ohne Lücken überdeckt.

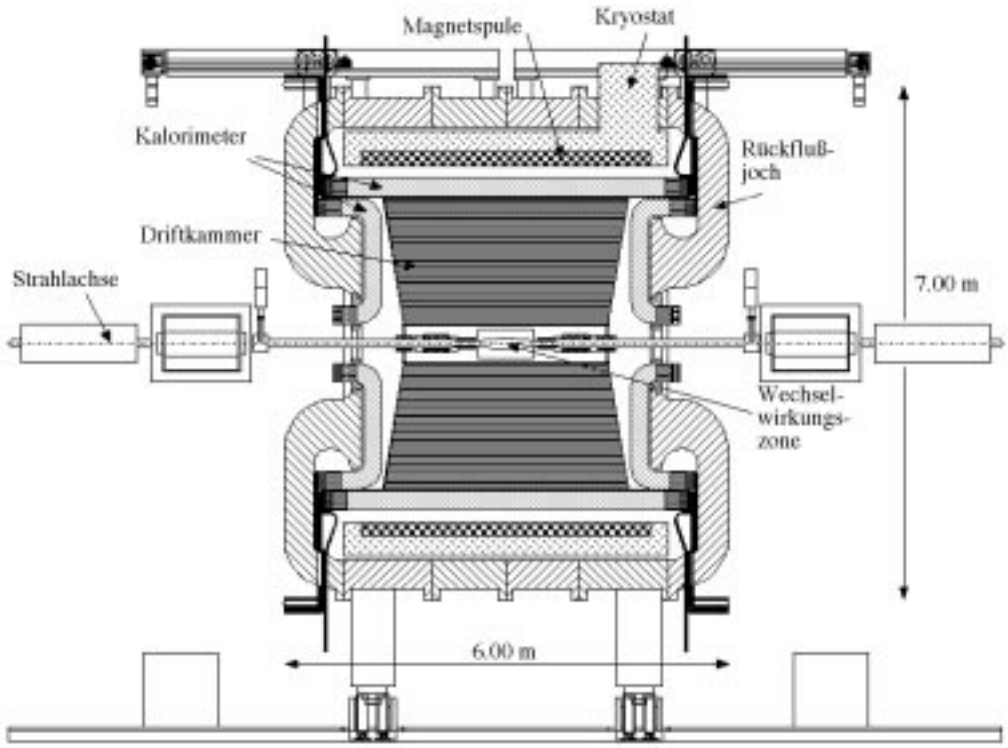



Abbildung 1.4: Querschnitt durch den KLOE-Detektor [13]

Abbildung 1.4 zeigt einen Querschnitt durch den KLOE-Detektor. Der Aufbau des Detektors folgt dem mittlerweile für die Physik an Speicherringen bewährten Schema. In einem zylindersymmetrischen Aufbau um die Wechselwirkungszone herum befindet sich im Inneren ein Spurdetektor, der von einem Kalorimeter umgeben ist. Bei KLOE befinden sich der Spurdetektor und das Kalorimeter innerhalb eines axialen Magnetfelds. Für den Spurdetektor ist dieses Magnetfeld notwendig, um durch die Krümmung der Spur eines geladenen Teilchens dessen Impuls messen zu können (siehe Abschnitt 1.1.1). Darüberhinaus befindet sich, entgegen der üblichen Praxis, bei KLOE auch das Kalorimeter im Inneren der Spule, um die Menge an Material – und damit die Vielfachstreuung – zwischen dem Entstehungsort der Teilchen und deren Nachweis im Kalorimeter so gering wie möglich zu halten,

**Driftkammer** Als Spurdetektor wird bei KLOE eine Driftkammer verwendet. Wie aus Abbildung 1.4 zu sehen ist, ist diese Driftkammer mit einer Länge von 3.5 m und einem Durchmesser von fast 4 m im Verhältnis zu anderen Experimenten relativ groß. Dies ist die Folge der oben erwähnten Bedingung, daß ein möglichst großer Anteil der Zerfälle der $K_L$ im Inneren der Kammer beobachtet werden muß. Technische und finanzielle Erwägungen beschränken allerdings die Kammergröße nach oben hin.

Diese Kammer wird in etwa 13000 Driftzellen unterteilt werden, wobei die Zellen der äußeren Lagen eine Größe von etwa $3 \times 3$ cm haben werden, die Zellen der zwölf inneren Lagen eine Größe von etwa $2 \times 2$ cm. Weil alle Lagen unter einem kleinen Winkel zur Strahlachse geführt werden sollen (‘Stereolagen’), sind die Zellenquerschnitte allerdings

nicht genau rechteckig. Als Driftgas wird bei KLOE eine Helium-Isobutan-Mischung zum Einsatz kommen.

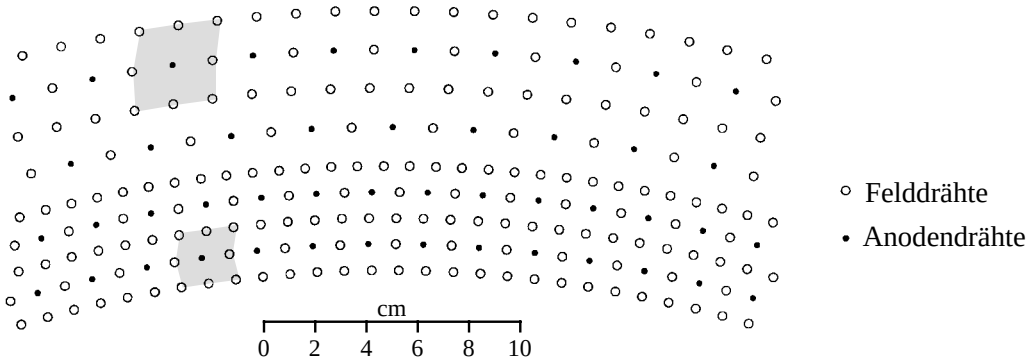



Abbildung 1.5: Ausschnitt des Lochplanes einer der KLOE-Endkappen an der Grenze zwischen den größeren ($\pi \times 3$ cm$^2$) und den kleineren ($2\pi/3 \times 2$ cm$^2$) Zellen [24]. Der Querschnitt jeweils einer Zelle in den beiden Bereichen ist grau unterlegt.

Mit der Verwendung von Helium als Grundlage für das Füllgas einer großen Driftkammer wird bei KLOE technisches Neuland betreten. Der Grund dafür war, daß die Menge an Material im Inneren des Detektors möglichst klein gehalten werden sollte. Dies wiederum hat zwei Gründe: Zum Einen müssen die durch Vielfachstreuung hervorgerufene Fehler klein gehalten werden, um eine genaue Spurrekonstruktion und damit auch eine genaue Vermessung der Kinematik einer Reaktion zu ermöglichen. Nach Gleichung 1.7 ist der durch die Vielfachstreuung hervorgerufene Fehler proportional zum Kehrwert der Wurzel der Strahlungslänge $X_0$ des durchquerten Mediums. Bei einer Kammer in der Größe der KLOE-Driftkammer würde eine Füllung mit einem konventionelle Driftgas wie zum Beispiel Argon zu im Rahmen dieses Experiments nicht mehr akzeptablen Fehlern führen [25]. Ebenfalls aus diesem Grund wird bei den Felddrähten im Inneren der Driftkammer anstelle der üblichen Kupfer-Beryllium-Legierung Aluminium zum Einsatz kommen.

Der zweite Grund, nur wenig Material im Inneren des Detektors zuzulassen, besteht darin, die Regeneration der Kaonen durch die Wechselwirkung mit diesem Material zu minimieren. Für die Messung des Doppelverhältnisses in Gleichung 1.20 ist es wichtig, die nachgewiesenen Zerfälle einem der beiden Zustände $K_L$ oder $K_S$ zuzuordnen. Durch die Regeneration des $K_S$-Anteils bei der Wechselwirkung mit der Materie im Inneren des Detektors werden für diese Messung zusätzliche Fehler eingeführt, die nach Möglichkeit vermieden, oder zumindest klein gehalten werden sollen.

Bei der Verwendung von Helium als Kammergas bestand allerdings anfänglich die Befürchtung, daß durch die geringere Zahl an primären Ionisationsereignissen im Vergleich zu z. B. Argon die Auflösung der Spurvermessung nicht den notwendigen Bedingungen genügen würde. Bei Messungen an Testkammern [26, 27] wurde jedoch festgestellt, daß sich auch mit Helium sehr gute Werte für die Genauigkeit der Spurvermessung erreichen lassen.

Ein Problem bei der Verwendung von Helium als Hauptbestandteil des Driftgases besteht allerdings darin, daß die Driftgeschwindigkeit der Elektronen unter den bei KLOE herrschenden Bedingungen sehr empfindlich von der genauen Zusammensetzung des Gasgemisches

abhängt[7]. Um diese Gaszusammensetzung und speziell die Driftgeschwindigkeit genau zu überwachen, wird das Gassystem bei KLOE eine aufwendige Analyse- und Reinigungsapparatur sowie eine kleine Testkammer zur Bestimmung der Driftgeschwindigkeit enthalten [28].

**Kalorimeter** Zur Identifikation der beim Zerfall der Kaonen entstehenden Teilchen ist bei KLOE ein ortsauflösendes, aus Blei und szintillierenden Fasern aufgebautes Kalorimeter vorgesehen. Dieses Kalorimeter wird neben der Energiemessung der bei den Zerfällen der einzelnen Teilchen entstehenden Photonen auch zur Flugzeitmessung herangezogen.

Versuche an einem Teststrahl des CERN mit einem Modul des Kalorimeters zeigen sehr gute Werte für die Zeit- und Energieauflösung [29].

**Vertexdetektor** Im ursprünglichen Entwurf von KLOE war ein Vertexdetektor aus sogenannten 'Straw Tubes'[8] vorgesehen. Dieser Plan wurde aber mittlerweile wieder fallengelassen.

**Magnetspule** Um eine Impulsmessung innerhalb der Driftkammer zu ermöglichen, befindet sich der gesamte Detektor (Kalorimeter, Driftkammer und Vertexdetektor) innerhalb einer supraleitenden Magnetspule. Diese erzeugt innerhalb der Driftkammer ein homogenes Magnetfeld von $0.6$ Tesla parallel zur Strahlachse.

| | |
|---|---|
| Schwerpunktsenergie | 1.020 $\mathrm{GeV}$ |
| Magnetfeldstärke | 0.6 T |
| Energieauflösung des Kalorimeters | $\frac{\sigma_E}{E} = \frac{4.96 \pm 0.01}{\sqrt{E[\mathrm{GeV}]}}\%$ |
| Zeitauflösung im Kalorimeter | $\sigma_T(E) = \frac{71.7 \pm 1.0}{\sqrt{E[\mathrm{GeV}]}} ps$ |
| Auflösung des Kalorimeters in $z$ | $\sigma_z(E) = \frac{1.2 \pm 0.1}{\sqrt{E[\mathrm{GeV}]}} cm$ |
| Ortsauflösung der Driftkammer in $r$ und $\phi$ | 200 $\mu$m pro Zelle |
| in $z$ | 2 mm pro Zelle |
| Impulsauflösung | $\delta_{p_\perp} = 0.5\% \cdot p_\perp$ |
| Winkelfehler | $\delta(\tan\theta) = (3.5 \oplus 2.5) \cdot 10^{-4}$ |

Tabelle 1.2: Übersicht der wichtigsten Parameter des KLOE-Detektors am DAΦNE-Speicherring

[7] Außerdem gehört Helium zu den Gasen, in denen die Driftgeschwindigkeit der Elektronen unter den innerhalb einer Driftzelle vorliegenden Bedingungen noch vom lokalen elektrischen Feld abhängt, siehe Abschnitt 1.1.2.

[8] 'Strohhalme', dünne (hier ca. 5 mm) Driftröhren

# Kapitel 2

# Cluster Counting

Wie in Abschnitt 1.1.2 gesagt, kann man über den spezifischen Energieverlust pro Weglänge $\frac{dE}{dx}$ eine Teilchenidentifikation innerhalb einer Driftkammer vornehmen (siehe auch Bild 1.3). Eines der größten Probleme dieser Methode ist, daß die Verteilung der gemessenen Werte für den spezifischen Energieverlust einer Landau-Statistik unterliegt. Um eine optimale Auflösung zu erhalten, hat es sich als zweckmäßig erwiesen, einen Teil der erhaltenen Information (d.h. die Meßwerte mit dem höchsten Energieverlust im 'Schwanz' der Verteilung) zu verwerfen. Unter diesen Umständen erreicht man in der Praxis eine Auflösung entsprechend Gleichung 1.8.

## 2.1 Cluster Counting - der Vorteil der Poisson-Statistik

Im Unterschied zur Landau-Verteilung des spezifischen Energieverlustes pro Weglänge (oder besser der gemessenen Energiedeposition pro Driftzelle) folgt die Anzahl der primären Stöße des ionisierenden Teilchens mit dem Kammergas einer Poisson-Verteilung mit deutlich kleinerer Varianz.
Diese Poisson-Verteilung resultiert aus den zugrundeliegenden physikalischen Prozessen: Zunächst einmal ist der Energieverlust pro Stoß eines geladenen Teilchens im interessierenden Energiebereich nur ein kleiner Bruchteil der Gesamtenergie dieses Teilchens. Damit wird die Wahrscheinlichkeit für einen Stoß innerhalb einer gewissen Strecke unabhängig von der Position dieses Teilchens innerhalb der Driftkammer. Damit wiederum können die einzelnen Stöße als statistisch unabhängige Prozesse betrachtet werden, woraus zum Einen eine Exponentialverteilung für die Abstände zweier aufeinanderfolgender Stöße und zum Anderen die erwähnte Poisson-Verteilung für die Anzahl der Stöße im Inneren der Driftzelle resultiert.
Durch das Zählen der primären Ionisationsereignisse anstelle des Messens der gesamten freigesetzten Ladung könnte man also den statistischen Fehler bei der Messung des spezifischen Energieverlustes verringern und somit die Auflösung verbessern. Diese Zähltechnik wird 'Cluster Counting' ('Clusterzählung') genannt. 'Cluster' deswegen, weil ein primäres Ionisationsereignis eine kleine Zahl von Elektronen freisetzen kann, die sich dann als Gruppe ('Cluster') zum Anodendraht hinbewegen und dort registriert werden.

### 2.1.1 Poisson- und Landauverteilung

Für eine Poisson-Verteilung $P_K^{N_{\mathrm{cl}}} = \frac{(N_{\mathrm{cl}})^K e^{-N_{\mathrm{cl}}}}{K!}$, mit $N_{\mathrm{cl}} = \delta_{cl} \cdot l$ der Zahl der Cluster als Produkt aus Clusterdichte und gesamter Weglänge, ist der auf die Zahl der Cluster normierte Fehler [30]:

$$\frac{\sigma_{N_{\mathrm{cl}}}}{N_{\mathrm{cl}}} = \frac{\sqrt{N_{\mathrm{cl}}}}{N_{\mathrm{cl}}} = \frac{1}{\sqrt{N_{\mathrm{cl}}}} = \frac{1}{\sqrt{\delta_{\mathrm{cl}} l}} \propto \frac{1}{\sqrt{l}}, \tag{2.1}$$

skaliert also mit dem inversen der Wurzel der Weglänge.
Unter der Annahme eines Detektors in den Abmessungen von KLOE, dessen Driftkammer mit 56 Lagen von Driftzellen zu je 3 cm Dicke ausgestattet ist (eine der Versionen, die für KLOE diskutiert wurden) und einem Winkel von 30° zwischen der Teilchenspur und dem Anodendraht[1], erhält man mit Gleichung 1.8[2] für Argon als Kammergas eine Auflösung

$$\frac{\sigma_{\frac{dE}{dx}}}{\left\langle \frac{dE}{dx} \right\rangle_{\mathrm{tr.\ mean}}} = 0.042, \tag{2.2}$$

also 4.2 %. Eine Verkleinerung aller Driftzellen auf 1.5 cm würde diesen Wert um 0.4 % auf 3.8 % verbessern.
In Argon unter Normalbedingungen werden von einem minimal ionisierenden Teilchen etwa 30 Cluster pro cm erzeugt [7], in Helium etwa 5 [30] und in einem Gemisch aus 90 % Helium und 10 % Isobutan etwa 12 [30, 31]. Bezogen auf eine Weglänge von 200 cm ergeben sich mit diesen Zahlen folgende relative Fehler:

| Messung | relativer Fehler |
|---|---|
| $\frac{dE}{dx}$ in Argon | $\frac{\sigma_{\frac{dE}{dx}}}{\left\langle \frac{dE}{dx} \right\rangle_{\mathrm{tr.\ mean}}} = 0.042$ |
| Cluster Counting in Argon | $\frac{\sigma_{N_{\mathrm{cl}}}}{N_{\mathrm{cl}}} = 0.013$[3] |
| $\frac{dE}{dx}$ in Helium-Isobutan [27] | $\frac{\sigma_{\frac{dE}{dx}}}{\left\langle \frac{dE}{dx} \right\rangle_{\mathrm{tr.\ mean}}} = 0.043$ |
| CC in Helium-Isobutan 90:10 | $\frac{\sigma_{N_{\mathrm{cl}}}}{N_{\mathrm{cl}}} = 0.020$ |

Damit wird deutlich, daß sich mit Cluster Counting im Vergleich zur Messung des spezifischen Energieverlustes und der Auswertung über die bekannte ‘truncated mean’-Methode eine deutlich bessere Auflösung, und damit auch eine verbesserte Teilchenidentifikation erreichen läßt.
Es bleibt die Frage, ob sich diese Methode auch in der Praxis eines Experiments einsetzen läßt. Mit der vorliegenden Arbeit soll diese Frage näher untersucht und bezüglich der elektronischen Verarbeitung beantwortet werden.

[1] Damit ergibt sich in der beschriebenen Geometrie eine Spurlänge pro Zelle von 3.6 cm, insgesamt 2 m.

[2] und unter der Annahme, daß das Verhältnis zwischen der Halbwertsbreite FWHM und der Varianz $\sigma$ der truncated-mean-Verteilung etwa dem einer Gaußverteilung entspricht

[3] Diese Auflösung läßt sich allerdings nicht realisieren. Die Clusterdichte in Argon ist zu hoch, um mit der gegenwärtig erhältlichen Elektronik noch eine Auslese zu erlauben, bei der die einzelnen Cluster voneinander unterschieden werden könnten.

### 2.1.2 Probleme bei der Umsetzung in die Praxis

So schön, wie sich der statistische Vorteil von Cluster Counting in der Theorie anhört, so schwierig ist es, Cluster Counting in die Praxis umzusetzen.
Bei der Verwendung von Argon als Driftgas und einer Zellengröße von 3 cm werden innerhalb einer Zelle beim Durchgang eines Teilchens im Schnitt 90 Cluster erzeugt. Bei einer Driftgeschwindigkeit[4] in Argon (oder einem Argon-Isobutan-Gemisch[5]) von etwa $1 \ldots 4$ cm/$\mu$s [6] treffen diese Cluster im Zeitraum von $\approx 0.4 \ldots 1.5$ $\mu$s am Anodendraht ein. Damit ergibt sich ein mittlerer zeitlicher Abstand zweier Cluster in der Größenordnung von $5 \ldots 10$ ns, zu kurz, um mit vertretbarem Aufwand beim heutigen Stand der Elektronik eine Unterscheidung zweier aufeinanderfolgender Cluster zu erlauben.
In einem Helium-Isobutan-Gemisch sieht dieses Verhältnis schon etwas besser aus. Die Driftgeschwindigkeit ist mit etwa 0.5...2 cm/$\mu$s etwas geringer als in Argon, und die Zahl der Cluster pro cm mit etwa 10 statt 30 deutlich kleiner. Somit ergibt sich für die Auslese bei einer Zellengröße von 3 cm ein mittlerer zeitlicher Abstand zwischen zwei Clustern von etwa 75 ns. Dies ist mit moderner Ausleseelektronik beherrschbar. Durch den geringen Abstand zwischen zwei Clustern müssen aber auch hier für die Auslese von Driftkammern neue Techniken zur Anwendung kommen.

## 2.2 Der Datenfluß beim Cluster Counting

Im Vergleich zur konventionellen $\frac{dE}{dx}$-Auslese müssen bei der Cluster Counting-Auslese deutlich höhere Frequenzen mit berücksichtigt werden. Bei der konventionellen Auslese wird die Form des gemessenen Signals nicht berücksichtigt, als Meßgrößen treten hier in den meisten Fällen nur die Ankunftszeit des Signals sowie das Integral des innerhalb eines gewissen Zeitfensters zu messenden Stromes auf. Bei Cluster Counting hingegen ist die Signalform der zunächst interessierende Parameter. Alle anderen Meßwerte werden aus der Form des Signals abgeleitet.
Aus diesen Voraussetzungen resultieren die unterschiedlichen Ansprüche an den Aufbau der Driftzellen und die Ausleseelektronik. Bei der konventionellen Auslese kann diese Elektronik – relativ – langsam sein. Die Anstiegszeiten der Vorverstärker in diesem Bereich liegen bei etwa 10 ns, die Abfallzeiten bei etwa 20 ns. Dies ist für die zeitliche Festlegung der steigenden Signalflanke ausreichend. Bei Cluster Counting dagegen setzt sich das zu beobachtende Signal aus der Überlagerung einzelner Clustersignale zusammen, von denen jedes etwa eine zeitliche Breite von 30 ns aufweist. Das durch die Überlagerung dieser Anteile entstehende Gesamtsignal muß nun möglichst unverfälscht über die Auslesekette bis zum ADC[6] weitergegeben werden. Hierfür muß der Aufbau Frequenzen bis an den GHz-Bereich ($\approx$ 500 MHz bei einer Abtast-Rate ('Sampling-Rate') von 1 GS/s[7] ) ohne nennenswerte frequenzabhängige Signalveränderungen übertragen können [31].

### 2.2.1 Von der Signalentstehung zum Vorverstärker

Da bei der konventionellen Auslese zur $\frac{dE}{dx}$-Messung die Form des Signals nur von untergeordneter Bedeutung ist, kann auf die Betrachtung der Hochfrequenzeigenschaften der einzelnen Baugrup-

[4] Siehe Abschnitt 1.1.2 und 1.2.2
[5] Abhängig von der genauen Gaszusammensetzung
[6] **A**nalog to **D**igital **C**onverter, Analog-Digital-Wandler
[7] **G**iga**S**amples pro **S**ekunde, $10^9$ Abtastungen pro Sekunde

pen verzichtet werden. Das Ersatzschaltbild für diesen Fall zeigt Abbildung 2.1.

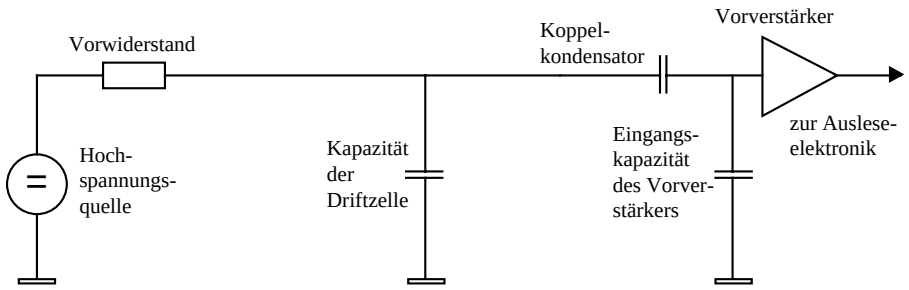



Abbildung 2.1: Ersatzschaltbild für eine Driftzelle bei analoger Auslese

Der Anodendraht ist hier ein einfacher ‘Draht’ ohne besondere Eigenschaften. Zur Bestimmung der Eigenschaften der Signalübertragung läßt sich das Verhalten der Driftzelle für diesen Fall im Wesentlichen durch ihre Kapazität beschreiben. Die Aufgabe des Vorwiderstandes beschränkt sich auf den Entladungsschutz. Über die aus dem Vorwiderstand und der Parallelschaltung von Driftzellenkapazität und Eingangskapazität des Vorverstärkers gebildete Zeitkonstante ergibt sich auch eine Auswirkung auf die Signalform.
Im Unterschied dazu muß die Auslesekette bei Cluster Counting so ausgeführt werden, daß eine Beeinflussung der Signalform möglichst vermieden wird, das Signal also so originalgetreu wie möglich am ADC ankommt.

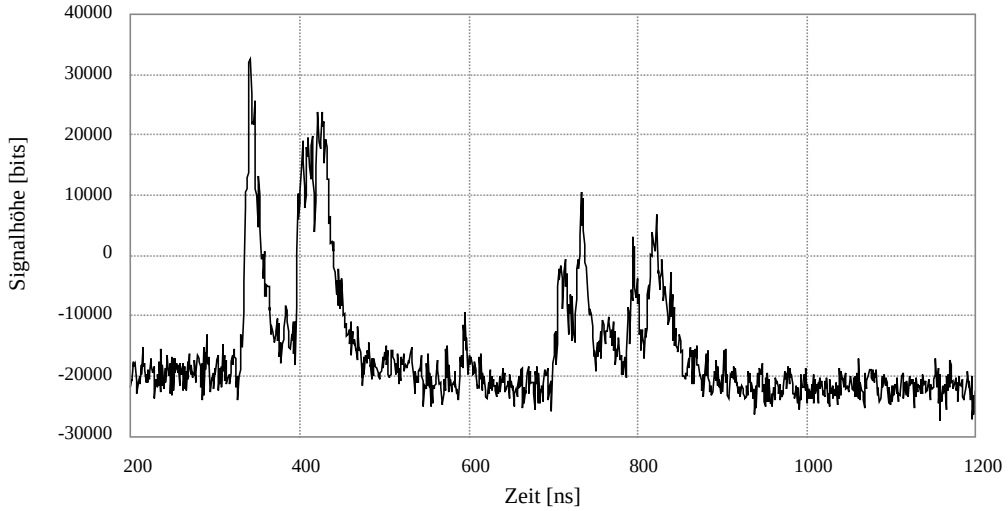
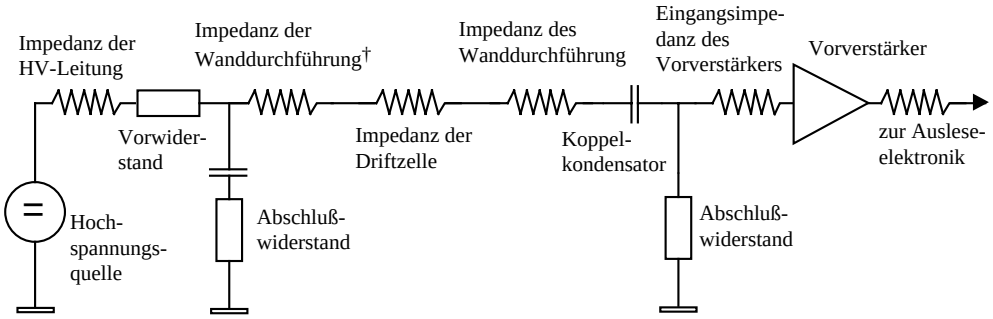



Abbildung 2.2: Ausschnitt eines Cluster Counting-Signals

Bild 2.2 zeigt einen Ausschnitt eines gemessenen Anodensignals. Die Signale der einzelnen Cluster sind hier deutlich zu erkennen. Um ein solches Signal ohne Verfälschungen vom Ort der Entstehung bis zum Analog-Digital-Wandler zu leiten, muß diese ganze Strecke als Hochfrequenz-Übertragungsleitung (‘Transmission Line’, [32, 33]) betrachtet werden (Abbildung 2.3).




† Auch "Feedthrough" genannt, kleines Bauteil, meist aus Kunststoff und Metall, mit dem die Drähte gasdicht durch die Wand der Driftkammer geführt und fixiert werden.

Abbildung 2.3: Ersatzschaltbild für eine Driftzelle unter Berücksichtigung des Hochfrequenzverhaltens

Dabei kommt auch das Hochfrequenzverhalten einiger weiterer Baugruppen zum Tragen, die in der Niederfrequenzbetrachtung keine Rolle spielten:

- Die Hochspannungsquelle (HV, 'high voltage'): Diese sollte bei korrekter Auslegung der Vor- und Abschlußwiderstände keine Auswirkungen auf das Signal haben. Ist jedoch der Vorwiderstand zu klein gewählt, so kann es zur Einkoppelung des Kammersignals auf das Hochspannungskabel kommen, wobei dieser Signalanteil dann am Netzteil reflektiert wird und unter Umständen als zeitlich verschobenes Signal am Ausgang der Driftzelle beobachtet werden kann.

- Die Hochspannungszuführung: Dies ist üblicherweise ein Koaxialkabel und weist somit aus geometrischen Gründen eine Impedanz im Bereich zwischen $50 \ldots 100\,\Omega$ auf.

- Das hochspannungsseitige Widerstandsnetzwerk: Diese Kombination aus Vorwiderstand, Kondensator und Abschlußwiderstand muß hier, neben dem Entladungsschutz, auch den impedanzangepaßten Abschluß der Driftzelle übernehmen. Dazu wird ein Abschlußwiderstand benötigt, dessen Wert in der Parallelschaltung mit dem Vorwiderstand möglichst genau der Impedanz der Driftzelle entsprechen sollte. Diese Impedanz beträgt bei den üblichen Abmessungen[8] einer Driftzelle etwa $300 \ldots 450\,\Omega$. Damit werden Reflektionen der Clustersignale an diesem Ende der Driftzelle verhindert, die sonst bei der Auslese weitere Clustersignale vorspiegeln können.

  Da zum Betrieb der Driftzelle eine hohe Spannung benötigt wird, muß der Abschlußwiderstand zur Vermeidung eines hohen Stromflusses und damit hoher Verlustleistung über einen Kondensator zur Abkopplung dieser Gleichspannung angeschlossen werden.

  Dem Hochspannungs-Vorwiderstand kommt in dieser Schaltung noch eine weitere Aufgabe zu: er soll verhindern, daß Anteile des Signals in die Hochspannungsleitung übertreten, und

[8] Für einen idealen, verlustfreien und unendlich langen koaxialen Leiter beträgt die Impedanz $Z_0 = \sqrt{l/c} \cong 60 \ldots 120\Omega\sqrt{\mu/\epsilon}\ln(r_a/r_i)$, wobei $l$ und $c$ die längenbezogene Induktivität bzw. Kapazität darstellen, $\mu$ und $\epsilon$ die relativen Permeabilitäts- und Dielektrizitätskonstanten sowie $r_a$ und $r_i$ den Radius des äußeren bzw. inneren Leiters. Da diese Funktion mit dem Logarithmus des Verhältnisses der Durchmesser skaliert, bedarf es großer Änderungen in diesem Verhältnis, um die Impedanz eines koaxialen Leiters nennenswert zu beeinflussen.

dann entweder direkt in anderen Driftzellen (über einen der Hochspannungsverteiler) oder indirekt nach einer Reflektion am Netzteil weitere, verzögert eintreffende Clustersignale vortäuschen.

Um diese Aufgaben erfüllen zu können, muß dieses Netzwerk sehr dicht (im Bezug auf die Wellenlängen der entsprechenden Signale), im Bereich weniger Millimeter, an der Wanddurchführung direkt auf der Endplatte des Detektors angeschlossen werden.

- Die Wanddurchführung: Die Durchführung des Anodendrahtes durch die Wand der Driftkammer muß neben der elektrischen Isolation dieses Drahtes auch für dessen exakte Positionierung sorgen. Besteht die Wand der Driftkammer aus elektrisch nichtleitendem Material, so gibt es hier keine weiteren Probleme. Besteht sie jedoch aus leitendem Material, so ändert sich aus der Sicht des Signales mit der Annäherung an die Wand das elektrische Feld und damit auch die Impedanz der Driftzelle, wodurch es an dieser Stelle wieder zu Reflektionen kommen kann. Durch einen geeigneten Entwurf der Wanddurchführung [34] kann diese Impedanzänderung klein gehalten werden.

- Die Driftzelle: Wie bereits oben erwähnt beträgt die Impedanz einer Driftzelle etwa 400 $\Omega$. Da diese durch die Dielektrizitätskonstante des Füllgases und dem Verhältnis zwischen dem Durchmesser des Anodendrahtes und dem der Driftzelle festgelegt wird, läßt sich dieser Wert nicht wesentlich beeinflussen.

- Die zweite Wanddurchführung: Hier gelten wieder dieselben Regeln wie für die erste Wanddurchführung. Eine zusätzliche Anforderung an diese Durchführung ist, daß die hochfrequenten Clustersignale möglichst dämpfungsarm aus der Kammer geführt werden müssen.

- Der Vorverstärker mit Koppelkondensator und Abschlußwiderstand: Auch hier gilt, daß dieser Aufbau möglichst dicht an der Wanddurchführung angeschlossen werden muß (im Bereich weniger Millimeter). Der Koppelkondensator muß die Hochspannung (Gleichspannung) vom Eingang des Vorverstärkers und dem Abschlußwiderstand fernhalten, gleichzeitig im Bereich der interessierenden Frequenzen (bis in den GHz-Bereich) eine möglichst kleine Dämpfung aufweisen. Dies läßt sich am günstigsten mit Keramik-Kondensatoren in SMD[9]-Technik realisieren, da diese ein besseres Hochfrequenzverhalten aufweisen als bedrahtete Bauformen.

  Für den Vorverstärker bieten sich die in den letzten Jahren für die Funktelefontechnik entwickelten Hochfrequenzverstärker an. In unseren Testaufbauten [35] haben wir hier Vorverstärker des Typs MAR-8 des Herstellers Mini-Circuits [36] verwendet. Dieser Typ unterscheidet sich von den meisten anderen ähnlichen Verstärkern dadurch, daß er eine Eingangsimpedanz im Kiloohm-Bereich besitzt[10]. Dadurch ist eine einfache Anpassung an die Impedanz der Driftkammer durch einen parallelgeschalteten Abschlußwiderstand möglich. Der genaue Wert diese Widerstandes muß im Aufbau experimentell ermittelt werden, für eine erste Näherung kann hier ein Standardwert in der Nähe des Wertes der Impedanz der Driftzelle, also etwa 300…450 $\Omega$, eingesetzt werden. Die Ausgangsimpedanz des Verstärkers beträgt 50 $\Omega$, so daß eine Weiterführung des Signals mit normalen Hochfrequenzkabeln möglich ist.

[9] **S**urface **M**ounted **D**evice, Oberflächenmontiertes Bauelement

[10] Die übliche Ein- und Ausgangsimpedanz für Verstärker dieser Art beträgt 50 $\Omega$. Eine Anpassung an die Impedanz der Driftzelle wäre in diesem Fall mit einer starken Verringerung der Amplitude verbunden.

### 2.2.2 Vom Vorverstärker zum ADC

In einem idealen Aufbau säße der ADC direkt am Vorverstärker, womit praktisch alle nachstehenden Probleme aus der Welt geschafft wären.
In einem idealen Aufbau hätte man dann aber auch keine Probleme mit dem Energieverbrauch und der damit verbundenen Wärmeentwicklung schneller ADCs, deren Größe und der für den Abtransport der digitalisierten Daten benötigten Übertragungsbandbreite.
In der Realität wird man also die ADCs in Schaltschränken (‘Racks’) in der Nähe des Detektors unterbringen müssen und sie über geeignete Leitungen mit den Vorverstärkern verbinden.
Um die Anzahl der ADCs und die damit verbundenen Kosten möglichst gering zu halten, könnte man die analogen Clustersignale zunächst in einem analogen Zwischenspeicher ablegen, um dann bei einem Triggersignal[11] nur die interessierenden Signale zur Weiterverarbeitung zu digitalisieren.
Der Nachteil einer solchen, in Abbildung 2.4.a gezeigten, Anordnung besteht in der Notwendigkeit für ein entsprechend aufwendiges analoges Zwischenspeicher- und Verteilungsnetzwerk. Trotz des nötigen Aufwandes wird an solchen analogen Zwischenspeicherstufen gearbeitet [37, 38].

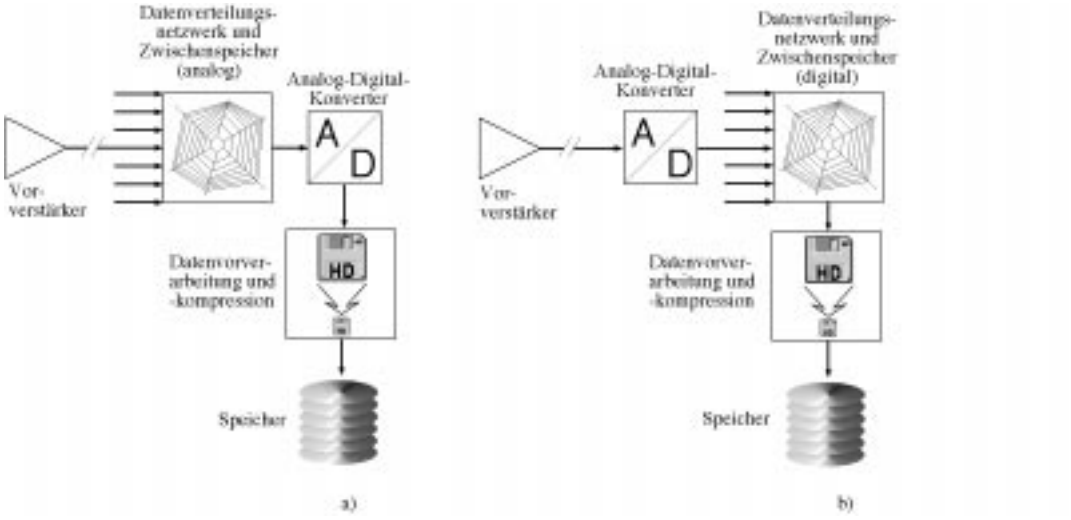



Abbildung 2.4: Symbolische Darstellung des Datenflusses, im Fall a) mit analogem Zwischenspeicher- und Verteilungsnetzwerk vor dem ADC, im Fall b) mit digitalem Zwischenspeicher- und Verteilungsnetzwerk nach dem ADC

Die Alternative bestünde darin, jedem Kanal einen eigenen ADC zuzuordnen. Dieser wäre dann über eine feste Leitung mit dem dazugehörenden Vorverstärker verbunden, während das Verteilungsnetzwerk auf der Ausgangsseite der ADCs sitzt und – in diesem Fall – digitale Daten verteilt (Abbildung 2.4.b).
Unabhängig davon, ob die Daten nun direkt oder über einen analogen Zwischenspeicher zum ADC gelangt sind, müssen sie hier nun digitalisiert werden. Um Signalanstiegszeiten im Bereich weniger Nanosekunden noch vernünftig abbilden zu können, sollte der ADC mit Abtastraten im GHz-Bereich arbeiten. Nach dem Shannon’schen Abtasttheorem [32, 39] muß die Abtastrate min-

[11] Ein solches Triggersignal (wörtl. ‘Auslöser’) wird von einer speziellen Detektorelektronik immer dann erzeugt, wenn im Detektor dem Anschein nach ein ‘interessantes’ Ereignis aufgetreten ist. Die Bedingungen für ein ‘interessantes’ Ereignis können im allgemeinen in weiten Grenzen definiert werden.

destens das Doppelte der höchsten vorkommenden Signalfrequenz betragen[12]. Für Cluster Counting heißt das, daß mit einer Abtastrate von 1 GS/s bei einer Anstiegszeit von etwa 5 ns 5 Punkte auf der ansteigenden Signalflanke vermessen werden, womit diese ausreichend definiert ist.
Unter diesen Voraussetzungen läßt sich die Datenrate beim Einsatz von Cluster Counting für einen Detektor ähnlich dem KLOE-Detektor folgendermaßen abschätzen:

- 10000 Driftzellen, entsprechend 10000 Auslesekanälen,
- eine Triggerrate von 10 kHz,
- eine Occupancy[13] von 1 %,
- 1 GS/s Abtastrate,
- 16 Bit pro Sample Auflösung[14] und
- eine Aufzeichnungsdauer von 2 $\mu$s

ergeben insgesamt eine primäre Datenrate von $10000 \times 10 \cdot 10^3 \frac{1}{\mathrm{s}} \times 1\% \times 1 \cdot 10^6 \frac{\mathrm{Samples}}{\mathrm{s}} \times 2 \frac{\mathrm{Bytes}}{\mathrm{Sample}} \times 2 \cdot 10^{-6}\mathrm{s} = 4 \frac{\mathrm{GB}}{\mathrm{s}}$.

