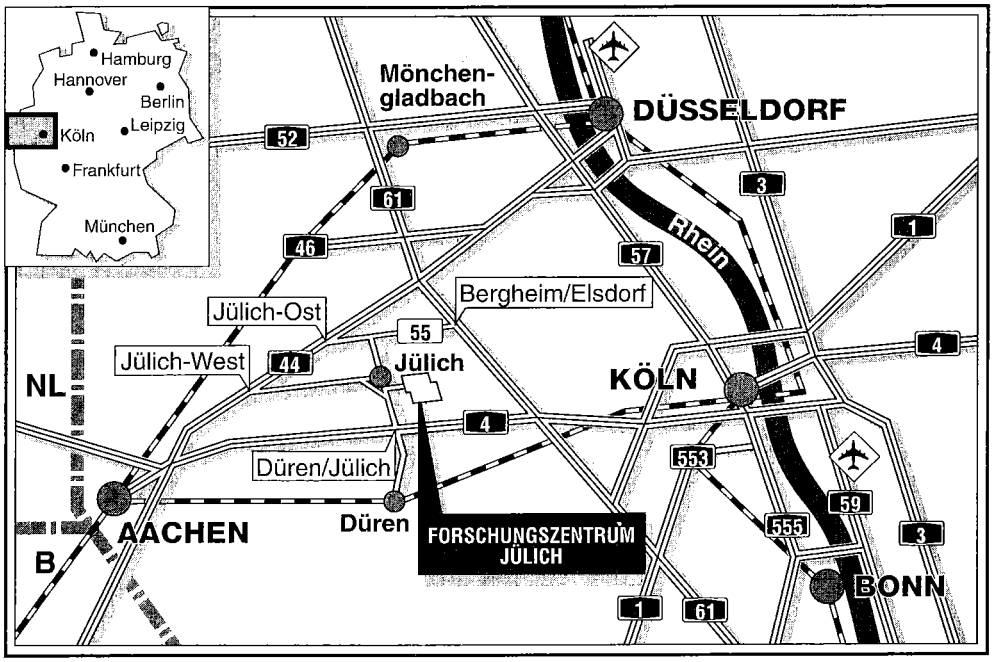
FORSCHUNGSZENTRUM JÜLICH GmbH

Zentrallabor für Elektronik

Aufbau eines flexiblen Datenaufnahmesystems für das GEM-Experiment am Jülicher Beschleuniger COSY und Messungen der Reaktion $pp \rightarrow \pi^+ d$ nahe der Produktionsschwelle

Matthias Drochner



Berichte des Forschungszentrums Jülich ; 3218
ISSN 0944-2952
Zentrallabor für Elektronik Jül-3218
D 88 (Technische Universität Dresden)

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Aufbau eines flexiblen Datenaufnahmesystems für das GEM-Experiment am Jülicher Beschleuniger COSY und Messungen der Reaktion $pp \rightarrow \pi^+ d$ nahe der Produktionsschwelle

Matthias Drochner

Inhaltsverzeichnis

Kapitel 1

Einführung

Seit 1993 steht an der KFA Jülich der Beschleuniger COSY für physikalische Experimente zur Verfügung.

COSY ist als Teilchenquelle für die Physik im mittleren Energiebereich vorgesehen. Besonderes Ziel bei der Konzeption war die Bereitstellung eines Strahls mit hoher Phasenraumdichte für Experimente mit großer Genauigkeit. Dem sollen zwei verschiedene Methoden der Strahlkühlung dienen: Elektronenkühlung ([1], [2]) und stochastische Kühlung ([3])[1]. Die angestrebte Emittanz ist 2.5 bis 1 π mm mrad, die Impulsunschärfe $\frac{\Delta p}{p} \approx 5 \cdot 10^{-4}$.

Der Grundstein für das Synchrotron wurde am 7. Juli 1988 gelegt. Ende 1993 konnte mit der Inbetriebnahme erster Experimente begonnen werden ([5], [6]). Zur Zeit werden Protonen von der Einschußenergie (40MeV) bis auf maximal 2.5 GeV beschleunigt. Strahlkühlung ist derzeit noch nicht möglich.

Ein Grundriß der gesamten Anlage ist in Bild 1.1 gezeigt. Das am unteren Rand sichtbare Isochronzyklotron JULIC beschleunigt die von einer Ionenquelle kommenden Teilchen bis zur Einschußenergie des Synchrotronringes. Der dort auf Endenergie gebrachte Strahl kann an internen Strahlplätzen genutzt oder über den Extraktionskanal zu einem der drei externen Zielpunkte geleitet werden. Bis zu $2 \cdot 10^{11}$ Teilchen können im Ring gespeichert werden (beschränkt durch die Phasenraumgrenze), $5 \cdot 10^9$ Teilchen/s sollen extrahiert werden können.

Einer der externen Experimentplätze ist das Magnetspektrometer BIG KARL[2]. Von einem dort installierten Experiment und dem dafür entwickelten Datenaufnahmesystem wird im folgenden die Rede sein.

Der erste Abschnitt stellt das GEM-Experiment an COSY vor. Besonders werden die für die Datenaufnahme relevanten Fragen behandelt, so die Detektoren und die kinematischen Bedingungen.

In folgenden Kapitel werden die dem Entwurf des neuen Datenaufnahmesystems zugrundeliegenden Gedanken erläutert. Das beginnt mit einer Einführung in die Prinzipien des objektorientierten Entwurfs und in die begrifflichen Grundlagen der Kommunikationstechnik. Darauf aufbauend werden die logi-

[1] Eine einführende Erklärung zu COSY findet sich in [4].

[2] nach einem Umbau BIG KARL II genannt

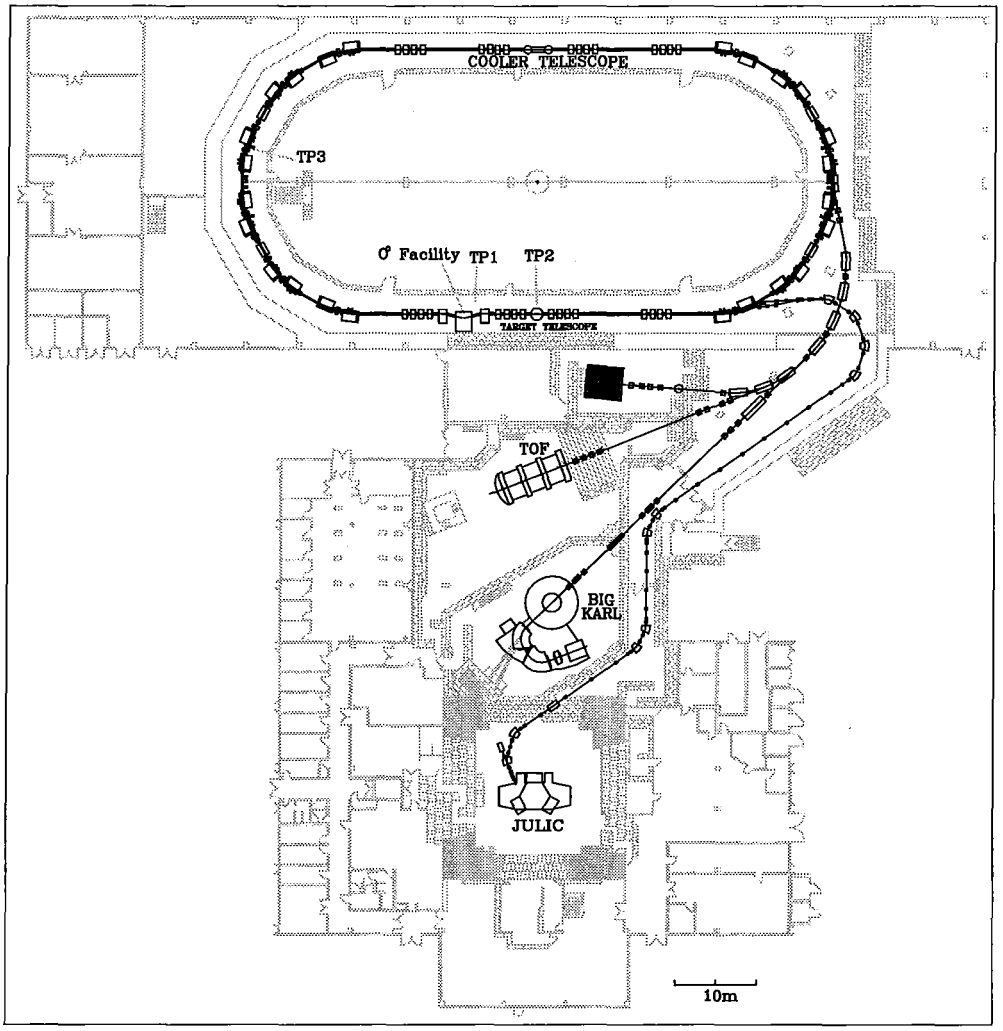


Abbildung 1.1: Der Jülicher Beschleuniger COSY.

sche Struktur des Systems und der Aufbau des Steuerungsprotokolls beschrieben.

Kapitel 3 zeigt, wie auf dem neuentwickelten Konzept aufbauend ein Datenaufnahmesystem für den Einsatz im GEM-Experiment realisiert wurde. Hier werden auch einige wichtige, durch das Kommunikationsprotokoll nicht abgedeckte, Spezifika erläutert.

Der letzte Abschnitt handelt vom Einsatz des neuen Systems im Experiment. Zuerst wird die konkrete Konfiguration der Hardware und deren Widerspieglung in logischen Strukturen des Datenaufnahmesystems beschrieben. Nach einer Übersicht über die Methoden der Auswertung folgen dann einige während einer Strahlzeit im Dezember 1994 gewonnene Resultate, wiederum mit Schwerpunkt auf die die Datenaufnahme betreffenden Fakten.

1.1 Die Reaktion $pp \rightarrow \pi^+ + d$

Die Erzeugung von Pionen bei Proton-Proton-Zusammenstößen wurde erstmalig 1950 beschrieben [8]. Der Nachweis von koinzidenten Deuteronen bestätigte, daß dabei die Reaktion

$$pp \rightarrow \pi^+ d$$

stattfindet.

Seitdem wurden in verschiedenen Messungen Wirkungsquerschnitte für diesen Prozeß ermittelt, sowohl mit polarisiertem als auch mit unpolarisiertem Strahl. ([9], [10])

Die meisten dieser Experimente wurden mit Zweiarmspektrometern durchgeführt, es wurden also nur jeweils einzelne Streuwinkel vermessen. Für den absoluten Wirkungsquerschnitt mußten die einzelnen differentiellen Querschnitte an eine Formel für die Winkelverteilung gefittet werden.

Zunächst wurden dafür empirische, phänomenologische Ansätze verwendet, später etablierte sich die Beschreibung durch LEGENDREsche Polynome ([11]). Wegen der Symmetrie der Reaktion (im Schwerpunktsystem) — die Protonen sind ununterscheidbar — treten keine Terme ungeradzahliger Ordnung auf.

$$4\pi \cdot \frac{d\sigma}{d\Omega} = a_0 + a_2 P_2(cos\theta^*) + \cdots \tag{1.1}$$

Hierbei ist θ^* der Streuwinkel im Schwerpunktsystem.

Die Messung unter kleinen Streuwinkeln wird dadurch erschwert, daß der primäre Protonenstrahl den Nachweis der Deuteronen behindert. Besonders schwellennahe Experimente leiden unter diesem Handikap.

Verschiedene Messungen wurden auch für verwandte Reaktionen durchgeführt. Dies sind besonders die Umkehrreaktion

$$\pi^+ d \rightarrow pp$$

[12], [13], [14] und der durch Ladungsunabhängigkeit verwandte Prozeß

$$np \rightarrow \pi^0 d$$

[15]. Die Resultate dieser Experimente erlauben nach den Regeln der Zeitumkehr bzw. der Ladungsunabhängigkeit Rückschlüsse auf die von uns betrachtete Reaktion (siehe besonders [12]).

Die zur Zeit verfügbaren Daten für den totalen Wirkungsquerschnitt sind in Bild 1.2 zusammengestellt. Für den schwellennahen Bereich liegen nur wenige Daten vor, und diese sind relativ älteren Datums.

Der totale Wirkungsquerschnitt wird in der Nähe der Schwelle mit einem Polynom

$$\sigma = \alpha_1 \cdot \eta + \alpha_2 \cdot \eta^3 \tag{1.2}$$

beschrieben. Dabei ist

$$\eta = \frac{p^*}{m_{\pi^+} \cdot c} \tag{1.3}$$

(p^* sei der Impuls im Schwerpunktsystem).

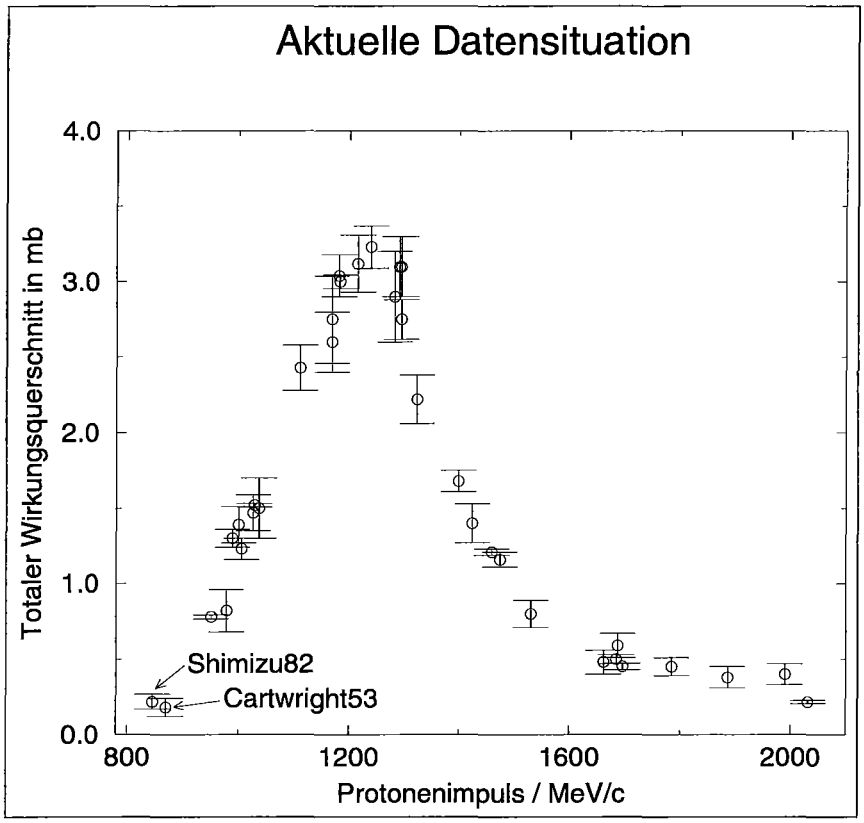


Abbildung 1.2: Derzeitig verfügbare Daten für den totalen Wirkungsquerschnitt von $pp \to d\pi^+$

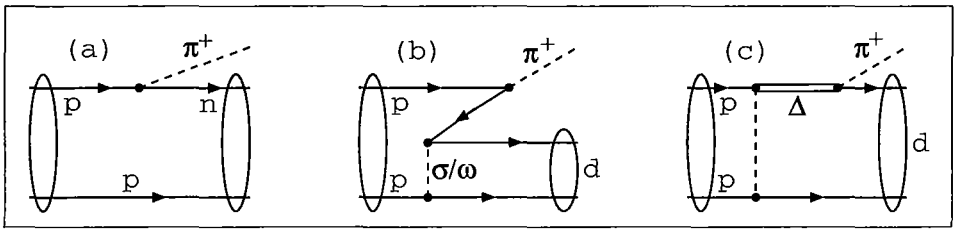


Abbildung 1.3: Beiträge zu $pp \to d\pi^+$ (a) einfach, (b) Austausch eines schweren Mesons, (c) intermediäres Δ

Auch die Winkelverteilung ist von theoretischem Interesse. Zu klären ist, welche Anteile an der Reaktion der Austausch eines schweren Mesons [16] und eine intermediäre Δ-Anregung [17] haben. Abb. 1.3 zeigt die Beiträge zum betrachteten Prozeß.

Der COSY-Strahl mit seiner hohen Phasenraumdichte ermöglicht Messungen mit einer (gerade in Schwellennähe wichtigen) hohen Genauigkeit. Das Magnetspektrometer BIG KARL gibt die Möglichkeit, nahe der Reaktionsschwelle eine 4π-Messung (im Schwerpunktsystem) durchzuführen, also Deuteronen mit allen kinematisch zulässigen Energien und Streuwinkeln nachzuweisen. Die Re-

aktionskinematik bringt es mit sich, daß die Impulse der Deuteronen nahe am Strahlimpuls liegen (vgl. Bild 1.7). So besteht auch hier das Problem, daß der Primärstrahl mit relativ hoher Intensität in der Nähe der erzeugten Deuteronen eintrifft. Die verwendete Meßanordnung mit Driftkammern als ortsauflösenden Detektoren in der Fokalebene erlaubt aber, den Einfluß dieser Störung gering-zuhalten.

1.2 Kinematik

1.2.1 Generelles

Die Experimente an BIG KARL entsprechen dem gebräuchlichen Aufbau, in dem ein vom Beschleuniger kommendes Projektil auf ein ruhendes Target trifft. Die Teilchenmasse des Targets sei mit m_T bezeichnet, die Masse des Projektils mit m_P, sein Impuls mit p.

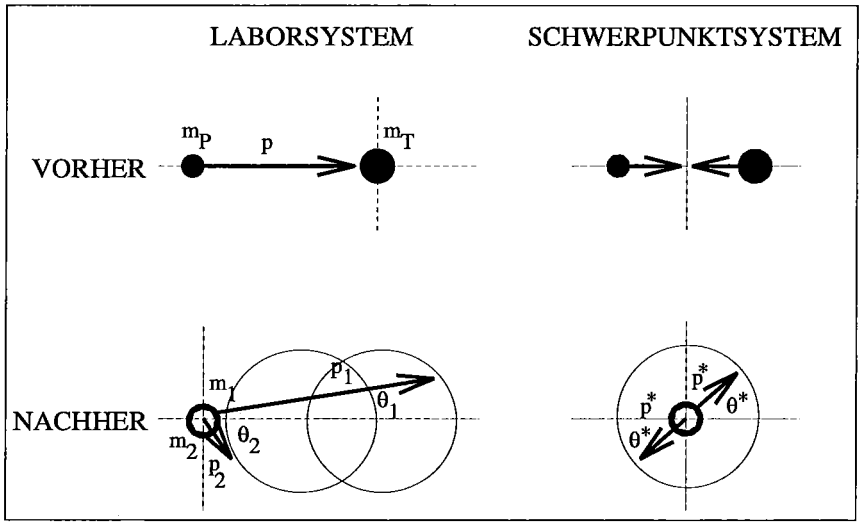


Abbildung 1.4: Bezeichnungen für die kinematischen Größen.

Größen im Schwerpunktsystem sind mit einem Stern (*) bezeichnet. Das Verhältnis zwischen dem Labor- und dem Schwerpunktsystem wird durch mit „CM“ indizierte Symbole ausgedrückt. Alle anderen Energien, Impulse und Winkel beziehen sich auf das Laborsystem.

Im Schwerpunktsystem steht die Energie

$$E^* = c \cdot \sqrt{m_P^2 c^2 + m_T^2 c^2 + 2 \cdot m_T c \cdot \sqrt{m_P^2 c^2 + p^2}} \tag{1.4}$$

zur Verfügung.

Vom Laborsystem zum Schwerpunktsystem führt eine LORENTZ-Transformation mit der relativen Geschwindigkeit

$$\beta_{CM} = \frac{p}{\sqrt{m_P^2 c^2 + p^2} + m_T c}. \tag{1.5}$$

Es wird der Fall von zwei entstehenden Teilchen betrachtet. Ihre (Ruhe-) Massen seien m_1 und m_2. Die Beträge ihrer Impulse sind im Schwerpunktsystem gleich, ihre Richtungen genau entgegengesetzt

$$p^* = \frac{\sqrt{\left(E^{*2} - (m_1 + m_2)^2 c^4\right) \cdot \left(E^{*2} - (m_1 - m_2)^2 c^4\right)}}{2 \cdot E^* \cdot c}. \tag{1.6}$$

Der von einem dieser Teilchen und der Strahlachse eingeschlossene Winkel sei θ. Ohne die Allgemeinheit einzuschränken, sei im folgenden der Winkel zwischen dem Teilchen der Masse m_1 und der Strahlachse (in Strahlrichtung) hiermit bezeichnet.

Die Winkel im Labor- und im Schwerpunktsystem lassen sich mit Hilfe der folgenden Beziehungen ineinander umrechnen:

$$\tan\theta = \frac{\sin\theta^*}{\gamma_{CM} \cdot \left(\cos\theta^* + \frac{\beta_{CM}}{\beta^*}\right)} \tag{1.7}$$

$$\tan\frac{\theta^*}{2} = \frac{1 \pm \sqrt{1 + \tan^2\theta \cdot \gamma_{CM}^2 \cdot \left(1 - \left(\frac{\beta_{CM}}{\beta^*}\right)^2\right)}}{\tan\theta \cdot \gamma_{CM} \cdot \left(\frac{\beta_{CM}}{\beta^*} - 1\right)} \tag{1.8}$$

Die Geschwindigkeit des betrachteten gestreuten Teilchens in Schwerpunktsystem ist hier, da unabhängig vom Streuwinkel, mit β^* abgekürzt. Sie ergibt sich aus der Beziehung

$$\beta^* = \left(\frac{pc}{E}\right)^* = \frac{p^*}{\sqrt{p^{*2} + m_1^2 c^2}}. \tag{1.9}$$

1.2.2 Zur Geometrie der Detektoren

Für den Streuwinkel im Laborsystem (Gleichung 1.7) existiert ein Maximum, falls das betreffende Teilchen nicht rückgestreut wird.

Um alle Teilchen im Ausgangskanal nachweisen zu können, muß der Akzeptanzbereich eines Detektors mindestens diesen maximalen Streuwinkel erfassen:

$$\tan\theta_{max} = \frac{1}{\gamma_{CM} \cdot \sqrt{\left(\frac{\beta_{CM}}{\beta^*}\right)^2 - 1}}. \tag{1.10}$$

Das Maximum wird bei einem Streuwinkel im Schwerpunktsystem von

$$\cos\theta^* = -\frac{\beta^*}{\beta_{CM}} \tag{1.11}$$

erreicht.

Für die untersuchte Reaktion $p+p \rightarrow d+\pi^+$ sind die maximalen Streuwinkel der π^+ in Bild 1.5 dargestellt.

Bild 1.6 zeigt eine analoge Darstellung für die Streuwinkel der Deuteronen.

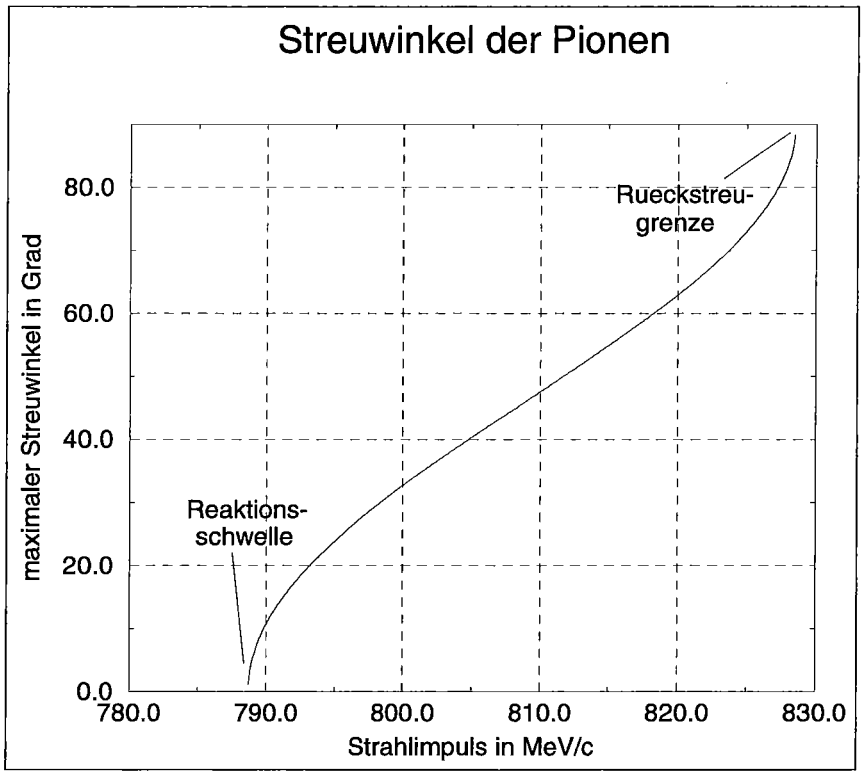


Abbildung 1.5: Maximale Streuwinkel der π^+ in Abhängigkeit vom Strahlimpuls.

1.2.3 Impulse, Energien und Flugzeiten

Die Kinematik beschränkt auch den Energiebereich der Teilchen im Ausgangskanal. Für die gestreuten Deuteronen zeigt Bild 1.7 die obere und untere Schranke.

Der Abstand zwischen der Streukammer und der Fokalebene beträgt ca. $s = 15m$.

Die Flugzeit relativistischer Teilchen ergibt sich aus

$$\beta = \frac{pc}{E} = \frac{p}{\sqrt{m^2c^2 + p^2}}, \tag{1.12}$$

also

$$t = \frac{s}{\beta c} = \frac{s \cdot \sqrt{m^2c^2 + p^2}}{pc}. \tag{1.13}$$

Für Protonen mit einem Impuls von 800 MeV/c erhält man eine Flugzeit von 77 ns. Da gestreute Deuteronen unterschiedliche Energien haben, ist deren Flugzeit über einen gewissen Bereich verteilt. Zu erwarten sind Werte etwa zwischen 127 ns (bei 800 MeV/c) und 143 ns (bei 700 MeV/c).

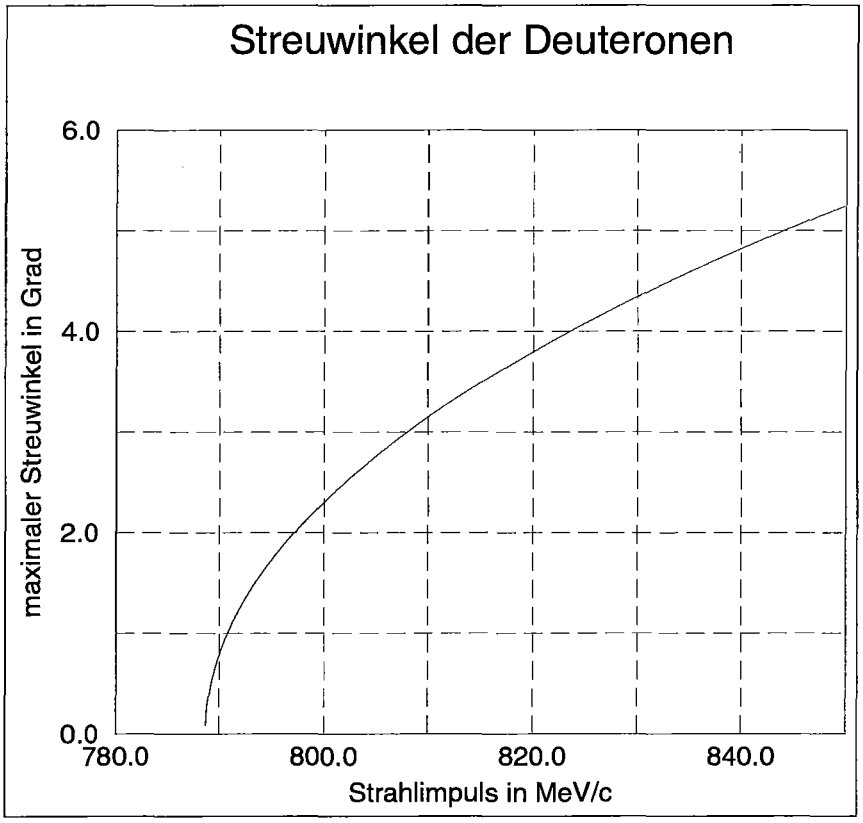


Abbildung 1.6: Maximale Streuwinkel der d in Abhängigkeit vom Strahlimpuls.

1.3 Energieverlust der Teilchen in Materie

Die Genauigkeit der Energie- und Impulsbestimmungen wird unter anderem von der durch den Energieverlust der Teilchen in der durchlaufenen Materie (in Form von Target, Luft, Folien oder Detektoren) bewirkten Unschärfe (straggling) begrenzt. Winkelmessungen unterliegen einer ähnlichen, durch Kleinwinkelstreuung bewirkten, Ungenauigkeit.

Auf der anderen Seite wird das Wissen über den Energieverlust der Teilchen benötigt, um Rückschlüsse auf die Teilchenart oder -energie zu gewinnen (entweder aus der Größe des Energieverlusts in einem Detektor oder aus der Tatsache, ein bestimmtes Medium überhaupt durchdringen zu können).

Der Energieverlust geladener Partikel beim Durchlaufen von Materie wird durch die BETHE-BLOCH-Formel beschrieben:

$$-\frac{dE}{dx} = \rho K z^2 \frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_e c^2\beta^2\gamma^2 T_{max}}{I^2} - \beta^2 - \frac{\delta}{2}\right] \qquad (1.14)$$

Dabei sind:

ρ : Dichte des durchlaufenen Materials

K : konstanter Faktor $\frac{4\pi r_e^2 m_e c^2}{m_p}$, r_e ist der klassische Elektronenradius $\frac{e^2}{4\pi\epsilon_0 c^2}$

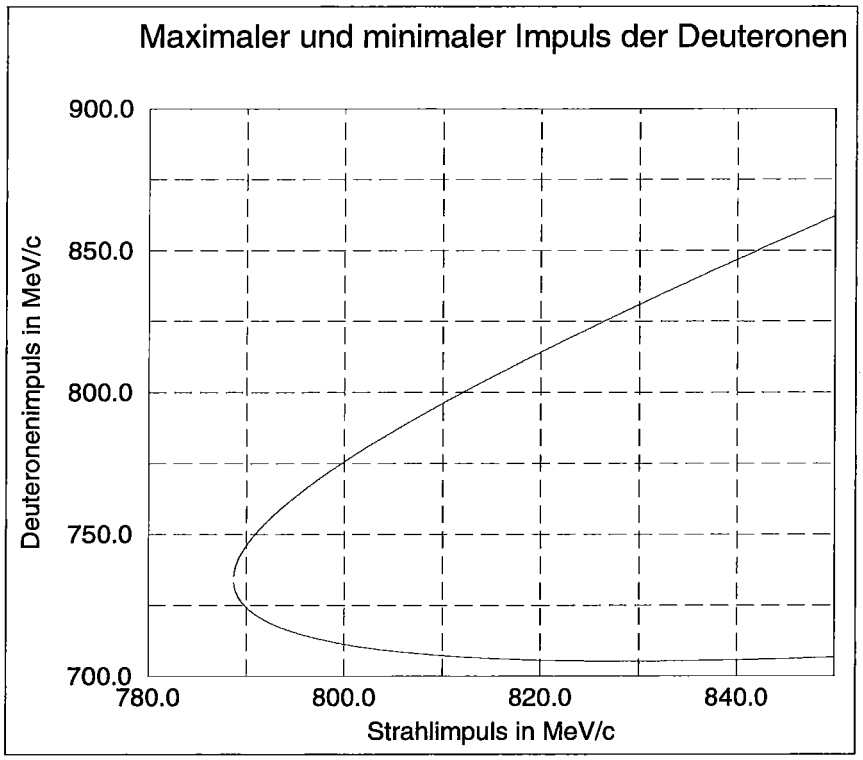


Abbildung 1.7: Kinematisch zulässiger Impulsbereich der gestreuten d in Abhängigkeit vom Strahlimpuls.

z : Ladungszahl des passierenden Teilchens

Z, A : Ladungszahl bzw. Massezahl des durchlaufenen Materials

T_{max} : maximale kinetische Energie, die durch einen einfachen Stoß vom passierenden Teilchen an ein Elektron abgegeben werden kann:

$$T_{max} = \frac{2m_e c^2 \beta^2 \gamma^2}{1 + 2\gamma\frac{m_e}{M} + \left(\frac{m_e}{M}\right)^2}$$

Hier ist M die Masse des passierenden Teilchens.

I : mittlere Anregungsenergie der Atome der durchlaufenen Materie (Tabellenwert)

δ : Korrektur von Polarisationseffekten

Reichweiten von Teilchen können durch Integration von Gleichung (1.14) berechnet werden.

Die Winkelunschärfe kann durch

$$\Theta_0 = \frac{13.6MeV}{\beta cp} z \sqrt{\frac{x}{X_0}} \left[1 + 0.038 \ln\left(\frac{x}{X_0}\right)\right] \qquad (1.15)$$

abgeschätzt werden.

Hier sind:

Θ_0 : Mittlere Abweichung (RMS) des Winkels von der Strahlmitte, wenn die Projektion des Streukegels auf eine zum Strahl parallele Ebene betrachtet wird.

X_0 : „Strahlungslänge", näherungsweise durch

$$X_0 = \frac{716.4 \frac{g}{cm^2} A}{\rho Z(Z+1) \ln\left(\frac{287}{\sqrt{Z}}\right)}$$

beschrieben.

In der Praxis können die benötigten Werte einem Tabellenwerk (z.B. [18]) entnommen werden.

Beim $pp \to d\pi^+$ - Experiment ist, wie schon erwähnt, die Unterscheidung von Primärstrahlprotonen und den erzeugten Deuteronen wichtig. Hier wird die unterschiedliche Reichweite dieser Teilchen in Aluminium ausgenutzt. Eine grobe Abschätzung anhand von Gl. 1.14 ergibt den vierfachen Energieverlust für Deuteronen, bei doppelter kinetischer Energie also die halbe Reichweite.

Eine Wand, die etwas dünner als die Reichweite der Primärstrahlprotonen ist, stoppt also die nachzuweisenden Deuteronen. Ein dahinter angebrachter Detektor liefert nur dann einen Impuls, wenn ein Proton eintrifft, und kann zur Erzeugung eines VETO-Signals für die Experimentelektronik genutzt werden.

1.4 Reaktionsrate

Die Dichte der Streuzentren im Flüssigwasserstofftarget ist

$$\begin{aligned} \frac{N_S}{\Delta V} &= \rho_{lH_2} \cdot \frac{N_A}{M_{H_2}/2} \\ &= 5 \cdot 10^{22} cm^{-3}. \end{aligned} \tag{1.16}$$

Für die Reaktion $pp \to d\pi^+$ ist bei 800 MeV/c nach der Literatur ([15]) ein totaler Wirkungsquerschnitt im Bereich um 50 μb zu erwarten.

Für ein 4 mm dickes Target erhält man damit eine Wahrscheinlichkeit für die Einzelreaktion von

$$\begin{aligned} \omega &= \frac{N_S}{\Delta V} \cdot \sigma \cdot \Delta x \\ &= 10^{-6}. \end{aligned} \tag{1.17}$$

Bei einer mittleren Strahlintensität von einigen 10^6 Teilchen pro Sekunde ist somit mit einer Ausbeute von einigen der gesuchten Reaktionen pro Sekunde zu rechnen. Die Leistungsfähigkeit des Datenaufnahmesystems stellt bei dieser Rate und bei ausreichender Reduzierung des Untergrundes keinen beschränkenden Faktor dar.

1.5 Detektoranordnung

Die Detektoranordnung wurde mit dem Ziel entworfen, dem Ideal einer kinematisch vollständigen Messung möglichst nahe zu kommen. Dafür empfiehlt sich eine zweckmäßige Kombination von einzelnen Detektoren, die unter Berücksichtigung der kinematischen Konfiguration des Experiments und der Teilchenarten eingesetzt werden.

Bei einem Beschleunigerexperiment mit stationärem Target steht die Vorwärtsstreurichtung im Vordergrund. Wenn schwellennahe Produktionen stattfinden, ist Rückwärtsstreuung aus Gründen der Kinematik ausgeschlossen. Weiter ist bei der Konzeption zu berücksichtigen, daß auch der Primärstrahl (mit einer vergleichsweise hohen Intensität) unter 0° aus dem Targetbereich austritt.

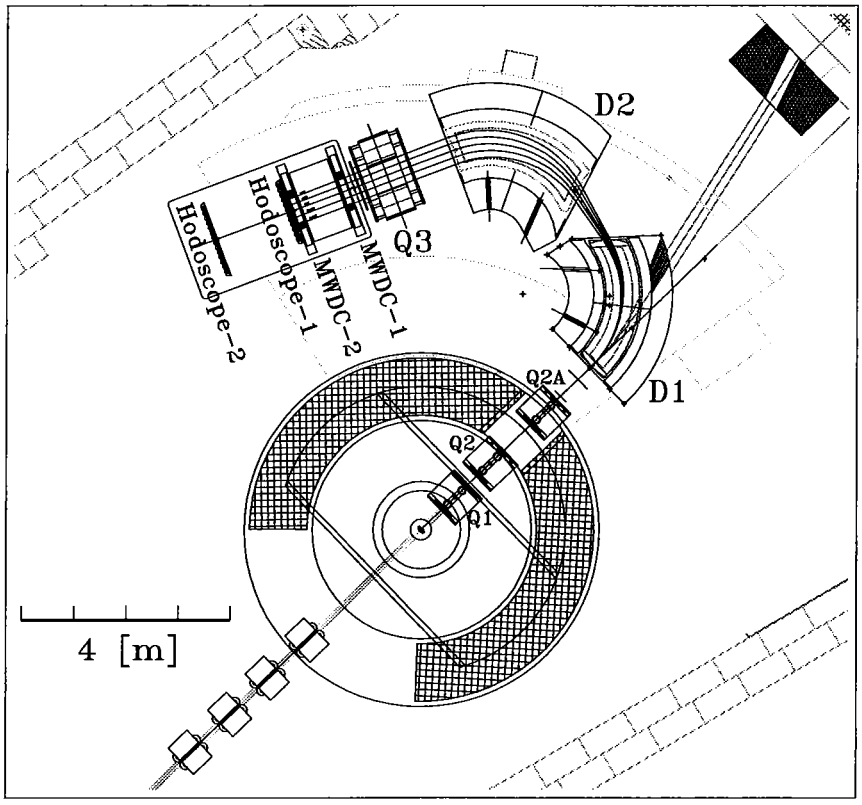


Abbildung 1.8: Grundriss des Magnetspektrometers BIG KARL mit den Detektoren in der Fokalebene.

Die unter kleinen Winkeln gestreuten Teilchen können mit Hilfe des Magnetspektrometers BIG KARL analysiert werden. Der Analysatormagnet besteht aus 2 Dipolen und 3 Quadrupolen, deren Anordnung ist in Bild 1.8 dargestellt. Die Winkelakzeptanz ist vom Azimutwinkel und vom Teilchenimpuls abhängig, sie beträgt ca. 2.9° in Horizontal- und 11.5° in Vertikalrichtung. Die Teilchen werden nach dem Verhältnis Impuls/Ladung analysiert, es kann ein Bereich von etwa 8% abgedeckt werden (Dispersion: 6.24 cm/%). Die Flugstrecke vom Target zur Fokalebene beträgt etwa 15m.

In der Fokalebene des Magnetspektrometers befindet sich eine Anordnung von Vieldraht-Driftkammern, die die Bestimmung von Ort und Richtung des Durchganges eines geladenen Teilchens ermöglicht. Aus diesen Informationen läßt sich mit Hilfe einer geeigneten Übertragungsfunktion der Streuwinkel am Target rekonstruieren.

Eine Kombination von Szintillatorwänden in der Fokalebene dient zur Erzeugung der schnellen Triggersignale sowie zur Gewinnung zusätzlicher Energie- und Flugzeitinformationen.

Für Teilchen mit größerem Streuwinkel, als der Akzeptanzbereich von BIG KARL umfaßt, sind Detektoren in der Streukammer vorgesehen. Bei den beschriebenen Messungen wurde nur ein ringförmiger Szintillator (Innendurchmesser 4mm) verwendet. Somit sind hier keine Ortsinformationen verfügbar. In der Zukunft wird an diese Stelle der Germaniumdetektor treten, nach dem das Experiment benannt wurde[3]. Durch dessen Einsatz werden dann vollständige Orts- und Energiedaten für die missing-mass-Analyse vorhanden sein.

Die Gesamtanordnung ist in Bild 1.9 dargestellt.

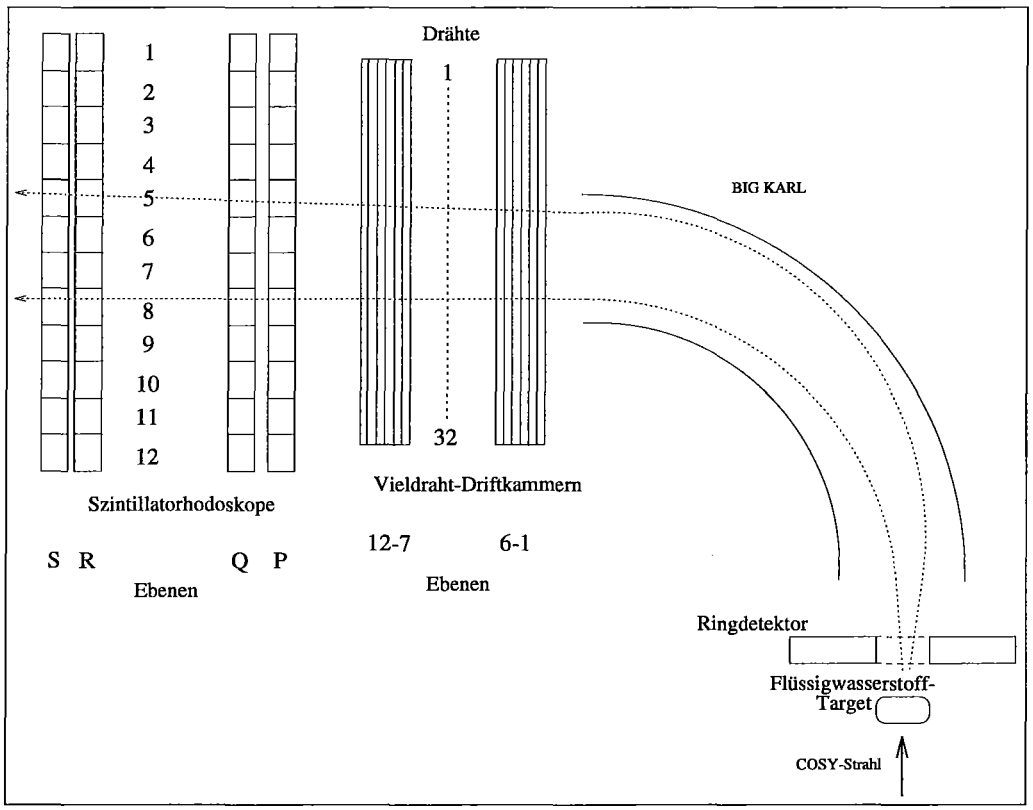


Abbildung 1.9: Grundanordnung der Detektoren in den GEM-Experimenten und Zuordnung der Kanalnummern.

Bild 1.10 zeigt einige Details des Aufbaus in der Fokalebene.

[3]GEM = **GE**rmanium + **M**agnet

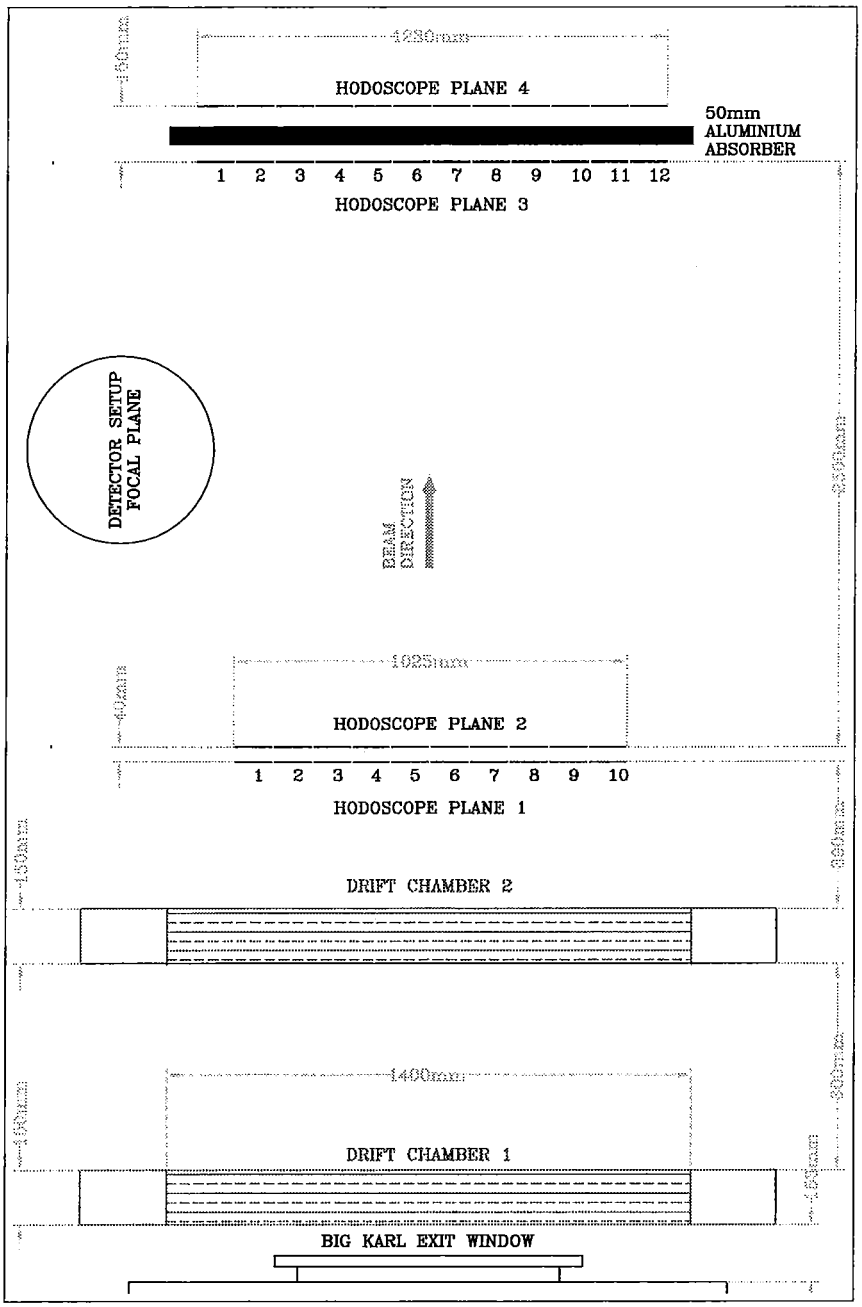


Abbildung 1.10: Drahtkammern und Szintillatoren in der Fokalebene von BIG KARL.

1.5.1 Drahtkammern

Für die genaue Orts- und Winkelbestimmung der Teilchen in der Fokalebene wurde eine Anordnung von Vieldrahtdriftkammern (MWDC) installiert. Als Füllgas dient eine Argon/Ethan-Mischung (Verhältnis 50:50, mit Ethanolzusatz). Die Driftgeschwindigkeit ist bei diesem Mischungsverhältnis im

Feldstärkebereich um 1 kV/cm maximal, die Abhängigkeit der Driftgeschwindigkeit von der Feldstärke relativ klein. Die Driftgeschwindigkeit beträgt ca. 50 μm/ns, bei einer Genauigkeit der Zeitmessung von 1 ns wird so die Ortsauflösung auf 50 μm begrenzt. Die tatsächlich erreichbare Auflösung wurde bei Vorexperimenten zu ca. 220 μm (FWHM) ermittelt. Die Nachweiswahrscheinlichkeit für Protonen im verwendeten Energiebereich liegt bei geeigneter Wahl der Betriebsspannung bei mehr als 99% pro Ebene; bei 12 Ebenen kann damit gerechnet werden, daß etwa 90% der Teilchen in jeder Ebene einen Treffer hinterlassen. Laut Beschreibung [19] sollen die Driftkammern mehrere beieinanderliegende Spuren bis zu einen minimalen Abstand von 3 mm auflösen können.

