Entwicklung eines neuen Datenakquisitionssystems für das CB-ELSA-Experiment

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch–Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich–Wilhelms–Universität Bonn

vorgelegt von

Christoph Schmidt

aus

Köln

Bonn 2004

Angefertigt mit Genehmigung der Mathematisch–Naturwissenschaftlichen Fakultät der Rheinischen Friedrich–Wilhelms–Universität Bonn

Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn
`http://hss.ulb.uni-bonn.de/diss_online` elektronisch publiziert

1. Referent:	Prof. Dr. E. Klempt
2. Referent:	Prof. Dr. B. Schoch
Tag der Promotion: 5.11.2004	

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die ersten Experimente in der Teilchenphysik basierten auf einfachen Detektionsmechanismen, wie z.B. dem Beobachten von Lichtblitzen mittels Leuchtschirmen. So führten Hans Geiger und Ernest Marsden unter der Anleitung von Ernest Rutherford 1906 ein Experiment durch, indem sie α-Teilchen an einer Goldfolie streuten und die abgelenkten Teilchen mittels eines Zinksulfid-Schirms beobachteten. Sie zählten die Lichtblitze, die unter bestimmten Winkeln auftraten.

Zu dieser Zeit waren Computer noch in weiter Ferne, d.h. diese Experimente wurden komplett "händisch" von den Experimentatoren durchgeführt, da in diesem Zeitraum noch an den Grundlagen von Computern geforscht wurde. So patentierte Nikola Tesla im Jahre 1903 elektrische und logische Schaltkreise (Gatter). Erst im Jahre 1931 erfand Konrad Zuse den ersten frei programmierbaren Rechner. Dieser "Z1" wurde in Deutschland gebaut und 1938 fertiggestellt.

Im Jahre 1952 erfand Donald Glaser die Blasenkammer. Mit Hilfe dieser Blasen- oder Nebelkammer wurden wichtige Erkenntnisse in der Teilchenphysik gewonnen. Im Fall der Blasenkammer befindet sich in einem Gefäß eine Flüssigkeit, die in einem superheated-Zustand ist. Tritt nun ein Teilchen durch dieses Medium, hinterlässt es eine Spur aus kleinen Bläschen. Diese Spur wird dann fotografisch auf einem Film festgehalten. Die Analyse dieser Filme wurde von Hand durchgeführt, und die interessanten Ereignisse (siehe Abbildung 1.1) mussten mühsam aus den Filmen herausgesucht und vermessen werden.

In dieser Zeit erschienen auch die ersten kommerziellen Computeranlagen auf dem Markt (IBM 650, System 701, 704). So wurden z.B. in den USA an der Universität Brookhaven die gemessenen Werte der Teilchenspuren auf Lochkarten übertragen und die kinematischen Größen mit Hilfe einer IBM 650 per Programm ausgerechnet [24]. Diese Systeme fanden aber nur Anwendung in der Analyse und waren kein Werkzeug für den Experimentator. Es fehlte die Möglichkeit, bestimmte Ereignisklassen während des Experimentes herauszufiltern, um die Anzahl von "wertlosen" Ereignissen zu reduzieren.

Die Rechnersysteme aus dieser Zeit waren geprägt von Mainframes. Mainframes waren sehr große Rechner, raumfüllend, stromverschlingend und viele Millionen kostend, die ausschließlich Rechenleistung zur Verfügung stellten, auch *number cruncher* genannt. Der erste echte "Minicomputer" auf der Basis von integrierten TTL-Schaltkreisen war die PDP-Serie der Digital Equipment Corporation (kurz DEC). In den Jahren 1960 - 1970 erschienen viele Versionen dieser PDP-Serie. Eine davon war die PDP 8, mit der eine interessante Neuerung eingeführt wurde. Alle Komponenten der PDP 8 wurden über einen einzigen Bus, den Omnibus, miteinander verbunden. Die Kommunikation der Recheneinheit (CPU) mit Speicher und Ein- und Ausgabegeräten erfolgte über ein gemeinsames Bündel paralleler Drähte. CPU, Speicher und E/A-Geräte teilten sich diesen Bus. Dies führte zu einer drastischen Leistungssteigerung und zu einer kompakteren Bauweise der Rechner, die zum ersten Mal einen Einsatz als Steuer- und Datenspeichereinheit für physikalische Experimente ermöglichten.

In den 80er Jahren machten nun neue Detektortechnologien es möglich, die Spur eines geladenen Teilchens elektronisch zu erfassen. Die Erfindung der Vieldraht-Proportionalkammer war einer der Hauptschritte in Richtung auf dieses Ziel. Georges Charpak erhielt hierfür 1992 den Nobelpreis in Physik. Dies führte dazu, dass am CERN ein hybrides System, bestehend aus einer Blasenkammer (BEBC Big European Bubble Chamber) und weiteren Detektoren (Drahtkammern, Szintillationszähler, ...) [1], aufgebaut wurde. Mit Hilfe dieser elektronischen Zusatzdetektoren verfügte das Experiment über eine Selektionslogik, die es ermöglichte, bestimmte Ereignisse in den elektronischen Detektoren zu selektieren. Die damit verbundene Erhöhung der Ereignisraten und die zwangsläufigen Vergrößerungen der Datenmengen machten den Einsatz von Computern zwingend notwendig. Die Steuerung und Datenerfassung des Systems wurde mittels dieser "Minicomputer" der PDP-Serie von DEC durchgeführt. Dieser Synergieeffekt von Detektor- und Rechnerentwicklung erlaubte es somit zum ersten Mal, auch seltene Zerfallskanäle zu studieren. Durch immer kompliziertere Fragestellungen in der Physik wuchsen die Wünsche nach leistungsfähigeren Systemen zur Identifikation von Teilchen und zur Bestimmung ihrer kinematischen Größen in seltenen Ereignissen, wie z.B. bei der Suche nach den Z^0-Bosonen in dem UA-1-Experiment am CERN (1976, Super-Protonen-Synchrotron).

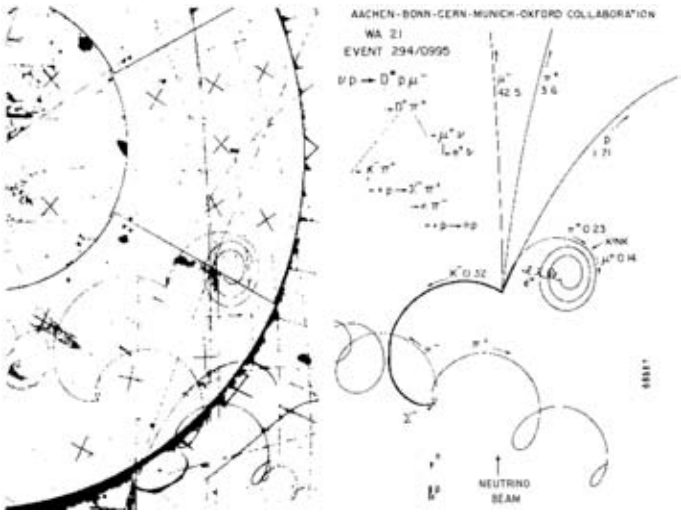


Abbildung 1.1: Ereignis in einer Blasenkammer

Die rasante Entwicklung der Rechnertechnologien und der massive Einsatz von leistungsfähigen Mehrprozessorsystemen bildeten die Grundlage zu diesen Erfolgen. Seitdem folgten weltweit viele weitere Experimente mit immer komplizierteren Systemen, bei denen in der Evaluierung der möglichen physikalischen Fragestellungen die derzeitig verfügbaren Datenakquisitionssysteme mit in Betracht gezogen werden müssen. Die Komplexität der Realisierbarkeit von modernen Detektorsystemen mit mehr als 10.000 elektronischen Erfassungskanälen wird unter anderem durch die Randbedingung dieser Rechnersysteme bestimmt, die mit den maximalen Datentransferraten und ihren Selektions- und Datenkompaktierungsalgorithmen eindeutige Machbarkeitsgrenzen vorgeben. Die Verwendung der leistungsfähigsten Rechner-Technologien steht somit in direktem Zusammenhang mit der Realisierbarkeit eines Experiments und der Kosteneffizienz (Betriebskosten für Beschleunigeranlagen). Die heutige Verfügbarkeit leistungsstarker Rechnerarchitekturen (Personal Computer als Konsumgut, Netzwerke als alltäglichen Kommunikationsweg) hat im Bereich der Datenakquisitionshardware zu kostengünstigen Lösungen geführt, die weitgehend auf Standardkomponenten und -konzepten aufbauen. Dies trifft allerdings nicht auf den Softwarebereich zu, da hier die Anforderungen von Seiten der Experimente so speziell sind, dass sie sich nur in geringem Umfang durch kommerzielle Standardprogramme erfüllen lassen. Daher kommt heute diesem Aufgabenbereich (Entwicklung von Auslese-, Steuer- und Datenspeicherprogrammen) bei der Erstellung von Datenerfassungssystemen eine besondere und entscheidende Bedeutung zu. Moderne Softwareentwicklungsmethoden, Programmqualitätskontrolle und geeignete Programmtestverfahren in Mehrprozessorsystemen sind zwingende Voraussetzungen für ein leistungsfähiges Gesamtsystem.