### 2.2.3 Vom ADC zum Speicher

In den meisten Fällen wird zwischen den ADCs und den Datenspeichern noch ein digitales Datenverteilungsnetzwerk nötig sein. Im Fall, daß die ADCs jeweils direkt einem Detektorkanal zugeordnet sind, wird dieses Netzwerk (Abbildung 2.4.b) komplexer ausfallen als in dem Fall, wenn die analogen Daten bereits über ein Verteilungsnetzwerk den ADCs zugeführt werden (Abb. 2.4.a). Aber auch in letzterem Fall wird man nicht für jeden ADC einen eigenen Massenspeicher einsetzen, sondern mehrere ADCs einem gemeinsamen Massenspeicher zuordnen.
Damit tritt aber eines der größten Probleme von Cluster Counting zu Tage: die hierbei auftretende Datenmenge und -rate überfordert die derzeit kommerziell erhältlichen Massenspeicher. Geht man von den größten derzeit erhältlichen Bandlaufwerken in DLT[15]-Technologie aus [40], so beträgt die Datenkapazität pro Kassette bis zu 20 GByte und die mögliche Datentransferrate 3 MBytes/Sekunde. Unter diesen Umständen wären etwa 1300 parallel arbeitende Laufwerke dieses Typs nötig, um die Datenrate bewältigen zu können. Die Kassetten dieser Laufwerke müßten dann jeweils nach knapp zwei Stunden gewechselt werden.
Selbst wenn es möglich wäre, ein solches System aufzubauen (real etwa 2000...3000 Laufwerke zusammen mit mehreren Robotern zum Bandwechsel), wäre es praktisch unmöglich, die Daten auf diesen Bändern für eine Auswertung zu benutzen. Die Daten eines Ereignisses wären über

[12] Dies gilt eigentlich nur für sinusförmige Signale. Bei nicht sinusförmigen Signalen müßte man strenggenommen das Fourierspektrum des Signals betrachten, um die nötige Abtastrate zu bestimmen. Für die bei CC zu beobachtende Signalform ist es in der Praxis ausreichend, 3...5 Abtastpunkte auf der ansteigenden Signalflanke zu haben (die fallende Signalflanke ist wesentlich flacher).

[13] 'Belegung', die Occupancy gibt an, wieviele Kanäle im Durchschnitt pro Ereignis ausgelesen werden müssen

[14] Für die Messung bei Cluster Counting ist eine reale Auflösung von 8 Bit pro Sample, entsprechend 256 Stufen im Signal, ausreichend. Um diese reale Auflösung zu erhalten, muß oft eine höhere Auflösung, z.B. 10 Bit, im ADC gewählt werden. Die in unseren Testaufbauten verwendeten ADCs geben diese Daten in 16 Bit Breite aus, so daß diese Datenwortgröße auch für diese Abschätzung gewählt wurde.

[15] **D**igital **L**inear **T**ape, digitales Bandlaufwerk mir Längsspuraufzeichnung, im Unterschied zu z.B. DAT-Laufwerken mit Schrägspuraufzeichnung

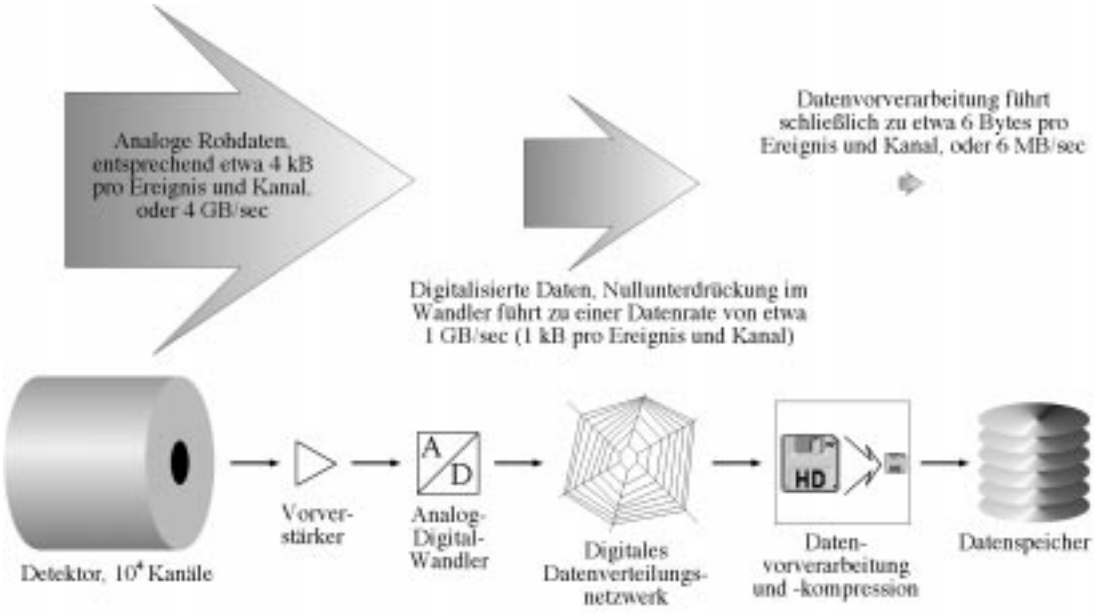



Abbildung 2.5: Symbolische Darstellung der quantitativen Abschätzung des Datenflusses vom Beispieldetektor über ein Datenkompressionssystem bis zum Massenspeicher.

die 1300 Bänder verteilt, und alle diese Bänder müßten für die Auswertung, unter Umständen mehrfach, gelesen werden.
Ein möglicher Ausweg aus diesem Problem besteht in der Kompression der Rohdaten. Dies wird dadurch besonders interessant, daß zur Auswertung der Daten nur zwei Parameter interessant sind: die Ankunftszeit des ersten Clusters relativ zum Trigger sowie die Gesamtzahl der Cluster. Schafft man es, die Rohdaten zwischen den ADCs und den Datenspeichern im laufenden Betrieb (‘online’) auf diese beiden Werte zu reduzieren, verringert sich die Menge der abzuspeichernden Daten auf etwa 6 Bytes pro Kanal und Ereignis[16], oder umgerechnet auf den gesamten Detektor, etwa 6 Megabytes pro Sekunde. Mit denselben Bandlaufwerken wie im obigen Beispiel ließe sich diese Datenrate mit zwei Geräten bewältigen. Eine quantitative Abschätzung des Datenflusses in diesem Fall zeigt Abbildung 2.5.
Die Datenreduktion erfolgt hier in zwei Stufen. In der ersten Stufe werden die Daten noch im Analog-Digital-Wandler durch einen einfachen Algorithmus (z.B. Nullunterdrückung) verlustfrei um etwa einen Faktor vier komprimiert. In der zweiten Stufe, dem Präprozessor, erfolgt dann eine – im Prinzip – verlustbehaftete Datenreduktion um etwa den Faktor 100. Bei dieser Datenreduktion werden jedoch keine physikalisch interessanten Daten verworfen, da sich die gesamte, für die spätere Auswertung nötige Information auf die Zahl der Cluster sowie die Ankunftszeit des ersten Clusters reduzieren läßt.
Damit ergibt sich eine Ausgangsdatenrate von 6 MBytes pro Sekunde, die sich mit zwei DLT-Bandlaufwerken bewältigen läßt.

[16] Als einfache Abschätzung: 1 Byte für die Anzahl der Cluster, 2 Bytes für die Startzeit und etwa 3. . . 4 Bytes für die Zuordnung zu Kanal und Ereignis.

## 2.3 Eine mögliche Realisierung einer Cluster Counting-Auslese

Anhand dieser Überlegungen wurde ein Vorschlag für ein Cluster Counting-Auslesesystem erstellt, der Arbeiten an der Università di Roma II [35, 41], am INFN Lecce [25, 30, 31] und der Universität Karlsruhe [42] mit einschließt (Abbildung 2.6).

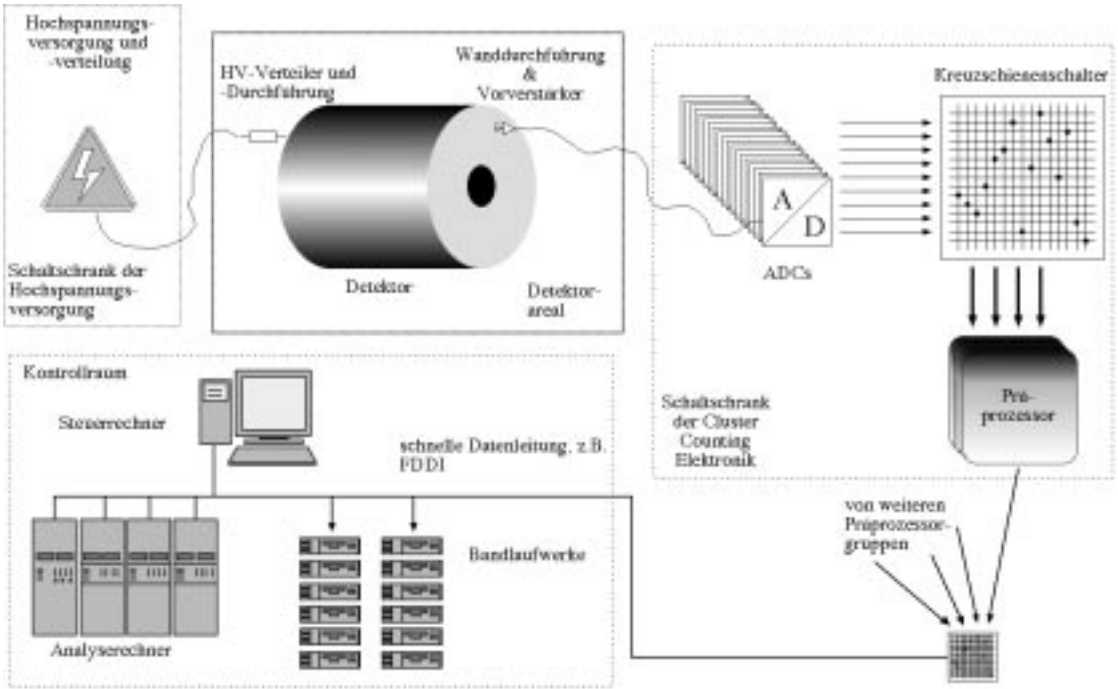



Abbildung 2.6: Vorschlag für eine Cluster Counting-Auslesekette

Dieses Bild zeigt zunächst einmal eine detailliertere Ansicht von Abbildung 2.5, unter Berücksichtigung der Details aus Abschnitt 2.2 und der räumlichen Anordnung der einzelnen Elemente.
In einem solchen Experiment wird nicht die gesamte Anlage gleichermaßen gut zugänglich sein, ein Zugang zum Detektorbereich selbst verbietet sich bei laufender Anlage (bei Hadron-Beschleunigern auch eine gewisse Zeit danach) alleine schon aus Strahlenschutzgründen.
Für diesen Aufbau wird zunächst einmal ein Schaltschrank für die Hochspannungsversorgung benötigt. Hier befinden sich die Hochspannungsnetzteile und -verteiler. Von diesem Schaltschrank aus wird die Hochspannung zunächst zu mehreren lokalen Verteilernetzwerken direkt am Detektor geführt, von wo aus schließlich die einzelnen Driftzellen versorgt werden.
An der Driftkammer wird, wie in Abschnitt 2.2.1 beschrieben, die Hochspannung über einen Widerstand zur Entkopplung zum Anodendraht geführt. Dieser ist auch an der Hochspannungsseite in einer impedanzangepaßten Wanddurchführung befestigt sowie mit einem – über einen Koppelkondensator angeschlossenen – angepaßten Abschlußwiderstand versehen
Am anderen Ende der Driftzelle ist der Anodendraht wieder in einer impedanzangepaßten Wanddurchführung befestigt. Von dort aus wird das Signal über einen Koppelkondensator und ein Anpassungsnetzwerk auf den Vorverstärker geführt, der direkt auf der Endplatte der Driftkammer befestigt ist.
Die Vorverstärker ihrerseits werden, mit einem Kabel pro Driftzelle, mit den ADCs verbunden. An dieser Stelle kamen in Testaufbauten Vorserienmodelle des ADCs ‘TDS 645’ des Herstellers

Tektronix zum Einsatz. Diese Geräte wurden in Zusammenarbeit mit Tektronix unter anderem für Cluster Counting entwickelt und sind mittlerweile auf dem Markt erhältlich.

| | |
|---|---|
| Analogbandbreite | 2 GHz |
| Samplingrate | 5 GS/s |
| Vertikale Auflösung | 8 Bit[17] |
| Pufferspeicher | 32kB |

Tabelle 2.1: Kurzdaten des ADCs TDS 645

Die Analog-Digitalwandler sind auf einem der üblichen Bussysteme aufgebaut (zum Beispiel VME oder VXI[18] ). Die Wandler können dann gruppenweise über einen speziellen Kreuzschienenschalter ('Crossbar Switch', [42]) mit einer Präprozessorgruppe verbunden werden. Dieser Kreuzschienenschalter ist eine Schaltmatrix, die es erlaubt, jeden beliebigen Eingang mit jedem beliebigen Ausgang zu verbinden. Optionell können auch mehrere Eingänge mit einem Ausgang (oder auch umgekehrt) verbunden werden. Diese Verbindungsmatrix kann dann jederzeit nach Bedarf umgeschaltet werden. Somit können einkommende Daten von den ADCs zu jeweils einem gering ausgelasteten Präprozessor weitergereicht werden .
In den Präprozessoren wird die Datenreduktion durchgeführt. Aus den Eingangsdaten, die die Kurvenform beschreiben, werden die Positionen und Höhen der einzelnen Cluster extrahiert. Diese Information steht nach der Verarbeitung am Ausgang der Präprozessoren zur Verfügung. Wahlweise kann auch nur die Startzeit des ersten Clusters zusammen mit der Gesamtzahl der gefundenen Cluster weitergegeben werden.
Diese gesamte Elektronik – Analog-Digital-Wandler, Kreuzschienenschalter und Präprozessoren für ein Segment des Detektors – wird in einen gemeinsamen Schaltschrank für die Cluster Counting-Elektronik eingebaut. Unter Verwendung der Elektronikmodule der verschiedenen Testaufbauten ließen sich in einem solchen Schaltschrank beispielsweise 36 ADCs für insgesamt 144 Kanäle zusammen mit 8 Präprozessorkarten mit jeweils 4 Prozessoren, einem Kreuzschienenschalter und den notwendigen Einschüben für die Kommunikation zwischen den unterschiedlichen Modulen unterbringen. Diese Integrationsdichte ist für den Einsatz am Beispieldetektor noch zu gering, bei 10000 Kanälen wären in einem solchen Aufbau 70 Schaltschränke alleine für die Cluster Counting-Auslese nötig.
Die Ausgänge der Präprozessoren aller Schränke werden über einen weiteren Schalter ähnlich einem der Kreuzschienenschalter mit einer schnellen Datenleitung verbunden. Dies kann zum Beispiel eine Glasfaserleitung (FDDI) sein.
Über diese eine Leitung werden die Ergebnisse der Präprozessoren zu den Analyserechnern, Bandlaufwerken und dem Steuerrechner im Kontrollraum übertragen. Während die vorher aufgeführten Baugruppen zweckmäßigerweise in der Nähe des Detektors untergebracht werden, kann eine sol-

[17] Die nominelle Auflösung dieses ADCs beträgt 8 Bit. Bei einer Sampling-Rate von 1 GS/s beträgt die effektive Auflösung noch 6.8 Bit. Die für den Testaufbau vorhandenen Prototypen arbeiteten intern mit einer Datenbreite von 10 Bit, so daß – als nächstgrößeres ganzzahliges Vielfaches von 8 – 16 Bit breite Datenworte übertragen und ausgewertet wurden.

[18] VME steht für **V**ersa **M**odule **E**urocard, einen Rechnerbus mit variabler Adress- und Datenbreite nach ANSI/IEEE STD1014-1987 sowie IEC 821 und 297 [43, 44]. VXI ist eine Erweiterung des VMEbus-Standards für meßtechnische Anwendungen mit höheren Ansprüchen

che Glasfaserleitung für praktische Zwecke nahezu beliebig lang sein (im Bereich bis etwa tausend Meter), so daß die Massenspeicher an einem gut zugänglichen Ort aufgebaut werden können.
Als Massenspeicher können die bereits erwähnten DLT-Bandlaufwerke verwendet werden. Bei einer Ausgangsdatenrate von 6 MB/s genügen zwei dieser Laufwerke, um den kontinuierlichen Datenstrom aufzunehmen. Eine solche Gruppe von Laufwerken hat damit bei einer Speicherdichte von 20 GB/Kassette eine Speicherkapazität von 40 GB, ohne daß ein Kassettenwechsel vorgenommen werden muß.
An den Steuerrechner werden, im Unterschied zu den Analyserechnern, keine besonderen Anforderungen gestellt. Hier kann im Prinzip jeder der derzeit üblichen Arbeitsplatzrechner zum Einsatz kommen.

# Kapitel 3

# Ein datenflußorientierter Präprozessor

Die in Abbildung 2.6 eingeführten Prozessoren zur Vorverarbeitung und Kompression der Rohdaten ('Präprozessoren') müssen nun, damit sich der Aufwand in finanziellen und räumlichen Grenzen halten läßt, einen möglichst hohen Datendurchsatz aufweisen.
Um diesen hohen Datendurchsatz durch den Präprozessor erreichen zu können, bieten sich zunächst folgende vier Möglichkeiten an:

- hohe Taktfrequenz
- Parallelverarbeitung
- Kopplung der einzelnen Verarbeitungsschritte in einer Pipeline[1]
- Anpassung der Prozessorstruktur an den verwendeten Algorithmus (die verwendeten Algorithmen)

Ein Beispiel für den letzten Punkt, die Anpassung der Prozessorstruktur an eine Klasse von Algorithmen, sind die sogenannten DSPs, **D**igitale **S**ignal**p**rozessoren, eine Gruppe von Prozessoren, die für Anwendungen in der digitalen Signalverarbeitung optimiert sind. Eines der Hauptelemente der gebräuchlichen Algorithmen in diesem Bereich sind FIR[2]- und IIR[3]-Filter, die in der Realisierung im Prozessor aus verketteten Multiplikationen mit Konstanten und Additionen bestehen. DSPs weisen demnach eine (oder auch mehrere) schnelle Multiplikation-und-Additions-Stufen[4] auf.
Die Struktur eines optimalen Algorithmus für Cluster Counting ist bisher noch nicht bekannt, aber Gegenstand unserer Untersuchungen; die Struktur der Daten selbst dagegen ist bekannt. Daraus lassen sich entsprechende Anforderungen an die Struktur des Präprozessors ableiten.

## 3.1 Prozessorarchitekturen

1966 wurde von M. J. Flynn [46] eine Klassifizierung für Computersysteme vorgeschlagen, die auch heute noch Verwendung findet. Diese unterteilt Computersysteme in vier Klassen:

[1] In einer Pipeline werden Prozessoroperationen in kleinere, unabhängige Teilschritte zerlegt, die für ein Datum nacheinander, für mehrere Daten in unterschiedlichen Stufen der Pipeline gleichzeitig, ausgeführt werden können.
[2] **F**inite **I**mpulse **R**esponse
[3] **I**nfinite **I**mpulse **R**esponse
[4] MAC, **M**ultiplier - **Ac**cumulator

**SISD** **s**ingle **i**nstruction stream, **s**ingle **d**ata stream (einfacher Instruktionspfad, einfacher Datenpfad) beschreibt eine Architektur, die in einem Zyklus eine Instruktion auf ein Datenwort anwendet, also beispielsweise

1. lade Datenwort A
2. lade Datenwort B
3. multipliziere A mit B
4. lade Datenwort C
5. addiere C zum Ergebnis von Schritt 3
6. speichere das Ergebnis von Schritt 5

und diesen Zyklus dann für einen Vektor von zum Beispiel 64 Elementen 64 mal wiederholt.

Eine Erweiterung dieser Struktur organisiert die einzelnen Prozessorelemente in einer Pipeline, um unabhängige Operationen, bei denen jeweils nur ein Teil der Prozessorstruktur beschäftigt ist, parallel auszuführen, wie zum Beispiel

1. lade Datenwort A
2. lade Datenwort B
3. multipliziere A mit B / lade Datenwort C
4. addiere C zum Ergebnis von Schritt 3
5. speichere das Ergebnis aus Schritt 4

Zur Verarbeitung eines Vektors können dann die Operationen der einzelnen Stufen noch ineinander geschachtelt werden:

1. ...
2. lade Datenwort A(i)
3. lade Datenwort B(i)
4. multipliziere A(i) mit B(i) / lade Datenwort C(i)
5. addiere C(i) zum Ergebnis von Schritt 3 / speichere das Ergebnis aus Schritt (i-1)
6. ...

Je nach dem Aufbau des Prozessors und nach der Anzahl der für eine Operation nötigen Schritte kann die Verschachtelung einzelner Operationen komplexer werden.

**SIMD** **s**ingle **i**nstruction stream, **m**ultiple **d**ata stream (einfacher Instruktionspfad, mehrfacher Datenpfad) beschreibt die Architektur der sogenannten Vektorrechner. Hier werden in einem Zyklus von einer Instruktion mehrere Datenworte synchron abgearbeitet, zum Beispiel

1. lade Datenworte A(1...64)
2. lade Datenworte B(1...64)
3. multipliziere A(1...64) mit B(1...64)
4. lade Datenworte C(1...64)

5. addiere C(1…64) zu den Ergebnissen aus Schritt 3
6. speichere die Ergebnisse aus Schritt 5

Auch hier können wieder Pipelines zur parallelen Abarbeitung unabhängiger Unterprozesse eingesetzt werden.

Der Vorteil dieser Architektur ist, daß ein solcher Rechner bei einer idealen Vektorgröße (in diesem Beispiel 64 Elemente) um den Faktor der Vektorlänge schneller ist als ein entsprechender SISD-Rechner. Bei einer ungünstigen Vektorgröße, also zum Beispiel 65 Elemente, reduziert sich dieser Vorteil aber schlagartig von 1:64 auf 2:65, relativ gesehen um etwa die Hälfte. Im zweiten Arbeitszyklus stehen dann 63 der 64 Verarbeitungseinheiten nutzlos herum ('ideln').

**MISD** **m**ultiple **i**nstruction stream, **s**ingle **d**ata stream (mehrfacher Instruktionspfad, einfacher Datenpfad). Diese Struktur fügt sich zwar symmetrisch in die anderen Strukturen ein, es gibt aber bislang keine reale Architektur, die dieser Struktur folgt. Dehnt man die Begriffe etwas, so ließen sich vielleicht Datenflußrechner [45] unter dieser Struktur einordnen (siehe Abschnitt 3.1.4).

**MIMD** **m**ultiple **i**nstruction stream, **m**ultiple **d**ata stream (mehrfacher Instruktionspfad, mehrfacher Datenpfad) schließlich beschreibt Parallelrechner, bei denen mehrere voneinander unabhängige Prozessoren oder Rechenknoten gleichzeitig mehrere unabhängige Datenstrukturen bearbeiten. Diese Struktur läßt sich aus der Parallelschaltung einfacher SISD-Rechner aufbauen. Je nach der Art und Weise, wie die einzelnen Knoten untereinander verbunden werden, zeigen diese MIMD-Rechner unterschiedliche Stärken und Schwächen. Ein allgemeines Problem dieser Parallelrechner aber bleibt eine möglichst effiziente Programmierung, die verhindert, daß einzelne Knoten längere Zeit nichts zu tun haben, weil sie auf Daten aus anderen Knoten warten müssen.

Von den Parallelrechnersystemen ist das letzte am einfachsten aufzubauen und am vielseitigsten. Ein Vektorrechner (SIMD) kann bei ähnlichem Aufwand zwar einen höheren Datendurchsatz erreichen, aber nur dann, wenn Rechnerarchitektur und Programm- und Datenstruktur gut aneinander angepaßt sind.

### 3.1.1 Grundstrukturen für MIMD-Parallelrechner

Um eine gute Rechnerstruktur für Cluster Counting zu finden, sollte man sich zuerst einmal mit dem Fluß und der Struktur der Daten befassen.
Bei Cluster Counting fallen nach der Digitalisierung und Nullunterdrückung pro Draht und Ereignis etwa 1 kByte an Rohdaten an. Multipliziert mit der hohen Anzahl an Drähten und der hohen Ereignisrate erhält man an dieser Stelle eine Rohdatenrate von 1 GByte/Sekunde. Diese Datenmenge besteht aber nach wie vor aus 1 Million einzelnen, unabhängigen Datensätzen von jeweils 1 kByte Größe. Gerade diese Datenstruktur ist ideal für die Verarbeitung in einem massiv parallelen MIMD-Rechner, da durch die geringe Größe der einzelnen Datensätze diese vollständig auf einem Knoten abgearbeitet werden können, ohne daß durch Abhängigkeiten zwischen den Datensätzen Kommunikationspfade zwischen den Rechenknoten nötig werden.
Solche Datenpfade von und zu den einzelnen Rechenknoten sind zwar weiterhin nötig, um die zu verarbeitenden Daten zu den Knoten zu bringen und die Ergebnisse dort abzuholen, der übliche

Engpaß eines MIMD-Rechners hingegen, die Gefahr, daß einer der Knoten nicht weiterarbeiten kann, weil durch die begrenzte Bandbreite der Datenübertragung zwischen den Knoten nötige Daten nicht übermittelt werden können, entfällt.
Für den Aufbau eines MIMD-Rechners sind verschiedene Topologien möglich und gebräuchlich [47, 48].

**Hyperwürfel ('Hypercubes')** In Hypercubes sitzen die einzelnen Prozessoren in der Vernetzungsstruktur auf den Ecken eines n-dimensionalen Würfels, dessen Kanten, eventuell auch Diagonalen, die Datenpfade bilden. Eine solche Struktur für vier Knoten mit einer gemeinsamen Ein-/Ausgabestruktur zeigt Abbildung 3.1.

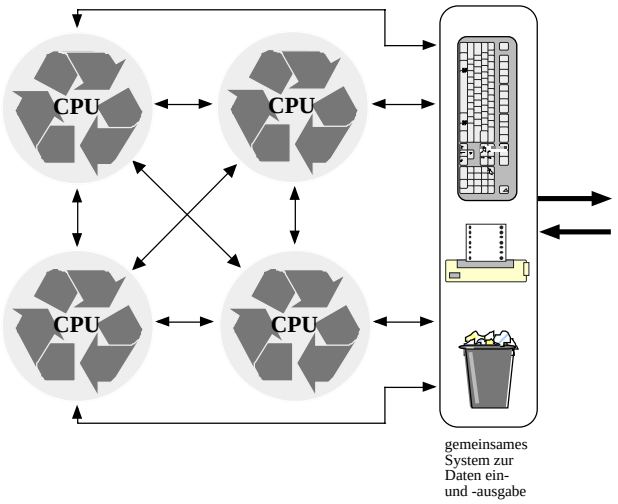



Abbildung 3.1: Ein Hypercube-Netzwerk mit vier Rechenknoten und einer gemeinsamen Ein-/Ausgabeeinheit

Ein solches Netzwerk weist im Vergleich zu den anderen Topologien die höchste Kommunikationsbandbreite auf – jeder Knoten hat eine direkte und unabhängige Verbindung zu jedem anderen Knoten. Dies stellt auch gleichzeitig das größte Problem dieser Topologie dar: der Aufwand für das Kommunikationsnetz wächst nahezu quadratisch[5] mit der Anzahl der Knoten, und was für einen 2-dimensionalen 'Würfel' mit vier Knoten und 6 Verbindungen noch recht übersichtlich ist, wird für einen 6-dimensionalen Hyperwürfel mit 64 Knoten und 2016 Verbindungen dagegen sehr aufwendig.

**Zweidimensionale Gitter** In einem zweidimensionalen Gitter sitzen die Knoten in der Vernetzungsstruktur auf den Kreuzungspunkten eines solchen Gitters. Ein spezifischer Knoten ist also direkt nur mit seinen nächsten Nachbarn verbunden und nicht, wie bei einem Hypercube, mit jedem anderen Knoten des Netzes. Der Vorteil hierbei liegt in der einfacheren Struktur des Netzes, der Nachteil in der verringerten Bandbreite für die Datenübertragung.

**Kreuzschienenschalter ('Crossbar Switch')** Eine weitere Möglichkeit, ein Prozessornetzwerk aufzubauen, ist, die Rechenknoten über einen Kreuzschienenschalter (Abbildung 3.2) miteinander zu verbinden.

[5] Ist n die Zahl der Knoten, wächst die Zahl der Verbindungen mit $\frac{n*(n-1)}{2}$.

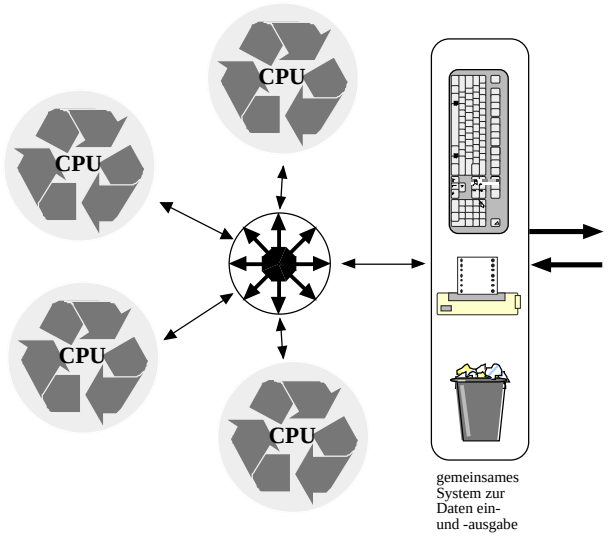



Abbildung 3.2: Vier über einen Kreuzschienenschalter verknüpfte Rechenknoten mit einer gemeinsamen Ein-/Ausgabeeinheit

In einer solchen Struktur übernimmt der Kreuzschienenschalter die Verschaltung der Kommunikationspfade zwischen den Prozessoren.

Damit erreicht man mit dieser Technologie eine ähnliche Kommunikationsbandbreite wie mit einem voll vernetzten Hypercube, der praktische Aufbau ist in der Regel aber einfacher als der eines zweidimensionalen Netzes, da pro Rechenknoten nur eine Verbindung zur Datenkommunikation nötig ist.

Im Prinzip gleicht ein solcher Kreuzschienenschalter dem in Abschnitt 2.3 vorgestellten Element, allerdings mit der Einschränkung, daß dieser Schalter hier nur eindeutige Verbindungen herzustellen braucht, das heißt, Verschaltungen eines Knotens mit mehreren anderen Knoten werden nicht benötigt.

**Ringe** In der nächsteinfacheren Netztopologie sind die einzelnen Rechenknoten in einem Ring miteinander verbunden. Dabei hat jeder der Knoten Kontakt mit seinem linken und rechten Nachbar. Die Kommunikationsbandbreite für Datenübermittlungen von einem Prozessor zu diesen Nachbarn kann sehr hoch sein. Folgt die Verdrahtung des Ringes einfach nur dem Schema einer Busverdrahtung (s.u.), so kann auch nur immer einer der Teilnehmer als Sender aktiv sein. Bestehen die Verbindungen dagegen immer nur zwischen zwei benachbarten Knoten, so können zwar mehrere unabhängige Kommunikationen parallel ablaufen, bei jeder Verbindung zwischen zwei nicht direkt benachbarten Knoten werden jedoch Ressourcen der dazwischenliegenden Knoten in Anspruch genommen.

**Busse** Als einfachste Topologie für ein Multiprozessornetzwerk bleibt am Schluß noch der Bus übrig (Abbildung 3.3).

Dieses Netzwerk hat die geringste Kommunikationsbandbreite der betrachteten Topologien, gleichzeitig aber auch den einfachsten Aufbau. Alle Rechenknoten kommunizieren über einen gemeinsamen Bus, so daß immer nur zwei Knoten miteinander, oder ein Knoten mit

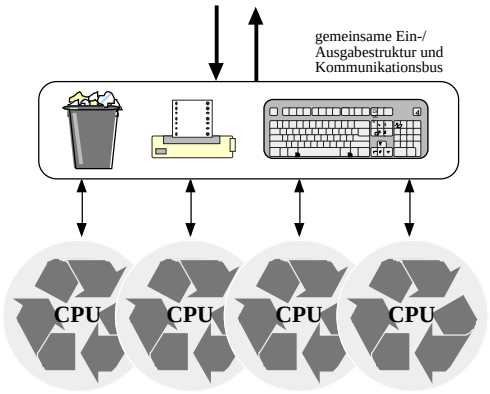



Abbildung 3.3: Vier über einen gemeinsamen Bus verbundene Rechenknoten; die Ein-/Ausgabeeinheit ist Teil des Bussystems

der Ein-/Ausgabeeinheit, sprechen können. Während dieser Zeit ist die Kommunikation für alle anderen Knoten gesperrt.

Ist allerdings im Rahmen der zu behandelnden Probleme die für Ein-/Ausgabeoperationen und Kommunikation zwischen den Knoten benötigte Zeit klein im Vergleich zur Verarbeitungszeit in einem Knoten, so kann der Vorteil des einfachen Aufbaus den Nachteil der geringen Kommunikationsbandbreite wieder aufwiegen.

Beim Entwurf eines für eine bestimmte Anwendung vorgesehenen MIMD-Parallelrechners sollte berücksichtigt werden, ob der hohe Aufwand für eine Datenkommunikationsstruktur mit einer hohen Bandbreite gerechtfertigt ist, oder ob sich durch denselben Aufwand an anderer Stelle nicht vielleicht eine höhere Leistung erzielen läßt.

### 3.1.2 Amdahls Gesetz

Formal wird dieser Zusammenhang durch das Amdahl'sche Gesetz [47] ausgedrückt. Dieses Gesetz besagt, daß für eine gegebene Änderung einer Prozessorstruktur im Vergleich zu einer anderen Struktur die

$$\text{Ausführungszeit nach der Änderung} = \left(\frac{\text{Durch die Änderung beeinflußte Ausführungszeit}}{\text{Faktor der Änderung}} + \text{Nicht beeinflußte Ausführungszeit}\right) \tag{3.1}$$

ist.
Als einfaches Beispiel sei von einem Programm ausgegangen, das auf einer gegebenen Rechnerstruktur eine Laufzeit von 100 Sekunden benötigt. Von diesen 100 Sekunden seien 90 Sekunden für die Datenverarbeitung nötig, die restlichen 10 Sekunden für die Ein- und Ausgabeoperationen. Wird nun die Rechenleistung dieses Prozessors um einen Faktor 10 gesteigert, so verringert sich die Laufzeit des Programmes nicht um einen Faktor 10 auf 10 Sekunden, sondern nur auf $\frac{90}{10}$ s + 10 s = 19 s, also ungefähr um einen Faktor 5. Wird nur die Geschwindigkeit der Ein-/Ausgabeoperationen um diesen Faktor 10 gesteigert, so ist der Gesamtgewinn noch geringer: von 100 s auf 90 + $\frac{10}{10}$ s = 91 s. Erst wenn sowohl die Verarbeitungsleistung als auch die

Ein-/Ausgabeleistung gesteigert werden, kann eine Verringerung der Laufzeit um den Faktor 10 erreicht werden.
Diese Zahlen gehen allerdings von der unrealistischen Annahme aus, daß es möglich sei, die Ein-/Ausgabebandbreite eines Parallerechnersystems genauso beliebig zu erhöhen wie die Rechenleistung. In der Realität bricht diese Vorstellung spätestens bei der Schnittstelle zwischen dem Parallelrechner und dem 'Rest der Welt' zusammen.
Aber selbst unter der Annahme, daß eine solche Steigerung der Leistung eines Rechnersystems möglich sei, zeigt dieses Gesetz, daß sich der Einsatz des entsprechenden Aufwandes nur an manchen Stellen lohnt, während an anderen Stellen kein noch so großer Aufwand eine nennenswerte Steigerung der Verarbeitungsleistung ermöglicht.

### 3.1.3 Harvard- und von Neumann-Struktur

Abgesehen von den unterschiedlichen Möglichkeiten, die Daten- und Instruktionsströme eines Prozessors miteinander zu verknüpfen sowie mehrere Prozessoren miteinander zu verbinden, gibt es auch (mindestens) zwei prinzipiell unterschiedliche Methoden, diese Daten- und Instruktionsströme zum Prozessorkern selbst zu transportieren.
Im ersten Fall werden Daten und Instruktionen in einem gemeinsamen Speicher abgelegt und über einen gemeinsamen Bus zum Prozessor transportiert (Abbildung 3.4.a). Seit dem Bau der ersten programmierbaren Digitalrechner gegen Ende des zweiten Weltkrieges werden Rechner mit dieser Architektur als von-Neumann-Rechner bezeichnet. ENIAC war der erste bekannte Vertreter dieser Gattung [47].

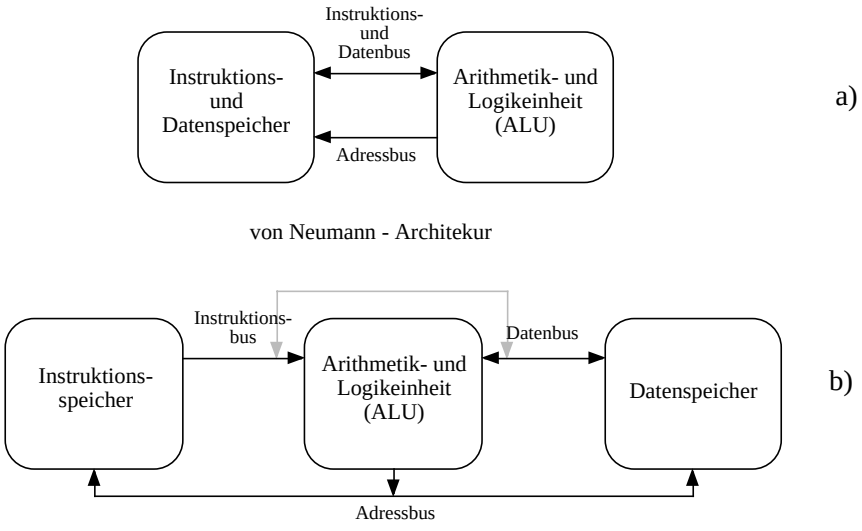



Abbildung 3.4: Vergleich von von Neumann- und Harvardarchitektur

Wenige Jahre nach dem Bau von ENIAC wurde in Harvard beim Bau des Mark-III-Rechners eine andere Architektur realisiert. Bei dieser Harvard-Architektur werden Daten und Instruktionen in getrennten Speicherbereichen abgelegt, und dann auch über getrennte Busse zum Prozessorkern geführt (Abbildung 3.4.b).
Auf den ersten Blick mag dies vielleicht einfach nur eine theoretische Klassifizierung sein, dieser

Unterschied hat jedoch weitreichende Konsequenzen, sowohl für den Bau als auch später für die Programmierung eines mit einem solchen Prozessor aufgebauten Rechners.
Zunächst einmal ist offensichtlich, daß beim Bau eines Rechners mit einem Prozessor mit Harvard-Architektur zwei getrennte Speicher für Daten und Instruktionen benötigt werden, während für einen von-Neumann-Prozessor nur ein gemeinsamer Speicher für Daten und Instruktionen benötigt wird. Soll bei einem Rechner mit Harvard-Prozessor der Prozessor selbst Zugriff auf den Instruktionsspeicher erhalten, muß eine zusätzliche externe Verbindung zwischen Instruktions- und Datenbus vorgesehen werden. Im allgemeinen ist also der Aufbau eines Rechners mit Harvard-Architektur aufwendiger als der eines Rechners mit von Neumann-Architektur.
Dieser kompliziertere Aufbau hat allerdings auch Vorteile. So ist es, sofern keine Querverbindung zwischen Instruktions- und Datenbus vorgesehen wird, zunächst einmal nicht möglich, daß fehlerhafte Zugriffe auf den Datenbereich Inhalte des Instruktionsspeichers zerstören. In der Praxis ist dieser Punkt allerdings von untergeordneter Bedeutung, da in der Regel diese Querverbindung in der einen oder anderen Weise vorgesehen sein wird, um zum Beispiel die Ausführung von selbstmodifizierenden Code oder auch (bei einem Einprozessorsystem) das Laden des eigenen Programms aus einem langsamen Massenspeicher zu ermöglichen. Hinzu kommt, daß bei Stack[6]-orientierten Programmiersprachen wie zum Beispiel C oder Pascal dieser Stack, zusammen mit Zeigeradressen und anderen Systemvariablen, im Datenspeicherbereich liegen muß und damit auch von fehlgeleiteten Datenzugriffen beschädigt werden kann, was normalerweise zum Totalabsturz des Systems führt.
In modernen Systemen liegt der Hauptvorteil der Harvard-Architektur in einem anderen Bereich. Ein Harvard-Prozessor kann, bei gleicher Taktfrequenz, im Prinzip bis zu doppelt so schnell arbeiten wie ein von-Neumann-Prozessor. Voraussetzung für diese Geschwindigkeitssteigerung ist zum Einen eine – mittlerweile gebräuchliche – Pipeline-RISC[7]-Architektur, die pro Taktzyklus eine Prozessorinstruktion entgegennimmt, sowie lokale Adressgeneratoren in den Speichermodulen, die einen Burst[8]-Zugriff ermöglichen.
In diesem Fall dient der Adressbus nur dazu, jeweils zum Beginn eines Bursttransfers eine Adresse an den Instruktions- bzw. Datenspeicher zu übermitteln, worauf dann dieser Speicher selbständig taktsynchron die nächsten Instruktionen oder Daten an den Prozessor liefert. Die Zugriffe auf Adressen und Daten laufen danach ineinander verschachtelt ab. Für einen 4-Wort[9]-Burst kann dieses Schema beispielsweise so ablaufen wie in Tabelle 3.1.
Im Unterschied dazu muß ein von Neumann-Prozessor für jede Instruktion zweimal auf den Bus zugreifen: das erstemal, um die Instruktion selbst zu laden, und das zweitemal, um ein Datum zu

[6] ‘Stapel’, auch unter der Bezeichnung ‘Heap’ (‘Haufen’) anzutreffen; Bezeichnung für einen Speicherbereich für (in der Regel) temporäre, systeminterne Variablen, dessen Größe nicht von vorneherein festliegt. Neu hinzukommende Variablen werden ‘oben’ auf den Stack gelegt, und müssen dann auch wieder als erste entfernt werden, um den Zugriff auf ‘tiefer’ liegende Variablen zu ermöglichen.