Insgesamt sind 12 Lagen mit je 32 Drähten vorhanden, jeweils 6 Lagen sind zu einem Block zusammengefaßt. Die beiden Blöcke stehen in einem Abstand von ca. 1m zueinander. Innerhalb eines Blockes sind jeweils 2 Lagen in 3 verschiedenen Ausrichtungen installiert (Drahtrichtung vertikal und um 31° in jede Richtung geneigt). Bild 1.11 zeigt schematisch den Aufbau eines solchen

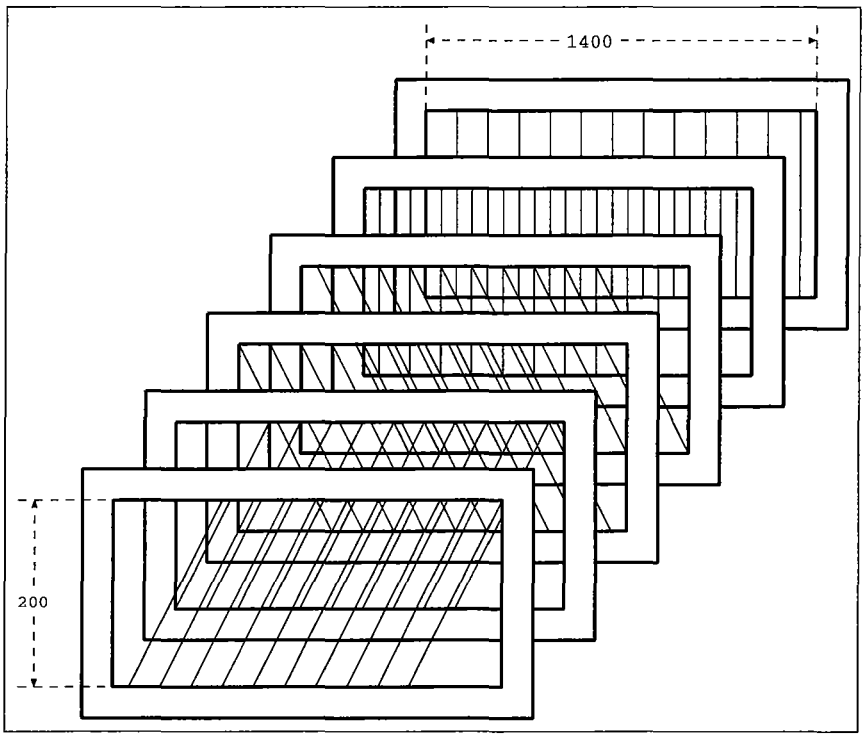


Abbildung 1.11: Aufbau einer Hälfte des Driftkammerdetektors.

Blockes. Eine Driftzelle ist 4 cm breit, so daß die maximale Driftstrecke 2 cm beträgt. Die beiden jeweils gleich ausgerichteten Lagen sind gegeneinander um 1/2 Zellenbreite versetzt, um die Auflösung der Links-Rechts-Mehrdeutigkeit zu ermöglichen.

Die Teilchenbahn wird aus den einzelnen Driftzeiten rekonstruiert, indem in mehreren Iterationsschritten eine Gerade gesucht wird, für die die Summe der

Fehlerquadrate

$$\sum_i (\Delta x_i)^2 \tag{1.18}$$

minimal ist. (Δx_i ist der Abstand zwischen gemessenem und berechnetem Durchstoßpunkt der Bahn durch die Ebene i.) Um unsinnige Resultate zu verhindern, werden Spuren, für die ein Δx einen gewissen Maximalwert überschreitet (z.B. 5 mm), nicht berücksichtigt.

1.5.2 Szintillatoren in der Fokalebene

In der Fokalebene von BIG KARL sind 4 Lagen von Szintillatoren angeordnet. Jede dieser Lagen besteht aus 12 senkrechten Streifen, die ohne Überlappung nebeneinander angebracht sind. Jeweils 2 Lagen sind dicht hintereinander angeordnet, zwischen diesen Paaren besteht ein Abstand von ca. 1 m.

Die Szintillatorstreifen innerhalb eines Lagenpaares sind so angeordnet, daß jeweils 2 Streifen in Strahlrichtung genau hintereinander liegen. Das erlaubt, zur Reduzierung des Untergrundes eine einfache Koinzidenzschaltung einzusetzen.

Für die Meßruns wurde die in Strahlrichtung letzte Szintillatorlage von den vorherigen Lagen durch eine 50mm dicke Aluminiumplatte getrennt. Diese wurde so bemessen, daß sie von den Protonen des Primärstrahls passiert wird, und die nachzuweisenden Deuteronen (mit einem Impuls in der gleichen Größenordnung) gestoppt werden.

Auf diese Weise kann aus dem Summensignal der vierten Szintillatorlage ein Signal erzeugt werden, das die Aufzeichnung von Protonenbahnen mittels VETO verhindert.

1.5.3 Detektoren in und an der Streukammer

Neben dem bereits in Bild 1.9 dargestellten Ringdetektor wurde in Strahlrichtung vor dem Target ein zweiter ringförmiger Szintillator montiert (Innen-$\oslash$ 3mm). Mit dessen Hilfe wird die Meßanordnung gesperrt (VETO), wenn ein Strahlproton zu weit außerhalb der Strahlachse eintrifft.

Zur Bestimmung der Intensität des Primärstrahls wurden zusätzlich zwei Detektoren an der Streukammer angebracht. Das Verhältnis der Zahl der von ihnen nachgewiesenen elastisch gestreuten Protonen zur Zahl der Primärteilchen ist für eine gegebene Meßanordnung konstant[4] und wurde durch Eichmessungen ermittelt. (Resultat: $1 : 2.9 \cdot 10^4$)

Bild 1.12 zeigt die Anordnung aller in und an der Streukammer angebrachten Detektoren.

1.6 Die Digitalisierungselektronik

Zur Digitalisierung der Detektorsignale werden Analog-zu-Digital-Konverter (ADCs) und Zeit-zu-Digital-Konverter (TDCs) verwendet. Dazu kommen noch

[4]solange sich keine Sättigungseffekte bemerkbar machen, was hier gegeben ist

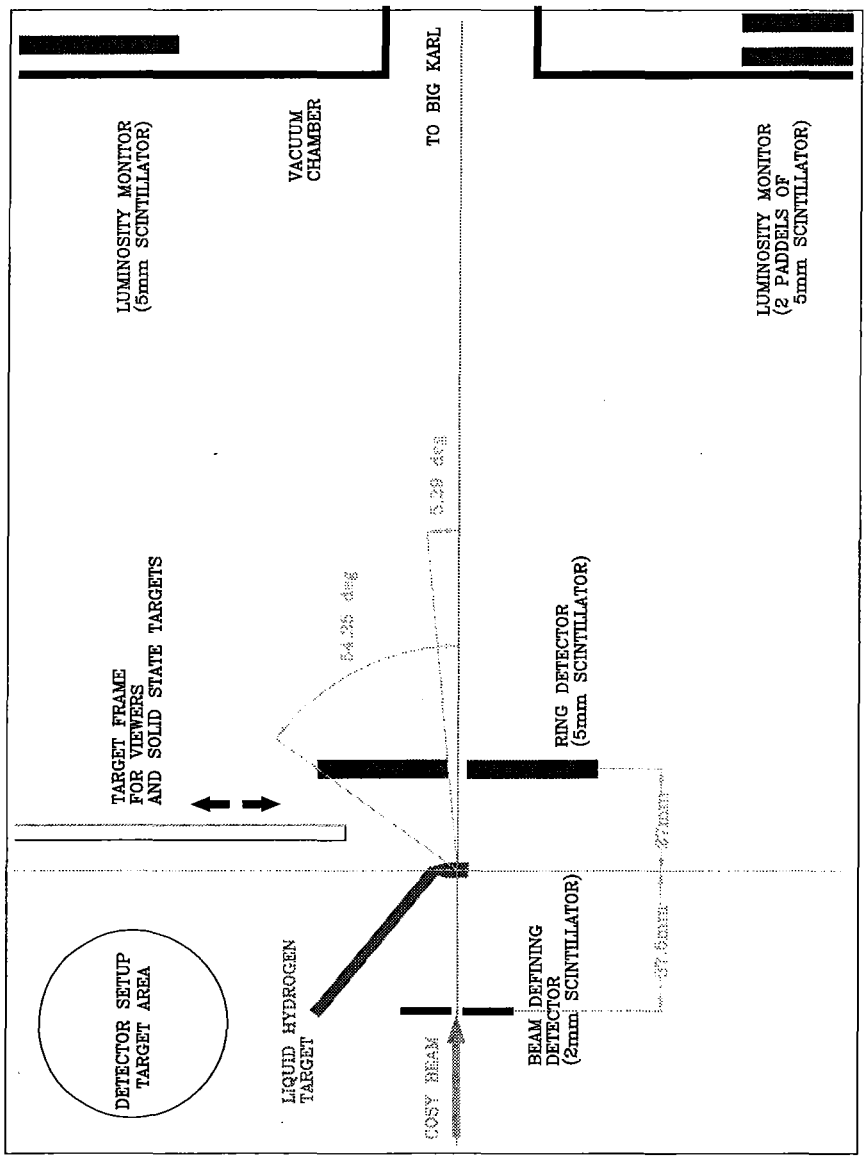


Abbildung 1.12: Geometrie der Detektoren in der Streukammer

Impulszähler (Scaler) zur Bestimmung aller nötigen Zählraten. Alle Digitalisierungsmodule sind im CAMAC-Standard gehalten.

Folgende Parameter werden von der Meßanordnung aufgenommen:

- Die Driftzeiten aller Treffer in den 12 Driftkammern. Dafür werden Multihit-TDCs (LeCroy 2277) mit je 32 Kanälen pro Modul verwendet (je 1 TDC-Modul pro Driftkammerlage).
- Energieverluste in den Szintillatoren der Fokalebene (4 x 12 Kanäle) sowie im Ringdetektor.
- Zeitdifferenzen der Impulse der genannten Szintillatoren relativ zum Triggerimpuls (siehe Bild 1.15 — t_1 tritt 12-fach, t_2 24-fach auf).

- Zählraten in einer Szintillatorlage, der Triggerimpulse, der aufgezeichneten Ereignisse und diverser Monitordetektoren.

Bild 1.13 zeigt ein Schema des Digitalisierungsteils, ohne die Ratenmesseng in den Scalern.

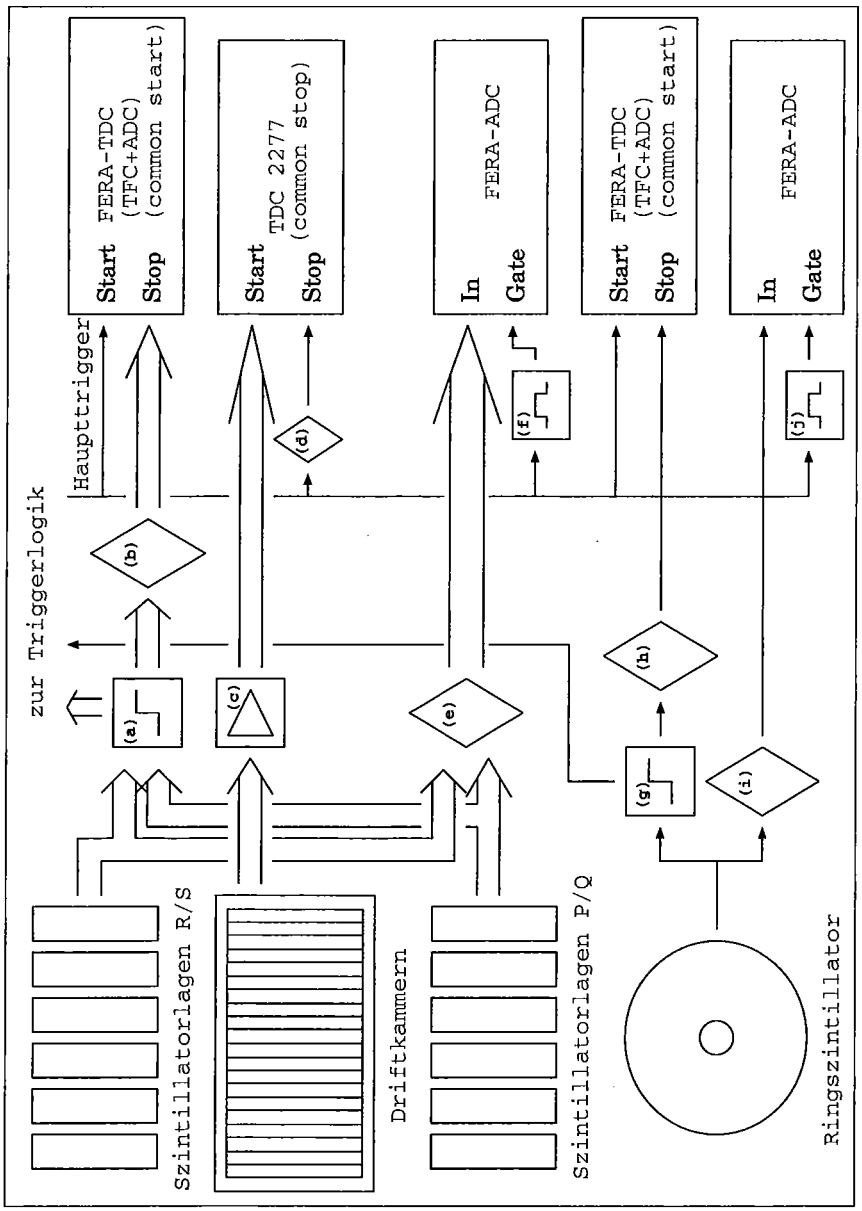


Abbildung 1.13: Aufbau der Digitalisierungselektronik für die Orts-, Energie- und Zeitkanäle. Erklärung der Symbole im Text.

Einige wichtige Einzelheiten sollen im folgenden genauer erklärt werden:

Die Signale der Szintillatoren werden jeweils in einen Analog- und einen Digitalzweig aufgespalten. Die Analogdaten werden durch Koaxialkabel (60m RG213, Symbol (e) in Bild 1.13, bzw. (i) für den Ringdetektor) verzögert und

in FERA-ADCs digitalisiert. Die Gatesignale werden aus dem Haupttrigger erzeugt. ((f) und (j) sind Gate- und Delaygeneratoren.) Die Triggerlogik wird im folgenden Abschnitt beschrieben.

Die Zeitsignale werden durch die Diskriminatoren (a) und (g) aus den Szintillatorimpulsen gewonnen. Für die Hodoskope der Fokalebene werden im ZEL der KFA Jülich entwickelte Diskriminatoren (16 Kanäle pro Modul, leading edge, fernsteuerbar [20]) eingesetzt (a), für den Ringszintillator, wie auch für die verschiedenen, hier nicht näher beschriebenen Monitordetektoren, kommen einzelne NIM-Module(g) zum Einsatz. Die Digitalisierung erfolgt durch FERA-TDC-Systeme (aus einem Zeit-Ladungs-Konverter und einem ADC bestehend).

Die 384 Zeitsignale der Driftkammern werden nicht verzögert, statt dessen werden die zugehörigen TDCs im „common stop mode" betrieben. Das heißt, alle von den Driftkammern kommenden Signale erzeugen Startimpulse für den jeweiligen TDC-Kanal. Ein Triggerimpuls stoppt alle TDC-Module. Dabei garantiert die Verzögerung (d), daß nach dem den Trigger auslösendenden Treffer die maximale Driftzeit vor dem TDC-Stop abgelaufen ist. Die TDC-Module speichern die Zeitdifferenzen für maximal 16 Treffer pro Kanal innerhalb eines Zeitfensters von $64\mu s$ vor dem Stop-Signal.

1.7 Trigger

In die Triggerentscheidung fließen die Informationen aus den Szintillatorwänden der Fokalebene sowie der beiden Ringdetektoren in der Streukammer ein. Bild 1.14 zeigt die logische Verknüpfung.

Ein Ereignis wird signalisiert, wenn sowohl der Ringdetektor einen Treffer (durch ein π^+) erhält als auch die Szintillatoren der Fokalebene das Passieren eines Deuterons anzeigen. Der erste (in Strahlrichtung vor dem Target liegende) Ringdetektor dient als Strahlbegrenzer; bei einem Treffer durch ein weit außerhalb der Strahlachse[5] liegendes Teilchen wird der Trigger gesperrt.

Die Erkennung von Deuteronen in der Fokalebene beruht auf folgenden 3 Kriterien:

- Zwischen der dritten (R) und der vierten (S) Lage befindet sich ein ca. 5 cm dickes Aluminiumblech. Dieses ist so bemessen, daß Deuteronen mit ca. 800 MeV/c darin gestoppt werden, während Protonen passieren können (siehe Abschnitt 1.3). Treffer in der vierten Lage sperren den Trigger.

- Die verbleibenden 3 Lagen sind zu einer Koinzidenzbedingung verschaltet, wie sie durch die große UND-Bedingung in Bild 1.14 angedeutet wird. Die zeitliche Abfolge der Signale für ein „gutes" Ereignis (Pion im Ringdetektor und Deuteron in der Fokalebene) ist in Bild 1.15 dargestellt.

- Die Diskriminatorschwellen für die Szintillatoren der ersten Lage (P) sind so eingestellt, daß nur die (größeren) von Deuteronen erzeugten Signale einen Ausgangsimpuls bewirken.

[5]Die Maße der Detektoren sind Bild 1.12 zu entnehmen.

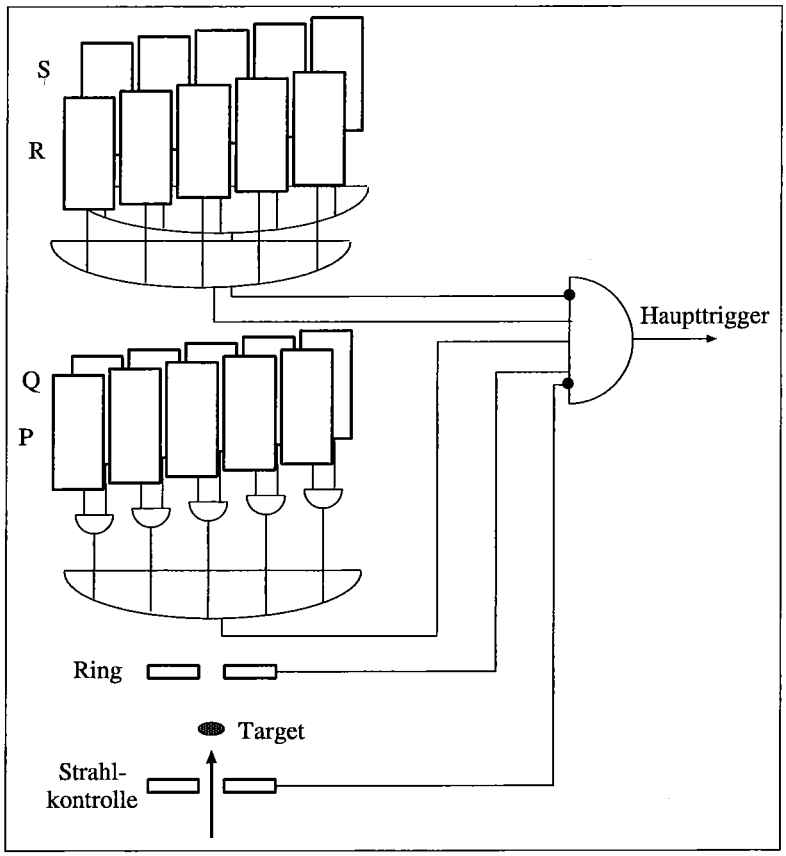


Abbildung 1.14: Prinzipschema des Triggersystems. Dargestellt werden die Teile der Triggerbedingung, Delays und Pulsformer wurden nicht berücksichtigt. Die Szintillatorlagen der Fokalebene wurden durch 5 Paddles (statt der 12 wirklich vorhandenen) angedeutet.

Dazu kommt die ebenfalls in Bild 1.15 sichtbare Flugzeitbedingung zwischen Ringdetektor und Fokalebene. Es wird die Tatsache ausgenutzt, daß (siehe Abschnitt 1.2.3) BIG KARL von Protonen in ca. 77 ns durchflogen wird, während die erzeugten Deuteronen etwa die doppelte Zeit benötigen.

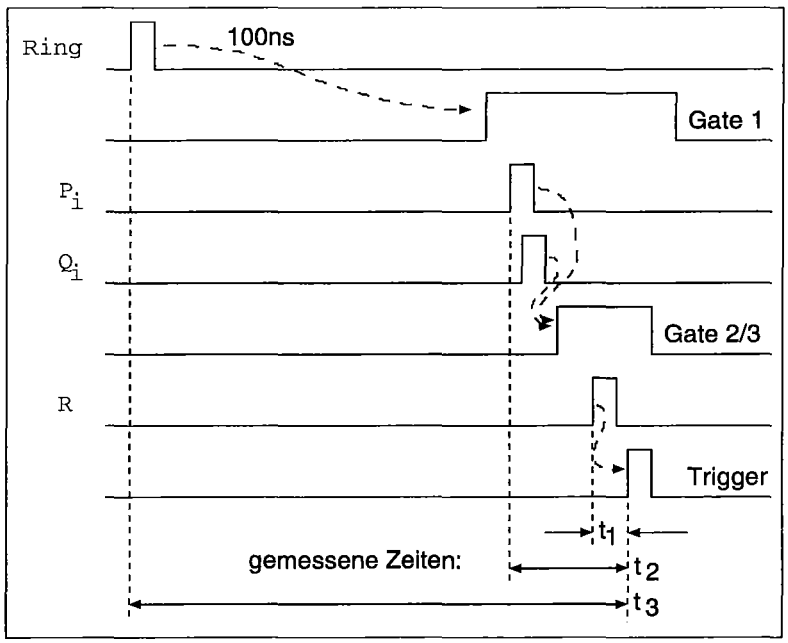


Abbildung 1.15: Zeitliche Abfolge der Signale im Triggersystem. Gate 1/2/3 sind Signale innerhalb des Triggersystems. Zusätzlich werden die von TDCs im Datenaufnahmeteil gemessenen Zeiten $t_1 \ldots t_3$ gezeigt.

Kapitel 2

EMS – ein objektorientiertes Konzept für die Steuerung von Datenaufnahmesystemen

2.1 Einführung

2.1.1 Warum noch ein Datenaufnahmesystem?

Für Experimente ähnlicher Größenordnung wie die an COSY existierte bereits eine Anzahl verschiedener Datenaufnahmesysteme Es wäre möglich gewesen, eines von diesen an die konkreten Anforderungen unserer Experimente anzupassen. Warum das nicht geschah und statt dessen beträchtlicher Aufwand in die Entwicklung eines weiteren Systems investiert wurde, soll hier kurz erläutert werden.

Zuerst sollen jedoch einige der bekannten und seit einiger Zeit verfügbaren Systeme mit ihren wesentlichen Merkmalen aufgelistet werden. Dabei werden vor allem Entwicklungen berücksichtigt, die zu einem möglichst großen Teil kommerziell erhältliche, standardkonforme Hardware verwenden, wie sie für die COSY-Experimente verwendet werden soll.

GOOSY („**G**SI-**O**nline-**O**ffline-**SY**stem") ist ein umfassender Ansatz für ein Datenerfassungssystem mit verteilter Intelligenz. Es wurde an der GSI Darmstadt ab 1983 entworfen und implementiert. Das Auslesen der Frontend-Hardware und das Zusammensetzen der Subevents zu Events erfolgt durch ein VME-Multiprozessor-System, das teils unter pSOS, teils unter OS-9 operiert. Ein VAX/VMS-Rechner dient zur Initialisierung, zur Experimentkontrolle sowie zur Datenaufbereitung.

Hardwarezugriffe auf CAMAC und FASTBUS erfolgen über eine symmetrische Variante von VSB („differential VSB", entwickelt an der GSI, Komponenten sind auch von Fa. Struck erhältlich). Der logische Datenfluß ist in Bild 2.1 dargestellt. Aus den dargestellten Komponenten lassen sich (durch Hinzufügen von Frontend-Prozessoren und CAMAC- bzw. Fastbus-Crates) auch komplexere Systeme zusammenbauen. Da es

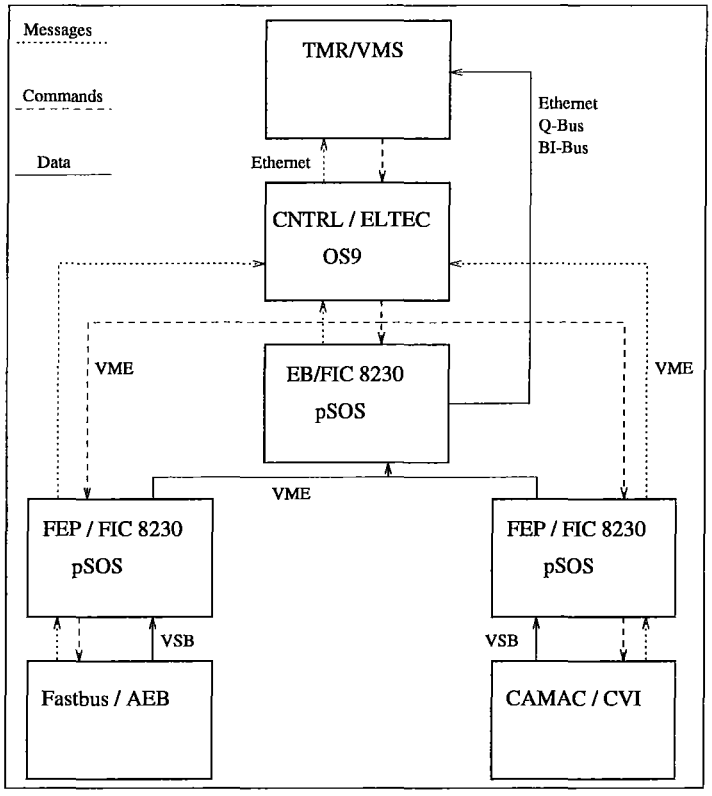


Abbildung 2.1: Systemstruktur und Datenfluß von GOOSY, nach [21] (TMR = Transport Manager, CNTRL = Control Processor, EB = Event Builder, FEP = Front End Processor)

sich um ein Multiprozessorsystem handelt, bei dem verschiedene Frontendprozessoren unabhängig voneinander die ihnen zugeordneten Module auslesen, stellt sich das Problem der Synchronisation. Hierfür wurde ein spezielles Triggermodul entwickelt, das die Annahme eines globalen Triggerimpulses nur dann ermöglicht, wenn alle Readoutprozessoren ihre Bereitschaft dazu gemeldet haben ([22]).

Vom GOOSY-System wurden viele konzeptionelle Anregungen für die neu zu entwickelnde, verteilte, Architektur gewonnen, ohne jedoch die Bindung an spezielle Hard- und Software zu übernehmen. Eine detailliertere Beschreibung des Systems ist in [23] zu finden, technische Einzelheiten (Prozesse, Schnittstellen, Datenformate) in [24].

TDAS wurde eigentlich als Vorstufe zu GOOSY geplant ([25]) (daher der Name „**T**emporary **D**ata **A**cquisition **S**ystem"). Das System wurde an der Uni Bochum und an der Uni Bonn weiterentwickelt und ist in einigen von diesen Gruppen betreuten Experimenten (an COSY: TOF, MOMO) im Einsatz.

Verwirklicht ist ein auf einer ELTEC-E6 VME-CPU unter OS-9 laufender

Acquisitionsprozeß. Dieser bietet die Möglichkeit, Eventdaten auf EXABYTE abzuspeichern oder über eine IP-Verbindung zur Onlineanalyse bereitzustellen. Teilaufgaben werden von separaten Prozessen abgewickelt.

Der Hardwarezugriff erfolgt, wie in GOOSY, über „differential VSB". Das Auslesen der Digitalisierungsmodule wird vom zentralen Acquisitionsprozeß auf der VME-CPU veranlaßt. Dieser kommuniziert dazu mit einem speziellen CAMAC-Interface-Modul („IO-LAM-Box"). Dieses simple, aber effektive Prinzip sorgt dafür, daß keine Synchronisationsprobleme auftreten können, beschränkt aber auch die Größe des Systems auf die Anzahl von Crates, die von einem VSB-Anschluß aus kontrolliert werden können[1].

Die verschiedenen TDAS-Versionen unterscheiden sich in der Art und Weise, in der die Anpassung an die Hardwarekonfiguration geschieht. Die Bochumer Variante ([26]) ist listengesteuert, d.h., alle Konfigurationsinformationen werden beim Start aus einer Textdatei eingelesen. Diese Datei enthält eine Liste der vorhandenen Hardware; die jeweils benötigten Routinen müssen beim Erstellen des Systems vorbereitet worden sein. Die Bonner Lösung verzichtet auf diese Form der Konfigurierbarkeit, da sie die Möglichkeit logischer Entscheidungen und arithmetischer Operationen zur Laufzeit für verschiedene, komplexere Hardware benötigt ([27]). Hier programmiert der Anwender die Hardwarezugriffe in „C" — durch entsprechende Systemfunktionen unterstützt.

SPIDER vom CERN ist ein Gerüst, mit dessen Hilfe sich der befähigte Anwender einfache Datenerfassungssysteme selbst zusammenstellen kann. Im wesentlichen wird die Verwaltung ringförmig organisierter Buffer abgedeckt, und eine Schnittstelle zur Steuerung und Kontrolle geboten. Die eigentlichen Steuerungs- und Ausleseprozeduren muß man selbst programmieren, wobei selbstverständlich vorbereitete Bibliotheken verwendet werden können [28]. Das System wird im CERN vorkonfiguriert auf OS-9-Prozessormodulen angeboten [29].

FATIMA ist ein Auslesesystem für CAMAC-FERA-Module. Es wurde in der KFA Jülich / IKP entwickelt und wird auch dort (noch) verwendet ([30]). Hard- und Software (QBUS, VMS) von VAX-Rechnern werden intensiv ausgenutzt, so daß eine Portierung problematisch ist.

Kölner FERA-Analysator : Hierbei handelt es sich um ein gemischtes VME/CAMAC-System (auch NIM-ADCs — SILENA 4928 — können eingebunden werden), das speziell zum Auslesen von FERA-Modulen optimiert ist [31]. CAMAC-Zugriffe sind dabei nur zur Initialisierung nötig, das eigentliche Auslesen der digitalisierten Daten findet innerhalb eines VME-Crates statt. Ursprünglich kontrollierte ein OS-9-Rechner das Ge-

[1] Eine genaue Zahl anzugeben, ist problematisch. In der Praxis beschränkt nicht der logische Adreßraum die Anzahl der Zielstationen, sondern die elektrische Übertragung der Signale. Real sind 2–15 Crates; je länger das Kabel, um so weniger.

samtsystem, inzwischen erfolgte die Portierung auf das Echtzeit-UNIX-Derivat LynxOS ([32]).

LabVIEW [33] wird hier nur am Rande erwähnt. Es ist ein kommerzielles System, das von einer komfortablen Oberfläche aus die Kontrolle verschiedener Peripheriegeräte und auch die Erstellung komplexer aus Gerätezugriffen und Kontrollanweisungen bestehender Strukturen ermöglicht. Jedes solche Modul stellt sich für den Benutzer als „virtual Instrument" mit einer graphischen Repräsentation dar. Diese wiederum sind beliebig verschachtelbar, so daß am Ende komplette Bedieneroberflächen aus überschaubaren Modulen entstehen. LabVIEW ist für PCs, Apple Macintosh und SUN-Workstations erhältlich.

Dieses Programmpaket wird, auch in physikalischen Experimenten, für Steuerungsaufgaben (Motoransteuerung, Netzteilüberwachung) verwendet ([34],[35]). Am DESY wird LabVIEW am H1-Experiment im Bereich der „slow control" eingesetzt, auch im CERN wird das Programmpaket inzwischen zentral unterstützt [36].

Die Anpassung an neue Hardware ist allerdings problematisch, da die Treiberschnittstellen unzureichend dokumentiert sind. Zudem ist die erreichbare Reaktionszeit und die Geschwindigkeit für die eigentliche Datenaufnahme völlig unzureichend.

Auch wenn LabVIEW nicht die Grundlage des eigentlichen Datenaufnahmesystems ist, kann es doch in einigen Bereichen in ein modulares System integriert werden. Besonders die Einfachheit, mit der graphische Oberflächen, die auch verwöhnten Ansprüchen genügen, erstellt werden können, macht es für Kontrollaufgaben interessant.

Dazu kommen sicher andere Pakete, die lokal verwendet werden und nicht in der Öffentlichkeit bekannt gemacht wurden.

Einige Fakten fallen auf, wenn man sich die Liste der vorhandenen Datenaufnahmesysteme betrachtet:

- Die meisten Systeme erfordern selbst bei relativ kleinen Änderungen des Experimentaufbaus, die Datenausleseroutinen in 'C', FORTRAN o.ä. neu zu programmieren. Das ist wohl ein Zugeständnis an die Maximierung der Auslesegeschwindigkeit, erschwert aber kurzfristige Umkonfigurierungen und Modultests. Als wünschenswert wurde hier eine Art von Listprozessor erachtet, dessen Ablauf durch mit minimalem Aufwand an die gegebene Konfiguration anpaßbare Definitionsdatensätze gesteuert wird. Dies soll selbstverständlich ohne größere Einbußen im Datendurchsatz geschehen.

- Es sind kaum verteilte Multiprozessorsysteme anzutreffen. Ausnahme ist hier GOOSY. Allerdings werden hier eine Vielzahl von Betriebssystemen (VMS, pSOS und OS-9), relativ unbekannte Programmiersprachen (PL/1) und veraltete Hardware (VAX) verwendet. Es muß bemerkt werden, daß inzwischen (nachdem das hier beschriebene System definiert und implementiert war) einige neuere Entwicklungen bekanntgeworden sind, die mehrere Prozessoren benutzen ([37], [38]).

- Keines der gezeigten Systeme verwirklicht eine strikte Trennung von eigentlichem Datenaufnahmesystem und Benutzerschnittstelle. Ein solcher Aufbau (verteilte Intelligenz) erfordert die Spezifikation eines Protokolls zur Kommunikation zwischen den verschiedenen Bestandteilen. Dieser Aufwand bringt u.a. folgende Nutzeffekte:

  - Das System wird modular, die Komponenten können separat entwickelt und getestet werden. (auch von verschiedenen Arbeitsgruppen)
  - Auch im Betrieb ist es möglich, einzelne Teile auszutauschen, ohne andere zu beeinflussen.
  - Bei der Verwendung neuer Hardware oder Betriebssysteme muß nur das jeweils betroffene Programmodul angepaßt werden.

Allgemein ist bei Software wie dieser (hardwareabhängig, in ständiger Entwicklung begriffen) zu beobachten, daß die Dokumentation für einen Außenstehenden häufig nicht ausreicht, um am System aktiv mitzuentwickeln oder es gar zu portieren. Das bedeutet, daß für diesen Zweck die Hilfe der ursprünglichen Entwickler benötigt wird, was im gegebenen Umfeld nicht immer praktikabel ist.

Es gab also einige Gründe, ein neues Datenaufnahmesystem zu konzipieren und dabei auch von Grund auf neue Wege zu gehen. Die wesentlichen Forderungen an diese Neuentwicklung sind:

- Skalierbarkeit: durch die Möglichkeit, die Rechenleistung einer wählbaren Anzahl von Prozessoren zu nutzen. Das gilt für das Frontend (Anpassung an das Datenaufkommen und die Datentransport- und -verarbeitungsbandbreiten), die Steuerung (separate Benutzeroberflächen für verschiedene Aufgaben) und die Online-Darstellung (Requirierung ausreichender Rechenleistung für aufwendigere Analysen).

- Portabilität: Auf die Verwendung von Funktionen, die nur auf bestimmten Plattformen verfügbar sind, sollte weitgehend verzichtet werden. Wenn unumgänglich, sollte für die Verwendung dieser speziellen Eigenschaften eine allgemeine Schnittstelle definiert werden, deren Funktionalität sich auf den anderen Plattformen emulieren läßt.

- Verwendung von Standards in Hard- und Software.

- Modularität, Verwendung moderner Prinzipien der Softwaretechnologie.

- Konfigurierbarkeit: Möglichkeit der Anpassung an wechselnde Experimentkonfigurationen durch Listensteuerung.

- Effektivität: Es ist ein Kompromiß zwischen Konfigurierbarkeit durch interpretative Abarbeitung und Geschwindigkeit durch Ausführung vorbereiteter Routinen zu finden.

2.1.2 Bestandteile, Befehls- und Datenfluß

Für die Digitalisierung und Steuerung in physikalischen Experimenten werden meist Module im CAMAC, FASTBUS- oder VME-Standard verwendet. Diese werden von intelligenten oder transparenten (ferngesteuerten) Controllern kontrolliert und ausgelesen. Um komplexere Systeme zu bilden, können die Überrahmen wiederum mit Bussen oder Netzwerken verbunden werden. Als universelle Lösung für solche *vertikalen* Verbindungen wurde der standardisierte VIC-Bus ([41], [42]) gewählt. Dies ist ein paralleler Kabelbus, der sich nicht nur mit den verbreiteten Frontend-Controllern, sondern auch mit verschiedenen zur Steuerung verwendeten Rechnern, wie PCs, Apple Macintosh, SUN- und DEC-Workstations, verbinden läßt.

Eine andere durch die Bedingungen des Experiments vorgegebene Entscheidung ist die Wahl des Speichermediums. Die 8mm-Schrägspurlaufwerke der Firma EXABYTE erreichen mit 500 kByte/s die in dieser Preisklasse höchste Datenrate. Zudem sind hier die Kosten für den laufenden Betrieb (Bänder) niedrig.

Die verwendete Hardware bestimmt den erreichbaren Datendurchsatz. An dieser Stelle ist ein Kompromiß zwischen Kosten und Nutzen vonnöten. Für physikalische Experimente in der gegebenen Größenordnung ist die Verwendung von Standardkomponenten, wie geschehen, wohl vernünftig.

Aus der Sicht des Datenflusses stellt sich eine solche Kombination von verschiedenen Bussen in Form eines Baumes dar: Die Daten werden von den Enden einer verzweigten Struktur zu deren Wurzel, dem Speichergerät, transportiert. An den Knotenpunkten fließen die Ströme ineinander, bis zum letzten Knoten, an dem alle abzuspeichernden Daten zusammenlaufen. Dieser Knoten heißt *Eventbuilder*. Zu diesem Zweck wird ein VME-Prozessormodul mit ausreichender Rechenleistung und einer Schnittstelle zum Bandgerät (SCSI) verwendet. Auch die untergeordneten Knoten, die *Subeventreader*, sind VME-Module, da VME den direkten Zugriff der Prozessormodule aufeinander erlaubt und so eine geeignete Basis für Mehrprozessorsysteme ist. Bild 2.2 zeigt an einem einfachen Beispiel den Aufbau einer solchen verzweigten Datenaufnahmeanordnung.

Das gesamte System soll von einer Workstation aus gesteuert werden können. Dazu müssen die Kontrollanweisungen von der Workstation zu den einzelnen Prozessormodulen im Frontend transportiert werden. Es ist nicht nur im Hinblick auf Modularität sinnvoll, den Kontrollfluß völlig vom Datenfluß zu trennen: Für den Transport der Daten sollten die Möglichkeiten der Hardware wie Blocktransfers und DMA möglichst effektiv genutzt werden, um den Durchsatz zu maximieren. Dagegen liegt beim Befehlstransport der Schwerpunkt mehr auf Sicherheit und Einfachheit, hier werden Dienste auf höherem Niveau (Betriebssystemservices und Netzwerke) bevorzugt.

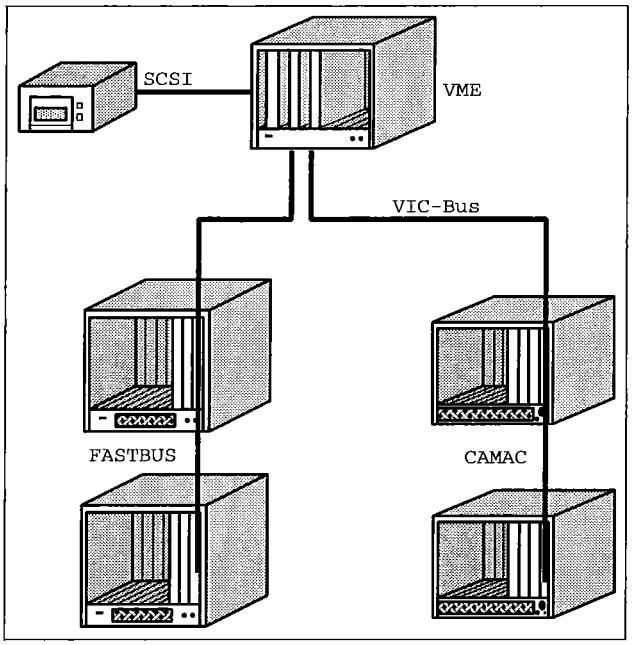


Abbildung 2.2: Verzweigte Struktur einer Datenaufnahmeanordnung.

2.2 Objektorientierung

2.2.1 Objektorientierter Entwurf

Zu den Standardverfahren in der Informatik gehören objektorientierte Techniken beim Entwurf von Systemen und bei der Programmierung von Anwendungen. Da der Begriff „objektorientiert" häufig verwendet und gern als Schlagwort eingesetzt wird, soll zunächst eine gewisse begriffliche Klarheit über dessen Bedeutung geschaffen werden.

Die Informatik im allgemeinen und die Lehre von objektorientierten Techniken im besonderen sind recht junge Fachgebiete. Es haben sich oft noch nicht allgemein anerkannte Definitionen für allgemein benutzte Verfahren herausgebildet. Dazu kommt, daß die gleichen technischen Begriffe auch, ohne ihren ursprünglichen Zusammenhang, für betriebswirtschaftliche Entscheidungen oder für Werbezwecke eingesetzt werden. Daraus resultiert eine gewisse Verwirrung, allein mit dem Wort „objektorientiert" ist eigentlich noch nichts erklärt.

Unsere Betrachtungen beziehen sich auf Programme, oder auf Systeme von Programmen. Ihr Zweck ist es, zur Behandlung eines Problems, das in der umgebenden Welt angesiedelt ist, beizutragen. Daß heißt, gewisse Aspekte der umgebenden Welt müssen im Programm durch wie auch immer geartete Strukturen abgebildet werden.

Objektorientiertheit kann dabei in verschiedenen Blickweisen eine Rolle spielen:

- Im Entwurf des Gesamtsystems: Einzelne Programme oder Routinen ent-

sprechen Teilaspekten des Problems.

- In der Repräsentation von Daten: In der umgebenden Welt zusammengehörige Daten werden auch im Programm als Einheit betrachtet.
- In der Implementierung der Routinen: Programmteile, die für ähnliche Teilprobleme zuständig sind, sind als Einheit organisiert.

Die Fachliteratur liefert für den Begriff der Objekte verschiedene Definitionen, die verschieden viele dieser Blickweisen widerspiegeln. Das Standardwerk „Object-Oriented Design" von G. BOOCH [43] gibt z.B. eine sehr allgemeine Sicht wieder:

> Ein Objekt repräsentiert eine individuelle, identifizierbare Einheit oder ein Gebilde, entweder real oder abstrakt, mit einer definierten Rolle bezüglich des Problems. [2]

RUMBAUGH [44] liefert eine ähnliche Definition, hier wird die klare Abgrenzung zu anderen Objekten in den Vordergrund gestellt („a thing with crisp boundaries").

Etwas weniger generell ist SHLAER [45]:

> Ein Objekt ist eine Abstraktion einer Menge von real existierenden Dingen, für die gilt:
>
> - Alle Elemente der Menge — die Instanzen — haben die gleichen Merkmale.
> - Alle Instanzen sind Gegenstand der und entsprechen den gleichen Regeln.
>
> „Dinge" fallen in eine der fünf folgenden Kategorien: berührbare Dinge, Rollen, Ereignisse, Wechselwirkungen und Spezifikationen.

Zuweilen reduziert sich die Objektdefinition offensichtlich auf Implementierungen von Programmiersprachen, hier wird dann zwischen objektorientierten und objektbasierten Systemen oder Programmiersprachen unterschieden. Im Sinne von [46] sind notwendige Kriterien für das Prädikat „objektorientiert":

- Es werden Objekte unterstützt. Objekte sind Datenabstraktionen mit einem verborgenen inneren Status. Sie stellen sich nach außen durch benannte Operationen dar.
- Objekte gehören einem Typ an.
- Typen können Eigenschaften anderer Typen erben.

Die Definitionen erwecken oft den Anschein recht freier Interpretationen eines allgemein gebräuchlichen Begriffs. Ursache dafür sind wohl die offensichtlichen Parallelen zur „normalen" Erfahrungswelt, zum alltäglichen Sprachgebrauch. Dazu kommt, daß ein zugkräftiges Schlagwort wie „objektorientiert"

[2] „An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain."

von vielen Seiten gern für die eigenen Schöpfungen vereinnahmt wird. In der Praxis sind die verwendeten Begriffe glücklicherweise weniger wichtig als eine dem gegebenen Problem angemessene, möglichst effektive Lösung — und alle genannten Ansätze bieten Vorteile in gewissen Bereichen.

Für die Anwendung objektorientierter Techniken sprechen z.B. folgende Argumente:

- Es ist eine natürlichere und dem wirklichen Leben eher angemessene Methode, zuerst die Funktion der einzelnen Programmbestandteile festzulegen, und deren Darstellung nach außen, d.h., die Interfaces.

- Diese Trennung von Interface und interner Wirkungsweise ermöglicht schrittweise Verbesserungen einzelner Module, ohne andere zu beeinflussen.

- Die Fehlersuche gestaltet sich einfacher, wenn die Funktionen der Programmteile klar voneinander abgegrenzt sind.

- Module mit klar definierter Funktion können unter Umständen in anderen Umgebungen wiederverwendet werden.

- Die Programme werden übersichtlicher, sie gewinnen eine gewisse „innere Logik".

Für die Implementierung objektorientierter Entwürfe ist es naheliegend, objektorientierte Sprachen zu verwenden, die die Konzepte der Abstraktionen und der Kapselung unterstützen. Andererseits ist dies nicht zwingend; es ist ebenso möglich, mit konventionellen Sprachen zu arbeiten. Es ist dann dem Entwickler überlassen, die gewünschte objektorientierte Struktur zu verwirklichen. Im Fall unserer Datenaufnahme bewirkten einige Schwachpunkte der derzeit erhältlichen objektorientierten Sprachen die Entscheidung für eine konventionelle Implementierung:

- Laufzeitaspekte: Mit „C" und Assembler erzeugter Code ist kompakter als mit objektorientierten Sprachen erzeugter, er läuft schneller.

- Berechenbarkeit: Bei in „C"geschriebenen Programmen ist gut abzuschätzen, welchen Maschinencode der Compiler für die erstellten Quellroutinen erzeugt. Die Optimierung der Programme durch Tabellensteuerung u.ä. ist einfach zu verwirklichen. „C++" und andere objektorientierte Sprachen erzeugen selbständig Initialisierungsroutinen und Speicherallozierungen. Argumentübergaben und Symbolnamen werden auf für den Benutzer nicht einsichtige Weise verändert. Das erschwert die Verwendung in hardwarenahen Bereichen.

- Verfügbarkeit: Die Programmiersprache „C" ist praktisch auf allen Plattformen verfügbar, insbesondere auch unter Echtzeitsystemen. „C++" ist zwar inzwischen auch recht verbreitet, aber ... (siehe die nächsten Punkte).

- Mangelnde Standardisierung: Bisher gibt es für „C++“ lediglich de-facto-Standards. Die Implementierungen unterscheiden sich, insbesondere in neueren Sprachelementen. Damit hängt zusammen:

- Schlechte Portierbarkeit: Es sind zum Teil erhebliche Änderungen nötig, um ein C++-Programm mit einen anderen Compiler übersetzen zu können. Das betrifft zun einen die eben erwähnter Unterschiede in den Implementierungen der Sprache, zum anderen sind auch Unterschiede in den Definitionen der Bibliotheksfunktionen festzustellen.