Die extrem schnelle Leistungsentwicklung im Bereich der Rechnerarchitekturen führt allerdings bei Experimenten, die über mehrere Jahre, bzw. Jahrzehnte betrieben werden, zur Notwendigkeit, in vertretbaren Zeitabständen die Hard- und Softwarelösungen zu überprüfen und den gegebenfalls geänderten Experimentbedingungen anzupassen. Dies traf auf das Crystal-Barrel-Experiment zu, das nach seinem Aufbau 1990 und seiner Messphase am LEAR[1] am Bonner Beschleuniger ELSA 1999 erneut zum Einsatz kam. Die Ausleseelektronik der einzelnen Detektorelemente wurde teilweise übernommen und in einigen Bereichen auf die neuen Gegebenheiten angepasst. Speziell der Crystal-Barrel [21] erhielt ein neues ADC-System zur schnelleren und besseren Auslese der Kristalle. Das Datenakquisitionssystem des LEAR-Experiments wurde an die Bedingungen bei ELSA adaptiert und in Betrieb genommen [19]. Dieses Setup lieferte Daten in der Zeit vom 1.2.2000 bis zum 5.4.2001, bei Energien von 1,4 GeV, 2,6 GeV und 3,2 GeV.

Zwischen 2001 und 2003 wurden mit dem Crystal-Barrel-Detektor und der TAPS-Kollaboration mit ihrem TAPS-Detektor weitere Messungen durchgeführt. Hierzu wurde dieser in Vorwärts-

[1] Low Energy Antiproton Ring des Europäischen Labors für Teilchenphysik, CERN

richtung modifiziert, so dass in dieser Richtung der TAPS-Detektor den Winkelbereich des modifizierten Crystal-Barrel-Detektors abdeckte. Dieser Aufbau lieferte Daten in der Zeit vom 10.12.2001 bis zum 24.4.2002, bei Energien bis zu 3,2 GeV.

Während dieser Strahlzeiten zeigte sich, dass auf Grund der zahlreichen Proposals für Experimente an ELSA das Datennahmesystem des CB-ELSA-Experimentes eine Erneuerung zu erfahren hatte. Dieses System lieferte dann in der Zeit vom 16.8.2002 bis zum 19.12.2003 weitere Daten im Bereich bis zu 3,2 GeV mit einer wesentlich höheren Datenrate und hieraus resultierend mit einer höheren Statistik und somit einer effektiveren Nutzung der Strahlzeiten.

Ziel dieser Arbeit war es, das Datennahmesystem neu zu entwickeln, sodass es folgenden Anforderungen genügt:

- Erhöhung des Datendurchsatzes
- Skalierbarkeit des Datennahmesystems
- Wiederverwendung der bestehenden Frontend-Elektronik
- Verwendung neuer leistungsfähiger Prozessorsysteme
- Berücksichtigung des möglichen Kostenrahmens unter zu Hilfenahme von kommerziellen Produkten (COTS[2]).
- Einfache Erweiterbarkeit durch Verwendung von Standardschnittstellen (Templates)

[2] **C**ommercial **o**f **t**he **S**hell

Kapitel 2

Physikalische Motivation

2.1 Hadronen - Der Teilchenzoo

In der Natur existieren Teilchen, wie Leptone (e, μ, ...), die wirklich elementare Teilchen zu sein scheinen. Sie sind nicht weiter teilbar, also punktförmig. Auf der anderen Seite existieren die Hadronen[1] (Proton, Neutron, π-Meson, ...), die komplexe ausgedehnte Gebilde zu sein scheinen.
Im Jahre 1947 fanden Rochester und Butler in einer Nebelkammeraufnahme die Spuren eines positiv und eines negativ geladenen Pions, die aus dem Zerfall eines neutralen Teilchens hervorgegangen sein mussten. Dieses Teilchen nannte man Kaon (K^0).
Im Jahre 1949 fand Powell das positiv geladene Kaon (K^+) durch den Zerfall $K^+ \rightarrow \pi^+\pi^+\pi^-$ in Kernreaktionen. Die Kaonen verhielten sich wie schwere Pionen (π-Mesonen) und wurden daher zusammen in eine Familie aufgenommen, die sich Mesonen nennt. Im Laufe der Zeit erweiterte sich diese Gruppe um weitere Teilchen, wie ω-, η-, ρ-, ϕ-Mesonen.
1950 fand Anderson am California Technology Institute eine ähnliche Spur wie bei der Entdeckung des K^0, jedoch entstanden nun ein π^- und ein Proton. Das Teilchen, das in Pion und Proton zerfallen ist, musste also wesentlich schwerer gewesen sein als das Proton. Man nannte es Lambda (Λ). In den nächsten Jahren wurden weitere schwere Teilchen gefunden, Σ, Ξ , Δ, usw. Auch diese Teilchen wurden in eine Familie eingeordnet und Baryonen genannt.
1952 ging in Brookhaven (USA) der erste moderne Teilchenbeschleuniger in Betrieb, so dass sich die Anzahl solcher "im Labor künstlich" erzeugter Teilchen stark vergrößerte. Diese Teilchen bezeichnete man auch als Hadronen (Teilchen, die stark wechselwirken).
Viele der gefundenen Hadronen hatten eine "seltsame" Eigenschaft. Sie konnten leicht erzeugt werden (in der sehr kurzen Zeit von 10^{-23}s), zerfielen aber relativ langsam (in "langen" 10^{-10}s). Diese "Seltsamkeit" (engl. Strangeness) beruht darauf - wie wir heute wissen -, dass für ihre Produktion die starke, für ihren Zerfall aber die schwache Wechselwirkung verantwortlich ist.
Nach der Euphorie über die Entdeckungen neuer Teilchen in den 30er bis 50er Jahren folgte nun

[1] griechisch hadros=schwer

die Ernüchterung, einem regelrechten "Teilchenzoo" gegenüberzustehen, dessen Zusammensetzung mit steigender Komplexität immer undurchsichtiger wurde.
Abhilfe sollten Modelle zur Einteilung und Beschreibung dieser großen Anzahl von Teilchen schaffen.

2.1.1 Das Quarkmodell - Teilchenspektrum

Im Jahre 1961 konzipierte Murray Gell-Mann eine Ordnungsstruktur, den *achtfachen Weg*. In diesem wurden Mesonen und Baryonen aufgrund bestimmter Quantenzahlen (elektr. Ladung Q, Leptonenzahl L, Seltsamkeit oder Strangeness S) unterschieden und in regelmäßige geometrische Figuren wie z.B. Sechsecke angeordnet.
Drei Jahre später führten Gell-Mann und Zweig noch kleinere, elementare Bausteine - die **Quarks** - ein. Die Quarks kamen bei Gell-Mann in drei Arten[2] vor. Es gab das up-, das down- und das strange-Quark. Diese drei Quarks (und drei Antiquarks) genügten zur Erklärung der damals entdeckten Hadronen. Heutzutage sind sechs Quarks bekannt:

Quarks

	Teilchen	Ladung [e]	Farbladung
1. Generation	up (u)	2/3	rot, grün, blau
	down (d)	-1/3	rot, grün, blau
2. Generation	charm (c)	2/3	rot, grün, blau
	strange (s)	-1/3	rot, grün, blau
3. Generation	top (t)	2/3	rot, grün, blau
	bottom (b)	-1/3	rot, grün, blau

Tabelle 2.1: Eigenschaften der heute bekannten Quarks

Im Konstituentenquarkmodell sind Baryonen Kombinationen aus drei Quarks qqq (Antibaryonen aus drei Antiquarks $\bar{q}\bar{q}\bar{q}$). So ist z.B. das Proton eine Kombination aus zwei up-Quarks und einem down-Quark. Mesonen sind aus einem Quark und einem Antiquark, $q\bar{q}$, aufgebaut. Da das Antiteilchen eines Antiquarks ein Quark ist, sind auch Antimesonen Kombinationen aus einem Quark und einem Antiquark. Das Kaon (K^+) ist z.B. eine Kombination aus einem up-Quark und einem strange-Antiquark ($u\bar{s}$).