[7] Zu Pipeline siehe Abschnitt 3.1, RISC = **R**educed **I**nstruction **S**et **C**omputer, Computer mit reduziertem Instruktionsumfang. Ursprünglich Prozessoren mit gegenüber den zum damaligen Zeitpunkt gängigen ‘CISC’-Prozessoren (**C**omplex **I**nstruction **S**et **C**omputer) stark verringerter Anzahl von Prozessorinstruktionen. Zum Ausgleich dafür werden diese wenigen Instruktionen alle innerhalb eines Taktes abgearbeitet, anstatt in bis zu einigen Dutzend Takten. Dieses erste Merkmal des verringerten Instruktionsumfangs ist mittlerweile wieder verschwunden – moderne RISC-Prozessoren kennen ähnlich viele Prozessorinstruktionen wie CISC-Prozessoren, geblieben ist die Fähigkeit, einen Befehl innerhalb eines Prozessorzyklus abzuarbeiten.

[8] ‘Ausbruch’, Datentransfer, bei dem nur für das erste Datum eine Adresse übertragen wird, worauf dann von der Datenquelle aus ein Datenblock mit üblicherweise 4…256 Datenworten taktsynchron übertragen wird.

[9] Ein Wort ist bei moderneren Prozessoren die kleinste normalerweise adressierbare Einheit. Die meisten Prozessoren arbeiten mittlerweile mit Worten von 4, 8 oder 16 Bytes, also 32, 64 oder 128 Bit.

| Adreßbus | Instruktionsbus | Datenbus |
|---|---|---|
| Startadresse Instr. | – | – |
| – | 1. Instruktion | – |
| Startadresse Daten | 2. Instruktion | – |
| – | 3. Instruktion | 1. Datenwort |
| – | 4. Instruktion | 2. Datenwort |
| Startadresse Instr. | – | 3. Datenwort |
| – | 1. Instruktion | 4. Datenwort |
| Startadresse Daten | 2. Instruktion | – |
| – | 3. Instruktion | 1. Datenwort |
| ⋮ | ⋮ | ⋮ |

Tabelle 3.1: Verschachtelung der Instruktions- und Datenzugriffe bei einem Prozessor mit Harvard-Architektur

dieser Instruktion zu laden. Als Ausgleich dafür benötigt man nur einen Speicher für Daten und Instruktionen, und demzufolge ist auch der Aufwand für die notwendigen Busse geringer.

### 3.1.4 Datenflußrechner [45]

Datenflußrechner benötigen im Gegensatz zu den Rechnern mit Prozessoren in Harvard- oder von-Neumann-Struktur keine Speicher. Bei diesen Rechnern werden einzelne Verarbeitungselemente so angeordnet, daß die Daten durch diese Elemente 'hindurchfließen' können, das heißt, die Ausgabe einer Verarbeitungsstufe wird direkt als Eingabe einer weiteren Stufe verwendet. Sollte es durch Abhängigkeiten zwischen den Daten nicht möglich sein, dieses Datum sofort weiterzuverarbeiten, muß es in dieser Verarbeitungsstufe auf die anderen notwendigen Daten warten.
Solche Rechner können für einen gegebenen Algorithmus einen maximalen Datendurchsatz erreichen. Der eine der beiden Hauptgründe dafür liegt darin, daß keine Zeit dafür aufgewendet werden muß, die Daten und Instruktionen aus den entsprechenden Speichern zu einer zentralen Verarbeitungseinheit zu transportieren. Zum Zweiten kann ein solcher Rechner als Pipeline aufgebaut werden, so daß mit jedem Taktimpuls nach der Füllung der Pipeline ein neues Ergebnis aus der letzten Verarbeitungseinheit entnommen werden kann.
Rechner in diesem Aufbau haben allerdings auch zwei große Nachteile. Ein Rechner einer bestimmten Struktur, der für einen festgelegten Algorithmus aufgebaut ist, kann keine anderen Algorithmen zur Verfügung stellen, da hierfür ja erst wieder der Aufbau des Rechners geändert werden müßte. Zum Zweiten wird hier ein sehr hoher Geräteaufwand nötig, da alle die Aufgaben, die ein Rechner mit einer konventionelleren Architektur der Reihe nach abarbeitet, in diesem Rechner parallel ablaufen, oder doch zumindest zur Verfügung stehen müssen.
Abbildung 3.5 zeigt ein einfaches Beispiel für einen solchen Datenflußrechner, der einfach die Summe zweier Zahlen mit einer dritten Zahl multipliziert.

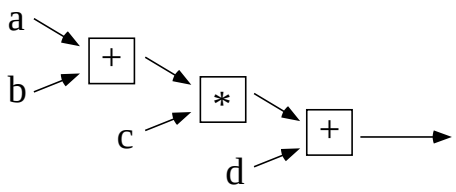



Abbildung 3.5: Beispiel für einen einfachen Datenflußrechner für die Operation ‘((a + b) * c) + d ’.

Obwohl der Aufbau eines vollständigen Datenflußrechners durch den hohen Aufwand und die eingeschränkte Flexibilität nur in den wenigsten Fällen sinnvoll sein wird, lassen sich einige Teilaspekte dieses Prinzips, wie die Verwendung von Pipelinestrukturen, vorteilhaft auch bei anderen Rechnern einsetzen.

## 3.2 Der Rechner im Experiment

Nach den Überlegungen zum Aufbau eines alleinstehenden Rechners im vorangegangenen Abschnitt muß nun auch die Integration eines solchen Rechners in ein physikalisches Experiment berücksichtigt werden. Ein Rechner zur Datenvorverarbeitung oder Datenkompression kann hier nicht als alleinstehendes System betrachtet werden, sondern muß als Teil eines Gesamtsystems zur Datennahme angesehen werden. Dies bedingt sowohl eine Kommunikationsschnittstelle, über die Zustandsinformationen unabhängig vom Fluß der Meßdaten ausgetauscht werden können, als auch gewisse Mindestanforderungen an die Reaktionszeiten auf von außen kommende Signale.

### 3.2.1 Totzeitfreie Auslese

Eine dieser zeitlichen Anforderungen bezieht sich auf die sogenannte ‘Totzeit’ eines solchen Datennahmesystems. Dies ist die Zeit, die ein solches System im Mittel nach dem Eintreffen eines Datensatzes ‘tot’ ist, bevor es den nächsten Datensatz akzeptieren kann. Treffen die Datensätze kontinuierlich ein, gehen die während der Totzeit des Systems eintreffenden Daten verloren. Um diesen Datenverlust möglichst klein zu halten, sollte demnach auch die Totzeit des Datennahmesystems möglichst klein sein. Im Idealfall sollte das System immer in der Lage sein, neue Daten zu akzeptieren, also totzeitfrei arbeiten. Das Problem liegt hierbei aber darin, daß die in der Hochenergiephysik zu messenden Abläufe normalerweise auf Poisson-Prozessen basieren und damit der zeitliche Abstand zweier aufeinanderfolgender Ereignisse einer Exponentialverteilung folgt. Damit ist der Aufwand für eine echte totzeitfreie Auslese im Vergleich zu einer Auslese, die gerade die mittlere Datenrate bewältigen kann, immens.
Aus diesem Dilemma gibt es, je nach Art der zu messenden Daten, mehrere Auswege. Einer davon ist, dem Auswertesystem Pufferstufen vorzuschalten, die die starken Schwankungen dieser exponentiellen Verteilung dämpfen [49]. Die Auswirkungen dieser Pufferstufen zeigt Abbildung 3.6.
Bei dieser Abbildung gilt zu beachten, daß die Bezeichnung ‘Puffer’ für alle Stufen gilt, die in der Lage sind, Daten anzunehmen, also auch die eigentliche Verarbeitungsstufe. Der Wert 1 auf der x-Achse entspricht also einer pufferlosen Auslese, bei der nur die eigentliche Verarbeitungsstufe vorhanden ist, ab einem x-Wert von 2 werden dann zusätzliche Pufferstufen berücksichtigt.

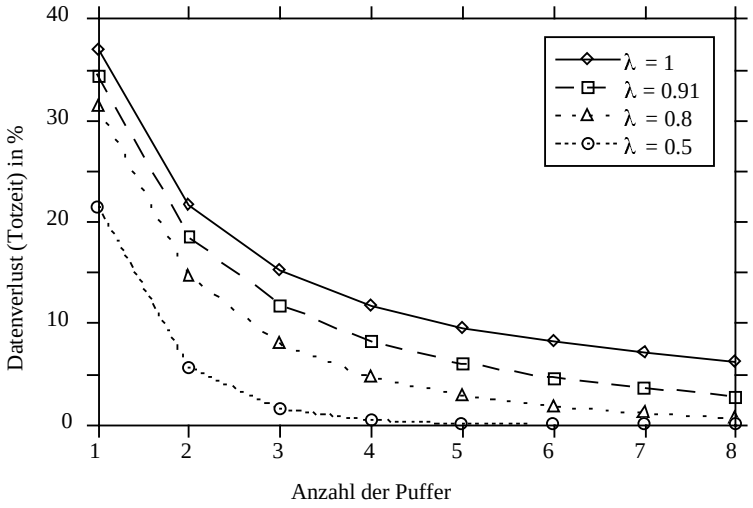



Abbildung 3.6: Die Auswirkungen von Pufferstufen auf Datenverluste bzw. die Totzeit eines Datennahmesystems. Der Parameter $\lambda = \frac{\text{Eingangsdatenrate}}{\text{Ausgangsdatenrate}}$ gibt die mittlere Datenrate am Eingang, normiert auf die Ausgangsdatenrate an. Bei der Anzahl der Pufferstufen zählt die Verarbeitungsstufe ebenfalls als ein Puffer.

Die der Simulation für dieses Diagramm zugrundeliegende Schaltung zeigt Abbildung 3.7.

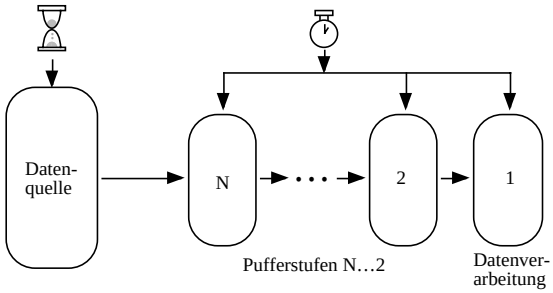



Abbildung 3.7: Schematische Darstellung der der Simulation für Abbildung 3.6 zugrundeliegenden Schaltung.

Sie besteht aus einer Datenquelle, deren Ausgangsdatenrate Poisson-verteilt um einen Mittelwert schwankt und einer Reihe von Pufferstufen, aus deren letzter mit einer konstanten Rate, d.h. in festgelegten Abständen, Datensätze entnommen werden. Das Verhältnis zwischen der mittleren Erzeugungsrate der Datenquelle und der Entnahmerate aus der letzten Pufferstufe beschreibt der Parameter $\lambda$.
Zunächst ist zu sehen, daß ohne Pufferstufen und bei gleicher mittlerer Rate für Entnahme und Erzeugung der Daten ein großer Teil der Daten verlorengeht. Aber auch eine Vergrößerung der Entnahmerate um eine Faktor zwei bringt hier keine große Verbesserung. Effektiver ist allerdings der Einsatz von mindestens zwei Pufferstufen, wobei bei mehr als vier Pufferstufen der Effekt

weiterer Puffer kaum noch ausgeprägt ist. Den größten Effekt zeigt wie zu erwarten eine Kombination dieser Maßnahmen. Hier ist dann aber wieder das Preis-Leistungsverhältnis zu beachten. Besteht die Verarbeitungseinheit, wie hier angenommen, aus einem modularen Parallelrechner, so skaliert die Entnahmeleistung aus der Pufferstufe annähernd linear mit dem Gesamtpreis des Systems: für die doppelte Entnahmerate wird auch die doppelte Anzahl an Rechenknoten benötigt. Eine zusätzliche Pufferstufe macht unter Umständen nur einen Bruchteil dieses Aufwandes aus.
Eine ähnliche Wirkung wie durch die Vorschaltung zusätzlicher Pufferstufen läßt sich auch die Kombination möglichst großer Segmente des Detektors und der Ausleseelektronik erreichen. Je höher die Anzahl der Detektorkanäle ist, die auf die entsprechende Anzahl von Präprozessoren verteilt wird, desto geringer werden die Schwankungen dieser Datenrate. In der Praxis stößt man hier aber rasch an Grenzen, die verhindern, daß die Daten des gesamten Detektors über einen zentralen Verteiler auf die Gesamtheit der Präprozessoren verteilt wird, so daß es hier unumgänglich ist, mit kleineren Untergruppen zu arbeiten.
Aber auch in diesem Fall können durch eine geschickte Aufteilung des Detektors die Schwankungen der Datenrate und damit der nötige Aufwand zur Einhaltung einer gewissen Totzeit reduziert werden. In einem symmetrischen Detektor wie KLOE bestehen die interessanten Ereignisse aus Teilchen, die sich (in der Ebene senkrecht zur Strahlachse) radial vom Wechselwirkungspunkt entfernen. Gruppiert man nun die Driftzellen längs eines Radius zu einer Auslesegruppe zusammen, so wird in den größten Zahl der Ereignisse keine der Zellen der Gruppe ausgelesen werden müssen. Trifft ein Teilchen aber eine Zelle der Gruppe, so werden in den meisten Fällen auch alle anderen Zellen dieser Gruppe davon betroffen sein und ausgelesen werden müssen (Abbildung 3.8).

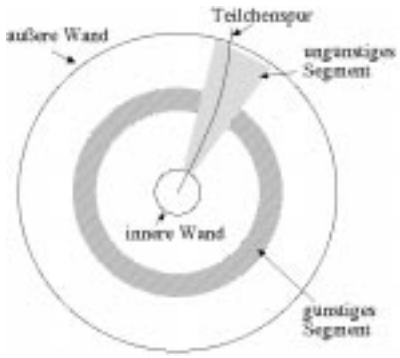



Abbildung 3.8: Beispiele für eine ‘günstige’ und eine ‘ungünstige’ Segmentierung eines Detektorauslesesystems

Im Falle der ‘günstigen’ Segmentierung in Abbildung 3.8 wird das Segment hingegen bei jedem Ereignis getroffen werden, dafür ist aber nur eine geringe Zahl der Zellen in diesem Segment auszulesen. Bei entsprechender Wahl der Segmente wird die mittlere Datenrate pro Segment in beiden Fällen dieselbe sein. Im Falle der günstigeren Segmentierung werden diese Daten aber im wesentlichen mit einer konstanten, niedrigen Rate geliefert, während sich bei der ungünstigeren Segmentierung dieselbe mittlere Datenrate aus kurzen Bursts mit einer hohen Datenrate, gefolgt von langen Pausen ohne jeglichen Datenfluß zusammensetzt.
Damit können im Falle einer günstigen Segmentierung die Daten gewissermaßen ‘fließend’ weiterverarbeitet werden, während im anderen Fall entweder entsprechend große Pufferstufen notwendig sind, oder aber eine Überkapazität in der Verarbeitung bereitgestellt werden muß, die dann über

einen großen Anteil der Zeit brachliegt.

### 3.2.2 Datenverarbeitung und Haushaltsführung

Neben dem Strom der Meßdaten müssen in einem Datennahmesystem auch Steuerinformationen ausgetauscht werden. Speziell bei einem Parallelrechnersystem ist eine der Hauptaufgaben dieses Steuerungssystems die Verteilung der einkommenden Rohdaten auf die verschiedenen parallelen Verarbeitungsknoten, so daß eine möglichst gleichmäßige Auslastung dieser Knoten gewährleistet wird.
Eine weitere Aufgabe eines solchen Steuersystems ist die Koordinierung gemeinsam genutzter Ressourcen. Darunter können Beispielsweise Ein- und Ausgabeschnittstellen oder auch Datenpuffer und Massenspeicher fallen.
Diese Aufgaben könnten natürlich auch von einem der Rechenknoten übernommen werden, oder auch gemeinsam von allen Rechenknoten. Der Vorteil eines solchen Aufbaus wäre, daß keine zusätzliche Hardware benötigt wird, der Nachteil, daß, je nach Vergabe der Prioritäten, entweder die Hausarbeit die Datenverarbeitung unterbricht, oder die Hausarbeit liegen bleibt, bis einer der Rechenknoten die Zeit hat, sich darum zu kümmern. Für ein im Verbund mit anderen Systemen arbeitendes Datennahmesystem ('Echtzeitsystem'[10]) ist letztere Option weniger praktikabel, da in einem solchen Verbund in der Regel externe Anfragen innerhalb garantierter Höchstzeiten beantwortet werden müssen, und im ungünstigsten Fall diese Antwortzeit gleich der Verarbeitungszeit für einen Datensatz wird. Soll während dieser Zeit etwa entschieden werden, wohin der nächste Datensatz zur Verarbeitung geleitet werden soll, wird diese Antwortzeit in der Praxis zu lange.
In einem solchen Fall muß also der Hausarbeit eine höhere Priorität eingeräumt werden. Damit wird aber die Zeit, die ein Knoten für die Verarbeitung eines Datensatzes benötigt, nicht mehr vorhersehbar. Im Extremfall leistet dann einer der Knoten kaum noch Arbeit für den Datendurchsatz, sondern ist nur noch mit der Haushaltsführung sowie der Umschaltung zwischen den beiden Tätigkeiten beschäftigt.
Es empfiehlt sich also, in einem schnellen Echtzeitsystem einen eigenen Prozessor für die Haushaltsführung vorzusehen. Hier kann ein Prozessor mit geringerer Rechenleistung eingesetzt werden, er benötigt aber vollen Zugriff auf alle Ressourcen des gesamten Prozessorsystems sowie Kommunikationspfade zu allen Rechenknoten.

### 3.2.3 Interne Datenverteilung - Die Datenstruktur bei Cluster Counting

Das Ziel eines Präprozessors für Cluster Counting ist es auch, den notwendigen Datendurchsatz mit einem möglichst geringen Aufwand zu erzielen. Um diesen hohen Datendurchsatz erreichen zu können, muß die zugrundeliegende Datenstruktur berücksichtigt werden.
Wie bereits erwähnt bestehen die Rohdaten aus einer großen Zahl (etwa 1 Million pro Sekunde) kleiner (etwa 1 kByte) und vor allem unabhängiger Datenblöcke. Die Unabhängigkeit dieser Datenblöcke heißt in diesem Fall, daß jeder Datenblock zunächst getrennt von allen anderen Datenblöcken verarbeitet werden kann. Ein Zusammenhang zwischen den Signalen der einzelnen Driftzellen wird erst durch die spätere Spurauswertung hergestellt.

[10] Als 'Echtzeitsystem' wird im allgemeinen ein System bezeichnet, daß in 'Echtzeit' antworten kann, also für alle praktischen Zwecke 'schnell genug' ist. Wie schnell 'schnell genug' ist, hängt vom jeweiligen Einsatz des Systems ab. Eine Bedingung ist allerdings, daß solche Systeme maximale Antwortzeiten garantieren können, das heißt, auf eine externe Anfrage kommt unter allen möglichen Umständen innerhalb einer definierten Zeit eine gültige Antwort.

Durch die geringe Größe der einzelnen Datensätze kann aber ein solcher Datensatz vollständig auf einem Rechenknoten mit relativ geringem Speicherbedarf verarbeitet werden. Die Unabhängigkeit der einzelnen Datensätze voneinander bedeutet, daß keine aufwendigen Querverbindungen zum Datenaustausch zwischen den Rechenknoten notwendig werden.
Nach der Verarbeitung im Präprozessor besteht ein solcher Datensatz noch aus etwa 10 Bytes. Damit wird für die Ausgangsdaten auch nur noch ein entsprechender Bruchteil der Busbandbreite der Eingangsdaten benötigt. Da der Aufbau des Präprozessors am Datenfluß orientiert werden soll, bietet es sich unter diesen Umständen an, zwei oder sogar drei getrennte Busse zu verwenden: einen Dateneingangsbus hoher Bandbreite, einen Datenausgangsbus entsprechend niedrigerer Bandbreite sowie einen dritten Bus für die Haushaltsführung.
Da in diesem Fall keine Querverbindungen zum Datenaustausch zwischen den Prozessoren notwendig sind, ist eine einfache Bustopologie für einen solchen Parallelrechner ausreichend. Ein Blockschaltbild für einen solchen Aufbau mit vier Rechenknoten zeigt Abbildung 3.9.

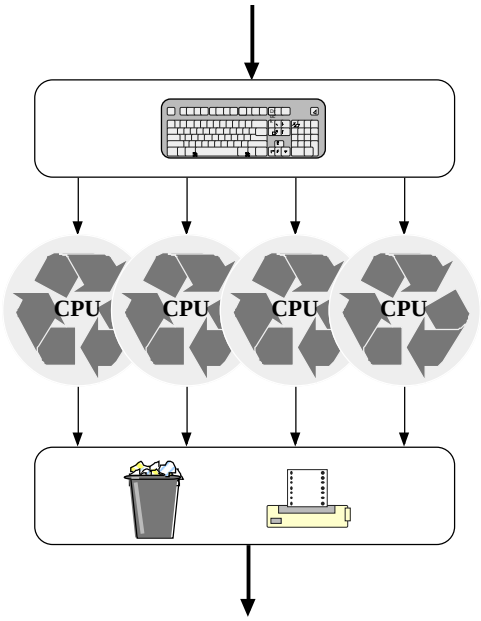



Abbildung 3.9: Vier Rechenknoten, die in einer parallelen Pipeline aufgebaut sind. Eingabeeinheit und Ausgabeeinheit sind hierbei getrennte Systeme.

Im Unterschied zur Busstruktur aus Abbildung 3.3 besitzt der Rechner in Abbildung 3.9 getrennte Eingangs- und Ausgangsstrukturen. Damit wird es möglich, einen schnellen Eingangsdatenbus und einen langsameren Ausgangsdatenbus zu verwenden.
Ein Vorteil dieser Struktur ist, daß der schnellere (und damit auch aufwendigere und teurere) Eingangsdatenbus nur für eine Richtung des Datenflusses aufgebaut werden muß. Prinzipiell wäre ein solcher unidirektionaler Aufbau auch beim Ausgangsdatenbus möglich, wenn für die Haushaltsführung ein separater dritter Bus zur Verfügung steht. Verwendet man für den Datenausgangsbus allerdings einen der gängigen Industriestandard-Busse, zum Beispiel den VMEbus, so steht hier genug Bandbreite für den Datenausgang zusammen mit der Haushaltsführung zur Verfügung. Für dieses Bussystem gibt es auch eine große Auswahl an industriell erhältlichen Chipsätzen zur Anbindung einer Prozessorplatine an den Bus.
Fügt man zu dieser Struktur noch einen Eingangsdatenpuffer (Abschnitt 3.2.1) und einen Prozessor

zur Haushaltsführung hinzu, ergibt sich eine Struktur wie in Abbildung 3.10.

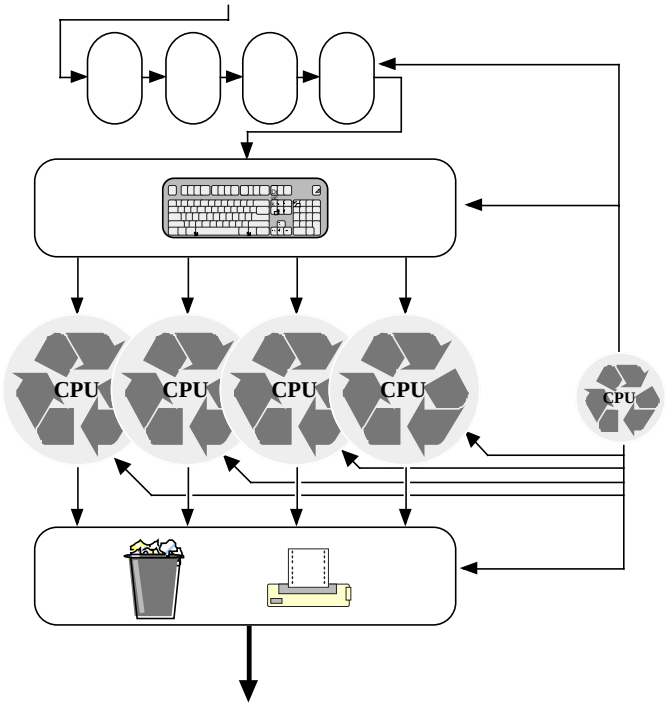



Abbildung 3.10: Parallelrechner mit 4 Knoten, Eingangsdatenpuffer, getrennten Ein- und Ausgangsdatenbussen und eigener CPU zur Haushaltsführung

## 3.3 Die Realisierung eines Präprozessors für Cluster Counting

Um ein Demonstrationssystem mit einem Präprozessor für Cluster Counting aufzubauen, mußte zunächst eine geeignete CPU[11] gefunden werden. Da an der Universität Roma II noch ein Entwicklungssystem für die Am29k™-Serie des Herstellers AMD (siehe Anhang A) zur Verfügung stand, fiel die Wahl auf einen Chip dieser Serie, obwohl sowohl im Bezug auf die Architektur als auch auf die Rechenleistung bereits zum damaligen Zeitpunkt bessere Systeme erhältlich gewesen wären.

### 3.3.1 Die Prozessoren Am29000™/Am29050™[51, 52, 53, 54]

Bei den Prozessoren Am29000™und Am29050™handelt es sich um RISC-Prozessoren, die in Harvard-Architektur aufgebaut sind (siehe Anhang A). Durch den Aufbau in Harvard-Architektur müssen diese Prozessoren im Prinzip auch mit mindestens zwei getrennten Speichersystemen, einem für Instruktionen und einem für Daten, ausgestattet werden. In der Praxis ist es zwar möglich, die Instruktionen im Datenspeicher abzulegen, sofern extern eine entsprechende Querverbindung

[11] eigentlich von **C**entral **P**rocessing **U**nit, zentrale Verarbeitungseinheit, herstammende Abkürzung aus der Anfangszeit elektronischer Rechenanlagen. Mittlerweile bezeichnet diese Abkürzung den zentralen Prozessorchip eines Rechners. In gängigen Tischrechnern findet sich hier beispielsweise ein PowerPC™ oder Pentium™-Chip.

zwischen Instruktions- und Datenbus geschaffen wird, durch einen solchen Aufbau wird der Prozessor allerdings stark gebremst. Ein Blockschaltbild des vom Hersteller empfohlenen Grundaufbaus zeigt Abbildung 3.11.

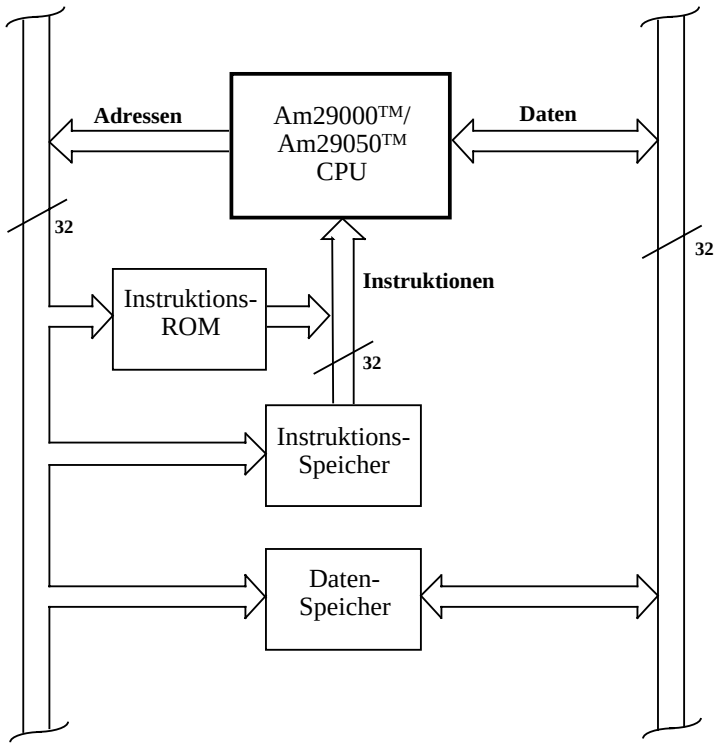



Abbildung 3.11: Blockschaltbild des Speicheraufbaus eines Rechners mit Am29050™-CPU, aus [52]

Dieser Aufbau wird von der CPU dadurch unterstützt, daß vier getrennte Speicherbereiche mit je 32 Bit Adressen (also je 4 GBytes) angesteuert werden können:

- Instruktions-ROM[12]: Dieser Speicher enthält zum Beispiel den primären Boot[13]-Code und ist in der Regel physikalisch als ROM ausgeführt.
- Instruktionsspeicher: Dieser kann als ROM oder RAM[14] ausgeführt sein, wird aber, solange kein zusätzlicher Pfad vom Daten- zum Instruktionsbus vorhanden ist, von der CPU ebenfalls nur gelesen.
- Datenspeicher: Dieser Speicher ist grundsätzlich als RAM ausgeführt und wird von der CPU gelesen und geschrieben.
- Ein-/Ausgabe: alle Ein- und Ausgabekanäle werden aus der Sicht der CPU wie Speicherzellen behandelt. Eine Umsetzung in übliche Datenkommunikationsstandards muß entweder durch eine geeignete Programmierung, externe Hardware oder beides erfolgen.

[12] **R**ead **O**nly **M**emory, nur Lese-Speicher, ein Speichertyp, der von der CPU her nur gelesen, nicht aber geschrieben werden kann.

[13] Mit ‘booten’ wird der Startvorgang eines Rechners bezeichnet

[14] **R**andom **A**ccess **M**emory, Schreib-/Lesespeicher mit wahlfreiem Zugriff. Ein solcher Speicher kann durch Anlegen einer entsprechenden Adresse an jeder Speicherstelle unabhängig ausgelesen oder beschrieben werden.

Es ist aber ebenso möglich, diese Speicherbereiche nicht zu dekodieren und alle Bereiche in einem ‘flachen’ Speichermodell über einen zusammenhängenden Adreßraum anzusprechen.
Um auf den Bussen einen möglichst hohen Durchsatz erreichen zu können, unterstützt die CPU auf beiden Bussen Pipeline- und Burst-Zugriffe. Erlauben die externen Speicher Burst-Zugriffe (d.h., es gibt – entweder im Speicherbaustein integriert oder extern – einen Adreßzähler), so kann in einem verschachtelten Schema kontinuierlich gleichzeitig auf Daten- und Instruktionsspeicher zugegriffen werden (siehe Tabelle 3.1).

### 3.3.2 Der Präprozessor

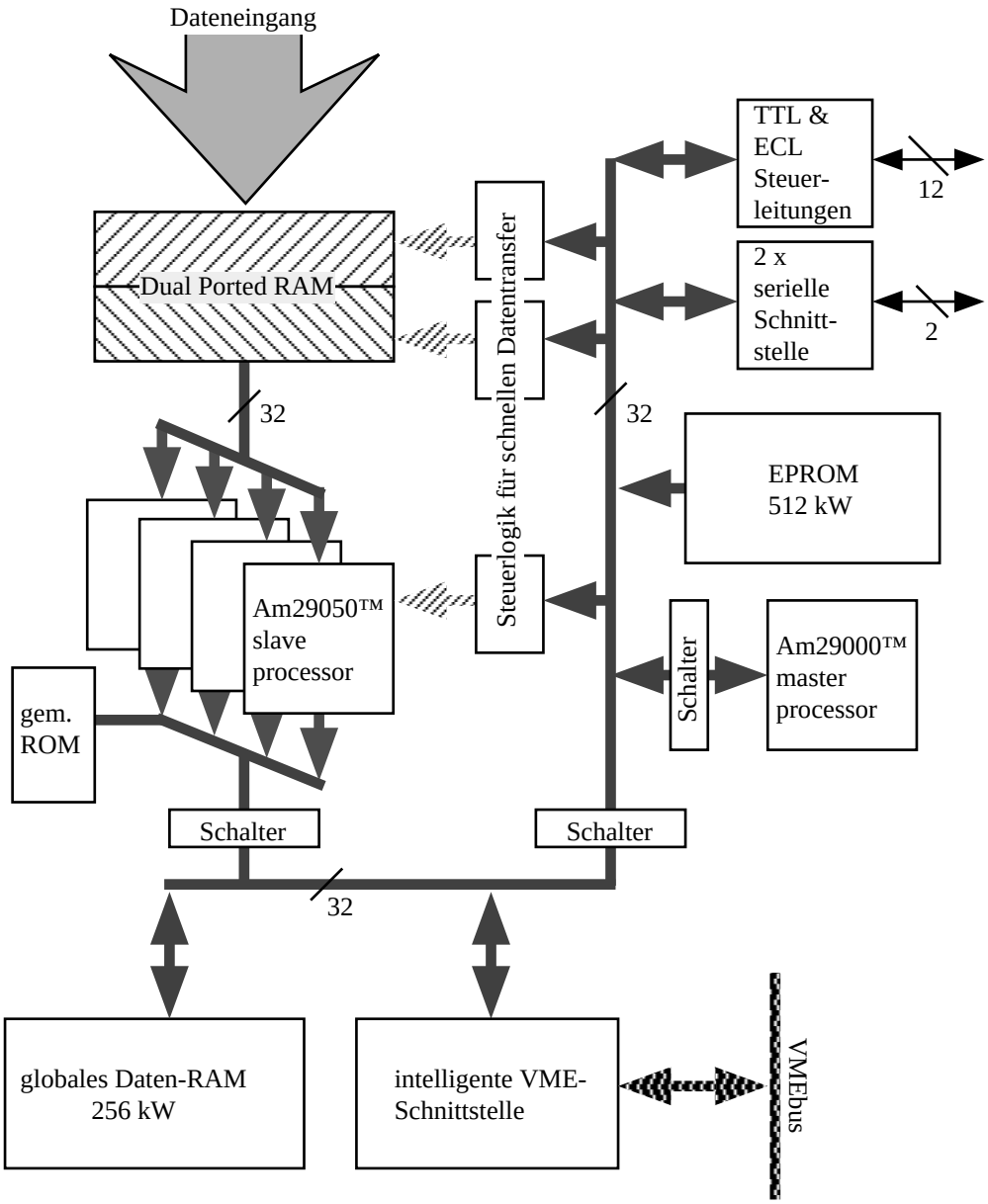



Abbildung 3.12: Blockschaltbild des Präprozessors für Cluster Counting

Die Grundstruktur aus Abbildung 3.10 findet sich im Blockschaltbild des Präprozessors (Abbildung 3.12) wieder.
Die Meßdaten erreichen die Prozessorplatine über einen schnellen Dateneingangsbus, von wo aus sie dann in einem Pufferspeicher ('Dual Ported RAM') abgelegt werden. Von diesem Pufferspeicher aus werden sie dann über ein internes Verteilungsnetzwerk auf die vier Rechenknoten ('slave processor') verteilt. Die Ergebnisse schließlich werden über eine VMEbus-Schnittstelle ausgegeben.
Die Haushaltsführung übernimmt eine weitere getrennte Einheit ('master processor'), der sowohl über den VMEbus als auch über einen Satz dedizierter Leitungen mit der Umgebung kommunizieren kann.
Hinzu kommen nun noch gemeinsam von allen Recheneinheiten genutzte RAM- und ROM-Speicherbereiche.
Alle diese Baugruppen sind über 32 Bit breite, schnelle Datenpfade miteinander verbunden. Diese Datenpfade sind über Schalter in mehrere Segmente unterteilt, so daß Aktionen auf einem Segment nicht notwendigerweise andere Aktionen auf einem anderen Segment behindern.
Der Präprozessor besteht im Einzelnen aus folgenden Elementen:

### Dateneingangsbus

Die Daten erreichen den Präprozessor über einen 32 Bit breiten differentiellen[15] ECL[16]-Bus. Am Eingang der Platine werden diese Signale wieder auf die ansonsten hier verwendeten TTL-Pegel umgesetzt. Die Datenübertragung wird vom Sender aus gesteuert (source-synchronous block transfer[17]), der pro Datenwort einen Steuerimpuls über eine Steuerleitung geben muß.

### Eingangsdatenpuffer

Als Eingangsdatenpuffer wird ein Dual Port RAM[18](DPR) verwendet. Dabei handelt es sich um ein Speichermodul, welches mit jeweils zwei unabhängigen Adreß- und Datenbussen ausgestattet ist, so daß von zwei Seiten her unabhängig voneinander auf die eigentlichen Speicherzellen zugegriffen werden kann. Die einzige Einschränkung ist, daß solche Zugriffe nicht gleichzeitig auf dieselbe Speicherstelle erfolgen.
Die Größe dieses Datenpuffers beträgt wahlweise 8 oder 16 kW[19], so daß sich bei einer Größe der Datenblöcke von 1 kB hier bis zu 64 Datenblöcke dicht gepackt zwischenspeichern lassen. Diese dichte Packung wird in der Praxis aber wegen der damit verbundenen Geschwindigkeitseinbußen

[15]Von differentiellen Signalen spricht man in der Elektronik, wenn ein Signal über zwei Leitungen so übertragen wird, daß auf der einen Leitung das Signal und auf der anderen Leitung das invertierte Signal übertragen wird. Am Empfänger wird dann die Spannungsdifferenz zwischen den beiden Leitungen ausgewertet. Der Vorteil dieser Übertragungsweise liegt zum Einen darin, daß Störungen sich in der Regel auf beiden Leitungen mit demselben Pegel zeigen, und somit bei der Differenzbildung ausgefiltert werden, und zum Anderen darin, daß vom Sender zum Empfänger kein Nettostrom fließt (der Strom, der in der einen Leitung zum Empfänger fließt, fließt in der zweiten Leitung zum Sender zurück), und somit Sender und Empfänger kein gemeinsames Potential aufweisen müssen. Letzteres hilft auch, Masseschleifen zu vermeiden.

[16]**E**mitter **C**oupled **L**ogic, emittergekoppelte Logik. Diese Familien von logischen Bauelementen arbeiten im Unterschied zur gebräuchlicheren TTL (**T**ransistor - **T**ransistor - **L**ogik) mit niedrigerem Spannungshub. Damit werden wesentlich geringere Schaltzeiten, und somit höhere Taktfrequenzen, möglich.