Alle diese Schwachpunkte sind kein prinzipielles Hindernis für die Verwendung von „C++“ oder anderen objektorientierten Sprachen. Schnellere Prozessoren relativieren die ersten beiden Einwände, und mit den kommenden Standards werden die restlichen Kritikpunkte allmählich gegenstandslos werden.

2.2.2 Anwendung in der Datenaufnahme

Welche Vorteile bringen nun Objekte für die Datenaufnahme mit verteilten Systemen?

Die Hardware, die zur Digitalisierung und zum Auslesen der Eventdaten verwendet wird, ist sehr vielfältig. Jedes Modul erfordert eine spezielle Behandlung, für die eine genaue Kenntnis der Wirkungsweise notwendig ist. Eine Hauptaufgabe bei der Vorbereitung eines Experimentes ist es, die Routinen für die Initialisierung und das Auslesen der Digitalisierungsmodule zu erstellen. Ebenfalls unterschiedlich sind die Transportwege für die Eventdaten vom auslesenden Controller bis zum Speichermedium oder zur Online-Analyse.

Für eine objektorientierte Modellierung gilt es, das Gesamtproblem in logisch eigenständige Teile zu zerlegen, die die Gemeinsamkeiten verschiedener Auslesesysteme repräsentieren. Diesen Teilen werden die Objekte zugeordnet, Objekte mit gleichen Eigenschaften werden in Objekttypen (Klassen) zusammengefaßt. Für jeden Typ werden Schnittstellen festgelegt, über die alle Wechselwirkungen mit den Objekten erfolgen, die diesem Typ angehören. Diese Schnittstellen heißen Services; man sagt, „das Objekt bietet Services an“.

Auf diesem Weg erhält man eine Möglichkeit des einheitlichen Zugriffs auf die Geräte, Ressourcen und Daten im Frontend. Von den spezifischen Eigenschaften wird abstrahiert, dem Benutzer bietet sich das Bild eines virtuellen Gerätes mit einer genau definierten Schnittstelle.

2.2.3 Das Vorbild: MMS

Wie schon in Abschnitt 2.1.2 ausgeführt wurde, lassen sich in einem Datenaufnahmesystem die Funktionselemente *Datentransport* und *Steuerung* logisch gut trennen: Mit Hilfe der Steuerung wird u.a. der Datentransport parametriert.

Die Aufgaben des Steuerungsteiles ähneln in vielerlei Hinsicht der Kontrolle automatisierter Fertigungseinrichtungen in der Industrie, es müssen komplexe Anordnungen mit verteilter Intelligenz mit Steuerungsdaten versorgt und überwacht werden.

Für Anwendungen in der industriellen Automatisierung wurde in den letzten 10 Jahren das sogenannte **M**anufacturing **A**utomation **P**rotocol (MAP [47]) spezifiziert. Es stellt einen Standard für die gesamte Kommunikation in einer Fertigungsumgebung von der physikalischen Verbindung der Anlagen bis zu übergeordneten Kontrollaufgaben zur Verfügung.

Der für uns interessanteste Teil von MAP ist die **M**anufacturing **M**essage **S**pecification (MMS — ISO 9506 [48]). Dies ist eine Kommandosprache zur Bedienung intelligenter Fertigungsanlagen. Darin werden Befehle für das Starten und Stoppen von Abläufen innerhalb der Geräte, zum Setzen und Auslesen von Variablen und andere in diesem Umfeld nötige Aufgaben definiert.

Die MMS-Kommandos beziehen sich auf eine Menge von abstrakten Objekten, die die Grundeigenschaften der zu steuernden Anlagen repräsentieren. Die Zuordnung der Objekte zu den realen Funktionseinheiten obliegt dem bedienten Gerät.

Dieses Konzept diente bei der Entwicklung des Datenaufnahmesystems als Vorbild (siehe auch [49]). In Anlehnung daran wurde auch die Bezeichnung **E**xperimental **M**essage **S**pecification (EMS) gewählt.

Es handelt sich bei EMS jedoch keineswegs um eine Implementierung (auch nicht teilweise) von MMS. Diese wäre viel zu komplex und der gegebenen Aufgabe nicht angemessen. Der standardisierten Begriffswelt von MMS wurden, soweit möglich, die Namen und auch die Semantik der Objekte und Services entnommen. Wegen der Ähnlichkeit der zugrundeliegenden Architekturen war dies recht problemlos möglich. Für die Behandlung der Besonderheiten der Datenaufnahme wurden einige neue Objekte und Services eingeführt. Völlig verschieden vom Original ist das für den Transport der Befehle und Daten verwendete Protokoll: Während MMS auf dem (komplexen und kaum verfügbaren) ISO-Stack (siehe auch Abschnitt 2.3.2) basiert, wurde für EMS eine Schnittstelle definiert, die mit verschiedenen Transportmechanismen, so insbesondere auch dem weitverbreiteten TCP/IP, zusammenarbeiten kann.

2.2.4 Objekte der Experimental Message Specification

Die Aufgabe der Objekte bei EMS ist es, wie bei MMS, ein abstraktes Abbild der zu steuernden Anlage zu präsentieren. Das Bild sollte jedoch genau genug sein, um die gezielte Interaktion mit den wesentlichen Komponenten zu ermöglichen.

Soweit sie sich auf analoge Funktionen beziehen, wurden die EMS-Objekte den MMS-Pendants nachgebildet. In der Experimentsteuerung ist es z.B., wie in der Prozeßkontrolle, sinnvoll, den Zustand des Gerätes zu überwachen. Andere Ähnlichkeiten bestehen in der Verwaltung von Variablen oder der Möglichkeit, bestimmte Befehlsfolgen (Prozesse) selbständig ablaufen zu lassen.

Diesen, recht universellen, Aufgaben dienen die folgenden Objekte:

VED: Das zentrale Objekt zur Darstellung eines Frontendsystems ist das VED (virtual experiment device, in Analogie zum virtual manufacturing device in MMS). Es entspricht in unserem Fall einem Frontend-Crate, einem Subeventreader oder Eventbuilder, oder einer ähnlichen abgeschlossenen

Einheit. Im Objekt VED sind die Kommunikation und der Status eines solchen Subsystems zusammengefaßt. Alle folgenden Objekte sind in das VED eingebunden, sie werden vom VED verwaltet.

Variablen: Variablen geben die Möglichkeit, Datenworte und -felder zwischen der Anwendung und dem Frontend auszutauschen. Die verschiedenen Objekte des VED können auf die Werte zugreifen und sie für ihre spezifischen Zwecke verwenden bzw. ihnen Resultate zuweisen.

Domains: Domains dienen zur Ablage verschiedener Informationen, die aus Sicht der Kommunikation transparent sind, d.h., sie werden als unstrukturierte Datenmengen angesehen. Die Interpretation erfolgt durch die Anwendung oder durch ein anderes Objekt. Domains werden für die Übermittlung von Konfigurations- oder Statusdaten verwendet.

Programminvokationen: Eine Programminvokation steht für einen eigenständig ablaufenden Vorgang. Dieser kann nach dem Start, durch Konfigurationsdaten (Domains) gesteuert, bestimmte Teilaufgaben des Gerätes behandeln.

Die Besonderheit der Datenaufnahme besteht darin, daß beim (selbständig ablaufenden, deshalb dem Objekt „Programminvokation" zugeordneten) Auslesen der Digitalisierungsmodule ein Strom von Daten entsteht, der auf ein Ausgabegerät geschrieben wird. Eine weitere Eigenheit ist, daß innerhalb eines Gerätes (d.h., eines Crates) verschiedene Funktionsgruppen existieren können, die zwar unabhängig voneinander initialisiert werden, beim Auslesen aber wieder synchron behandelt werden müssen.

Um diesen Eigenschaften ein entsprechendes Abbild zu verschaffen, wurden zwei neue Objekttypen eingeführt:

Instrumentierungssysteme: Eine logisch oder physikalisch definierte Menge von Frontendelektronik (Digitalisierungshardware oder externe Systeme) wird durch ein Instrumentierungssystem modelliert. Dieses umfaßt eine bestimmte, durch eine Liste definierte, Gruppe von Modulen oder ähnliche einzeln adressierbaren Komponenten. Kommandos zur Beeinflussung der Hardware beziehen sich auf eine solche Gruppe.

Beim Readout erzeugt ein Instrumentierungssystem einen eigenen Datenblock (Subevent), der mit einer benutzerdefinierten Kennung (Instrumentierungssystem-ID) versehen wird.

Datenausgabegeräte: Das Dataout-Objekt stellt eine Abstraktion der Eigenschaften der Datenausgabe- oder -speichergeräte dar. Aufgrund der Verschiedenheit dieser Geräte ist die Wirkung der Services hardwareabhängig. Manche Operationen sind für alle Gerätetypen sinnvoll (z.B. Statusabfrage), andere ergeben nur für bestimmte Geräte einen Sinn (Positionierung bei Bandlaufwerken).

Die Abbildung der realen Hardware auf abstrakte Objekte wird in Bild 2.3 gezeigt. Hier sind auch die Beziehungen der Objekte untereinander und zur Umgebung verdeutlicht. Diese werden in Abschnitt 2.5 genauer erläutert.

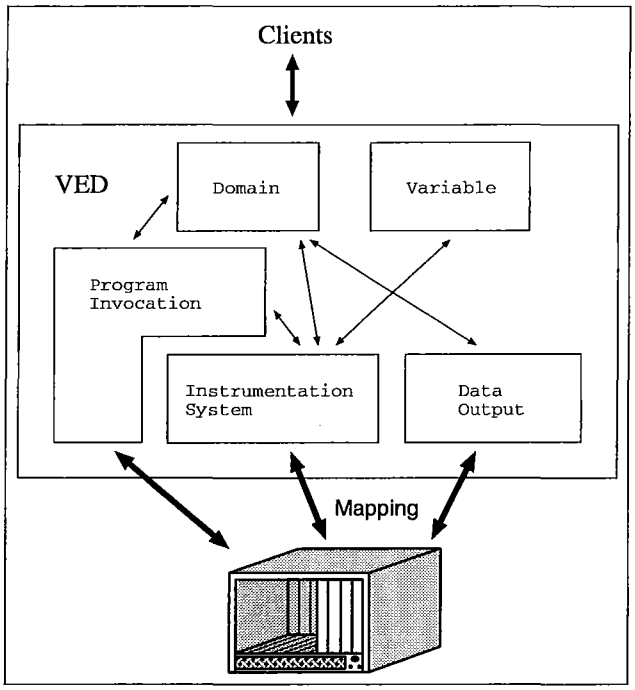


Abbildung 2.3: Abbildung der Hardware auf EMS-Objekte.

2.3 Kommunikation

2.3.1 Einführung — Clients und Server

Verteilte Systeme sind der Versuch, komplexere Datenverarbeitungsoperationen so auf verschiedene Rechner zu verteilen, daß jeder seinen Möglichkeiten entsprechend genutzt wird. Ihre Einführung und Verbreitung korrespondiert mit dem Trend zur Dezentralisierung der Rechentechnik. Dieser wiederum konnte erst Bedeutung erlangen, nachdem die Netzwerktechnik ausgereift, erschwinglich und damit für die allgemeine Nutzung brauchbar war.

Hinter dem in diesem Zusammenhang vielfach überstrapazierten Begriff „Client-Server-Prinzip“ steht keine allgemeine Theorie. Das, was man heute darunter versteht, ist eher historisch gewachsen und in einer gewissen Weise auch zwingend aus den Erfordernissen der Anwendungen hervorgegangen [50].

Das Grundprinzip ist simpel, dem Leben nachempfunden: Einer der Beteiligten (der Client) erteilt die Aufträge, und der andere (der Server) führt sie aus. Das schafft eine saubere Trennung der Verantwortlichkeiten, eine klare Organisationsstruktur. Diese Methode überzeugt vor allem durch Einfachheit; auf einer solchen Grundlage entworfene Systemarchitekturen sind leicht überschaubar und beherrschbar.

Um die Übermittlung des Auftrags vom Client an den Server und die Lieferung des Resultats zu ermöglichen, müssen beide Partner über eine Möglichkeit zur Kommunikation verfügen. Darüber hinaus muß es eine Übereinkunft über die Art und Weise (die Form, die Auslegung des Inhalts) der Auftragserteilung

geben. Für verschiedene Einsatzbereiche existieren Standards, die mehr oder weniger große Sektoren abdecken (RPC, X11, DCE/CORBA, MMS). Andere Lösungen sind nicht für allgemeine, plattformübergreifende Anwendung vorgesehen, sie sind dafür genau auf ihren — eng begrenzten — Einsatzbereich zugeschnitten (NOVELL-Net, COMPUSERVE-CIS).

Die Grundfragen beim Entwurf einer Client-Server-Anwendung sind:

1. Wie werden die Bestandteile der Anwendung auf die einzelnen Rechner verteilt?

2. Auf welche Weise erfolgt die Kommunikation zwischen den Partnern?

Weit verbreitete Beispiele für die erste Entscheidung sind File-Server und Datenbank-Server, die ihren Clients (hier Workstations) ihre Fähigkeiten zum Dateizugriff bzw. zur Datensuche zur Verfügung stellen. Die zweite Frage ist etwas vielschichtiger, es wird das Gebiet der Netzwerkprotokolle (in den nächsten Kapiteln behandelt) berührt, sowie die Semantik, die die übertragenen Daten repräsentieren.

Die Entscheidung nach der Aufteilung der Anwendung wird für unser Datenaufnahmesystem vor allem von den Eigenheiten des Problems bestimmt: Es werden zum einen Rechner mit der Möglichkeit zum schnellen Zugriff auf Digitalisierungshardware benötigt — das sind die Controller im Frontend. Für die Steuerung des Experiments und die graphische Darstellung von Spektren etc. sind Workstations angemessen, die einen gewissen Benutzungskomfort bieten und über die Voraussetzungen zu aufwendiger Graphik verfügen.

Da Netzwerkprotokolle zur Komplexität tendieren, ist man beim Design der Kommunikation praktisch daran gebunden, eine vorhandene Implementierung eines Standardprotokolls zu verwenden. Das verbreitetste Protokoll ist TCP/IP[3]. Es dient als Basis der Entwicklung.

Für die Erfordernisse der Datenaufnahme wurden die anwendungsspezifischen Teile auf der Grundlage des in Abschnitt 2.2.4 eingeführten Objektmodells und im Hinblick auf Effektivität entworfen.

2.3.2 Abstrakte Modellierung der Kommunikation — OSI-Referenzmodell

Das Problem der Übermittlung von Nachrichten führt in den Bereich der Kommunikationsprotokolle. Darunter versteht man einen Satz von Regeln, denen beide Seiten bei der Codierung der Daten und bei der Abwicklung des Transports folgen.

Das beginnt auf niedrigster Ebene mit einer Übereinkunft über das physikalische Medium und dessen Nutzung (z.B. Pegel, Modulation, Zugriffsverfahren). Weitere Aspekte sind die Formate der Datenpakete, die Fehlererkennung und -behebung, das Routing (Wegwahl in komplexen Netztopologien) und die Flußkontrolle (Anpassung der Sendegeschwindigkeit an die Möglichkeiten des Übertragungsweges und des Empfängers). Schließlich müssen die Daten den zugehörigen Anwendungen zugeteilt und übermittelt werden.

[3]**T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol, siehe auch Abschnitt 2.3.3

Zur logischen Strukturierung der verschiedenen Funktionen bietet sich ein hierarchisches Schichtenmodell an, wie es in Bild 2.4 gezeigt ist.

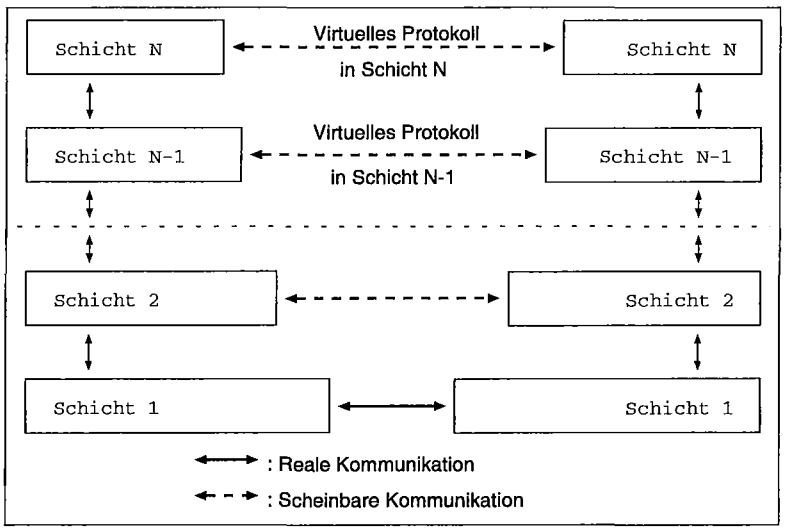


Abbildung 2.4: Hierarchisches Schichtenmodell für die Kommunikation in verteilten Systemen.

In diesem Modell führt jede Schicht bestimmte Operationen mit den ihr übergebenen Daten und Kontrollinformationen aus und reicht die Daten (die sogenannten **P**rotocol **D**ata **U**nits — PDUs) an die nächste Lage weiter. Jede Schicht repräsentiert ein bestimmtes Niveau an Funktionalität, das auf dem darunterliegenden Niveau aufbaut. Die Interaktion zwischen den Schichten erfolgt über *Dienstprimitive* genannte Schnittstellen.

Auf unterster Ebene, der Schicht 1 (physikalische Verbindung) erfolgt der eigentliche Transport. Die darüberliegenden Lagen erzeugen in den kommunizierenden Partnern jeweils ein analoges Bild, indem sie ihre Funktionen auf jeder Seite hinzufügen. Aus Sicht dieser Schicht entsteht so eine sogenannte virtuelle Kommunikation zwischen den entfernten Teilnehmern. Auf höchster Ebene kommunizieren schließlich die Anwendungen auf den verschiedenen Rechnern miteinander.

Im Sinne der Systematisierung der Netzwerkarchitekturen wurde durch die International Standards Organization (ISO) das OSI[4]-Referenzmodell eingeführt [51]. Dieses definiert die in Bild 2.5 gezeigten 7 Schichten, die typischen logisch trennbaren Dienstleistungen der Kommunikationshard- und software entsprechen.

Die Funktionen der einzelnen Schichten seien hier kurz umrissen (aus [52], weiteres in [51]):

1 - physikalische Schicht Diese Schicht sorgt für die eigentliche Übertragung von Bits auf dem Medium. Dazu müssen z.B. Festlegungen zu Signalpegel, Modulationsart und Frequenz getroffen werden.

[4]**O**pen **S**ystems **I**nterconnection

7	Application	Layer	Anwendungs-	Schicht
6	Presentation		Darstellungs-	
5	Session		Sitzungs-	
4	Transport		Transport-	
3	Network		Netzwerk-	
2	Data Link		Verbindungs-	
1	Physical		physikalische	

Abbildung 2.5: Schichten-Einteilung des ISO-Referenzmodells (englische und deutsche Bezeichnungen)

2 - Verbindungsschicht In der Verbindungsschicht werden Datenpakete aus Schicht 3 fehlerfrei auf einer physikalischen Verbindung übertragen. Hier werden Adressierung und Medienzugang realisiert.

3 - Netzwerkschicht Die Aufgabe von Schicht 3 besteht im Routing von Nachrichten durch ein Datenübertragungsnetz.

4 - Transportschicht Die Transportschicht ist die erste Schicht, die einen zuverlässigen Datentransport zwischen den Endpunkten anbietet. Durch sie werden die Eigenheiten des Netzes (Topologie, Fehler) von den höheren, anwendungsbezogenen, Schichten abgeschirmt, nach oben wird eine virtuelle Verbindung bereitgestellt.

5 - Sitzungsschicht Diese Schicht besitzt nur eine niedrige Funktionalität. Sie stellt dem Benutzer die Dienste der Transportschicht in einer komfortablen Weise zur Verfügung. Ihre Dienste umfassen den Sitzungsauf- und -abbau, Authentifizierung und Kontrolle des Sitzungsstatus.

6 - Darstellungsschicht Die Darstellungsschicht befaßt sich mit der Darstellung der Daten. Aspekte dieser sind z.B. Datenformate, Byteordnung, Verschlüsselung und Kompression.

7 - Anwendungsschicht Das ist die Schnittstelle zum Benutzer, hier wird den Daten ihr eigentlicher Sinn gegeben. Beispiele für Anwendungen sind Filetransfer und E-Mail.

Dieses Referenzmodell ist abstrakter Natur, es definiert eine mögliche Einteilung der Dienste eines Kommunikationsnetzes. Weitergehende Standardisierungen betreffen auch die einzelnen Protokolle. Eine Netzwerkimplementierung, die diesen Vorgaben folgte, wäre vergleichsweise komplex, so daß sich die ISO-Protokolle nicht auf breiter Basis durchgesetzt haben. Neuere Entwicklungen basieren teilweise auf den ISO-Normen. Bestehende, historisch gewachsene, Protokolle können mehr oder weniger genau auf dieses Modell abgebildet werden. Ein Nutzen des ISO-Modelles besteht hier in der Vereinheitlichung der Begriffsbildung.

2.3.3 TCP/IP und die zugehörige Terminologie

Zur Vernetzung von Rechnern, besonders in heterogenen Umgebungen, ist TCP/IP ein weithin benutztes Protokoll. Es ist auf praktisch allen bedeutenden Rechnerplattformen verfügbar. Seine Ursprünge liegen in einem Forschungsprojekt des US-Verteidigungsministeriums zur Entwicklung eines Rechnernetzes (ARPANET), das etwa 1969 seine ersten Schritte unternahm. Bis etwa 1981 wurden die Netzwerkprotokolle und die wesentlichen darauf aufbauenden Dienste spezifiziert. 1982 waren 315 Rechner an das entstehende Internet angeschlossen [53]. Heutige Angaben über die Anzahl der angeschlossenen Rechner bewegen sich um 20–30 Millionen.

Zur gleichen Zeit (1969) begann in den Bell Laboratories die Entwicklung des Betriebssystemes UNIX (an der legendären „nutzlos herumstehenden PDP-7"). Im Laufe weiterer Entwicklungen an den Bell Labs und der University of California at Berkeley wurde daraus ein portables, relativ stabiles und ausgereiftes Betriebssystem.

Mit der Version 4.2 (1983) wurden die ARPA-Netzprotokolle in das BSD-UNIX (Berkeley Software Distribution) integriert. Dies war die erste einer größeren Anzahl von Benutzern zugängliche TCP-Implementierung, sie wurde zum Vorbild für fast alle später entstandenen kommerziellen Versionen. Die Netzwerkfunktionen sind, entsprechend der UNIX-Philosophie, im gleichen Stil wie alle anderen Ein- und Ausgabefunktionen eingebunden. Das heißt, eine Netzwerkverbindung stellt sich dem Programmierer nicht anders als ein Speichermedium oder Peripheriegerät dar.

Ein Endpunkt einer Kommunikationsverbindung ist ein *Socket*. Zur Identifikation der Partner wird jedem Socket eine Adresse zugeordnet. Um wohldefinierte Adressen, die von anderen Programmen „gefunden" werden können, zu generieren, kann ein Programm eine bestimmte Adresse an einen Socket *binden*. Auf diese Weise werden *Services* bereitgestellt, die Clients, die darauf zugreifen wollen, verbinden sich mit der ihnen bekannten Adresse.

Das Socket-Konzept ist recht allgemein, es sind verschiedene Kommunikationsprotokolle damit erreichbar. Für jedes Basisprotokoll existiert ein eigener Adreßraum. Im Fall von TCP/IP besteht eine Adresse aus einer Rechneradresse (IP-Adresse) und einer *Portnummer*.

Sockets werden in verschiedene Typen unterteilt, die nach den Eigenschaften der durch sie repräsentierten Verbindung benannt werden. Die TCP/IP-Familie stellt die Typen *Stream* und *Datagramm* zur Verfügung[5]. Der Sockettyp *Datagramm* steht für die Übermittlung einzelner Pakete, es wird weder garantiert, daß alle Pakete beim Empfänger ankommen, noch, daß die Reihenfolge erhalten bleibt. *Streams* garantieren die Ankunft aller gesendeten Daten in der richtigen Reihenfolge, allerdings kann sich dabei die Anzahl und Größe der Teilblöcke ändern. Zusätzlich wird hier die Illusion einer stehenden Verbindung geschaffen, d.h., es werden Informationen über die Sende- bzw. Empfangsbereitschaft des Partners ausgetauscht.

In der ISO-Protokollfamilie gibt es (wie auch in anderen Protokollen, aber

[5]Genaugenommen ist IP ein Protokoll auf Netzwerkebene, auf dem die Untertypen TCP (für *Streams*) und UDP (für *Datagramme*) aufbauen.

nicht in TCP/IP) auch den Sockettyp der *seqentiellen Pakete*, der sowohl die verlustlose Übertragung der Daten als auch die Erhaltung der Paketierung garantiert. Da EMS aber auf Grundlage des weithin verfügbaren TCP/IP arbeiten soll, wird von dieser Option kein Gebrauch gemacht.

Ein Vergleich mit dem im vorigen Abschnitt dargestellten Referenzmodell zeigt, daß TCP der Transportschicht zuzuordnen ist. Im folgenden werden die Netzwerkdienste bis zu diesem Niveau auch „Transportsystem" genannt.

2.3.4 Datendarstellung

Um Daten zwischen verschiedenen Rechnerarchitekturen austauschen zu können, ist eine gemeinsame Art und Weise der Datendarstellung nötig (siehe auch in Kapitel 2.3.2: Schicht 6 im Referenzmodell).

Das augenfälligste Problem besteht darin, daß verschiedene Rechnerarchitekturen Zahlen verschieden repräsentieren: Das höchstwertige Byte einer Zahl wird entweder zuerst (Motorola, Silicon Graphics) oder zuletzt (Intel, DEC) im Speicher abgelegt.

Eine Variante der Byteanordnung, das von Motorola verwendete Big-Endian-Format (höchstwertiges Byte zuerst), ist herausgehoben, da es üblicherweise in IP-basierten Netzwerken verwendet wird (daher auch „network byte ordering" genannt). Glücklicherweise ist dies das Format, das auch von den im Frontend der Datenaufnahme verwendeten Motorola-Prozessoren verwendet wird, so daß hier keine rechenzeitaufwendige Konversion nötig wird.

Für andere Aspekte existieren etablierte Standards, für uns sind von Bedeutung:

- Die IEEE[6]-Formate für Gleitkommazahlen.
- Der ASCII-Zeichensatz[7].

Die Verwendung dieser Formate ist heute so selbstverständlich, daß andersartige Rechnerarchitekturen bzw. Betriebssysteme als exotisch bezeichnet werden können.

Von SUN wurde für den „remote procedure call" in verteilten Umgebungen die Datenkapselung durch XDR (e**X**ternal **D**ata **R**epresentation, [55]) eingeführt. Diese Spezifikation deckt neben den genannten Aspekten auch die Übermittlung komplexer Datentypen (Strukturen, Felder) ab.

Deren wichtigste Merkmale sind:

- Es wird Network-Byte-Ordering verwendet.
- Der Standard-Ganzzahl-Typ ist 32 Bit breit.
- Für Gleitkommazahlen sind die 32- und 64-Bit-IEEE-Typen erlaubt.
- Alle Einträge sind an 32-Bit-Grenzen ausgerichtet.

[6] IEEE-Standard 754 von 1985 (Reaff. 1990, ANSI seit 1991) [54]
[7] ISO-Standard 646 von 1965

- Zeichenketten und uninterpretierten (opaquen) Daten geht die Zahl der zugehörigen Bytes als Ganzzahl voraus. Der Datenblock wird mit Nullen bis zur nächsten 32-Bit-Grenze aufgefüllt.

- Das erste Element überlagerter Typen (unions) ist ein Selektor vom Typ Ganzzahl.

- Feldern geht die Anzahl ihrer Elemente als Ganzzahl voraus. Die Elemente selbst können beliebige andere Typen (auch mit variabler Länge — Zeichenketten oder unions) sein.

Es werden keine Informationen über die Typen der übertragenen Daten mitgeliefert. Sender und Empfänger müssen die Struktur der Daten kennen oder aus dem Kontext ableiten.

XDR ist für unsere Zwecke gut geeignet, da es nur wenig Verwaltungsaufwand bedingt und die Wahl der Datentypen eine effektive Implementierung auf den verwendeten Frontend-Prozessoren (big-endian, 32 Bit) erlaubt.

Da die Implementierung für EMS besonders einfache Zugriffe auf Ganzzahlen und Ganzzahlfelder erlauben sollte, wurde für die Konversion der anwendunsspezifischen Daten in das XDR-Format ein zweistufiges Verfahren entworfen (siehe Bild 2.6). Auf relativ niedriger Ebene wird der gesamte Datenstrom so behandelt, als enthielte er nur 32-Bit-Ganzzahlen (d.h., auf Little-Endian-Maschinen wird die Byteordnung durchgängig geändert, auf Big-Endian-Maschinen geschieht nichts). Hier kann, falls vorhanden, auch Hardware-Unterstützung in Anspruch genommen werden.

Die Behandlung der XDR-Strukturen erfolgt in den anwendungsnahen Routinen. Im Fall von Ganzzahlen ist hier keine Arbeit nötig, die Behandlung von Feldern ist trivial (vorangestellter Zähler). Somit kann auf die am häufigsten verwendeten Datentypen direkt, ohne den (in den verbreiteten UNIX-Implementierungen nötigen) Aufruf einer Konversionsfunktion, und ohne zusätzliches Kopieren, zugegriffen werden. Zeichenketten, wie auch opaque Daten, bedürfen einer speziellen Umwandlung, da sie als Byte-organisierte Daten nicht der Zahlen-Byteordnung unterliegen, aber in der unteren Schicht fälschlicherweise so behandelt worden sind.

Da Zeichenketten relativ selten verwendet werden, ist der Einfluß auf den Gesamtdurchsatz eher klein.

2.3.5 Befehls- und Statustransport durch Messages

Der Informationstransport zwischen Client und Server erfolgt durch *Messages*. Eine solche Message ist in der oben eingeführten Nomenklatur die zwischen den EMS-Anwendungen und dem unterliegenden Protokoll ausgetauschte *Protocol Data Unit* (PDU).

Ein *Request* wird abgesetzt (d.h., ein Service eines Objekts in Anspruch genommen), indem ein Datenpaket mit den nötigen Parametern vom Client zum Server übermittelt wird. Der Server antwortet mit einem Datenpaket, das die Resultate der Aktion enthält. Die Datenpakete heißen Request- bzw. Confirmation-PDU.

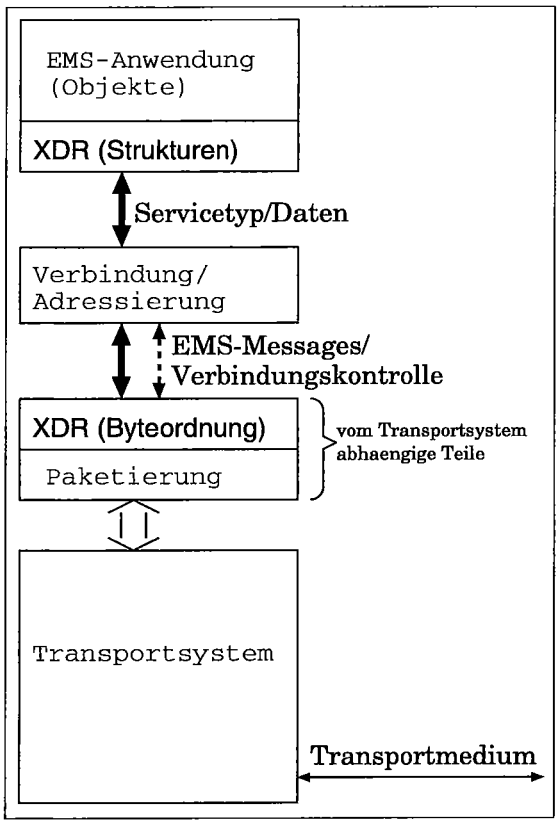


Abbildung 2.6: Protokollschichten von EMS und ausgetauschte Daten (PDUs). Dazu wird gezeigt, an welchen Punkten die zweistufige Konversion in das XDR-Format stattfindet.

Da ein Server verschiedene Aktionen nach der durch Messages gesteuerten Initialisierung selbständig vornimmt, kann auch außerhalb der „synchronen" Befehlsabarbeitung das Absetzen einer Mitteilung an den Client oder die Clients nötig werden. In diesen Fällen wird ein eigener PDU-Typ, eine *unsolicited Message*, abgesetzt.

Die Übermittlung wird initiiert, indem *Dienstprimitive* der unterliegenden Protokollschicht aufgerufen werden. Die dabei verwendete Terminologie ist in Bild 2.7 gezeigt.

Ein Datenpaket enthält in einem festen Kopf Informationen über Quelle und Ziel, Servicetyp und Länge der folgenden Nutzdaten. (Der genaue Aufbau ist im Anhang B.1.1 beschrieben.) Die Nutzdaten sind vom Servicetyp abhängige Parameter (bei Requests) bzw. Resultate (bei Confirmations und unsolicited Messages).

Um auf der Basis verschiedener, auch einfacherer, unterliegender Netzprotokolle arbeiten zu können, werden im EMS-Protokoll sowohl die Längen der zu übermittelnden Blöcke als auch der Zustand einer Verbindung unabhängig

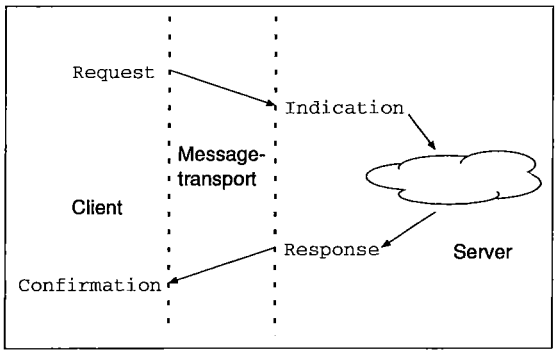


Abbildung 2.7: Bezeichnungen für die Dienstprimitive in der Client-Server-Beziehung.

von den Möglichkeiten des benutzten Protokolls verwaltet. Im Fall der Länge geschieht das auf einfache Weise, indem der feste Kopf der Messages eine Information über die Länge des Datenfeldes enthält. Der Verbindungsauf- und abbau erfolgt durch Austausch eigener Datenblöcke (*connection-Request-PDU*s ...).

Bild 2.6 zeigt die Protokollschichten, die EMS beinhaltet. Eine direkte Zuordnung zu den Schichten des ISO-Modells in Bild 2.5 ist etwas problematisch, da die Funktionalität des unterliegenden Transportsystems unterschiedlich sein kann. Das Transportsystem deckt etwa die Schichten 1–4 ab, wobei verschiedene Funktionalitäten (Paketierung, Verbindungskontrolle) eventuell vom EMS-Teil gedoppelt werden. Die EMS-Teile implementieren die Funktionen der Schichten 5–7. Die dem Bereich Datendarstellung zuzuordnenden XDR-Routinen sind, wie in Abschnitt 2.3.4 beschrieben, in zwei Schritte geteilt.

2.3.6 Adressierung und Namensraum

Unabhängig von den diesbezüglichen Möglichkeiten des Transportsystems beinhaltet EMS eine eigenes System für die Adressierung der Server.

Jeder EMS-Server wird durch einen logischen Namen bezeichnet, die dahinterstehenden Verbindungsparameter bleiben den Anwendungsprogrammen verborgen. Die Übersetzung der logischen Bezeichner in die Adressen des unterliegenden Protokolls erfolgt durch Software auf der Clientseite (Kommunikationsprozeß). Im Fall einer TCP/IP-Verbindung erfolgt eine einfache Zuordnung zwischen den logischen Namen und den IP-Adresse-Portnummer-Paaren. Falls für den Transport der Messages zum Server mehrere unterliegende Verbindungen benötigt werden (durch einen „Gateway“ aneinandergekettet), entspricht dem logischen Namen des Servers eine Folge von Parametersätzen. Die Daten, die die Gateways benötigen, um die folgenden Verbindungen aufzubauen, werden mit den *Connection-Request-PDU*s transportiert.

Auf diese Weise wird ein eigener Namensraum über die eventuell bereits vorhandenen Adressierungsmöglichkeiten gelegt. Das dient der Abstraktion, Änderungen in den Transportwegen haben keinen Einfluß auf die Anwendungsprogramme. Der Umfang dieses Namensraums wird durch den (praktisch im

Kommunikationsprozeß verwirklichten — siehe Abschnitt 3.1) Nameservice festgelegt.

Der Anwender kann das Kommunikationssystem als „Black Box“ betrachten, für ihn erfolgt der Transport der Messages über logische Punkt-zu-Punkt-Verbindungen zwischen einem Client auf einer Workstation und einem Server auf einem Frontendprozessor. Bild 2.8 verdeutlicht das.

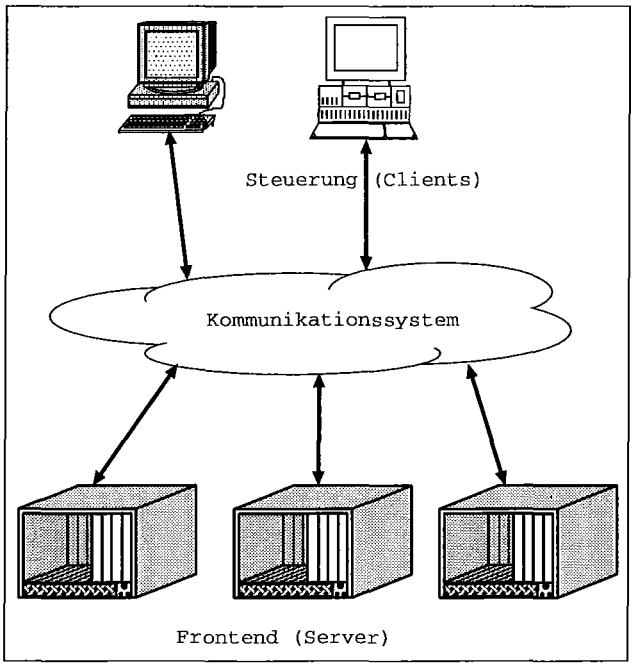


Abbildung 2.8: Flaches Kommunikationsmodell: Der Messagetransport erfolgt über logische Punkt-zu-Punkt-Verbindungen.

2.4 Konfigurierbarkeit durch „lokale Prozeduren“ und Listprozessor

In Abschnitt 2.2.4 wurde eine objektorientierte Modellierung des Frontends der Datenaufnahme entworfen. Damit wurde das Ziel realisiert, eine Abstraktion von den realen Eigenschaften der Geräte vorzunehmen.

Die realen Eigenschaften können aber sehr verschieden sein, und für die Datenaufnahme ist typisch, daß sie sich zudem im Betrieb ändern können (z.B. durch Umkonfiguration der Module). Es wird besonderer Wert darauf gelegt, daß auf solche Änderungen schnell reagiert werden kann.

Deshalb wurden die hardwarespezifischen Funktionen logisch vom Rest des EMS-Servers getrennt. Sie werden als „lokale Prozeduren“ getrennt implementiert und identifizieren sich gegenüber dem restlichen System durch einen eindeutigen Namen.

Wenn der EMS-Server die von ihm kontrollierte Hardware bedienen soll, wird ihm über entsprechende Requests mitgeteilt, welche der lokalen Prozeduren er (mit welchen Parametern) ausführen soll. Das ermöglicht, durch Änderung von Requestparametern das System an eine geänderte Hardwarekonfiguration anzupassen. Welche Prozeduren zur Verfügung stehen, kann über einen entsprechenden Dienst ermittelt werden.

Das eben beschriebene Prinzip wird an zwei Punkten im Frontend der Datenerfassung angewendet:

- Wenn ein Controller selbständig (*Programminvokation*!) Digitalisierungsmodule ausliest, wird dies von einer *Triggerprozedur* synchronisiert, d.h., die Triggerprozedur teilt dem Programminvokations-Objekt mit, ob ein gültiges Ereignis stattfand. Die zu verwendende Prozedur und deren Parameter werden als *Domain* definiert.

- Die *Instrumentierungssysteme* ermöglichen Operationen auf Gruppen von Modulen. Hier wurde vorgesehen, Prozeduraufrufe in *Prozedurlisten* zu gruppieren. Ein in den Server integrierter Listprozessor wird zur Interpretation dieser Listen aufgerufen.

2.5 Die EMS-Services und deren Bedeutung

Aufbauend auf der kurzen Übersicht in Abschnitt 2.2.4, folgt hier eine etwas ausführlichere Darstellung der Services, die in den VEDs (durch Absenden von Requests) aktiviert werden können. Dazu wird beschrieben, wie sich solche abstrakten Operationen auf konkrete Frondend-Geräte auswirken (siehe auch [56]).

Manche Services sind hier nur am Rande erwähnt, eine vollständige Liste der Servicenamen befindet sich im Anhang B.3.

VED : Ein VED (**V**irtual **E**xperiment **D**evice) bildet die Verwaltungsstruktur um alle anderen Objekte, und es stellt logisch den Partner (Server) der Kommunikation mit den Clients dar. Das heißt, eine Adresse im Kommunikationssystem entspricht einem VED, Requests werden an dieses gerichtet und intern an die angesprochenen Objekte weitergeleitet.

Einige Services werden direkt vom VED erbracht:

- *Initiate* und *Conclude* dienen der Eröffnung bzw. dem Abschluß der Kommunikation. *Initiate* tauscht die Versionsnummern verschiedener zwischen Clients und Server vereinbarter Codierungstabellen aus. *Conclude* ist aus Symmetriegründen vorhanden und hat bisher keine Wirkung.
- Auf den Status des Gesamtsystems beziehen sich *GetVEDStatus* und *ResetVED*. Die Bezeichnungen sollten sich im wesentlichen selbst erklären. Der Status ist allerdings eher unspektakulär, es werden momentan nur die bei der letzten Kommunikationsaufnahme erhaltenen Daten zurückgeliefert.

- Welche Objekte zur Zeit definiert sind, kann durch den Service *GetNameList* erfragt werden.
- *GetCapabilityList* ermittelt, welche Operationen das Gerät auf der zu bedienenden Hardware ausführen kann. Dafür gibt es bisher zwei Kategorien, die durch einen Aufrufparameter ausgewählt werden. Eine enthält die den Instrumentierungssystemen über einen Listprozessor zugänglichen Prozeduren, die andere die wählbaren Triggerprozeduren für das Readout (siehe unten).

Variable : Variablen bieten die Möglichkeit, Daten zwischen Clients und Prozeduren im Server auszutauschen. Sie werden durch einen Service (*CreateVariable*) alloziert und stehen dann zum Schreiben und Lesen durch Clients oder Prozeduren im Server zur Verfügung. Die Adressierung erfolgt durch einen (im VED eindeutigen) Index. Für den Zweck der Datenaufnahme wurde es als ausreichend erachtet, nur einen einzigen Variablentyp zu definieren: alle Variablen werden als 32-Bit-Ganzzahl oder Vektor daraus betrachtet.

Domain : Ein *Domain*-Objekt steht für einen Datensatz, dessen Struktur für die Kommunikationsschicht uninteressant ist, die Interpretation erfolgt auf der Ebene der Anwendung (der Objekte). Solche Datensätze werden für hardwareabhängige Konfigurations- und Statusdaten verwendet. Die zugehörigen Services können Domains zum VED und zurück transportieren sowie löschen. Jede Domain wird durch eine Typkennung und einen Index adressiert. Bisher werden folgende Typen verwendet:

Modulliste : beschreibt, welche Module über das VED bedient werden können. Die Liste besteht aus Paaren von physikalischen Adressen (d.h., Stationsnummern in CAMAC) und global vereinbarten Typbezeichnern. Die Typinformation kann von den lokalen Prozeduren zur automatischen Anpassung an die Hardwarekonfiguration oder zur Konsistenzprüfung verwendet werden.

Trigger : enthält die Definition (lokale Prozedur und deren Parameter) der für das Readout zu verwendenden Triggerprozedur.

LAM : definiert die Aktionen, die beim Auftreten einer Ausnahmebedingung in der Hardware (in der CAMAC-Nomenklatur LAM genannt) auszuführen sind. Es kann eine Prozedurliste abgearbeitet werden, die von dieser erzeugten Daten werden optional als „unsolicited Message“ den Clients übermittelt.

Datain : existiert nur in Subeventreadern oder Eventbuildern. Diese Domains enthalten jeweils eine Spezifikation einer „Datenquelle“. Es wird der Zugriffsweg auf die Daten, die vom (im Sinne des Datenflusses) Vorläufer erzeugt werden, beschrieben. Dazu gehören z.B. Informationen, ob ein Treiber verwendet werden soll, oder Details der Adressierung im verwendeten Bussystem.

Dataout : ist das logische Gegenstück zu „Datain“. Es wird ein „Datenziel“ beschrieben. Das kann ein im Datenfluß folgendes VED sein,

oder ein „Endverbraucher“, wie ein Bandgerät oder eine Netzwerkverbindung (zur Online-Analyse).

Event : Speziell für die Belange der Datenaufnahme wurde dieser Typ eingeführt: Die Event-Domain kann nur gelesen werden, sie enthält die beim letzten Triggerereignis ausgelesenen Daten.

Instrumentierungssystem : Ein *Instrumentierungssystem* steht für einen aus logischen oder physikalischen Gründen zusammengehörigen Teil der durch das VED repräsentierten Hardware. Im Fall von CAMAC und ähnlichen Modulstandards ist das eine Gruppe von Modulen, die gemeinsam initialisiert oder ausgelesen werden. Die zugehörigen Module werden in einer Liste (durch die Services *DownloadISModulList*, *UploadISModulList* und *DeleteISModulList* verwaltet) erfaßt. Diese Liste enthält die Adressen (im Beispiel von CAMAC die Stationsnummern) der Module, die Typinformation kann aus der globalen Modulliste (Domain) gewonnen werden.

Das Objekt „Instrumentierungssystem“ selbst ist ein abstraktes Gebilde. Um mit realer Hardware sinnvolle Aktionen ausführen zu können, greift es auf die *lokalen Prozeduren* zurück. So kann durch den Service *DoCommand* eine Prozedurliste (hier: Kommandoliste) sofort ausgeführt werden. Bild 2.9 zeigt das Zusammenspiel der Objekte mit den lokalen Prozeduren beim Ausführen eines solchen Kommandos. Im Instrumentierungssystem können auch Prozedurlisten (hier: Readoutlisten) für die Verwendung beim Readout (Programminvokation, siehe unten) verwaltet werden. Dafür sind die Services *DownloadReadoutList*, *UploadReadoutList* und *DeleteReadoutList* vorgesehen.

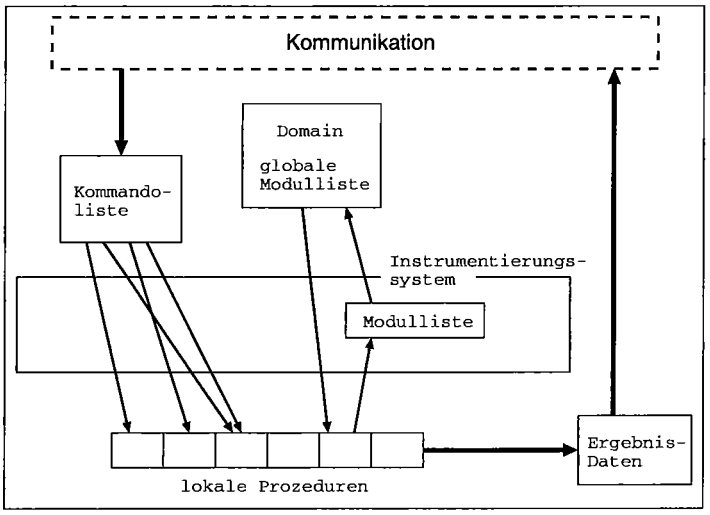


Abbildung 2.9: Ausführung eines Kommandos durch den *DoCommand*-Service des Instrumentierungssystem-Objektes.