Zur Vereinfachung werden im weiteren Verlauf nur die drei leichtesten Quarks u,d und s, die im Energiebereich des CB-ELSA-Experimentes eine Rolle spielen, berücksichtigt.

[2] sog. "Geschmacksrichtungen", engl. "flavors"

Betrachtet man den Aufbau eines Baryons, so besteht es aus drei Quarks. Mit Hilfe der Flavor u, d und s ist es möglich, jedem Teilchen eine Quark-Kombination zu zuordnen. Nimmt man weiter an, dass die Quarks Fermionen sind und einen Spin von 1/2 tragen, können die Spins im energetisch niedrigsten Zustand (keine radiale Anregung und kein relativer Bahndrehimpuls) zu zwei möglichen Drehimpulsen koppeln.

Es ergibt sich eine Zusammenstellung aus zwei Quarks mit einem parallelem und einem antiparallelen Spin. Diese koppeln zu $J^P = 1/2^+$. Die andere Möglichkeit, wobei alle Quarks parallelen Spin besitzen, koppelt zu $J^P = 3/2^+$ (siehe Abbildung 2.1).

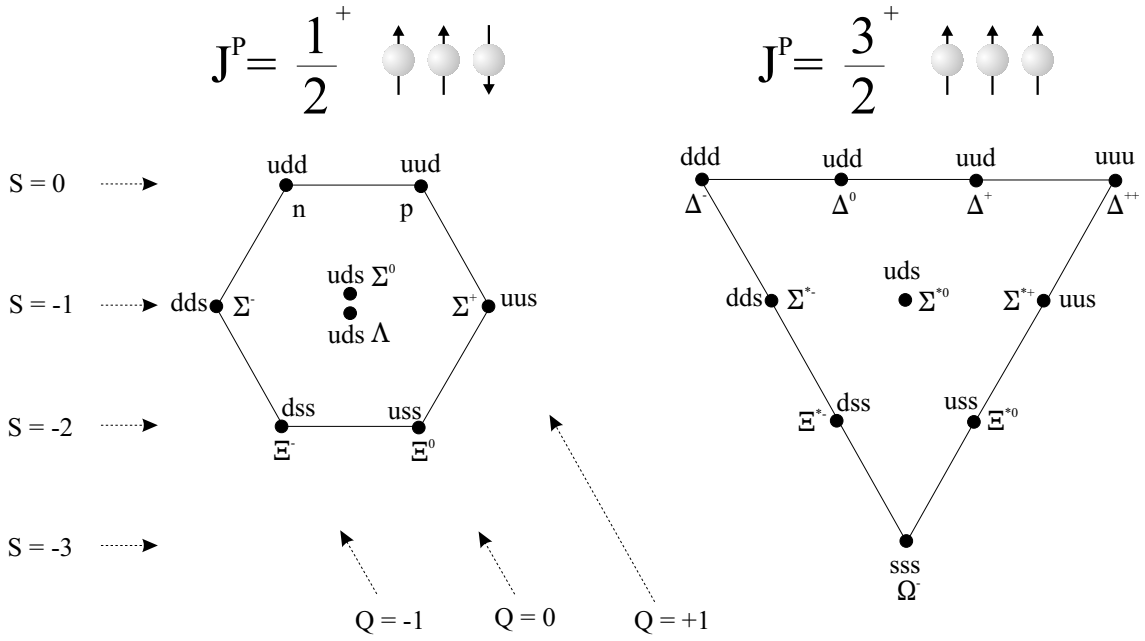


Abbildung 2.1: Oktett und Dekuplett der leichtesten Baryonen

Dieses Konstruktionsprinzip liefert eine hervorragende Übereinstimmung mit den bisher gefundenen Teilchen. Jedoch handelt man sich ein fundamentales Problem ein. Die Gesamtwellenfunktion solcher Zustände lässt sich faktorisieren in Wellenfunktionen mit dem jeweiligen Freiheitsgrad:

$$\psi_{qqq} = \psi_{Ort} \cdot \psi_{Flavor} \cdot \psi_{Spin}$$

Betrachtet man z.B. die Wellenfunktion des Zustands $\Delta^{++} = |uuu\rangle = \psi_{uuu}$, so ergibt sich, dass die Wellenfunktion unter Vertauschung zweier Quarks völlig symmetrisch ist. Da es sich um

einen Zustand ohne Radialanregung und relativen Bahndrehimpuls handelt, ist ψ_{Ort} symmetrisch. ψ_{Flavor} und ψ_{Spin} sind ebenfalls symmetrisch, da der Flavorinhalt gleich ist, bzw. alle Spins gleich ausgerichtet sind. Der Zustand ψ_{uuu} ist somit komplett symmetrisch. Das Pauli-Prinzip fordert aber für Fermionen, dass die Wellenfunktion unter Vertauschung zweier Quarks total antisymmetrisch sein muss.
Dieses fundamentale Problem kann gelöst werden durch Einführung eines weiteren Freiheitsgrades, der **Farbe**. Man gibt den Quarks unterschiedliche Farben. Im Baryon erhalten die Quarks die Farben rot, grün und blau. Die Mesonen sind aus Farbe und Antifarbe aufgebaut. In Anlehnung an die Farbenlehre ergibt sich somit für Mesonen und Baryonen immer die Farbe weiß. Gruppentheoretisch gesprochen liegen die Zustände in Farbsinguletts vor. Der Farbzustand ist nun total antisymmetrisch und läßt sich folgendermaßen schreiben:

$$\psi_{Farbe} = \frac{1}{\sqrt{6}} \sum_{\mathrm{k=r,g,b}} \sum_{\mathrm{l=r,g,b}} \sum_{\mathrm{m=r,g,b}} \epsilon_{\mathrm{klm}} |q_{\mathrm{k}} q_{\mathrm{l}} q_{\mathrm{m}}\rangle$$

Hiermit ergibt sich die Gesamtwellenfunktion zu

$$\psi_{qqq} = \psi_{\mathrm{Ort}} \cdot \psi_{\mathrm{Flavor}} \cdot \psi_{\mathrm{Spin}} \cdot \psi_{\mathrm{Farbe}}.$$

Da ψ_{Ort}, ψ_{Flavor} und ψ_{Spin} rein symmetrisch sind und ψ_{Farbe} rein antisymmetrisch ist, wird das Pauli Prinzip im Falle des Δ^{++} offensichtlich erfüllt. Treten nun gemischte Symmetrien auf (wie beim Δ^{+}), so muss der Flavor-Anteil entsprechend konstruiert sein, so dass die Gesamtwellenfunktion antisymmetrisch wird. Unter Verwendung der drei leichtesten Quarks (u, d, s) ergeben sich zehn mögliche Zustände, die auf der rechten Seite der Abbildung 2.1 im Dekuplett gezeigt sind.
Im Falle von $J^P = 1/2^+$ sieht die Situation etwas komplizierter aus, da der Spinanteil ψ_{Spin} eine gemischte Symmetrie aufweist. Zur Kompensation dieser gemischten Symmetrie muss der Flavor-Anteil ψ_{Flavor} ebenfalls eine gemischte Symmetrie aufweisen. Durch diese Vorgabe werden die rein symmetrischen Zustände $|uuu\rangle$, $|ddd\rangle$ und $|sss\rangle$ im Flavoranteil eliminiert (linkes Diagramm in Abbildung 2.1).
Diese einfache Erläuterung der beiden Multipletts lässt sich mit gruppentheoretischen Überlegungen auch quantitativ und mathematisch untermauern. Die zugrundeliegende Symmetrie der drei leichtesten Quark-Flavors ist die $\mathrm{SU(3)_{Flavor}}$. Zieht man alle Quark-Generationen in Betracht, erweitert sich die Symmetrie zur $\mathrm{SU(6)_{Flavor}}$; sie ist aber in der Natur extrem stark gebrochen.

2.1.2 Quantenchromodynamik - QCD

Durch die Einführung von Flavor und Farbe konnte eine Ordnung im Spektrum der entdeckten Teilchen erreicht werden.

Die Farbe spielt aber noch eine weitere wichtige Rolle. Sie wird als Ladung der starken Wechselwirkung interpretiert. Die Theorie dieser Wechselwirkung ist die weitestgehend anerkannte Theorie der Quantenchromodynamik (QCD).
Ähnlich der Quantenelektrodynamik (QED) basiert die QCD auf einer Eichtheorie, die eine lokale Eichinvarianz unter Orts- und Zeittransformationen der Wellenfunktion fordert. Dies führt zu einem Vektorfeld mit Spin 1, das an die Farbladung der Quarks koppelt. Diese Eichbosonen bezeichnet man als **Gluonen**. Der wesentliche Unterschied ist jedoch, dass es in der Quantenelektrodynamik nur einen Ladungszustand ($\pm e$) gibt, während in der QCD drei Ladungszustände (mit jeweils der Antiladung) existieren: $r/\bar{r}, g/\bar{g}$ und $b/\bar{b}$.