[17]Quellensynchrone Blockübertragung

[18]'Zweitüriges' RAM, je Modul 4 Stück Cypress CY7B144 [55]

[19]kilo Worte, ein Wort ist die kleinste Einheit, die im Präprozessor adressiert werden kann, und besteht in diesem Fall aus 4 Bytes (32 Bit).

nicht verwendet werden[20], womit dann 8 bzw. 16 Datensätze im Puffer zwischengespeichert werden können. Bei voller Bestückung des Präprozessors mit beiden Puffermodulen und vier Rechenknoten ergibt sich somit eine Speichertiefe von 4 Ereignissen pro Rechenknoten.
Die maximale Rate, mit der die Meßwerte zum Präprozessor übertragen werden können, hängt von der Schreibgeschwindigkeit dieser Dual Port RAMs ab. Bei einer Zykluszeit der verwendeten Typen von 15 ns läßt sich theoretisch eine Transferrate von 66 MHz erreichen. Bei der Busbreite von 4 Bytes ergäbe dies eine maximale Datenrate von 260 MB/s.
Diese Datenrate liegt allerdings weit über dem, was die Rechenknoten einer Präprozessorplatine verarbeiten können. Der Entwurf wurde trotzdem so gewählt, um mehrere Präprozessorplatinen gemeinsam über einen solchen ECL-Bus mit den Analog-Digital-Wandlern verbinden zu können, ohne daß es wegen der Übertragungsbandbreite dieses Busses zu Engpässen kommt.
Damit diese Datentransferrate erreicht werden kann, müssen die Adressen für das DPR in einem schnellen Logikbaustein (PAL[21]) generiert werden. Ähnliche Bausteine erzeugen auch die Adressen und Steuersignale für den Transfer der Daten zu den Rechenknoten (‘slave processor’).

**Datenverteilung**

Die im Eingangspuffer zwischengespeicherten Daten müssen nun zu einem der maximal vier Rechenknoten weitergeleitet werden. Der Transfer selbst kann, wie überall auf der Platine, über einen 32 Bit breiten Bus mit einer Rate von 1 Wort/Takt erfolgen. Für die Abwicklung dieser Übertragung gibt es aber zwei grundsätzlich verschiedene Möglichkeiten:

- a) der Rechenknoten holt sich den ihm zugeteilten Datensatz aus dem Pufferspeicher ab, oder
- b) der Datensatz wird dem Rechenknoten vom Pufferspeicher aus zugestellt.

In beiden Fällen muß der Haushaltsprozessor eingreifen. Im ersten Fall, um dem Rechenknoten mitzuteilen, welchen Datensatz er wann woher holen soll, im zweiten Fall, um den RAM – oder einer entsprechenden Transferelektronik – mitzuteilen, welcher Datensatz wann wohin geschickt werden soll. Dazu kommt noch die Buchführung, welcher Datensatz abgearbeitet wurde, welche Rechenknoten beschäftigt sind, und wo freier Speicherplatz vorhanden ist.
Der Ablauf des Datentransfers folgt in den beiden Fällen den folgenden Schemata:

**a)** Der Rechenknoten holt sich die Daten selbst ab. In diesem Fall muß für jeden Datensatz folgender Zyklus durchlaufen werden:

1. Adresse vom Prozessor des Rechenknotens an den Pufferspeicher ausgeben
2. Datum aus dem Puffer in ein Prozessorregister schreiben
3. Adresse an den lokalen Datenspeicher des Rechenknotens ausgeben und Datum in diesen Speicherplatz schreiben.

[20] Bei der dichten Packung werden jeweils zwei 2 Bytes große Meßdaten-Worte in einem der Worte des Präprozessors abgespeichert. Da der Prozessor auf die Bytes innerhalb eines Wortes nicht direkt zugreifen kann, müssen diese durch ein Programm aus dem Präprozessor-Wort extrahiert werden, was zusätzlich Zeit kostet.

[21] **P**rogrammable **A**rray **L**ogic, programmierbarer Logikbaustein. Die Begriffe ‘PAL’, ‘PLA’ oder ‘GAL’ bezeichnen im wesentlichen die selben programmierbare Logikbauelemente unterschiedlicher Hersteller

Dazu kommen zusätzliche Taktzyklen zur Berechnung der jeweiligen Speicheradressen sowie für die Programmschleife, minimal also etwa acht Prozessortakte pro Datum. Eine Organisation dieses Vorgangs in einer Pipeline ist hier nicht möglich, da der Adressbus ständig zwischen dem (externen) Pufferspeicher und dem (internen) lokalen Datenspeicher umgeschaltet werden muß.

**b)** Die Daten werden dem Rechenknoten zugestellt. In diesem Fall kann eine Besonderheit der Am29050™-CPU ausgenutzt werden. Diese CPU ist für den Betrieb in einem Multiprozessorsystem vorbereitet und kann, auf ein äußeres Signal hin, dazu veranlaßt werden, sämtliche Ausgänge in eine hochohmigen Zustand zu schalten, während alle Eingangssignale (mit Ausnahme dieses Steuersignales) ignoriert werden. In diesem Zustand ist die CPU für den Rest des Systems effektiv nicht vorhanden. Damit kann nun von einem äußeren System aus direkt auf die lokalen Datenspeicher zugegriffen werden, um dort die Meßdaten abzulegen. Dieser Datentransfer kann folgendermaßen ablaufen:

1. ...
2. Adresse (i+1) an das DPR ausgeben, Adresse (i) an das RAM des Rechenknotens anlegen, Datensatz (i) in den Rechenknoten schreiben
3. Adresse (i+2) an das DPR geben, Adresse (i+1) an das RAM des Rechenknotens anlegen, Datensatz (i+1) – der mittlerweile am Rechenknoten angelangt ist – schreiben
4. ...

In einem solchen Zyklus wird mit jedem Taktimpuls ein Datenwort übertragen. Ermöglicht wird ein solcher Ablauf durch die Verwendung schneller synchroner statischer RAM-Bausteine. Im Unterschied zu den in Bürorechnern gebräuchlicheren 'dynamischen' Speichern ermöglichen diese Bausteine eine einfachere Adressierung und schnellere Zugriffe auf die gespeicherten Daten, allerdings um den Preis höherer Kosten.

Die für den schnellen Transfer benötigte externe Steuerlogik besteht aus zwei, mit jeweils einem PAL realisierten, schnellen ladbaren Zählern, die die Adressen für das DPR und die RAMs der Rechenknoten erzeugen, sowie einer, ebenfalls wieder mit PALs aufgebauten, Logik für die Erzeugung und Verteilung der Takt- und Steuerimpulse.

Die Übertragung der Daten erfolgt über ein Bussegment, das im normalen Betrieb von den anderen Bussen der Präprozessorplatine isoliert ist. Der Sinn dieser Maßnahme besteht darin, den Bus jederzeit für diesen Datentransfer zur Verfügung zu haben, ohne daß erst Verhandlungen mit anderen Busteilnehmern über die Wegerechte geführt werden müssen ('Arbitrierung').

Abbildung 3.13 zeigt eine Ansicht der Oberseite der Hauptplatine des Präprozessors. Der Anschluß für den parallelen ECL-Eingangsbus befindet sich in der rechten oberen Ecke. Direkt darunter liegt der Steckplatz für das oder die (maximal 2) DPR-Modul(e). Darunter wiederum folgen die vier Steckplätze für die Slave-Module. Unterhalb dieser Steckplätze wiederum liegt der Großteil der Logik für die Steuerung der Datenverteilung zwischen den einzelnen Bussegmenten der Hauptplatine sowie, am unteren Rand der Platine, die Steuerbausteine für die VMEbus-Schnittstelle. Auf der linken Seite der Platine ist der Master-Prozessor mit den zugehörigen Speicherbereichen zu sehen. Die physikalische Trennung zwischen Instruktions- und Datenspeicher ist deutlich zu erkennen. Der Rest der linken Seite wird zum größten Teil von Bausteinen der Hilfsfunktionen

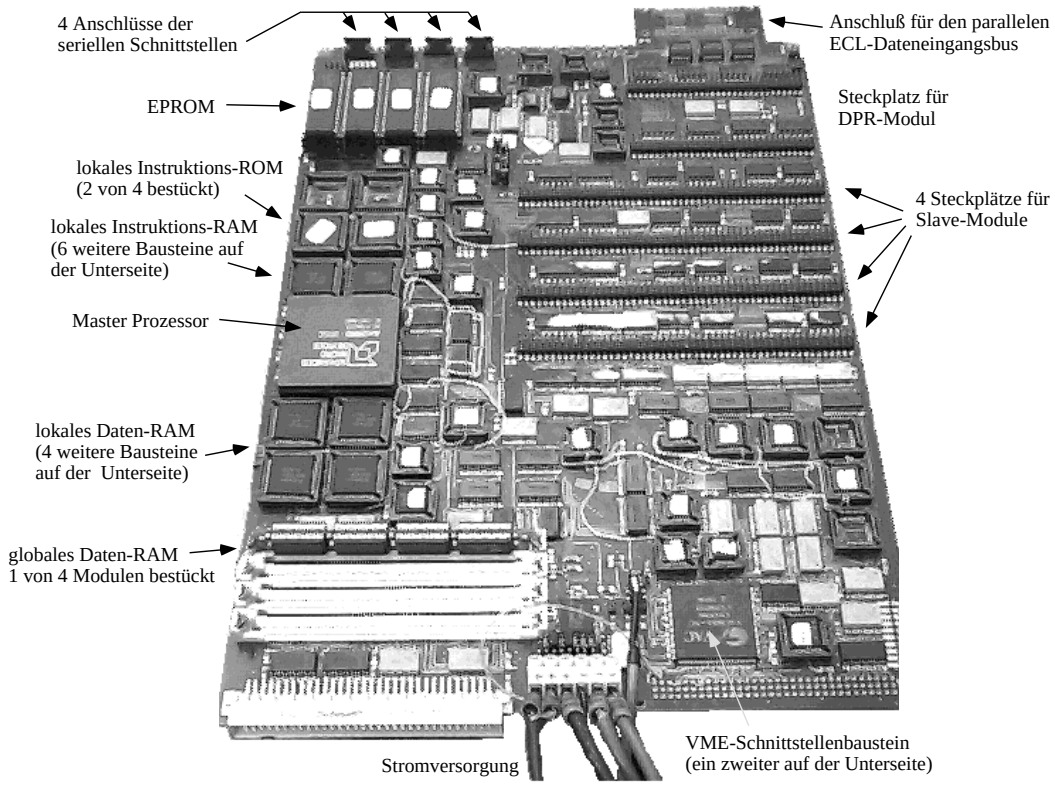



Abbildung 3.13: Foto der Präprozessor-Hauptplatine

eingenommen. Hier befinden sich unter anderem die EPROMs für die Speicherung unveränderlicher Daten (oder Programme) sowie die gemeinsam genutzten RAMs zur Zwischenspeicherung der Ausgangsdaten.

**Die Rechenknoten (‘Slaves’)**

Das Blockschaltbild der vier Rechenknoten (oder einfach ‘Slaves[22]’) (Abbildung 3.14) hält sich relativ eng an die allgemeine Schaltung aus Abbildung 3.11.
Ein Slave besteht im wesentlichen aus der CPU AMD Am29050™ mit integriertem mathematischen Coprozessor, einem Block Datenspeicher-RAM und jeweils einem Block Instruktionsspeicher-RAM und -ROM. Dazu kommen (in der Abbildung nicht dargestellte) PALs als Steuerlogik und zur Adressauswertung sowie ein bidirektionaler Busschalter, um Instruktions- und Datenbus miteinander zu verbinden. Auf der Mutterplatine sitzen weitere dieser Busschalter (‘Transceiver’), um den Slave wahlweise mit einem der beiden Bussysteme (Dateneingang oder Datenausgang) zu verbinden.
Die Daten- und Instruktionsspeicher-RAMs besitzen jeweils eine Kapazität von 32 kWorten, also 128 kBytes, und sind mit schnellen synchronen RAM-Bausteinen des Herstellers Cypress [55] aufgebaut[23]. Diese RAMs ermöglichen einen synchronen Burstzugriff auf vier konsekutive Adressen.

[22] ‘Sklave’, wegen der Relation zum haushaltsführenden Prozessor, der demzufolge auch ‘Master’ (‘Meister’) genannt wird.

[23] je 4 Stück CY7B174-14 JC

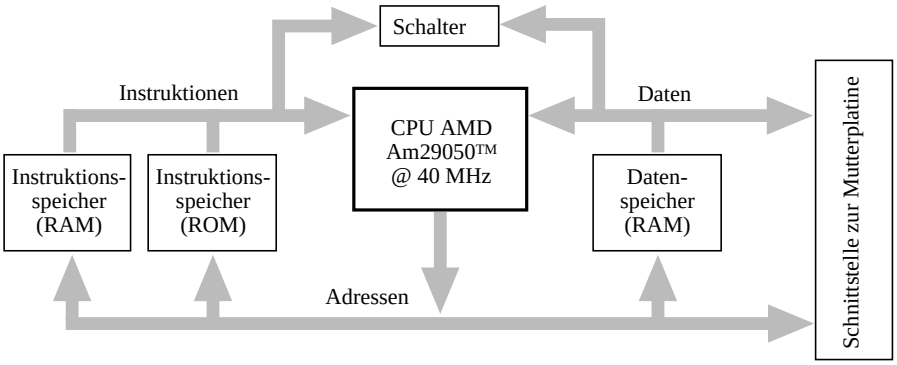



Abbildung 3.14: Blockschaltbild eines Rechenknotens (’Slaves’)

Durch einen Fehler beim Platinenentwurf ist es nicht möglich, diesen Burstzugriff in der vorliegenden Version der Slaveplatine einzusetzen.
Im Unterschied dazu besitzt das Instruktionsspeicher-ROM[24] die Hälfte dieser Kapazität, also 64 kBytes oder 16 kWorte, unterstützt aber synchrone Burstzugriffe auf bis zu 256 konsekutive Adressen, wodurch es etwa 20 % schneller arbeiten kann als die RAMs. In der Praxis ist aufgrund des oben erwähnten Designfehlers das Instruktions-ROM die einzige Baugruppe der Platine, auf die die CPU im Burst-Modus zugreifen kann, so daß die erzielbare Übertragungsrate aus dem ROM etwa 2…3 mal so hoch ist wie aus dem RAM. Dies hat zur Konsequenz, daß man versuchen wird, im Einsatz des Präprozessors alle häufig benötigten sowie möglichst alle längeren Programmroutinen entgegen der üblichen Praxis statt im RAM in diesem ROM unterzubringen.
Der Schalter zur Verbindung von Instruktions- und Datenbus schließlich ermöglicht das Laden von Instruktionen von der Schnittstelle zur Mutterplatine aus in das Instruktions-RAM. Dazu ist der Prozessor des Slave allerdings nicht selbst in der Lage, sondern diese Aufgabe muß von der Master-CPU übernommen werden, die, wie auch sonst auf der gesamten Platine, bei Bedarf vollen Zugriff auf jede Speicherzelle hat.
Die Schnittstelle zur Mutterplatine beschränkt sich auf dem Slave im wesentlichen auf den Steckverbinder, die gesamte Logik dieser Schnittstelle ist an entsprechender Stelle der Mutterplatine untergebracht.
Ein Foto einer Slaveplatine zeigt Abbildung 3.15.
Auf der Oberseite der Platine, im linken Teil der Abbildung, sind die wichtigsten Elemente des Slaves gut zu erkennen. Etwa in der Mitte der Platine sitzt die eigentliche CPU Am29050™. Darunter befinden sich die Instruktionsspeicher, getrennt als RAM und ROM. Oberhalb der CPU, sowohl logisch als auch räumlich von den Instruktionsspeichern getrennt, sind die Bausteine des Datenspeichers untergebracht. Am oberen Ende der Platine ist der Treiberbaustein zur Erzeugung und Verteilung der verschiedenen Taktsignale, zusammen mit den Pfostensteckern zur Einstellung der Taktverzögerung, zu sehen. Am rechten Rand schließlich befinden sich drei der sechs PALs, die zusammen die Adreßdekodier- und Steuerlogik der Slaveplatine bilden.
Auf der Platinenunterseite befinden sich zusammen mit den anderen drei PALs die restlichen Speicherbausteine. Die vier integrierten Schaltungen in der Mitte der Platine, also genau gegenüber der CPU, bilden den Schalter, mit dem Instruktions- und Datenbus miteinander verbunden werden können.

[24] 2 Stück CY7C270-20 HC

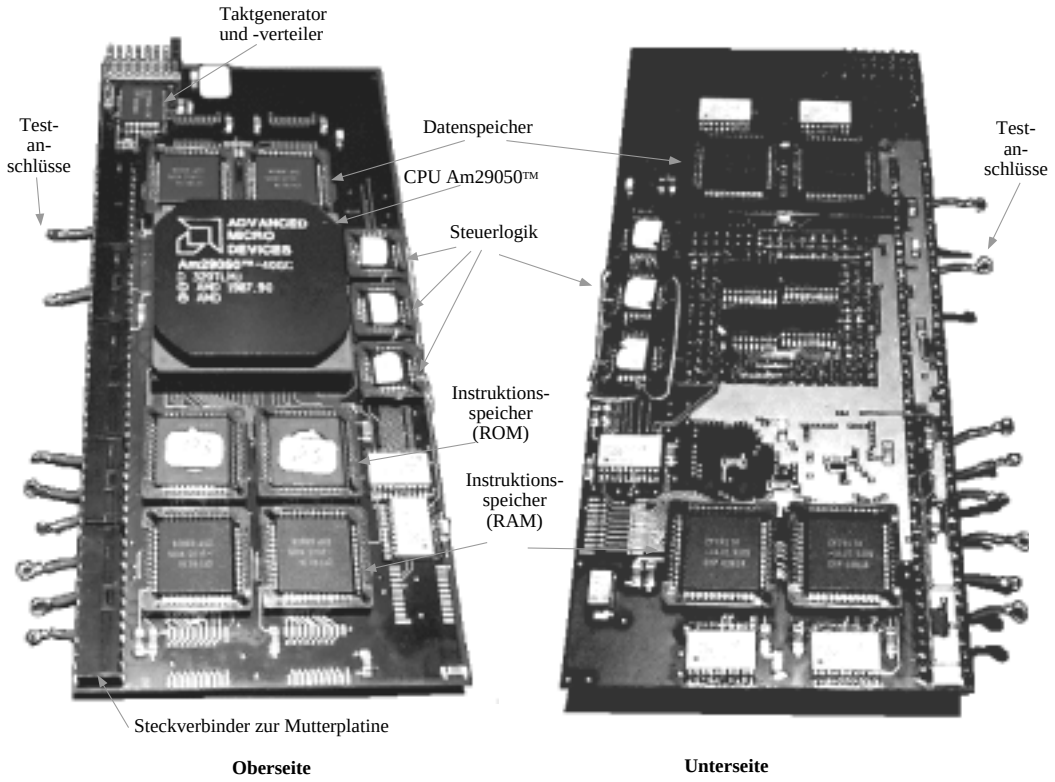



Abbildung 3.15: Foto einer Slaveplatine

Im Blockschaltbild des Präprozessors (Abbildung 3.12) ist noch ein weiterer gemeinsamer ROM-Bereich am Ausgangsbus der Slaves vorgesehen. Dieser Busbereich kann durch eine Gruppe von Datenschaltern vom Rest der Platine abgekoppelt werden, so daß die Slaves ‘unter sich’ sind. Aus dem gemeinsamen ROM können dann selten benötigte Programmroutinen direkt abgearbeitet werden.
Alle diese angesprochenen Speicher, zusammen mit Ein-/Ausgabebereichen und Kommunikationsregistern, liegen in einem ‘flachen’ Speichermodell, das heißt einem einzigen (von den vier möglichen) Speicherbereichen der CPU und werden – aus Sicht der Slave-CPU – direkt adressiert. Eine Speicherkarte befindet sich in Anhang B.

### Datenausgabe und VME-Schnittstelle

Nach der Verarbeitung der Daten in den Rechenknoten müssen die Ergebnisse an ‘den Rest der Welt’ übermittelt werden. Ob es sich dabei direkt um ein Aufzeichnungssystem handelt, oder ob die Daten in anderen Stufen weiterverarbeitet werden sollen, ist in diesem Zusammenhang von untergeordneter Bedeutung.
Um die modulare Erweiterbarkeit des Systems zu gewährleisten, wurde als Basis für die Präprozessorkarten der VMEbus-Standard [43, 44] gewählt. Um für den Fall, daß mehrere Präprozessorkarten gleichzeitig auf den Bus zugreifen wollen, einen Datenstau auf diesem Bus zu verhindern, ist auf den Präprozessorplatinen ein großzügig dimensionierter allgemeiner Datenspeicher vorgesehen, in dem die auszugebenden Daten zwischengespeichert werden können.
Dieser Zwischenspeicher führt auch zu einer weiteren Entlastung der Rechenknoten, da die Ergeb-

nisse vom Slave schnell in diesen Zwischenspeicher geschrieben werden können, worauf dann der Slave wieder frei ist, den nächsten Datensatz zu übernehmen. Die Transferrate beträgt in diesem Fall allerdings nur etwa 12 MB/s (30 MHz * 10 Takte * 4 Bytes/Takt), da an dieser Stelle die CPU des Slave selbst den Transfer steuern muß.
Auf dem VMEbus ist innerhalb eines Crates[25] eine theoretische maximale Datentransferrate im Blocktransfer von 40 MB/s möglich, im praktischen Betrieb sind bis zu 34 MB/s erreichbar[26] [56], bei der Verbindung mehrerer Crates über einen VMVbus[27] ist die Transferrate für Datenpakete, die über diesen Bus laufen müssen, allerdings auf etwa 2 MB/s begrenzt.
In beiden Fällen kommt allerdings noch zusätzlicher Aufwand für das Aufsetzen des Blocktransfers hinzu, was sich bei den kleinen Datenpaketen, wie sie am Ausgang des Präprozessors anliegen, besonders nachteilig bemerkbar macht.
Die Ergebnisse werden also zunächst von den Rechenknoten in den Zwischenspeicher geschrieben, und später, entweder auf eine externe Anforderung hin oder vom Master-Prozessor veranlaßt, von einer intelligenten VME-Schnittstelle aus blockweise an den ‘Rest der Welt’ übermittelt. Der Zwischenspeicher selbst besteht aus vier schnellen RAM-Modulen[28] mit einer Speicherkapazität von je 256 kBytes, insgesamt also 1 MByte beziehungsweise 256 kWorte. Die VME-Schnittstelle ist mit einem speziellen Chipsatz des Herstellers Cypress aufgebaut[29] [57]. Dieser Chipsatz übernimmt die vollständige VMEbus-Kommunikation einschließlich der Übertragung vollständiger Speicherblöcke (Blocktransfer) , und besitzt die Fähigkeit, bei Bedarf direkt auf jede Speicherzelle des Präprozessors selbständig zuzugreifen (DMA[30]).
Der Chipsatz stellt weiterhin einen parallelen 8-Bit-Bus für langsame Peripheriebauelemente sowie zwei serielle Schnittstellen zur Verfügung. Der parallele Bus wird in der vorliegenden Version der Präprozessorplatine nicht genutzt, denkbar wäre hier zum Beispiel der Anschluß einer Flüssigkristallanzeige, um Informationen zum Prozessorstatus direkt anzuzeigen. Die beiden seriellen Schnittstellen sind über entsprechende Treiber und Anschlüsse an der Frontplatte herausgeführt.

### Hilfsfunktionen

Über einen weiteren Schalter ist ein Bussegment angeschlossen, an welchem gemeinsam genutzte Hilfsschaltungen angesiedelt sind.

**EPROM** Das gemeinsame EPROM enthält Programme, die nach dem Start des Präprozessors vom Master in die Programmspeicher der Slaves geladen werden können. Die Programme für den eigentlichen Startvorgang befinden sich zwar in den lokalen EPROMs der einzelnen CPUs, aber beispielsweise das Programm zur Clustersuche kann auf diese Weise aus dem gemeinsamen EPROM geladen werden.

Eine andere Möglichkeit wäre, Clustersuchprogramme mit verschiedenen Parametersätzen abrufbereit in diesem EPROM zu speichern, um sie dann bei Bedarf in die Rechenknoten zu transferieren. Ohne dieses EPROM müßten diese Programme dann jeweils aus dem externen Steuerrechner zum jeweiligen Präprozessor übertragen werden.

[25] ‘Kiste’, Bezeichnung für einen Einschubrahmen mit bis zu 21 Steckplätzen, einer Rückplatine (‘Backplane’), über die der VMEbus geführt wird, und einer Stromversorgung.
[26] Von uns wurden bisher in verschiedenen Aufbauten Transferraten von bis zu 30 MB/s erreicht.
[27] Ein Bussystem der Firma C.E.S. zur Verbindung mehrerer VMEbus-Crates
[28] Cypress CYM1841PM-20C, [55]
[29] Cypress VIC068A/VAC068A
[30] **D**irect **M**emory **A**ccess, direkter Speicherzugriff

Aufgebaut ist dieser Speicherbereich mit vier ‘normalen’ EPROM-Bausteinen[31], die zusammen eine Speicherkapazität von bis zu 2 Megabytes (512 kW) aufweisen.

**Serielle Schnittstellen** Insgesamt sind auf der Präprozessorplatine vier serielle (RS-232) Schnittstellen vorhanden. Zwei davon sind über den Chipsatz der VME-Schnittstelle zugänglich, zwei weitere befinden sich im Bussegment der Hilfsfunktionen. Die beiden letzteren Anschlüsse weisen Leitungen für einen Hardware-Handshake[32] auf.

Während der Entwicklungsphase stellen diese Schnittstellen den hauptsächlichen Kommunikationspfad zwischen der Präprozessorplatine und dem Entwicklungsrechner dar.

**Steuerleitungen** Um in einem Experiment die Kommunikation sowohl mit anderen Präprozessorplatinen als auch mit Beispielsweise den Datenquellen zu ermöglichen, sind an der Frontseite der Platine eine Anzahl von Steuerleitungen herausgeführt. Dazu gehören jeweils 8 frei programmierbare Ein- und Ausgänge mit normalen TTL-Pegeln sowie jeweils vier Ein- und Ausgänge nach dem in der Hochenergiephysik weit verbreiteten NIM-Standard. Von diesen vier Anschlüssen ist jeweils einer fest für die Steuerung des schnellen Dateneingangsbusses verdrahtet, die anderen drei können vom Master ausgelesen (bzw. geschrieben) werden und stehen für allgemeine Aufgaben zur Verfügung.

**Haushaltsführung** Der unmittelbare Bereich der CPU für die Haushaltsführung (‘Master’) ist im wesentlichen genauso aufgebaut wie einer der Rechenknoten (Abbildung 3.16).

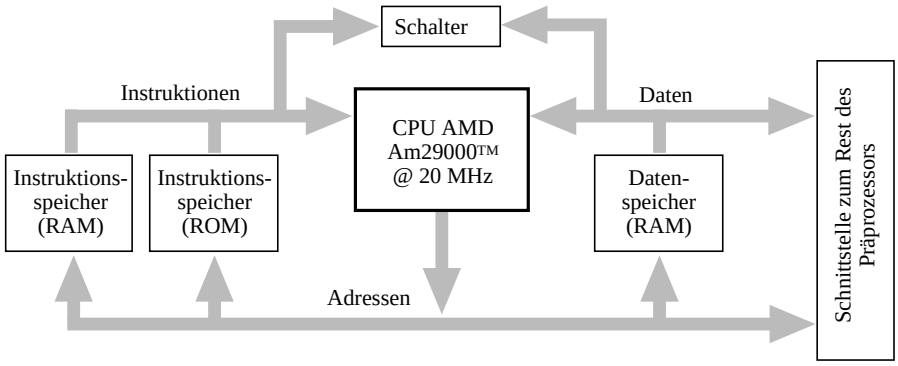



Abbildung 3.16: Blockschaltbild der unmittelbaren Umgebung des Master-Prozessors

Die Unterschiede liegen in der Größe der lokalen Speicherbereiche sowie in der Logik zur Adreßdekodierung. In den lokalen Speicherbereichen steht dem Master im Vergleich zum Slave jeweils doppelt soviel Platz zur Verfügung, also 128 kBytes (32 kWorte) Instruktionsspeicher-ROM und jeweils 256 kBytes (64 kWorte) Instruktionsspeicher-RAM und Datenspeicher-RAM.

Der Master kann, im Unterschied zu den Slaves, auf den gesamten, 32 Bit breiten Adreßraum der Präprozessorplatine zugreifen. Hierbei wird zwischen zwei Adreßbereichen un-

[31] 32-Pin JEDEC-kompatible EPROMS mit 1...4 Megabit Speicherkapazität

[32] ‘Händeschütteln’, bezeichnet den Vorgang der Datenflußkontrolle zwischen zwei Teilnehmern einer Datenübertragung. Hierzu gibt es drei Möglichkeiten: ohne – dafür müssen beide Teilnehmer exakt synchron laufen, Software – ein Teil des Datenstromes enthält die Steuerinformationen – und Hardware – die Datenflußkontrolle erfolgt über dedizierte Leitungen.

terschieden. Über den unteren Adreßbereich werden alle diejenigen Speicher und Register angesprochen, die sich in der unmittelbaren Umgebung des Masters befinden (Abbildung 3.16). Dieser Bereich steht ausschließlich dem Master zur Verfügung. Zugriffe auf Bauelemente im oberen Adreßbereich können dagegen sowohl vom Master aus als auch von einem externen Rechner über die VMEbus-Schnittstelle oder, in manchen Fällen, auch von einem der Slaves aus stattfinden. Deswegen muß vor einem Zugriff auf diesen Bereich eine Arbitrierung zur Klärung der Zugriffsrechte stattfinden. Eine Übersicht über diese Adreßräume befindet sich in Anhang B.

Die Steuerfunktionen des Master-Prozessors erfolgen größtenteils über spezielle Kontroll- und Statusregister. Die Hauptaufgabe des Masters im normalen Betrieb besteht darin, diese Statusregister sowie die Register des VME-Chipsatzes und der Steuerschaltung der seriellen Schnittstellen zyklisch auszulesen und daraufhin die notwendigen Aktionen zu unternehmen. Eine solche Vorgehensweise wird auch als ‘polling’[33] bezeichnet. Eine andere Möglichkeit, Reaktionen des Masters auf bestimmte Zustände oder Ereignisse hin auszulösen, wäre gewesen, beim Auftreten dieser Zustände einen Interrupt[34] auszulösen, worauf die CPU sobald als möglich in der augenblicklichen Tätigkeit innehält und die diesem Interrupt zugeordneten Programmteile abarbeitet, bevor sie ihre vorhergehende Tätigkeit wieder aufnimmt.

Ein solches Interrupt-System hat den Vorteil, daß die Antwortzeiten kürzer sein können als beim Polling, da in letzterem Fall ungünstigstenfalls erst alle anderen Statusregister abgearbeitet werden müssen, bevor das eigentlich nach Aufmerksamkeit verlangende Register an die Reihe kommt. Die Nachteile eines Interrupt-Systems kommen jedoch vor allem dann zum Tragen, wenn mehrere Interrupt-Quellen gleichzeitig bearbeitet werden wollen, oder auch nur, wenn während der Bearbeitung eines Interrupts ein zweiter Interrupt auftritt. In einem solchen Fall wird dann der nötige Overhead, das heißt die Zeit, die zur Umschaltung zwischen den verschiedenen Interrupt-Behandlungsroutinen benötigt wird, relativ groß. Speziell bei kurzen Interrupt-Behandlungsroutinen kann dann dieser Overhead mehr Rechenzeit verschlingen als die eigentliche Interrupt-Behandlung.

Ereignisse, bei denen die im Prinzip kürzeren möglichen Antwortzeiten eines Interrupt-Systemes von Bedeutung wären, sind beim Präprozessor nicht vorgesehen, so daß dieses einfacher zu implementierende Polling-System verwendet werden kann.

Abbildung 3.17 schließlich zeigt die Hauptplatine zusammen mit einem der Slaves in seinem Steckplatz. Parallel zu diesem ersten Slave können dahinter noch drei weitere Slave-Module eingesteckt werden. Der gesamte Aufbau (Master mit vier Slaves) könnte dann in ein VMEbus-Crate eingeschoben werden. Dies wird aber bei diesem Prototypen dadurch verhindert, daß zusätzliche Leitungen um die Kante der Platine herumgeführt werden mußten. Ein solcher Einschub würde in der gegenwärtigen Bauform etwa fünf Steckplätze innerhalb eines solchen VMEbus-Crates belegen, so daß bis zu vier dieser Präprozessoren mit insgesamt 16 Slaves in einem Crate Platz finden.

[33] ‘Abfragen’
[34] ‘Unterbrechung’

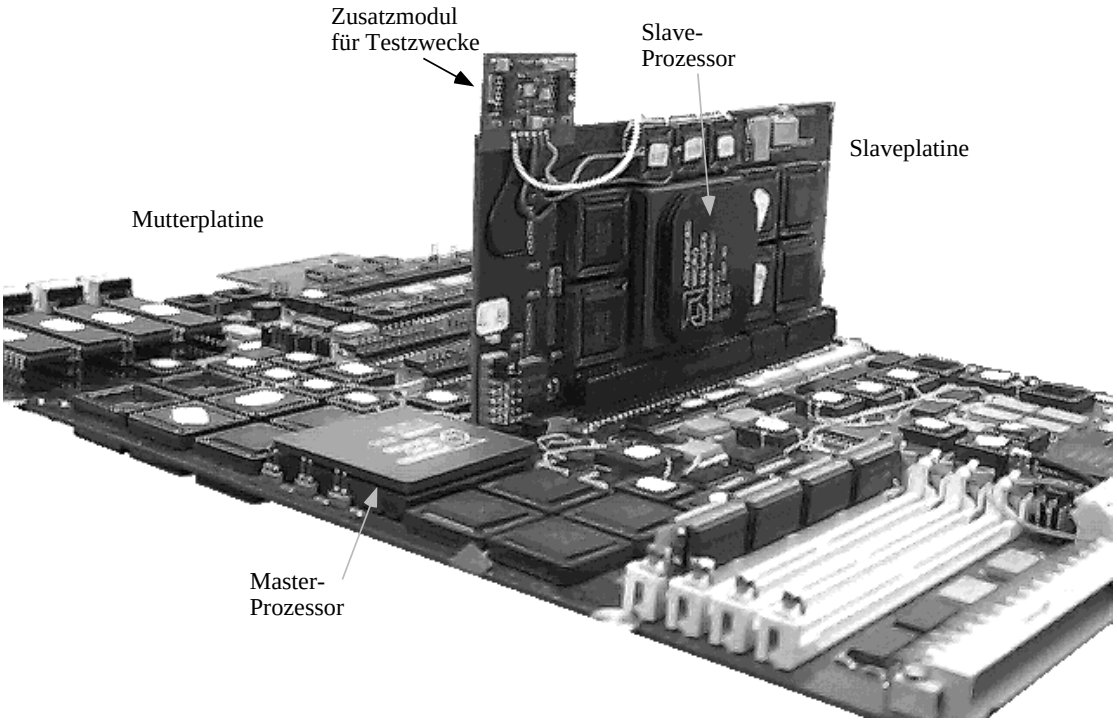



Abbildung 3.17: Foto des Präprozessors mit einem Slave

### 3.3.3 Maximierung des Datendurchsatzes

Die Prozessoren der Rechenknoten arbeiten mit einer Taktfrequenz von 30 MHz[35]. In einer Zeit, in der Personalcomputer mit Taktfrequenzen von über 200 MHz arbeiten, ist dies nicht allzuviel, aber bei Beginn des Projekts waren dies mit die schnellsten erhältlichen Prozessoren.
Der Datendurchsatz, den ein Prozessor erreichen kann, wird aber nicht ausschließlich durch dessen Taktrate bestimmt. Die meisten der modernen Prozessoren sind als RISC-Prozessoren mit internen Pipelines aufgebaut, so daß sie pro Taktzyklus eine Instruktion annehmen können, so lange keine Abhängigkeiten zwischen aufeinanderfolgenden Instruktionen bestehen. Unterschiede gibt es allerdings in der Tiefe der Pipeline, was sich in der Latenzzeit[36] niederschlägt, und sich somit speziell bei der Abarbeitung kurzer Programmstrukturen auswirkt. Hat die Pipeline nur eine Tiefe von vier Prozessortakten (z.B. Am29000™), so werden auch nur vier Takte benötigt, um die Pipeline zum Beispiel nach einem Programmsprung neu zu füllen, bei einer Pipelinetiefe von zehn Takten (z.B. DEC Alpha 21064 [58]) sind eben auch diese zehn Takte zum Füllen der Pipeline nötig. Eine kurze Schleife mit zehn Instruktionen braucht so im einen Fall 14 Takte, im anderen Fall 20 Takte

[35] Der Slave selbst ist für eine Taktfrequenz von 40 MHz ausgelegt, der Master für eine Taktfrequenz von 20 MHz. Beide arbeiten getrennt auch mit dem jeweiligen Designtakt, beim Datenaustausch zwischen Master und Slave gab es jedoch Probleme mit der Synchronisation, so daß die Taktrate auf 30/15 MHz verringert werden mußte.

[36] Zeit, die beim Füllen der Pipeline vergeht, bis das erste Ergebnis am Ende der Pipeline herauskommt.

– falls keine lokalen Instruktionsspeicher (‘Cache’) zusammen mit einer Branch Prediction Unit[37] dies abfangen.
Für den Datendurchsatz durch ein gegebenes System kommt es aber nicht nur auf den Prozessor an. Genauso wichtig ist es, Daten und Instruktionen ausreichend schnell zum Prozessor hinzubringen und die Ergebnisse von dort abzuholen.
Ein kurzes Programm, in dem nur wenige Daten verarbeitet werden, findet in der Regel zusammen mit den Daten in einer prozessorinternen ersten Cache-Ebene (‘first-level-cache’) [38] Platz. In einem solchen Fall wird die Ausführungsgeschwindigkeit dieses Programmes von der Prozessorgeschwindigkeit bestimmt.
Passen Programm oder Daten nicht mehr in den Speicher der ersten Cache-Ebene, muß auf die nächste Speicherebene, die zweite (externe) Cache-Ebene (‘second-level-cache’) ausgewichen werden. Dies wiederholt sich, wenn auch diese Ebene nicht mehr ausreicht, wieder mit dem Hauptspeicher sowie, im Extremfall, der Festplatte oder anderen externen Massenspeichern. Bei modernen Hochleistungsprozessoren tritt hier aber schnell der Fall ein, daß die Rechenleistung nicht mehr vom Datendurchsatz des Prozessors, sondern von der maximal möglichen Zugriffsgeschwindigkeit auf die in den unterschiedlichen Ebenen dieser Speicherhierarchie abgelegten Daten begrenzt wird.
Ideal wäre es demnach, wenn alle Instruktionen und Daten direkt in den Speichern der ersten Cache-Ebene abgelegt werden könnten. Daß dies in der Praxis nicht gemacht wird, hat im Wesentlichen zwei Gründe. Zunächst einmal ist da der Preis. Je schneller ein Speicherelement sein soll, desto höher liegt auch der Preis pro Speichervolumen. Der zweite Grund liegt im Stromverbrauch der Speicherelemente. Schnelle, statische Speicher haben einen relativ hohen Stromverbrauch pro Speichervolumen, und damit auch eine entsprechend hohe Verlustleistung, so daß diese Elemente in einem Rechnersystem nur in begrenztem Umfang eingesetzt werden können.
Die Aufteilung des Speichers in mehrere hierarchische Ebenen hat noch einen weiteren Nachteil. Immer dann, wenn ein Eintrag in einer der Ebenen geändert wird, muß sichergestellt werden, daß diese Änderung (mit allen Auswirkungen) in den anderen Ebenen abgespeichert wurde, bevor aus diesen wieder Daten in die erste Ebene übertragen werden (‘Cache-Kohärenz’).
Die schnellsten derzeit (1996) erhältlichen Cache-Module weisen Zugriffszeiten von 7 ns auf [59]. Unter Berücksichtigung unvermeidbarer Verzögerungen[39] erlauben diese Module den Betrieb von Prozessoren bis zu einer Taktfrequenz von 75 MHz ohne Wartezyklen. Schnellere Prozessoren müssen entweder mit dem Inhalt des internen Cache arbeiten, beim Zugriff auf die Speicher Wartezyklen (‘wait states’) einlegen oder mehrere Speicherbänke im abwechselnden Zugriff verwenden (‘interleaving’). Zum Zeitpunkt des Entwurfs des Präprozessors hatten die schnellsten verfügbaren Speicherbausteine eine Zykluszeit von 15 ns, womit Taktfrequenzen von bis zu 50 MHz möglich waren.
Greift ein Prozessor nun ohne Wartezyklen in einem vier Datenworte umfassenden Burst auf einen solchen Speicher zu, spricht man auch von einem 2-1-1-1 Zyklus : 2 Takte für das erste Datenwort und je ein Takt für die drei nächsten Datenworte. Von den beiden Takten für das erste Datenwort

[37] Ein ‘Vorhersagesystem’, das aufgrund vergangener Ergebnisse einer Sprungentscheidung versucht, die nächste Sprungentscheidung vorherzusagen und dann die relevanten Instruktionen im Voraus an den Prozessor liefert.