Programminvokation : Selbständig ablaufende Vorgänge werden über ein

Programminvokations-Objekt kontrolliert. Die Adressierung erfolgt, ähnlich wie bei Domains, durch einen Typ und einen Index. Folgende Typen werden bisher verwendet:

Readout : Dieser Teil macht die Hauptfunktionalität der Frontend-Controller, das Auslesen von Digitalisierungsmodulen nach Triggerereignissen, aus. Bild 2.10 zeigt das Zusammenwirken von Instrumentierungssystemen, Domains und lokalen Prozeduren mit der Readout-Programminvokation. Die durch die Trigger-Domain beschriebene Prozedur wird fortlaufend aufgerufen (gepollt), sie hat die Aufgabe, durch Interaktion mit der Hardware zu ermitteln, ob eine Triggerbedingung vorliegt. Innerhalb der Instrumentierungssysteme sind Readoutlisten definiert (maximal für jede Triggerbedingung eine Liste). Nach einem Triggerereignis werden alle zur Triggerbedingung (auch Triggermuster genannt) gehörigen Listen abgearbeitet. Die dabei von den lokalen Prozeduren erzeugten Daten werden, zusammen mit einem (bei der Erzeugung des Instrumentierungssystems festgelegten) Identifikator, in den Ausgabedatenstrom geschrieben.

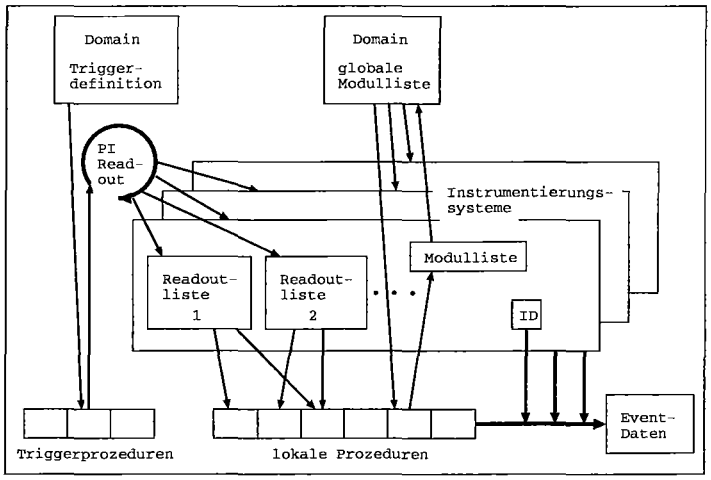


Abbildung 2.10: Ausführung von Readoutlisten nach Triggerereignissen und Erzeugung von Eventdatenblöcken. (PI = Programminvokation)

LAM : Auf Ausnahmebedingungen der Hardware kann durch Ausführung einer Prozedurliste reagiert werden. Die Listen werden als Domains in den EMS-Server geladen, die Ablaufkontrolle erfolgt durch je ein Programminvokations-Objekt. (Der Index selektiert die konkrete Ausnahmebedingung, z.B. die LAM-Nummer in CAMAC.)

Dataout : Dieses Objekt definiert einige Operationen, die direkt ein angeschlossenes Datenausgabegerät beeinflussen. Diese sind:

- Schreiben benutzerdefinierter Datenblöcke in das Ausgabegerät (*WriteDataout*)

- Ermittlung des Status der Geräte (Betriebszustand, aufgetretene Fehler, gegebenenfalls Restkapazität etc. — *GetDataoutStatus*)
- Positionierung (Spulen) des Speichermediums Band (*WindDataout*)
- Trennung von Datensätzen (ebenfalls durch *WindDataout*)
- zeitweiliges Ein- und Ausschalten (*EnableDataout* bzw. *DisableDataout*)

Wenn es, besonders aus Gründen der Effektivität, sinnvoll erscheint, können Objekte im VED auch vordefiniert werden (das VED ist immer vordefiniert). So kann die Art der Datenausgabe für Frontend-Controller festgelegt sein, die Dataout-Domain ist dann weder beschreibbar noch löschbar. Die Readout-Programminvokation ist ebenfalls in ihrer Funktion festgelegt, so daß sich auch hier eine unveränderliche Implementierung anbietet. Es hat sich auch als sinnvoll erwiesen, ein Instrumentierungssystem, das das gesamte Crate enthält, statisch anzulegen, um allgemeine Operationen mit allen Modulen ohne weitere Mühe vornehmen zu können.

2.6 Ein Anwendungsbeispiel

Um die Beziehung des eben eingeführten Objektmodells zur Praxis etwas zu verdeutlichen, soll gezeigt werden, welche EMS-Transaktionen zur Bedienung eines einfachen Datenaufnahmesystems ausgeführt werden müssen.

Das Beispielsystem soll dem in Bild 2.2 gezeigten Aufbau entsprechen. Details, wie z.B. die Synchronisation der Frontend-Controller (über die Triggerprozeduren), Zugriffe auf konkrete Digitalisierungshardware (über lokale Prozeduren) und Busse (hinter Dataout- und -in-Domains verborgen) werden nicht weiter ausgeführt.

Wie am Ende des vorigen Kapitels angedeutet, sollen die Dataout-Domains der Readoutcontroller und die Readout-Programminvokationen aller VEDs vordefiniert sein[8].

Verbindungsaufnahme und allgemeine Initialisierung

Für jeden Frontend-Controller (im Beispiel Eventbuilder, 2 Subeventreader und 4 Readoutcontroller) ist auszuführen:

- physikalische Verbindungsaufnahme (je nach Transportprotokoll, in der Praxis mit Hilfe vorbereiteter Bibliotheken)
- Senden eines *Initiate*-Requests, Vergleich der zurückgelieferten Versionsnummern mit dem eigenen Stand (mit Hilfe vorbereiteter Bibliotheken)
- Senden eines *ResetVED*, um Reste vorhergehender Aktivitäten zu eliminieren, oder Ermittlung des gegenwärtigen Zustandes durch Services wie *GetNameList*, *UploadDomain* oder *GetProgramInvocationAttr*

[8]Das ist genaugenommen ein Vorgriff auf die künftige Implementierung, es erscheint aber sinnvoll, das Anwendungsbeispiel bereits hier einzufügen.

Vorbereitung der Datenaufnahme

Für einen Readoutcontroller typische Aktionen wären:

- Ermittlung der vorhandenen „Triggerprozeduren“ und der „lokalen Prozeduren“ durch *GetCapabilityList*, Vergleich mit den benötigten Funktionen
- Laden der Modulliste durch *DownloadDomain*, Typ *Modulliste*
- für jede Modulgruppe (Funktionsblock) im Crate:
  - Anlegen eines Instrumentierungssystemes mit *CreateIS*
  - Definition der zugehörigen Module durch *DownloadISModulList*
  - Initialisierung der Hardware durch *DoCommand* mit einer entsprechenden Prozedurliste als Argument
  - Laden der Ausleselisten für die verschiedenen Triggerbedingungen durch *DownloadReadoutList*
- Laden der Triggerprozedur mittels *DownloadDomain*, Typ *Trigger*
- Ermittlung der Konfiguration der (vordefinierten) Datenausgabe durch *UploadDomain*, Typ *Dataout* zur späteren Verwendung bei der Initialisierung des zugehörigen Subeventreaders.

Die Subeventreader bedürfen der Definition der Datenein- und -ausgabekanäle:

- Einstellen der Parameter des Datenzugriffs (z.B. Treibername, VIC-Bus-Stationsnummern) für die (im Beispiel jeweils 2) auszulesenden Readoutcontroller durch *DownloadDomain*, Typ *Datain*. Dabei werden die oben gewonnenen Daten benötigt.
- Festlegen der Parameter für die Datenausgabe (im Beispiel ein über VME erreichbarer Pufferbereich) mittels *DownloadDomain*, Typ *Dataout*.

Der Eventbuilder ähnelt den Subeventreadern:

- Einstellen der Parameter des Datenzugriffs (z.B. VME-Adressen der Pufferbereiche) für die (im Beispiel jeweils 2) auszulesenden Subeventreader durch *DownloadDomain*, Typ *Datain*.
- Festlegen der Datenausgabekanäle mittels *DownloadDomain*, Typ *Dataout*. Als primäres Medium kommt ein Bandgerät (beschrieben durch einen systemabhängigen Gerätenamen) zum Einsatz. Zusätzlich können weitere Datenziele, z.B. Netzwerkverbindungen zur Online-Analyse (beschrieben durch IP-Adresse und Portnummer), definiert werden.

Start und Stop des Readouts

Die eigentliche Datenaufnahme wird durch Absetzen von *StartProgramInvocation*-Requests, Typ *Readout* (vordefiniert!) an alle Frontendcontroller gestartet.

Der Verlauf des Auslesens kann durch *GetProgramInvocationAttributes*, auf beliebige VEDs angewendet, überwacht werden. Geräteabhängige Statusinformationen des Bandgerätes werden mit *GetDataoutStatus* ermittelt. Zusätzlich zur Online-Analyse kann durch den Service *UploadDomain*, Typ *Event*, der jeweils letzte ausgelesene Eventdatenblock in jedem Controller begutachtet werden.

Durch Senden von *ResetProgramInvocation*-Requests an alle VEDs wird das Auslesen der Digitalisierungshardware abgeschlossen.

Kapitel 3

Die Implementierung von EMS für das GEM-Datenaufnahmesystem

3.1 Kontroll- und Datenfluß

Die im Frontend der Datenaufnahme verwendeten modularen Einschubsysteme erlauben, völlig verschiedene Funktionseinheiten in einem Überrahmen (Crate) unterzubringen. Diese sollen möglichst durch getrennte Kontrollanwendungen bedient werden können.

Ein Crate wird aber insgesamt durch ein VED repräsentiert, ein VED wird durch nur einen Prozeß im entsprechenden Controller implementiert. Davon soll auch nicht abgegangen werden, da gewisse Vorgänge in einem Crate praktisch nicht oder nur mit großem Aufwand teilbar sind: Beispielsweise existieren zum einen globale Signale (z.B. CAMAC-Inhibit), zum anderen dürfen manche Transaktionen (CAMAC-Blocktransfers) nicht unterbrochen werden.

Mehreren Clients den quasi-gleichzeitigen Zugriff auf einen EMS-Server zu erlauben, kompliziert jedoch die Implementierung. Der Zustand der Kontrollverbindungen muß erfaßt werden, um unsolicited Messages zustellen zu können. Bei manchen Transportprotokollen (z.B. OS/9-Net) ist es nicht möglich, im Fehlerfall das unvorhergesehene Verschwinden eines Clients festzustellen, so daß es schwierig wird, das System robust gegen solche Mängel in Anwenderprogrammen zu machen.

Um diesen Problemen aus dem Wege zu gehen, wurde ein *Kommunikationsprozeß* entwickelt, der diese Verwaltungsaufgaben übernimmt. Im einzelnen hat er folgende Aufgaben:

- Serialisierung der Requests mehrerer Clients für einen Server, so daß dieser nur jeweils einen Request zu einer Zeit zu bearbeiten hat
- evtl. Zwischenspeichern von Requests mit dem gleichen Ziel
- Reaktion auf Fehlerzustände, Debugging, Logging

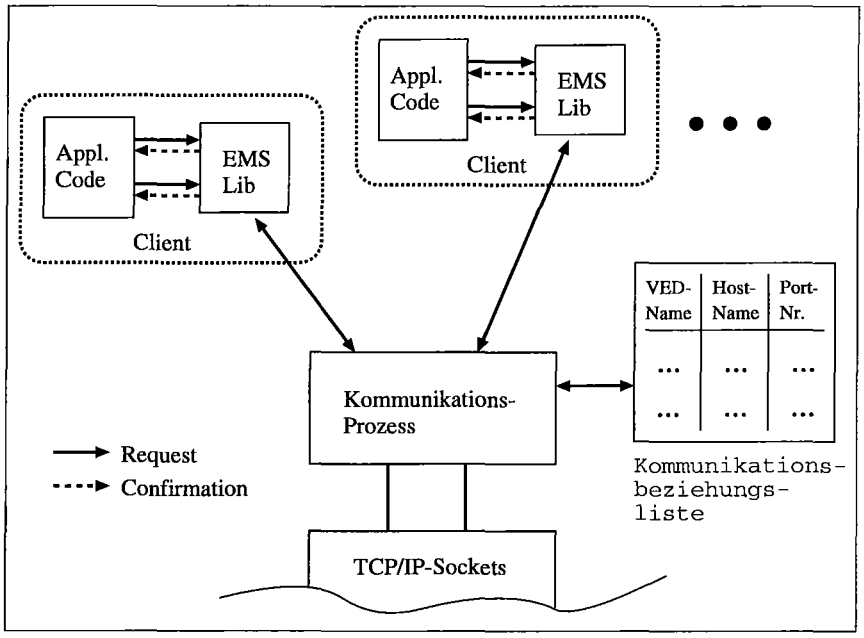


Abbildung 3.1: Implementierung der EMS-Client-Seite — Vermittlung der Kommunikation durch einen Kommunikationsprozeß. (Darstellung nur für TCP/IP-Verbindungen.)

- Zuordnung eines logischen Namens zu einem physikalischen Datenpfad (Nameservice)

Ein solcher Prozeß kann jeweils mehrere Clients und Server verwalten, es darf aber nur ein Kommunikationsprozeß zu einer Zeit mit einem Server verbunden sein.

Der Kommunikationsprozeß läuft auf einer UNIX-Workstation[1]. Die Verbindungen der Clients mit ihm verwenden TCP oder einen UNIX-lokalen Transport. Diese Protokolle können einen Verbindungsabbruch feststellen, so daß bei einem unerwarteten Ende eines Client-Programmes die entsprechende Fehlerbehandlung stattfinden kann.

Für Anwendungsprogramme steht eine Funktionsbibliothek zur Verfügung, die einen bequemen Zugriff auf die EMS-Services über einen Kommunikationsprozeß ermöglicht. Bild 3.1 zeigt den Aufbau der Client-Seite der EMS-Implementierung.

Die Gegenseite im Kontrollfluß, der EMS-Server, ist in Bild 3.2 schematisch dargestellt. (Zur modulare Struktur siehe auch die Bemerkungen im folgenden Abschnitt.) Der Listprozessor und die Datenausgabe, die (neben dem Kommunikationsprotokoll) für das entwickelte Datenaufnahmesystem charakteristischen und für die eigentliche Datenaufnahme wichtigsten Bestandteile, werden im Abschnitt 3.3 genauer erläutert.

[1]Die derzeitige Implementierung (in C++, von P. WÜSTNER, KFA Jülich/ZEL) läuft auf DEC-Systemen.

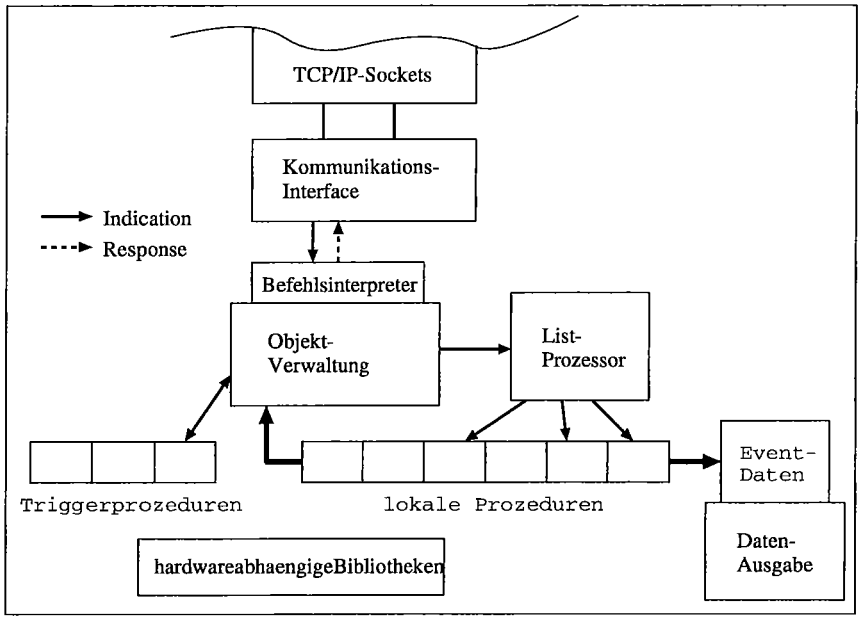


Abbildung 3.2: Implementierung des EMS-Servers (Readout-Controller) (Darstellung nur für TCP/IP-Verbindungen.)

Der Fluß der Eventdaten, die Verkettung der Datenaus- und -eingaben vom Readoutcontroller bis zum Eventbuilder, ist in Abbildung 3.3 genauer gezeigt. Der Aufbau entspricht der schon in Bild 2.2 gezeigten Baumstruktur. Alle Daten werden vom schreibenden Gerät lokal in einen Ringpuffer abgelegt, der lesende Controller übernimmt die Verwaltung des dazwischenliegenden Busses. Dieses Prinzip vereinfacht die Verwaltung der Busse und der Puffer. Der eigentliche Grund liegt allerdings in den Möglichkeiten der vorhandenen Hardware — einige Interfaces sind unidirektional.

Vom Standpunkt des Systemdurchsatzes wäre eine Anordnung mit mehreren Bus-Mastern etwas günstiger: Ein einzelnes Interface kann einen Bus häufig nicht bis zur maximalen Bandbreite auslasten, und Schreibzugriffe sind infolge der Eigenheiten der Busprotokolle meist schneller als Lesezugriffe.

Im Eventbuilder werden die verschiedenen Ausgabegeräte durch spezielle Hilfsprogramme bedient. Diese „Handler“ trennen die Eigenschaften der Geräte vom restlichen Eventbuilder, sie erfüllen auch die Funktionen der *Dataout*-Objekte wie Bandpositionierung und Statusermittlung. Sie werden vom Hauptprogramm gestartet und verwaltet, so daß ihre Existenz für den Anwender nicht direkt von Bedeutung ist. Da sich das das Zeitverhalten der Datenausgabewege sehr unterscheiden kann, werden die Hilfsprogramme intern durch zusätzliche Ringpuffer vom auslesenden Eventbuilderkern entkoppelt.

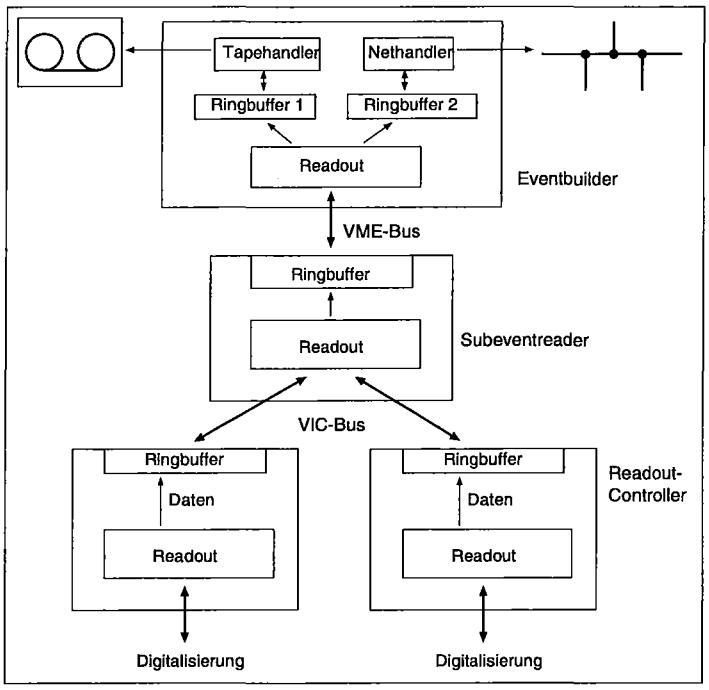


Abbildung 3.3: Datenfluß zwischen den Komponenten des Datenaufnahmesysstems.

3.2 Implementierte Bestandteile, unterstützte Hardware

Das beschriebene Konzept von EMS ist in seinen Grundzügen unabhängig von der unterliegenden Hardware. Auch Details des Betriebssystems werden für die Grundfunktionen nicht vorausgesetzt.

Allerdings wird angenommen, daß das System bestimmte Dienste anbietet oder mindestens ermöglicht:

- Allen Komponenten ist gemeinsam, daß sie mit der Außenwelt kommunizieren müssen. TCP/IP wird hier wegen seiner breiten Verfügbarkeit bevorzugt, es können aber auch andere Wege beschritten werden.

- In den Servern liegt der Schwerpunkt in den Zugriffen auf die zu kontrollierende Hardware. Es muß also die Möglichkeit bestehen, direkt oder mit Treiberunterstützung die gewünschten Aktionen auszuführen.

- Die Clients sollen im allgemeinen die Schnittstelle zum Benutzer bilden. Dazu ist mindestens eine Kommandoeingabe im Textmodus nötig, erstrebenswert ist eine graphische Oberfläche. Weiter ist sinnvoll, (Konfigurations- o.ä.) Daten abspeichern und wiederverwenden zu können, wozu sich eine Datenbank[2] anbietet.

[2]Da Datenvolumen und Transaktionsgeschwindigkeit nicht ausschlaggebend sind, genügt eine Minimalimplementierung.

- Der Kommunikationsprozeß wirkt im Hintergrund. Er sollte in der Lage sein, mehrere Ein- und Ausgabekanäle zu verwalten (zu multiplexen).

Die erstellte Implementierung des Serverprogrammes besteht aus einem Kern und einem Satz von system- oder hardwareabhängigen Modulen. Die Kernimplementierung ist prinzipiell auf allen modernen Rechnern (d.h., mindestens 32 Bit Wortbreite, ausreichend großes Speichermodell) lauffähig. Sie enthält die Verwaltung der einzelnen EMS-Objekte, den Befehlsinterpreter, die grundlegende Ablaufsteuerung und den Listprozessor. Die systemspezifischen Module untergliedern sich in folgender Weise:

- Ein Satz von Routinen für das Senden und Empfangen von Befehlen über einen Kommunikationsweg.

- Bibliotheken für den Zugriff auf die Hardware und andere spezielle Aufgaben. Diese Bibliotheken werden als unabhängige Module angesehen. Jede wird von der zentralen Ablaufsteuerung initialisiert und ggf. zurückgesetzt (und kann damit die zu kontrollierende Hardware initialisieren bzw. zurücksetzen). Diese Module können einen Satz von Funktionen bereitstellen (z.B. CAMAC-ESONE).

- Die vom Listprozessor abzuarbeitenden „lokalen Prozeduren". Diese können wiederum die eben genannten Bibliotheken verwenden. Auf diese Weise ist es möglich, die controller- von den modulspezifischen Teilen zu trennen, also auch hier eine Schichtenarchitektur (in diesem Beispiel: untere Schicht: CAMAC-Protokoll, obere Schicht: Behandlung spezifischer Module) zu verwenden.

- Die zur Gewinnung der Triggerinformation zu verwendenden Triggerprozeduren. Auch diese können auf die hardwarenahen Bibliotheken zugreifen.

- Routinen zur Weiterleitung der beim Readout anfallenden Daten.

Der interne Aufbau der Client- und Serverprogramme und insbesondere das Zusammenspiel der eben genannten Module sind im folgenden Kapitel beschrieben. Hier folgt nun ein Überblick, für welche Betriebssysteme und welche Hardware bisher EMS-Server erstellt und erprobt wurden:

Kommunikation

Für die EMS-Kommunikation wurde die TCP/IP-Schnittstelle implementiert. Sie verwendet das gebräuchliche BSD-Socket-API[3], ist also relativ portabel. Getestet wurde dieses Modul auf den OS/9-Frontend-CPUs sowie unter verschiedenen UNIX-Varianten (ULTRIX, OSF/1, LynxOS). Außerdem wurde (speziell für kleinere OS/9-Systeme) eine Version implementiert, die mit dem ressourcenschonenden OS/9-Net auskommt.

[3] **A**pplication **P**rogramming **I**nterface

Hardwareabhängige Bibliotheken

Bisher werden unter anderem folgende Plattformen unterstützt:

- CAMAC-Controller VCC-2117/2118 (CES, [58])
- FASTBUS-Controller CHI STR-330 (Struck)
- VIC-VSB-Interface (im ZEL der KFA Jülich entwickelt [59]), an einen der VME-Controller FIC 8232, FIC 8234 (CES), E6 oder E7 (ELTEC), oder über das Turbochannel-VSB-Interface (im ZEL der KFA Jülich entwickelt) an eine DEC-Workstation (MIPS- oder Alpha-basiert) angeschlossen
- VME-Zugriffe auf den oben genannten VME-Controllern
- Turbochannel-VIC-Interface TVIC 7214 (CES, [60])
- PROFIBUS am VME-Prozessor VM30 (PEP)
- SMP und IEEE488 über VME-Interfaces

Lokale Prozeduren und Triggerprozeduren

Darauf aufbauend, wurden „lokale Prozeduren" geschaffen, die die von den hardwarenahen Bibliotheken bereitgestellten Funktionen dem EMS-Protokoll zugänglich machen. Für verschiedene Module und Auslesesysteme wurden spezielle Prozeduren erstellt, einige davon sind:

- für CAMAC:
  - FERA-Systeme, auch mit Doppelpufferung (durch das in der KFA entwickelte Controllermodul FSC [61] gesteuert), mit den Digitalisierungsmodulen 4300B (LeCroy), 4418V (SILENA), 812F, 1612F (Gan'Elec) oder dem im ZEL der KFA Jülich entwickelten Patternmodul
  - TDCs 2228 / 2229, 2277, Scaler 2551, 4434 (LeCroy)
  - DRAMS-Readout-System (CHESI et al., CERN [62])
  - CAENNET-Controller C139 (CAEN)
- für FASTBUS
  - ADC 1881, TDCs 187x, CAT[4] 1810 (LeCroy)
  - Multiblock-Readout für die dafür eingerichteten Module (Philips, LeCroy)
- GSI-Triggermodul (siehe [63]) in CAMAC oder FASTBUS
- Fernüberwachung und -steuerung von WIENER-Überrahmen mittels PROFIBUS

[4] Calibration **A**nd **T**rigger Module

- im ZEL der KFA Jülich entwickelte Diskriminatormodule [20] (über VIC-Bus)
- verschiedene VME-, V24-, SMP- und IEC-Geräte, die für das Neutronen-Spinecho-Experiment am Reaktor DIDO (KFA Jülich) benötigt werden ([64])

Datentransport

Als universeller Transportweg für die ausgelesenen Daten werden von beiden Seiten erreichbare Speicherbereiche verwendet. Dieser Speicher wird durch den in Kapitel 3.3.2 beschriebenen Ringpuffermechanismus verwaltet. Für den Datentransport können die DMA-Möglichkeiten der VME-Prozessoren FIC 8232 und FIC 8234 (über Treiber) genutzt werden.

Für den Einsatz in Eventbuildern wurden zusätzlich Hilfsprogramme entwickelt, die die Spezifika verschiedener Ausgabewege behandeln („Tapehandler" und „Nethandler" in Bild 3.3). Die bisher unterstützten Varianten der Datenausgabe sind:

- EXABYTE 8500 und 8505
- TCP-Verbindungen
- lokale Interprozeßkommunikation
- Dateien im Filesystem des Controllers
- „Dummy", Daten werden vernichtet — zum Test

Diese laufen vorläufig nur unter dem Betriebssystem OS-9.

3.3 Der Server im Detail

3.3.1 Der Listprozessor

Der Listprozessor, der die Ausführung der „lokalen Prozeduren" steuert, wurde in Hinblick auf eine hohe Abarbeitungsgeschwindigkeit äußerst einfach ausgeführt. Eine Prozedurliste, Bild 3.4 zeigt ihre Struktur, wird grundsätzlich sequentiell abgearbeitet. Es existieren auf dieser Ebene keine Kontrollstrukturen wie bedingte Ausführung, Sprünge oder Schleifen.

Die Prozeduren werden, wie in Abschnitt 2.4 beschrieben, durch Namen identifiziert. In der vorliegenden Implementierung besteht dieser Name aus einer Zeichenkette und einer Versionsnummer.

Die Umrechnung eines solchen Namens in den in der Prozedurliste verwendeten Index erfolgt im Client. Mit Hilfe des Services *GetCapabilityList* kann er die Liste der im Server implementierten Prozeduren auslesen. Jeder Eintrag besteht aus dem Index und dem Namen, damit kann die Zuordnung der Indices durch Suchen in dieser Tabelle erfolgen.

Vom Listprozessor werden die Argumente der Prozeduren als Array aus 32-Bit-Werten betrachtet. Die Prozeduren bekommen als Übergabeparameter

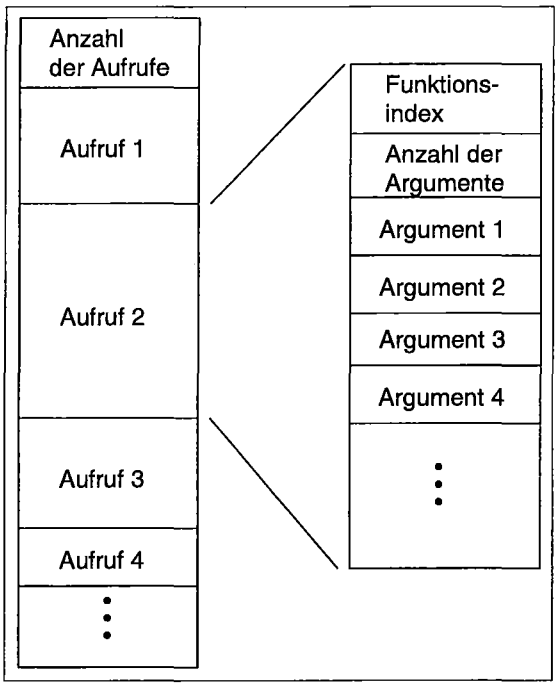


Abbildung 3.4: Struktur einer Prozedurliste

einen Zeiger auf dieses Feld. Sie können von ihnen auch anders interpretiert werden: Auch hier wird die in Abschnitt 2.3.4 beschriebene zweistufige XDR-Implementierung verwendet. Zur Konversion in andere Formate (z.B. Zeichenketten) müssen also spezielle Funktionen verwendet werden.

3.3.2 Die Pufferung von Daten in Ringpuffern

Um Eventdaten vom auslesenden Controller möglichst effektiv zum Subeventreader bzw. zum Eventbuilder zu transportieren, werden gemeinsame Speicherbereiche (dual-Port-Memory) verwendet.

Der Daten liefernde Prozessor legt die Eventblöcke in diesem Speicher ab, und der im Datenfluß folgende liest sie (eventuell zu einem späteren Zeitpunkt) aus. Dabei ist es nötig, die Belegung des verfügbaren Speichers zu verwalten. Das heißt, der lesende Controller muß erfahren, welche Bereiche mit aktuellen Daten gefüllt worden sind, und dem Schreiber müssen die gelesenen und somit wieder verwendbaren Blöcke mitgeteilt werden.

Da die Größe der Eventdatenblöcke häufig nicht im voraus zu bestimmen ist und diese auch stark schwanken kann, wurde ein Pufferverwaltungsalgorithmus verwendet, der mit Speicherbereichen variabler Größe arbeitet. Dadurch wird der vorhandene Pufferbereich besser ausgenutzt, als das mit festen Blockgrößen möglich wäre (wo man als Verwaltungseinheit die maximale Eventgröße ansetzen müßte). Der Preis dafür ist ein etwas höherer Aufwand, um die unterschiedlich langen Speicherfragmente zu verwalten. Die Datenblöcke werden im Speicher einfach hintereinander, ohne Zwischenraum, angeordnet. Wenn

das Ende des Pufferbereiches erreicht ist, wird zum Anfang zurückgesprungen. Da der physikalische Speicher nicht wirklich ringförmig organisiert ist, und Datenblöcke nicht geteilt werden sollen, muß garantiert werden, daß das Ende des Puffers nicht überschritten wird. Zwischen dem Beginn des letzten Events und dem Ende des Speicherbereiches muß deshalb noch ausreichend Raum für einen Block der maximal möglichen Länge bleiben. Die maximale Länge eines Eventdatenblocks ist eine Konstante, die bei der Übersetzung der Programme festgelegt wird. Sie wird im folgenden mit `EVENT_MAX` bezeichnet.

Bild 3.5 zeigt die Wirkungsweise der Pufferung etwas detaillierter.

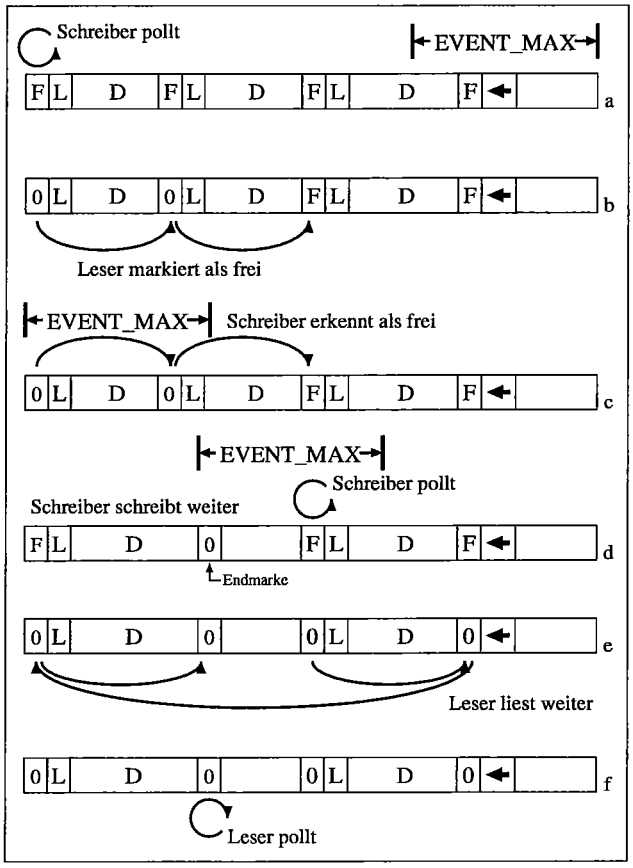


Abbildung 3.5: Verwaltung der Speicherblöcke beim Datenaustausch über Ringpuffer. Bedeutung der Markierungen: F=Es folgen gültige Daten. 0=Der folgende Block ist frei (d.h., er wurde gelesen). L=Länge des folgenden Blocks. ←=Ende des verwendeten Pufferbereichs. D=Daten. Weitere Erklärung im Text.

Jeder der 6 waagerechten Streifen symbolisiert den Zustand des gesamten Pufferbereiches zu einer bestimmten Zeit. Der Zustand wird vom lesenden (oberhalb des Streifens dargestellte Operationen) und vom schreibenden (unterhalb dargestellt) Prozeß beeinflußt. Die 6 im Bild gezeigten Zustände stellen

einen möglichen Verlauf des Datenaustauschs dar, der konkrete Ablauf hängt neben den realen Blocklängen von den Geschwindigkeiten der lesenden bzw. schreibenden Prozesse ab. In den meisten praktischen Fällen werden viel mehr (bis zu mehreren 1000) Eventdatenblöcke in einen Ringpuffer passen.

Jeder Datenblock besteht aus einem Flag (`F` oder `0`), das anzeigt, ob der Block ungelesene Daten enthält, einem Längenfeld (`L`) und den eigentlichen Daten (`D`). Das Längenfeld kennt einen speziellen Zustand, der anzeigt, daß das Ende des Ringpuffers erreicht ist.

Zum Zeitpunkt (a) hat der schreibende Prozeß den Ringpuffer gefüllt, bis der verbleibende Raum nicht mehr ausreichend war, um einen Datenblock der Länge `EVENT_MAX` aufzunehmen. Der nächste Datenblock würde wieder an den Pufferbeginn geschrieben werden. Der lesende Prozeß hat seine Tätigkeit noch nicht begonnen. Der schreibende Prozeß wartet, daß der lesende Prozeß Speicherblöcke wieder freigibt (durch Rücksetzen des Flags).

Zur Zeit (b) hat der lesende Prozeß 2 Datenblöcke gelesen und als frei markiert.

Bild (c) zeigt, daß der schreibende Prozeß zunächst erkannt hat, daß der erste Datenblock wieder frei ist. Da der so gewonnene Raum kleiner als `EVENT_MAX` ist, wird der folgende Block überprüft und als frei erkannt.

Da nun genügend Platz für einen weiteren Eventdatenblock ist, wird mit dem Schreiben fortgefahren, wie in Bild (d) zu sehen ist. Der Schreiber muß sicherstellen, daß hinter dem geschriebenen Block ein Flag mit dem Wert „`0`“ steht (und nicht zufällige Eventdaten), da sonst der lesende Prozeß fälschlicherweise einen Blockanfang erkennen könnte. Der verbleibende Platz ist nun wieder kleiner als `EVENT_MAX`, also muß der schreibende Prozeß auf das Freiwerden weiterer Datenblöcke warten.

Der lesende Prozeß liest in Bild (e) weitere Datenblöcke. Das Ende des Pufferbereichs erkennt er am speziellen Eintrag im Längenfeld.

In Bild (f) hat der lesende Prozeß alle Daten gelesen und wartet, daß der Schreiber ein neues Paket durch Setzen des Flags ankündigt.

Diese Art der Pufferverwaltung ist für den Einsatz in den EMS-basierten Datenerfassungssystemen entworfen und implementiert worden. Ziel war, eine leichte Konfigurierbarkeit des Datentransports über verschiedene Bussysteme zu ermöglichen. In der Tat ist die einzige Information, die der lesende Prozeß braucht, um die Daten lesen zu können, die Anfangsadresse des Pufferbereichs. Außerden war die Effektivität des Verfahrens eine Motivation für diesen Algorithmus. Der zusätzliche Speicherbedarf für die Verwaltungsinformationen ist mit 2 Worten sehr klein, und der Speicher wird fast vollständig ausgenutzt.

Das Konzept bedingt allerdings auch einige Schwächen: Die Datenblöcke sind nicht an bestimmten, z.B. von DMA-Hardware bevorzugten, „runden“ Adressen angeordnet, was unter Umständen Geschwindigkeitseinbußen mit sich bringt. Zum anderen ist der Verwaltungsaufwand recht einseitig verteilt: Während das lesende Programm sehr einfach ist (Flag lesen, Länge lesen, Daten lesen, um Länge weiterspringen), hat der schreibende Prozeß nicht nur die eigene Position beim Schreiben, sondern auch die Position des lesenden Prozesses zu erfassen. Bei etwa konstanter Datenblocklänge ergibt sich folgende Konsequenz für das Zeitverhalten: Beim Füllen des Puffers ist im Mittel ein Puffer als frei zu

erkennen, um einen Datenblock schreiben zu können. Beim Rücksprung an den Pufferstart sind mehrere solche Operationen nötig, da ein Bereich der Länge `EVENT_MAX` frei sein muß, bevor ein (normalerweise kleinerer) Block geschrieben werden kann. Bei im Verhältnis zu `EVENT_MAX` sehr kleinen Blocklängen kann sich das in gelegentlich auftretenden, ungewöhnlich großen Totzeiten (in Tests bis zu 4ms, siehe auch A) niederschlagen.

3.3.3 Das Format der Eventdaten

Das Format der beim Auslesen der Digitalisierungshardware erzeugten Daten ist eine der Schnittstellen des Datenerfassungssystems zur Außenwelt. Eine gewisse vorausschauende Planung ist nötig, damit die von anderen Entwicklern erstellten Auswerteprogramme nicht an jede Änderung der Interna des Frontends angepaßt werden müssen.

Die grundsätzlichen Anforderungen an das Format sind:

- Das Format muß alle zur Identifizierung des Events nötigen Informationen enthalten.
- Die Teilevents müssen den einzelnen Datenquellen zuzuordnen sein.
- Die Erzeugung des Formats sollte möglichst wenig Aufwand vom auslesenden Controller erfordern. Das Zusammenfügen von Datenblöcken im Eventbuilder soll ebenfalls unkompliziert sein.
- Der Datenstrom soll unproblematisch auszuwerten sein, d.h., auf die wichtigsten Informationen (Identifizierung, Länge) soll schnell zugegriffen werden können.
- Eine gewisse Redundanz hilft bei Funktionstests und Konsistenzprüfungen.
- Die den ursprünglich ausgelesenen Daten hinzugefügten Kontrolldaten sollten möglichst klein sein, um die Bandbreite des Datentransportes und das Speichermedium nicht unnötig zu belasten.
- Die Struktur der Daten sollte einen effektiven Transport auf den verwendeten Datenbussen mit den verwendeten Prozessorarchitekturen ermöglichen. (Wortbreite, Alignment)
- Das Format sollte zukünftige Erweiterungen zulassen.

Diese Überlegungen führten zu dem in Bild 3.6 dargestellten Format. Alle Werte werden als 32-Bit-Ganzzahlen betrachtet.

Als Event wird ein Datenblock bezeichnet, der die zu einem physikalischen Event in Verbindung stehenden Daten enthält. Er besteht aus einer Anzahl von Subevents. Ein Subevent wiederum ist der Datenblock, der beim Auslesen eines Instrumentierungssystems im Readoutcontroller entsteht. Ein Event ist die Einheit, die zwischen den Komponenten eines Datenerfassungssystems ausgetauscht wird. Welche Subevents dazu gehören, hängt vom jeweiligen Standpunkt ab: Für einen Cratecontroller sind es die Daten aus seinem Crate, für

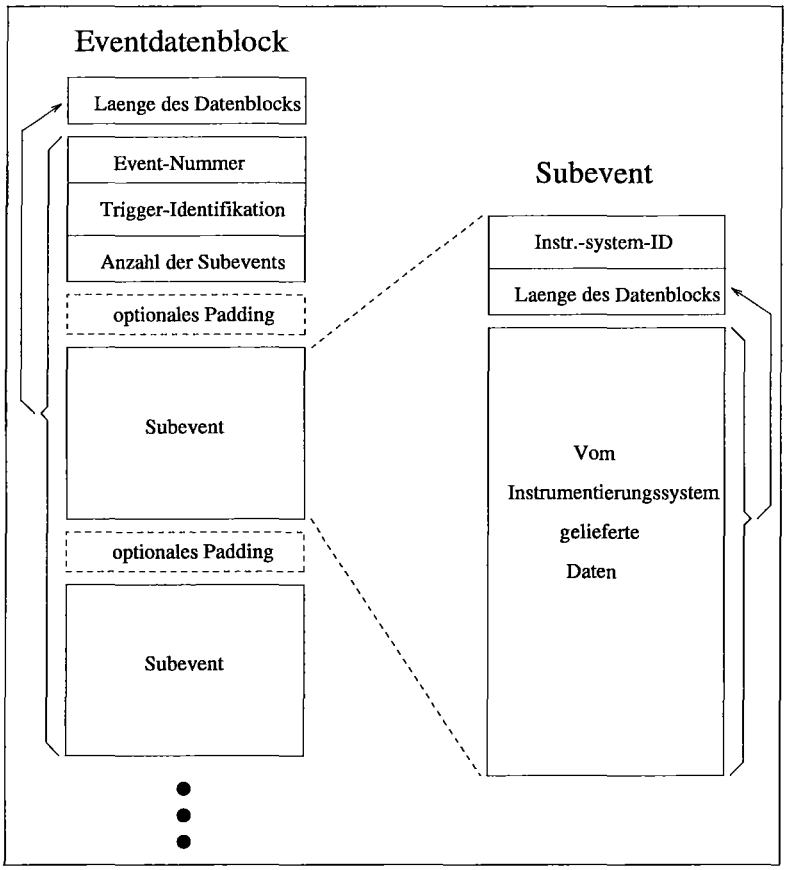


Abbildung 3.6: Aufbau eines Eventdatenblocks aus Eventheader und Subevents, Aufbau eines Subevents aus Subeventheader und Rohdaten.

einen Subeventreader die Daten aus dem von ihm ausgelesenen Zweig. Der Eventbuilder sollte schließlich alle im System aufgenommenen Daten erhalten.

Dieses beschriebene Format wird durchgängig im ganzen Datenaufnahmesystem verwendet, sowohl intern als auch zur Speicherung bzw. zur Onlineanalyse. Es ist erweiterbar, indem neue Subevents (mit neuen Subevent-IDs) hinzugefügt werden. So können auch sekundäre Daten (z.B. bei der Spurrekonstruktion gewonnene Parameter) mit den Originaldaten abgespeichert werden. Die Zuordnung der IDs zu Detektoren, Digitalisierungsmodulen oder Datentypen liegt dabei in der Hand des Benutzers.

Spezielle Eventheader werden verwendet, um nicht zu Events gehörende Informationen (Konfiguration, Parameter von Hilfssystemen, Logging) mit den ausgelesenen Daten abspeichern zu können. In diesen sind die Felder „Eventnummer“ und „Trigger-ID“ auf „0“ gesetzt, das Feld „Anzahl der Subevents“ enthält die Anzahl der im Datenblock enthaltenen Worte. Subeventheader werden nicht benötigt, das Format im Datenblock obliegt wieder dem Benutzer.

3.4 Clientprogramme

Als Entwicklungswerkzeug und zur Fehlersuche wurde in der Anfangsphase von EMS ein einfaches, textbasiertes Steuerprogramm entwickelt ([65]). Es erwies sich allerdings auch für die Steuerung von Experimenten als geeignet und wurde in den GEM-Strahlzeiten dazu verwendet. Die Bedienung erfordert eine gewisse Kenntnis der Semantik der EMS-Requests, auch fehlen Elemente, die z.B. eine automatisierte Überwachung des Frontends oder eine angemessene Reaktion auf Fehlerzustände erleichtern.

Im IKP der KFA Jülich wurde eine graphische Benutzeroberfläche für das EMS-Protokoll entwickelt ([66]). Sie besteht aus einen globalen Steuerprozeß und speziellen Steuerprogrammen für bestimmte, häufig verwendete Instrumentierungssysteme. Die Parameter dieser Instrumentierungssysteme können so leicht auch von Experimentatoren ohne EMS-Kenntnis eingestellt werden. Probleme treten auf, wenn neue, bisher unbekannte, Instrumentierungssysteme oder Module eingefügt werden sollen.

Dem soll die zur Zeit laufende Entwicklung eines objektorientierten, frei konfigurierbaren Steuerprogramms im ZEL der KFA Jülich abhelfen. Dabei sollen moderne Entwicklungswerkzeuge wie die Graphikbibliothek InterViews ([67]) und eine objektorientierte Datenbasis für Konfigurationsdaten zum Einsatz kommen [68].

Es wurde auch eine Anbindung an die Scriptsprache Tcl/Tk ([69], [70]), die auch komplexe Strukturen zuläßt, vorgenommen. Der zweite Bestandteil, Tk, erlaubt die Erstellung graphischer Oberflächen. Somit ist diese Sprache besonders für die rasche Improvisation von Testroutinen oder kleineren Benutzeroberflächen geeignet. Aufgrund ihrer Struktur (schlechte Modularisierung, rein interpretative Abarbeitung) ist sie weniger für komplexe Steuerprogramme prädestiniert.

3.5 Ausblick

3.5.1 Datenkompression im Frontend und deren Nutzen

Unter Umständen ist der Durchsatz der Übertragungsstrecken eine Schwachstelle im Datenaufnahmesystem. Die Messungen im Anhang A zeigen z.B., daß in manchen Konfigurationen der VIC-Bus die Geschwindigkeit des Gesamtsystems begrenzt.

Der Gedanke liegt nahe, die Eventgrößen durch Datenkompression zu verringern und somit die vorhandenen Transportmedien besser auszunutzen. Über die Aussichten dafür soll in den folgenden Abschnitten nachgedacht werden.

Verwendbare Techniken

Datenkompression hat eine große wirtschaftliche Bedeutung. Sie ermöglicht die Speicherung größerer Datenmengen auf einem gegebenen Medium oder den schnelleren Transport auf einer Übertragungsstrecke. Manche Anwendungen

sind ohne die Verwendung solcher Methoden undenkbar, z.B. die Darstellung bewegter Bilder auf Rechnern der PC-Klasse.