Die Gluonen sind ein Farb-Antifarb-Zustand und tragen im Gegensatz zur QED eine Ladung. Folglich können auch Gluonen untereinander wechselwirken. Aus dem Farb-Antifarb-Zustand ergeben sich neun mögliche Kombinationen, die in ein Singulett und ein Oktett eingeordnet werden können. Das Singulett ist ein Mischzustand aus allen Farben und Antifarben ($r\bar{r} + g\bar{g} + b\bar{b}$). Aufgrund seines Aufbaus ist dieser Zustand nicht in der Lage, einen Farbaustausch zwischen Quarks zu bewirken. Dieser Singulett-Zustand findet also in der QCD keine Anwendung, nur das Oktett wird benutzt. Ein weiterer Unterschied zur QED ist, dass außerhalb eines Baryons oder Mesons der Raum farbfrei ist, d.h. das Gluonenfeld ist nur im Inneren eines Teilchens präsent.

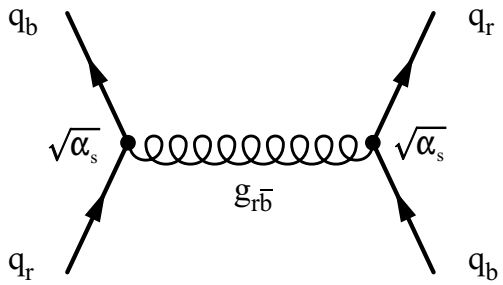


Abbildung 2.2: Gluon-Austausch bei der Quark-Quark-Wechselwirkung

Der Austausch eines Gluons ist in Abbildung 2.2 mit Hilfe eines Feynman-Diagramms zwischen zwei Quarks dargestellt. An den Vertices tritt bei der starken Wechselwirkung die Kopplungskonstante α_s auf. Sie ist das Analogon zur Feinstrukturkonstante α in der Elektrodynamik. Im Gegensatz zur Feinstrukturkonstante, welche eine feste Zahl ist, tritt bei α_s eine Abhängigkeit vom Impulsübertrag q^2 ein. Sie wird auch *running coupling constant* genannt. Die Abhängigkeit von q^2 lässt sich durch folgenden Ausdruck angeben:

$$\alpha_s(q^2) = \frac{12 \cdot \pi}{(33 - 2n_f) \cdot ln\frac{q^2}{\Lambda^2}}$$

wobei n_f die Anzahl der Flavors und Λ der Skalenparameter ist. Der Parameter Λ liegt im Bereich von 200 MeV, der einem Abstand von 1 fm entspricht.
Durch die Abhängigkeit vom Impulsübertrag q^2 lassen sich drei Bereiche im Bezug auf den Skalenparameter Λ charakterisieren:

$Q^2 >> \Lambda^2$ Hoher Energieübertrag

Bei großen Impulsüberträgen, die kleinen Abständen entsprechen, wird die Kopplungskonstante α_s klein, und es wird eine störungstheoretische Betrachtung möglich. Diese Region wird auch *pertubativer Bereich der QCD* oder als *asymptotische Freiheit* bezeichnet.

$Q^2 \approx \Lambda^2$ Mittlerer Energieübertrag, *Confinement*

In diesem Bereich ist die Kopplungskonstante nicht mehr vernachlässigbar, die QCD-Lagrange-Gleichung ist mittels pertubativer Methoden nicht lösbar. Hier befinden sich die Grundzustände des Protons oder Neutrons und auch angeregte Zustände dieser, aus denen die normale Materie aufgebaut ist. Weiterhin gibt es keine Erklärungsmöglichkeit für den Quarkeinschluss (*Confinement*).

$Q^2 << \Lambda^2$ Kleiner Energieübertrag

Bei kleinen Energien weist die QCD eine besondere Eigenschaft auf, die *chirale Symmetrie*. Sie entsteht, wenn man die Masse der leichtesten Quarks vernachlässigt; es wird eine störungstheoretische Betrachtung in Ordnungen der Masse möglich. In der chiralen Störungstheorie treten acht masselose Bosonen (Goldstone-Bosonen) auf. Gibt man den Quarks wieder eine Masse, wird die chirale Symmetrie spontan gebrochen, und in der Theorie entstehen somit ein $q\bar{q}$-Kondensat und die Konstituentenquarks.

2.1.3 Aktuelle Quarkmodelle

In dem mittleren Energiebereich, der für das CB-ELSA-Experiment interessant ist, schlagen sowohl Methoden der perturbativen QCD und der chiralen pertubativen Störungstheorie fehl. Um nun Vorhersagen über Eigenschaften von hadronischen Zuständen in diesem Bereich machen zu können, wurden Quarkmodelle entwickelt.
Die Baryonen werden in diesen Modellen aus drei Konstituentenquarks aufgebaut. Als Basis dieser Modelle dient die Annahme, dass sich diese Quarks in einem Confinement-Potential bewegen, welches über einen Farbaustausch zwischen den Quarks erzeugt wird. Das hiermit verbundene Potential V_{conf} steigt für große Abstände linear mit dem Abstand zwischen zwei Quarks an und soll dem Confinement der Quarks Rechnung tragen, da Quarks isoliert nicht beobachtet worden sind. Es besitzt folgende Form:

$$V = V_0 + a \cdot r \text{ mit einer String-Konstanten } a = 0,2 \text{ GeV}^2.$$

Dies beschreibt aber die Wechselwirkung noch nicht vollständig. Es sind weitere Restwechselwirkungen notwendig, die in den Modellen unterschiedlichen Ursprungs sind.

Ein-Gluon-Austausch

Das relativisierte Quarkmodell von Capstick und Isgur [7] ist eine Erweiterung des Modells für Mesonen von Godfrey und Isgur [6] auf die Beschreibung von Baryonen . Hierbei wird die Schrödinger-Gleichung im Hilbertraum gelöst, wobei die leichten Quarks (u, d) eine Masse von 220 MeV und das s-Quark eine Masse von 420 MeV besitzen. Die Hamiltonfunktion ist gegeben durch

$$H = \sum_i \sqrt{\mathbf{p}_i^2 + m_i^2} + V$$

und ein Potential, welches im nicht relativistischen Grenzfall folgende Form annimmt:

$$lim_{p_i/m_i \to 0} V = \underbrace{V_{\text{string}} + V_{\text{Coul}}}_{=V_{\text{i}}} + V_{\text{Hyp}} + V_{\text{SB}} + V_{\text{TP}}$$

Dieses Potential besteht aus dem genanntem Confinement-Potential und weiteren Restwechselwirkungen. Diese sind ein coulombähnliches spinunabhängiges Potential V_{Coul} bei kleinen Abständen, das durch den Austausch eines Gluons gegeben ist.

$$V_{\text{Coul}} \sim \sum_{i<j} -\frac{4}{3}\frac{\alpha_s}{r_{ij}}$$

Dies stellt lediglich den spinunabhängige Teil V_i des Potentials dar. Es können aber wie beim Potential im Wasserstoffatom weitere Wechselwirkungen eine Rolle spielen. Hierbei treten eine Feinstruktur (Spin-Bahn-Wechselwirkung) und eine Hyperfeinstruktur (Spin-Spin-Wechselwirkung) auf. Die Hyperfeinstruktur zwischen den Spins zweier Quarks hat folgende Form:

$$V_{\text{Hyp}} \sim \sum_{i<j} \frac{2\alpha_s}{3m_i m_j}\left(\frac{8\pi}{3}\mathbf{S_i}\cdot\mathbf{S_j}\ \delta^3(r_{ij}) + \frac{1}{r_{ij}^3}\left(\frac{3\mathbf{S_i}\cdot r_{ij} S_j \cdot r_{ij}}{r_{ij}^2} - \mathbf{S_i}\cdot\mathbf{S_j}\right)\right)$$

Ein weiteres spinabhängiges Potential betrachtet die Wechselwirkung zwischen Spin und Bahndrehimpuls V_{SB} (Hyperfeinstruktur) und ein zusätzliches Potential V_{TP}, das durch die Thomas-Präzession gebildet wird.