[38] Ein lokaler, meist kleiner Zwischenspeicher innerhalb der CPU, der Daten und Instruktionen im Idealfall mit dem CPU-Takt an die ausführenden Einheiten liefern kann.

[39] Zu diesen zählen vor allem die setup- und hold-Zeiten an den Eingängen des Prozessors. Dies sind die Zeiten, die ein Signal vor (‘setup’) beziehungsweise nach (‘hold’) der Flanke eines Taktsignals am Eingang einer Schaltung stabil anliegen muß, um eine ordnungsgemäße Funktion sicherzustellen.

ist der erste zur Übertragung der Adresse notwendig, erst im zweiten kann dann dieses Datenwort vom Prozessor übernommen werden. Die folgenden Adressen innerhalb eines Bursts werden dann im RAM selbst erzeugt. Ein Zugriff auf einen Speicher mit der doppelten Zugriffszeit könnte dann ein 3-2-2-2-Burst sein.
Ein Prozessor in Harvard-Architektur kann beim Burst-Zugriff den Vorteil dieser Architektur voll ausspielen. Unter der Voraussetzung, daß sowohl Daten- als auch Instruktionsspeicher (Cache) Burstzugriffe unterstützen, kann mit überlappenden Bursts in jedem Zyklus gleichzeitig eine Instruktion gelesen und ein Datenwort gelesen oder geschrieben werden. Damit kann ein Harvard-Prozessor den doppelten Datendurchsatz eines von-Neumann-Prozessors erreichen.
Beim Präprozessor sind die lokalen Speicher mit schnellen synchronen Burst-RAM-Modulen aufgebaut, so daß alle lokalen Zugriffe[40] als 2-1-1-1 Burst ausgeführt werden können. Auf die in den lokalen EPROMs abgelegten Instruktionen kann in einem 2-1-1-1...1 Burst mit bis zu 256 Schritten zugegriffen werden.

## 3.4 Designziele und erreichte Ergebnisse

Das Ziel beim Entwurf des Präprozessors war, ein Testsystem aufzubauen, mit dem die Anwendbarkeit von Cluster Counting in der Praxis demonstriert werden kann. In der vorliegenden Arbeit wurde der Kern des Prozessorsystems entwickelt und aufgebaut. Die Vervollständigung des Prozessorsystems oder mögliche Erweiterungen sind ein Thema zukünftiger Arbeiten.
Beim Entwurf des Präprozessorsystems waren folgende Ziele ins Auge gefaßt worden:

- Schneller Dateneingangsbus.
- Schnelle interne Datenverteilung.
- Eine von der Datenverarbeitung unabhängige Haushaltsführung.
- Serielle Schnittstellen für die langsame Kommunikation während der Entwicklung.
- Slave-Prozessoren mit einer Taktrate von 40 MHz und einem kontinuierlichen Durchsatz von möglichst $40 \cdot 10^6$ Instruktionen und Datenworten pro Sekunde, also jeweils eine Instruktion und ein Datenwort pro Takt.
- Datenausgabe und modulare Erweiterbarkeit über ein VMEbus-Interface.
- Hilfsfunktionen und -schnittstellen zur Einbindung des Präprozessors in ein Experiment der Hochenergiephysik.

Nach der Fertigstellung kann der Präprozessor zusammen mit den ADCs von Tektronix in ein Testsystem eingebaut werden, um die Realisierbarkeit von Cluster Counting zur Teilchenidentifizierung zu demonstrieren.
Im Bezug auf die obigen Ziele ergibt sich für den gegenwärtigen Zustand des Präprozessors das Folgende:

[40] Im Prinzip - ein Designfehler verhindert dies zur Zeit

### 3.4.1 Der schnelle Dateneingangsbus

Ursprünglich war, in Zusammenarbeit mit Tektronix, vorgesehen, daß die Meßwerte aus mehreren schnellen ADCs über einen schnellen parallelen Datenbus zu einem Präprozessormodul übertragen werden. Wegen der Geschwindigkeit und der Möglichkeit, viele Sender mit einem Empfänger zu verbinden, wurde dieser Bus in ECL-Technologie ausgeführt. Während der Entwicklung der ADC-Module hin zum nun kommerziell erhältlichen TDS 645 wurde jedoch, aus verschiedenen Gründen, vom Hersteller Tektronix entschieden, den schnellen Datenausgangsbus nicht mit in den ADC aufzunehmen. Damit entfiel auch eine mögliche Quelle für Testdaten für den schnellen Dateneingang des Präprozessormoduls, womit der Aufbau und die Überprüfung dieses schnellen Eingangsbusses eine niedrige Priorität erhielt. Zum gegenwärtigen Zeitpunkt ist nur ein Teil der notwendigen Bauteile bestückt, ein Test des schnellen Dateneingangsbusses war demzufolge auch nicht möglich.

### 3.4.2 Die interne Datenverteilung

Die interne Datenverteilung ist eng an den Dateneingangsbus gekoppelt. Da aufgrund der oben beschriebenen Umstände dieser Dateneingangsbus bei der Fertigstellung des Prozessors nicht mehr benötigt wurde, wurden auch die Elemente der internen Datenverteilung zunächst nicht getestet.

### 3.4.3 Unabhängige Haushaltsführung

Der Master-Prozessor, zusammen mit seiner unmittelbaren Umgebung, die aus Instruktions-RAM, Instruktions-ROM und Daten-RAM besteht, ist vom Entwurf her in der Lage, unabhängig vom Rest der Platine zu arbeiten. Die Verbindungen zwischen dem Master und dem Rest des Präprozessors werden im normalen Betrieb über eine Gruppe von programmgesteuerten Datenschaltern hergestellt. Indem die Steuerleitungen dieser Datenschalter zunächst fest verdrahtet wurden, wurde die Möglichkeit zur Übertragung von Informationen durch diese Schalter unterbunden. Damit war es möglich, den Bereich des Master-Prozessors unabhängig vom Rest der Platine in Betrieb zu nehmen und zu testen.
Nach anfänglichen Schwierigkeiten konnte der Master-Prozessor in Betrieb genommen werden. Der Prozessor selbst ist in der Lage, innerhalb der lokalen Umgebung mit der vollen Taktrate von 20 MHz zu arbeiten. Da es später beim Betrieb des Slaves zu Problemen bei der Datenübertragung zwischen dem Slave und der Hauptplatine kam, wurde die Taktfrequenz des Hauptoszillators von 40 MHz auf 30 MHz verringert. Da die Takterzeugung für alle Vorgänge auf der Platine von diesem einen Oszillator aus erfolgt, wurde damit auch die Taktfrequenz des Master-Prozessors auf 15 MHz reduziert. Abbildung 3.18 zeigt das Zeitverhalten während der lokalen Zugriffe des Masters.
Die drei Diagramme zeigen den Beginn eines Burst-Zugriffes auf das Instruktions-ROM (a), einen einzelnen Zugriff auf das Instruktions-RAM (b) und einen einzelnen Zugriff auf das Daten-RAM (c). Kanal 1 zeigt das Signal am Takteingang des Master-Prozessors, Kanal 2 die Anfrage des Master-Prozessors ($\overline{\mathrm{IREQ}}$[41] in den Teilbildern a und b, $\overline{\mathrm{DREQ}}$[42] in Teilbild c), Kanal 3 die da-

[41] **I**nstruction **Req**uest, Anfrage nach Instruktionen, der Überstrich indiziert ein Signal, das im aktiven Zustand den niedrigeren Pegel annimmt (‘active low’)

[42] **D**ata **Req**uest, Anfrage nach Daten

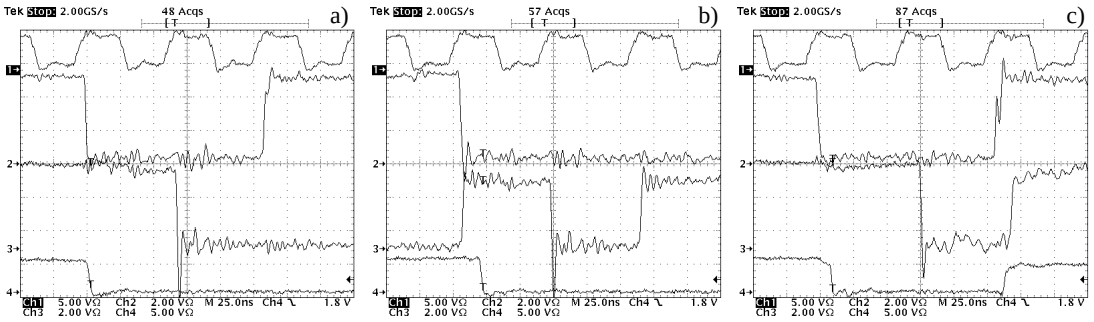



Abbildung 3.18: Zeitverhalten des Master-Prozessors bei Zugriffen in der unmittelbaren Umgebung. Die Zeitbasis beträgt 25 ns pro Einheit, die Amplitudenauflösung 5 (oberste und unterste Kurve) bzw. 2 Volt pro Einheit (mittlere Kurven).

zugehörige Antwort der Steuerlogik ( $\overline{\mathrm{IRDY}}$[43] beziehungsweise $\overline{\mathrm{DRDY}}$[44]) und Kanal 4 das Triggersignal, das in diesen Fällen vom $\overline{\mathrm{CS}}$[45]-Anschluß des jeweiligen Bausteines abgeleitet wurde.
In allen drei Diagrammen ist die Ausgabe des $\overline{\mathrm{IREQ}}$- beziehungsweise $\overline{\mathrm{DREQ}}$- Signales zeitlich an die erste im jeweiligen Teilbild zu sehende steigende Flanke des Taktsignals gekoppelt. Die entsprechende $\overline{\mathrm{IRDY}}$- beziehungsweise $\overline{\mathrm{DRDY}}$-Antwort wird mit der nächsten steigenden Taktflanke registriert, die der fallenden Flanke des Antwortsignals folgt.
Der Zugriff im Burst-Modus auf das Instruktions-ROM enthält einen Wartezyklus zu Beginn des Zugriffs. Mit der ersten steigenden Flanke des Taktsignals wird das $\overline{\mathrm{IREQ}}$-Signal ausgegeben. Bis nach der zweiten steigenden Flanke gibt es keine Pegeländerungen, dieser zweite Taktzyklus ist als Wartezustand notwendig. Zur dritten steigenden Flanke liegt das Instruktionswort an der CPU an, dies wird durch den entsprechenden Pegel des $\overline{\mathrm{IRDY}}$-Signales angezeigt. Mit jedem weiteren Taktimpuls kann eine weiteres Instruktionswort übernommen werden, der Zugriff läuft also in einem 3-1-1-...-Zyklus ab.
Beim einzelnen Zugriff auf das Instruktions-RAM in Abbildung 3.18.b muß ebenfalls ein Wartetakt eingelegt werden. In der Abbildung ist zu sehen, daß das $\overline{\mathrm{IREQ}}$-Signal nach der $\overline{\mathrm{IRDY}}$-Antwort der Steuerlogik nicht wieder in den Ruhezustand zurückkehrt. Dies ist darauf zurückzuführen, daß die CPU hier einen sogenannten 'Pipeline'-Zugriff nutzt, bei dem die Adresse für den nächsten Zyklus bereits während der Datenübertragung des laufenden Zyklus übermittelt wird. Dadurch kann der Zugriff des Master-Prozessors auf sein lokales Instruktions-RAM in einem 3-2-2-...-Zyklus ablaufen. Ein Burst-Zugriff ist hier durch einen Fehler in der Verdrahtung der Adreßleitungen der RAMs nicht möglich.
Der dritte Teil der Abbildung schließlich zeigt den Zugriff auf das lokale Daten-RAM des Masters. Hier gilt wieder das, was auch schon für das Instruktions-RAM gesagt wurde. In der Abbildung ist allerdings nur ein einzelner Zugriff zu sehen, auf den kein unmittelbarer weiterer Zugriff erfolgt. Dies liegt mit daran, daß konsekutive Zugriffe im Burst-Modus auf den Datenspeicherbereich vom C-Compiler[46] nicht unterstützt werden. Im Unterschied zu Zugriffen auf den Instruktionsspeicher,

[43] **I**nstruction **R**e**a**d**y**, Instruktion steht zur Verfügung

[44] **D**ata **R**e**a**d**y**, Daten stehen zur Verfügung

[45] **C**hip **S**elect, allgemeine Aktivierung des Bausteines

[46] Zur Erstellung eines lauffähigen Programms werden hier drei Programme benötigt: a) der Compiler ('Übersetzer'), der das in einer Hochsprache (hier 'C') geschriebene Programm in die Maschinensprache des Prozessors (Assembler-

die von der CPU aus prinzipiell als Burst-Zugriffe initiiert werden, ist für solche Zugriffe auf den Datenspeicherbereich ein spezieller Assembler-Befehl[47] notwendig, der vom Compiler nicht erzeugt wird. Unter diesen Umständen benötigt jeder Zugriff auf den Datenspeicher 3 Takte.
Nachdem der Master-Prozessor innerhalb seiner unmittelbaren Umgebung arbeitete, konnten die Zugriffe auf die Hilfsfunktionen programmiert und überprüft werden. Die beiden für die weitere Entwicklung notwendigen Elemente in diesem Bereich waren die gemeinsamen EPROMs sowie das globale RAM. Aus den gemeinsamen EPROMs kann nun nach dem Bootvorgang des Masters (der dazu notwendige Code befindet sich im lokalen Instruktions-ROM) ein Programm in das Instruktions-RAM geladen werden. Über einen Satz von Miniaturschaltern kann eines von 8 möglichen Programmen ausgewählt werden.

### 3.4.4 Serielle Schnittstellen

Nachdem die Funktion des Master Prozessors in seiner unmittelbaren Umgebung überprüft war, konnten die ersten beiden der seriellen Schnittstellen in Betrieb genommen werden. Diese beiden, mittels des Interface-Bausteines Z85C30 [60] realisierten Schnittstellen waren ursprünglich für die direkte Fehlersuche und -beseitigung in den Programmen des Master-Prozessors vorgesehen. Nach der Inbetriebnahme dieser Schnittstellen stellte sich jedoch heraus, daß es in der vorliegenden Version der Entwicklungsumgebung nicht möglich war, die entsprechenden Programmkomponenten zum Funktionieren zu bringen.
Gegenwärtig findet der größte Teil der externen Kommunikation der Präprozessorplatine über eine dieser seriellen Schnittstellen statt. Ein angeschlossener Rechner (PC) dient dann als Terminal zur Eingabe von einfachen Befehlen und zur Anzeige der Ergebnisse eines Programms, das auf dem Präprozessor abläuft.
Über diese Terminalverbindung können auch neue Programme für den Master- und die Slave-Prozessoren geladen werden, und es können die Daten für die Clustersuche zwischen den Rechnern ausgetauscht werden. Die Datenübertragungsrate dieser Verbindung beträgt bis zu 1400 Bytes pro Sekunde, was für die gestellten Anforderungen ausreichend ist.
Die zweite Möglichkeit, der Umgebung etwas über den Zustand des Präprozessors mitzuteilen, besteht aus einer kleine Anzeige mit 8 Leuchtdioden, die durch entsprechende Programmanweisungen von der parallelen TTL-Schnittstelle aus angesteuert werden. Dieses System ist zwar sehr primitiv und nicht zur Übermittlung komplexer Informationen geeignet, gleichzeitig aber auch sehr robust.

### 3.4.5 Slaves

Für den Slave[48] gilt ähnliches wie für den Master-Prozessor. Auch hier war es möglich, durch Abkoppeln der externen Leitungen den Prozessor des Slaves zunächst einmal ausschließlich im lokalen Speicherbereich arbeiten zu lassen. Der größte Teil der Systemprogramme konnte hierfür direkt vom Master übernommen werden, lediglich die Adressierungsgrenzen für die einzelnen Speicherbereiche waren dem leicht unterschiedlichen Aufbau anzupassen.

---

Sprache) übersetzt; b) der Assembler (‘Zusammenbauer’), der aus dieser Assemblersprache Programmsegmente im Binärcode aufbaut und c) der Linker (‘Verbinder’), der diese Programmsegmente zu einem einheitlichen Programm zusammenbindet.

[47] STOREM bzw. LOADM, store/load multiple, mehrfaches schreiben oder lesen

[48] Eigentlich ‘die Slaves’, aber gegenwärtig ist nur eine der Platinen vollständig bestückt und getestet.

Nach der Freigabe der Datenübertragung zwischen Master und Slave mußte jedoch festgestellt werden, daß ein zuverlässiger Datenaustausch zwischen den beiden Prozessoren nicht immer möglich war. Für diesen Datenaustausch müssen verschiedene Registerstufen auf der Platine sowie die beiden beteiligten Prozessoren zueinander synchronisiert werden. Dies geschieht über eine relative Verschiebung der Ausgänge eines gemeinsamen Taktgenerators. Die Verteilung dieser Ausgänge auf die beteiligten Register hat sich in der vorliegenden Version der Präprozessorplatine als ungünstig herausgestellt, eine Änderung dieser Verteilung ließe sich mit einem neuen Platinenlayout realisieren.
Für eine zuverlässige Datenübertragung zwischen Master und Slave wurde schließlich die Taktfrequenz des Hauptoszillators von 40 MHz auf 30 MHz reduziert. Damit wurde gleichzeitig die Taktfrequenz des Masters auf 15 MHz reduziert. Zusätzlich wurden für die Datenübertragung zwischen Master und Slave Wartezyklen eingefügt.

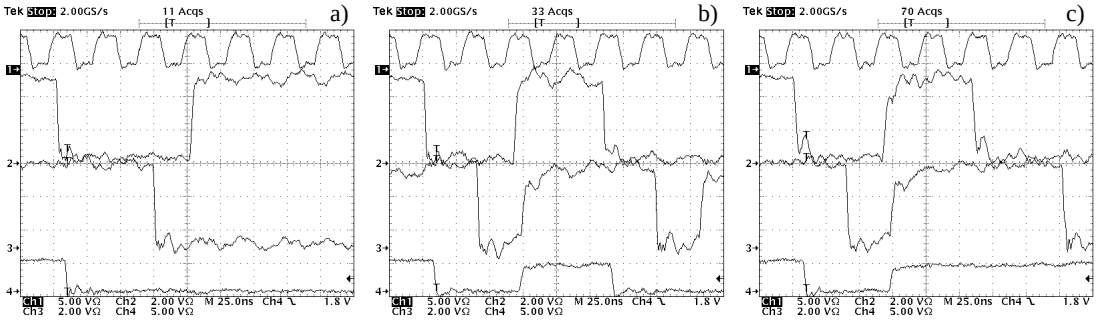



Abbildung 3.19: Zeitverhalten des Slave-Prozessors bei Zugriffen in der unmittelbaren Umgebung. Zeitbasis und Amplitudenauflösungen stimmen mit Abbildung 3.18 überein.

Im Ergebnis ist in Abbildung 3.19.a zu sehen, daß für den ersten Zugriff auf das Instruktions-ROM des Slaves insgesamt 4 Takte notwendig sind. Jeder weitere Zugriff während eines Bursts kann dann jedoch innerhalb eines Taktes abgearbeitet werden, so daß hier eine 4-1-1-. . . -Sequenz vorliegt.
Wegen des etwas günstigeren Zeitverhaltens beim ersten Zugriff kann beim Instruktions-RAM in Teil b der Abbildung einer der Wartezyklen entfallen, ein einzelner Zugriff also innerhalb von drei Taktzyklen abgearbeitet werden. Wie auch beim Master können aufeinanderfolgende Zugriffe auf das Instruktions-RAM in einem 3-2-2-. . . -Zyklus erfolgen.
Mit einer angepaßten Verteilung der Taktsignale sollte ein 2-1-1-. . . -Burst beim Instruktions-ROM sowie durchgehend 2-2-2-. . . -Zugriffe bei den RAMs möglich sein, womit sich die Verarbeitungsgeschwindigkeit des Präprozessors um einen beträchtlichen Faktor steigern ließe. Wie groß dieser Faktor ist, hängt davon ab, wie hoch der Programmanteil ist, der aus den Instruktions-RAMs heraus abgearbeitet werden muß. Durch die Korrektur der Verdrahtung der RAM-Bausteine schließlich sollte auch hier der Zugriff im Burst-Modus möglich sein. Gegenüber von einzelnen Zugriffen bewirkt dies eine Verringerung der Zugriffszeit auf 5/8 des Wertes ohne Burstzugriff (fünf Takte für vier Instruktionen im 2-1-1-1 Burst im Verhältnis zu 8 Takten für vier Instruktionen ohne Burst). Zusammen mit dem dann ebenfalls möglichen verschachtelten Zugriff auf Instruktions- und Datenspeicher sollte eine Beschleunigung des gesamten Präprozessorsystems um einen Faktor zwei sicher, einen Faktor drei wahrscheinlich möglich sein.
Durch die Vergrößerung der Taktrate auf die im ursprünglichen Entwurf vorgesehenen 40 MHz

schließlich kann der Präprozessor etwa um einen Faktor vier schneller arbeiten, als es in der jetzigen Version möglich ist.

### 3.4.6 Die VMEbus-Schnittstelle

Die Bausteine dieser Schnittstelle befinden sich auf der Platine. Auf die (insgesamt 100) internen Register der beiden Bausteine kann vom Master aus zugegriffen werden. Die weitere Funktion dieses Chipsatzes war für die Überprüfung der Kernfunktionen nicht notwendig.

### 3.4.7 Hilfsfunktionen

Zu den Hilfsfunktionen zählen auch mehrere Schnittstellen mit NIM-Pegel, die bis jetzt nicht weiter erwähnt wurden. Die Bauelemente dieser Schnittstellen sind auf der Platine montiert, und die grundsätzliche Funktionsfähigkeit der Schaltung wurde an einem Prototypen überprüft. Ansonsten sind auch diese Elemente unabhängig von den Kernfunktionen des Präprozessors.

### 3.4.8 Aufgetretene Probleme

Abschließend seien hier die größten beim Aufbau des Präprozessors aufgetretenen Probleme kurz angerissen:

- Platinenentwurf: Zum Zeitpunkt des Entwurf des Präprozessors handelte es sich um ein relativ umfangreiches Projekt, wodurch sowohl das verwendete Entwicklungsprogramm als auch der zur Verfügung stehende Rechner sehr stark belastet waren. Nach der Fertigstellung der Platine zeigte sich dies in einigen Fehlern, die durch eine freie Verdrahtung der entsprechenden Leitungen behoben werden mußten.

- Entwurfsfehler: Bei einem Projekt der vorliegenden Komplexität werden sich Fehler im Entwurf wohl nie ganz vermeiden lassen. Der schwerwiegendste Fehler in diesem Fall liegt bei der Verdrahtung der Adressen der RAM-Speicherbausteine, wodurch ein Zugriff im Burst-Modus gegenwärtig nicht möglich ist.

- Platinenfertigung: Auch hier gilt wieder, daß eine Platine dieser Größe hohe Ansprüche an die Fertigungsqualität stellt. Die Qualität der vorliegenden Platine ist für einen Prototypen noch akzeptabel, für den Fall einer Serienfertigung sollten aber strengere Ansprüche an die Kontrolle der Platinenfertigung gestellt werden.

- Programmierung: Zu der gewählten CPU Am29000™/Am29050™gehört auch eine vollständige Programmierumgebung. Hier mußte festgestellt werden, daß eines der Module[49] dieser Umgebung, welches für die Entwicklung des Präprozessors eine große Hilfe dargestellt hätte, in der vorliegenden Version dieser Entwicklungsumgebung nicht funktionsfähig war. In der letzten (aktuellen) Version der Entwicklungsumgebung fehlt diese Version dieses Moduls ganz.

[49] Bei diesem Modul handelt es sich um einen residenten Debugger, der Test und Fehlersuche auf dem Präprozessor über zwei serielle Schnittstellen hätte ermöglichen sollen. In der ursprünglichen Version der Entwicklungsumgebung war dieser Debugger zwar vorhanden, aber nicht funktionsfähig, in späteren Versionen wurde diese Option nur noch für eine Auswahl festgelegter Entwicklungsplatinen unterstützt.

Dazu kam, daß durch vereinzelte Fehler in den Bibliotheken[50] und im Compiler große Teile der Programme im erzeugten Assembler-Code schrittweise von Hand überprüft werden mußten. Durch eine aktuelle Version der Entwicklungsumgebung hätte sich dieses Problem wahrscheinlich beheben lassen, die fehlende Unterstützung für eine nicht von AMD festgelegte Entwicklungsumgebung schloß diese Möglichkeit jedoch aus.

[50] Zu einer Programmierumgebung für einen bestimmten Prozessor gehören auch Bibliotheken, die vom Hersteller bereits compilierte und assemblierte (s.o.) Unterprogramme zur eigenen Verwendung bereithalten. Zum Inhalt dieser Bibliotheken können zum Beispiel Unterprogramme zur optimierten Berechnung mathematischer Funktionen gehören.

# Kapitel 4

# Algorithmen für Cluster Counting

Neben dem im vorangegangenen Abschnitt beschriebenen Rechneraufbau mußte auch ein geeigneter Ansatz für passende Programme zur Clustersuche entwickelt werden. Durch ein Programm, das die Fähigkeiten eines gegebene Rechneraufbaus nicht ausnutzt, kann jeder Rechner auf einen nahezu beliebig geringen Datendurchsatz abgebremst werden.
Umgekehrt kann bei einer gegebenen Problemstellung, die nach einer bestimmten Klasse von Algorithmen verlangt, durch die Wahl (oder den Entwurf) eines ungeeigneten Rechners die Verarbeitungszeit ebenfalls nahezu beliebig in die Höhe getrieben werden. Im Falle des Präprozessors für Cluster Counting sollte die Möglichkeit offengehalten werden, möglichst verschiedene Algorithmen testen zu können. Aus den Eigenschaften der Daten lassen sich allerdings gemeinsame Eigenschaften der für diese Anwendung sinnvollen Algorithmen definieren:

- Kurze Datensätze sollen möglichst schnell nach Clustersignalen abgesucht werden
- Wegen der gewünschten kurzen Verarbeitungszeit müssen die Programme kompakt sein
- Ein maximaler Datendurchsatz kann mit einem Datenflußalgorithmus erreicht werden, für eine maximale Leistung ist aber auch ein spezieller Rechneraufbau nötig (siehe Abschnitt 5.2)

Diese Überlegungen flossen mit in den Entwurf des Präprozessors ein und sollten auch bei der Entwicklung eines passenden Algorithmus mit einfließen, um den Rechner nicht unnötig auszubremsen.

## 4.1 Wünschenswerte Eigenschaften eines Algorithmus zur Clustersuche

Abbildung 2.2 zeigt ein typisches Beispiel für einen Ausschnitt eines bei Cluster Counting anfallenden Rohdatensatzes aus einer Testdatennahme von L. Cerrito et. al. im Oktober 1995 am CERN in Genf [35]. Während dieser Datennahme wurden Strahl T9 des Protonensynchrotrons (PS) Daten genommen. Der Teilchenstrahl stellte unterschiedliche Mischungen aus Pionen, Protonen und Kaonen bei Impulsen zwischen 1 und 5 GeV zur Verfügung. Die Signale aus einer für diesen Zweck gebauten Testkammer wurden über Vorverstärker auf der Basis des MAR-8 (siehe Abschnitt 2.2.1) ausgelesen und einer Gruppe von Analog-Digitalwandlern des Typs TDS 645

von Tektronix (siehe Abschnitt 2.3) zugeführt. Zwei Apple-Macintosh-Rechner wurden für die Auslese der ADCs und eine Vorab-Kontrolle der Signale verwendet.
Die Signale setzen sich zusammen aus:

- Einer Grundlinie ('Baseline'), deren Wert vom Auslesekanal abhängt und sich zeitlich langsam ändern kann, hauptsächlich durch die Temperaturdrift in den Vorverstärkern und ADCs

- Einem Rauschanteil, der zum Teil von den Vorverstärkern, hauptsächlich aber von den ADCs herrührt. Bei den für diese Datennahme verwendeten ADCs handelte es sich noch um Prototypen, bei denen neben einem relativ starken Rauschen noch weitere Probleme zutage traten, die sich mit der Einstreuung von internen Taktsignalen, entweder in den Vorverstärkerstufen oder den Wandlerstufen selbst, erklären lassen. In Abbildung 4.3 sowie dem aus etwa 100 solcher Einzelsignale zusammengesetzten Signal in Abbildung 4.4 sind systematische Oszillationen sowie eine Schwebung im Rauschen zu erkennen, als deren Ursache eigentlich nur die ADCs in Frage kommen.

  Das Rauschen kann durch geeignete Wahl der Bauelemente und ihrer Betriebsparameter minimiert werden, bei den erforderlichen Verstärkungsfaktoren wird es aber immer eine gewisse Rolle spielen. Die in den ADCs erzeugten Störungen sollten in den Serienmodellen allerdings nicht mehr auftreten.

- Dem eigentlichen Signal, das wiederum aus einer Überlagerung der Signale einzelner Cluster besteht. Wenn der Clustererzeugung ein Poisson-Prozeß zugrundeliegt, sollte der zeitliche Abstand zweier Cluster einer Exponentialverteilung folgen. Die Größenverteilung der Cluster ist schwierig statistisch zu beschreiben, da neben der Verteilung der Zahl der Primärelektronen zusätzliche Fluktuationen bei der Elektronendrift und der Gasverstärkung eine Rolle spielen können.

Die Aufgabe eines Algorithmus zur Clustersuche ist es nun, aus diesem Signalgemisch die einzelnen Clustersignale zu extrahieren. Dazu müssen die Position und die Höhe jedes einzelnen dieser Signale über dem Untergrund aus Baseline, Rauschen und den restlichen Signalen erkannt werden. Diese Clustererkennung sollte nun in einer möglichst kurzen Zeit abgewickelt werden. Je weniger Zeit für die Verarbeitung eines Drahtsignales benötigt wird, desto weniger Prozessoren werden benötigt, um die Gesamtheit aller Drahtsignale in dem durch die Ereignisraten vorgegebenen Zeitrahmen zu verarbeiten. Auf der anderen Seite sollen die Cluster mit möglichst hoher Effizienz aus dem Untergrund herausgefiltert werden. Um diese Ziele zu erreichen können folgende Punkte hilfreich sein:

- Möglichst wenige Operationen pro Datenpunkt anwenden.

- Auf die 'Parallelisierbarkeit' des Algorithmus achten. Auch wenn ein gegebener Algorithmus zunächst nicht auf einem Parallelrechner ablaufen soll, sollte die Möglichkeit nicht ausgeschlossen werden. Darüberhinaus kann ein gut parallelisierbarer Algorithmus, d.h. ein Algorithmus mit geringer Abhängigkeit zwischen den Ergebnissen der einzelnen Stufen, auch die Architektur eines Pipeline-Prozessors effektiver nutzen.

- So weit wie möglich die *gesamte* in den Meßwerten vorhandene Information zur Clustererkennung heranziehen.

- Divisionen, Wurzeln und trigonometrische Funktionen so weit als möglich vermeiden. Mit der Ausnahme von hochspezialisierter Rechnerbestandteilen benötigen diese Funktionen ein mehrfaches der Rechenzeit der einfacheren Funktionen (Addition, Subtraktion, Multiplikation). Die effektivste Funktion für einen großen Teil der modernen Prozessoren ist 'Multiplizieren-und-Addieren', bei der eine Variable mit einer Konstanten (oder einer in einem vorherigen Schritt gewonnenen Variablen) multipliziert und anschließend zu einem Akkumulatorregister addiert wird. Viele Prozessorkerne können diese Funktion als kombinierte Operation in einem Takt ausführen.

## 4.2 Einfache Signalerkennung

Ein erster Algorithmus zur Clustersuche wurde am INFN Lecce von einer Arbeitsgruppe um F. Grancagnolo entwickelt und erfolgreich getestet [30, 31]. Für diesen Algorithmus wird das Signal eines einzelnen Clusters durch seine Amplitude, seine Anstiegszeit $T_r$ und seine Abfallzeit $T_f$ charakterisiert.
Zur Clustersuche werden zunächst die Meßwerte $Y(t)$ durch ein Tiefpaßfilter gefiltert, um das – überwiegend hochfrequente – elektronische Rauschen zu verringern. Als Grundlage für diese Filterung dient das Signal der ersten Meßwerte, bei dem davon ausgegangen werden kann, daß dort noch keine Clustersignale vorkommen.
Lokale Maxima in $Y(t)$ werden dann über folgende Kriterien als Clustersignale identifiziert:

- Amplitude des Maximums
- Amplitude der ersten Ableitung
- Konsistenz der Anstiegs- und Abfallzeiten mit den erwarteten Zeiten $T_r$ und $T_f$ für ein Clustersignal
- Abstand zwischen dem Maximum und den nächsten Minima
- Weitere topologische Kriterien zur Unterscheidung zwischen isolierten Impulsen und Impulsen auf der steigenden oder fallenden Flanke eines größeren Impulses

Mit diesem Algorithmus können in Simulationen bei einer Samplingrate von 2 GS/s Cluster ab einem zeitlichen Abstand von 2 ns voneinander unterschieden werden, ab einem zeitlichen Abstand von etwa 5 ns ist die Effizienz der Clustererkennung größer als 90 %.

## 4.3 Kurvenanpassung

Ein zweiter Algorithmus, der im Rahmen dieser Arbeit entwickelt und untersucht wurde, basiert auf einer direkten Anpassung einer Musterkurve nach der Methode der kleinsten Quadrate[1] an die Meßdaten.

[1] 'least-squares-fit' oder auch '$\chi^2$-Fit', Parameter einer Referenzkurve werden so lange variiert, bis die Summe der quadratischen Abweichung $\chi^2$ zwischen der Kurve und den Meßwerten minimal wird [39].

### 4.3.1 Ansatz

Als Ansatz für diese Kurvenanpassung wird eine durch äquidistante Punkte beschriebene Referenzkurve verwendet (Abbildung 4.1.a). Die Einschränkung auf äquidistante Punkte spielt in dieser Anwendung keine Rolle, da die Meßpunkte ebenfalls äquidistant sind, vereinfacht aber die Rechnung.

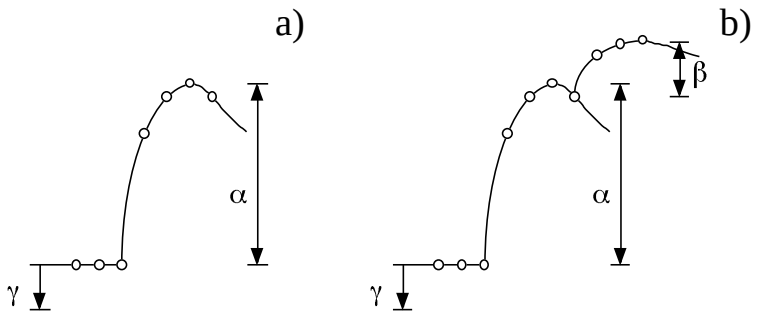



Abbildung 4.1: Schematische Darstellung der Referenzkurven

Die Musterkurve in Abbildung 4.1.a besitzt zwei freie Parameter, den Skalenfaktor in y, $\alpha$, und den Versatz in y, $\gamma$ ('Offset'). Der Versatz $\gamma$ entspricht der Grundlinie im gemessenen Signal, der Skalenfaktor $\alpha$ einer relativen Signalhöhe. Diese Referenzkurve wird nun versuchsweise an jedes Teilsegment der Meßkurve angepaßt, wobei als Maß für die Güte diese Anpassung die Summe der quadratischen Differenzen zwischen den y-Werten der Meßkurve und der angepaßten Referenzkurve verwendet wird. Durch diese Kurvenanpassung wird ein Maximum der verfügbaren Information ausgewertet, da jeder der Meßpunkte mit in diese Anpassung einfließt.
Bei ersten Versuchen mit diesem Algorithmus zeigte es sich, daß eine Kurve bekannter Form auch in einem stark verrauschten Signal noch gut gefunden werden kann. Die Erkennungsleistung hängt allerdings stark von der Zahl der für die Anpassung verwendeten Punkte ab, so daß durch eine Erweiterung der Referenzkurve nach Rechts einzelne Signale immer besser erkannt werden können. Für die reale Clustersuche stellt sich hierbei aber das Problem, daß die Clustersignale oft nur einen geringen Abstand zueinander aufweisen. Liegen aber mehrere Signale innerhalb des durch die Referenzkurve beschriebenen Bereichs, so können diese nur sehr schlecht voneinander getrennt werden. Beim Versuch der Anpassung der Referenzkurve an eine solche Gruppe von Clustersignalen wird in der Regel nur ein einzelnes, schlecht angepaßtes Clustersignal mit zu großer Amplitude erkannt.
Ein einfacher Ausweg aus diesem Problem ist, an der frühesten sinnvollen Stelle in der Musterkurve das Signal eines zweiten Clusters hinzuzufügen (Abbildung 4.1.b). Die früheste sinnvolle Stelle hierfür ist kurz nach dem Maximum des ersten Clustersignals. Dieses zweite Signal wird mit einem weiteren Skalierungsparameter $\beta$ beschrieben. Ein nächster Schritt wäre, den Abstand zwischen den beiden Signalen durch einen vierten Parameter zu beschreiben. Der zusätzliche Nutzen im Verhältnis zum nötigen Aufwand wird dann aber sehr gering, da dieses zweite Signal ja ausschließlich dazu dient, die Parametrisierung für das erste Signal zu verbessern – jegliche Information über das zweite Signal wird nach der Erkennung und Parametrisierung des ersten verworfen.
Die Anpassungsparameter $\alpha$, $\beta$ und $\gamma$ werden folgendermaßen berechnet:

Die $n$ Punkte der Referenzkurve $y_r(i)$ werden beschrieben durch

$$y_r(i) = \alpha * y_0(i) + \beta * y_1(i) + \gamma. \tag{4.1}$$

Die Abweichung $\chi'^2$ zwischen den Punkten der Referenzkurve und den Meßwerten $y_m(i)$ sei definiert als

$$\chi'^2 = \chi^2 \cdot \sigma^2 := \sum_{i=1}^{n} \left[ y_m(i) - [\alpha y_0(i) + \beta y_1(i) + \gamma] \right]^2 . \tag{4.2}$$

Hierbei beschreibt $\sigma$ die Standardabweichung des Rauschens.
Die partielle Ableitung nach den drei Parametern ergibt:

$$\frac{\partial \chi'^2}{\partial \alpha} = \sum_{i=1}^{n} \left[ -2 y_m y_0 + 2\alpha y_0^2 + 2\beta y_0 y_1 + 2\gamma y_0 \right] \tag{4.3}$$

$$\frac{\partial \chi'^2}{\partial \beta} = \sum_{i=1}^{n} \left[ -2 y_m y_1 + 2\beta y_1^2 + 2\alpha y_0 y_1 + 2\gamma y_1 \right] \tag{4.4}$$

$$\frac{\partial \chi'^2}{\partial \gamma} = \sum_{i=1}^{n} \left[ -2 y_m + 2\gamma + 2\alpha y_0 + 2\beta y_1 \right] \tag{4.5}$$

Nullsetzen dieser Ableitungen, ineinander Einsetzen und Auflösen nach $\alpha$, $\beta$ und $\gamma$ liefert schließlich:

$$\beta = \frac{\sum_i y_m y_0 - \frac{1}{n} \sum_i y_m \sum_i y_0 - \frac{\left( \sum_i y_m y_1 - \frac{1}{n} \sum_i y_m \sum_i y_1 \right) \left( \sum_i y_0^2 - \frac{1}{n} \left[ \sum_i y_0 \right]^2 \right)}{\sum_i y_0 y_1 - \frac{1}{n} \sum_i y_0 \sum_i y_1}}{\sum_i y_0 y_1 - \frac{1}{n} \sum_i y_0 \sum_i y_1 - \frac{\left( \sum y_1^2 - \frac{1}{n} \left[ \sum y_1 \right]^2 \right) \left( \sum_i y_0^2 - \frac{1}{n} \left[ \sum_i y_0 \right]^2 \right)}{\sum_i y_0 y_1 - \frac{1}{n} \sum_i y_0 \sum_i y_1}} \tag{4.6}$$

$$\alpha = \frac{\sum_i y_m y_1 - \frac{1}{n} \sum_i y_m \sum_i y_1 - \beta \left( \sum_i y_1^2 - \frac{1}{n} \left[ \sum_i y_1 \right]^2 \right)}{\sum_i y_0 y_1 - \frac{1}{n} \sum_i y_0 \sum_i y_1} \tag{4.7}$$

$$\gamma = \frac{1}{n} \left( \sum_i y_m - \alpha \sum_i y_0 - \beta \sum_i y_1 \right) \tag{4.8}$$

Dafür, daß diese Anpassung für *jeden* Punkt der Meßkurve durchgeführt werden soll, scheinen obige Gleichungen zunächst sehr viel Rechenaufwand zu bedeuten. Bei näherem Hinsehen aber ist zu erkennen, daß sich diese Summen fast nur aus konstanten Termen zusammensetzen. Die nötigen Rechenoperationen pro Meßpunkt beschränken sich damit darauf, zunächst folgende drei Summen zu bilden:

$$s_m = \sum_{i=1}^{n} y_m \tag{4.9}$$

$$s_{m0} = \sum_{i=1}^{n} y_m y_0 \tag{4.10}$$

$$s_{m1} = \sum_{i=1}^{n} y_m y_1 \tag{4.11}$$

und diese Summen dann in die Formeln einzusetzen, wobei alle anderen Summen durch die entsprechenden Konstanten ersetzt werden:

$$\beta = \frac{1}{c_{xx}} \left\{ s_{m0} - \frac{1}{n} c_0 s_m - \frac{1}{c_{01}} \left[ \left( s_{m1} - \frac{1}{n} c_1 s_m \right) c_{00} \right] \right\} \tag{4.12}$$

$$\alpha = \frac{1}{c_{01}} \left( s_{m1} - \frac{1}{n} c_1 s_m - \beta c_{11} \right) \tag{4.13}$$

$$\gamma = \frac{1}{n} \left( s_m - \alpha c_0 - \beta c_1 \right) . \tag{4.14}$$

In diesen Gleichungen tauchen zwar keine Wurzeln oder trigonometrischen Funktionen auf, es sind allerdings noch einige Divisionen vorhanden. Da es sich hierbei aber durchwegs um Divisionen durch Konstanten handelt, können diese im Programm dann durch Multiplikationen mit den entsprechenden Inversen ersetzt werden.
Die Berechnung der Parameter aus den Summen läßt sich weiter zusammenfassen:

$$\beta = a * s_m + b * s_{m0} + c * s_{m1} \tag{4.15}$$

$$\alpha = d * s_m + e * s_{m1} + f * \beta \tag{4.16}$$

$$\gamma = g * s_m + h * \alpha + i * \beta \tag{4.17}$$

wobei sich die Werte $a, b, c, d, e, f, g, h$ und $i$ folgendermaßen aus den Summen zusammensetzen:

$$a = \frac{c_{00} c_1}{c_{xx} c_{01} n} \tag{4.18}$$

$$b = \frac{1}{c_{xx}} \tag{4.19}$$

$$c = \frac{-c_{00}}{c_{xx} c_{01}} \tag{4.20}$$

$$d = \frac{-c_1}{c_{01} n} \tag{4.21}$$

$$e = \frac{1}{c_{01}} \tag{4.22}$$

$$f = -c_{11} \tag{4.23}$$

$$g = \frac{1}{n} \tag{4.24}$$

$$h = \frac{-c_0}{n} \tag{4.25}$$

$$i = \frac{-c_1}{n} \tag{4.26}$$

Abbildung 4.2 zeigt eine Übersicht des Ablaufs des Algorithmus. Aus den Meßwerten werden zunächst nach obigen Formeln die drei Parameter $\alpha$, $\beta$ und $\gamma$ abgeleitet. Im nächsten Schritt wird dann die quadratische Abweichung $\chi^2$ zwischen den Meßwerten und der durch $\alpha$, $\beta$ und $\gamma$ beschriebenen Kurve ermittelt. In einem dritten Schritt wird entschieden, ob an der entsprechenden Stelle ein Clustersignal vorhanden war. In der gegenwärtigen Version des Programms hängt diese Entscheidung von folgendem ab:

- $\chi^2$ kleiner als ein oberer Grenzwert $\chi^2_{\max}$

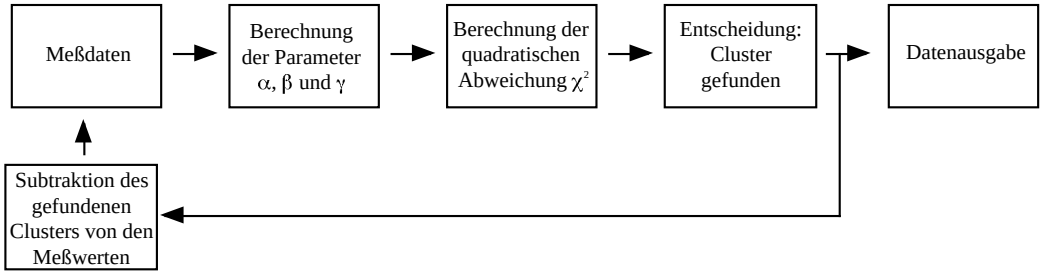



Abbildung 4.2: Blockdiagramm des vollständigen Algorithmus

- ein (lokales) Minimum in $\chi^2$
- $\alpha$ größer als ein unterer Grenzwert $\alpha_{\min}$.