Das bringt mit sich, daß effektive Implementierungen von leistungsfähigen Kompressionsalgorithmen Gegenstand eines harten Wettbewerbs und kaum zugänglich sind. Die mathematischen Grundlagen der Kompressionsverfahren sind meist bekannt, die effektive Implementierung entscheidet aber über die Geschwindigkeit und somit über die Brauchbarkeit in zeitkritischen Bereichen. Eine Vielzahl von Patenten deckt einen Großteil der leistungsfähigen Kompressionsalgorithmen ab.

In industriellen Produkten findet die Kompression, wenn die Verarbeitungszeit wichtig ist, in fast allen Fällen in speziell für diesen Zweck entworfenen Schaltkreisen statt. Die einzig bekannte Ausnahme sind die „Festplattenverdopplungsprogramme", die gelegentlich in PC-Betriebssysteme eingebunden werden.

Die hardwarebasierten Lösungen kommen, da zu aufwendig, für die Anwendung in der Datenaufnahme nicht in Betracht. Ebensowenig sind Verfahren, die für die Übertragung von für die (beschränkten) menschlichen Sinne bestimmten Daten (Musik, Bilder) konzipiert sind und Verluste an Information in Kauf nehmen, hier brauchbar.

Bei Kenntnis der Eigenheiten der zu komprimierenden Daten lassen sich Strategien finden, die diese Daten besonders schnell oder wirkungsvoll behandeln. Digitalisierungsmodule in Datenaufnahmesystemen können z.B. Unter- und Überläufe selbständig aussondern, ähnliche Funktionen in Software sind viel schneller auszuführen als generische Kompressionsalgorithmen. Denkbar ist auch, nichtsignifikante Teile von ausgelesenen Datenworten abzuschneiden oder bestimmte Vorverarbeitungen auszuführen. All dies erfordert aber Kenntnis der konkreten Eigenschaften der Daten und ist schon bei der Erstellung der Ausleseroutinen einzuplanen.

Hier soll aber betrachtet werden, inwieweit es sinnvoll ist, allgemeine Kompressionsverfahren bei der Datenaufnahme einzusetzen.

Die wichtigsten bekannten Verfahren der Datenkompression sind:

Huffman-Kompression Die Länge der Daten wird verringert, indem die Codierung der einzelnen Zeichen optimiert wird.

Die mittlere Länge der Zeichen eines Datenblocks ist:

$$l = \sum_i p_i l_i \tag{3.1}$$

Hier sind die l_i die Längen der einzelnen Zeichencodes und die p_i ihre Häufigkeiten. In den unkomprimierten Daten sind alle l_i gleich, z.B. 8 Bit bei byteweise organisierter Datenstruktur. Wenn man die Häufigkeitsverteilung p_i kennt, kann man die Codierung so ändern, daß die mittlere Länge minimal wird. Das heißt, daß häufiger verwendete Zeichen einen kürzeren Code als seltener verwendete erhalten. Von den technischen Erfordernissen abgesehen, ist z.B. die Vergabe von Telefonnummern damit vergleichbar — kurze Nummern für wichtige Ziele. Auch das Morsealphabet berücksichtigt (teilweise) solche

Gesichtspunkte: Häufiger verwendete Buchstaben benötigen weniger Tastendrücke.

Einfache Mengenüberlegungen führen zu einer Randbedingung für die Unterscheidbarkeit der Zeichen:

$$\sum_i \frac{1}{b^{l_i}} \leq 1 \tag{3.2}$$

b ist die Basis des übertragenden Zahlensystems, also 2 für bitweise Organisation und 10 für das Beispiel der Telefonnummern.

Gesucht ist eine ganzzahlige Lösung von (3.1) und (3.2). Diese ist optimal (Gleichheit in (3.2)), wenn die relativen Häufigkeiten der Zeichen Potenzen von 1/2 sind.

Die Leistungsfähigkeit dieser Kompressionsmethode ist begrenzt, in vorhandenen Datenkompressionsprodukten wird sie nur als zusätzliche Option neben anderen, höhere Kompressionsraten erzielenden Verfahren verwendet.

Arithmetische Codierung Der Übergang von ganzen, diskreten Zeichencodelängen zu einer rationalen Zahlendarstellung (die einzelnen Zeichen sind nicht mehr separierbar) bewirkt, daß die Lösung von 3.1 und 3.2 nicht mehr ganzzahlig sein muß. So ist eine bessere Annäherung an die Ideallösung möglich. Eine genauere Beschreibung findet sich in [71].

Die arithmetische Codierung liefert sehr gute Resultate, ist jedoch sehr Rechenzeitaufwendig. IBM verwendet dieses Verfahren, als Schaltkreis implementiert, in Bandlaufwerken. Gleichfalls EXABYTE, lizensiert von IBM.

Beide bisher genannten Verfahren betrachten in ihrer ursprünglichen Form jedes Zeichen einzeln. Weitere Entwicklungen berücksichtigen häufig vorkommende Folgen aufeinanderfolgender Zeichen.

Ziv-Lempel-Algorithmen Das Grundprinzip dieser Verfahren ist, wiederholt vorkommende Zeichenketten nicht neu zu übertragen, sondern kürzere Rückbezüge auf das vorher gesendete Exemplar. Das eigentliche Problem besteht darin, beim Komprimieren möglichst effektiv mehrfach vorkommende Zeichenfolgen zu erkennen.

Fast alle bekannten Kompressionsprogramme verwenden ZIV-LEMPEL-Verfahren. In Form eines Schaltkreises wird diese Methode auch in Modems (V.42bis) und Bandlaufwerken (HP, QIC, DLT) eingesetzt. Vergleiche zwischen arithmetischer Codierung und ZIV-LEMPEL-Algorithmen ehren keinen klaren Sieger, eine jüngere Untersuchung ([72]) favorisiert die ZIV-LEMPEL-Methode für typischerweise auftretende Daten.

Die Verfahren sind näher in [73], wie auch in vielen anderen Werken dargestellt.

Nutzen für die Datenaufnahme

Vergleiche in Hinblick auf die Kompressionsgeschwindigkeit zeigen, daß (bei Realisierung in Software auf normalen Mikroprozessoren) die ZIV-LEMPEL-Algorithmen die höchsten Datendurchsätze ermöglichen. Mit PCs, die in der Verarbeitungsgeschwindigkeit etwa den für die Datenaufnahme verwendeten Controllern entsprechen (80386 / 33 MHz), sind Raten von ca. 100 kByte/s (des unkomprimierten Eingabestroms) erreichbar.

Typische Datendurchsätze im vorhandenen Datenaufnahmesystem liegen bei einigen 100 kByte/s bis zu einigen MByte/s (kurzzeitig), wie Anhang A zu entnehmen ist. Die erreichbaren Kompressionsraten liegen also, verglichen mit der Leistungsfähigkeit der anderen Komponenten, an der Untergrenze. Bei Parallelbetrieb mehrerer Frontendcontroller, und in Erwartung zukünftig wachsender Prozessorgeschwindigkeiten ist die Kompression jedoch nicht von vornherein sinnlos.

Der Nutzen der Kompression wird davon bestimmt, um welchen Teil die Datenmenge verringert wird. Diese Kompressionsrate hängt stark von den Eigenheiten der Daten ab, besonders vom Anteil der gleichen Zeichenketten. Für eine praktische Erprobung wurden die jeweils schnellsten frei verfügbaren Implementierungen einer arithmetischen Codierung und einer ZIV-LEMPEL-Kompression anhand von Daten, die in ihrer Struktur realen Eventdaten nachempfunden sind, getestet.

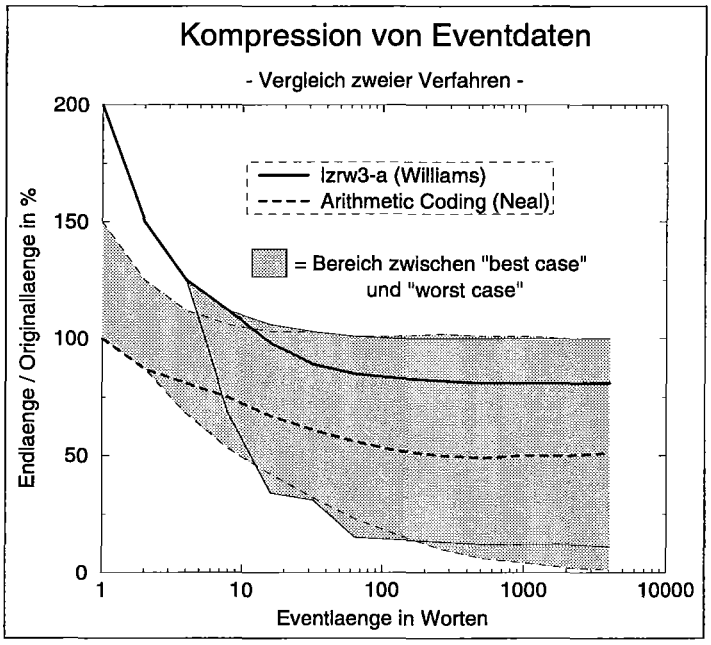


Abbildung 3.7: Kompressionsraten an typischen Eventdaten des Datenerfassungssystems, mit 2 verschiedenen Kompressionsalgorithmen.

Bild 3.7 zeigt, welche Kompressionsraten an den simulierten Daten erreicht wurden, zusammen mit den Resultaten für den günstigsten (konstante Daten)

und den ungünstigsten (Zufallszahlen) Fall. Es wird jeweils ein Event als Block komprimiert, um auch jedes Event einzeln decodieren zu können. Bei kleinen Blockgrößen und bei stochastischen (nicht weiter zu komprimierenden) Daten kann sich die Datenmenge durch die zu übertragenden Verwaltungs- oder Codierungsinformationen auch erhöhen.

Wie zu sehen ist, komprimiert der ZIV-LEMPEL-Algorithmus lzrw3-a die Testdaten lediglich um maximal 20%. Bei einem angenommen Durchsatz von 100 kByte/s werden also nur 20 kByte/s an Bandbreite in den folgenden Teilen des Datenaufnahmesystems eingespart.

Bessere Kompressionsverhältnisse, bis zu ca. 50%, liefert das getestete arithmetische Codierungsverfahren. Seine Geschwindigkeit liegt allerdings bei nur etwa 1/3 im Verhältnis zu lzw3-a. Wenn, wie oben, als Gesamtdurchsatz des Systems trotzdem 100 kByte/s angesetzt werden (unter Einsatz der gleichen Prozessorleistung für die Kompression, die anderen 2/3 der Daten würden unverändert übertragen), ist die gewonnene Bandbreite sogar nur ca. 17%.

Die eben gemachten Abschätzungen erfolgten unter der Randbedingung, daß dem Kompressionsprogramm die gesamte Prozessorleistung des Frontendprozessors zur Verfügung steht. In der Praxis wären die Ergebnisse noch bedeutend ungünstiger. Es folgt also, daß die Verwendung von Kompressionsprogrammen bei der Datenaufnahme momentan, mit den heute zur Verfügung stehenden Prozessoren, keine Vorteile bringt.

In der Zukunft, mit ständig steigenden Verarbeitungsgeschwindigkeiten, könnte es unter Umständen sinnvoll werden, Readout-Daten im Frontend zu komprimieren. Es hat sich gezeigt, daß die Leistungen der Bussysteme nicht im selben Maße wachsen wie die Prozessorgeschwindigkeiten, so daß die Übertragung der Daten immer mehr zum beschränkenden Faktor wird.

3.5.2 Datenintegrität durch CRC

Datenübertragungsmedien sind nie völlig zuverlässig. Es ist, wie gering die Wahrscheinlichkeit auch sein mag, damit zu rechnen, daß Daten beim Transport verfälscht werden. Wenn es wichtig ist, dies festzustellen, werden gewöhnlich Prüfsummen mit den Daten mitgesendet. Das Verfahren ist im Grunde konträr zur oben behandelten Datenkompression: Während die Kompression Redundanzen aus den Daten entfernt, und die Datenstruktur näher an einen ungeordneten Zustand bringt, werden sie hier hinzugefügt, um Konsistenzprüfungen anhand der neuen Ordnungskriterien zu ermöglichen.

Die einfachste Möglichkeit der Datenprüfung (und die, welche am wenigsten Rechenaufwand erfordert) ist eine Quersumme über den Datenblock. Es ist aber offensichtlich, daß sich die Einflüsse mehrerer Fehler auf die Prüfsumme gegenseitig aufheben können. Das ist zwar prinzipiell nie völlig auszuschließen, doch kann die Wahrscheinlichkeit dafür durch Einsatz geschickterer Algorithmen minimiert werden.

Diese Verfahren heißen CRC (**C**yclic **R**edundancy **C**odes). Das Grundziel ist, daß auch kleine Änderungen in den Daten eine beliebig große Änderung in der Prüfsumme bewirken können. In den üblichen Verfahren geschieht das, indem der gesamte Datenblock als eine große Zahl betrachtet wird und diese

durch einen feststehenden Divisor geteilt wird. Als Prüfsumme wird der Rest verwendet [74]. Zur Vereinfachung der Berechnung wird eine sogenannte polynomische Arithmetik verwendet, die im Gegensatz zur üblichen Arithmetik auf Überträge verzichtet [75].

Der eigentliche Schlüssel ist die Wahl des Divisors: Wenn ein Übertragungsfehler, als Zahl ausgedrückt, ein Vielfaches des Divisors ist, hat er keinen Einfluß auf den Rest und bleibt unerkannt. Der Divisor muß also möglichst unähnlich zu den zu erwartenden Fehlern sein. Nach [76] kann gezeigt werden, daß es möglich ist, alle Fehler zu erkennen, die 1, 2 oder jede ungerade Anzahl von Bits verändern, außerdem alle gehäuften Fehler (mehrere aufeinanderfolgende fehlerhafte Bits) bis zu einer gegebenen Länge.

In unserem Datenaufnahmesystem werden die Daten vorrangig über parallele Busse übertragen, die 32 oder auch 64 Bit breit sind. Der Natur solcher Verbindungen entsprechend, ist es wahrscheinlich, daß Fehler wiederholt auf bestimmten Bits auftreten, im Datenstrom ist dann eine Periodizität der Länge 32 bzw. 64 zu erkennen. Eine gute Wahl wären folglich Divisoren mit einer Länge, die größer als diese Periode ist. Leider steht dem der Rechenaufwand entgegen, den die Arithmetik mit Zahlen, die nicht mehr in die Prozessorregister passen, mit sich bringt.

Bei der Inbetriebnahme eines Datenaufnahmesystems ist es sehr wünschenswert, Übertragungsfehler sicher erkennen zu können. Beim späteren Betrieb bleibt die Konfiguration aber relativ konstant, und es treten keine unbilligen mechanischen Belastungen auf. Es ist also recht unwahrscheinlich, daß nach einer gewissen Einlauf- und Testphase noch spontane Übertragungsfehler auftreten.

Deshalb, und in Anbetracht des Aufwands, wurde bei der Datenaufnahme bisher darauf verzichtet, zur Laufzeit CRCs zu übertragen. In der Erprobungsphase ist es möglich, Datenblöcke mit festen Testmustern im Datenstrom zu übertragen, später könnte dies stichprobenartig geschehen.

3.5.3 Zukünftige Entwicklungen in Hard- und Software, Perspektiven von EMS

Die EMS-Komponenten wurden bisher vor allem für die in der Datenaufnahme meist verwendeten Komponenten im CAMAC- FASTBUS- oder VME-Standard entwickelt. Als universeller Datenweg wurde der VIC-Bus eingesetzt.

Die Technologie, sowohl im Bereich der Rechnerhardware als auch der Betriebssysteme, ist jedoch einem laufenden Wandel unterworfen; es ist abzusehen, daß sich neue Entwicklungen am Markt durchsetzen und heutige Standards veralten werden.

Einige Trends werden wohl für die kommenden Entwicklungen der Datenaufnahmesysteme bedeutsam sein:

- Gewöhnliche PCs erreichen eine Leistungsfähigkeit, die der der bisher für die Datenaufnahme verwendeten Plattformen mindestens gleichkommt. Zwar werden neue, schnellere Prozessoren auch z.B. auf VME-Boards eingesetzt, doch ist das Preis-Leistungs-Verhältnis des Massenproduktes

PC praktisch nicht zu schlagen. Einige Argumente, die bisher gegen die Verwendung von PCs sprachen, wie mangelnde Konsistenz in der Konfiguration und unzureichende Eignung für Mehrprozessorsysteme, werden mit dem seit einiger Zeit eingeführten

- PCI-Bus ausgeräumt. PCI entwickelt sich zum plattformübergreifenden Standard als schneller I/O-Bus für den Anschluß der Peripherie. Somit können unter Verwendung allgemein verfügbarer Komponenten modulare Systeme aufgebaut werden. Ein Schwachpunkt von PCI für die Datenaufnahme ist die Begrenzung der elektrischen Last am Bus, die den Anschluß von nur (größenordnungsmäßig) sechs externen Bauteilen an einen Bus erlaubt. PCI wird also keine Alternative zu CAMAC sein, doch kleinere VME-Konfigurationen, wie sie für Eventbuilder verwendet werden, können so preiswert ersetzt werden.

- Das Echtzeitbetriebssystem OS-9, das bisher vorzugsweise im Frontend-Bereich verwendet wird, ist auf Prozessoren der Motorola-68000-Serie beschränkt. Mit neuen Prozessoren entsteht die Notwendigkeit, einen Nachfolger für diese bewährte Lösung zu finden. Aus Gründen der Portierbarkeit wird UNIX (bzw. UNIX-Ableger) bevorzugt. Dies hat den Vorzug, (wie auch OS-9, aber im Gegensatz zu vielen „einbettbaren" Echtzeitkernen) eine eigenständige Entwicklungsumgebung zu haben. Abstriche müssen bei der Modularität und der Eignung für „kleine" Systeme gemacht werden, vorteilhaft ist die Homogenität der gesamten Entwicklung — gleiche Werkzeuge und Schnittstellen im Frontend und auf Workstations.

  Für die absehbare Zukunft wird das Echtzeit-UNIX-Derivat LynxOS favorisiert; es bietet auch die Möglichkeit, plattenlose Konfigurationen zu erzeugen, und sein Ressourcenbedarf ist — für UNIX-Verhältnisse — moderat. LynxOS ist für viele der zur Zeit verbreiteten Prozessoren verfügbar (Motorola 68k, Intel 386, PowerPC, MIPS).

- Die Entwicklung der Netzwerktechnik hat serielle Hochgeschwindigkeitsverbindungen hervorgebracht, die sich im Durchsatz mit den in der Datenaufnahme verwendeten parallelen Bussen (z.B. VIC) messen lassen oder diese übertreffen. Das beginnt mit FDDI und Fast-Ethernet (100 MBit/s), als Standard der Zukunft wird ATM (derzeit typischerweise 155 MBit/s, auch bis zu einigen GBit/s) angesehen. Mit der wachsenden Verbreitung der Datenkommunikation werden schnelle Netzwerke den Massenmarkt erobern, was sich günstig auf das Preis-Leistungs-Verhältnis auswirken sollte. Das macht solche seriellen Verbindungen nicht nur für die Kommunikation im verteilten Rechnersystem, sondern auch für den Transport der Eventblöcke im Datenerfassungsbereich interessant.

Das Konzept von EMS ist unabhängig von Hardware und Betriebssystem angelegt. Es verfügt auch über genügend Abstraktion, um neue Varianten des Zugriffs auf die Eventdaten implementieren zu können. Somit gibt es keine

prinzipiellen Hindernisse, die neuen Techniken in EMS-basierten Datenaufnahmesystemen zu verwenden.

Künftige Entwicklungen der EMS-Serversoftware werden sich zunächst auf die Implementierung abstrakter Programmschnittstellen für gewisse Systemspezifika (Zeitgeber, Interrupts, Treiberzugriffe) beziehen. Das ist teilweise schon geschehen, die Erfahrung zeigt allerdings, daß sich die verschiedenen UNIX-Varianten untereinander zu stark unterscheiden, um hier effektiv durch gleiche Routinen behandelt zu werden.

Ein weiteres Ziel ist es, die Pufferverwaltung dahingehend zu optimieren, daß maximale Transferraten durch Verwendung möglichst großer Datenblöcke erreicht werden können. Eine Durchsatzerhöhung könnte sich auch durch Parallelschaltung mehrerer Bandgeräte realisieren lassen.

Das EMS-Protokoll selbst verfügt bisher nicht über Möglichkeiten, Zugriffe verschiedenen Clients auf den selben Server voreinander zu schützen. Es wäre für Anwendungen, in denen die Einzelprogramme weniger aufeinander abgestimmt sind, wünschenswert, ein Konzept von Objekt-Eigentümern und der Blockierung von Objekten oder Ressourcen zu haben.

Auch die Kontrolle des Systems, die Überwachung der Betriebsparameter wie Datendurchsatz und Pufferausnutzung ist bisher eher spartanisch.

Dies sind Ideen, die neben anderen die zukünftigen Arbeiten am EMS-Frontend beeinflussen werden. Die Entwicklung der Bedieneroberfläche wird durch die Verwendung adäquater objektorientierter Techniken (wie in Abschnitt 3.4 angedeutet) gekennzeichnet sein.

Kapitel 4

Messungen am Experiment

4.1 Durchgeführte Messungen

4.1.1 Die Datenaufnahmeapparatur

Für die Datenaufnahme wurde eine Anordnung gemäß Bild 4.1 verwendet.

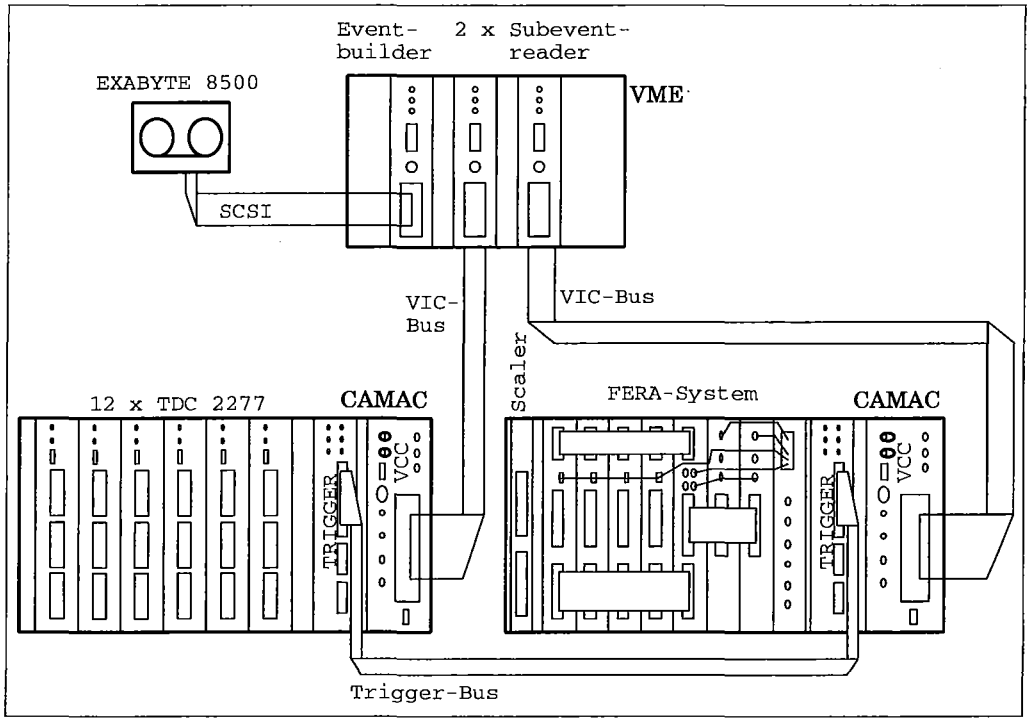


Abbildung 4.1: Konfiguration des Datenaufnahmesystems für die GEM-Experimente.

Die Digitalisierungsmodule (Die Zuordnung zu den Detektoren wurde in Abschnitt 1.6 — Bild 1.13 — erläutert.) sind auf 2 CAMAC-Überrahmen verteilt, die jeweils durch einen Subeventreader über einen VIC-Bus ausgelesen werden. Beide CAMAC-Crates sind mit intelligenten Controllern (CES-VCC2117) aus-

gerüstet und werden parallel ausgelesen. Die Synchronisation der Triggersignale und der Auslesesoftware erfolgt durch GSI-Triggermodule.

Als Subeventreader und Eventbuilder werden Module vom Typ FIC8234 (CES) verwendet. Die Subeventreader kontrollieren lediglich je einen VIC-Bus und stellen Pufferspeicher (bis zur Größe des freien RAMs im System — verwendet wurden je 2MByte) zur Verfügung. Der Nutzen dieses Aufwandes für den Gesamtdurchsatz des Datenaufnahmesystems wird zwar leider durch ein Hardwareproblem in den Businterfaces der Prozessormodule geschmälert (siehe Diskussion im Anhang A.7). Der zusätzliche Pufferspeicher wirkt sich trotzdem positiv auf die Leistung im stoßweisen (Burst-) Betrieb (vor allem bei unterschiedlicher Datenmenge auf den beiden VIC-Bus-Strängen) aus. Die Eventbuilder liest die Daten aus den beiden Subeventreadern ein und schreibt sie auf das Bandgerät. Auch hier wird noch einmal Pufferspeicher (2...8 MByte) zwischengeschaltet, der in erster Linie die Anlaufphasen des Bandgerätes überbrücken soll.

Ein zusätzliches CAMAC-Crate enthält die Logikmodule des Triggersystems sowie Steuereinheiten für die Hochspannungsnetzteile. Dieses wird von einem PC gesteuert, eine Integration in EMS ist für einen späteren Zeitpunkt vorgesehen.

Für die verbleibenden Aufgaben (Signalaufbereitung und -verteilung, Erzeugung der Gates, zusätzliche Logik) werden NIM-Module verwendet.

Alle interessanten Zählraten werden von einem Scalermodul (im FERA-Crate — rechts in Bild 4.1) aufgenommen. Das sind vor allem die Anzahl der Haupttrigger und der aufgezeichneten Ereignisse sowie die Raten der Monitordetektoren. Zusätzlich werden die Raten auf einer kompletten Szintillatorlage in der Fokalebene (P) gezählt.

Das Scalermodul wird in den Pausen zwischen den Extraktionsphasen von COSY ausgelesen. Dazu wurde ein Signal aus der COSY-Steuerung als zweiter Triggerimpuls an das Datenaufnahmesystem angeschlossen. (Die GSI-Triggermodule erlauben, bis zu 15 verschiedene Triggerbedingungen zu unterscheiden.)

Ein zusätzlicher Scalerkanal (im Bild nicht sichtbar) wird für die Gewinnung von Informationen über die Zeitpunkte der Ereignisse verwendet. Er wird mit einer konstanten Frequenz von 1 MHz getaktet und mit dem eben erwähnten COSY-Signal zurückgesetzt. Mit jedem Ereignis wird der Zählerinhalt ausgelesen. So wird die Lage der Events relativ zum COSY-Zyklus gemessen.

4.1.2 Abbildung der Hardware durch EMS-Objekte

Die beiden Readout-Controller, die beider Subeventreader und der Eventbuilder werden durch je ein VED (**V**irtual **E**xperiment **D**evice) repräsentiert. Alle diese VEDs kommunizieren direkt über eine TCP-Verbindung mit dem Kommunikationsprozeß in der Experimentworkstation.

Readout-Controller

Die Digitalisierungshardware in den Frontend-Crates wird durch vier Instrumentierungssysteme abgebildet:

1. Die 12 für das Auslesen der Driftkammern verwendeten Multihit-TDCs (2277).

2. Das Doppelpuffer-FERA-System zur Aufnahme der Energie- und Zeitdaten aus den Szintillatoren.

3. Der einzelne Scalerkanal mit den Zeitmarken.

4. Der 32-Kanal-Scaler für die Monitorinformationen.

Die ersten drei Instrumentierungssysteme werden mit jedem Ereignis ausgelesen, das vierte zu den durch das COSY-Timing gegebenen Zeiten.

Diese beiden Triggerimpulse werden durch die GSI-Triggermodule erfaßt. Die Definition der Trigger aus EMS-Sicht erfolgt durch eine speziell für diese Module erstellte Triggerprozedur (als Domain geladen).

Zusätzlich wurde eine LAM-Programminvokation definiert, die einen Überlauf des 32-Kanal-Scalers erfaßt und behandelt.

Die Ausgabe der Eventdaten (nach außen als Domain — Typ Dataout — dargestellt) ist in den beiden Controllern fest vordefiniert. Sie besteht aus im Dual-Port-RAM der Controller liegenden Ringpuffern.

Subeventreader

Beide Subeventreader sind (von unterschiedlichen VME-Basisadressen abgesehen) identisch konfiguriert.

Eine Domain vom Typ Datain enthält die Parameter für den Zugriff auf den Datenausgabepuffer des zugehörigen CAMAC-Controllers (VIC-Bus-Stationsnummer und -Adresse).

Eine Dataout-Domain definiert einen lokalen (dynamisch allozierten) Speicherbereich, der wiederum einen Ringpuffer enthält.

Eventbuilder

Dem Eventbuilder werden durch zwei Domains (Datain) die (VME-Bus-) Adressen der beiden Subeventreader mitgeteilt.

Für die Datenausgabe wurden zwei Wege eingerichtet: Die Aufzeichnung der Daten auf das Bandgerät wird durch eine Domain (Parameter: Gerätename, höchste Priorität) veranlaßt. Eine weitere Dataout-Domain (Parameter: Netzwerkadresse der Workstation, untergeordnete Priorität) dient der Übermittlung von Eventdaten an das Programm zur Online-Analyse auf der Workstation.

4.1.3 Parameter der Messungen

Die hier gezeigten Daten wurden in einer Strahlzeit im Dezember 1994 aufgenommen.

Der COSY-Strahlimpuls war 800 MeV/c. In einem 5-Sekunden-Zyklus wurden jeweils etwa $5 \cdot 10^6$ Protonen extrahiert; die Extraktion dauerte jeweils 0.5 s.

BIG KARL war auf einen Mittenimpuls von 740 MeV/c eingestellt, was dem mittleren Impuls der Deuteronen im Ausgangskanal entspricht (siehe Bild 1.7). Der Bereich der Deuteronenimpulse ist etwas größer als die Akzeptanz von BIG KARL. Der Primärstrahl lag außerhalb des Akzeptanzbereiches.

Das Flüssigwasserstofftarget hatte einen Durchmesser von 6 mm und eine Dicke von 4.4 ± 0.2 mm

In einem 2 1/2 h dauernden Run wurden ca. 20000 Ereignisse aufgezeichnet (Datenlänge ca. 5.5 MByte). Die mittlere Datenrate liegt somit unter 1 kByte/s; das Datenaufnahmesystem war nur minimal beansprucht. (Das Verhältnis der Zahl der in die Totzeit der Datenaufnahme fallenden Trigger zur Gesamtzahl der Haupttriggerimpulse lag bei etwa 1%.)

Anschließend wurde eine 45-minütige Messung mit leerem Target durchgeführt.

4.2 Methoden der Auswertung

4.2.1 Spurrekonstruktion

Die Abbildungseigenschaften des Analysators BIG KARL sind nach den Gesetzen der Ionenoptik durch eine Transformation

$$\begin{pmatrix} x \\ x' \\ y \\ y' \\ l \\ \delta \end{pmatrix}_f = \begin{pmatrix} R_{11} & R_{12} & 0 & 0 & 0 & R_{16} \\ R_{21} & R_{22} & 0 & 0 & 0 & R_{26} \\ 0 & 0 & R_{33} & R_{34} & 0 & 0 \\ 0 & 0 & R_{43} & R_{44} & 0 & 0 \\ R_{51} & R_{52} & 0 & 0 & 1 & R_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ x' \\ y \\ y' \\ l \\ \delta \end{pmatrix}_t \tag{4.1}$$

zu beschreiben.

Hier sind

x, y: Position des Teilchens in der betrachteten Ebene

x', y': Komponenten des Winkels zwischen z-Achse und Teilchenbahn

l: Längendifferenz zwischen der betrachteten Teilchenbahn und einer zentralen (d.h., in Bahnmitte verlaufenden) Trajektorie

δ: relative Abweichung des Impulses zwischen dem betrachteten Teilchen und einem die zentralen Trajektorie beschreitenden Teilchen

Die Indices f und t stehen für Fokalebene bzw. Targetbereich.

Die obige Matrix ist als Reihenentwicklung über δ darzustellen:

$$R_{ij} = R_{ij}(\delta) = R_{ij}^{(0)} + R_{ij}^{(1)} \cdot \delta + R_{ij}^{(2)} \cdot \delta^2 + \cdots \tag{4.2}$$

Selbst wenn man nur die erste Ordnung dieser Reihe berücksichtigt, ist die Ausgleichsrechnung zur Bestimmung der R_{ij} sehr kompliziert. Bei ersten Versuchen der Auswertung führte sie nicht zu den gewünschten Ergebnissen.

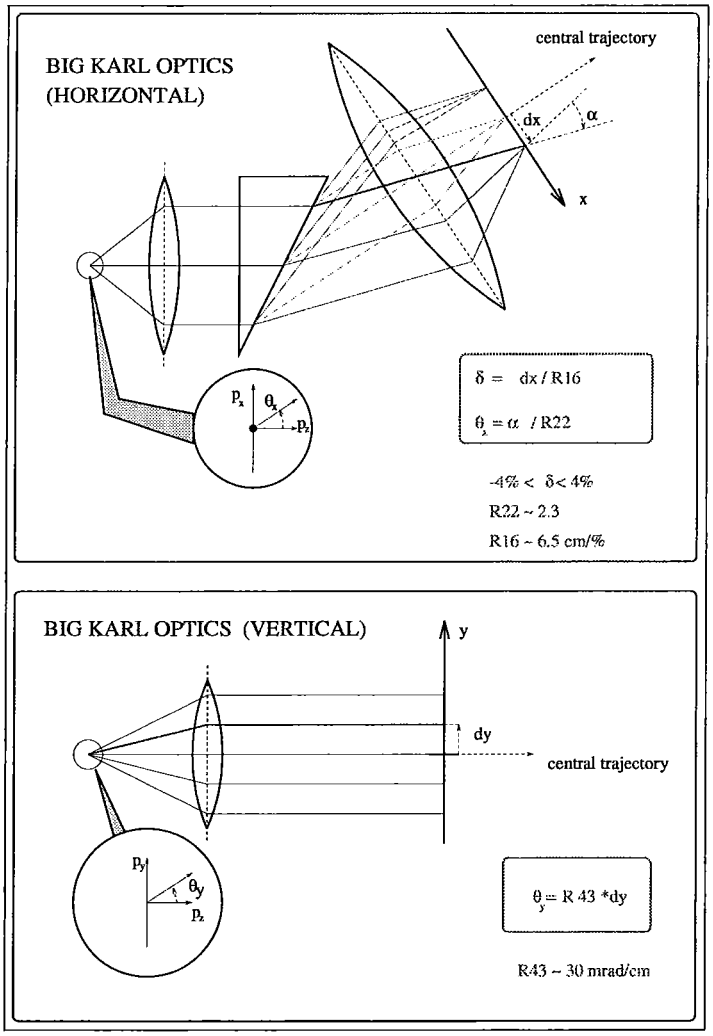


Abbildung 4.2: Optisches Modell für die Abbildung der Teilchenbahnen durch BIG KARL

Deshalb wurde ein vereinfachtes Modell angewendet, in Anlehnung an die gewöhnliche Strahlenoptik. Bild 4.2 zeigt einen Überblick. Es berücksicht die Betriebsweise von BIG KARL: Bezüglich der horizontalen Richtung fokussiert das Spektrometer ähnlich einer gewöhnlichen optischen Abbildung. Vertikal werden Punkte in Winkel transformiert. Im Bild sind diejenigen Elemente der Transformationsmatrix (4.1) gezeigt, die mit Hilfe des Modells eine anschauliche Bedeutung gewinnen. Die anderen Komponenten werden vernachlässigt.

Die so vorgenommene Vereinfachung mag wichtige Eigenschaften der Transformation ignorieren. Durch sie konnten aber erst die mathematischen Probleme bei der Rekonstruktion überwunden werden; für die gemessenen Daten ergaben sich plausible Resultate. Im Abschnitt 4.5 wird gezeigt, wie die erhaltenen Impulskomponenten weiterverarbeitet wurden und dabei versucht wurde, einen Teil der hier begründeten Fehler zu korrigieren.

4.2.2 Werkzeuge

Für die Auswertung der vom Datenaufnahmesystem gelieferten Rohdaten (sowohl online als auch bei der späteren Analyse) wurde das Programmpaket „XD“ ([79], [80]) verwendet. Der Hauptteil ist eine C++-Klassenbibliothek, sie kennt Objekte wie Histogramme, Scaler, Gates und Filter.

Die Bibliothek wird zusammen mit den anwendungsspezifischen Routinen (Ein- und Ausgabe, Dekodierung des Eventformats, evtl. Spurrekonstruktion) zu einem experimentspezifischen „Sorter“ verarbeitet, einem Programm, das die Rohdaten nach wählbaren Kriterien in Spektren und ähnliche aufbereitete Informationen konvertiert.

„XD“ selbst enthält in der verwendeten Version keine Möglichkeiten zur graphischen Aufbereitung der Daten, zur Darstellung dient das CERN-Programm PAW [81]. Dieses liest die darzustellenden Daten (ein- und zweidimensionale Histogramme) über einen geteilten Speicherbereich — es ist keine Anpassung nötig.

4.3 Resultate aus Teilsystemen

4.3.1 Zeitverteilung der Ereignisse

Die mit den Eventdaten aufgezeichneten Zeitmarken (siehe Abschnitt 4.1.1) waren ursprünglich dazu bestimmt, eventuelle Korrelationen von Strahlparametern mit der Zeit innerhalb der COSY-Extraktion zu diagnostizieren. Es wurden in dieser Beziehung keine signifikanten Abhängigkeiten gefunden.

Eine einfache Darstellung dieser Zeiten als Histogramm, wie in Bild 4.3, zeigt den Verlauf der Strahlintensität über der Extraktionszeit. Pro COSY-Spill wurden im Mittel 12 Ereignisse aufgezeichnet.

Eine Analyse der Zeitdifferenzen zwischen aufeineinanderfolgenden Ereignissen (Bild 4.4) zeigt einen exponentiellen Verlauf der Häufigkeitsverteilung. Das weist darauf hin, daß die Zeitverteilung (im auf diese Weise der Datenaufnahme zugänglichen Zeitrahmen — nach unten durch die Totzeit begrenzt) statistisch ist.

> Der Zusammenhang zwischen statistischer Verteilung und Exponentialverteilung sei der Vollständigkeit halber kurz gezeigt:
>
> Die Wahrscheinlichkeit, daß nach Eintreffen von N Primärteilchen n der betrachteten Ereignisse stattgefunden haben, ist
>
> $$p(n,N) = \binom{N}{n} \cdot \omega^n \cdot (1-\omega)^{N-n} \tag{4.3}$$
>
> (Binomialverteilung). ω sei die Wahrscheinlichkeit für die Einzelreaktion (Wirkungsquerschnitt × Targetdicke ... — von Fragen der Akzeptanz und der Nachweiseffizienz wird hier abstrahiert).
>
> Die Wahrscheinlichkeit, daß nach Eintreffen von N Primärteilchen noch kein Ereignis stattgefunden hat, ist also
>
> $$p(0,N) = (1-\omega)^N; \tag{4.4}$$

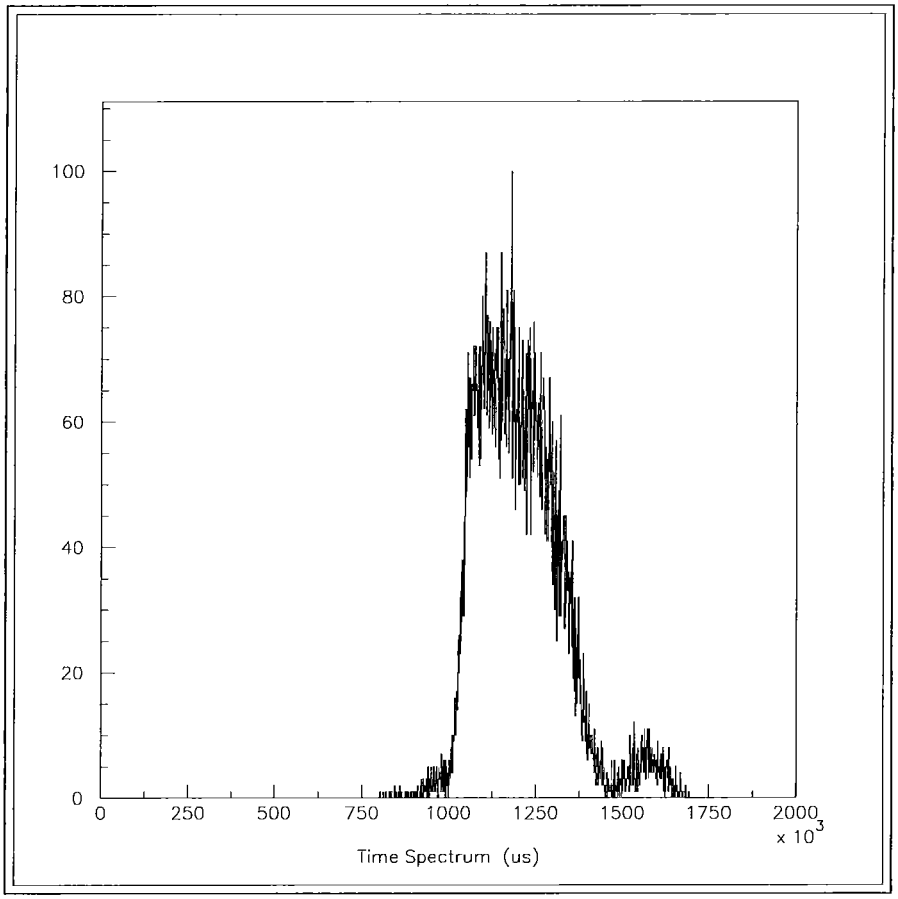


Abbildung 4.3: Zeitliche Verteilung der aufgenommenen Ereignisse im COSY-Zyklus

mit der Wahrscheinlichkeit

$$q(N) = 1 - p(0, N) = 1 - (1 - \omega)^N \tag{4.5}$$

hat nach N Teilchen mindestens eine Reaktion stattgefunden.

Das ist die Verteilungsfunktion, die die Anzahl der einlaufenden Teilchen (diskrete Zufallsgröße) bis zur nächsten Reaktion beschreibt. Die zugehörige Wahrscheinlichkeitsfunktion entsteht durch Differenzenbildung:

$$\Delta q(N) = q(N) - q(N-1) = p(0, N-1) - p(0, N) \tag{4.6}$$

Mit der Wahrscheinlichkeit $\Delta q(N)$ bewirkt genau das N-te Teilchen die nächste Reaktion.

Bei Übergang zu stetigen Größen erhält man analog:

$$p(0, N) = (1 - \omega)^N = e^{N \cdot \ln(1-\omega)} \tag{4.7}$$

$$q(N) = 1 - p(0, N) = 1 - e^{N \cdot \ln(1-\omega)} \tag{4.8}$$

$$\frac{dq(N)}{dN} = -\ln(1-\omega) \cdot (1-\omega)^N = -\ln(1-\omega) \cdot e^{N \cdot \ln(1-\omega)}$$

$$= \lambda \cdot e^{-\lambda \cdot N}, \quad \text{mit} \quad \lambda = -\ln(1-\omega) \tag{4.9}$$

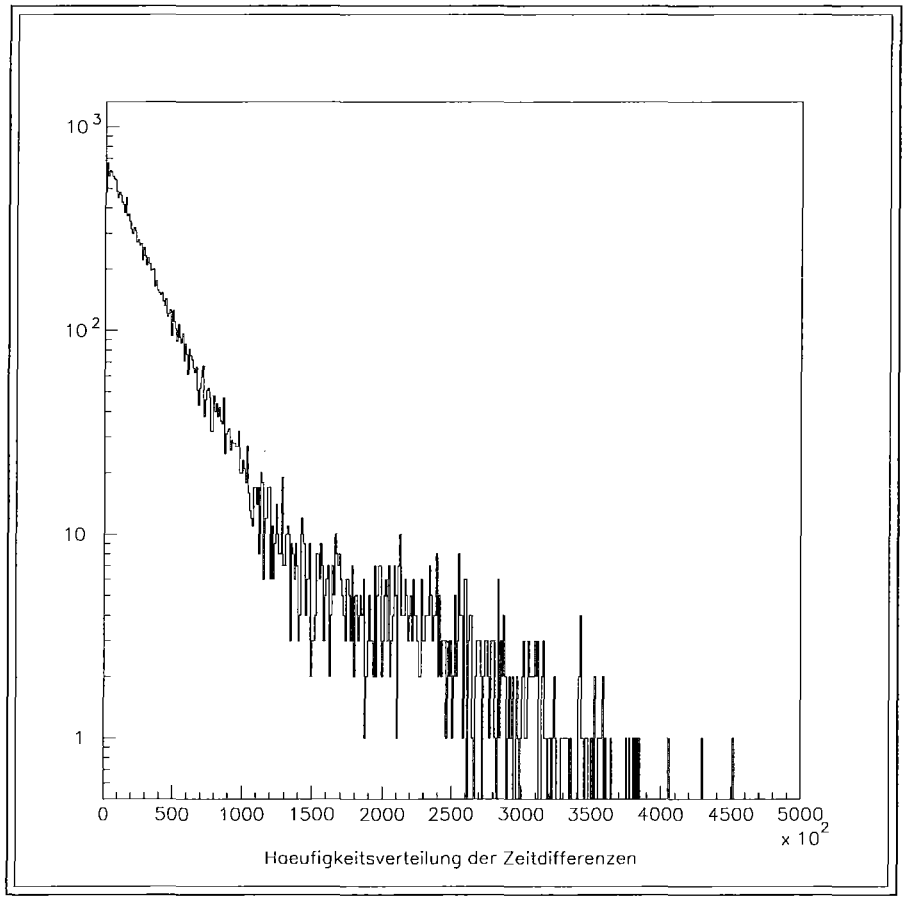


Abbildung 4.4: Verteilung der Zeitdifferenzen zwischen nacheinander aufgenommenen Ereignissen innerhalb eines Spills (Zeiteinheit: 1 μs)

Da wir die Verteilung der Zeiten betrachten, wird diese exponentielle Verteilung bei nichtkonstanter Strahldichte ($N \not\sim t$) von der Zeitstruktur des Strahls überlagert (genauer: N(t) ist eine weitere Wahrscheinlichkeitsverteilung, es muß über alle möglichen Anfangszeiten gemittelt werden — analog zur Autokorrelation). Dieser Einfluß ist klein für kleine betrachtete Zeitdifferenzen.

Zum anderen zeigt sich, daß der kleinste Abstand der Zeitmarken und damit die (minimale) Totzeit der Datenaufnahmeapparatur 244 μs beträgt.

Hier ergibt sich eine interessante Möglichkeit zur Abschätzung des Totzeitanteils bei der Messung: Es wird eine Exponentialverteilung (Gerade in Abb. 4.4) an die gemessene Verteilung angefittet:

$$\begin{aligned} f(t) &= e^{a+bt}, \quad \text{man erhält} \\ & a = 6.46 \quad \text{und} \\ & b = -0.343 \cdot 10^{-4} \mu s^{-1}. \end{aligned} \tag{4.10}$$

Der Anzahl der durch die Totzeit (t_t) nicht aufgezeichneten Ereignisse

kann dann durch

$$\int_0^{t_t} f(t)dt \tag{4.11}$$

abgeschätzt werden, der Anteil am Gesamtintegral $-e^a/b$ ist

$$1 - e^{b \cdot t_t}. \tag{4.12}$$

Mit $t_t = 244\ \mu s$ ist dieser Anteil ca. 1%; dieser Wert stimmt mit dem Resultat der Monitorzähler (siehe Abschnitt 4.1.3) überein.