$$V_{\text{SB}} \sim \sum_{i<j} \frac{2\alpha_s}{3r_{ij}^3}\left(\frac{\mathbf{r_{ij}}\times\mathbf{p_i}\cdot\mathbf{S_i}}{m_i^2} - \frac{\mathbf{r_{ij}}\times\mathbf{p_j}\cdot\mathbf{S_j}}{m_j^2} - \left(\frac{\mathbf{r_{ij}}\times\mathbf{p_j}\cdot\mathbf{S_i} - \mathbf{r_{ij}}\times\mathbf{p_i}\cdot\mathbf{S_j}}{m_i m_j}\right)\right)$$

$$V_{\text{TP}} \sim -\sum_{i<j} \frac{1}{2r_{ij}}\frac{\partial V_i}{\partial r_{ij}}\left(\frac{\mathbf{r_{ij}}\times\mathbf{p_i}\cdot\mathbf{S_i}}{m_i^2} - \frac{\mathbf{r_{ij}}\times\mathbf{p_j}\cdot\mathbf{S_j}}{m_j^2}\right)$$

Die Beschreibung der $N-\Delta$-Aufspaltung im Grundzustand wird unter Hinzunahme von V_{Coul} und V_{Hyp} gut wiedergegeben. Jedoch wird diese gute Übereinstimmung durch das Hinzufügen

der Spin-Bahn-Wechselwirkung zerstört; sie steht somit im Widerspruch zu den gemessenen Daten. Es wird nun vermutet, dass die Thomas-Präzession die Spin-Bahn-Wechselwirkung zu Null kompensiert, da sie ein entgegengesetztes Vorzeichen besitzt.

Goldstone-Boson-Austausch

Im Modell von L.Z. Glozman et al. [8] wird die Restwechselwirkung zwischen den Quarks durch den Austausch von Goldstone-Bosonen ausgedrückt. Durch die spontane Brechung der chiralen Symmetrie treten acht Goldstone-Bosonen auf, welche direkt an die Konstituentenquarks koppeln.
Es wird folgende Hamiltonfunktion, wie auch beim Gluon-Austausch, zu Grunde gelegt:

$$H = \sum_i \sqrt{\mathbf{p}_i^2 + m_i^2} + V$$

Das Potential V setzt sich hierbei aus zwei Teilen zusammen: Zum einen aus dem Confinement-Potential V_{conf} und zum anderen aus weiteren Potential $V_\chi(\mathbf{r_{ij}})$, das durch den Austausch von Goldstone-Bosonen gebildet wird.

$$V = V_{\text{conf}} + V_\chi(\mathbf{r_{ij}})$$

Diese Goldstone-Bosonen werden mit dem Oktett und dem Singulet der pseudoskalaren Mesonen verbunden ($\pi,\ K,\ \eta,\ \eta'$), und mit Hilfe des Austauschs dieser Mesonen ergibt sich das folgende Potential:

$$V_\chi(\mathbf{r_{ij}}) = (\sum_{F=1}^{3} V_\pi(\mathbf{r_{ij}})\lambda_i^F\lambda_j^F + \sum_{F=4}^{7} V_K(\mathbf{r_{ij}})\lambda_i^F\lambda_j^F + V_\eta(\mathbf{r_{ij}})\lambda_i^8\lambda_j^8 + \frac{2}{3}V_{\eta'}(\mathbf{r_{ij}}))\sigma_\mathbf{i}\cdot\sigma_\mathbf{j}$$

Die Potentiale V_π, V_K, V_η und $V_{\eta'}$ bauen auf einem Yukawa-Potential auf:

$$V_\gamma(\mathbf{r_{ij}}) = \frac{g_\gamma^2}{4\pi}\frac{1}{12m_im_j}(\mu_\gamma^2\frac{e^{-\mu_\gamma r_{ij}}}{r_{ij}} - 4\pi\delta(\mathbf{r_{ij}}))$$

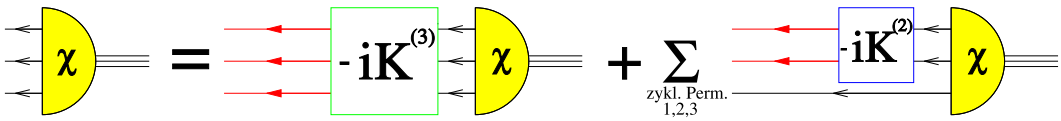
μ_γ ($\gamma = \pi,\ K,\ \eta,\ \eta'$) ist die phänomenologische Masse der Mesonen und $g_\gamma^2/4\pi$ die Kopplungskonstante zwischen Mesonen und Quarks.

Das Bonn-Modell; Instanton-induzierte Kräfte

Das Bonn-Modell [9], [10] baut mit Hilfe der Bethe-Salpeter-Gleichung mit der Restwechselwirkung durch Instantonen ein kovariantes Quarkmodell für leichte Baryonen auf. Die Bethe-Salpter-Gleichung für die Bethe-Salpter-Amplitude

$$\chi(x_1, x_2, x_3)_{\overline{P}} = \langle 0|T\Psi(x_1)\Psi(x_2)\Psi(x_3)|\overline{P}\rangle$$

lässt sich durch folgende grafische Darstellung veranschaulichen:



Hierbei ist die Wechselwirkung mittels sogenannter Wechselwirkungskerne durch folgende Symbole dargestellt:

$K^{(3)}$ beinhaltet die 3-Teilchen-Wechselwirkung und $K^{(2)}$ die 2-Teilchen-Wechselwirkung.

Zur Beschreibung der leichten Baryonen werden folgende Wechselwirkungen angenommen:

1. Ein phänomenologisches Confinementpotential

   Confinement wird ausschliesslich durch ein lineares Potential im Ruhesystem definiert und kann mittels Lorentztransformation in jedes System geboostet werden. Die Diracstrukur des Potentials wird so gewählt, dass die Spin-Bahn-Effekte klein werden.

2. Instanton induzierte 2-Quark-Wechselwirkung

   Die Restwechselwirkung zwischen zwei Konstituentenquarks besteht nicht aus einem Ein-Gluon-Austausch, sondern wird über die 't Hooftsche Kraft aus Instanton-Lösungen der klassischen QCD Yang-Mills-Gleichungen [11] bestimmt.

2.2 Resonanzspektrum der Baryonen

Berechnet man die Zustände des Anregungsspektrums in diesen Modellen, ergibt sich eine große Anzahl von Resonanzen.
In Abbildung 2.2 ist das Δ^*-Resonanzspektrum gezeigt. Im Vergleich sind die experimentell gefundenen Reonanzen und zwei Modelle dargestellt (blau: Bonn-Modell, grün: Modell nach Capstick, Roberts, Isgur).
Die Übereinstimmung zwischen den Vorhersagen und den experimentellen Daten ist bei niedrigen Anregungen recht gut. Es fällt aber auch auf, dass sich einige Diskrepanzen zwischen Theorie und Experiment ergeben. Es existieren mehr von der Theorie vorhergesagte Resonanzen als experimentell gefunden. Zum anderen ist ein großer Teil der gefundenen Resonanzen nicht gesichert. In Tabelle 2.2 sind die Anzahlen der experimentell gefundenen Resonanzen eingeteilt in N^*, Δ^*, Σ^*, Λ^*, Ξ^* und Ω^* zusammengefasst. Es wird deutlich, dass nur ca. 50% der Resonanzen (mit Rating $****$ und $***$) als experimentell gesichert gelten.