Weitere Entscheidungskriterien können eingeführt werden. Nach den durchgeführten Tests scheint die Suche nach einem lokalen Minimum in $\chi^2$ aber das wirksamste Kriterium für die Existenz eines Clustersignals an der entsprechenden Stelle zu sein.
Findet der Algorithmus an einer Stelle ein Clustersignal, so wird diese Information zur späteren Auswertung abgespeichert. Danach wird an der Stelle, an der ein Clustersignal gefunden wurde, die auf die entsprechende Größe skalierte Musterkurve von den Meßdaten subtrahiert. Nach dieser Subtraktion sollte im entsprechenden Bereich nur noch das Grundrauschen zu finden sein, sofern nicht mehrere Clustersignale überlagert waren. Um eventuell überlagerte Clustersignale zu finden, werden die Daten bis vor den Beginn des Clustersignals 'zurückgespult' (das heißt, der Zeiger, der auf die aktuelle Position innerhalb des Datensatzes zeigt, wird auf eine Stelle vor den Beginn des Clustersignals zurückgesetzt) und der Suchvorgang von neuem gestartet.

### 4.3.2 Ausführung

Der kritische Punkt dieses Algorithmus ist natürlich die Übereinstimmung zwischen der gewählten Musterkurve und der tatsächlichen Kurvenform eines Clustersignals. Abbildung 2.2 zeigte ein Beispiel für ein typisches Cluster Counting-Signal.
Um die Form für ein Einzelclustersignal zu extrahieren, wurde aus insgesamt 105 Meßdatensätzen, die während der Testdatennahme im Oktober 1995 an CERN gewonnen wurden, jeweils ein isoliertes Clustersignal von Hand ausgewählt. Bei der Auswahl dieser Signale wurde darauf geachtet, möglichst schmale Signale zu wählen, um die Wahrscheinlichkeit, zwei überlagerte Cluster in die Auswahl mit einzuschließen, gering zu halten. Ein Beispiel für eines der ausgewählten Signale zeigt Abbildung 4.3.
Die Abtastrate bei diesen Messungen betrug 1 GS/s, so daß der Abstand zweier aufeinanderfolgender Datenpunkte genau 1 ns beträgt.
Diese 105 Signale wurden dann, entsprechend skaliert und verschoben, addiert, um die 'durchschnittliche' Referenzkurve zu ermitteln. Die Skalierung der Höhe richtete sich nach dem Mittelwert der 7 Punkte um die Maxima der einzelnen Kurven, die Verschiebung orientierte sich an den steigenden Flanken.
Um Zahlenwerte für die in den Anpassungsalgorithmus einzugebende Musterkurve zu erhalten, wurde die durchschnittliche Kurve durch eine angepaßte Funktion beschrieben. Als Ansatz für diese Funktion wurde ein Produkt aus einem Quotienten zweier Potenzfunktionen zur Beschreibung der ansteigenden Flanke und der Spitze sowie eines exponentiellen Abfalls zur Beschreibung der

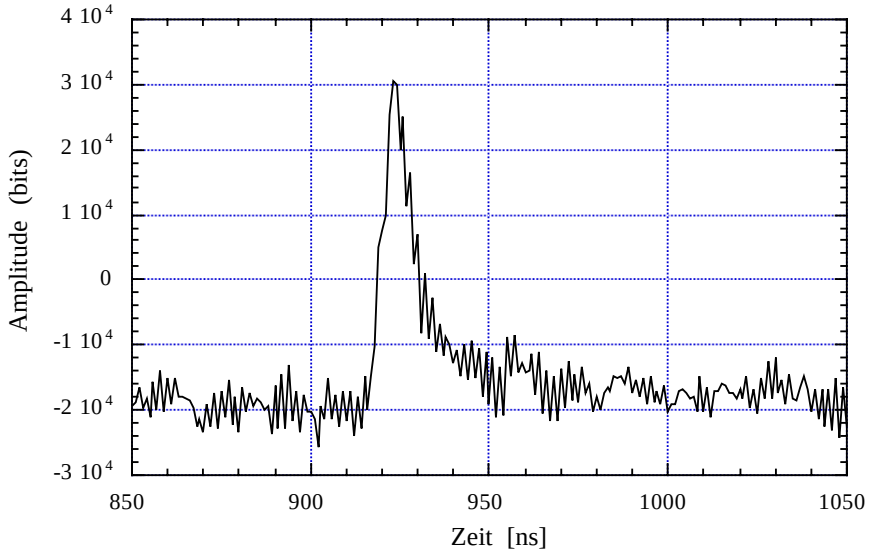



Abbildung 4.3: Einzelnes Clustersignal

fallenden Flanke gewählt. Für die Grundlinie wurde ein Mittelwert aus den Punkten am rechten und linken Rand angenommen. Die resultierende Funktion wird durch

$$y(x_i) = -0.022 + 410 * \frac{(x_i - 22.7)^{4.66}}{(x_i - 19.2)^{6.44}} * e^{-0.072(x_i - 32)} \tag{4.27}$$

für

$$23 \leq x_i < 100 \tag{4.28}$$

gut beschrieben. Der Wert $\chi^2$ für die mittlere quadratische Abweichung dieser 76 Punkte mit 6 freien Parametern von den Punkten des gemittelten Clustersignales beträgt $\chi^2 = 59.8$.
In Abbildung 4.4 ist eine Oszillation des Signales um die angepaßte Kurve zu erkennen. Hierbei handelt es sich nicht um ein Artefakt der Addition, diese Oszillation ist auch in den Meßdaten zu sehen. Die Frequenz der Schwingung legt nahe, daß es sich um Störungen aus den Analog-Digital-Wandlern handelt, die genaue Ursache ist aber unbekannt.
Der C-Quellcode für den schließlich in den Tests verwendeten Clustersuchalgorithmus, basierend auf der Formulierung 4.12 . . . 4.14, befindet sich in Anhang C.

## 4.4 Test des Algorithmus zur Kurvenanpassung

Der im vorangegangenen Abschnitt vorgestellte Algorithmus zur Kurvenanpassung wurde mit simulierten Daten getestet. Die Größe der simulierten Datensätze entspricht der der echten Datensätze aus der CERN-Datennahme 1995 (2048 Worte). Innerhalb dieser Datensätze wurden Kopien der aus der Anpassung erhaltenen Referenzkurve (Abbildung 4.4) mit vorgegebener Verteilung der Größe und des Abstands verteilt. Anschließend wurde ein nach einer Gaußfunktion verteiltes Zufallsrauschen zu dem Signal addiert, um das (weiße) elektronische Rauschen zu simulieren [32].

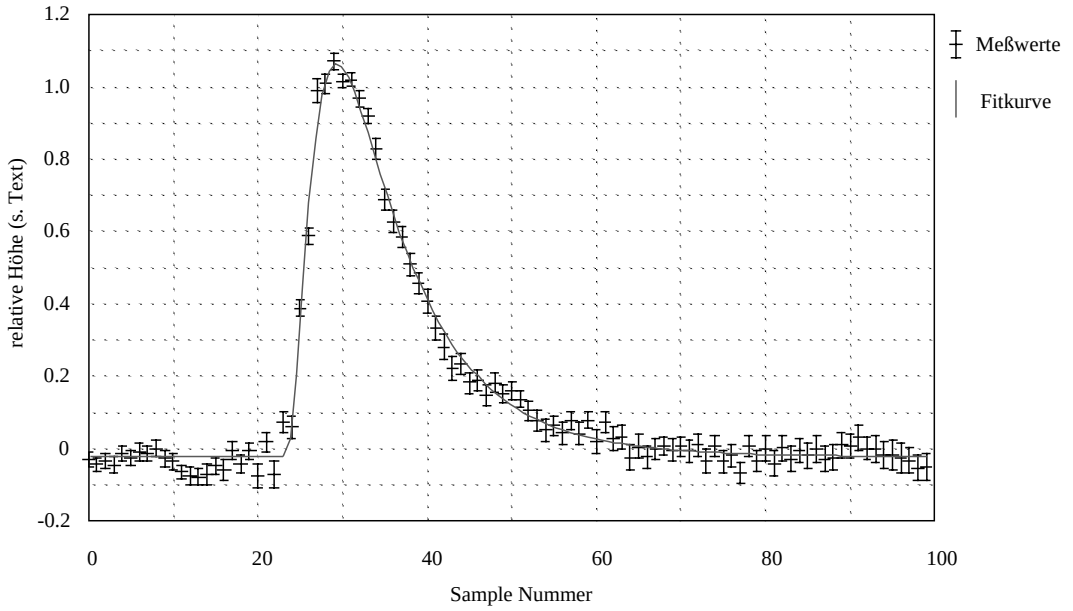



Abbildung 4.4: Das Referenzsignal mit der angepaßten Kurve

Das Ziel dieser Simulation war es, die entsprechenden gemessenen Signale während der Datennahme am CERN 1995 zumindest grob nachzubilden. Für die Clustergrößen galt hierbei, daß der Bereich der ADCs etwa 64000 Bit (genauer LSBs[2]) überstrich[3], während sich das Ausgangssignal ohne Vorliegen eines Clustersignales etwa um -20000 Bit bewegte. Eine Zuordnung zwischen diesen 'Bit' und einer Eingangsspannung ist zwar möglich, erfüllt hier aber keinen praktischen Zweck. Das Rauschen in den gemessenen Daten wies einen quadratischen Mittelwert von etwa $6 \cdot 10^6$ (Bit)$^2$ auf. Dies entspricht, bei einer Höhe der größten gemessenen Cluster von etwa 50000 Bit, einem Rauschen mit einem RMS[4]-Wert von etwa 5 % des Skalenendwerts.

### 4.4.1 Gemessene und simulierte Daten

Zur groben Simulation der gemessenen Daten wurde die Verteilung der Clustergrößen als doppelte Exponentialverteilung angesetzt. Für die Verteilung der Abstände zwischen zwei aufeinanderfolgenden Clustern wurde davon ausgegangen, daß es sich hier um statistisch unabhängige, poissonverteilte Ereignisse handelt. Wie in Abbildung 4.5 zu sehen ist, trifft zumindest die letztere Annahme offensichtlich nicht zu.
Die oberen beiden Teilabbildungen (4.5.a und 4.5.b) zeigen gemessene Clustersignale, die dritte Teilabbildung 4.5.c ein simuliertes Signal. Es ist deutlich zu erkennen, daß im Vergleich zum simulierten Signal die Cluster in den gemessenen Signalen zu Gruppen zusammengefaßt sind. In-

[2] **L**east **S**ignificant **B**it, geringstwertiges Bit

[3] Die nominelle Auflösung des ADCs betrug, wie schon in Abschnitt 2.3 erwähnt, 8 Bit, mit einer effektiven Auflösung bei der gewählten Sampling-Rate von 6.8 Bit. Durch die interne Struktur des ADCs wurden diese 6.8 Bit auf einen 16 Bit breiten Datenraum abgebildet

[4] **R**oot **M**ean **S**quare, Wurzel des quadratischen Mittelwertes

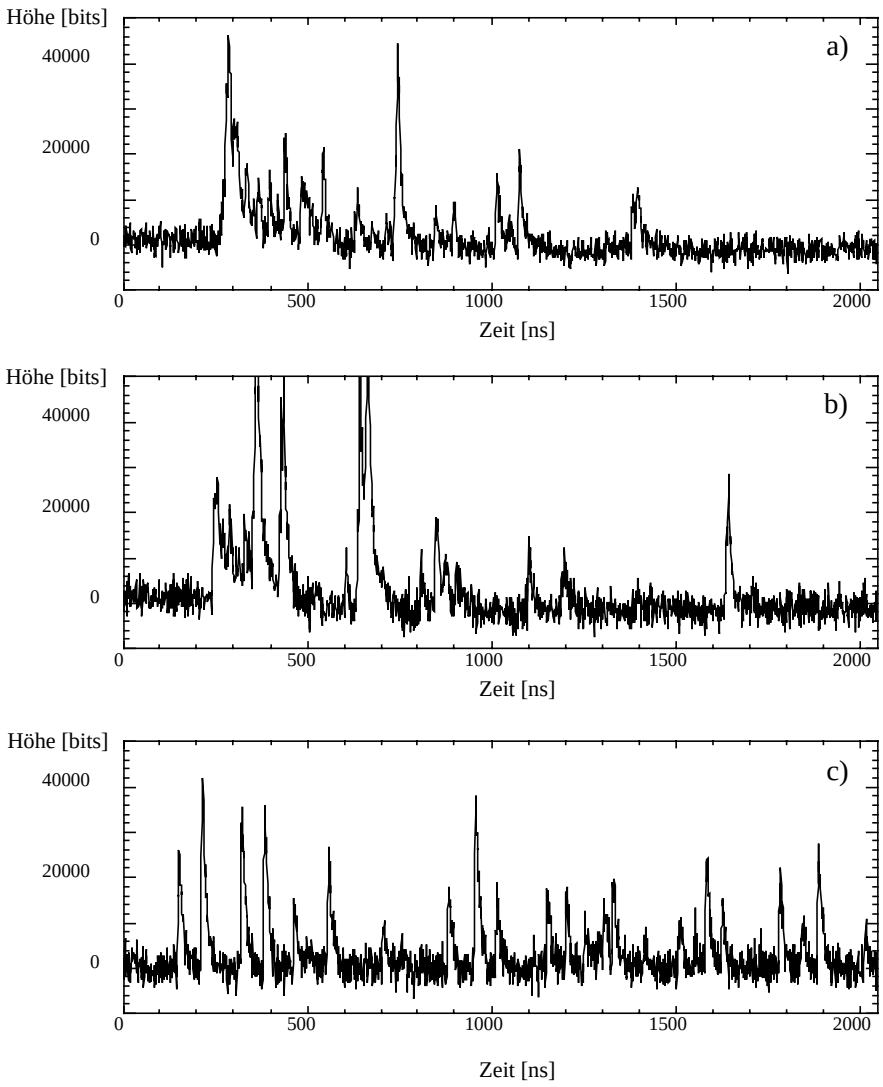



Abbildung 4.5: Vergleich zwischen gemessenen (Teilabbildungen a und b) und simulierten (Teilabbildung c) Clustersignalen

nerhalb dieser Gruppen ist die Clusterdichte zu Beginn relativ hoch und nimmt dann langsam ab. Um diese Struktur realistisch nachzubilden, ist jedoch ein weit höherer Aufwand bei der Simulation der Daten notwendig. Mit einer vollständigeren Simulation der Abläufe im Inneren einer Driftzelle werden von G. Cataldi [30] und von G. Cataldi, F. Grancagnolo und S. Spagnolo [31] auch realistischere simulierte Clustersignale erreicht.
Für den Test dieses Algorithmus sollte die Gruppierung der einzelnen Clustersignale jedoch von untergeordneter Bedeutung sein, da der Suchalgorithmus selbst jeweils nur mit einem 30 ns langen Ausschnitt arbeitet.

### 4.4.2 Verwendete Parameter

Um den Algorithmus testen zu können, mußten zunächst Parameter sowohl für die Erzeugung der simulierten Signale als auch für den Suchalgorithmus gewählt werden. Die aus den für die Signalerzeugung gewählten Parametern resultierenden Verteilungen für die Größen und Abstände der simulierten Cluster zeigt Abbildung 4.6.

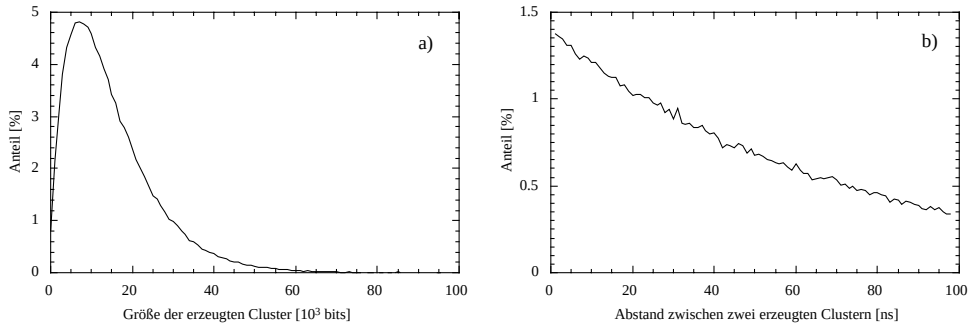



Abbildung 4.6: Die Verteilungen der generierten Clustergrößen und -abstände

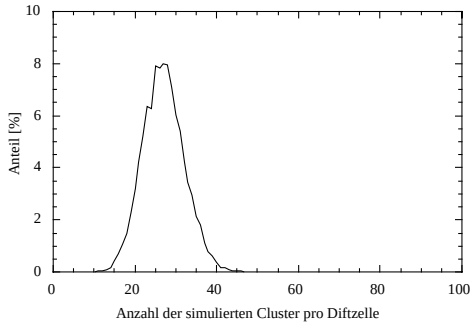



Abbildung 4.7: Die Verteilung der Anzahl der generierten Cluster pro Driftzelle

Das elektronische Rauschen wurde durch ein gaußverteiltes weißes Rauschen angenähert, wobei das Quadrat der Standardabweichung der Rauschamplitude auf $\sigma^2 = 6 \cdot 10^6$ ($\sigma \approx 2500$) festgelegt wurde.
Für die Clustererkennung wurden im Kurvenanpassungsalgorithmus folgende Parameter eingestellt:

- Vorlauf (flacher Signalanteil der Fitkurve zur Bestimmung der Grundlinie): 15 ns
- Zeit vom Beginn des ersten Signals bis zum Beginn des zweiten Signals : 8 ns
- Zeit vom Beginn des zweiten Signals bis zum Ende der Referenzkurve : 7 ns
- $\chi^2$ des Clusterfits $< 200$
- Größe $\alpha$ des gefitteten Clusters $> 3000$ Bit
- Lokales Minimum in $\chi^2$: $\chi^2_{i-2} > \chi^2_{i-1} > \chi^2_i < \chi^2_{i+1} < \chi^2_{i+2}$

Diese Parameter haben sich in vorangegangenen Test als akzeptabler Kompromiß zwischen Rechenzeitbedarf, Erkennungswahrscheinlichkeit für tatsächlich vorhandene Clustersignale und Unterdrückung falscher Signale erwiesen.

### 4.4.3 Ergebnisse der Simulation

Mit diesen Parametern wurde eine Simulation des Clustersuchalgorithmus durchgeführt.
Ein Cluster galt in diesem Fall als 'gefunden', wenn unter Anwendung der oben aufgeführten Parameter ein Clustersignal in den simulierten Daten vorzuliegen schien. Ein in der Simulation erzeugtes Clustersignal galt als 'nicht gefunden', wenn sich innerhalb eines Intervalls von $\pm$ 3 Samples ($\hat{=} \pm$ 3 ns) um die Stelle, an der es erzeugt worden war, mit dem Suchalgorithmus kein Clustersignal finden ließ. Ein 'fehlerhaft gefundenes' Cluster schließlich bezeichnet ein Clustersignal, das außerhalb dieses Intervalles um eines der erzeugten Signale herum gefunden wurde.
Unter diesen Voraussetzungen wurde ein Anteil von etwa 87 % der erzeugten Cluster gefunden. Dieser Anteil setzt sich zusammen aus etwa 83 % 'echter' Fundstellen, an denen auch im entsprechenden Intervall ein Clustersignal erzeugt worden war, sowie etwa 4 % 'falscher' Fundstellen, an denen Clustersignale gefunden wurden, obwohl an der entsprechenden Stelle kein Signal erzeugt worden war. An etwa 17 % der Stellen, an denen ein Signal erzeugt worden war, wurde vom Suchalgorithmus kein Clustersignal gefunden.

| | | |
|---|---|---|
| Anzahl der erzeugten Cluster : | 268 846 | |
| Gesamtzahl der gefundenen Cluster : | 233 118 | (86.7 %) |
| Zahl der fehlerhaft gefundenen Cluster : | 9 986 | (3.7 %) |
| Anzahl der nicht gefundenen Cluster : | 45 714 | (17 %) |

Tabelle 4.1: Globale Ergebnisse des Suchalgorithmus mit simulierten Daten

Die folgenden zwei Diagramme zeigen die Effizienz des Algorithmus in Abhängigkeit von der Größe der erzeugten Clustersignale sowie deren Abstand. Die Effizienz ist hier definiert als die Zahl der an der richtigen Stelle gefundenen Cluster dividiert durch die Zahl der mit den entsprechenden Parametern erzeugten Cluster.
Abbildung 4.8 zeigt die Effizienz des Kurvenanpassungsalgorithmus in Abhängigkeit vom Abstand der Clustersignale. Für alle Clustergrößen entsprechend der Verteilung in Abbildung 4.6.a wird eine maximale Effizienz von ca. 90 % erreicht. Beschränkt man sich auf erzeugte Cluster, die größer sind als die dreifache Standardabweichung der Verteilung des Rauschens, erreicht die Effizienz für ausreichend große Abstände zwischen den Clustern Werte um 99 %. Die entsprechenden Fehler bewegen sich im Bereich um 0.7 % für alle Cluster und 0.3 % für Cluster $> 3 \cdot \sigma_{\mathrm{Rauschen}}$.
Bei Abständen zwischen zwei aufeinanderfolgenden Clustern von weniger als 7 ns beginnt die Effizienz abzufallen. Dieses Verhalten ist zu erwarten, da der Algorithmus in der gegenwärtigen Form nicht in der Lage ist, zwei Cluster sicher voneinander zu trennen, wenn einer der Cluster auf der steigenden Flanke des anderen beginnt. Durch eine Verringerung des Abstands zwischen den beiden Anpassungskurven läßt sich der Knickpunkt der Effizienzkurve zwar weiter zu kleineren Zeiten verschieben, gleichzeitig steigt aber die Zahl der falsch erkannten Cluster.
Die Kurven in Abbildung 4.9 zeigen die Effizienz des Suchalgorithmus in Abhängigkeit von der Clustergröße für drei verschiedene minimale Clusterabstände. Für alle Clusterabstände wird für

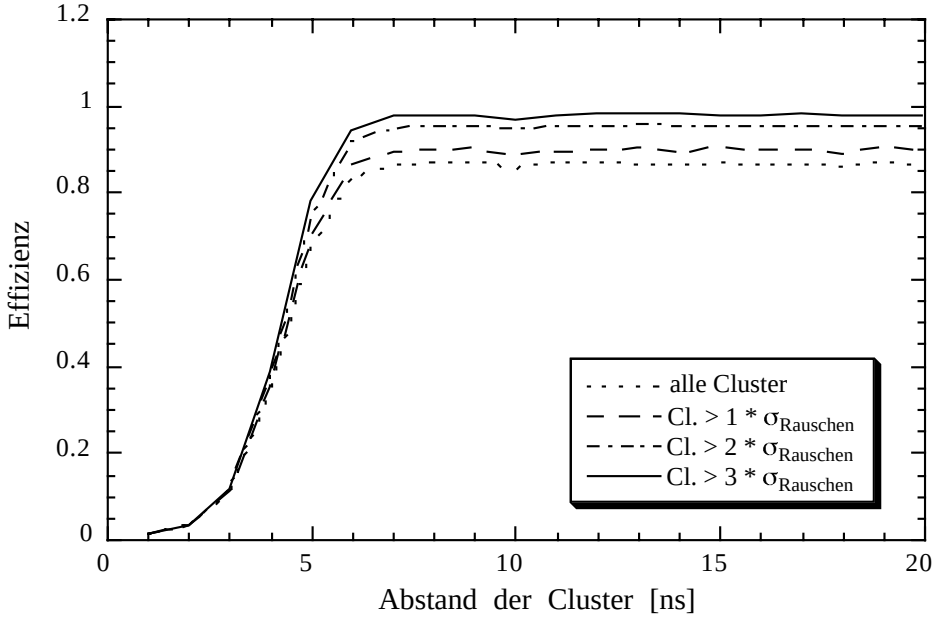



Abbildung 4.8: Effizienz des Kurvenanpassungsalgorithmus in Abhängigkeit vom Abstand der Cluster

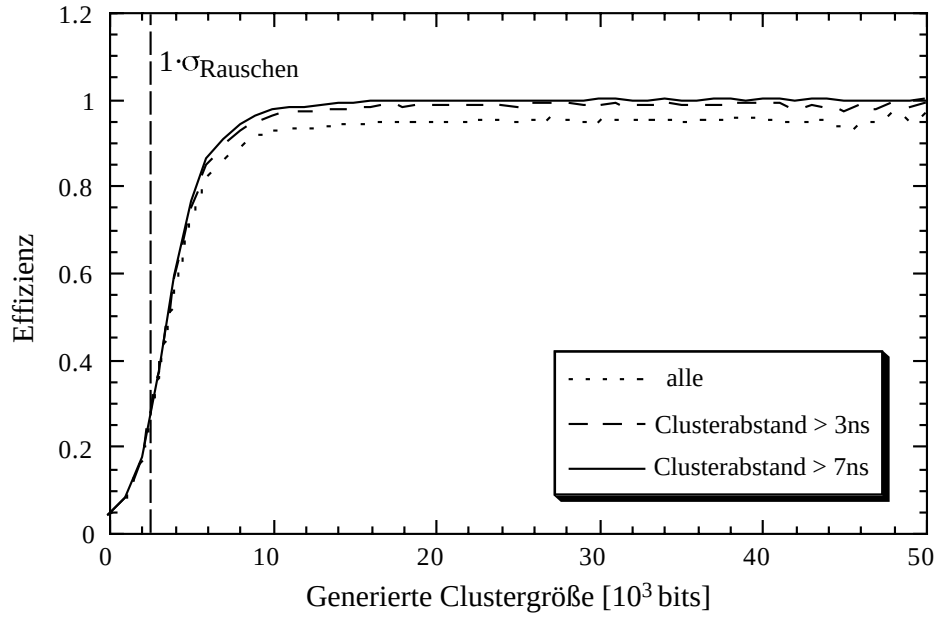



Abbildung 4.9: Effizienz des Kurvenanpassungsalgorithmus in Abhängigkeit von der Größe der Cluster

große Cluster eine Effizienz von etwa 95 % erreicht. Beschränkt man sich auf Clusterabstände von mehr als 7 ns steigt die Effizienz auf über 99%. Die Fehler auf diese Kurven liegen unter 1 % für alle Clusterabstände und unter 0.3 % für Clusterabstände größer als 7 ns. Wie zu erwarten fällt die Effizienz der Clustererkennung stark ab, sobald die Clustergröße in den Bereich des Rauschens ( $1\sigma \approx 2500$ Bit) gelangt.

### 4.4.4 Anwendung des Algorithmus

Zur Illustration der Anwendbarkeit des Algorithmus auf echte Daten wurde abschließend ein Teil eines Datensatzes der während des Testlaufs 1995 am CERN gewonnenen Meßwerte damit untersucht.
Bei diesem Datensatz handelt es sich um eine Mischung verschiedener Teilchen mit einheitlichem Impuls, die für die folgenden Untersuchungen nicht weiter unterschieden wurden. Eine Trennung dieses Datensatzes nach unterschiedlichen Teilchensorten wäre anhand der verlangten Triggerbedingungen möglich, eine qualitative Aussage über den Algorithmus läßt sich aber auch in diesem Falle nicht treffen. Hierzu wäre eine detaillierte Simulation aller Prozesse von der Ionisation der Gasmoleküle im Inneren der Driftkammer bis – in eingeschränkterem Umfang – zur Digitalisierung der Signale notwendig, um eine Grundlage für den Vergleich zwischen der Erwartung und dem Experiment zur Verfügung zu haben. Eine solche detaillierte Simulation, die am Ende auch die Form der beobachteten Signale erklären sollte, ist anderweitig in Arbeit[30].
Das Ziel der folgenden Abbildungen ist es, zu zeigen, daß nicht nur in der Simulation, sondern auch in echten Datensätzen Cluster gefunden werden können. Bei der Betrachtung einzelner Kurven, zusammen mit einer Darstellung der Positionen und Größen der gefundenen Cluster, zeigt sich für das bloße Auge eine gute Übereinstimmung. Eine statistische Auswertung der Untersuchung eines Teiles dieses Datensatzes ergibt die folgenden Ergebnisse:

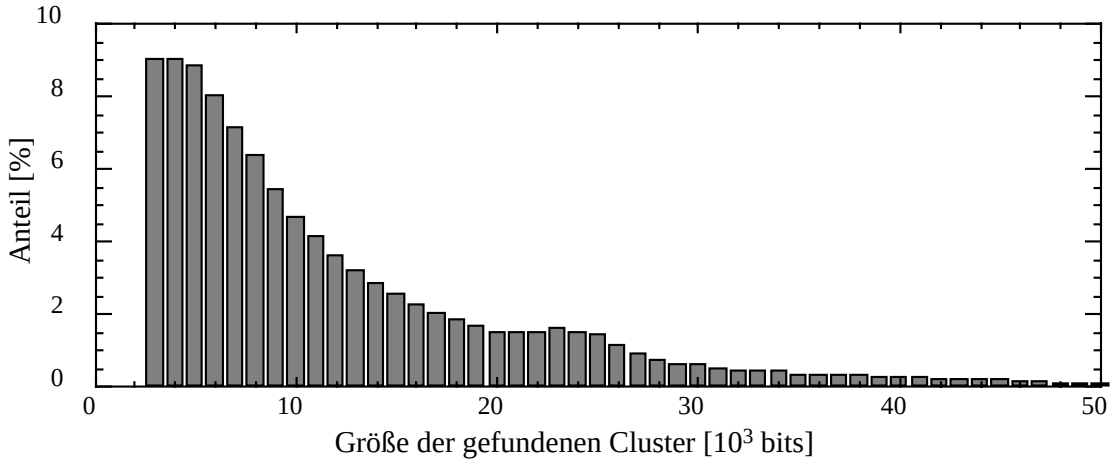



Abbildung 4.10: Größenverteilung der gefundenen Cluster

Die gezeigten Verteilungen in den Abbildungen 4.10 bis 4.12 ergeben sich aus der Untersuchung von 9295 Ereignissen. In diesen Ereignissen wurden unter Anwendung der Kriterien in Abschnitt 4.4.2 insgesamt 171188 Clustersignale gefunden. Zur Unterdrückung kleiner, dicht beieinanderliegender Cluster wurden in einem zusätzlichen Schnitt alle Signale, die kleiner als 2 $\sigma_{\mathrm{Rauschen}}$ waren und einen geringeren Abstand als 4 ns zum vorangegangenen Cluster aufwiesen, aus dieser Auswahl entfernt, wobei 170088 Cluster zurückblieben.
Die Größenverteilung der so gefundenen Signale in Abbildung 4.10 folgt der groben Erwartung, daß mehr kleine als große Clustersignale gefunden werden sollten. Nicht erwartet wurde jedoch der große Anteil an kleinen und kleinsten Signalen, was darauf hindeutet, daß entweder ein signifikanter Anteil der gefundenen Signale aus ‘falschen’ Rauschsignalen besteht, oder aber, daß das Rauschen in diesem Testaufbau zu hoch war, um die kleineren Clustersignale noch erkennen zu

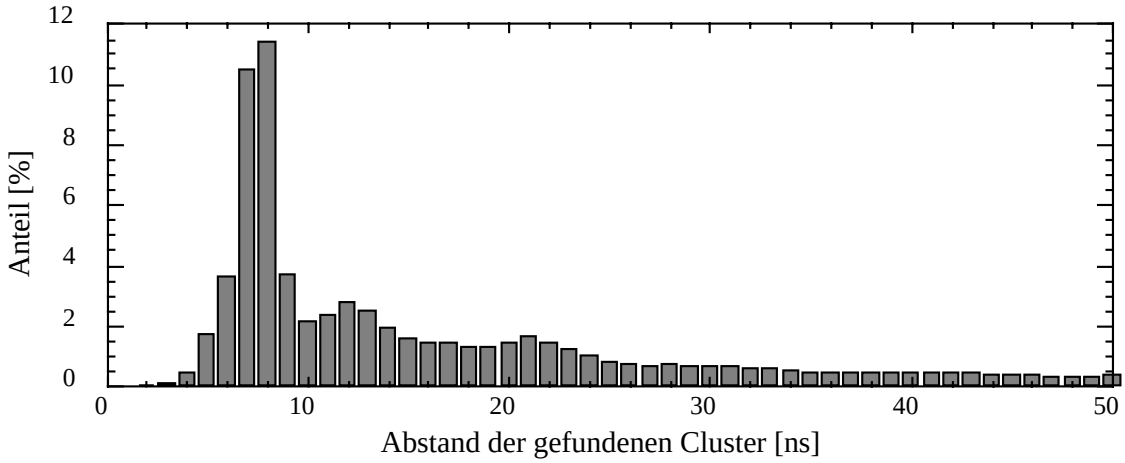



Abbildung 4.11: Verteilung der Abstände der gefundenen Cluster

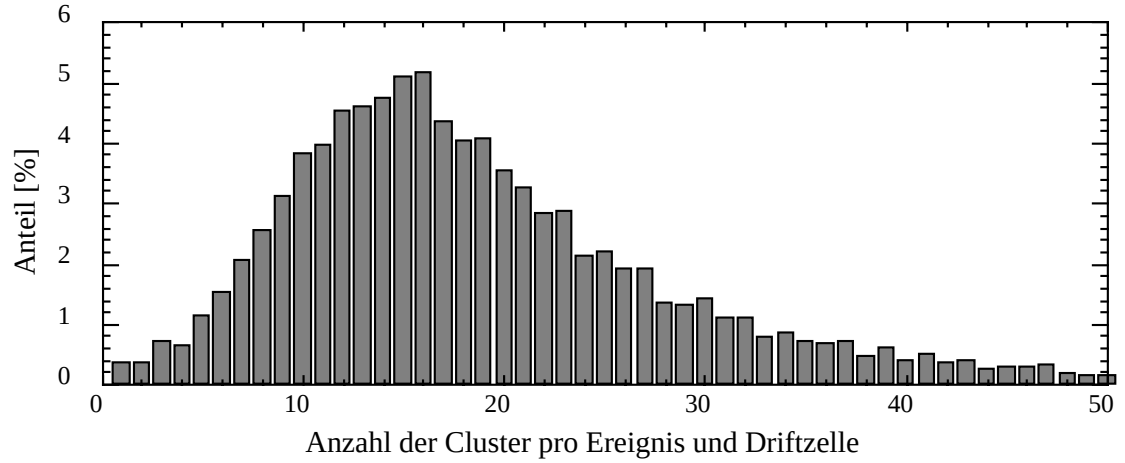



Abbildung 4.12: Verteilung der Anzahl der Cluster pro Ereignis und Driftzelle

können.

Die Verteilung der Abstände der gefundenen Cluster in Abbildung 4.11 zeigt ebenfalls ein starkes Maximum bei kleinen Clusterabständen, was wieder ein Anzeichen für Probleme mit dem Rauschen der Apparatur sein könnte, sowie zwei weitere schwache Maxima bei größeren Clusterabständen.

Die Verteilung der Anzahl der Cluster pro Driftzelle in Abbildung 4.12 zeigt zwar grob die erwartete Form einer Poisson-Verteilung, aber sowohl die Breite dieser Verteilung als auch der Mittelwert entsprechen nicht den Erwartungen der Abschätzungen. Für den Mittelwert der Verteilung würde man bei den gegebenen Bedingungen (Zellengröße $3 \times 3$ cm$^2$, Helium-Methan-Gasmischung 80:20, senkrechter Strahleinfall) etwa 35 Cluster pro Ereignis und Driftzelle erwarten [30], also grob das Doppelte des hier gefundenen Wertes. Die gefundene Breite der Verteilung läßt sich durch die Überlagerung der Verteilungen mehrere unterschiedlicher Teilchensorten sowie noch fehlende Korrekturen bei der Auswertung im Bezug auf die Effizienz der Clustersuche in Abhängigkeit vom Abstand der Teilchenspur vom Anodendraht erklären.