4.3.2 Funktion der Detektoren

Die Drahtkammern zeigten eine Nachweiseffizienz nahe 100%. Bild 4.5

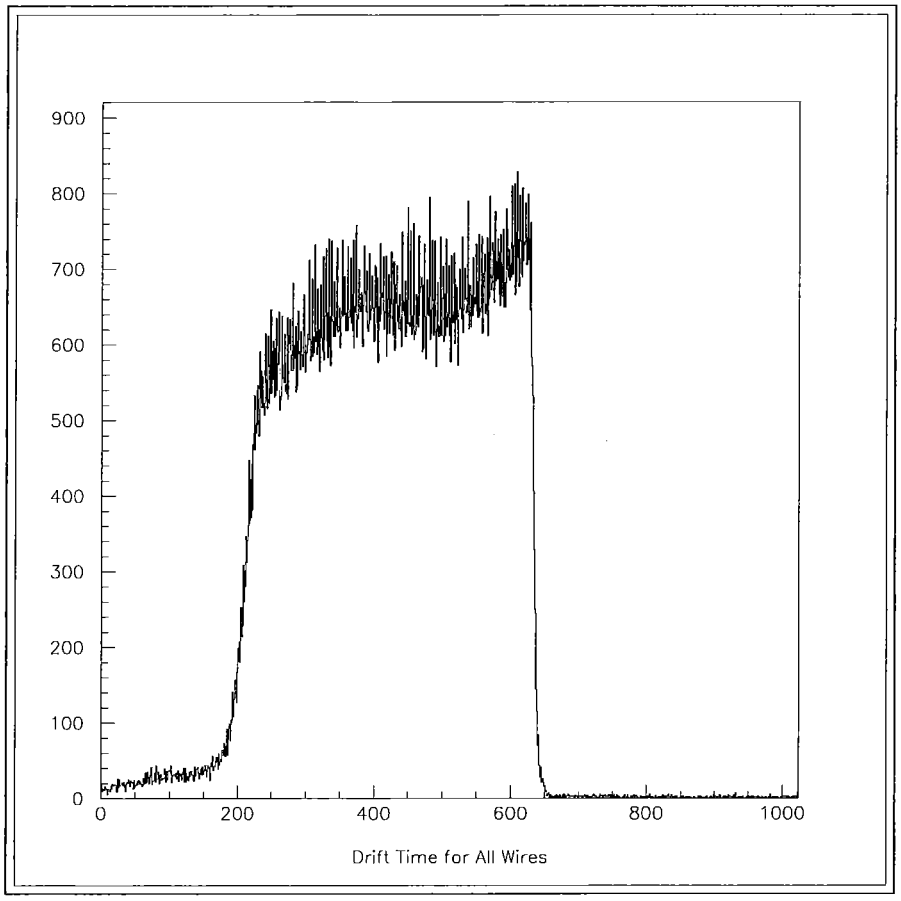


Abbildung 4.5: Häufigkeiten der Driftzeiten, Summe aller Driftkammern

zeigt die Häufigkeiten aller Driftzeiten für alle 12 Drahtkammern. Unter der Annahme, daß die Positionen der nachgewiesenen Teilchen modulo Zellbreite statistisch verteilt sind, kann daraus durch Integration die für die Spurrekonstruktion wichtige Driftzeit-Abstand-Beziehung gewonnen werden.

In Bild 4.6 sind die Zeitdifferenzen zwischen dem Teilchennachweis im Ringdetektor (Streukammer) und in der Fokalebene dargestellt. Im Fall der Reaktion

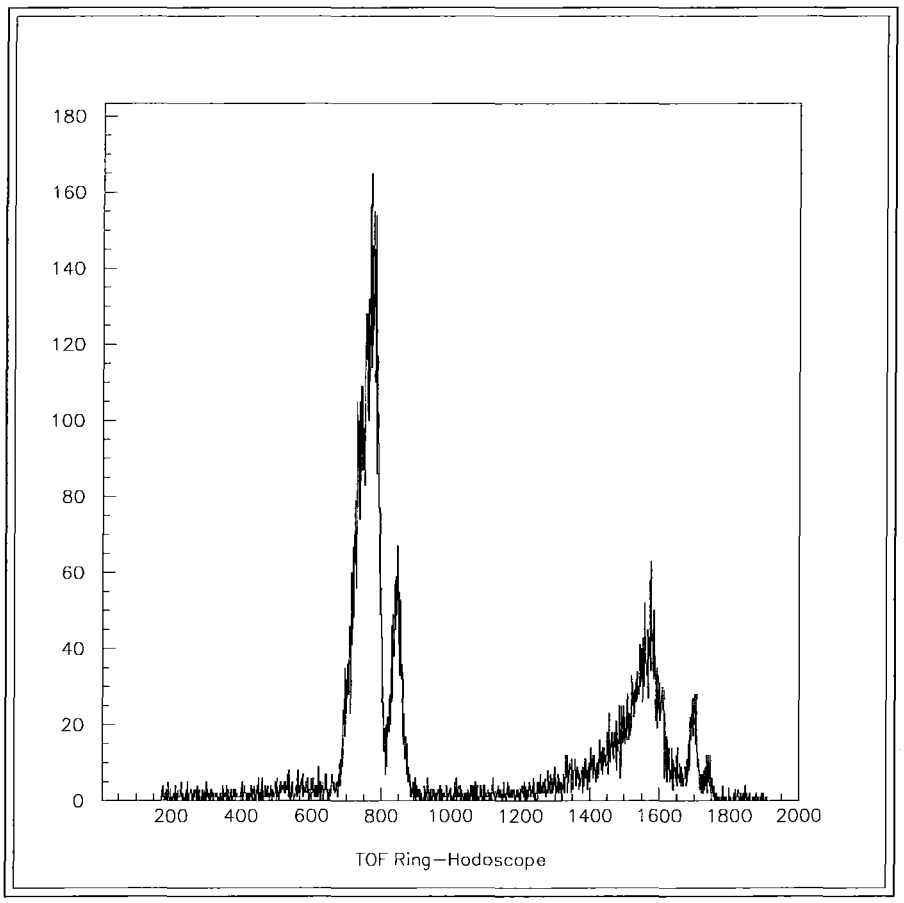


Abbildung 4.6: Spektrum der Zeitdifferenzen zwischen Ringdetektor und Fokalebene

$pp \rightarrow d\pi^+$ wird das π^+ im Ringdetektor und das Deuteron in der Fokalebene detektiert.

Die Flugzeit der Deuteronen ist, wie in Abschnitt 1.2.3 gezeigt, etwa doppelt so groß wie die der Untergrundteilchen (Protonen).

4.4 Rekonstruktion und Schnittbedingungen

Mit Hilfe des in Kapitel 4.2.1 gezeigten Modells wurden die in der Fokalebene gemessenen Spuren (Ort, Richtung) zum Targetpunkt rekonstruiert. Das Resultat zeigt Bild 4.7.

Nach Anwendung einiger Schnittbedingungen (Geometrie von BIG KARL, Flugzeit wie in Bild 4.6) ergab sich das in Diagramm 4.8 gezeigte Bild.

4.5 Aufbereitung der rekonstruierten Spurdaten

Die rekonstruierten Spuren der Deuteronen bestehen aus je einem Zahlentripel für jedes Ereignis: die Komponenten des Impulses in x- y- und z-Richtung

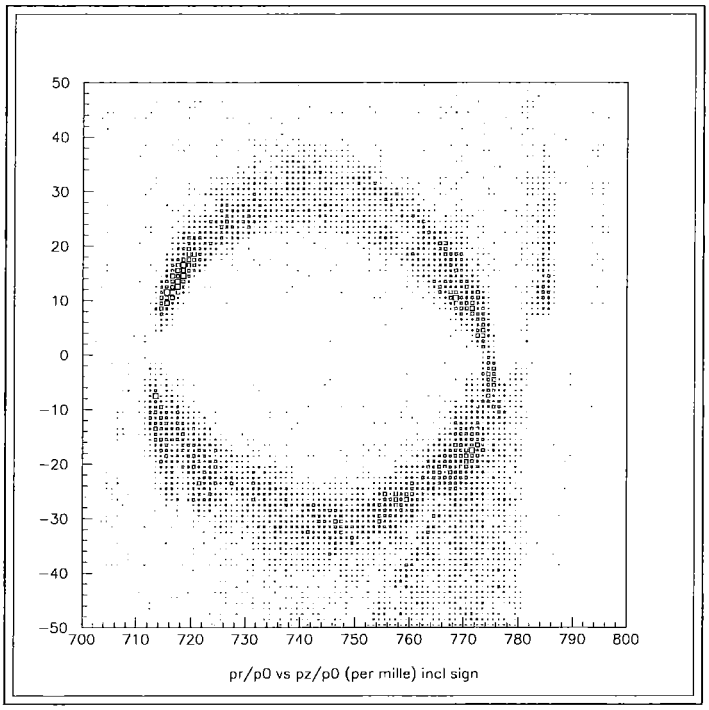


Abbildung 4.7: Rekonstruierte Impulskomponenten

in willkürlichen Einheiten (hier: Promille des BIG-KARL-Mittenimpulses). Im Idealfall müßten die Vektoren einen bezüglich der z-Achse symmetrischen Rotationsellipsoiden bilden, der in z-Richtung um den Faktor γ_{CM} (siehe Abschnitt 1.2.1) gestreckt ist. (γ_{CM} ist für die betrachteten Daten etwa 1.07.)

Die realen Daten sind in Bild 4.9 dargestellt. Es ist zu erkennen, daß infolge der begrenzten Akzeptanz von BIG KARL Teile des Körpers nicht besetzt sind (besonders wird die x-Richtung beschnitten).

Neben den durch Unschärfen und Streuungen verursachten, zufälligen, Fehlern können bei der Rekonstruktion verschiedene systematische Fehler auftreten:

- Wenn COSY-Impuls oder BIG-KARL-Einstellung ungenau sind, oder die Geometrie der Experimentieranordnung nicht genau justiert ist, verschiebt sich der gesamte Ellipsoid gegenüber dem Mittelpunkt (0, 0, 1000).

- Die Vereinfachungen im Modell für die Rekonstruktion und Ungenauigkeiten in den Parametern (R_{ij} im Kapitel 4.2.1) bewirken im allgemeinen (anisotrope) Skalierungen und Drehungen.

- Die Vernachlässigung der Terme höherer Ordnung in Gleichung 4.2 erzeugt zusätzliche Nichtlinearitäten.

Um die mögliche Verschiebung und die linearen Transformationen (Skalierungen und Drehungen) zu erfassen, wurde ein allgemeiner Ansatz für einen

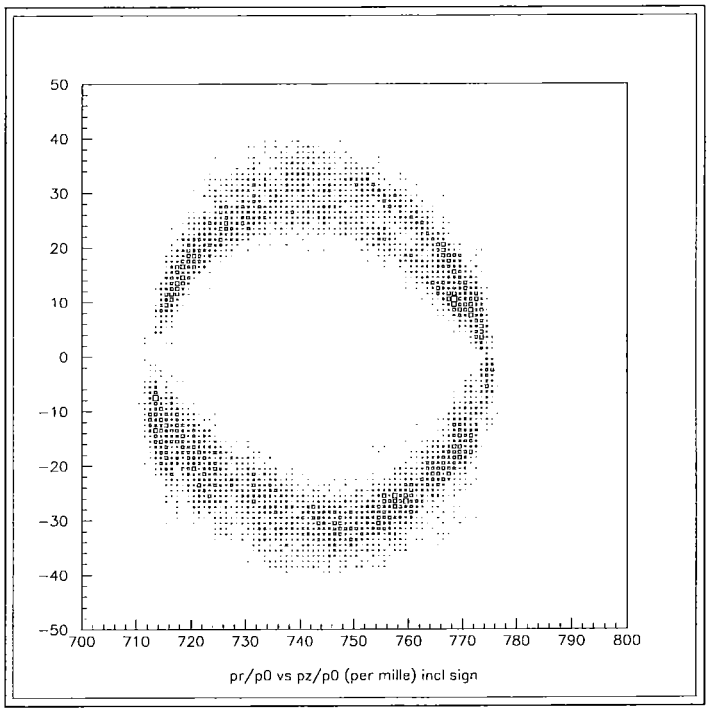


Abbildung 4.8: Rekonstruierte Impulskomponenten, nach Selektion (Flugzeitbedingung und Geometriecut)

Ellipsoiden

$$1 = \frac{(x-x_0)^2}{r_x^2} + \frac{(y-y_0)^2}{r_y^2} + \frac{(z-z_0)^2}{r_z^2} \tag{4.13}$$
$$+ a \cdot (x-x_0) \cdot (y-y_0) + b \cdot (x-x_0) \cdot (z-z_0) + c \cdot (y-y_0) \cdot (z-z_0)$$

durch eine Ausgleichsrechnung an die Daten angepaßt. Dies geschah unter Verwendung des kommerziellen Datenanalysepakets SAS[1] mit Hilfe des NEWTONschen Näherungsverfahrens.

Für die Parameter wurden folgende Werte ermittelt:

$$\begin{array}{lll} x_0 = -4.8, & y_0 = 0.2, & z_0 = 997.8 \\ r_x = 40.1, & r_y = 42.6, & r_z = 42.6 \\ a = 3.1 \cdot 10^{-4}, & b = 1.1 \cdot 10^{-4}, & c = 5.6 \cdot 10^{-5} \end{array} \tag{4.14}$$

Es zeigt sich, daß der erhaltene Ellipsoid signifikant vom Ideal abweicht:

- Verschiebung: ist moderat und aufgrund der Experimentbedingungen nicht unerwartet.
- Skalierung: r_y ist im Vergleich mit r_x und r_z zu groß.
- Drehung: Die Werte für die Mischterme (a, b, c) sind relativ groß — sie müssen mit $1/r_i^2$ verglichen werden.

[1] **S**tatistical **A**nalysis **S**ystem — [82]

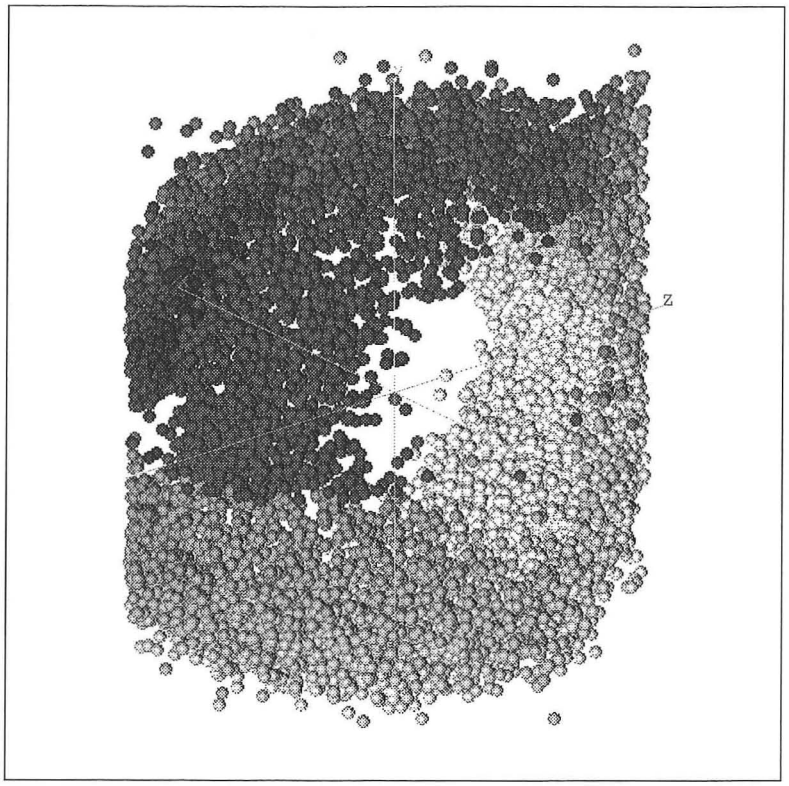


Abbildung 4.9: Rekonstruierte Impulse der Deuteronen. Die 8 Oktanten (durch die Koordinatenrichtungen definiert) sind durch verschiedene Farben gekennzeichnet.

Es wird eine Transformation gesucht, die diesen Ellipsoiden in eine Kugel überführt, die die Impulsvektoren im Schwerpunktsystem bilden müssen. Es wird angenommen, daß die Anwendung dieser Operation auf die Impulsdaten die Fehler der Rekonstruktion bestmöglich ausgleicht. Im Idealfall besteht die Transformation, wie oben beschrieben, aus einer Verschiebung um (0,0,-1000) und einer Skalierung in z-Richtung um den Faktor $1/\gamma_{CM}$. Auf die realen Experimentdaten soll nach der Verschiebung um den ermittelten Mittelpunkt (x_0, y_0, z_0 in Gl. 4.14) eine aus Skalierungen und Drehungen bestehende, lineare, Transformation angewendet werden, die sich durch eine Matrix ausdrücken läßt.

Offensichtlich existieren unendlich viele solche Transformationen: Kugeln sind invariant gegenüber beliebigen Drehungen. Hier muß nach Kriterien der physikalischen Plausibilität ausgewählt werden.

Zuerst erfolgt der Übergang zur Matrixschreibweise: Mit Hilfe der beiden Ersetzungen

$$\mathbf{A} = \begin{pmatrix} \frac{1}{r_x^2} & \frac{a}{2} & \frac{b}{2} \\ \frac{a}{2} & \frac{1}{r_y^2} & \frac{c}{2} \\ \frac{b}{2} & \frac{c}{2} & \frac{1}{r_z^2} \end{pmatrix} \tag{4.15}$$

und

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{4.16}$$

kann der durch Gleichung 4.13 definierte Ellipsoid wie folgt dargestellt werden:

$$\vec{r}^T \cdot \mathbf{A} \cdot \vec{r} = 1 \tag{4.17}$$

Für die Matrix **A** wurden die durch

$$\mathbf{A} \cdot \vec{e_i} = \lambda_i \cdot \vec{e_i} \tag{4.18}$$

definierten Eigenwerte und Eigenvektoren bestimmt. Für die gewählte symmetrische Darstellung der Matrix erhält man reelle Eigenwerte und orthogonale Eigenvektoren.

Die Eigenvektoren bestimmen die Lage der Hauptachsen im Raum, die Hauptachsenlängen sind $1/\sqrt{\lambda_i}$.

Es wurden folgende Werte ermittelt:

$$\begin{array}{ccc} \vec{e_1} = \begin{pmatrix} 0.76 \\ 0.59 \\ 0.28 \end{pmatrix}, & \vec{e_2} = \begin{pmatrix} 0.64 \\ -0.76 \\ -0.12 \end{pmatrix}, & \vec{e_3} = \begin{pmatrix} -0.14 \\ -0.27 \\ 0.95 \end{pmatrix} \\ \lambda_1 = 7.7 \cdot 10^{-4}, & \lambda_2 = 4.2 \cdot 10^{-4}, & \lambda_3 = 5.3 \cdot 10^{-4} \end{array} \tag{4.19}$$

Die Winkel zwischen den Haupt- und den Koordinatenachsen sind groß (41° zur x- und y-Achse). Die Lage der abgeschnittenen Gebiete in Bild 4.9 entspricht jedoch sehr gut unserem (bisherigen) Wissen über die Akzeptanz von BIG KARL. So muß der naheliegende Gedanke, die Hauptachsen in die Koordinatenrichtungen zu drehen, verworfen werden.[2]

Statt dessen wird der Ellipsoid lediglich entlang der errechneten Hauptachsen auf Kugelform skaliert. Die formale Operation wird im folgenden beschrieben:

Zur Vereinfachung der Schreibweise werden die Matrizen

$$\mathbf{X} = \begin{pmatrix} \vdots & \vdots & \vdots \\ \vec{e_1} & \vec{e_2} & \vec{e_3} \\ \vdots & \vdots & \vdots \end{pmatrix} \tag{4.20}$$

und

$$\mathbf{L} = \begin{pmatrix} \sqrt{\lambda_1} & & 0 \\ & \sqrt{\lambda_2} & \\ 0 & & \sqrt{\lambda_3} \end{pmatrix} \tag{4.21}$$

definiert. Nun kann das System der Eigenwertgleichungen (4.18) durch

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{X} \cdot \mathbf{L}^2 \tag{4.22}$$

[2]Die Berechnung der Hauptachsenrichtungen ist kritisch, da die Rohdaten nahezu in Kugelform sind und die Daten im Vergleich zu den Asymmetrien erheblich gestreut sind.

ausgedrückt werden.

Die gesuchte Transformation ist dann

$$\vec{r}' = \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{X}^T \cdot \vec{r}. \tag{4.23}$$

Daß diese Operation zu einer Kugel führt, sei kurz nachgewiesen:
$\mathbf{X}$ ist, wie oben erwähnt, orthogonal:

$$\mathbf{X}^{-1} = \mathbf{X}^T \tag{4.24}$$

Somit lautet die Auflösung von (4.23) nach $\vec{r}$

$$\vec{r} = \mathbf{X} \cdot \mathbf{L}^{-1} \cdot \mathbf{X}^T \cdot \vec{r}'. \tag{4.25}$$

Damit wird aus der Ellipsengleichung (4.17)

$$\vec{r}'^T \cdot \mathbf{X} \cdot \mathbf{L}^{-1} \cdot \mathbf{X}^T \cdot \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{L}^{-1} \cdot \mathbf{X}^T \cdot \vec{r}' = 1. \tag{4.26}$$

(Es gilt $\mathbf{L}^{-1^T} = \mathbf{L}^{-1}$!)

Mit den Eigenwertgleichung (4.22) ergibt sich nach Vereinfachung unter Anwendung der Orthogonalitätsbedingung (4.24)

$$\vec{r}'^T \cdot \vec{r}' = \left|\vec{r}'\right|^2 = 1. \tag{4.27}$$

Die $\vec{r}'$ liegen also auf der Einheitskugel[3].

Für die betrachteten Daten hat die Transformationsmatrix (zerlegt in einen Skalierungsfaktor und eine Matrix mit der Determinante 1) folgende Gestalt:

$$\mathbf{X} \cdot \mathbf{L} \cdot \mathbf{X}^T = \frac{1}{42.34} \cdot \begin{pmatrix} 1.045 & 0.138 & 0.048 \\ 0.138 & 0.985 & 0.022 \\ 0.048 & 0.022 & 0.992 \end{pmatrix} \tag{4.28}$$

Die Auflösung des Experimentes bezüglich des Impulsbetrages kann bestimmt werden, indem die Streuung der Beträge der $\vec{r}'$ ermittelt wird. Eine Analyse der Daten ergibt eine GAUSS-förmige Verteilung, die Standardabweichung liegt bei 8.0% des Kugelradius[4].

Zur Veranschaulichung der Struktur der Daten und der Akzeptanz von BIG KARL wurde die durch Gleichung 4.23 beschriebene Transformation auf vorher geglättete Daten angewendet. Das Resultat zeigt Bild 4.10.

Die Glättung erfolgte durch Berechnung neuer y-Impulskomponenten nach Gleichung 4.13. Entlang welcher Koordinate geglättet wird, unterliegt einer gewissen Willkür. Entscheidendes Kriterium war im vorliegenden Fall das vom Augenschein „schönste“ Resultat. Weitere Motivationen sind die Verteilung der Punkte in Bild 4.9, die deutliche Abweichungen besonders in y-Richtung zeigt, und die Abweichung des Parameters r_y in den gefitteten Daten (4.14).

[3]Die Transformation 4.23 kann auch als Kombination — $\mathbf{X}^T$=Drehung ins Hauptachsensystem, $\mathbf{L}$= Skalierung entlang der Achsen, $\mathbf{X}$=Rückdrehung — beschrieben werden.

[4]Das ist ein Maß für die Auflösung im Schwerpunktsystem. Auf den Impuls im Laborsystem bezogen, entspricht dieser Wert ca. 0.3%.

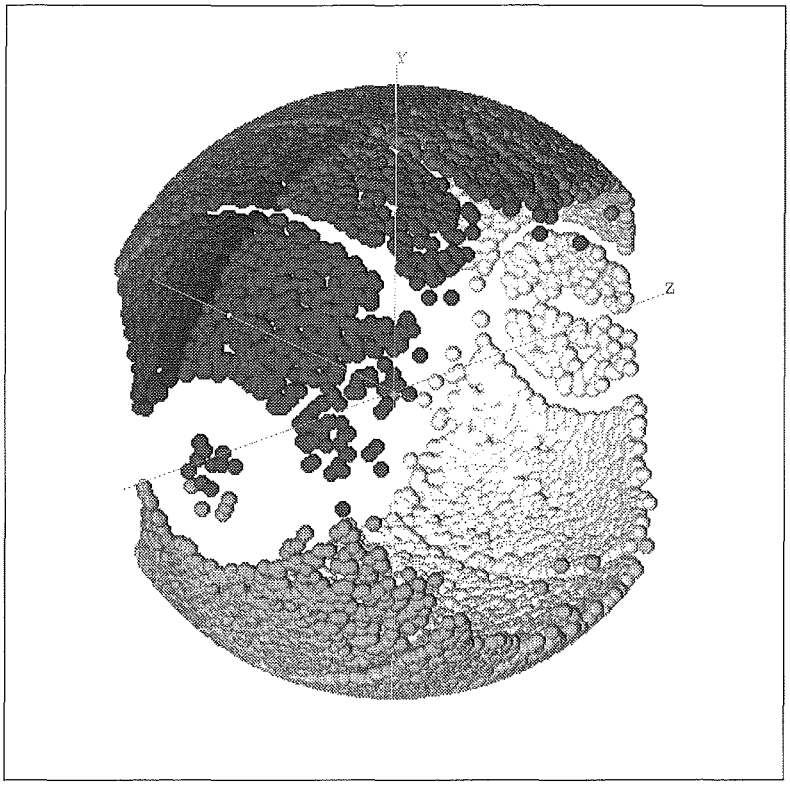


Abbildung 4.10: Rekonstruierte Impulse der Deuteronen, geglättet und auf eine Kugelfläche transformiert. Die 8 Oktanten (durch die Koordinatenrichtungen definiert) sind durch verschiedene Farben gekennzeichnet.

Bild 4.10 zeigt, daß mehrere charakteristische Bereiche auf der Kugeloberfläche unbesetzt sind. Ursache dafür ist, daß die Fokalebene von BIG KARL nicht lückenlos mit den zur Triggerung verwendeten Szintillatoren bedeckt war, sondern zwischen den einzelnen Streifen ein gewisser Abstand (ca. 1cm) bestand. Weiter ist erkennbar, daß an den Durchstoßpunkten der z-Achse keine Ereignisse aufgezeichnet wurden, da die zugehörigen, unter 0° gestreuten, Pionen das Loch im zur Triggerung verwendeten Ringdetektor passieren (vgl. Bild 1.14).

Um die Verteilung der gemessenen Ereignisse bezüglich des Raumwinkels zu erfassen, wurden die (transformierten, aber nicht geglätteten) Daten in Bild 4.11 auf die Oberfläche der „Soll"-Kugel und auf die Strahlachse projiziert. Die obere Abbildung ist dabei flächentreu, d.h., gleiche Flächenabschnitte entsprechen gleichen Teilen des Raumwinkels, und die Belegungsdichte ist dem differentiellen Wirkungsquerschnitt $d\sigma/d\Omega$ proportional — hier fehlt allerdings noch eine die anisotrope Nachweiswahrscheinlichkeit ausgleichende Korrektur. Sinngemäßes, nur eindimensional, gilt für das untere Bild.

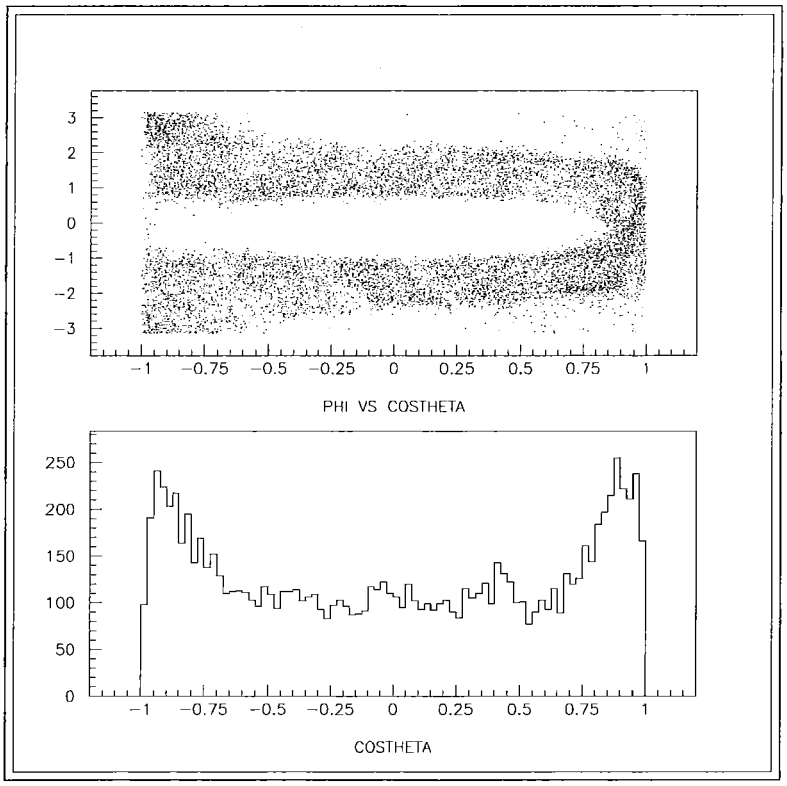


Abbildung 4.11: Projektion der Oberfläche der kinematischen Kugel.

4.6 Behandlung der Nachweiswahrscheinlichkeit

Aus der in Bild 4.11 gezeigten Verteilung der Ereignisse über den Kosinus der Breitenkoordinate soll eine Aussage über die Verteilung des differentiellen Wirkungsquerschnittes $d\sigma/d\theta$ gewonnen werden (das Verhältnis der Parameter a_2 und a_0 in Gleichung 1.1).

Dazu muß die Nachweiswahrscheinlichkeit für die einzelnen Raumwinkelbereiche berücksichtigt werden. Aussagen über diese wurden nach 2 Methoden ermittelt:

1. Monte-Carlo-Simulation, unter vereinfachenden Annahmen (einfache Geometrie, keine Bahnkrümmung durch das Magnetfeld).

2. Ausmessen des Akzeptanzbereiches in Abb. 4.11 — oberes Diagramm —. Die Methode geht von einem scharf beschnittenen Gebiet in der θ-ϕ-Fläche aus, in dem alle Ereignisse nachgewiesen werden, während im komplementären Teil die Wahrscheinlichkeit dafür 0 beträgt.

   Als innerhalb des Akzeptanzbereiches liegend wurden die Punkte angesehen, an denen die gemessene Ereignisdichte eine gewisse, empirisch ermittelte, Schwelle überstieg. Die Nachweiswahrscheinlichkeit für ein gegebenes θ entspricht dann dem Teil des Breitenkreises (in der Projektion:

senkrechte Linie), der innerhalb des Akzeptanzbereiches liegt.

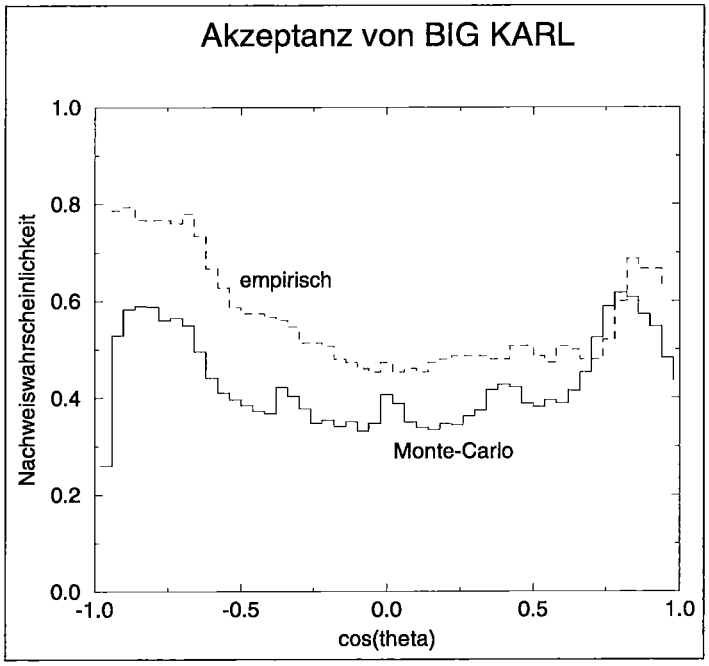


Abbildung 4.12: Akzeptanz von BIG KARL in Abhängigkeit von $cos(\theta)$, nach 2 verschieden Methoden ermittelt

Bild 4.12 zeigt die so erhaltenen Wahrscheinlichkeitsverteilungen.

Die mit diesen Korrekturen behandelten Häufigkeitsverteilungen zeigt Bild 4.13. Die Resultate in den Randbereichen, entlang der Durchstoßpunkte der Strahlachse, sind dabei prinzipbedingt mit einer großen Unsicherheit behaftet (simulierte Nachweiswahrscheinlichkeit ist nahe 0, Singularität bei der Projektion in Kugelkoordinaten).

4.7 Resultate, Diskussion

Welcher Wert für das gesuchte Verhältnis a_2/a_0 bei einer Anpassungsrechnung resultiert, ist stark davon abhängig, welche Datenpunkte in die Rechnung eingehen.

Für die hier gezeigten Resultate wurden die nach der Monte-Carlo-Methode (siehe voriges Kapitel) korrigierten Verteilungen verwendet. Nachdem die der z-Achse nächsten Punkte ausgeschlossen wurden, ergab sich der Wert

$$\frac{a_2}{a_0} = 0.019 \pm 0.018 \pm 0.018, \tag{4.29}$$

für die Anisotropie, was deutlich unter den Vergleichswerten aus der Literatur ($\approx$0.3 [14]) liegt.

Für den totalen Wirkungsquerschnitt wurde

$$\sigma = 49.1 \pm 0.7 \pm 5.4 \mu b \tag{4.30}$$

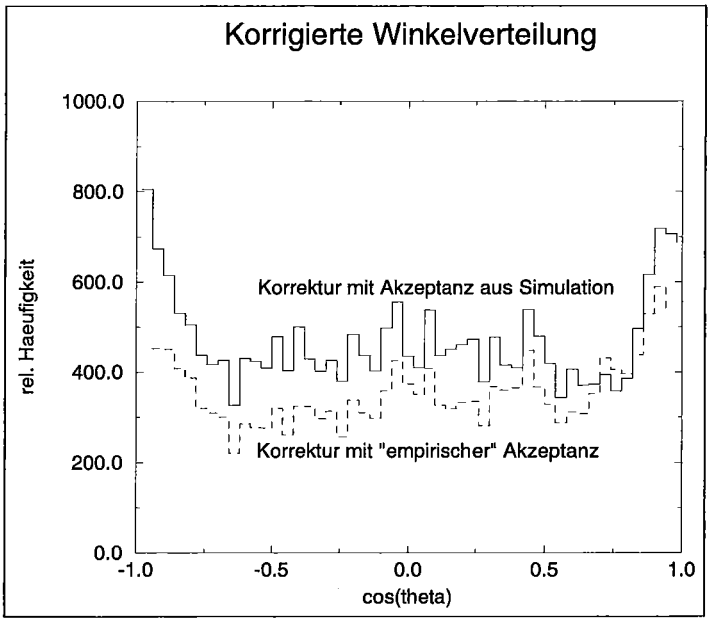


Abbildung 4.13: Akzeptanzkorrigierte Verteilung der Ereignishäufigkeit über $cos(\theta)$ (Korrektur mit den Wahrscheinlichkeitsverteilungen aus Abb. 4.12)

ermittelt. Das entspricht den Daten aus [15].

Die angegebenen Toleranzen sind dabei (in dieser Reihenfolge) statistischer und systematischer Fehler.

Bei Anwendung der empirischen Akzeptanzkorrektur erhält man einen um ca. 20% kleineren Wirkungsquerschnitt. Zur Verbesserung des dabei verwendeten Modells wären noch Korrekturen sinnvoll, die die Nachweiswahrscheinlichkeit der Szintillatoren in der Fokalebene (speziell die in Bild 4.10 sichtbaren, bei der Dichtebestimmung teilweise herausgemittelten, Lücken) und den Anteil nichtrekonstruierbarer Spuren in den Driftkammern berücksichtigen.

Die Situation bezüglich der Nachweiswahrscheinlichkeit ist etwas unbefriedigend. Die Unterschiede zwischen den beiden ermittelten Verteilungsfunktionen deuten darauf hin, daß (a) die Simulation ungenau ist und/oder (b) der Restfehler aus der Spurrekonstruktion einen signifikanten Einfluß auf die Winkelverteilung hat.

In Bild 4.9 (oben rechts) sind vom restlichen kinematischen Gebilde abweichende Impulse zu erkennen. Auch die etwas stärker besetzten Gebiete in Abb. 4.11 (rechter und linker Rand) lassen auf verbliebenen Untergrund schließen. Da aber die Verteilung der Impulsbeträge nach der Transformation, wie auf S. 89 erwähnt, eine Glockenkurve beschreibt, ergibt sich (anhand der Impulsdaten) kein Anhaltspunkt für eine zusätzliche Auswahlbedingung.

Um belastbare Aussagen zu treffen, sollten weitere Messungen mit einer höheren Ereigniszahl berücksichtigt werden und eine größere Sicherheit in der Abschätzung der Nachweiswahrscheinlichkeit gewonnen werden.

Anhang A

Leistungsmessungen

A.1 Generelles

Der Hauptzweck eines Datenaufnahmesystems ist es, die in den Digitalisierungsmodulen nach einem Triggerereignis entstehenden Daten zum Ort der Speicherung oder der Auswertung zu transportieren. Als Maß, wie gut dieser Zweck erfüllt wird, werden die erreichbare Triggerfrequenz (-rate) f und der Datendurchsatz (*Datenmenge/Zeit*) D angesehen, beide mit der einfachen Beziehung

$$D = f \cdot N \tag{A.1}$$

verknüpft, wobei N die Menge der zu einem Triggerereignis gehörenden Daten (Eventgröße) ist. Das gilt sowohl für stationäre Zustände (Rate und Eventgröße bleiben konstant) als auch, im allgemeinen Fall, für jedes beliebige Zeitintervall. (Die Größen sind dann als Mittelwerte zu verstehen.)

Die Maximalfrequenz f ist das Reziproke der für die Bearbeitung eines Events minimal nötigen Zeit T. Diese wiederum läßt sich in in einem einfachen Modell durch

$$\frac{1}{f} = T = T_c + N \cdot T_t \tag{A.2}$$

beschreiben. Der konstante Teil T_c ist auf Systemlatenzen und Programmlaufzeiten (Verwaltung, Konsistenztests) zurückzuführen, während der zweite Summand die für den eigentlichen Datentransport benötigte Zeit beschreibt. T_t ist die für die Übertragung einer Dateneinheit gebrauchte Zeit, oder auch das Reziproke der nutzbaren Bandbreite des gewählten Transportmediums.

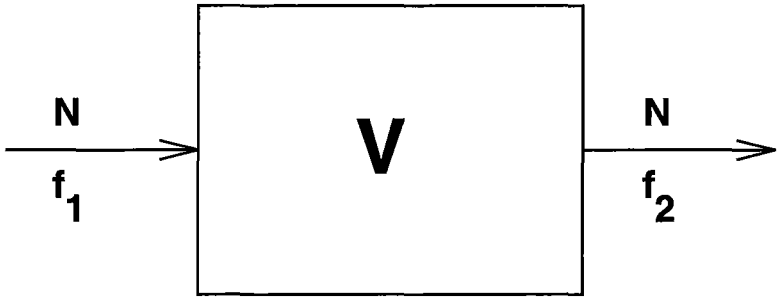
Für einfache, nur aus einer einzelnen Übertragungsstrecke bestehende, Systeme sind damit die wesentlichen Zusammenhänge dargestellt.

Komplexere Anordnungen, wie die hier untersuchten, bestehen aus einer Verknüpfung solcher Übertragungsstrecken. Dabei entstehen vielfältige Abhängigkeiten, die mit diesem Ansatz bei weitem nicht ausreichend beschrieben werden können, da die Kombination der Teilsysteme, zusammen mit neuen Komponenten, wie Datenpuffern und Synchronisationsbedingungen, gänzlich neue Effekte in Erscheinung treten läßt. Dessen ungeachtet sind die obigen Formeln durchaus geeignet, z.B. das Verhalten einzelner Teile eines größeren Systems unter einfachen Bedingungen zu beleuchten. In den Abschnitten, die

sich mit dem Durchsatz einzelner Komponenten des Datentransports befassen, werden immer wieder solche linearen Ansätze verwendet.

In den folgenden Punkten sollen einige der neuen, komplexeren Gesichtspunkte im einzelnen dargestellt werden:

Pufferung: Datenpuffer sind Speicherbereiche oder auch FIFOs[1], in denen ein oder mehrere Events zwischengespeichert werden können. Sie werden von der Datenquelle (einem auslesenden Controller) gefüllt und von einem folgenden Controller über einen im allgemeinen verschiedenen Datenweg gelesen. Je nach Organisation kann ein Puffer entweder ein bestimmtes Volumen V haben, das gegebenenfalls auf mehrere Events aufgeteilt wird, oder eine bestimmte Anzahl von Events (mit einer Maximalgröße) fassen. Hier wird, entsprechend der vorhandenen Implementierung, der Fall eines vorgegebenen Volumens betrachtet.



Ein- und Ausgabeprozesse sind voneinander unabhängig, die beiden transportierenden Busse zeitlich entkoppelt. Die Frequenzen f_1 und f_2 können deshalb verschieden sein. Wegen der endlichen Größe der Datenpuffer müssen jedoch die zeitlichen Mittel beider Frequenzen übereinstimmen:

$$\bar{f}_1 = \bar{f}_2 \tag{A.3}$$

Abweichungen sind nur für eine begrenzte Zeit

$$t_{diff} \leq \frac{V}{N \cdot |f_1 - f_2|} \tag{A.4}$$

möglich.

Die Pufferung von Daten bringt in folgenden Fällen Vorteile mit sich:

- Die Frequenz der Datenquelle ist nicht konstant, die Daten fallen in Schüben (Bursts) an. Das ist z.B. im Frontend von physikalischen Experimenten mit gepulstem Strahl üblich. Der Durchsatz des ersten, auslesenden Bussystems ist höher als der der folgenden Komponenten.

  Dann wird infolge der Pufferung die Geschwindigkeit des Auslesens der Digitalisierungshardware nicht mehr vom Durchsatz des restlichen Datenaufnahmesystems beeinflußt (siehe auch A.3). Während

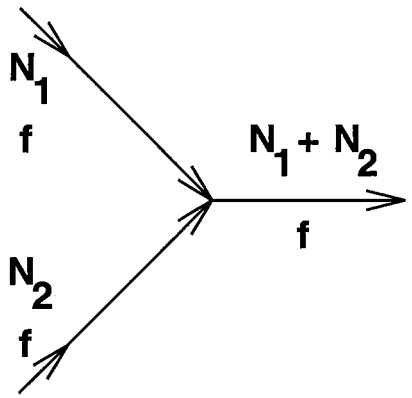
[1] **F**irst in - **F**irst out: spezielle (integrierte) Schaltungen zur Zwischenspeicherung von Daten

der Auslese ist dann $f_1 > f_2$. Die Puffergröße sollte, wenn möglich, nach der zu erwartenden Burstdauer gemäß Formel (A.4) bemessen werden.

- Der Durchsatz des Daten abtransportierenden Bussystems ist höher als der des anliefernden. Der abtransportierende Controller könnte also, wenn nur ein Teil seiner Zeit für den Datentransport benötigt wird, andere Aufgaben erfüllen. Hier wäre für die aktive Zeit $f_1 < f_2$, und der Puffer müßte mindestens 1 Event fassen.

Parallelisierung: Mehrere Datenwege (Busse, serielle Verbindungen o.ä.) können, wenn entsprechend mehrere Controller vorhanden sind, gleichzeitig genutzt werden, um Daten auf parallelen Wegen zu transportieren. Der Gesamtdurchsatz des Systems kann im Idealfall die Summe der einzelnen Teile erreichen.

Genauere Betrachtung verdient das Zusammenfügen der Teileventströme zu einem Gesamteventstrom. Die Synchronisation bedingt, daß die Eventfrequenz an den Eingängen und am Ausgang gleich ist. (Ein Event soll je einen —möglicherweise leeren— Datenblock in jedem Frontendcrate erzeugen. In den Eventbuildern werden dann diese Blöcke zu größeren Einheiten zusammengefügt.)



Ein Controller muß die Daten beider Eingangskanäle lesen und in den Ausgangskanal schreiben. Das kann (mit der zur Verfügung stehenden Hardware) nur nacheinander geschehen. Um also die Bandbreiten beider Zubringerbusse voll ausnutzen zu können, ist eine Pufferung vonnöten, die die Daten der Zubringerbusse zwischenspeichert, während der Controller die jeweils anderen Eingangskanäle liest. Das entspricht dem zweiten oben unter „Pufferung“ genannten Einsatzfall.

Rückkopplungen: Die Lage kompliziert sich weiter, wenn Wechselwirkungen zwischen verschiedenen Teilen des Systems stattfinden oder Beeinflussungen entgegengesetzt zur Richtung des Datentransports auftreten. Beispiele dafür sind:

- Ein Prozessor arbeitet (pollt) im Speicher des Controllers, von dem er Daten liest. Unter Umständen benötigen beide Prozessoren Daten

aus dem selben Speicherbereich, oder der CPU-Bus wird beim externen Zugriff blockiert. Das kann den Controller sowohl beim Daten- als auch beim Programmcodezugriff verlangsamen.

- Die Frontendkomponenten werden über entsprechende Hardware miteinander synchronisiert. Ein Triggerimpuls wird dadurch solange verhindert, wie mindestens einer der auslesenden Controller nicht zur Datenaufnahme bereit ist. Das bewirkt, daß jeweils der langsamste in dieses Triggersystem eingeschlossene Teil die Geschwindigkeit des Gesamtsystems vorgibt und unter Umständen Durchsatzverbesserungen in anderen Teilen nutzlos sein können.
- Schwingungsähnliche Effekte können auftreten, wenn in Verbindung mit Datenpuffern Füllstandsschwellen (High-/Low-Watermarks) verwendet werden. Diese dienen der Effektivität von Blocktransfers und bewirken, daß nur dann Daten transportiert werden, wenn eine gewisse Mindestmenge zur Verfügung steht. Die dadurch bewirkte schubweise Datenübertragung kann infolge der Bedingungen (A.3) und (A.4) über mehrere Stationen zurückwirken.

Multitasking-Systeme: In den heutigen preemptiven[2] Multitasking-Betriebssystemen werden den verschiedenen aktiven Programmen (Prozessen) Zeitscheiben fester Größe nach bestimmten Prioritätskriterien zugeteilt. Das hat zum einen trivialerweise zur Folge, daß nicht die gesamte Rechenleistung eines Controllers dem datenübertragenden Programm zur Verfügung steht, sondern auch mit konkurrierenden Prozessen und dem Verwaltungsaufwand der Prozessorzeitzuteilung im System gerechnet werden muß.

Interessantere Effekte treten auf, wenn mehrere Prozesse in einem CPU-System auf einen gemeinsamen Datenpuffer zugreifen. So kann möglicherweise ein Prozeß in seiner Zeitscheibe nur einen gewissen Teil des Puffers füllen, bevor der andere ihn wieder leert, so daß der Puffer nur teilweise genutzt wird. Andererseits kann der Puffer auch schon nach Verstreichen eines Teils der Zeitscheibe gefüllt sein, und der Prozeß ist für den Rest zur Untätigkeit verdammt. Wie sich das in praktischen Leistungsdaten niederschlägt, ist in Abschnitt A.5 zu sehen.

Zusätzlicher Verwaltungsaufwand: Es liegt auf der Hand, daß zur Verwaltung eines komplexeren Datenaufnahmesystems, besonders wenn es konfigurierbar sein soll, ein höherer Verwaltungsaufwand als für vergleichsweise simple Anordnungen nötig ist.

Dazu kommen diverse Sicherheitsvorkehrungen und Konsistenzprüfungen. Die gegenwärtige Implementierung der Eventmanager testet z.B. zur Laufzeit jeweils die korrekte Anzahl von Subevents und die lückenlose Folge der in den Datenblocks abgelegten Eventzähler.