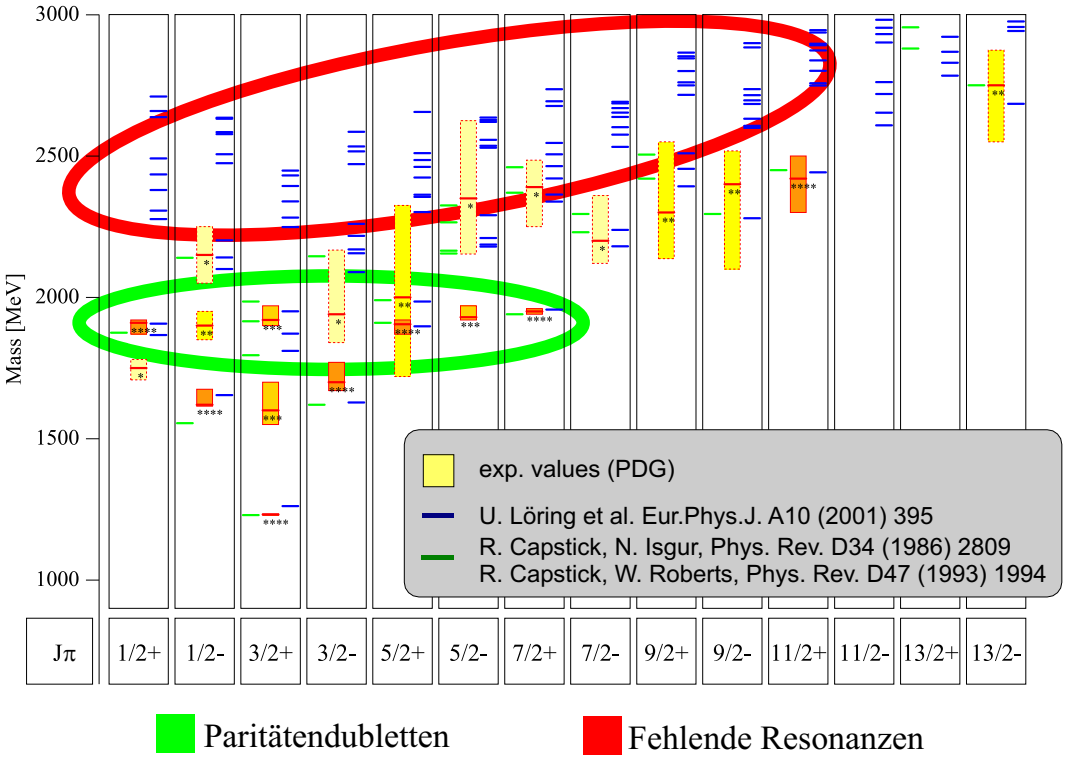


Abbildung 2.3: Übersicht der Δ^*-Resonanzen, sortiert nach Spin und Parität; Vergleich zwischen experimentellen Daten und Modellen

Rating	N^*	Δ^*	Σ^*	Λ^*	Ξ^*	Ω^*
* * **	11	7	6	9	2	1
* * *	3	3	4	5	4	1
Summe	14	10	10	14	6	2

Rating	N^*	Δ^*	Σ^*	Λ^*	Ξ^*	Ω^*
**	6	6	8	1	2	2
*	2	6	8	3	3	0
Summe	8	12	16	4	5	2

Gesamtzahl der Resonanzen: 103

Tabelle 2.2: Status der Resonanzen im Baryonenspektrum nach PDG

2.2.1 Fehlende Resonanzen

In Abbildung 2.2 sieht man deutlich, durch einen roten Kreis gekennzeichnet, eine erhebliche Anzahl von nicht gefundenen Zuständen im Spektrum. Für diese fehlenden Resonanzen bestehen verschiedene Erklärungsversuche.

Lichtenberg [4] schlug schon 1969 vor, dass die fehlenden Resonanzen durch ein Einfrieren von Freiheitsgraden erklärt werden können. Läge eine Quark-Diquark-Struktur im Baryon vor, so könnte die Anzahl der Zustände verringert werden.

Die Grundlage der Daten stammt zum großen Teil aus früheren πN-Streuexperimenten. Falls die fehlenden Resonanzen nur wenig oder gar nicht an πN koppeln [2], können diese auch

nicht experimentell gefunden werden. Prozesse, die keinen πN-Vertex enthalten, sind eine gute Möglichkeit, fehlende Resonanzen zu finden. Beispielrechnungen für den $\Delta\pi$-Kanal ergeben für viele fehlende Resonanzen eine bedeutsame Partialbreite. Untersuchungen in Daten des CB-ELSA-Experimentes im Zerfall $\gamma p \to p\pi^0\pi^0$ können Hinweise auf diese Resonanzen geben.

2.2.2 Paritätsdubletten

Ein weitere Auffälligkeit in Abbildung 2.2 sind Zustände mit gleichem Gesamtdrehimpuls J und entgegengesetzter Parität π. So weisen z.B. $\Delta^{\frac{1}{2}^+}(1910)$ und $\Delta^{\frac{1}{2}^-}(1900)$ jeweils eine ähnliche Masse auf. Solche Paritätsdubletten werden auch in anderen Spektren beobachtet (z.B. N^*, Λ^*).
Eine Erklärung für diesen Effekt ist von Cohen und Glozman [3] vorgeschlagen worden. Sie nehmen an, dass diese Dubletten eine Folge der Wiederherstellung der chiralen Symmetrie sind. Die chirale Symmetrie wird durch das QCD-Vakuum gebrochen, d.h. im Niedrigenergiebereich treten solche Dubletten nicht auf. Wandert man im Spektrum zu immer höheren Anregungsenergien, nehmen die Effekte der Symmetriebrechung immer mehr ab. Ab einem gewissen Punkt sind die Effekte so klein, dass sich diese Zustände in die Darstellung der chiralen $SU(2)_L \times SU(2)_R$-Symmetrie einordnen lassen und in Dubletten auftreten. Weiterhin sagt dieses Modell auch noch nicht gefundene Zustände voraus.

2.3 Exotische Zustände

Im Skyrme-Modell [5] werden die Eigenschaften von Hadronen nicht mittels Quarks und ihrer Wechselwirkungen ausgedrückt, sondern über Pionenfelder. Wird zur Pionenfeld-Gleichung ein nicht-linearer sogenannter σ-Term hinzugefügt, ergeben sich stabile Lösungen (Solitonen). Natürlich schließt das Skyrme-Modell nicht die Existenz von Quarks aus, aber es erlaubt eine andere Sicht des Baryons.
Übersetzt man die Ergebnisse in das Quarkbild, so sind Baryonen nicht nur aus drei Quarks aufgebaut, sondern es existieren zusätzlich Sea-Quarks. Aufgrund der starken Wechselwirkung polarisieren die Quarks den Dirac-See, die Kombination aus Quark und korrelliertem Sea-Quark ergibt ein Soliton (eine stabile Lösung der Pionenfeld-Gleichung). In dieser Konstellation erwartet man nicht nur Oktett- und Dekuplett-Zustände, sondern auch noch weitere Multipletts. Im Skyrme-Modell sind der Spin und der Isospin gekoppelt, $J = I$. Es werden Rotationsbanden erwartet. Bei Oktett-Baryonen ist $J = 1/2$ und $I = 1/2$, beim Dekuplett ist $J = 3/2$ und $I = 3/2$. Man erwartet nun ein Multiplett mit $J = 5/2$, aber Baryonen mit $I = 5/2$ sind bisher noch nicht beobachtet worden. Es ist allerdings zu erwarten, daß diese Resonanzen extrem breit sind.

Im chiralen Solitonen-Modell wird weiterhin die Existenz eines Antidekupletts, wie in Abbildung 2.4 gezeigt, vorhergesagt.

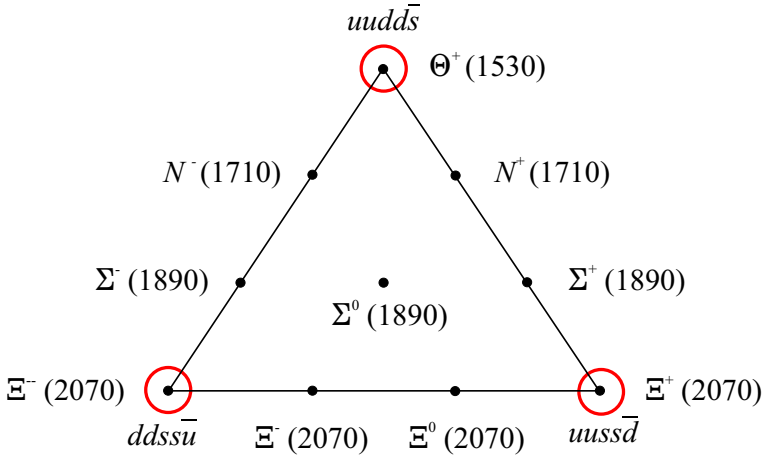


Abbildung 2.4: Exotische Zustände

Betrachtet man den Flavorinhalt der Quarkkombination des $\Theta^+ = uudd\bar{s}$, so ergibt sich ein Zustand aus fünf Quarks, ein Pentaquark. Im Jahre 2003 wurde ein Signal von Nakano et al. [12] im Kanal $\gamma n \rightarrow K^+K^-n$ entdeckt, welches mit dem Θ^+ assoziiert werden kann. Weitere Experimente wie das SAPHIR-Experiment, DIANA-Experiment, CLAS-Experiment, etc. haben ebenfalls ein Signal, das mit dem Θ^+ assoziiert werden kann, teilweise auch in anderen Reaktionen gesehen.