Abschließend läßt sich sagen, daß der entwickelte Algorithmus eine sehr gute Clustererkennung ermöglicht, so lange sich die Clustersignale ausreichend stark vom Rauschen abheben, wie die Ergebnisse der Simulation zeigen (Abbildungen 4.8 und 4.9). Unter diesen Umständen sind die schlechten Ergebnisse bei der Untersuchung der gemessenen Signale auf die schlechte Qualität der Rohdaten zurückzuführen. Als weitere Schritte hin zur praktischen Anwendung von Cluster Counting als Meßverfahren und zum Verständnis dieser Verteilungen sind daher auf der praktischen Seite eine Optimierung des Versuchsaufbaus zur Erzielung eines besseren Signal-Rausch-Verhältnisses, auf der theoretischen Seite eine qualitativ aussagefähige Simulation der zu den beobachteten Signalen führenden Prozesse notwendig. In beiden Richtungen wird an den beteiligten Instituten gearbeitet.

### 4.4.5 Rechenzeiten

Beim Einsatz dieses Algorithmus zur online-Datenreduktion in einem Experiment ist auch die Ausführungszeit von Bedeutung. Um die Ausführungszeiten vergleichen zu können, wurde der Algorithmus mit 5 unterschiedlichen Datensätzen (Meßwerte aus dem CERN-Testlauf) auf drei Rechnersystemen getestet:

- Cluster Counting Präprozessor (CCPP), der im ersten Teil dieser Arbeit vorgestellt wurde. Die Laufzeit auf diesem Rechner wurde dadurch gemessen, daß innerhalb des Programms vor Beginn ein externes Register gesetzt und nach Ablauf der Clustersuche wieder rückgesetzt wurde. Der Zustand dieses Registers wurde über ein Oszilloskop (Tektronix TDS 644) beobachtet.

- PowerMacintosh 7100/80 (PM 7100/80): dieser Rechner ist mit einem mit 80 MHz getakteten PowerPC 601 RISC-Prozessor ausgerüstet, welcher auch eine Speicherverwaltungseinheit und eine Fließkommaeinheit beinhaltet. Zusätzlich besitzt er ausreichend externen Cache-Speicher (256 kB), so daß das Clustersuchprogramm vollständig in diesem ablaufen kann.

- PowerBook 165c (PB 165c): dieser etwas ältere Rechner besitzt einen mit 30 MHz getakteten Motorola 68030 Prozessor mit externer Fließkommaeinheit (Motorola 68882) und keinen Cache.

Zu dieser Tabelle ist zunächst zu sagen, daß das PowerBook nur zum Vergleich mit aufgenommen wurde. Die in diesem Rechner verwendete Technologie ist seit mehreren Jahren veraltet, was in den Meßergebnissen auch deutlich zu sehen ist. Trotzdem ist dieser Vergleich interessant, da auch zu sehen ist, daß der Präprozessor bei etwa derselben Taktfrequenz fünfzehnmal so schnell arbeitet.
Im Vergleich mit einem etwas aktuelleren Bürorechner, dem PowerMacintosh 7100/80, schneidet der Präprozessor schon deutlich schlechter ab. Zu berücksichtigen ist hier aber, daß der Präprozessor noch einige Fehler aufweist (siehe Abschnitt 3.4). Nach Beseitigung dieser Fehler sollten die Verarbeitungszeiten bei Präprozessor und PowerMacintosh etwa gleich sein.
Wie anfangs schon erwähnt trägt aber die gesamte Rechnerarchitektur zum Datendurchsatz bei und nicht nur die reine Prozessorleistung. Hier ist für den Präprozessor etwa folgendes zu erwarten:

- Dateneingang: Die vorgesehene Dateneingabe über den schnellen ECL-Bus wird mit einer Taktfrequenz von 40 MHz arbeiten. Bei einer Busbreite von 32 Bit kann ein Wort pro Takt

| | CCPP | PM 7100/80 | PB 165c |
|---|---|---|---|
| Ereignis a (7 Cl.) | 206 ms | 31.9 ms | 2883 ms |
| Ereignis b (17 Cl.) | 220 ms | 35.6 ms | 3053 ms |
| Ereignis c (19 Cl.) | 232 ms | 39.1 ms | 3189 ms |
| Ereignis d (26 Cl.) | 238 ms | 40.6 ms | 3275 ms |
| Ereignis e (29 Cl.) | 249 ms | 44.3 ms | 3413 ms |
| linearer Fit | 192 ms + 1.9 ms/Cl. | 27.8 ms + 0.5 ms/Cl. | 2707 ms + 23.2 ms/Cl. |

Tabelle 4.2: Ausführungszeiten des Clustersuchprogrammes auf unterschiedlichen Rechnern

übertragen werden. Für die Übertragung von 2048 Worten beträgt die reine Transferzeit damit etwa 51 $\mu$s. Hinzu kommen die notwendigen Zyklen zum Aufsetzen und Beenden des Transfers. Wenn diese mit jeweils 200 CPU-Zyklen des Master-Prozessors abgearbeitet werden können, kommen hierfür nochmals etwa 20 $\mu$s hinzu.

- Datentransfer: Der Datentransfer vom Eingangspuffer zu einem der Slaves sollte ebenfalls mit einer Transferrate von einem Wort pro Takt bei 40 MHz Taktfrequenz ablaufen. Um auf der sicheren Seite zu liegen, sei hier nun eine Transferzeit von zwei Takten, also 50 ns bei 40 MHz, pro Wort angenommen. Damit benötigt dieser Transfer für 2048 Worte etwa 102 $\mu$s. Für die Zeit zum Aufsetzen und Beenden des Transfers seien wieder 20 $\mu$s angenommen. Abgesehen von der Zeit zum Aufsetzen und Beenden des Transfers, für die der Master in Anspruch genommen wird, läuft dieser Vorgang vollständig unabhängig von den folgenden Vorgängen ab. Für die Verarbeitungspipeline bedeutet dies, daß die Transferzeit nur bei der Füllung des Präprozessors mit Daten berücksichtigt werden muß, bei der kontinuierlichen Datenverarbeitung nicht mehr.

- Verarbeitung: Aus der obigen Tabelle läßt sich im gegenwärtigen Zustand eine mittlere Verarbeitungszeit von etwa 240 ms für einen Datensatz von 2048 Worten abschätzen. Nach Korrektur der Fehler und Optimierung der Taktimpulsverteilung sollte es möglich sein, diese Verarbeitungszeit auf ein Viertel, also etwa 60 ms pro Datensatz, zu reduzieren.

- Datenausgabe: Nach der Clustersuche beträgt der Umfang der im gemeinsamen Speicher abzulegenden Daten nur noch wenige Worte. Den größten Teil an der Transferzeit dieser Worte wird die Busarbitrierung und wieder die Haushaltsführung einnehmen. Daher sei hier diese Zeit wieder auf 20 $\mu$s abgeschätzt.

Unter diesen Umständen beträgt die eigentliche Verarbeitungszeit ein Vielfaches der notwendigen Zeit für den Datentransfer. Es kann also davon ausgegangen werden, daß sich die Slaves beim Datentransfer nicht gegenseitig behindern, und somit die einzelnen Slaves praktisch unabhängig voneinander betrachtet werden können. Des weiteren ist der Anteil der Transferzeit an der gesamten Datenverarbeitung so klein, daß im Rahmen der hier zu erwartenden Fehler die notwendige Gesamtzeit sehr gut durch die Verarbeitungszeit innerhalb eines der Slaves angenähert werden kann.

Bei einer Verarbeitungszeit von etwa 60 ms pro Datensatz werden nun aber bei einem Hochraten-Experiment wie KLOE mit einer mittleren Datenrate von 100 Hz pro Kanal etwa 6 Prozessoren pro Kanal benötigt. Unter diesen Voraussetzungen ist der beschriebene Ansatz, sowohl vom finanziellen Aufwand her als auch von der Praxis her, nicht tragbar. Für Experimente mit geringeren Datenraten pro Kanal dagegen könnte diese Art der Auslese mit der beschriebenen Elektronik ein möglicher Weg zur Verbesserung der Teilchenidentifikation sein.
Um die Technik des Cluster Counting in einem Hochraten-Experiment sinnvoll einsetzen zu können, müssen die Kosten pro Kanal der Ausleseelektronik gesenkt werden, gleichzeitig sollte die Leistungsfähigkeit einer Präprozessorplatine gesteigert werden. Mit den Entwicklungen im Bereich der benutzerprogrammierbaren Baulemente innerhalb der letzten Jahre sollte eine solche Leistungssteigerung bei gleichzeitiger Kostensenkung möglich sein.

# Kapitel 5

# Ausblick - zukünftige Möglichkeiten

Während der Durchführung der vorliegenden Arbeit hat sich die Elektronik stark weiterentwickelt. Insbesondere im Bereich der Integrationsdichte sowie der programmierbaren Bauelemente hat es starke Fortschritte gegeben. Als Beispiel für die gesteigerte Integrationsdichte sei hier der Digitale Signalprozessor TMS320C80 von Texas Instruments genannt [61, 62]. Dieser Prozessor beinhaltet auf einem Chip vier DSP-Kerne, einen RISC Master Prozessor mit Fließkommaeinheit, 50 kB gemeinsames RAM, einen Kreuzschienenschalter, um die Zugriffsmöglichkeiten auf dieses RAM zwischen den Prozessorkernen aufzuteilen, sowie zusätzliche Steuerlogik für externe Zugriffe. Dieser Baustein ist als Ganzes in der Lage, bis zu 2 Milliarden Operationen pro Sekunde abzuarbeiten. Dabei enthält dieser Digitale Signalprozessor alle wesentlichen Bestandteile einer vollständigen Platine eines Cluster Counting Präprozessors mit vier Slaves.
Im Bereich der benutzerspezifischen Bauelemente hat diese höhere Integrationsdichte zur Folge, daß immer komplexere Schaltungen auf einem einzelnen Chip möglich sind. Speziell im Bereich der FPGAs[1] führt dies dazu, daß diese Bauteile mittlerweile als problemangepaßt konfigurierbare 'Rechenknechte' benutzt werden können.

## 5.1 FPGAs als problemangepaßt konfigurierbare Prozessoren

Ein FPGA enthält zunächst einmal eine gewisse Anzahl konfigurierbarer logischer Elemente (Logikzellen, 'logic cells'), die über interne Leitungen miteinander verbunden sind. Spezielle Zellen dienen als Treiber oder Empfänger für die Signale, die zu (oder von) den äußeren Anschlüssen des Bauteils laufen. Zunächst einmal waren diese Bauelemente als schnelle, benutzerkonfigurierbare Logikbausteine gedacht, um konventionelle logische Schaltungen auf einer Platine zu ersetzen. Die hohe Zahl an Logikzellen innerhalb moderner FPGAs erlaubt nun aber auch die Realisierung komplexer logischer Schaltungen bis hin zu den grundlegenden, für die digitale Signalverarbeitung notwendigen Funktionen [63, 64, 65, 66, 67, 68, 69].
Eine der Besonderheiten bei der Verwendung von FPGAs als Prozessorelemente ist, daß mittlerweile viele der am Markt erhältlichen FPGA-Familien es erlauben, die logische Konfiguration in eingebauten Zustand durch einfache Neuprogrammierung zu ändern ('in-circuit-programming', Programmierung innerhalb der Schaltung). Mit dieser Möglichkeit läßt sich ein solches Bauelement in Abhängigkeit vom gestellten Problem zu einem speziellen, problemangepaßten Prozessor konfigurieren [70, 71].

[1] **F**ield **P**rogrammable **G**ate **A**rray, etwa übersetzbar als 'programmierbare Anordnung von (logischen) Gattern'

Obwohl die Zahl der innerhalb eines FPGAs derzeit (1996) zur Verfügung stehenden logischen Gatter (bis zu 100000 Gatter, oder etwa 5000 Logikzellen[2] und etwa 5000 Registern mit etwa 500 externen Anschlüssen [72]) nicht an die mögliche Komplexität einer benutzerspezifischen integrierten Schaltung ('ASIC'[3]) oder eines modernen Hochleistungsprozessors heranreicht, hat ein solches FPGA gegenüber diesen einen Vorteil: seine interne Struktur läßt sich unter der Steuerung eines Kontrollrechners zur Anpassung an einen gegebenen Algorithmus jederzeit verändern. Für einzelne Operationen ist dies durch den Zeitaufwand kaum rentabel, für die Anpassung der Hardware an einen vorgegebenen Algorithmus bietet sich diese Möglichkeit jedoch an, und wird auch bereits in der Praxis eingesetzt [69, 70].
Mit der durch die steigende Integrationsdichte zu erwartenden Vergrößerung der Zahl der innerhalb eines FPGAs zur Verfügung stehenden Strukturen sollte es dann auch möglich sein, vollständige Programme nicht mehr als Steueroperationen innerhalb eines Vielzweckprozessors zu implementieren, sondern als Struktur eines FPGAs, durch das die Daten hindurchgeschoben werden [45, 71]. Mit einer solchen Pipeline lassen sich die höchsten Datendurchsatzraten realisieren, da – bei einem geeigneten Algorithmus – nach dem Füllen der Pipeline mit jedem Prozessorzyklus ein neues Ergebnis geliefert werden kann. Ungeeignet sind in diesem Falle Algorithmen, die bei einer großen Länge der Pipeline eine häufige Neufüllung verlangen.

## 5.2 Der Algorithmus zur Kurvenanpassung als Pipeline

Unter der Voraussetzung, daß entsprechend komplexe FPGAs erhältlich sein werden, läßt sich der Kurvenanpassungsalgorithmus zur Clustersuche als Pipeline innerhalb eines (oder auch mehrerer) solchen Bausteins implementieren. Eine Implementierung in einem ASIC ist in diesem Fall problematisch, da einige der Parameter des Algorithmus, die die Struktur der Pipeline festlegen, anhand der Daten optimiert werden sollten. Wird die Struktur in einem ASIC festgelegt, ist diese Optimierung kaum noch möglich.
Abbildung 5.1 zeigt einen Vorschlag für eine Realisierung des Kurvenanpassungsalgorithmus in einer parallelen Pipeline. Damit das Diagramm noch in lesbarer Größe auf einer Seite darstellbar ist, wird hier im Unterschied zu Abschnitt 4.4.2 nur ein 12 Samples langer Ausschnitt der Meßwerte betrachtet, mit jeweils 4 Samples für das erste und das zweite Clustersignal. Bezogen auf die Clustersuche im vorangegangenen Kapitel entspräche dies etwa einer Halbierung der Abtastrate.
Die Meßwerte laufen von links ein und werden auf vier Stränge verteilt: einen zunächst einfach durchzuschiebenden Strang mit den unbearbeiteten Daten sowie drei Stränge, in denen die Summen der Produkte der Meßwerte mit den Koeffizienten der Referenzkurve gebildet werden. Die Ergebnisse dieser Summationen werden darauf miteinander verknüpft, um zunächst die Werte $a \ldots i$ (siehe Gleichungen 4.18 bis 4.26) zu erhalten, aus denen dann die drei Koeffizienten $\alpha$, $\beta$ und $\gamma$ (Gleichungen 4.15 bis 4.17) resultieren.
Mittels dieser Koeffizienten wird nun die Referenzkurve skaliert und von den Meßwerten subtrahiert, worauf schließlich $\chi^2$ als Maß für die Abweichung der Meßwerte von der skalierten Referenzkurve berechnet wird. In diesem Beispiel werden dieselben Bedingungen für das Erkennen eines Clustersignals zugrundegelegt wie in Abschnitt 4.3.1. Damit wird das wesentliche Kriteri-

[2] Eine 'Logikzelle' bezeichnet die programmierbare Einheit innerhalb eines FPGAs. Im Falle der Bauelemente der '10k'-Serie des Herstellers Altera besteht eine solche Logikzelle aus einer Tabelle, mit der sich jede beliebige Funktion von 4 Eingangsvariablen bilden läßt, einem Register und einer Anzahl von Steuerelementen zur Datenverteilung.

[3] **A**pplication **S**pecific **I**ntegrated **C**ircuit, eine integrierte Schaltung, die speziell für einen bestimmten Zweck (oder Benutzer) angefertigt wird.

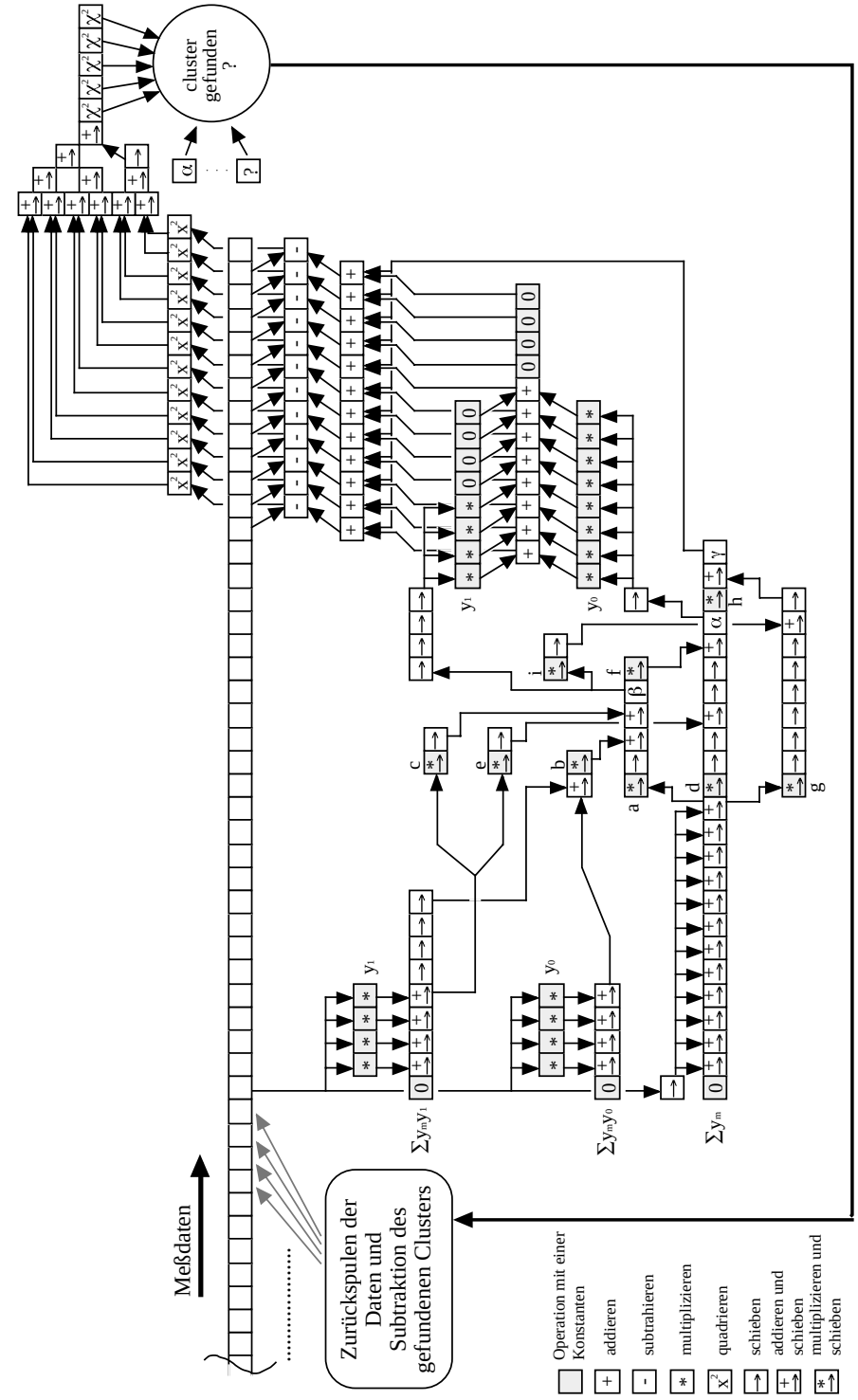



Abbildung 5.1: Ein Vorschlag für eine Realisierung des Clustersuchalgorithmus in einer parallelen Pipeline

um für ein Clustersignal die Anwesenheit eines lokalen Minimums in $\chi^2$. Darüberhinaus können weitere Kriterien in die Entscheidung einfließen.
Kommt der Entscheidungsprozeß zu dem Schluß, daß an dieser Stelle in den Meßwerten ein Clustersignal vorliegt, muß dieses vor der weiteren Verarbeitung von den Meßwerten subtrahiert werden. Diese Subtraktion, zusammen mit der anschließenden Suche nach weiteren Clustern in diesem Bereich, bedingt eine Neufüllung der Pipeline. Im ungünstigsten Fall, wenn während der Subtraktionsphase ein weiterer Cluster gefunden wird, ist zumindest ein weiterer Zwischenspeicher, oder entsprechende Verzweigungsstrukturen, notwendig.
Die Subtraktion an sich kann zum Beispiel mittels eines Puffers vorgenommen werden, in dem die Werte der 'typischen' Kurve eines Clustersignals abgespeichert sind. Diese Werte werden in einer kurzen Pipeline mit dem Wert $\alpha$ des gefundenen Clusters multipliziert und anschließend von den Meßwerten subtrahiert. Diese Subtraktion kann dann abgebrochen werden, wenn das Produkt aus der typischen Kurve und $\alpha$ klein genug geworden ist. Wie klein 'klein' hier ist, hängt von den Parametern des Suchalgorithmus und dem Rauschen der Meßwerte ab.

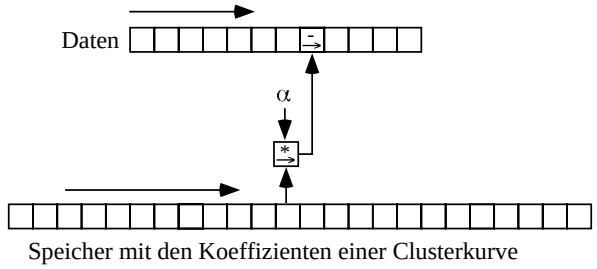



Abbildung 5.2: Eine mögliche Technik für die Subtraktion gefundener Cluster

Mit den Längen für die Abschnitte der Referenzkurve aus Abbildung 5.1, das heißt insgesamt 12 Punkten für den Vorlauf, das erste und das zweite Clustersignal, werden etwa 50 Takte für die Füllung der Pipeline benötigt. Diese 50 Takte teilen sich auf in 12 Takte für die Summationen der Produkte aus Referenzkurve und Meßwerten, etwa 25 Takte für die Berechnung der Koeffizienten und wieder etwa 12 Takte zur Berechnung der quadratischen Abweichung $\chi^2$. Unter ungünstigen Bedingungen (alle Clustersignale überlappen maximal) muß, neben diesem Füllvorgang, für jedes gefundene Clustersignal noch ein Subtraktionszyklus von etwa 100 Takten eingefügt werden. Sowohl der Füllvorgang als auch der Subtraktionsprozeß können hier nicht als Elemente der Pipeline ablaufen. Damit ergibt sich, bei einer Ereignislänge von 2048 Samples und im Mittel 15 Clustersignalen pro Ereignis, ein Bedarf von etwa 50 (erste Füllung) + 15 · (100 + 50) (Subtraktionen und Wiederbefüllung) + 2048 (Anzahl der Samples) ≈ 4300 Takten für die Verarbeitung der Meßwerte eines Ereignisses.
Hierfür werden 71 Schieberegister sowie insgesamt 120 Operationsmodule, die die Verknüpfung zweier Zahlen mit einem Schieberegister zur Zwischenspeicherung verbinden, benötigt. In allen diesen Modulen muß die volle Datenbreite der Meßwerte, also 16 Bit, zur Verfügung stehen. Unter der Annahme, daß die Ressourcen zur Verbindung der Elemente im Inneren der Logikbausteine zur Verfügung stehen, bräuchte man zur Durchführung dieser Operationen mindestens eine Logikzelle für jedes Bit jeder Operation. Damit ergibt sich für dieses Beispiel ein Gesamtbedarf von (71 + 120) · 16 = 3056 Logikzellen. Unter diesen Voraussetzungen ließe sich der Algorithmus in einem der derzeit erhältlichen FPGAs implementieren.
Dieser Aufbau hat jedoch noch einen Nachteil. Sollen alle Operationen mit einer Logikzelle pro

Bit aufgebaut werden, müssen die einzelnen Bits einer Zahl in einer Kaskade nacheinander abgearbeitet werden, wodurch die Ausführungszeit entsprechend groß wird. Als Alternative bietet es sich an, entsprechend optimierte Elemente aus der Bibliothek des Bauelementeherstellers zu verwenden [73]. Hier werden für einen 16-Bit-Addierer mit einer Zykluszeit von 11 ns 18 Logikzellen benötigt, für einen 16-Bit-Multiplizierer mit einer Zykluszeit von 46 ns 535 Logikzellen. Durch den Einsatz weiterer Logikzellen läßt sich der Multiplizierer auch als Pipeline aufbauen, wodurch bei einem Einsatz von 552 Logikzellen und einer Latenzzeit von vier Takten die Zykluszeit auf 14 ns verringert werden kann.
Für die 79 Additionen, 41 Multiplikationen und 71 Schiebeoperationen des Beispiels ergibt sich somit ein Bedarf von etwa 25000 Logikzellen[4] benötigt. Das wiederum bedeutet, daß sich diese Version des Algorithmus – rein rechnerisch – bei einer Taktfrequenz von 70 MHz in 5 der größten der derzeit erhältlichen FPGAs unterbringen ließe. Die gegenwärtige Entwicklung auf diesem Sektor läßt erwarten, daß es innerhalb eines Jahres möglich sein wird, die Zahl der notwendigen Bausteine auf 1 zu reduzieren.
Für die Auswertung der simulierten Daten und der Meßwerte in Kapitel 4 wurde nun allerdings nicht von 12, sondern von 30 Punkten für die Referenzkurve ausgegangen. Durch eine entsprechende Skalierung ergibt sich somit eine Füllzeit der Pipeline von (30 + 25 + 30) = 85 Takten. Die notwendige Zeit für die Verarbeitung eines Datensatzes mit 2048 Meßwerten steigt nur unwesentlich auf etwa 5000 Takte an. Der Anstieg in der Zahl der nötigen Operationen ist wesentlich ausgeprägter, hier werden jetzt etwa 70 Multiplikationen, 160 Additionen und 120 reine Schieberegister benötigt. Diese lassen sich, unter Verwendung der oben erwähnten Bibliotheksfunktionen, in insgesamt etwa 43000 Logikzellen unterbringen, was wiederum etwa 10 der größten der derzeit erhältlichen FPGAs entspricht.
Es steht also für die Ausführung dieses Algorithmus in einer Hardware-Pipeline ein ganzer Bereich möglicher Implementationen zur Verfügung, von einer kleinen Version, die sich innerhalb eines der derzeit erhältlichen FPGAs programmieren läßt, aber eine große Ausführungszeit benötigt, bis hin zu einer Version, die zwar bei einer Taktrate von 70 MHz ein Ergebnis pro Takt liefern kann, für deren Programmierung jedoch bei der gegenwärtigen Komplexität der verfügbaren FPGAs 5...10 der größten Bausteine benötigt würden.
Der Anstieg der Füllzeit der Pipeline durch die Unterteilung der Multiplikationen in kleinere Unterstrukturen hält sich in Grenzen, da der größte Teil der Multiplikationen unabhängig voneinander parallel ablaufen kann. Somit ergibt sich, für 5000 Takte und eine Taktrate von 70 MHz, eine Verarbeitungszeit von etwas mehr als 70 $\mu$s.
Vergleicht man diese Zahl mit den Werten aus Abschnitt 4.4.5, so ist zu sehen, daß eine solche spezialisierte Pipeline wesentlich schneller arbeiten kann als alle der aufgeführten Mikroprozessoren. Ein solcher spezialisierter Prozessor könnte, selbst in einem Hochraten-Experiment wie KLOE, etwa 100 Detektorkanäle verarbeiten.

[4] $71 \cdot 16 + 79 \cdot 18 + 41 \cdot 552 = 25190$

# Kapitel 6

# Zusammenfassung

Zur Durchführung einer Präzisionsmessung zur CP-Verletzung im Zerfall des neutralen Kaonensystems $K_S/K_L$ am Experiment KLOE am Speicherring DA$\Phi$NE in Frascati wurde eine zuverlässige Methode zur Unterscheidung zwischen Myonen und Pionen gesucht. Die konventionelle Methode der Messung des differentiellen Energieverlusts pro Wegstrecke $\frac{d\mathrm{E}}{d\mathrm{x}}$ ist durch die hohe Varianz der Verteilung (Landau-Verteilung) in ihrem Auflösungsvermögen beschränkt. Bei Anwendung einer 'truncated mean'-Analyse läßt diese Methode für einen Detektor wie KLOE eine Auflösung von etwa 4.2 % erwarten.
Deswegen wurde als Alternative zur $\frac{d\mathrm{E}}{d\mathrm{x}}$-Messung die Methode des 'Cluster Counting' untersucht. Hierbei wird nicht der Energieverlust eines Teilchens bei Durchqueren einer Driftzelle anhand der freigesetzten Ladung gemessen, sondern es wird die Anzahl der ionisierenden Stöße dieses Teilchens mit den Atomen oder Molekülen des Kammergases gezählt. Die Zahl dieser Stöße sollte einer Poisson-Verteilung folgen, die eine deutlich kleinere Varianz als die oben erwähnte Landau-Verteilung aufweist. Bei der Verwendung einer Helium-Isobutan-Mischung als Driftgas ergibt sich aus der Form dieser Verteilung unter denselben Bedingungen wie oben eine Auflösung von etwa 2 %.
Diese Meßmethode hat aber zwei wesentliche Nachteile, die beide daraus resultieren, daß zur Clustersuche die Form des Drahtsignals betrachtet werden muß. Zum Einen muß die Signalstrecke von der Driftzelle bis zum ADC als Übertragungsleitung ('transmission line') betrachtet werden, die auch die hochfrequenten Anteile des Drahtsignals originalgetreu bis zum ADC führt. Zum Anderen wird ein schneller ADC mit einer Abtastrate im Bereich von 1 GS/s benötigt, was wiederum in einer extrem hohen Ausgangsdatenrate resultiert. Für ein Datennahmesystem bei einem Hochraten-Experiment wie KLOE ergibt sich hier eine Ausgangsdatenrate des gesamten Systems von etwa 4 Gigabytes pro Sekunde.
Diese Datenrate läßt sich aber durch eine Vorverarbeitung ohne Verlust an für die spätere Auswertung relevanter Information auf etwa 6 Megabytes pro Sekunde reduzieren. Diese Daten bestehen dann noch aus der Information über die Startzeit des ersten Clusters sowie der Anzahl der gefundenen Cluster innerhalb jeder Driftzelle, zusammen mit einer Identifikationsnummer für Ereignis und Driftzelle.
Nach ersten Untersuchung zum Potential und der Anwendung von Cluster Counting im Rahmen einer Zusammenarbeit zwischen der Universität Karlsruhe, dem INFN Lecce, dem INFN Rom und der Università di Roma II wurde in einer engeren Zusammenarbeit zwischen der Università di Roma II und der Universität Karlsruhe ein Versuchssystem zu Cluster Counting aufgebaut. Ziel dieses Aufbaus ist die Demonstration der Durchführbarkeit von Cluster Counting mit einer online-

Datenvorverarbeitung. Zu diesem Versuchssystem wurde im Rahmen der vorliegenden Arbeit ein an den Datenfluß angepaßter Präprozessor entworfen und gebaut.
Dieser Präprozessor ist als paralleler MIMD-Rechner mit einem getrennten Prozessor zur Verwaltung der Kommunikationswege und gemeinsamen internen Strukturen aufgebaut. Auf eine Mutterplatine, die als Haushaltsprozessor eine CPU vom Typ AMD Am29000™ enthält, können bis zu vier Rechenknoten mit jeweils einem AMD Am29050™ aufgesteckt werden. Die Verteilung der Eingangsdaten zu den einzelnen Rechenknoten erfolgt über einen schnellen, unabhängigen Datenverteilungsbus.
Zu diesem Präprozessor wurde weiterhin ein Algorithmus zur Clustersuche entwickelt, der im Unterschied zum bisher verwendeten Algorithmus die gesamte, in den Meßwerten vorhandene Information ausnutzt. Bei den Tests des Programms mit simulierten und echten Daten zeigte sich, daß die Clustersuche damit möglich ist. Gleichzeitig zeigte sich aber auch, daß ein Vielzweckprozessor keine ausreichende Rechenleistung zur Verfügung stellen kann, um diese Datenvorverarbeitung in einem Hochraten-Experiment wie KLOE online durchzuführen. In diesem Fall wären etwa fünf bis zehn Rechenknoten pro Detektorkanal notwendig, um die Datenrate bewältigen zu können.
Es gibt jedoch noch die Möglichkeit, den Algorithmus in einem (oder mehreren) programmierbaren Logikbaustein(en) als Hardware-Pipeline aufzubauen. Eine solche Pipeline könnte entweder mit einer niedrigen Taktrate innerhalb eines derzeit erhältlichen FPGA-Bausteins aufgebaut werden, oder aber, entweder bei Verwendung mehrerer der größten derzeit erhältlichen FPGAs oder bei steigender Integrationsdichte in einem einzelnen zukünftigen FPGA, mit einer Taktfrequenz von bis zu 70 MHz arbeiten. Im letzteren Fall könnte eine solche Pipeline bis zu 100 Kanäle eines Hochraten-Experiments in Echtzeit verarbeiten.
Als Antwort auf die in Abschnitt 2.1 aufgeworfene Frage läßt sich hier also sagen, daß Cluster Counting in der Praxis verwendbar ist. Der in Kapitel 4 vorgestellte Algorithmus zur Kurvenanpassung nach der Methode der kleinsten Quadrate nutzt zwar alle in den Rohdaten vorhandene Informationen über die Cluster und auch alles a priori-Wissen über die Form der Signale aus, benötigt in der vorliegenden Version allerdings noch zu viel Rechenzeit. Die Möglichkeit, diese Berechnungen durch neuere Prozessoren und programmierbare oder anwenderspezifische Bauelemente zu beschleunigen, steht leider erst seit jüngerer Zeit, am Ende dieser Arbeit, zur Verfügung.
Aufgrund der Herausforderungen des KLOE-Experimentes [13, 14] wurden 1993 Arbeiten zum Cluster Counting [30, 31, 35, 41, 42] begonnen. Es zeigte sich, daß diese, wie auch schon frühere Arbeiten [1], ihrer Zeit zu weit voraus waren. Durch die Fortschritte in der Elektronik werden die in diesen Arbeiten gewonnenen Erfahrungen aber neue Anregungen im Rahmen geeigneter zukünftiger Experimente liefern können.

# Anhang A

# Der Aufbau der CPUs Am29000™ und Am29050™

Bei der CPU Am29000™/Am29050™ handelt es sich um einen RISC-Prozessor in Harvard-Architektur (Abbildung A.1) des Herstellers AMD [51, 52, 53, 54].

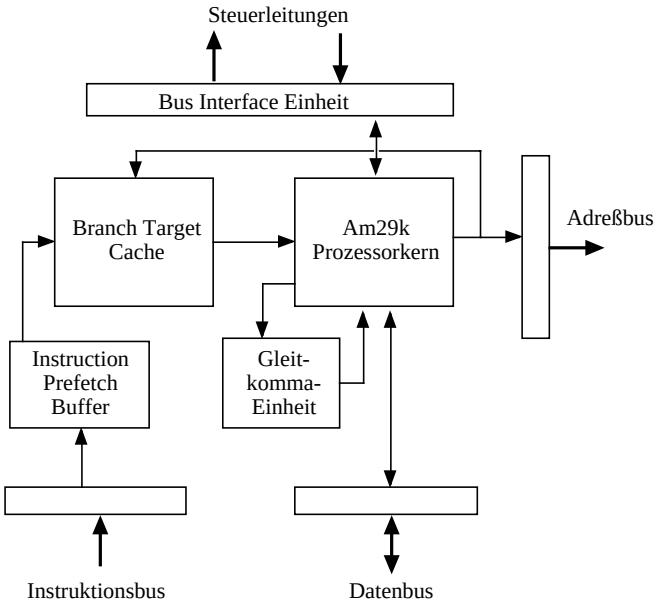



Abbildung A.1: Blockschaltbild der CPU Am29050™ , aus [54]

Das abgebildete Blockschaltbild zeigt den Prozessor Am29050™. Beim einfacher aufgebauten Am29000™ entfallen die Gleitkommaeinheit, der Instruction Prefetch Buffer und der Branch Target Cache. Die Funktionen der Baugruppen sind im Einzelnen:

**Instruktionsbus** ein externer, 32 Bit breiter unidirektionaler Bus, über den die Instruktionen aus einem externen Instruktionsspeicher an die CPU geführt werden

**Datenbus** ein externer, 32 Bit breiter bidirektionaler Bus, über den die CPU Daten mit einem externen Datenspeicher austauscht

**Adreßbus** ein externer, 32 Bit breiter unidirektionaler Bus, über den die CPU die Adressen für Instruktions- und Datenzugriffe ausgibt.

**Steuerleitungen** eine Gruppe von größtenteils unidirektionalen Steuerleitungen, über welche die CPU mit ihrer Umgebung kommuniziert. Darunter fallen die Schreib-/Lese-Steuerung, Interruptleitungen sowie Steuerleitungen für die Art des Daten- und Instruktionszugriffs.

**Bus Interface Einheit** diese Baugruppen stellen die Verbindungen zwischen den internen Leitungen der CPU und den externen Leitungen her. Im Falle der Am29k-Serie schließt dies ein Überwachungssystem mit ein, das bei Diskrepanzen zwischen den internen und externen Signalen auf einer der Steuerleitungen eine entsprechende Fehlermeldung ausgibt.

**Instruction Prefetch Buffer** Dies ist eine Pufferstufe, die wesentlich zu der Fähigkeit der CPU beiträgt, eine Instruktion pro Taktzyklus verarbeiten zu können. In der Am29k-Serie läuft die Ausführung einer Instruktion in einer vierstufigen Pipeline nach folgendem Schema ab:

1. Instruktion laden
2. Instruktion dekodieren
3. Instruktion ausführen
4. Ergebnisse in Register schreiben

Um eine kontinuierliche Ausführung zu gewährleisten, muß eine Instruktion also spätestens drei Takte vor ihrer Ausführung bereits aus dem externen Instruktionsspeicher angefordert werden. Wenn die Instruktion angefordert worden ist, muß sie dann aber auch von der CPU zum festgelegten Zeitpunkt entweder verwendet oder verworfen werden. Damit, falls durch die verzögerte Ausführung einer Instruktion der Instruktionsfluß ins Stocken kommt, die angeforderten Instruktionen nicht verworfen werden müssen, um dann später erneut angefordert zu werden, werden zunächst alle Instruktionen im Instruction Prefetch Buffer zwischengespeichert. Diese Pufferstufe faßt vier Instruktionen und kann somit eine komplette Füllung der Instruktionspipeline zwischenlagern.

**Branch Target Cache** : Dies ist ein weiteres Hilfssystem, um die CPU mit einem kontinuierlichen Instruktionsstrom zu versorgen und gleichzeitig die externen Busse zu entlasten. Im Fall, daß die CPU zum Beispiel eine Programmschleife abarbeitet, können im Branch Target Cache die Instruktionen für die bisher durchlaufenen Verzweigungen (‘Branches’) der Schleife zwischengelagert und bei Bedarf direkt wieder der Verarbeitungseinheit zugeführt werden, ohne daß Zugriffe auf den externen Instruktionsspeicher nötig wären.

**Am29k-Prozessorkern** : Dies ist die eigentliche Verarbeitungseinheit.

**Gleitkomma-Einheit** : Ein Zusatzprozessor, der den Prozessorkern bei zeitaufwendigen Gleitkommaoperationen entlastet. Beim Am29000™ werden stattdessen Gleitkommaoperationen (zeitaufwendiger) in einer Softwareemulation im Prozessorkern selbst abgearbeitet.

# Anhang B

# Die Speicheraufteilung des Präprozessors

Die Adreßbereiche von Master und Slaves sind gemäß der folgenden Tabellen in Untersegmente aufgeteilt.