[2]Die Umschaltung zwischen den aktiven Prozessen wird durch Interrupts erzwungen, im Gegensatz zum kooperativen Multitasking, das darauf beruht, daß die Prozesse selbst ihre Unterbrechung ermöglichen.

Die Folge ist, daß der letztendlich erzielte Durchsatz hinter den theroretisch erreichbaren Werten zurückbleibt.

A.2 Zur Methodik der Messungen

Zweck der Messungen ist es nicht nur, zu zeigen, welcher Gesamtdurchsatz sich mit dem neu entwickelten Datenaufnahmesystem erzielen läßt. Vielmehr sollen durch gezielte Meßreihen Informationen darüber gewonnen werden, wie leistungsfähig jede einzelne Komponente der Anordnung ist und wo die Schwachstellen des Systems liegen. Wenn es für ein gegebenes Problem mehrere Möglichkeiten der Konfiguration gibt, sind auch Hinweise nützlich, welche Variante die besten Resultate liefert.

Dazu müssen einzelne Controller oder Übertragungsstrecken einzeln untersucht werden. Es würde aber zu unrealistischen Ergebnissen führen, würde man nur einzelne Teile mittels spezieller Programme testen, die unter für sie optimalen Bedingungen laufen. Wollte man z.B. die Übertragungsrate zwischen einem Subeventreader und einem Eventbuilder messen, sollte man bedenken, daß es auch Aufgabe des Subeventreaders ist, Daten vom Frontend einzusammeln und einen Datenausgabepuffer zu verwalten. Sinngemäßes gilt für den Eventbuilder.

Auf der anderen Seite ist, um beim obigen Beispiel zu bleiben, die reale Leistung eines Subeventreaders durchaus davon abhängig, wie schnell er das ihm überantwortete Frontend auslesen kann, also von der Hard- und Software des jeweiligen Frontend-Controllers und der dortigen Übertragungsstrecke. Es wäre kaum möglich, aus den erhaltenen Resultaten das eigentlich Interessierende, den Durchsatz vom Subeventreader zum Eventbuilder zu extrahieren.

Es gilt also, die Einzelkomponenten für die separate Untersuchung nicht völlig aus ihrer vertrauten Umgebung herauszureißen, sondern die „Umwelt" lediglich so zu verändern, daß sie zwar vorhanden ist, den üblichen Verwaltungsaufwand verursacht, aber die eigentliche Messung nicht behindert oder verfälscht. Das geschieht, in dem sie (im Rahmen der Möglichkeiten) „unendlich schnell" gemacht wird. Das heißt, daß im verwendeten Beispiel der Subeventreader das Vorhandensein eines Frontends simuliert, also Datenblöcke bestimmter Länge in einen Ausgabepuffer schreibt, aber in Wirklichkeit keine Daten transportiert. Die Daten, die der Eventbuilder liest, entstehen im Subeventreader, es sind Daten ohne eine reale Bedeutung. Analog im Eventbuilder: Die Daten werden nicht etwa auf ein Speichenmedium geschrieben (was in vielen Fällen der langsamste Teil der gesamten Datenaufnahme ist), sondern „vergessen" — allerdings erst, nachdem sie die Ausgabepufferverwaltung beschäftigt haben.

Auf diese Weise wird das Gesamtsystem in überschaubare Tranches zerlegt, die voneinander möglichst unabhängig je einen Teilaspekt des Systems repräsentieren. Oft folgt das Verhalten eines solchen Subsystems recht gut der Formel (A.2) oder ähnlich simplen Modellen. Wenn sich unerwünschte Abhängigkeiten nicht ausreichend vermeiden ließen, ist das unten bei der Erklärung der Resultate vermerkt.

A.3 Zeitverhalten der hardwarenahen Readoutsoftware

Die Digitalisierungshardware sollte nach einem Triggerimpuls in möglich kurzer Zeit ausgelesen werden, um das System für die Aufnahme des nächsten Events freizumachen. Durch Pufferung kann diese Ausleseoperation vom übrigen Datentransport zeitlich entkoppelt werden. Auch wenn der Durchsatz des gesamten Datenaufnahmesystems kleiner als der des Readouts ist, bringt eine kurze Readoutzeit den Vorteil einer höheren, wenn auch nur für begrenzte Zeit erreichbaren, Spitzenfrequenz — siehe auch Formel (A.4).

Die wesentlichen Bestandteile der im auslesenden Controller entstehenden Totzeit sind:

1. Konversionszeit der Hardware

2. Reaktionszeit (Latenz) der Readoutsoftware

3. Verwaltungsaufwand für den Readout (Eventzählung, Ermittlung und Aufruf der Readoutprozeduren)

4. Eigentliche Auslesezeit, d.h., Datentransport über den Readout-Bus in einen lokalen Puffer

Punkt 1, die Konversionszeit, ist nur von der Hardware selbst und deren Installation abhängig. Deshalb wird diese hier ausgeklammert, die Hardware wird als „unendlich schnell" betrachtet. Auch der Einfluß der vierten Komponente, des eigentlichen Auslesens, ist von der Hardware bestimmt (Anzahl der auszulesenden Worte × Zykluszeit des Auslesebusses) und im speziellen Einsatzfall leicht abzuschätzen.

Hier soll der Einfluß des konfigurierbaren Triggersystems und der listengesteuerten Readoutsoftware ermittelt werden und somit ein Maß für die maximal erreichbare Triggerrate gewonnen werden. Die Messungen wurden an einem CAMAC-Controller VCC2117 durchgeführt, in einer Konfiguration, wie sie in den COSY-Experimenten meist verwendet wird. (In Kapitel A.6 sind die Merkmale dieser Konfiguration aufgelistet.) Es hat sich gezeigt, daß einige Parameter des Systems in Hinblick auf die Verarbeitungsgeschwindigkeit nicht völlig optimal eingestellt sind. Verbesserungen in der Speicherverwaltung und im Cache-Management könnten zu besseren Resultaten führen. Diese Optimierungen werden in spätere Versionen der Betriebssystem-Implementierung einfließen. Ihr Einfluß auf die Meßresultate dürfte nur quantitativ sein, bezüglich der Speicherverwaltung wurde ihr Ausmaß abgeschätzt (siehe A.6).

Der Readoutcontroller arbeitet nach Erhalt eines Triggersignals für jedes aktive Instrumentierungssystem eine Readoutliste ab. Jede dieser Listen enthält eine Anzahl von Readoutprozeduren, die nacheinander aufgerufen werden. Unter der Annahme, daß alle Readoutprozeduren die gleiche Zeit beanspruchen, ist der Zeitbedarf für einen kompletten Auslesezyklus in der Variablen Listen-

anzahl (l) und Prozeduranzahl (p_i für Liste i) näherungsweise linear:

$$T = T_c + l \cdot T_l + \sum_{i=1}^{l} p_i \cdot T_p \tag{A.5}$$

Die Parameter wurden nach der üblichen Methode (Minimierung der Summe der Fehlerquadrate) an die gemessenen Auslesezeiten angepaßt:

$$\begin{aligned} T_c &= 18.0\mu s \\ T_l &= 10.5\mu s \\ T_p &= 5.0\mu s \end{aligned} \tag{A.6}$$

Die Genauigkeit liegt dabei im Bereich der letzten angegebenen Dezimalstelle, also bei ca. $0.1\mu s$.

Die Abbildungen A.1 und A.2 stellen die gemessenen Zykluszeiten den errechneten linearen Verläufen gegenüber.

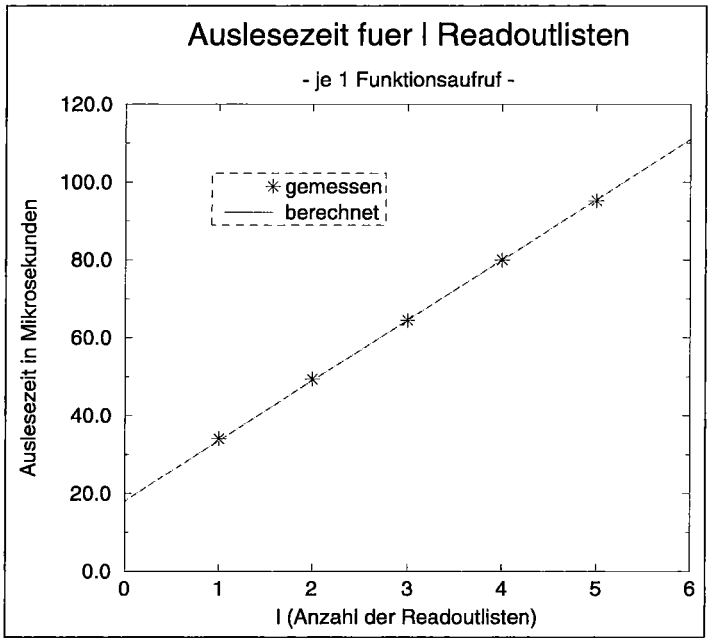


Abbildung A.1: Zykluszeit des Auslesens in Abhängigkeit von der Anzahl der Readoutlisten. Jede Readoutliste enthält 1 Funktionsaufruf. Dargestellt sind gemessene und mittels (A.5) und (A.6) berechnete Werte.

Die Diagramme zeigen, daß die experimentell erzielten Ergebnisse den linearen Ansatz (Gleichung A.5) rechtfertigen.

In den Experimenten werden häufig die GSI-Triggermodule zur Synchronisation der Frontend-Komponenten untereinander verwendet. Die zugehörige Triggerprozedur testet, ob der Triggerpuls vom Modul als gültig anerkannt wurde und vergleicht den Hardware-Eventzähler mit dem von der Readoutsoftware

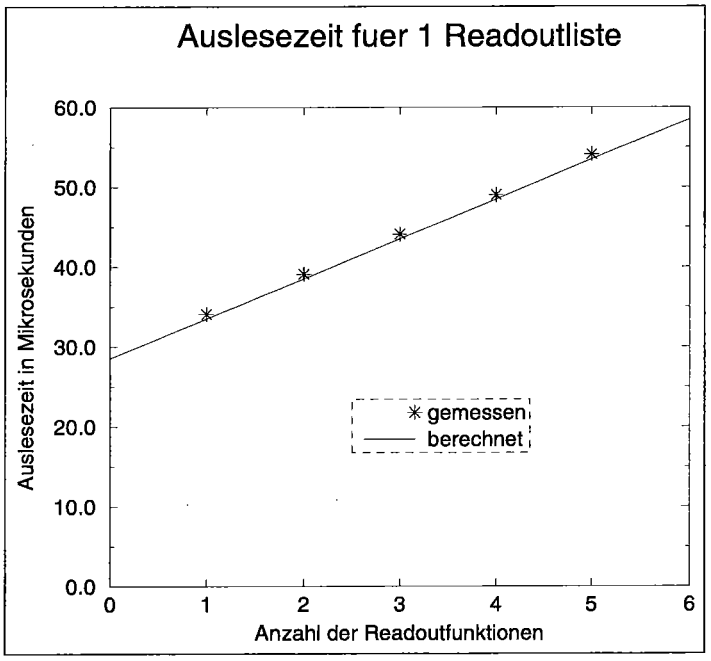


Abbildung A.2: Zykluszeit des Auslesens in Abhängigkeit von der Anzahl der Funktionsaufrufe in der Readoutliste. Dargestellt sind gemessene und mittels (A.5) und (A.6) berechnete Werte.

verwalteten. Die Messung mit dieser Zusatzbelastung ergab:

$$\begin{aligned} T_c &= 30.0\mu s \\ T_l &= 10.5\mu s \\ T_p &= 5.0\mu s \end{aligned}$$

Pro Event wird also eine konstante zusätzliche Zeit von $12\mu s$ für die genannten Operationen benötigt.

A.4 Durchsatz der verwendeten Übertragungsmedien

A.4.1 VIC-Bus

Der VIC-Bus wird häufig für die Ankopplung der Frontend-Crates an das Eventbuilder-Crate verwendet. Master auf dem VIC-Bus ist dabei ein Subeventreader, Slaves sind die Frontend-Controller (CAMAC, FASTBUS oder auch VME). Der VIC-Bus ist ein Kabelbus[3], die Länge kann im Bereich von einigen Zentimetern bis zu 100m variieren. Da die Datentransfers nach einem asynchronen Protokoll ablaufen, ist die Dauer eines Buszyklus und damit der maximale Durchsatz von der Kabellänge abhängig.

[3]im Gegensatz zu Rückwandbussen in Einschubrahmen

Die hier gezeigten Daten wurden mit einem VME-Prozessor FIC8234 als Master und einem CAMAC-Controller VCC2117 als Slave, verbunden durch ein ca. 5m langes VIC-Bus-Kabel, ermittelt.

Die mit dieser Anordnung gemessenen Durchsätze für die Datenübertragung vom Ausleseprogramm zum Subeventreader sind in Bild A.3 dargestellt.

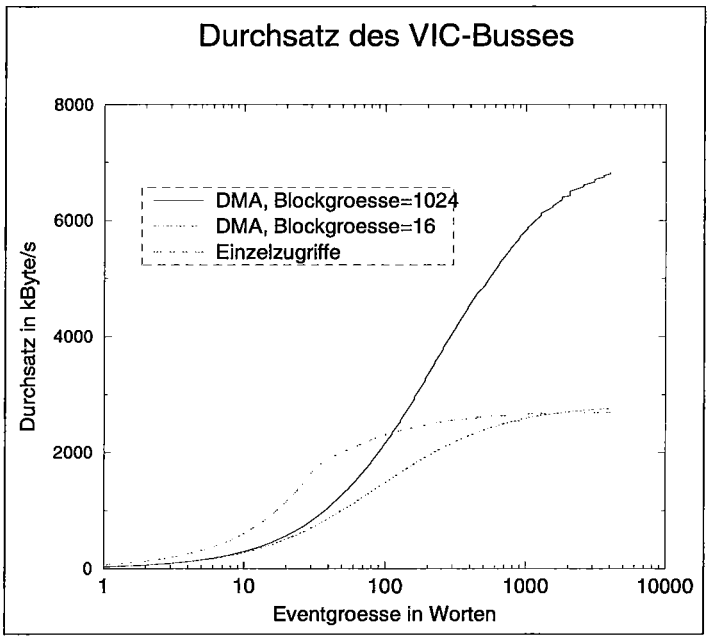


Abbildung A.3: Durchsatz des VIC-Busses in verschiedenen Übertragungsmodi.

Wie nicht anders zu erwarten, werden die besten Resultate bei maximalen Blockgrößen erzielt. Leider erzwingt die hier verwendete Hardware, in Konfigurationen, die einen Subeventreader verwenden, auf lange Blocktransfers zu verzichten. Das Problem liegt darin, daß der Subeventreader während eines solchen VIC-Transfers „blockiert“ ist — auch der VME-Port. Zugriffe des Eventbuilders werden, wenn sie zu einem ungünstigen Zeitpunkt stattfinden und der VIC-Transfer zu lang ist, mit Timeout abgebrochen. Die Folge ist, daß der praktisch erzielbare Durchsatz des VIC-Busses höher ist, wenn kein Subeventreader verwendet wird. Hier liegt auch die Ursache, daß in den Abschnitt A.7 beschriebenen Vergleichsmessungen die simplere Konfiguration ohne Subeventreader in gewissen Bereichen (nämlich denen, wo der VIC-Bus-Durchsatz begrenzend wirkt) Vorteile für sich verbuchen kann.

Die in Abbildung A.3 gezeigten Kurven lassen mit sehr kleinem Fehler an eine Funktion nach Ansatz (A.2) anpassen. Lediglich die Kurve für Einzelzugriffe zeigt eine auffällige Abweichung. Abbildung A.4 zeigt die zugehörigen Eventfrequenzen.

Für die Auslesefrequenz existiert eine obere Grenze, die vom CAMAC-Controller vorgegeben wird. In Abschnitt A.3 wurde die Geschwindigkeit des Controllers untersucht. Hier kommt noch die Busbelastung durch den Subevent-

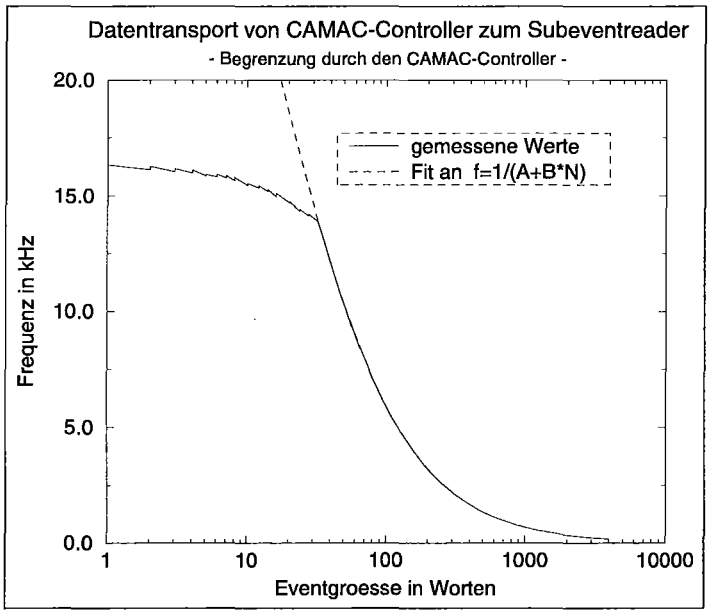


Abbildung A.4: Die maximale Auslesefrequenz des CAMAC-Controllers begrenzt die Frequenz auf dem VIC-Bus.

reader zum Tragen, ihr Einfluß wird neben anderen in Kapitel A.6 gezeigt.

Für die hier vorhandene Konstellation (1 Readoutliste mit 1 Funktion, Pollen durch den Subeventreader) wurde die Grenzfrequenz des CAMAC-Controllers in den erwähnten Messungen zu 17.9 kHz ermittelt. Ein Vergleich mit Abb. A.4 zeigt, daß die Obergrenze der gemessenen Übertragungsfrequenzen etwa diesem Wert entspricht.

A.4.2 VME-Bus

Der VME-Bus dient in erster Linie der Datenübertragung innerhalb des Eventbuildercrates, also von den Subeventreadern zum Eventbuilder. Theoretisch liegt seine Bandbreite bei $80 Mbyte/s$. Mit nur einem Busmaster ist es aber praktisch kaum möglich, diese Grenze auch nur näherungsweise zu erreichen. Sowohl die masterseitigen als auch die Slaveinterfaces der meisten VME-Prozessormodule bleiben in ihrer Leistungsfähigkeit weit hinter dem Möglichen zurück. Die in den Eventbuildern verwendeten Prozessoren (Motorola 68040) sind außerdem nicht in der Lage, mit Einzeltransfers signifikante Teile der Busbandbreite auszunutzen. Der verwendete Eventbuilder (FIC8234) besitzt eine DMA-Logik, die VME-Blocktransfers ohne Eingriff des Prozessors durchführt.

Abbildung A.5 zeigt, welche Durchsätze von der Eventbuilderhard- und -software auf dem VME-Bus erzielt wurden.

Als Subeventreader wurde ebenfalls ein FIC8234 eingesetzt. Dieser erlaubt, auch den besonders effektiven D64-Modus (64 Bit Daten Parallel) zu nutzen. Gemessen wurden Einzelzyklen („Single D32“) und DMA-Blocktransfers („BLK D32“ bzw. „BLK D64“).

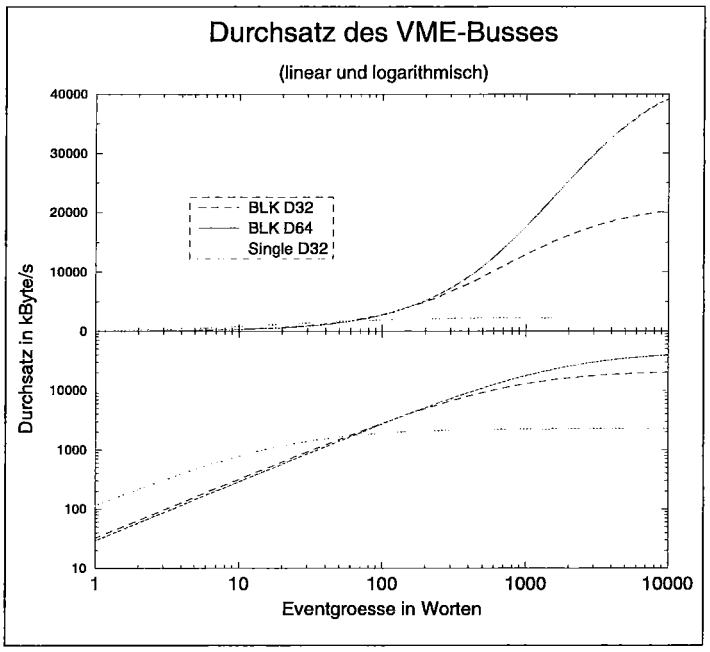


Abbildung A.5: Durchsatz des VME-Busses in verschiedenen Übertragungsmodi. Die gleichen Daten sind zur besseren Vergleichbarkeit einmal mit linearer und einmal mit logarithmischer Ordinatenachse dargestellt.

Der Vergleich mit den vom Hersteller der VME-Prozessorboards genannten Daten [83] zeigt, daß die für die Boards als Maxima angegebenen Werte zu ca. 80% erreicht werden. Dieses Resultat ist befriedigend, vor allem in Anbetracht der komplizierteren Alignmentverhältnisse[4] im realen Datenaufnahmesystem. Die gleiche Quelle stellt auch eine Erhöhung des Durchsatzes um 50–100% bei Verwendung des Nachfolgetyps FIC8243 in Aussicht.

Eine Eigenheit der Implementierung hardwareabhängiger Routinen in unserem Datenaufnahmesystem wirkt sich ungünstig auf die Verarbeitungsgeschwindigkeit aus: Um das eigentliche Eventbuilderprogramm hardwareunabhängig zu halten, und um einen reibungslosen Mehrbenutzerbetrieb zu gewährleisten, finden die Zugriffe auf DMA-Logik u.ä. in Treibern statt, die Bestandteil des Betriebssystems sind. Das Lesen von Daten über diese Schnittstelle erfordert damit einen Systemaufruf, dieser bringt einen zusätzlichen Zeitbedarf von etwa $100\mu s$ pro zu lesenden Block mit sich. In Anbetracht der mittelfristig geplanten (und teilweise erfolgten) Portierung des Datenaufnahmesystems auf UNIX-Plattformen wurde darauf verzichtet, die Treiberschnittstelle (unter Verwendung nichtstandardgemäßer Methoden) zu umgehen. Der Nutzen solcher, vom Standpunkt der Softwaretechnologie, unschönen Maßnahmen wäre ohnehin begrenzt, solange das Datenspeichergerät (EXABYTE) den Datendurchsatz auf einen Bruchteil der vom VME-Bus gegebenen Möglichkeiten beschränkt.

[4] Ausrichtung der Datenblöcke bezüglich eines von der Hardware vorgegebenen Rasters

A.4.3 Ethernet

Für die Online-Analyse der ausgelesenen Daten, aber auch für die Speicherung bei geringem Datenanfall, können Events über einen TCP-Socket vom Eventbuilder zu einer Workstation übertragen werden. Die mit der Paarung FIC8234 (Eventbuilder) - DEC 5000/240 (Workstation) erzielten Durchsätze sind Abb. A.6 zu entnehmen.

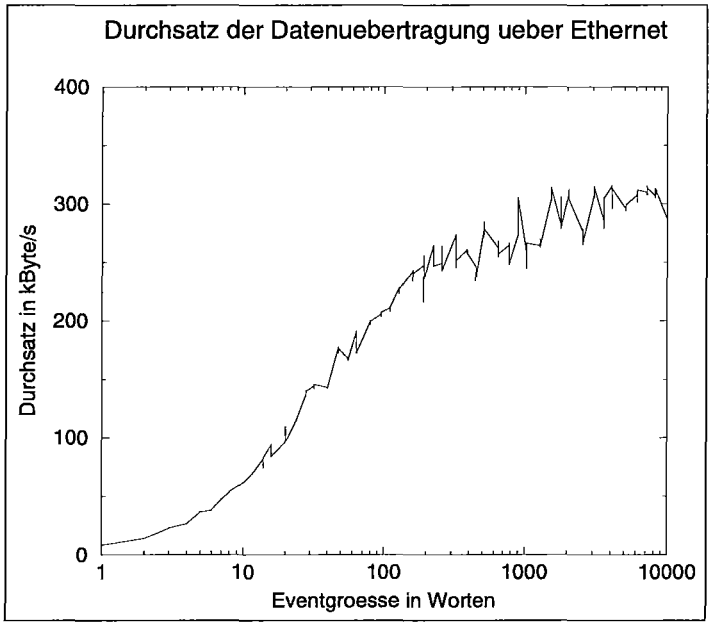


Abbildung A.6: Durchsatz der Datenübertragung über Ethernet (TCP/IP)

Das Übertragungsmedium (Ethernet — 10 MBit/s) läßt theoretisch einen Durchsatz von bis ca. 1 MByte/s zu. Workstations sind oft in der Lage, dieses Limit nahezu auszureizen, die Netzwerkimplementierung unter OS-9 bleibt hier weit zurück.

Auch die hier gezeigte Messung wurde unter den in Abschnitt A.2 erläuterten „idealen" Bedingungen durchgeführt. Das heißt im einzelnen:

- Der Eventbuider liest keine Daten von Subeventreadern, sondern erzeugt diese selbst.
- Die Workstation liest die eingehenden Daten so schnell wie möglich und verarbeitet sie nicht weiter.

Die Online-Analyse der Daten, zusätzliche Berechnungen (z.B. Spurrekonstruktion) und die graphische Darstellung der Resultate können der Workstation erheblichen Rechenaufwand abverlangen. In der Praxis ist auch eine relativ leistungsfähige Workstation nicht in der Lage, die hier gemessenen Übertragungsraten auszunutzen. So konnten z.B. mit einer DEC 5000/240 ca. 20–50 Events/s analysiert und dargestellt werden, während theroretisch bis zu 500 Events/s (eine Größe von 100 Worten angenommen) geliefert werden könnten.

A.5 Eventbuilding und Pufferverwaltung

Die Puffer, die die verschiedenen Abschnitte der Datenaufnahme voneinander entkoppeln, sind ringförmig organisiert. Ihre Arbeitsweise ist in Kapitel 3.3.2 erläutert.

Ein solcher Puffer kann die Daten von

$$n = \text{ceil}\left(\frac{V - N_{max}}{N + 7}\right) \tag{A.7}$$

mit

V	...	Volumen (Gesamtlänge) des Puffers
N_{max}	...	maximale Eventgröße
N	...	tatsächliche Eventgröße
$\text{ceil}(x)$	...	kleinstes $a \in \mathbb{G}$ mit $a \geq x$

Events aufnehmen[5].

Während der Puffer mit Eventdaten gefüllt wird, muß bei einem kontinuierlichen Betrieb die gleiche Menge an Speicherplatz vom schreibenden Prozeß als freigegeben erkannt und dem Puffer wieder zur Verfügung gestellt werden. Unterbrochen wird dieser Ablauf, wenn der Puffer bis zum Ende gefüllt ist. Dann beginnt das Füllen wieder am Pufferanfang, und es muß Raum für das größte denkbare Event geschaffen werden. Das heißt, es muß eine Anzahl von Events, die beim vorherigen Durchlauf geschrieben worden waren, als gelesen erkannt werden. Diese Zahl bestimmt sich wie folgt:

$$n_{max} = \text{ceil}\left(\frac{N_{max}}{N + 7}\right) \tag{A.8}$$

Da an diesem Punkt die „alten" Events einzeln überprüft werden müssen, um ihre jeweiligen Größen zur Menge des derzeit freien Speichers addieren zu können, ist dieser Vorgang relativ aufwendig. Je kleiner die tatsächlichen Events im Verhältnis zur maximalen Eventgröße N_{max} sind, um so größer ist die beim „Überschlag" des Puffers auftretende Totzeit.

Für einen CAMAC-Controller VCC2117 wurde diese Totzeit näher untersucht. Tabelle A.1 zeigt die gemessenen Daten zusammen mit den jeweiligen Resultaten der Formeln (A.7) und (A.8).

T_{wrap} ist die für das Prüfen der n_{max} Events nötige Zeit, *Periode* der zeitliche Abstand zwischen 2 solchen Ereignissen. Die Daten wurden mit einem Oszilloskop unter Zuhilfenahme der „Profile"-Option (siehe A.6) ermittelt, die beobachteten Eventfrequenzen (entspricht $\frac{n}{Periode}$ in Tabelle A.1) sind also kleiner als die Werte aus Kapitel A.3. Die Puffergröße V ist hier, wie im Standard-Setup der CAMAC-Controller in den Experimenten, 512 kByte; die maximale Eventgröße N_{max} ist 4096 Worte, also 16 kByte.

[5]Die magische Zahl 7 ist die minimale Anzahl der Worte, die pro Event mit Verwaltungsdaten belegt werden. Sie setzt sich zusammen aus Subeventheader (2 Worte), Eventheader (4 Worte) — siehe Erklärung des Datenformats in 3.3.3 — und dem Overhead der Ringpufferverwaltung (Flag und Länge, siehe Abschnitt 3.3.2, das Längenfeld ist identisch zu dem im Eventheader und deshalb nur einmal vorhanden).

Eventgröße	n	n_{max}	Dauer $T_{wrap}/\mu s$	$Periode/ms$
0	18140	586	2570	1840
10	7470	241	1060	757
20	4703	152	674	476
1000	127	5	33	12.9
2000	64	3	25	6.5
4000	32	2	20	3.25

Tabelle A.1: Dauer und Häufigkeit der verlängerten Totzeit nach vollständiger Füllung des Datenpuffers

Auch wenn T_{wrap} teilweise recht erheblich ist ($2570\mu s$ entsprechen 25 „verpaßten" Events bei einer Auslesefrequenz von 10 kHz), ist doch der Einfluß auf die mittlere Totzeit recht gering: $\frac{T_{wrap}}{n}$ ist in jedem Fall deutlich kleiner als $1\mu s$. Gegenüber der totalen Auslesezeit von $\frac{Periode}{n} \approx 100\mu s$ ist dies vernachlässigbar.

T_{wrap} ist linear von der Zahl der zu prüfenden „alten" Events abhängig, ein Ansatz

$$T_{wrap} = T_c + n_{max} \cdot T_n \tag{A.9}$$

liefert für die Daten aus Tab. A.1 nach einer einfachen Anpassungsrechnung mit sehr kleinem Fehler ($< 1\%$) die Parameter:

$$\begin{aligned} T_c &= 10.8\mu s \\ T_n &= 4.37\mu s \end{aligned}$$

Ebenfalls mit dem Oszilloskop wurde ermittelt, daß im Normalfall, während des Füllens des Ringpuffers, ca. $16\mu s$ für das Finden des nächsten freien Speicherplatzes benötigt werden. In diesem Normalfall muß genau 1 Speicherblock getestet werden, um Platz für 1 Event zu gewinnen. Es ist zu erwarten, das dafür die Zeit $T_c + 1 \cdot T_n$ benötigt wird. Die Feststellung, daß die beiden Zahlen nahezu übereinstimmen, rundet das erfreulich vollständige Bild vom zeitlichen Verhalten der Pufferverwaltung ab.

Wie schon eingangs (auf Seite 98) erwähnt, kann sich die Verteilung der Zeitscheiben in Multitasking-Betriebssystemen auf charakteristische Weise im Zeitverhalten und im Durchsatz der gepufferten Datenströme niederschlagen. Diese Effekte werden makroskopisch, wenn die zum Füllen eines Puffers benötigte Zeit und die Dauer einer Zeitscheibe in vergleichbare Größenordnungen rücken. Im Normalfall, bei ausreichenden Puffergrößen, werden viele Zeitscheiben benötigt, um die Menge von Daten zu transportieren, die die Puffer fassen können. Mit sinkender Puffergröße oder wachsender Zeitscheibenlänge bewegen sich die Zahlen aufeinander zu. Bild A.7 zeigt die Wirkung auf den Durchsatz einer Übertragungsstrecke, wenn weniger als eine Größenordnung zwischen den beiden Zeiten liegt.

Es fallen 2 deutlich getrennte Bereiche mit einem charakteristischen Aussehen auf:

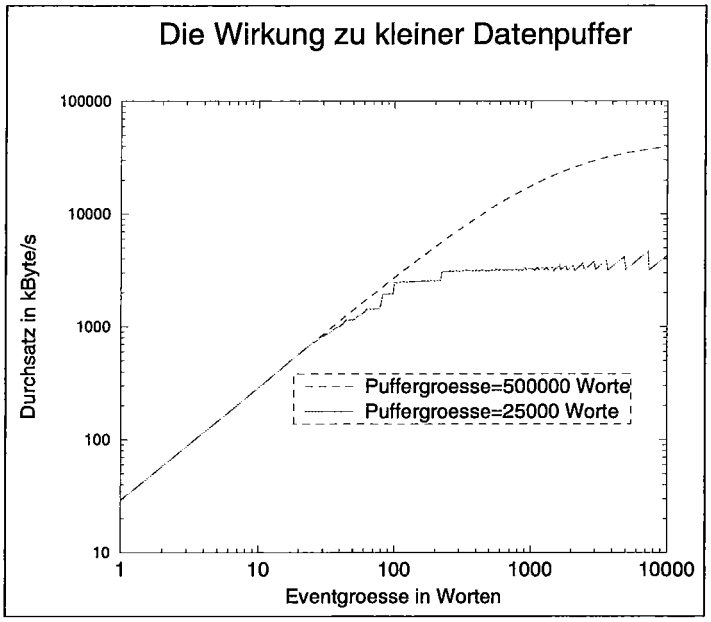


Abbildung A.7: Durchsatz bei zu kleiner Puffergröße im Vergleich zum „Idealwert".

- Ein Gebiet bei einer Eventgröße von ca. 100 Worten, in dem der Durchsatz stückweise konstant bleibt. In Abbildung A.8 ist dieses etwas detaillierter dargestellt.

  Hier beginnt sich die diskrete Zuteilung von Zeitscheiben an die Daten lesenden und schreibenden Prozesse bemerkbar zu machen. Der schreibende Prozeß benötigt kein genaues Vielfaches der Zeitscheibenlänge, um seinen Puffer zu füllen. Die letzte Zeitscheibe wird nicht vollständig ausgenutzt, sondern teilweise mit Warten auf das Freiwerden von Pufferspeicher verbracht. Am Ende dieses Bereichs ist der Puffer im Bruchteil einer Zeitscheibe gefüllt, und die zugeteilte Rechenzeit wird zunehmend durch (sinnloses) Pollen verschwendet, bis der lesende Prozeß aktiv wird und den Puffer wieder leert.

  Noch ist die Eventgröße N klein gegen die nutzbare Puffergröße $V - N_{max}$, der Puffer wird (lt. Gleichung (A.7)) recht gleichmäßig ausgenutzt. So kommt es dazu, daß die näherungsweise konstante Pufferfüllung effektiv in einem Vielfachen der Zeitscheibendauer transportiert wird.

  Man könnte einwenden, daß der schreibende Prozeß, statt bei vollem Puffer ohne Aussicht auf Erfolg zu pollen, den Rest der Zeitscheibe abtreten könnte. (Das ist in den verwendeten Multitasking-Systemen in der Tat vorgesehen.) Das stößt jedoch auf technische Probleme, da zum einen die Eventbuildersoftware dafür ausgelegt ist, mehrere Ausgabekanäle zu bedienen, und das Beenden der Zeitscheibe nur dann sinnvoll ist, wenn alle zugehörigen Puffer gefüllt sind. Zum anderen haben die Pufferverwaltungsroutinen auch keine Information darüber, ob der jeweilige Lese-

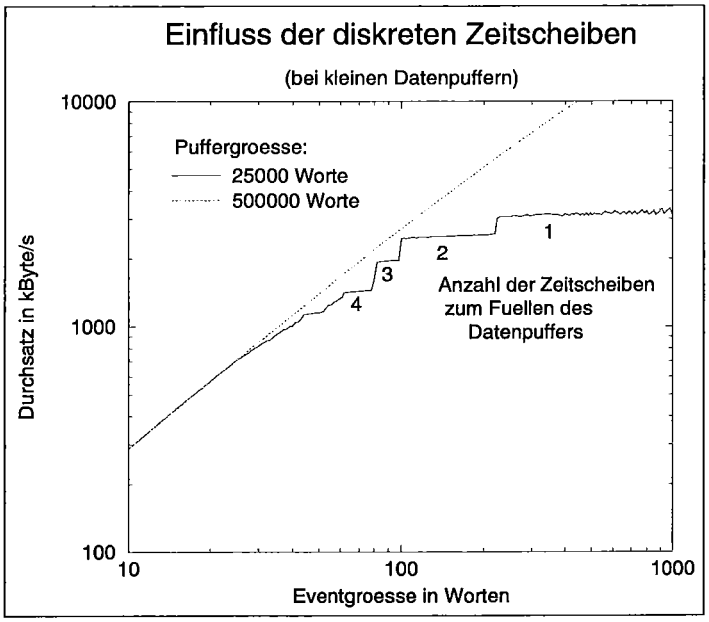


Abbildung A.8: Ausschnitt aus Abb. A.7, zeigt, wie der Durchsatz infolge unvollständig ausgenutzter Zeitscheiben nicht mit der Eventgröße wächst.

prozeß auf dem gleichen Prozessor läuft oder, völlig asynchron, von einem anderen Rechner aus über VIC- oder VME-Bus auf die Daten zugreift.

- Bei größeren Events, jenseits der 1000 Worte, ist ein regelmäßiges Sägezahnmuster in der Durchsatzkurve zu vermerken. Abb. A.9 zeigt diesen Bereich in linearer Darstellung.

  Der Durchsatz ist hier stückweise proportional zur Eventgröße. Da, wie oben festgestellt wurde, für den Transport einer Pufferfüllung vom füllenden Prozeß eine (ganze) Zeitscheibe verwendet wird, muß die Anzahl der in der Zeitscheibe bearbeiteten Events in den Geradenstücken konstant sein. In Bild A.10 ist zu sehen, daß die Auslesefrequenz in gleichmäßigen Stufen bei wachsender Eventgröße fällt.

  Ursache ist, daß nur noch eine relativ kleine Zahl von Events in den Ringpuffer paßt. Bei diesen Messungen war:

$$\begin{aligned} V &= 25000\ Worte \text{ und} \\ N_{max} &= 10240\ Worte \end{aligned}$$

  Die Zahl der Events im Puffer ist mit Gleichung (A.7) berechenbar, die Sprünge in den gemessenen Kurven befinden sich an den Positionen

$$N_n = \frac{V - N_{max}}{n} - 7 \quad \text{mit } n = 2, 3, \ldots \tag{A.10}$$

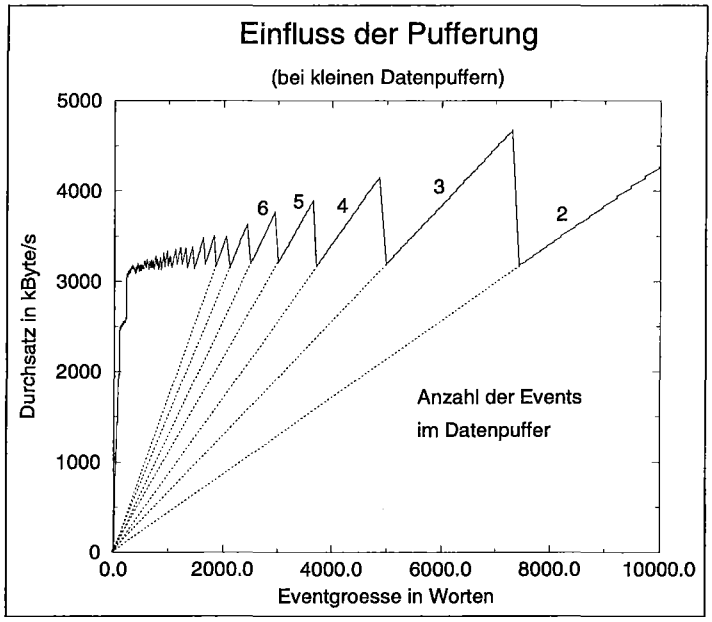


Abbildung A.9: Ausschnitt aus Abb. A.7, zeigt die zunehmend ineffektive Pufferausnutzung bei großen Events.

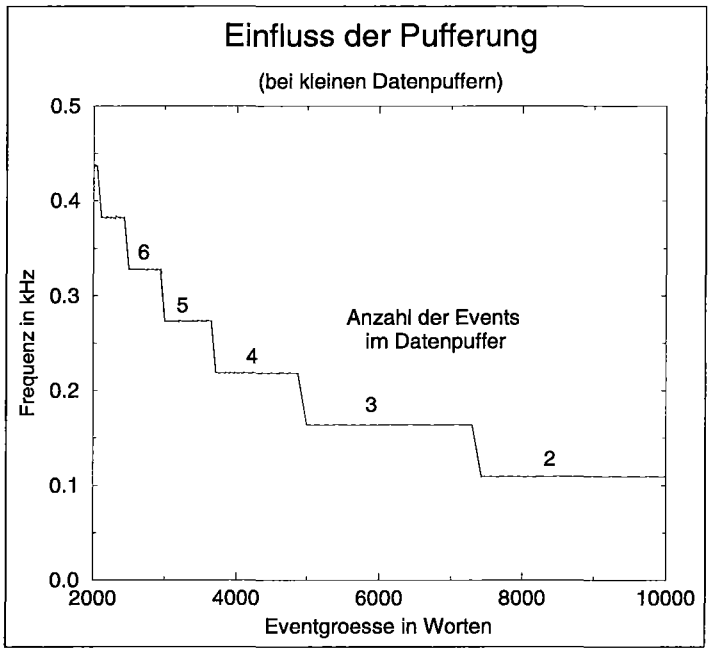


Abbildung A.10: Andere Darstellung von Abb. A.9. Sie zeigt, wie jeweils nur eine bestimmte Zahl von Events in einer Zeitscheibe transportiert werden kann.

A.6 Einfluß verschiedener Randbedingungen

Neben der Hardware selbst und den direkt vom Anwender steuerbaren Parametern (z.B. Lastverteilung im System) beeinflussen noch einige eher in den

Bereich der Systeminstallation gehörige Faktoren die Geschwindigkeit der Datenaufnahme.

In Bild A.11 ist dargestellt, wie stark sich die wichtigsten von diesen auf die Auslesegeschwindigkeit eines CAMAC-Controllers VCC2117 auswirken. Alle Werte beziehen sich auf eine Standardkonfiguration, wie sie der in den Experimenten derzeit verwendeten entspricht. Deren Merkmale sind:

- Die beiden Speicherbereiche des VCC2117 (lokaler Speicher und Dual-Port-RAM) werden beide vom Betriebssystem verwendet. Der Speicherzuweisungsalgorithmus von OS-9 bewirkt dabei, daß sich der Code des Ausleseprogramms im Dual-Port-RAM befindet und sich somit die Programmabarbeitung und die externen Zugriffe durch den Subeventreader gegenseitig behindern. (Das Problem ist lösbar, siehe auch die Bemerkungen auf Seite 100.)

- In das Ausleseprogramm sind die optionalen Zusatzfunktionen „Erzeugen von Debug-Ausschriften“ und „Profiling“ nicht eingebaut.

- Einige zeitkritische Programmteile sind in Assembler geschrieben. (Es existiert auch eine portable C-Implementierung.)

- Der CAMAC-Controller wird nicht dadurch belastet, daß ein Subeventreader versucht, Daten aus dem Speicher (dem Dual-Port-RAM) zu lesen. Diese Annahme ist ein wenig problematisch, da in einem realen System meist Daten übertragen werden sollen. Daß sie dennoch getroffen wurde, hat 2 Gründe:

  - Wenn sich der Code des Ausleseprogramms im lokalen RAM des Controllers befindet (was der später anzustrebende Zustand ist), ist der Einfluß dieser Busbelastung, wie sich gezeigt hat, recht klein.
  - Anderenfalls ist die dadurch verursachte Verlangsamung leicht abzuschätzen, indem der hier ermittelte Faktor angewendet wird.

Die in Bild A.11 untersuchten Veränderungen gegenüber der Referenzkonfiguration sind im einzelnen

local: Der Programmcode befindet sich im lokalen RAM des Controllers. Das läßt sich durch manuelle Einwirkung auf die Speicherverwaltung erzwingen, diese Methode kann aber nur schwer auf Controller im „harten“ Meßeinsatz übertragen werden. Die sichtbare Geschwindigkeitserhöhung resultiert nur aus der kleineren Zugriffszeit dieses Speichers. In der Praxis käme noch der Gewinn aus dem nicht stattfindenden Pollen (siehe unten) dazu.

poll: Der CAMAC-Controller wird dadurch belastet, daß ein Subeventreader in kurzen Abständen auf den Dual-Port-RAM zugreift. Diese Verlangsamung wäre bei nur ca. 10%, wenn das Ausleseprogramm im lokalen RAM laufen würde. Dies kompensiert etwa den oben festgestellten Geschwindigkeitsunterschied zwischen lokalem Speicher und Dual-Port-RAM, so daß die Auslese in lokalem RAM unter Pollbelastung annähernd

mit der Geschwindigkeit der hier untersuchten Standardkonfiguration — ohne Pollbelastung — läuft.

debug: Das Ausleseprogramm enthält Programmcode, der aktiviert werden kann, um Statusausschriften und Informationen über den Programmablauf (Tracing) auszugeben.

noopt: Die zur Zeitoptimierung in Assembler implementierten Routinen werden nicht verwendet, statt dessen kommen portable, in „C" verfaßte, Programmteile mit gleicher Funktionalität zum Einsatz.

profile: Das Ausleseprogramm enthält Code, der Informationen über den Programmstatus über ein Hardwareregister ausgeben kann und so der Untersuchung mittels Oszilloskop o.ä. zugänglich macht. Diese Option erzwingt gleichzeitig den Verzicht auf die optimierten Assembler-Routinen, die rein hierdurch bedingte Geschwindigkeitseinbuße liegt also bei ca. 8%.

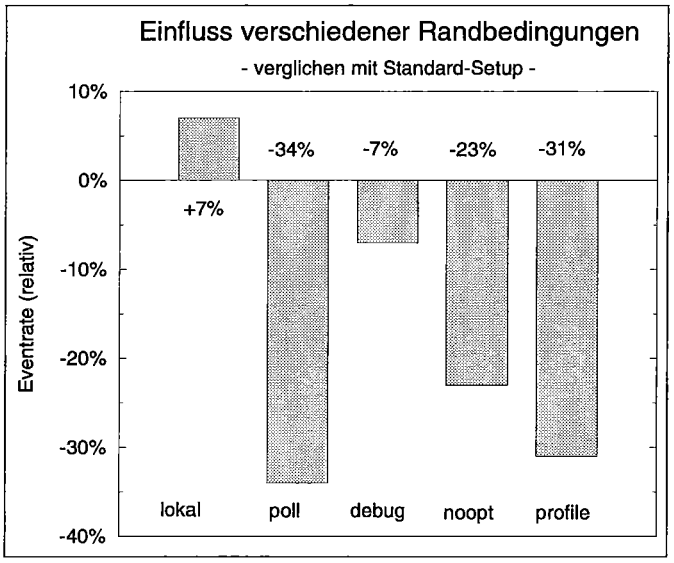


Abbildung A.11: Wirkung verschiedener Einflüsse auf die Verarbeitungsgeschwindigkeit eines CAMAC-Controllers VCC2117. Erklärung der Abkürzungen im Text.

Die hier analysierten Einflüsse wirken sich nicht unbedingt gleichmäßig auf alle Teile des Ausleseprogramms aus, ihre Auswirkung ist also auch davon abhängig, wie häufig bestimmte Programmfunktionen ausgeführt werden, also won der konkreten Konfiguration und der Belastung des Systems. Die in Abb. A.11 gezeigten Daten wurden ermittelt, indem jeweils 5 verschiedene Varianten untersucht wurden, die Endresultate sind über die Einzelergebnisse gemittelt.