2.4 Anforderungen an die Datennahme

Offene Fragen im Bereich der Quarkmodelle im Baryonenspektrum und die Frage nach der Existens von exotischen Teilchen zeigen deutlich die Notwendigkeit nach neuen und mehr Daten in diesem Bereich. Weiterhin erfordert es auch eine möglichst hohe Statistik in den jeweiligen Kanälen, um die Bestimmung weiterer Resonanzparameter außer Breite und Masse, wie Winkelverteilungen oder Helizitätsamplituden, zu bestimmen.
Der totale Wirkungsquerschnitt der Reaktion $\gamma N \rightarrow NX$ ist in Abbildung 2.5 gezeigt. Der Wirkungsquerschnitt wird in der ersten Resonanzregion dominiert durch die Δ-Resonanz mit nahezu $500\mu b$. Bei höheren Energiebereichen sinkt der Querschnitt sogar jedoch auf ca. $150\mu b$ ab. Um in Kanälen mit exotischen Teilchen, welche in Bereichen unter 1 μb liegen können, genügend Statistik anzusammeln, müssen Experimente mit einer sehr hohen Photonenrate ($> 10^7$ Photonen/s) durchgeführt werden.
Die Ereignisse können mit einem sehr speziellen Selektionskriterium während der Datennahme herausgefiltert werden, so dass die Anforderungen an das Datennahmesystem klein ausfallen.

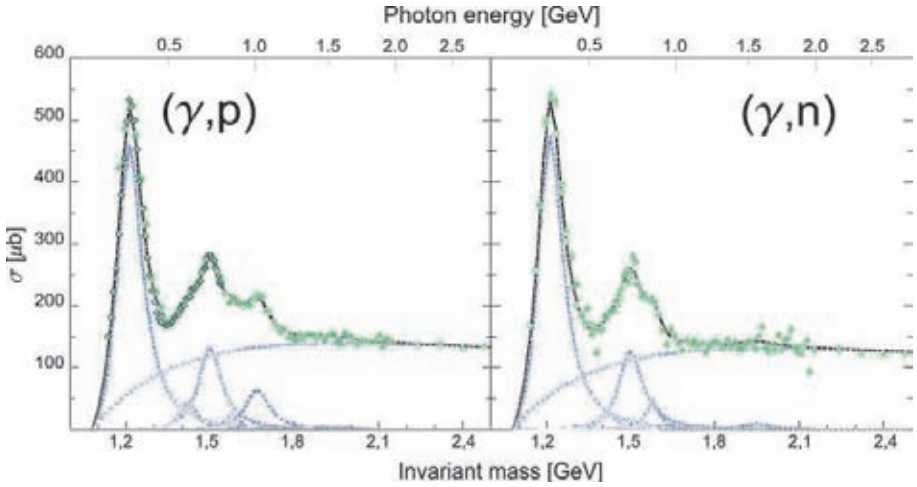


Abbildung 2.5: Totaler Wirkungsquerschnitt [13] für die Reaktion $\gamma p \rightarrow pX$

Der Nachteil hierbei ist aber, dass man unbekannte Reaktionen und andere interessante Ereignisse, die nicht auf das Selektionskriterium passen, verliert. Um diesem vorzubeugen, wählt man die Selektionskriterien so, dass nur Untergrundereignisse verworfen werden, die Selektion also nicht speziell auf ein gesuchtes Ereignis ausgerichtet ist. Dies erfordert aber für die Datennahme, dass sie hoch performant sein muss, um möglichst viele Ereignisse pro Sekunde aufzeichnen zu können. Jeder Faktor in der Steigerung der Verarbeitungsrate kann die Statistik in den gewünschten Ereignissen im gleichen Maße erhöhen. Für die eigentliche Statistik in den untersuchten Daten spielt auch die Rekonstruktionseffizienz eine große Rolle; diese stellt ebenfalls Anforderungen an den Detektor.

Kapitel 3

Das CB-ELSA-Experiment

Die ersten Messungen des Crystal-Barrel-Detektors begannen 1990 am LEAR des CERN mit der Untersuchung von $\bar{p}p$-Anhilationen. Hierbei lag der Schwerpunkt im wesentlichen auf der Mesonen-Spektroskopie. Durch die Stilllegung des LEAR 1997 wurde die erfolgreiche Zeit des Experimentes am CERN beendet.

1999 begann der Wiederaufbau des Crystal-Barrel-Detektors am Elektronenspeicherring ELSA in Bonn. Die neue Aufgabe des Experimentes ist es, weitere Daten zu liefern, die Antworten auf offene Fragen in der Baryonenspektroskopie geben sollen. Dazu werden mit Hilfe des ELSA Beschleunigers hochenergetische Photonen (bis 3,2 GeV) erzeugt, die Protonen und Neutronen in angeregte Zustände versetzen. Die Signatur dieser angeregten Zustände wird über den Nachweis der neutralen Zerfallsprodukte im CB-ELSA-Experiment mit einer großen Raumwinkelüberdeckung analysiert.

Für diese Aufgabe wurde der Crystal-Barrel an ELSA wieder aufgebaut und um weitere Detektorkomponenten ergänzt. Auch das Datenakquisitionssystem, welches am CERN eingesetzt wurde, ist wieder aufgebaut und für den Einsatz an ELSA modifiziert [19] worden; im Zeitraum von 2000 bis 2001 sind damit erfolgreich Daten aufgezeichnet worden. Das Experiment wurde im Jahre 2001 um den Detektor der TAPS-Kollaboration erweitert. Dieser Aufbau mit seinen Unterkomponenten soll in den nächsten Abschnitten kurz vorgestellt und erläutert werden.

3.1 Die Elektronen-Stretcher-Anlage ELSA

Die Elektronen-Stretcher-Anlage (kurz ELSA) ist ein Beschleuniger, der einen kontinuierlichen Elektronenstrahl mit einer Energie von bis zu 3,5 GeV liefern kann.

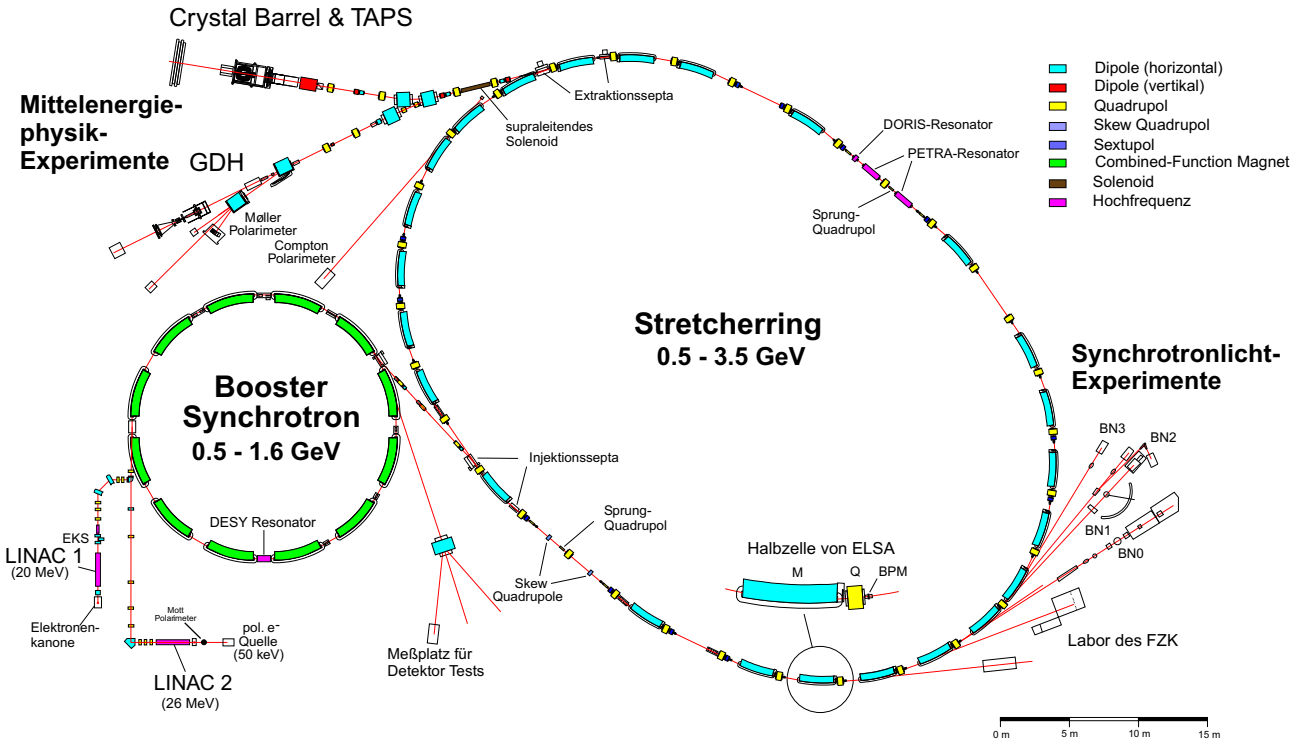


Abbildung 3.1: Bonner Elektronen–Stretcher–Anlage.