## B.1 Slaves

Die Speicherkarte eines Slaves zeigt Tabelle B.1.

| Bereich | Adreßbereich(Hexadezimal) | Inhalt |
|---|---|---|
| 0 | 0x00000000 ... 0x0000FFFC | Instruktionsspeicher - ROM |
| | 0x00010000 ... 0x0001FFFC | gespiegeltes Instruktions-ROM |
| 1 | 0x00020000 ... 0x0002FFFC | gemeinsames ROM |
| | 0x00030000 ... 0x0003FFFC | gespiegeltes gemeinsames ROM |
| 2 | 0x00040000 ... 0x0005FFFC | Instruktionsspeicher - RAM |
| 3 | 0x00060000 ... 0x0007FFFC | Datenspeicher - RAM |
| 4 | 0x00080000 ... 0x0009FFFC | leer |
| 5 | 0x000A0000 ... 0x000BFFFC | Kommunikationsregister |
| 6 | 0x000C0000 ... 0x000DFFFC | Zugriff auf externen Datenspeicher |
| 7 | 0x000E0000 ... 0x000FFFFC | leer |
| ext | 0x00100000 ... 0xFFFFFFFC | ungenutzt |

Tabelle B.1: Speicherkarte der Slaves

Einzelne Speicherbereiche, insbesondere der Bereich des Datenspeicher-RAM, sind von der Programmierung her noch weiter unterteilt, da für verschiedene spezielle Programmbereiche feste Speicheradressen benötigt werden. Des weiteren besteht der Bereich ‘Kommunikationsregister’

eigentlich nur aus einem einzigen, 8 Bit breiten Register und nicht, wie die Tabelle suggeriert, aus 128 kBytes Speicherbereich. Jeder Zugriff auf diesen Bereich liest (oder schreibt) physikalisch in dasselbe 8-Bit-Register. Über dieses Register können kurze Nachrichten zwischen Master und Slave ausgetauscht werden, mit denen dann Aktionen wie zum Beispiel die Datenübergabe von beiden Seiten her koordiniert werden können.
Wie aus der Speicherkarte ersichtlich ist, werden die oberen 12 Adreßbits des Slave-Prozessors nicht benutzt. Die acht aufgeführten Speicherbereiche ergeben sich daraus, daß die Adreßdekodierlogik der Slaves die drei Adreßbits A[17]...A[19] auswertet. Die gespiegelten Bereiche entstehen, wenn der physikalisch vorhandene Speicher kleiner ist als einer dieser Bereiche. In diesem Fall überdecken die beiden ROM-Speicherbereiche jeweils gerade die Hälfte eines adressierbaren Speicherblocks, das heißt, A[18] wird von diesen Speicherbausteinen nicht ausgewertet. Ansonsten werden innerhalb der Speicherbereiche alle Adressen von den Speicherbausteinen dekodiert. Die Ausnahme von dieser Regel ist der Speicherbereich 5, in dem sich das schon angesprochene Kommunikationsregister befindet, in dem keine weiteren Adreßbits ausgewertet werden. Oberhalb von 0x00100000 schließlich befindet sich aus der Sicht des Slaves nur noch ‘leerer Raum’, Zugriffe auf diesen Adreßbereich werden nicht beantwortet.
Da in diesem ’leeren Raum’ aber die externen Datenspeicher liegen, auf die der Slave von Zeit zu Zeit zugreifen muß, müssen Zugriffe auf diesen Bereich über einen Umweg adressiert werden. Dazu schreibt der Master-Prozessor vor einem Zugriff des Slaves in diesen Adreßraum die Adreßbits A[17]...A[24] in ein Register. Damit können Zugriffe, die aus Sicht der Slave-CPU in den Adreßbereich zwischen 0x000C0000 und 0x000DFFFC gehen, von den Adreßdekodern auf der Mutterplatine im gesamten Bereich zwischen 0x00000000 und 0x00FFFFFC erkannt werden. Der oberste Adreßbereich ab 0x01000000 ist für VMEbus-Zugriffe vorgesehen, die vom Slave nicht durchgeführt werden sollen.

## B.2 Master

Der Speicherbereich des Masters ist in zwei Hauptsegmente unterteilt, einen inneren Teil (Tabelle B.2) und einen äußeren Teil (Tabelle B.3).
Der interne Speicherbereich steht dem Master permanent und exklusiv zur Verfügung. Hier sind, ähnlich wie bei den Slaves, die lokalen Instruktions- und Datenspeicher untergebracht. Im Unterschied zu den Slaves steht dem Master hier allerdings jeweils der doppelte Speicherplatz zur Verfügung.
Zugriffe auf Adressen im externen Speicherbereich des Masters müssen zuvor über eine Arbitrierungslogik freigegeben werden. Abhängig vom Adreßbereich und eventuell gesetzten Konfigurationsregistern können die entsprechenden Zugriffsrechte entweder automatisch vergeben werden oder auch exklusiv von einem Busteilnehmer gehalten werden. Mögliche Wettbewerber um die Busrechte können, je nach Zieladresse, entweder der Master, einer der Slaves oder aber ein über den VMEbus zugreifender externer Rechner sein.
Die automatische Vergabe der Zugriffsrechte wird über programmierbare Logikbausteine gesteuert, so daß bei Bedarf spezifische Arbitrierungsschemata verwendet werden können. In der gegenwärtigen Phase wird sowohl eine Arbitrierung auf ‘first come - first served’-Basis (Busbereich mit insgesamt drei Wettbewerbern) als auch eine ‘round robin’-Arbitrierung (gemeinsamer Datenausgangsbus der Slaves mit bis zu fünf Wettbewerbern) verwendet.

| Bereich | Adreßbereich(Hexadezimal) | Inhalt |
|---|---|---|
| 0 | 0x00000000 . . . 0x0000FFFC | Instruktionsspeicher - ROM |
| 1 | 0x00020000 . . . 0x0002FFFC | Instruktionsspeicher - RAM Block 0 |
| 2 | 0x00040000 . . . 0x0005FFFC | Instruktionsspeicher - RAM Block 1 |
| 3 | 0x00060000 . . . 0x0007FFFC | Datenspeicher - RAM Block 0 |
| 4 | 0x00080000 . . . 0x0009FFFC | Datenspeicher - RAM Block 0 |
| 5 | 0x000A0000 . . . 0x000BFFFC | leer |
| 6 | 0x000C0000 . . . 0x000DFFFC | leer |
| 7 | 0x000E0000 . . . 0x000FFFFC | leer |
| ext | 0x00100000 . . . 0xFFFFFFFC | externer Speicherbereich |

Tabelle B.2: Speicherkarte der Masters, interner Bereich

| Bereich | Adreßbereich(Hexadezimal) | Inhalt |
|---|---|---|
| 0 | 0x00000000 . . . 0x000FFFFC | Interner Bereich des Masters |
| 1 | 0x00100000 . . . 0x001FFFFC | Kontroll- und Statusregister |
| 2 | 0x00200000 . . . 0x002FFFFC | langsames EPROM, unterer Bereich |
| 3 | 0x00300000 . . . 0x003FFFFC | langsames EPROM, oberer Bereich |
| 4 | 0x00400000 . . . 0x004FFFFC | globales Daten-RAM, Modul 0 |
| 5 | 0x00500000 . . . 0x005FFFFC | globales Daten-RAM, Modul 1 |
| 6 | 0x00600000 . . . 0x006FFFFC | globales Daten-RAM, Modul 2 |
| 7 | 0x00700000 . . . 0x007FFFFC | globales Daten-RAM, Modul 3 |
| 8 | 0x00800000 . . . 0x008FFFFC | leer |
| 9 | 0x00900000 . . . 0x009FFFFC | leer |
| A | 0x00A00000 . . . 0x00AFFFFC | leer |
| B | 0x00B00000 . . . 0x00BFFFFC | größtenteils leer, am oberen Ende Kontrolle der Reset-Leitungen der Slaves und VME-Interface-Chips |
| C | 0x00C00000 . . . 0x00CFFFFC | Zugriff auf die Slave-Module |
| D | 0x00D00000 . . . 0x00DFFFFC | leer |
| E | 0x00E00000 . . . 0x00EFFFFC | größtenteils leer, am oberen Ende Kontroll- und Statusregister des VME-Interface-Chipsatzes |
| F | 0x00F00000 . . . 0x00FFFFFC | größtenteils leer, am unteren Ende Kontroll- und Ein-/Ausgaberegister der seriellen Schnittstellen |
| VME | 0x01000000 . . . 0xFFFFFFFC | Zugriffe auf den VMEBus |

Tabelle B.3: Speicherkarte der Masters, externer Bereich

# Anhang C

# Der Quellcode des Clustersuchalgorithmus

Hier nun der Quellcode des Clustersuchprogramms. Dieser Code läuft ohne Änderungen sowohl auf den beiden Apple Macintosh Rechnern mit dem CodeWarrior-Compiler [74] als auch auf dem Präprozessor bei Compilierung mit dem AMD-Compiler. Notwendige Anpassung an die jeweilige Rechnerhardware werden innerhalb der *#ifdef macintosh...#endif*-Anweisungen für den Betrieb auf einem der Apple-Rechner und innerhalb der *#ifndef macintosh...#endif*-Anweisungen für den Betrieb auf dem Präprozessor vorgenommen. Die Umgebungsvariable *macintosh* wird durch den CodeWarrior-Compiler definiert.
Im code befinden sich neben den im Abschnitt 4.4.2 erwähnten Kriterien für die Erkennung eines Clusters (die entsprechenden Variablen hier sind *chi_cut* für das maximal $\chi^2$, *alpha_cut* für die minimale Clustergröße sowie *center->chi_square* für die Bestimmung des lokalen Minimums in $\chi^2$) noch zwei weitere Kriterien. Die Variable *center->beta*, die Größe des zweiten angepaßten Clusters, wird daraufhin überprüft, ob sie in der Summe $\alpha + \beta$ ebenfalls die Anforderungen an die minimale Clustergrößer erfüllt, und es wird ein minimaler Wert für die Fläche unter der Signalkurve an der Stelle eines vermuteten Clusters verlangt (*int_cut* bzw. *reft->sum*).
Das erste dieser beiden Kriterien soll durch Schwankungen im Rauschen hervorgerufene 'falsche' Clustersignale unterdrücken, da in diesen Fällen oft die Anpassung für $\beta$ einen negativen Wert ergibt, so daß die Summe $\alpha+\beta$ wieder kleiner als der Referenzwert *alpha_cut* wird. Für tatsächlich vorliegende Signale sollte $\beta$ etwa Null sein, oder bei überlagerten Signalen entsprechend größer als Null.
Das zweite Kriterium dient ebenfalls der Unterdrückung von durch Schwankungen im Rauschen hervorgerufenen 'falschen' Signalen, da hier gilt, daß der Mittelwert des Rauschens (oder auch das Integral) über einen ausreichend großen Zeitraum Null sein sollten.
In beiden Fällen stellt jedoch die Anforderung an ein lokales Minimum in $\chi^2$ ein wesentlich schärferes Kriterium dar. Das erste der beiden Kriterien wurde für die Erstellung der Daten in Abschnitt 4.4 beibehalten, das zweite durch einen sehr kleinen Abschneidewert (*int_cut* = $-10^{-4}$) ausgeschaltet. Die entsprechenden Codezeilen wurden beibehalten, da diese einfachen Abfragen im Rahmen des gesamten Programms nur einen geringen Bruchteil der Verarbeitungszeit benötigen.

```
#ifndef macintosh
    #include variable.h
    #include slaveadd.h
    #include slavecod.h
    #include slavfunc.h
    #include <stdlib.h>
    #include globals.h
    #include <math.h>
#endif

#ifdef macintosh
    #include "mactypes.h"
    #include <stdlib.h>
    #include <OSUtils.h>
    #include <math.h>
    #include <Timer.h>
    #include <stdio.h>
#endif

/*****************************************************************
*
*     find_cluster wird aus dem Unterprogramm sl_run aufgerufen
*     und erledigt die Clustersuche.
*     Ein- und Ausgangsdaten sind:
*     0x70000 ... 0x74000  Referenzkurve (float)
*     0x74000 ... 0x78000  Messdaten, Uebergabe als Int32
*                          waehrend des Programmlaufs Formatkonversion
*                          auf float
*     0x78000 ... 0x7A000  Ausgangsdaten (gefundene Cluster) im Format:
*                          - Groesse (float)
*                          - chi^2   (float)
*                          - Abstand zum letzten Cluster (float)
*                          - Position absolut  (float)
*     0x7A000 ... 0x7B000  Parameter (alle float):
*                           0: 1/n
*                           1: c_0
*                           2: c_1
*                           3: c_11
*                           4: c_00
*                           5: 1/denom = 1/(c_01 - (c_11 * c_00)/c_01)
*                           6: 1/c_01
*                           7: (float) ref_length
*                           8: (float) start
*                           9: (float) offset
*                          10: chi_cut
*                          11: alpha_cut
*                          12: 1/sigma^2
*                          13: Integral of reference curve
*                          14: integral cut
*                          15: (float) sample_length
```

```
*     0x7B000 ...          Programminterne Variablen, z.B. fitring
*
*************************************************************************/

/**********************************

   Modifiziert fuer Mac:

   die Dateien werden von Platte geladen, die Speicherbereiche werden
   ueber  malloc/calloc zugewiesen und verwaltet

   Zeitnahme erfolgt aber auch nur fuer die innere Schleife, d.h.
   nachdem die Daten von der Platte geladen wurden und bevor die
   Ergebnisse auf die Platte zurueckgeschrieben werden
   Dateien:
    Referenzkurve:     "curve.dat"
    Daten:             "idata.dat"
    Parameter:         "parameter.dat"
    gefundene Cluster: "clusters.dat"


**********************************/

#ifndef macintosh

 pragma Code ("eprom");

#endif

struct fitring
{
 struct fitring * prev;
 float alpha;
 float beta;
 float gamma;
 float chi_square;
 struct fitring * next;
};
struct refring
{
 struct refring * prev;
 float sum;
 struct refring * next;
};


int find_clusters (void)
{
 int i, j, k, l, m, clus, cl_pos;
 int32 idum;
 int sample_length, ref_length, start, offset;
 int max_clus;
```

```
 float s_m, s_m0, s_m1;
 float chi_temp, dummy, timp;
 float sub;
 float * fdata;
 int32 * idata;
 float * curve;
 float * parameter;
 float * found_clusters;
 float * cc0, *cc1;
 struct fitring * fit, * center, * fit0;
 struct refring * ref, * ref0, * reft;
 float one_n, c_0, c_1, c_11, c_00, one_denom, one_c_01;
 float chi_cut, alpha_cut, one_sigma2, int_ref, int_cut;

#ifndef macintosh

 uint32  pointer;

 curve          = (float *) 0x00070000;
 idata          = (int32 *) 0x00074000;
 fdata          = (float *) 0x00074000;
 found_clusters = (float *) 0x00078000;
 parameter      = (float *) 0x0007A000;
 pointer        = 0x0007B000;

 i = 0;

#endif

#ifdef macintosh

 /*******
 FILE * log_file;
 *******/
 FILE * in_file;
 FILE * out_file;
 char * filename;
 int retval;

 struct TMTask time_queue;

 long start_time;
 int time;

 i = 0;

 filename = (char*) calloc (255, sizeof (char));

 curve          = (float *) calloc (4096, sizeof (float));
 if (curve == NULL) return (-1);
 fdata          = (float *) calloc (4096, sizeof (float));
 if (fdata == NULL) return (-1);
```

```
 found_clusters = (float *) calloc (2048, sizeof (float));
 if (found_clusters == NULL) return (-1);
 parameter      = (float *) calloc (1024, sizeof (float));
 if (parameter == NULL) return (-1);
 idata          = (int32 *) calloc (4096, sizeof (int32));
 if (idata == NULL) return (-1);

 /********
 filename = "cluster.log";
 log_file = fopen (filename, "w");
 fprintf (log_file, " i     alpha    beta    gamma    chi^2 \n");
 *******/

 filename = ":curve.dat";
 in_file = fopen (filename, "r");
 if (in_file == NULL) return (-6);
 retval = 1;
 i = 0;
 while ((retval == 1) && (i < 4096))
 {
  retval = fscanf (in_file, "%e", &curve[i]);
  i++;
 }
 fclose (in_file);

 filename = ":idata.dat";
 in_file = fopen (filename, "r");
 if (in_file == NULL) return (-6);
 retval = 1;
 i = 0;
 while ((retval == 1) && (i < 4096))
 {
  retval = fscanf (in_file, "%i", &idata[i]);
  i++;
 }
 fclose (in_file);

 filename = ":parameter.dat";
 in_file = fopen (filename, "r");
 if (in_file == NULL) return (-6);
 retval = 1;
 i = 0;
 while ((retval == 1) && (i < 1024))
 {
  retval = fscanf (in_file, " %e", &parameter[i]);
  i++;
 }
 fclose (in_file);

#endif

 max_clus = 450;     /* maximum that fits inside the alloted
```

```
                        memory space is 500, so take less */

 ref_length     = (int) parameter[7];
 start          = (int) parameter[8];
 offset         = (int) parameter[9];
 sample_length  = (int) parameter[15];

 if (sample_length > 4096) return (-2);
 if (ref_length > sample_length) return (-3);

 one_n          = parameter[0];
 c_0            = parameter[1];
 c_1            = parameter[2];
 c_11           = parameter[3];
 c_00           = parameter[4];
 one_denom      = parameter[5];
 one_c_01       = parameter[6];

 chi_cut        = parameter[10];
 alpha_cut      = parameter[11];
 one_sigma2     = parameter[12];
 int_ref        = parameter[13];
 int_cut        = parameter[14];

 for (i = 0; i < sample_length; i++)
 {
  idum = idata[i];
  dummy = (float) idum;
  fdata[i] = dummy;
 }

#ifndef macintosh

 fit0 = (struct fitring *) pointer;
 fit  = fit0;
 for (i = 0; i < 5; i++) fit[i].next = &fit[i+1];
 for (i = 1; i < 6; i++) fit[i].prev = &fit[i-1];
 fit[5].next = &fit[0];
 fit[0].prev = &fit[5];
 fit = &fit[0];
 pointer = pointer + 6 * sizeof (struct fitring);

 ref0 = (struct refring *) pointer;
 ref = ref0;
 for (i = 0; i < ref_length+1; i++) ref[i].next = &ref[i+1];
 for (i = 1; i < ref_length+2; i++) ref[i].prev = &ref[i-1];
 ref[ref_length+1].next = &ref[0];
 ref[0].prev = &ref[ref_length+1];
 ref = &ref[0];
 pointer = pointer + (ref_length + 2) * sizeof (struct refring);
```

```
 cc0 = (float *) pointer;
 pointer = pointer + ref_length * sizeof (float);
 cc1 = (float *) pointer;
 pointer = pointer + ref_length * sizeof (float);

#endif

#ifdef macintosh

 fit0 = (struct fitring*) calloc (6, sizeof (struct fitring));
 if (fit0 == NULL) return (-4);
 fit  = fit0;
 for (i = 0; i < 5; i++) fit[i].next = &fit[i+1];
 for (i = 1; i < 6; i++) fit[i].prev = &fit[i-1];
 fit[5].next = &fit[0];
 fit[0].prev = &fit[5];
 fit = &fit[0];

 ref0 = (struct refring *) calloc (ref_length + 2, sizeof (struct
   refring));
 if (ref0 == NULL) return (-4);
 ref = ref0;
 for (i = 0; i < ref_length+1; i++) ref[i].next = &ref[i+1];
 for (i = 1; i < ref_length+2; i++) ref[i].prev = &ref[i-1];
 ref[ref_length+1].next = &ref[0];
 ref[0].prev = &ref[ref_length+1];
 ref = &ref[0];

 cc0 = (float *) calloc (ref_length, sizeof (float));
 if (cc0 == NULL) return (-5);
 cc1 = (float *) calloc (ref_length, sizeof (float));
 if (cc0 == NULL) return (-5);

 /* some time-taking code ? */

 time_queue.tmAddr = 0;
 time_queue.tmWakeUp = 0;
 time_queue.tmReserved = 0;
 InsXTime ((QElemPtr) &time_queue);
 start_time = 6000;
 PrimeTime ((QElemPtr) &time_queue, start_time);

#endif

 for (i = 0; i < ref_length; i++)
 {
  cc0[i] = 0;
  cc1[i] = 0;
  if (i >= start)
   cc0[i] = curve[i - start];
  if (i >= (start + offset))
   cc1[i] = curve[i - start - offset];
```

```
}

/****  begin main loop   ****/

m = 2;
clus = 0;
for (i = 0; i < sample_length - ref_length; i++)
{
 /* start with calculating the sums  */

 s_m  = 0.0;
 s_m0 = 0.0;
 s_m1 = 0.0;
 for (j = 0; j < ref_length; j++)
 {
  s_m += fdata[i+j];
  s_m0 += (fdata[i+j] * cc0[j]);
  s_m1 += (fdata[i+j] * cc1[j]);
 }
 ref->sum = s_m;

 /* now for alpha, beta and gamma */

 fit->beta = one_denom * (s_m0 - one_n * c_0 * s_m -
                   one_c_01 * (( s_m1 - one_n * c_1 * s_m) * c_00));
 fit->alpha = one_c_01 * (s_m1 - one_n * c_1 * s_m -
                   fit->beta * c_11);
 fit->gamma = one_n * (s_m - fit->alpha * c_0 - fit->beta * c_1);
 chi_temp = 0.0;
 for (j = 0; j < ref_length; j++)
 {
  dummy = fdata[i+j] - fit->gamma;
  dummy -= (fit->alpha * cc0[j]);
  dummy -= (fit->beta * cc1[j]);
  chi_temp += (dummy * dummy);
 }
 fit->chi_square = one_sigma2 * chi_temp;

 /* now check for clusters:
     - chi^2 < chi_cut
     - alpha > alpha_cut
     - relative minimum in chi^2
 */

 if ((i > ref_length) && (clus < max_clus))
 {
  j = i - 2;
  center = fit->prev->prev;
  /********
  fprintf (log_file, " %5i %+13.8f %+13.8f %+13.8f %+13.8f \n ",
  j, center->alpha, center->beta, center->gamma, center->chi_square);
  ********/
```

```
  if (j > m)
  {
   if (center->chi_square < chi_cut)
   {
    if ((center->alpha > alpha_cut)  && ((center->alpha +
         center->beta) > alpha_cut))
    {
     reft = ref;
     for (k = 0; k < ref_length; k++)
      reft = reft->prev;
     if ((((center->gamma - reft->sum) * ref_length) +
           (center->alpha * int_ref)) > int_cut)
     {
      timp = (20 * (center->gamma * ref_length + center->alpha
              * int_ref)  - reft->sum);
      if ((center->chi_square < center->next->chi_square) &&
          (center->chi_square < center->prev->chi_square) &&
          (center->next->chi_square < center->next->next->chi_square)
           &&
          (center->prev->chi_square < center->prev->prev->chi_square))
      {
       cl_pos = 4 * clus;
       found_clusters[cl_pos] = center->alpha;
       found_clusters[cl_pos+1] = center->chi_square;
       found_clusters[cl_pos+2] = (float)(j - m);
       found_clusters[cl_pos+3] = (float)(j + start);
       clus++;
       sub = 100 * fabs(center->alpha);
       if ((j + start + sub) > sample_length)
        sub = sample_length;
       else
        sub = j + start + sub;
       for (k = j+start, l = 0; k < sub; k++, l++)
        fdata[k] -= (center->alpha * curve[l]);
       m = j;
       i = j-3;
      }
     }
    }
   }
  }
 }
 if (clus >= max_clus)  /* too many clusters found  */
 {
 clus = max_clus;
 cl_pos = 4*(clus);
 found_clusters[cl_pos] = 0.0;
 found_clusters[cl_pos+1] = -1.0;
 }
 fit = fit->next;
 ref = ref->next;
}
```

```
 if (clus < max_clus)   /* within valid range */
 {
  cl_pos = 4 * clus;
  found_clusters[cl_pos] = 0.0;
  found_clusters[cl_pos+1] = 1.0;
  found_clusters[cl_pos+2] = (float) (clus);
 }

#ifdef macintosh

/* some time-taking code ? */
 RmvTime ((QElemPtr) &time_queue);
 if (time_queue.tmCount < 0)
 {
  time = (int) ((1000 * start_time + time_queue.tmCount));
  printf (" Time taken for main loop: %8i usec \n", time);
 }
 else
 {
  time = (int) ((-1) * (start_time + time_queue.tmCount));
  printf (" Time taken for main loop: %8i msec \n", time);
 }
 time_queue.tmAddr = 0;
 time_queue.tmWakeUp = 0;
 time_queue.tmReserved = 0;
 InsXTime ((QElemPtr) &time_queue);
 start_time = 1;
 PrimeTime ((QElemPtr) &time_queue, start_time);
 RmvTime ((QElemPtr) &time_queue);
 time = (int) (1000 * start_time + time_queue.tmCount);
 printf (" Time taken for time manager routines : %8i usec \n", time);

 /********
 fclose (log_file);
 *******/


 filename = ":clusters.dat";
 out_file = fopen (filename, "w");
 if (out_file == NULL) return (-6);
 for (i = 0; i < (4 * (clus + 1)); i+=4)
 retval = fprintf (out_file, " %+13.6e    %+13.6e    %+13.6e
     %+13.6e \n ", found_clusters[i], found_clusters[i+1],
     found_clusters[i+2], found_clusters[i+3]);
 retval = fprintf (out_file, "eof");
 fclose (out_file);

 filename = ":flat.dat";
 out_file = fopen (filename, "w");
 if (out_file == NULL) return (-6);
 for (i = 0; i < sample_length; i++)
  retval = fprintf (out_file, " %+13.6e \n", fdata[i]);
```

```
 retval = fprintf (out_file, "eof");
 fclose (out_file);

 free (cc1);
 free (cc0);
 free (ref0);
 free (fit0);
 free (idata);
 free (parameter);
 free (found_clusters);
 free (fdata);
 free (curve);
 free (filename);

#endif
 return (1);
}

#ifndef macintosh

pragma Code;

#endif
```

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis


[1] A. H. Walenta, *The Time Expansion Chamber and Single Ionization Cluster Measurement*, IEEE NS-26, No. 1 (1979), S. 73 ff.

[2] E. Lohrmann, *Einführung in die Elementarteilchenphysik*, Teubner Studienbücher Physik, 1983

[3] Particle Data Group, *Review of Particle Properties*, Physical Review **D50**, 3 (1994)

[4] L. Montanet et al., Physical Review **D50**, 1173 (1994) und die zwischenzeitliche Ergänzung von 1995 für die Ausgabe von 1996, erhältlich über die WWW-Seiten der PDG `http://pdg.lbl.gov/`.

[5] R. L. Gluckstern, *Uncertainties in Track Momentum and Direction due to Scattering and Measurement Errors*, Nucl. Instr. Meth. **24** (1963) S. 381 ff

[6] K. Kleinknecht, *Detektoren für Teilchenstrahlung*, Teubner Studienbücher Physik, 1984, ISBN 3-519-03058-6

[7] W. W. M. Allison, J. H. Cobb, *Relativistic Charged Particle Identification by Energy Loss*, Ann. Rev. Nucl. Part. Sci. **30** (1980), 253

[8] A. H. Walenta, J. Fischer, H. Okuno und C. L. Wang, *Measurement of the Ionization Loss in the Region of Relativistic Rise for Noble and Molecular Gases*, Nucl. Instr. Meth.. **161** (1979), S. 45 ff.

[9] J. D. Jackson, *Classical Electrodynamics*, 2nd Edition, John Wiley & Sons, New York, 1975

[10] L. C. L. Yuan and C.-S. Wu, editors, *Methods of Experimental Physics* Vol. 5a, Academic Press, 1961

[11] W. W. M. Allison and P. R. S. Wright, 'The Physics of Charged Particle Identification: $\frac{d\mathrm{E}}{d\mathrm{x}}$, Čerenkov Radiation and Transition Radiation' in T. Ferbel (editor), *Experimental Techniques in High Energy Physics*, Addison-Wesley, 1987

[12] Prof. Dr. W. deBoer, Universität Karlsruhe, private Kommunikation

[13] F. Murtas, *The KLOE Experiment at LNF-Frascati*, `http://www.lnf.infn.it/kloe/`

[14] The KLOE Collaboration, *The KLOE Detector Technical Proposal*, INFN Laboratori Nazionali di Frascati, **LNF-93/002** 1993

[15] *The DAΦNE Physics Handbook Vol. I & II*, L. Maiani, G. Pancheri und N. Paver (editors), veröffentlicht von Servizio Documentazione dei Laboratori Nazionali di Frascati, 1993

[16] *The Second DAΦNE Physics Handbook Vol. I & II*, L. Maiani, G. Pancheri und N. Paver (editors), veröffentlicht von Servizio Documentazione dei Laboratori Nazionali di Frascati, 1995

[17] The DAΦNEProject Team / G. Vignola, 'Status Report on DAΦNE', *Workshop on Detectors and Physics for DAΦNE'95*, INFN 1995, S. 19 ff.

[18] G. D. Coughlan und J. E. Dodd, *The ideas of particle physics*, 2. Auflage, Cambridge University Press, 1991

[19] H. J. Lipkin, 'CP Violation for Pedestrians', in: *Workshop on Detectors and Physics for DAΦNE'95*, INFN 1995, S. 203 ff.

[20] R. D. Peccei, *Overview of Kaon Physics*, HEP-PH-9504392, 1995

[21] W. Kluge, *Physik mit DAΦNE*, Vortrag auf dem XXVII. Arbeitstreffen 'Kernphysik' in Schleching, 1996

[22] M. Calvetti, 'Experimental Search for Direct CP-Violation: Status of the NA48 Experiment at CERN', in: R. Baldini, F. Bossi, G. Capon und G. Pancheri (editors), *Workshop on Detectors and Physics for DAΦNE'95*, INFN 1995, S. 81 ff.

[23] P. Sartori, D. Bisello und M. Nigro, 'A Fast Čerenkov Device for a Detector at DAΦNE', in: G. Pancheri (editor), *Workshop on Physics and Detectors for DAΦNE*, INFN 1991, S. 533 ff.

[24] G. Finocchiaro, F. Grancagnolo und S. Spagnolo, *The KLOE drift chamber geometry*, KLOE memo n. 48, März 1996

[25] F. Grancagnolo, 'DAΦNE: a case for a cylindrical drift chamber in a Helium based gas mixture', in: *Workshop on Physics and Detectors for DAΦNE 1991*, INFN Laboratori Nazionali di Frascati, 1991

[26] The KLOE Collaboration, *The KLOE Central Drift Chamber - Addendum to the KLOE Technical Proposal*, **LNF-94/028**, 1994

[27] The KLOE Collaboration, 'The KLOE tracking chamber', in: E. Bertolucci, F. Cervelli, A. Scribano (editors), *Frontier Detectors for Frontier Physics*, Nucl. Instr. Meth.. Phys. Res. **A360** (1995), S. 48

[28] P. Bernardini, G. Fiore, R. Gerardi, F. Grancagnolo, U. von Hagel, F. Monittola, V. Nassisi, C. Pinto, L. Pastore, M. Primavera, *Precise measurements of drift velocities in helium gas mixtures*, Nucl. Instr. Meth.. Phys. Res. **A355** (1995) S. 428

[29] J. Lee-Franzini, 'Status Report on KLOE', in: *Workshop on Detectors and Physics for DAΦNE'95*, S. 31 ff.

[30] G. Cataldi, *Il problema della separazione $\pi/\mu$ nella camera a drift di KLOE*, Dipartimento di Fisica - Lecce, Dottorato di Ricerca, 1995

[31] G. Cataldi, F. Grancagnolo, S. Spagnolo, *Cluster Counting in Helium Based gas Mixtures*, vorgesehen zur Veröffentlichung in Nucl. Instr. Meth.., 1996

[32] Paul Horowitz, Winfield Hill, *The Art Of Electronics*, second edition, Cambridge University Press, 1989, ISBN 0-521-37095-7, `http://www.artofelectronics.com/`

[33] Howard W. Johnson, Martin Graham, *High-speed digital design : a handbook of black magic*, Prentice-Hall, 1993, ISBN 0-13-395724-1

[34] Dr. S. Weseler, Universität Karlsruhe, private Kommunikation

[35] L. Cerrito et.al., *A Set Up for Testing the Primary Ionisation Counting in a Drift Chamber at Normal Condition*, vorgesehen zur Veröffentlichung in Nucl. Instr. Meth.., 1996

[36] *RF/IF Designer's Handbook*, The Mini-Circuits Division of Scientific Components, 13 Neptune Avenue, Brooklyn, NY 11235, 1992

[37] K. Borer et. al., 'Readout electronics development for the ATLAS silicon tracker', in: E. Bertolucci, F. Cervelli und A. Scribano (editors), *Frontier Detectors for Frontier Physics*, Nucl. Instr. Meth.. Phys. Res. **A360** (1995)

[38] J. Virdee, *Physics at the LHC*, Vortragsserie auf dem Workshop Herbst '96 des Graduiertenkollegs 'Elementarteilchenphysik' der Universität Karlsruhe

[39] W. H. Press, S. A. Teukolsky, W. T. Vetterling und B. Flannery, *Numerical Recipes in C - The Art of Scientific Computing*, 2. Ed., Cambridge University Press, 1992, ISBN 0-521-43108-5

[40] *transtec Gesamtkatalog*, Ausgabe Frühjahr 1996, transtec AG, Postfach 2000, 72017 Tübingen, 1996

[41] L. Cerrito, Vortrag im Seminar der Università di Roma II, 1995

[42] M. Billich, *Simulation und Entwicklung eines schnellen Datenschalters zur Anwendung in künftigen Experimenten der Hochenergiephysik*, Diplomarbeit der Universität Karlsruhe 1995, IEKP-KA/95-3, `http://www-ekp.physik.uni-karlsruhe.de/`

[43] *The VMEbus Specification*, VMEbus International Trade Association VITA, 10229 N. Scottsdale Rd., Suite E, Scottsdale, AZ 85253, 1987

[44] *The VMEbus International Trade Organisation*, `http://www.vita.com/`

[45] *Selected Reprints on Dataflow and Reduction Architectures*, Computer Society Press of the IEEE, 1987

[46] M. J. Flynn, *Very high-speed computing systems*, Proc. IEEE **54**:12 (December) S. 1901-1909

[47] David A. Patterson, John L. Hennessey, *Computer organization and design: the hardware/software interface*, Morgan Kaufmann Publishers, 1994

[48] *High Performance Computing: Technology, Methods and Applications*, edited by J. J. Dongarra, L. Grandinetti, G. R. Joubert, J. Kowalik, Elsevier Science, 1995

[49] P. G. Harison, N. M. Patel, *Performance Modelling of Communication Networks and Computer Architectures*, Addison-Wesley Publishing Company, 1992, ISBN 0-201-54419-9

[50] P. Bratley, B. L. Fox und L. E. Schrage, *A Guide To Simulation*, Springer Verlag, 1983, ISBN 3-540-90820-X

[51] *Am29000 User's Manual*, 1990, Advanced Micro Devices, Inc., 901 Thompson Place, Sunnyvale, California 94088-3453, Literatur in Europa über email an `euro.tech@amd.com` oder `euro.lit@amd.com`

[52] *Am29050™ Microprocessor User's Manual*, Advanced Micro Devices, 1991

[53] *Am29050 Streamlined Instruction Microprocessor*, Datenblatt, Advanced Micro Devices, 1990

[54] Trainings- und Seminarunterlagen der Firma AMD zur Am29k-Prozessorfamilie, Advanced Micro Devices, 1995

[55] *CMOS/BiCMOS Data Book*, Cypress Semiconductor, 1992, Cypress Semiconductor, 3901 North First Street, San Jose, CA 95134

[56] M. Rudyk, 'Spezifikation des VMEbusses', in: *Der VMEbus: ein Bussystem für 16/32 Bit-Mikroprozessoren*, Franzis Verlag, München, 1986

[57] *VIC068A/VAC068A User's Guide*, Cypress Semiconductor, 1992

[58] G. Schnurer, 'Prozessor-Poker', *c't – magazin für computertechnik*, Verlag Heinz Heise GmbH & Co KG, Hannover, Heft 12/1993, Seite 134 ff.

[59] *High Performance Data Book*, Cypress Semiconductor, 1995

[60] *Z85C30 Enhanced Serial Communications Controller*, Datenblatt, Advanced Micro Devices, 1989

[61] *MVP: The dawn of a new era in digital signal processing: Introducing TMS320C8x*, Texas Instruments, 1994

[62] *TMS320C80 Multimedia Video Processor MVP, Technical Brief (Preliminary)*, 1994, Texas Instruments

[63] *Digital Signal Processing in FLEX Devices*, Product Information Bulletin 23, January 1996, ver. 1, Altera Corporation, 2610 Orchard Parkway, San Jose, CA 95134-2020, `http://www.altera.com/`

[64] *FIR Filters*, Functional Specification 1, January 1996, ver. 1, Altera Corporation

[65] *fp_add_sub – Floating Point Adder/Subtractor*, Functional Specification 2, January 1996, ver. 1, Altera Corporation

[66] *Integer Dividers*, Functional Specification 3, January 1996, ver. 1, Altera Corporation

[67] *fp_mult – Floating Point Multiplier*, Functional Specification 4, January 1996, ver. 1, Altera Corporation

[68] *Implementing FIR Filters in FLEX devices*, Application Note 73, January 1996, ver. 1, Altera Corporation

[69] B. Niedermeier, Altera GmbH,'DSP Funktionen mit FLEX8000 CPLDs', in: W. Rosenstiel und A. Ditzinger, *2. GI/ITG Workshop Anwenderprogrammierbare Schaltungen*, FZI-Publikation 2/95, Forschungszentrum Informatik an der Universität Karlsruhe

[70] B. Fawcett und P. Alfke, Xilinx Inc, 'FPGAs as Reconfigurable Computing Elements', in: W. Rosenstiel und A. Ditzinger, *2. GI/ITG Workshop Anwenderprogrammierbare Schaltungen*, FZI-Publikation 2/95, Forschungszentrum Informatik an der Universität Karlsruhe

[71] A. McIver, *Software, Who Needs It ?*, New Scientist **2054** 1996, S. 40

[72] *Altera - 1996 Data Book*, Altera Corporation, 1996

[73] *Altera MaxPlus II Software*, LPM Bausteinbibliothek

[74] *CodeWarrior*, Metrowerks Corporation, Austin, Texas, 1995, `http://www.metrowerks.com/`

# Danksagung

Zum Schluß möchte ich denen danken, die die Erstellung dieser Arbeit ermöglicht haben.
Herrn Prof. Dr. K. R. Schubert, von dem die ursprüngliche Themenstellung im Rahmen der KLOE-Kollaboration stammte und ohne dessen Einsatz ich das Stipendium des Graduiertenkollegs nicht erhalten hätte.
Dem Graduiertenkolleg ‘Elementarteilchenphysik’ der Fakultät für Physik der Universität Karlsruhe, welches diese Arbeit über drei Jahre hinweg finanziert hat.
Herrn Prof. Dr. W. Kluge, der diese Arbeit über eine lange Strecke betreut hat.
Den Herren Prof. Dr. H. Gemmeke vom Forschungszentrum Karlsruhe und Prof. Dr. Th. Müller, die als Referenten die Betreuung dieser Arbeit in der Schlußphase sowie deren Bewertung übernommen haben.
Mein besondere Dank gilt Herrn Prof. Dr. Paoluzi und seinen Mitarbeitern, vor allem Herrn Dr. L. Cerrito, an der Università di Roma II. Von ihnen wurde ich zu einem Aufenthalt an der Università di Roma II eingeladen, während dessen der größte Teil der Grundlagen dieser Arbeit gelegt wurde. Ohne diese Einladung, die persönliche Betreuung in Rom und die finanzielle Unterstützung wäre die vorliegende Arbeit in dieser Form nie entstanden.
*Sono sinceramento grato al Prof. Luciano Paoluzi ed al suo gruppo dell’Università di Roma II, in particolare al Dott. Leonardo Cerrito. La struttura fondamentale di questo lavoro è stata sviluppata durante la mia permanenza, su loro invito, nell’Università di Roma II. Grazie a questo invito, alla loro personale supervisione ed al loro supporto finanziario è stato possibile il completamento di questa tesi nella sua attuale forma.*
Weiterer besonderer Dank geht an Herrn Dr. S. Weseler für das entgegengebrachte Interesse, die kritischen Fragen, zahlreichen Antworten und praktische Hilfe.
Ebenfalls danken möchte ich Herrn M. Billich, der mir während meiner Zeit in Karlsruhe gegenübersaß, für die vielen Diskussion und notwendigen Aufmunterungen, Anregungen und Ablenkungen während dieser Arbeit.
Und nicht zuletzt möchte ich hier meinen Eltern danken, ohne deren Unterstützung ich diese Arbeit nicht hätte fertigstellen können.