A.7 Beispiele kompletter Datenaufnahmesysteme

Als Anwendungsbeispiel für ein Datenaufnahmesystem, wie es auch bisher in den physikalischen Experimenten benutzt wurde, wurde das Auslesen von 2 CAMAC-Crates gewählt. Diese Aufgabe läßt sich unter Verwendung der in den vorangegangenen Kapiteln untersuchten Bestandteile auf verschiedene Weisen lösen. 3 sinnvolle Konfigurationen aus Frontend-Crates, Subeventreadern (SER) und Eventbuildern (EB) sind in Abb. A.12 skizziert.

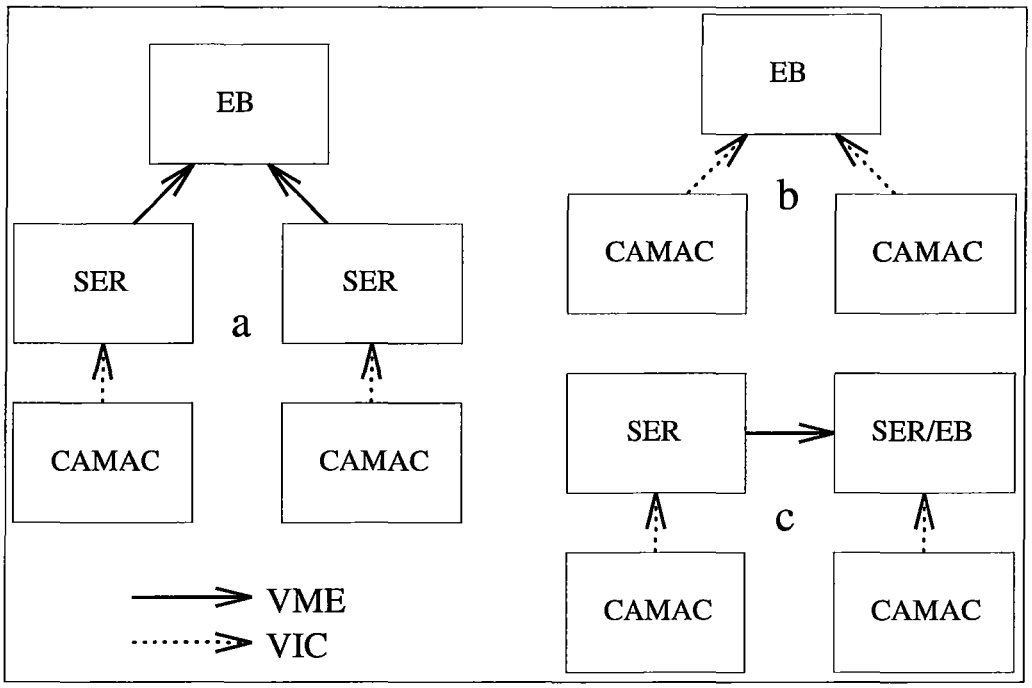


Abbildung A.12: Schemata der 3 untersuchten Konfigurationen für das Auslesen von 2 CAMAC-Crates. (a): mit 2 SER und EB, (b): ohne SER, (c): 2 SER, kein dedizierter EB

Variante (a) verspricht zunächst den größten Gesamtdurchsatz, da durch Verwendung der Subeventreader mit eigenen Datenpuffern der Durchsatz beider VIC-Busse voll ausgenutzt werden kann (siehe auch Seite 97). Folgen sollten die Varianten (c) und (b).

Die in Kapitel A.4.1 beschriebenen Probleme zwingen leider dazu, in den von Subeventreadern kontrollierten VIC-Bus-Zweigen (beide in Variante (a), der linke in Variante (c)) auf effektive Blocktransfers zu verzichten. Da, wie in Bild A.3 zu sehen, kleine Blocktransfers keinen Nutzen gegenüber Einzeltransfers bringen, wurden an diesen Stellen die Einzeltransfers verwendet. Die bedauerliche Folge ist, daß der Maximaldurchsatz des VIC-Busses dadurch auf weniger als die Hälfte sinkt und der Vorteil der vollen Busauslastung erst bei mehr als 2 parallel verwendeten VIC-Bus-Strängen zum Tragen kommt.

Es war beim Aufbau nach Variante (a) auch nicht möglich, den D64-Modus für die VME-Transfers von den Subeventreadern zum Eventbuilder zu verwen-

den. Grund hierfür war ein (reparabler) Fehler in der VME-Interfacelogik eines der Subeventreader.

Die in Abb. A.13 gezeigten Durchsatzdaten sind also unter diesem Vorbehalt zu interpretieren.

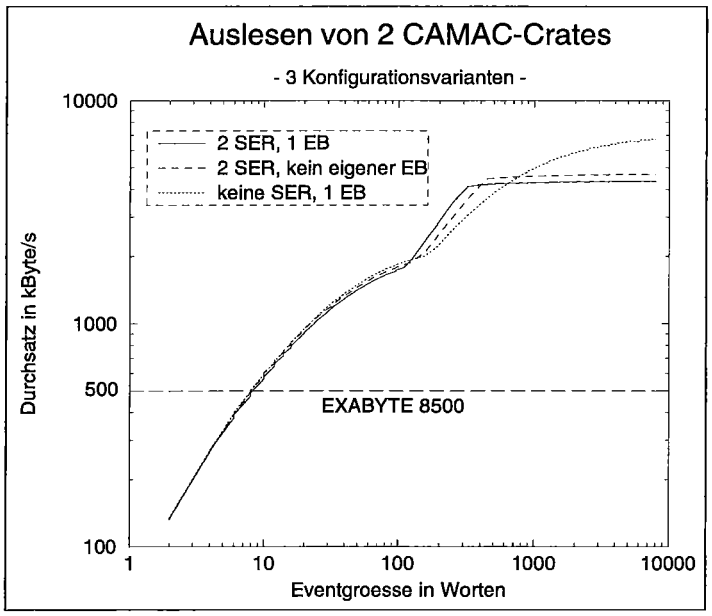


Abbildung A.13: Erreichbare Systemdurchsätze mit 3 verschiedenen Konfigurationen. Der Maximaldurchsatz entspricht dem von 1 bzw. 2 VIC-Bus-Verbindungen (siehe Abb. A.3). Beschreibung der Konfigurationen im Text.

Den Abbildungen A.3 bzw. A.5 ist zu entnehmen, daß es aufgrund des hohen Verwaltungsaufwands für DMA-Blocktransfers für kleine Events angebracht ist, die Daten mittels Einzeltransfers zu übertragen. Die im Bild gezeigten Durchsatzdaten sind mit der jeweils optimalen Methode erreicht worden. Im Bereich der kleinen Eventlängen ($N \lessapprox 100$) werden durchgängig Einzelworte übertragen.

Bild A.14 zeigt, daß die maximale Auslesefrequenz bei kleinen Events unabhängig von der Konfigurationsvariante ist.

Ein Vergleich mit Abb. A.4 zeigt, daß auch hier die Grenze vom CAMAC-Controller vorgegeben wird.

Der Einfluß der Lastverteilung In Variante (c) der in Bild A.12 gezeigten Konfigurationen haben die beiden VIC-Bus-Zweige wegen der oben erwähnten Beschränkungen verschiedene nutzbare Bandbreiten. In Folge der im Eventbuilder stattfindenden Synchronisation (siehe die Bemerkungen zur Parallelisierung in Abschnitt A.1) bestimmt der Zweig, der die Eventdaten mit der kleineren Frequenz liefert, die Frequenz der Gesamtanordnung. Wenn die Teileventgrößen gleich sind, wird dies der VIC-Bus mit dem kleineren Durchsatz sein, der andere

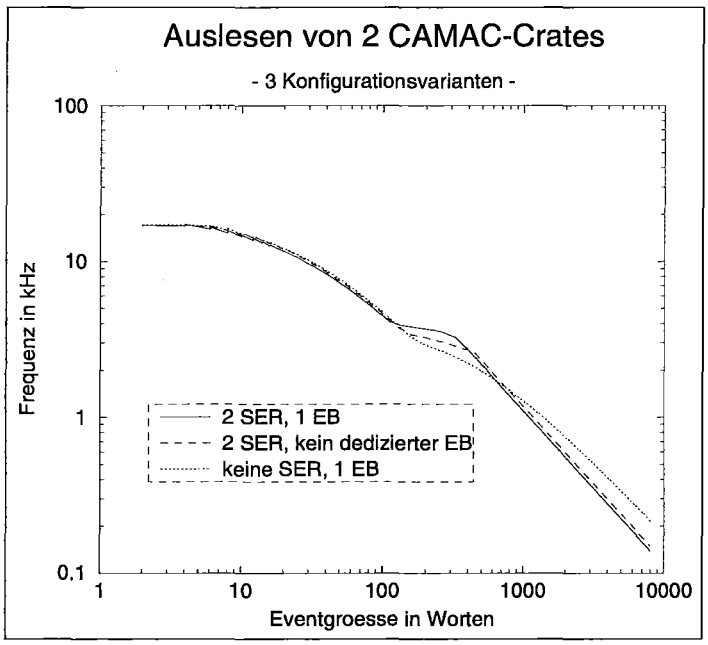


Abbildung A.14: Erreichbare Auslesefrequenzen bei 3 verschiedenen Systemkonfigurationen. Die Maximalfrequenz wird vom CAMAC-Controller bestimmt. Beschreibung der Konfigurationen im Text.

wird nur zu einem Teil ausgenutzt.

Sofern es der Experimentaufbau erlaubt, sorgt hier eine der Hardware angemessene Lastverteilung (d.h., die Aufteilung der Gesamteventgröße auf die VIC-Bus-Zweige) für eine bessere Ausnutzung des langsameren Zweigs und somit für eine höhere Gesamtperformance. Bild A.15 zeigt dies für die obige Konfiguration (c).

A.8 Zusammenfassung

Es ist schwierig, das dynamische Verhalten eines Datenaufnahmesystems zu erfassen und zu modellieren. Das liegt besonders an den vielen verschiedenen möglichen Konfigurationsdetails und Randbedingungen. So gibt es zum Beispiel beliebig viele Verteilungen der Triggerimpulse über die Zeit, Häufigkeitsverteilungen der Eventdatenlänge und Korrelationen aus beiden.

Um Aussagen zu gewinnen, die von allgemeineren Interesse sind, und die für Vergleiche herangezogen werden können, mußten einige vereinfachende Annahmen gemacht werden: Es wurde der maximale Dauerdurchsatz für jeweils konstante Datenblocklängen ermittelt. Da dies aber nicht nur für komplette Systeme, sondern auch für einzelne Komponenten getan wurde, ist auch feststellbar, welche Teile jeweils leistungsbegrenzend sind, und weiter, für welche Zeit bzw. wieviel Events eine solche Hemmung durch einen zwischengeschalteten Puffer überbrückt werden kann.

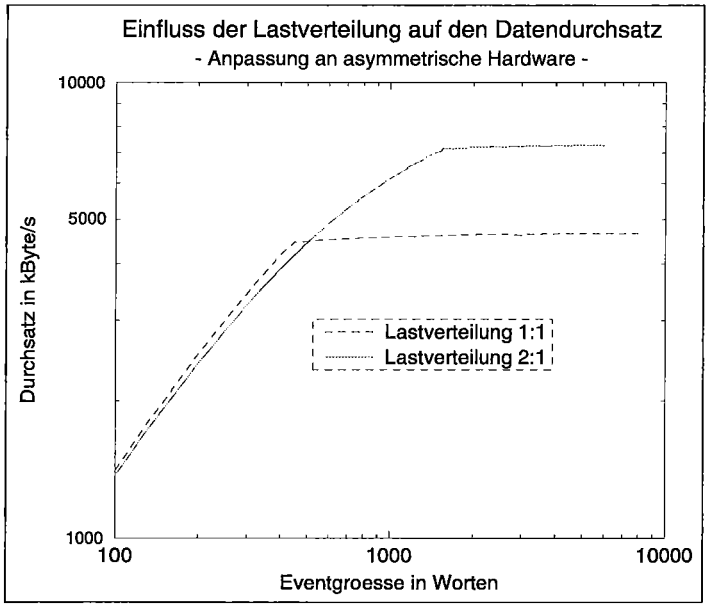


Abbildung A.15: Erhöhung des Systemdurchsatzes bei Anpassung der Last an die Hardwarebedingungen.

Die Messungen zeigten, daß bei realen Eventgrößen im Dauerbetrieb das Speichermedium (EXABYTE) den Durchsatz begrenzt. Die Maximalfrequenz ist durch den Readoutcontroller selbst limitiert. Es ergaben sich auch Hinweise auf für uns ungünstige Eigenschaften der Hardware (Speicher im CAMAC-Controller, Blocktransfers in den VME-CPUs). Soweit möglich, werden diese in zukünftigen Versionen beseitigt. Die Listensteuerung des Readoutvorganges fordert ebenfalls einen gewissen Tribut, der Aufruf einer Ausleseprozedur in einem separaten Instrumentierungssysten kostet ca. $15\mu s$. Eine ausprogrammierte Implementierung würde für eine vergleichbare Operation weniger als $5\mu s$ benötigen. In manchen Fällen mögen diese $10\mu s$ wertvoll sein, meistens sind sie jedoch nicht erheblich: In dieser Größenordnung liegen übliche Konversionszeiten von Digitalisierungsmodulen, oder so lange dauern 9 CAMAC-Zyklen. Da die Flexibilität, die durch die Listensteuerung ermöglicht wird, eine der wesentlichen mit diesem Datenaufnahmesystem erzielten Fortschritte ist, wird dieser Zeitverlust als prinzipbedingt angesehen und wissentlich in Kauf genommen.

Das entwickelte Datenaufnahmesystem ist für einen weiten Bereich von Anwendungen konzipiert. Voraussetzung dafür, daß in einem konkreten Einsatzfall eine optimale Leistung erreicht wird, ist, daß es für diese Aufgabe geeignet konfiguriert wird. Das betrifft vor allem die Verteilung der Last auf die Prozessoren bzw. Busse sowie die Bereitstellung ausreichender Pufferbereiche.

Anhang B

Beschreibung der EMS-Messages

B.1 Format der Messages

B.1.1 Aufbau

Messages sind Datenpakete, die zwischen den Beteiligten der EMS-Kommunikation ausgetauscht werden (PDUs). So werden die Requests, Confirmations, unsolicited Messages und spezielle, für den Benutzer unsichtbare „interne Messages" (siehe Erklärung in B.1.2) in Form solcher Pakete übertragen. Diese verschiedenen Arten von Messages mit verschiedenen Einsatzzwecken werden im folgenden als *Messageklassen* bezeichnet.

Das EMS-Protokoll beinhaltet keine Vorkehrungen zur Fehlerkorrektur oder zur Neuübertragung eventuell verlorengegangener Messages. Vorausgesetzt wird also ein zuverlässiger Transportweg bzw. ein entsprechendes unterliegendes Kommunikationsprotokoll.

EMS-Messages bestehen aus einem festen Header, der im wesentlichen die für den Transport wesentlichen Informationen enthält, und dem Datenteil. Die Interpretation des Datenteils ist vom Typ des Requests, der unsolicited Message bzw. vom Kontext, in dem die Confirmation entstand[1], abhängig.

Das Format der Daten folgt der XDR-Spezifikation ([55]). Die wichtigsten Konsequenzen daraus sind:

- Die Grundeinheit der Datenübertragung ist 32 Bit, der Standard-Zahlentyp ist also die 4-Byte-Ganzzahl. Bei Datenmengen, die kein Vielfaches dieser Einheit sind, entsteht Verschnitt.

- Es wird Network-Byte-Ordering (in der Bedeutung von IP, also „Big Endian") verwendet. Dank einer glücklichen Fügung verwenden gerade die weniger schnellen Prozessoren im Frontend diese Variante, so daß hier keine Leistungseinbuße entsteht[2].

[1] D.h., Parameter des Requests können das Format der Confirmation bestimmen.

[2] Es sei ausdrüchlich darauf hingewiesen, daß sich alle hier getroffenen Aussagen auf die Kommunikation, nicht auf die Eventdaten, beziehen.

Wenn im folgenden von einer 'Ganzzahl' oder auch nur einem 'int'-Wert die Rede ist, ist damit, wenn nicht ausdrücklich anders erklärt, ein 32-Bit-Wort gemeint. Wichtigster zusätzlicher Datentyp ist die Zeichenkette (String). Deren Darstellung besteht aus einer Bytefolge, der die Länge vorangestellt wird.

Zur Beschreibung XDR-konformer Datenstrukturen existiert eine eigene strukturierte Notation, sie ist ebenfalls in [55] beschrieben. In diesem Abschnitt sollen nur ausgewählte Aspekte des Protokolls behandelt werden, dafür wird eine einfache, allgemeinverständliche Beschreibung verwendet.

Der Messageheader hat folgenden Aufbau:

Message-Header (6 Worte)	
Wort No.	Bedeutung
1	Länge
2	Client-ID
3	Server-ID
4	Messagetyp
5	Flagfeld
6	Transaction-ID

Die einzelnen Felder haben folgende Bedeutung:

Länge: Länge des Datenteils (Netto-Länge, der Header ist nicht eingerechnet.)

Client-ID: Eindeutige Kennzeichnung des Clients

Server-ID: Eindeutige Kennzeichnung des Servers (des VEDs)

Die Client- und Server-IDs werden von der Client-Seite bzw. dem Kommunikationsprozeß vergeben und dienen der Identifikation und der Verteilung der Messages. Anhand dieser IDs kann der Kommunikationsprozeß die Messages multiplexen, d.h., mehrere logische Client-Server-Verbindungen auf derselben unterliegenden physikalischen Verbindung organisieren. (Näheres zur Rolle des Kommunikationsprozesses in Kapitel B.2.) Der zulässige Wertebereich der IDs umfaßt alle möglichen 32-bit-Zahlen, mit Ausnahme einiger für spezielle Fälle reservierter Kennungen. Diese sind:

Wert	Bedeutung	Verwendung
-1	ungültig	wird für interne Messages verwendet, falls keine Verbindung (mehr) besteht, sowie im Fehlerfall
-2	Kommunikationsprozeß	als Ziel von unsolicited Messages oder internen Requests von Clients
-3	unbekannt	als Quelle von unsolicited Messages, wenn der Server seine ID nicht kennt

Messagetyp: Requesttyp, Typ der internen Message oder der unsolicited Message. Wie diese Zahl zu interpretieren ist, hängt von der durch das Flagfeld festgelegten Messageklasse ab.

Flagfeld: Unterscheided Requests und Confirmations[3], interne und unsolicited Messages, kennzeichnet somit die Messageklasse. Die einzelnen Bits haben folgende Bedeutung:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
unbenutzt															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
unbenutzt												NL	UM	IM	C

Bit No.	Name	Bedeutung	Verwendung
0	C	Confirmation	Unterscheidung zwischen Request und Confirmation
1	IM	interne Message	kennzeichnet interne Messages
2	UM	unsolicited Message	kennzeichnet unsolicited Messages
3	NL	nicht loggen	Message soll nicht im Log-Protokoll erscheinen (z.B., weil sie selbst eine Log-Message ist und ihr Loggen zur Rekursion führen würde)

Transaction-ID: Kennzeichnung des Requests, wird von Client vergeben, um die Confirmation später dem Request zuordnen zu können. Die '0' wird für unsolicited Messages verwendet.

Der Aufbau des Datenteils ist, wie oben erwähnt, vom Requesttyp abhängig. Eine Beschreibung findet sich in [84], hier soll nur ein Beispiel gezeigt werden: Bild B.1 zeigt die Varianten beim Erzeugen einer EMS-Domain.

DownloadDomain-Request (mind. 2 Worte)					
Wort No.	Bedeutung				
1	Typ				
	Modulliste	LAM	Trigger	Dataout	Datain
2	Index				
3	Anzahl	Output	Prozedur	Buffertyp	
4	Adresse	Prozedur-liste	Parameterzahl	Größe	Adreßtyp
5	Typ		Parameter	Priorität	Adresse
6	⋮		⋮	Adreßtyp	⋮
7				Adresse	
8				⋮	

Abbildung B.1: Aufbau des Datenbereichs eines Download-Domain-Requests

[3]Hier sind Request und Indication bzw. Response und Confirmation äquivalent.

Alle Confirmations enthalten als erstes Datenwort einen Fehlercode, der den Erfolg der Operation anzeigt, oder den Grund des Scheiterns. Eine Auswahl aus den (ca. 50) Meldungen zeigt Bild B.2.

OK	kein Fehler
IllRequest	falscher Requesttyp
NotImpl	Request ist nicht implementiert
ArgNum	falsche Anzahl der Argumente
ArgRange	ein Argument liegt nicht im zulässigen Bereich
IllObject	unbekannter (oder nicht implementierter) Objekttyp
ObjNonDel	Objekt ist nicht löschbar
ObjDef	Objekt ist schon definiert
⋮	
NoMem	zu wenig Speicher
BufOverfl	Datenpuffer zu klein
NoVED	VED-ID nicht bekannt
System	Fehler bei Systemaufruf
⋮	
HW	Fehler in der Hardware
BadModTyp	Falscher Modultyp
NotOwner	nicht Eigentümer des Objekts

Abbildung B.2: Auswahl aus den definierten EMS-Fehlercodes.

B.1.2 Messageklassen

Eine Message ist entweder ein Request, eine Confirmation oder eine unsolicited Message. Diese 3 Arten können sowohl in ihrer „Normalform“ als auch als „interne Messages“ auftreten. (D.h., es gibt 6 Messageklassen.) Die Unterscheidung erfolgt anhand von 3 Bits im Flagfeld des Messageheaders, sie sind oben mit 'C', 'IM' und 'UM' bezeichnet. Bei unsolicited Messages ist auch das Confirmation-Bit gesetzt, die Kombination $\neg$'C' $\cdot$ 'UM' ergibt also keinen Sinn. Alle anderen Kombinationen sind zulässig. Die folgende Tabelle zeigt alle Klassen, dazu ist aufgelistet, auf welche Weise das Typ-Feld im Header gedeutet wird. Die einzelnen Messagetypen werden in den folgenden Kapiteln aufgelistet: 'Requesttyp' in B.3, 'Int-Request-Typ' in B.4 und 'Unsol-Message-Typ' in B.5.

UM	IM	C	Klasse	Bedeutung des Typ-Feldes
0	0	0	EMS-Request	Requesttyp
0	0	1	EMS-Confirmation	Requesttyp
0	1	0	Interner Request	Int-Request-Typ
0	1	1	Interne Confirmation	Int-Request-Typ
1	0	1	unsolicited EMS-Message	Unsol-Message-Typ
1	1	1	interne unsolicited Message	Unsol-Message-Typ

EMS-Requests und -Confirmations werden benutzt, um EMS-Services in Anspruch zu nehmen. Ein Request wird von einem Client an einen Server übermittelt, bei Confirmations ist die Richtung umgekehrt.

Interne Requests und Confirmations werden zwischen Client und Kommunikationsprozeß oder zwischen Kommunikationsprozeß und Server ausgetauscht. Sie dienen der Übermittlung von Verbindungsparametern, der Kontrolle des Kommunikationsprozesses und der Identifizierung von Clients. Diese Funktionen werden von unteren Schichten der Client-Bibliotheken und vom Kommunikationsprozeß erledigt, so daß der Anwendungsprogrammierer sich nicht um ihre Existenz kümmern muß.

Jeder Request wird mit einer Confirmation beantwortet, auch, wenn die befohlene Aktion fehlerfrei ausgeführt wurde und keine Daten liefert. Im Header der Confirmation sind die Felder 'Länge', 'Client-ID', 'Server-ID', 'Messagetyp' und 'Transaction-ID' mit den Einträgen im zugehörigen Request-Header identisch, im 'Flagfeld' ist das Flag 'Confirmation' gesetzt. In das Feld 'Länge' wird von Server die Länge des Datenbereiches der Antwort eingetragen.

Unsolicited Messages werden zur Signalisierung besonderer Zustände benutzt, sie treten asynchron auf und werden nicht beantwortet. Die Transaction-ID wird für unsolicited Messages auf '0' gesetzt.

In ihrer internen Form werden unsolicited Messages zur Protokollierung von Aktionen und Datenpaketen verwendet. Dies liegt ebenfalls außerhalb der Sichtweite des normalen Anwendungsprogrammierers.

B.2 Der Kommunikationsprozeß

Die Verwendung eines Client-Server-Systems wie EMS für die Experimentkontrolle und Datenaufnahme hat, neben vielen anderen Vorzügen, den Zweck, die rechenintensiven, aber weitgehend hardwareunabhängigen, Aufgaben von der eher auf Reaktionsgeschwindigkeit spezialisierten, hardwarebezogenen eigentlichen Datenaufnahme zu trennen. Sinnvoll ist es deshalb, die auf den Frontend-Prozessoren laufenden Server-Programme so einfach und effektiv wie möglich zu implementieren. Besonders Netzwerkaktivität erzeugt relativ lange Latenzzeiten, die sich in der Praxis als Totzeit niederschlagen.

So wurde bei der Implementierung der EMS-Server-Programme darauf verzichtet, die Verwaltung mehrerer Netzwerkverbindungen vorzusehen. So kann ein Server nur jeweils einen Request über eine physikalische (d.h., vom unterliegenden System bereitgestellten) Verbindung empfangen, und erst nach dessen Bearbeitung (und Rücksendung der Confirmation) den nächsten.

Um zu ermöglichen, daß mehrere Clients quasi gleichzeitig mit einem Server arbeiten können, wurde das Konzept des 'Kommunikationsprozesses' eingeführt. Dieser unterhält jeweils eine Verbindung zu jedem Client und jedem Server, und leitet alle Messages an den oder die entsprechenden Empfänger weiter. Für einen Server erscheinen so alle Clients als ein einziger. (Die Identifizierung von Client und Server durch die IDs im Messageheader ist für den Server transparent.) Auch wird eine unsolicited Message eines Servers mit der Zielangabe 'Kommunikationsprozeß' an alle logisch damit verbundenen Clients

weitergeleitet.

Der Kommunikationsprozeß bildet mit der Kommunikationsbibliothek der Serverprogramme eine Einheit. Diese beiden tauschen 'interne Messages' aus, um Aktionen auszuführen, die im eigentlichen EMS-Protokoll nicht vorgesehen sind. Näheres dazu ist in Kapitel B.4 beschrieben. Sie müssen auch gemeinsamen Konventionen bezüblich der Vergabe der Client-und Server-IDs folgen. Falls die Clients die Server-IDs selbst festlegen, sollte der Kommunikationsprozeß diese auf systemweit einheitliche Werte abbilden. Die Einzelheiten sind implementierungsabhängig.

B.3 Übersicht über die EMS-Services

Request-typ	Name	angesprochenes Objekt	Funktion
1	Initiate	VED	Initialisierung der Verbindung
2	Conclude		Deinitialisierung der Verbindung
3	ResetVED		Zurücksetzen
4	GetVEDStatus		Ermittlung von Parametern
5	GetNameList		Ermittlung der definierten Objekte
6	GetCapabilityList		Ermittlung der lokalen Prozeduren
7	DownloadDomain	Domain	Beschreiben
8	UploadDomain		Lesen
9	DeleteDomain		Löschen
10	CreateProgramInvocation	Programm-invokation	Erzeugen
11	DeleteProgramInvocation		Löschen
12	StartProgramInvocation		Starten
13	ResetProgramInvocation		Stoppen und zurücksetzen
14	ResumeProgramInvocation		Wiederaufnahme nach Stop
15	StopProgramInvocation		zeitweiliges Stoppen
16	GetProgramInvocationAttr		Ermittlung des Status
17	CreateVariable	Variable	Erzeugen
18	DeleteVariable		Löschen
19	ReadVariable		Lesen
20	WriteVariable		Beschreiben
21	GetVariableAttributes		Ermittlung des Status
22	CreateIS	Instr.-Sys.	Erzeugen
23	DeleteIS		Löschen
24	DownloadISModulList		Definieren der Bestandteile
25	UploadISModulList		Lesen der Bestandteile
26	DeleteISModulList		Löschen der Bestandteilliste
27	DownloadReadoutList		Definieren einer Readoutliste
28	UploadReadoutList		Lesen einer Readoutliste
29	DeleteReadoutList		Löschen einer Readoutliste
30	EnableIS		zum Readout zulassen
31	DisableIS		Readout verbieten
32	GetISStatus		Ermittlung des Status
33	ResetISStatus		unbenutzt
34	TestCommand		Testen einer Kommandoliste
35	DoCommand		Ausführen einer Kommandoliste
36	WindDataout	Dataout	Verändern der Position
37	WriteDataout		Schreiben eines Datensatzes
38	GetDataoutStatus		Ermittlung des Status
39	EnableDataout		Einschalten
40	DisableDataout		Ausschalten

B.4 Typen der internen Requests

Message-typ	Name	Partner			Funktion
		Client	Kommu	Server	
1	ClientIdent	▷	◁		Identifizierung des Clients
2	Open		▷	◁	Öffnen einer physikalischen Verbindung
3	Close		▷	◁	Schließen einer physikalischen Verbindung
4	OpenVED	▷	◁		Öffnen einer logischen Verbindung zum VED
5	CloseVED	▷	◁		Schließen einer logischen Verbindung
6	GetHandle	▷	◁		Überprüfung des VED-Handles
7	GetName	▷	◁		Ermittlung des VED-Namens zum Handle
8	GetVEDs	▷	◁		Ermittlung aller bekannten VEDs
9	GetOpenVEDs	▷	◁		Ermittlung aller geöffneten VEDs
10	SetUnsolicited	▷	◁		Erlauben/Verbieten von unsolicited Messages
11	SetSignal	▷	◁		unbenutzt
12	SetSignalUnsolicited	▷	◁		unbenutzt
13	SetExitCommu	▷	◁		Beendigung des Kommunikationsprozesses
14	SetLog	▷	◁		Anfordern von Log-Messages
15	SetDebuglevel	▷	◁		Steuerung von Debugausgaben des Kommu

B.5 Typen der unsolicited Messages

Message-typ	Name	Funktion
1	ServerDisconnect	Verbindung ist abgebrochen
2	Bye	Kommunikationsprozeß wird beendet
3	Test	Testpaket vom Server
4	LAM	Mitteilung über LAM
5	LogText	Protokoll über Aktionen
6	LogMessage	Protokoll über Messages
7	RuntimeError	Mitteilung über Fehler in Server
8	Data	Datenpaket vom Server

Abbildungsverzeichnis

Literaturverzeichnis

[1] G. I. Budker. *Atomnaya Energiya* **22**, 346 (1967).

[2] G. I. Budker, N.S. Dikanskii, V.I. Kudelainen, I.N. Meshkov, V.V Parkhomchuk, D.V. Pestrikov, A.N. Skrinskii, B.N. Sukhina. *Particle Accelerators* **7**, 197 (1976).

[3] S. van der Meer. CERN/ISR, P.O./72-31 (1972).

[4] Theo Mayer-Kuckuk. „Das Kühler-Synchrotron COSY und seine physikalischen Perspektiven". In *Vorträge*. Rhein.- Westf. Akad. d. Wissenschaften (1988), N 359.

[5] J. Bisplinghoff, F. Hinterberger, W. Scobel. „First Events from EDDA at COSY". In *IKP Annual Report*, Band 2879, S. 3, Berichte des Forschungszentrums Jülich (1993), ISSN 0944-2952.

[6] A. Berg, G. Bohlscheid, M. Drochner, J. Ernst, L. Freindl, C. Henrich, S. Igel, R. Jahn, L. Jarczyz, R. Joosten, S. Kistryn, S. Kliczewski, G. Lippert, H. Machner, R. Maschuw, C. Nake, T.v. Oepen, D. Rosendaal, P.v. Rossen, K. Scho, J. Smyrski, A. Strzałkowski, R. Tölle, P. Żołnierczuk. „First External COSY-Beam at BIG KARL". In *IKP Annual Report*, Band 2879, S. 5, Berichte des Forschungszentrums Jülich (1993), ISSN 0944-2952.

[7] H. Machner. „The Experimental Program at COSY". Technischer Bericht KFA-IKP(I)-1993-3, KFA Jülich / IKP (1993), internal paper.

[8] W. F. Cartwright, C. Richman, M. N. Whitehead, H. A. Wilcox. *Phys. Rev.* **78**, 823 (1950).

[9] Frank S. Crawford, M. Lynn Stevenson. *Physical Review* **97(5)**, 1305 (1955).

[10] D. Aebischer, B. Favier, L.G. Greenhaus, R. Hess, A. Junod, C. Lechanoine, J.-C. Nikles, D. Rapin, D.W. Werren. *Nuclear Physics B* **106**, 214 (1976).

[11] B. G. Richie. *Phys. Rev. C* **44(1)**, 533 (1991).

[12] Carl M. Rose. *Physical Review* **154(5)**, 1305 (1967).

[13] C. Richard-Serre, W. Hirt, D.F. Measday, E.G. Michaelis, M.J.M. Saltmarsh, P. Skarek. *Nuclear Physics B* **20**, 413 (1970).

[14] B.G. Richie, R.C. Minehart, T.D. Averett, G.S. Blanpied, K. Giovanetti, B.M. Preedom, D. Rothenberger, L.C. Smith, J.R. Tinsley. *Physical Review Letters* **66(5)**, 568 (1991).

[15] D.A. Hutcheon, E. Korkmaz, G.A. Moss, R. Abegg, N.E. Davison, G.W.R. Edwards, L.G. Greeniaus, D. Mack, C.A. Miller, W.C. Olsen, I.J. van Heerden, Ye Yanlin. *Physical Review Letters* **64(2)**, 176 (1990).

[16] C.J. Horowitz. *Phys. Rev. C* **48(6)**, 2920 (1993).

[17] J. A. Niskanen. *Phys. Rev. C* **49**, 1285 (1994).

[18] L. Montanet et al. (Particle Data Group). *Physical Review D* **50**, 1173 (1994).

[19] B. Gugulski, J. Ernst, A. Heczko, C. Henrich, L Jarczyk, K. Kilian, A. Kozela, J. Majewski, A. Misiak, W. Oelert, K. Scho, J. Smyrski, J. Strzałkowski. „Construction and Test of Drift Chambers for the new Jülich Accelerator COSY". Technischer Bericht, KFA Jülich, IKP (1992), Interner Bericht KFA-IKP(I)-1992-3.

[20] M. Jochmann, S. Fuß, K.-D. Müller, R. Reinartz. „A New Sixteen Channel Leading Edge Discriminator for Time-Of-Flight Experiments at COSY". In *Conference Record of the 1992 IEEE Nuclear Science Symposium and Medical Imaging Conference, Vol. 1*, S. 362–364 (October 1992), Orlando, Florida.

[21] H.G. Essel, J. Hoffmann, W. Ott, M. Richter, D. Schall, H. Sohlbach, W. Spreng. „GOOSY-VME The Data Acquisition and Analysis System at GSI". *IEEE Trans. Nucl. Sci.* **39(2)**, 248–251 (1992).

[22] H.G. Essel, J. Hoffmann, W. Ott, M. Richter, D. Schall, H. Sohlbach, W. Spreng, T. Batsch. „GOOSY-VME Hardware Components and Trigger System". Technischer Bericht, GSI Darmstadt (1991).

[23] H.G. Essel, J. Hoffmann, M. Richter, H. Sohlbach, W. Spreng. „An Integrated Data Acquisition and Analysis System at GSI". *IEEE Trans. Nucl. Sci.* **36(5)**, 1523–1527 (1989).

[24] H.G. Essel, J. Hoffmann, M. Richter, H. Sohlbach, W. Spreng. GOOSY VME System Desciption. GSI Darmstadt, Planckstr. 1, Darmstadt, erste Auflage (February 1989).

[25] T. Plaich, R. Freifelder, N. Herrmann, J.G. Keller, P. Koczon, M. Krämer, Y. Leifels, V. Lindenstruth, W.F.J. Müller, R. Schmidt, K. Teh. „Acquisition, Analysis and Control Software for the Experiments in CAVE B". *GSI Annual Report* (1989).

[26] M. Steinke, S. Brand, P. Ringe. „Data Acquisition and Experiment Control at the COSY TOF Spectrometer". In *Conference Record of the eigth Conference on real-time Computer Applications in nuclear, particle and plasma Physics, Vancouver*, S. 404–406 (June 1993).

[27] F. Schwandt, „TDAS OS-9 ISKP User's Guide“ (May 1994), (Universität Bonn).

[28] CERN/ECP-DS, „SPIDER User Manual“.

[29] C. Parkman. „For Hire: Turn-Key VMEbus Systems“. *Online (CERN Computing Newsletter)* **7**, 16 (1993).

[30] K.-H. Watzlawik, R. Nellen, T. Noll, M. Karnadi, H. Machner. „FATIMA. A Data Acquisition System for Medium Scale Experiments“. *IEEE Transactions on Nuclear Science* **39**, 154 (1992).

[31] Norbert Nicolay, „FERA-Analysator“. Bericht der Universität zu Köln / IKP (März 1992).

[32] M.W. Luig, S. Albers, F. Giesen, N. Nicolay, E. Radermacher, M. Wilhelm, R. Wirowski. „Das Kölner Analysator- und Auswertesystem“. In *Verhandlungen*, S. 1930. DPG (März 1994), Poster.

[33] National Instruments Corporation, Austin / USA. LabVIEW for Windows User Manual (August 1993).

[34] Peter Eisenbach. „LabVIEW Überwachung von Crates über ein PROFIBUS-Netzwerk“. Diplomarbeit, FH Aachen, Fachbereich Elektrotechnik (Mai 1994), Interner Bericht KFA-ZEL-IB-500-4/94.

[35] N. Dolfus, „Schrittmotorsteuerung mit LabVIEW“. KFA Jülich / IKP (Internal Paper).

[36] D. Jacobs. „Support of LabVIEW“. *Online (CERN Computing Newsletter)* **6**, 16 (1993).

[37] S. Vascotto. „CASCADE — Distributed Real-Time Data-Acquisition“. *Online (CERN Computing Newsletter)* **6**, 8 (1993).

[38] H. Kramer. „Das Experimentsteuerungssystem der Drei-Spektrometer-Anlage an MAMI“. In *Verhandlungen*, S. 1960. DPG (März 1994), Gruppenbericht.

[39] CANU (COSY-Arbeitsgemeinschaft Nordrhein-Westfälischer Universitäten) — Kommission Datenaufnahme COSY, „Summary of Recommendations“. Internal Paper (December 1991).

[40] W. Erven, J. Holzer, H. Kopp, H. W. Loevenich, W. Meiling, K. Zwoll, M. Karnadi, R. Nellen, K.-H. Watzlawik. „COSY Data Acquisition System for Physical Experiments“. *IEEE Trans. Nucl. Sci.* **39(2)**, 148–153 (1992).

[41] International Organization for Standardization, „ISO/IEC 26.11458 VICbus“ (1993).

[42] C. F. Parkman. „VICbus: VME inter-crate-bus — A versatile cable bus“. *IEEE Trans. Nucl. Sci.* **39(4)**, 77–84 (1992).

[43] Grady Booch. Object-Oriented Design With Applications. Benjamin Cummings (1991).

[44] James Rumbaugh et al. Object-Oriented Modeling and Design. Prentice-Hall (1991).

[45] Sally Shlaer, Stephen J. Mellor. Object-Oriented Systems Analysis: Modeling the World in Data. Prentice-Hall (1988).

[46] L. Cardelli, P. Wegner. „On understanding Types, Data Abstraction, and Polymorphism". *ACM Computing Surveys* **17(4)** (1985).

[47] General Motors Corperation, „Manufacturing Automation Protocol — Version 3.0" (1988).

[48] International Organization for Standardization, „ISO9506 — Manufacturing Message Specification" (1990).

[49] K. Zwoll, M. Drochner, W. Erven, J. Holzer, H. Kopp, H. W. Loevenich, P. Wüstner, K.-H. Watzlawik, N. Brummund, M. Karnadi, R. Nellen, J. Stock, S. Dienel, K.-H. Leege, W. Oehme. „Flexible Data Acquisition System for Experiments at COSY". *IEEE Trans. Nucl. Sci.* **41(1)**, 37–44 (1994), also at: Conference Records of the Eighth Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics (RT93), Vancouver, Canada, June 8–11, 1993.

[50] Doug Van Kirk. „What is client/server computing, anyway?" *InfoWorld* **15(46)**, 107 (1993).

[51] International Standards Organization. Information Processing Systems — Open Systems Communications — Basic Reference Model (1984), ISO 7498.

[52] H. Kleines. „Leistungsanalyse eines ISO-Transportprotokoll-Profils am Beispiel der iNA 960". Diplomarbeit, RWTH Aachen (Juni 1987).

[53] D. Smallberg, „RFC 832: Who talks TCP?" (July 1982).

[54] W.J. Cody et al. „A Proposed Radix- and Word-length-independent Standard for Floating-point Arithmetic". *MICRO (IEEE Magazine)* (1984).

[55] Sun Mikrosystems, „RFC 1014: XDR: External Data Representation Standard" (1987).

[56] K. Zwoll, M. Drochner, P. Wüstner. „Objektorientierte Modellierung des On-Line-Datenerfassungssystems für COSY-Experimente — V2.0". Internal Report IB-KFA-ZEL 501293, KFA Jülich / ZEL (November 1993).

[57] K. Zwoll, M. Drochner, W. Erven, J. Holzer, H. Kopp, P. Wüstner. „Objektorientierte Modellierung des On-Line-Datenerfassungssystems für COSY-Experimente". Internal Report IB-KFA-ZEL 501392, KFA Jülich / ZEL (1992).

[58] W. Erven, J. Holzer, H. Kopp, H. W. Loevenich, W. Meiling, K. Zwoll, J. Bovier, G. Re. „Intelligent CAMAC Controller with CC-A2 Functionality and VICbus Interface". *IEEE Trans. Nucl. Sci.* **39(4)**, 853–857 (1992), also at: IEEE Symposium on Nuclear Science, Santa Fe, November 1991.

[59] J. Holzer, M. Drochner, W. Erver, K. Zwoll. „VICbus-VSB-Interface". Interner Bericht IB-KFA-ZEL 500592, KFA Jülich / ZEL (1992).

[60] J. Holzer, F.-J. Kaiser, H. Stoff, G. Re. „TURBOchannel-VICbus-Interface". In *Open Bus Systems '92, October 13-15, 1992, Zürich, Proceedings*, S. 409–412. KFA Jülich / CES Genf (1992).

[61] R. Nellen, R. Krause, K.-H. Watzlawik. „FERA System Controller, FSC". In *IKP Annual Report*, Band 2590, S. 278, Berichte des Forschungszentrums Jülich (1991), ISSN 0366-0885.

[62] E. Chesi, P. Martinengo. „Digital Read-out for analog multiplexed Signal". Technischer Bericht, CERN (January 1988), PS 202.

[63] H.G. Essel, J. Hoffmann, W. Ott, M. Richter, D. Schall, H. Sohlbach, W. Spreng. GOOSY VME System Desciption — Trigger and Synchronisation. GSI Darmstadt, Planckstr. 1, Darmstadt (September 1988).

[64] J. Twardowski, H. Pohl, K. Zwoll. „Softwarebeschreibung für Anwender: Neutronen Spin Echo Experiment — V 1.0". Technischer Bericht, KFA Jülich / ZEL (Februar 1995), Interner Bericht KFA-ZEL-IB-501594.

[65] M. Drochner. Das EMS-Client-Programm 'klient' – Benutzeranleitung –. KFA Jülich / ZEL (Juli 1994), Interner Bericht KFA-ZEL-IB-501094.

[66] K.-H. Watzlawik, N. Brummund, S. Dienel, M. Drochner, W. Erven, M. Karnadi, H. Kopp, K.-H. Leege, H. Loevenich, R. Nellen, W. Oehme, J. Stock, P. Wüstner, K. Zwoll. „COSY Experiment Data Acquisition". In *IKP Annual Report*, Band 2879, S. 217, Berichte des Forschungszentrums Jülich (1993), ISSN 0944-2952.

[67] Mark A. Linton, Paul R. Calder, John A. Interrante, Steven Tang, John M. Vlissides. InterViews Reference Manual — Version 3.1 (December 1992).

[68] P. Wüstner, M. Drochner, G. Kemmerling, K. Zwoll, H. Loevenich, J. Hagedorn, W. Erven. „Konzept und Implementierung einer objektorientierten grafischen Benutzeroberfläche für EMS". In *Bericht der Frühjahrstagung der Studiengruppe für Elektronische Instrumentierung*, S. 34–36. FZ Rossendorf (Juni 1995), FZP-96.

[69] John Ousterhout. Tcl. University of California at Berkeley, 7. Auflage (1993).

[70] John Ousterhout. Tk. University of California at Berkeley, dritte Auflage (1993).

[71] Witten, Neal, Cleary. „Arithmetic Coding for Data Compression". *Communications of the Association of Computing Machinery* **30(6)**, 520–540 (1987).

[72] David C. Cressman. „Analysis of Data Compression in the DLT2000 Tape Drive". *Digital Technical Journal* **6(2)**, 62ff. (1994).

[73] T.C. Bell, J.G. Cleary, I.H. Witten. Text Compression. Prentice-Hall (1989).

[74] Ross N. Williams, „A painless Guide to CRC Error Detection Algorithms". Internet Article (August 1993).

[75] Donald E. Knuth. Seminumerical Algorithms, Band 2 von *The Art of Computer Programming.* Addison-Wesley, Reading, MA, USA (1969), ISBN 0-201-03802-1.

[76] Andrew S. Tanenbaum. Computer Networks. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, zweite Auflage (1988), ISBN 0-13-162959-X.

[77] M. Drochner. „Multiprozessor VME-Eventbuilder auf Basis der objektorientierten Experimental Message Specification". Technischer Bericht, KFA Jülich / ZEL (November 1993), Berichte zum Datenerfassungssystem für physikalische Experimente an COSY (Internal Paper).

[78] M. Drochner, G. Kemmerling, H. Machner, K. Zwoll. „The Data Acquisition System for GEM". In *IKP Annual Report*, Band 2879, S. 227, Berichte des Forschungszentrums Jülich (1993), ISSN 0944-2952.

[79] M. Gasthuber, „Ein System zur Auswertung und graphischen Darstellung von Vielparameterexperimenten bei COSY". Report beim COSY-Arbeitstreffen, KFA Jülich (October 1992).

[80] P.A. Żołnierczuk, H. Machner, L. Jarczyk, A. Strzałkowski. „The On-Line/Off-Line Data Analysis System for the GEM Experiment at COSY". In *IKP Annual Report*, Band 2879, S. 229, Berichte des Forschungszentrums Jülich (1993), ISSN 0944-2952.

[81] Computing CERN Application Software Group, Networks Division, „PAW, Physics Analysis Workstation (CERN Program Library entry Q121)" (July 1992), Version 1.14.

[82] SAS Institute Inc., Cary, NC, „SAS 6.09" (1992).

[83] M. Weymann. „Measurements of VME and VSB data transfer rates between the CES dual 68040 processor boards FIC 8234 (25 MHz) and FIC 8243 (33 MHz)". Technischer Bericht, CES S.A., Genf (November 1993).

[84] M. Drochner. „Spezifikation der EMS-Messages (Typen und Formate)". Technischer Bericht, KFA Jülich / ZEL (1994).

Die vorliegende Arbeit wurde in den Jahren 1994 und 1995 am Zentrallabor für Elektronik des Forschungszentrums Jülich angefertigt.

Die Betreuung erfolgte, soweit die Entwicklung des Datenaufnahmesystems betroffen war, durch Herrn Dr. K. Zwoll. Für das GEM-Experiment ist dessen Sprecher, Herr Dr. H. Machner, verantwortlich.

Entsprechend der Natur der Aufgabe, einer Verbindung verschiedener Fachbereiche, und der Komplexität der Materie haben viele Personen, ob durch Schaffung der organisatorischen oder technischen Voraussetzungen, durch Mitarbeit am EMS-Projekt bzw. am GEM-Experiment oder durch Beratung und Hilfe bei auftretenden Problemen einen Beitrag zum Zustandekommen dieser Arbeit geleistet. Allen, die sich in den obigen Worten wiedererkennen, und allen, die sich sonst in irgendeiner Weise um den Autor oder sein Werk verdient gemacht haben, sei hiermit herzlich gedankt.

FORSCHUNGSZENTRUM JÜLICH GmbH

KFA

Jül-3218
April 1996
ISSN 0944-2952