Die Anlage besteht aus drei Komponenten (siehe Abbildung 3.1):

- Zwei ca. 20 MeV Linearbeschleunigern

  LINAC 1 für unpolarisierte Elektronen; LINAC 2 für polarisierte Elektronen

- Eim gepulsten 2,5 GeV Synchrotron

- Eim Stretcherring mit einem Energiebereich von 0,5 - 3,5 GeV

ELSA verfügt über 2 Elektronenquellen. Die unpolarisierte Elektronenquelle speist den LINAC 1 und die polarisierte Quelle den LINAC 2. Beide LINACs beschleunigen die Elektronen auf eine Energie von ca. 20 MeV. In der nächsten Stufe wird mit Hilfe des Booster-Synchrotron eine weitere Beschleunigung durchgeführt. Bei einer Energie von ca. 1,6 GeV werden diese Elektronen in den Stretcherring ELSA (siehe Abbildung 3.1) injiziert. Bei Strahlenergien von weniger als 1,6 GeV wird ELSA nur im Stretcher-Modus betrieben, d.h. in diesem Modus werden die Strahlpakete des Synchrotrons nicht nachbeschleunigt, sondern es erfolgt im Stretcherring ein Auseinanderziehen der Elektronenpakete, so dass bei der Extraktion ein kontinuierlicher Strahl entsteht. Bei Energien oberhalb von 1,6 GeV wird erst eine gleichmäßige Füllung in ELSA akkumuliert. Nach Erreichen der gewünschten Füllmenge wird der Strahl mittels Einspeisung von Hochfrequenzenergie in die Resonatoren bei gleichzeitiger Erhöhung der magnetischen

Führungsfelder auf die gewünschte Endenergie nachbeschleunigt. Anschließend kann der Strahl durch Anregung von Betatronschwingungen langsam extrahiert werden. Während der Füll- und Rampphase kann kein Strahl im Experimentbereich zur Verfügung gestellt werden. Je nach Zykluszeiten (Füll-, Ramp- und Extraktionsphase) und Extraktionsintensität kann ein Tastverhältnis[1] zwischen 70% und 90% erreicht werden.

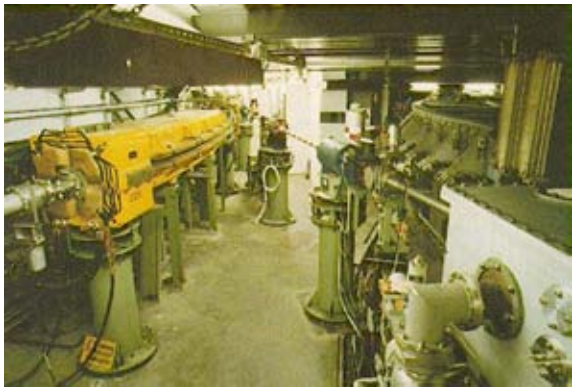


Abbildung 3.2: Transfer von Sync. nach ELSA

Die typischen Extraktionsströme liegen bei mehreren nA und erlauben somit Photonenintensitäten von bis zu 10^8 Photonen pro Sekunde in einem Energiebereich von 0,4-3 GeV. Soll eine Photonenrate (I_{Photon}) von 40 MHz mit Hilfe eines Radiatortarget und einer Strahlungslänge von ca. 1/1000 erreicht werden, so ergibt sich für die Anzahl der primären Elektronen (I_{e}):

$$I_{\text{Photon}} = 4 \cdot 10^7 \ s^{-1}$$

$$I_{\text{e}} = I_{\text{Photon}} \cdot 1000 \cdot e = 3,2 \cdot 10^{-9}\text{A} = 3 \text{ nA}$$

3.2 Das CB-ELSA-Experiment

Die Komponenten des Experimentaufbaus sind in Abbildung 3.3 gezeigt (von links nach rechts):

- Ein Radiatortarget zur Erzeugung von Bremsstrahlphotonen
- Ein Tagging–System zur Bestimmung der Photonenenergien über die Impulsbestimmung der zugehörigen Bremsstrahlelektronen
- Ein Targetsystem, das mit Flüssigwasserstoff oder Deuterium gefüllt werden kann
- Ein Innen-Detektor zur Identifizierung von geladenen Teilchen
- Der Crystal-Barrel zur Detektion von Photonen und zur Bestimmung ihrer Energie
- Der TAPS-Detektor zur Energiebestimmung von Zerfallsphotonen in Vorwärtsrichtung
- Eine Flugzeitwand (ToF) zur Identifikation von Protonen und Neutronen
- Ein Photonenintensitätsmonitor zur Bestimmung des Photonenflusses

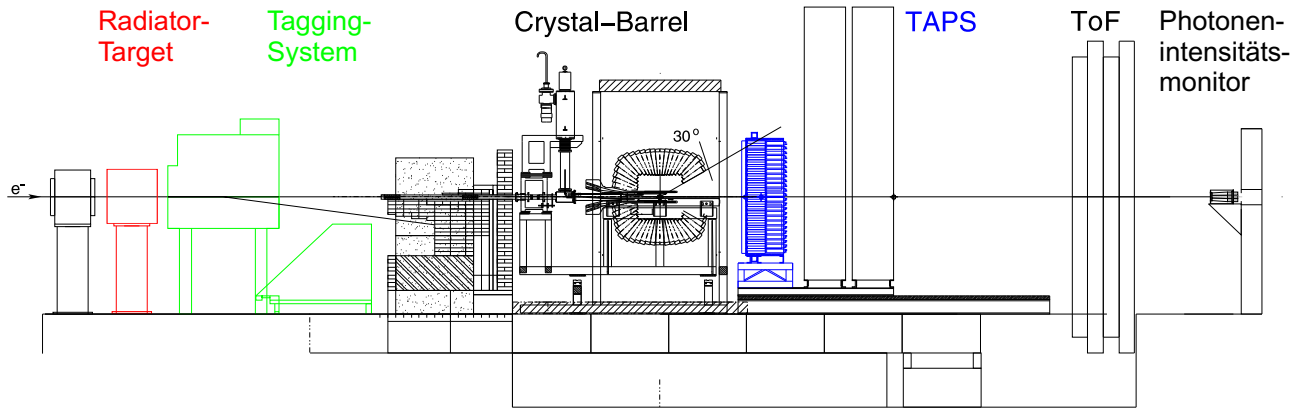


Abbildung 3.3: Experimentaufbau am ELSA-Strahlplatz

Der Elektronenstrahl wird in Abbildung 3.3 von links in das Experiment geführt. Ganz links ist der letzte Quadrupol-Magnet der ELSA-Strahlführung gezeigt. Die extrahierten Elektronen treffen nach dem Magneten auf ein Bremsstrahltarget (Radiator). Hinter dem Radiator werden die Elektronen mittels eines Dipol-Magneten aus der Horizontalen herausgelenkt. Elektronen, die keine Beeinflussung erfahren haben, werden unter einem Winkel von 7,5° in einen Strahlvernichter (Beam-Dump) geleitet. Die Sekundärteilchen (Neutronen, Photonen, etc.) aus der Vernichtung der Elektronen werden über eine Schicht aus Blei, Polyäthylen, Borkarbid und Eisen absorbiert, um den Untergrund in den großvolumigen Photonendetektoren hinter dem Beam-Dump zu minimieren. Die Elektronen, die ein Photon durch Bremsstrahlung erzeugt haben, werden durch das Tagging-System, welches unterhalb des Magneten positioniert ist, energiemarkiert. Der so erzeugte Photonenstrahl trifft im Zentrum des Crystal-Barrel-Detektor auf ein Flüssigwasserstofftarget. Die erzeugten Reaktionsprodukte werden durch den Innen-, den Crystal-Barrel-, den TAPS- und den ToF-Detektor nachgewiesen. Photonen ohne eine Reaktion im Target werden vom Photonenintensitätsdetektor erfasst.
Diese beteiligten Komponenten sollen im folgenden in der genannten Reihenfolge näher erläutert werden.

3.3 Das Radiatortarget

In den geplanten Experimenten wird mittels reeller Photonen das Anregungsspektrum der Baryonen untersucht. Reelle Photonen werden durch Abbremsen von Elektronen im Coloumbfeld eines Kerns über Bremsstrahlung erzeugt. Hierzu werden hochenergetische Elektronen aus ELSA (bis zu 3,2 GeV) auf ein Bremsstrahltarget geleitet. Die Targeteinrichtung verfügt über neun Aufnahmeplätze für verschiedene Bremsstrahltargets.

[1]Verhältnis Extraktionszeit zur Gesamtzeit eines Zyklus

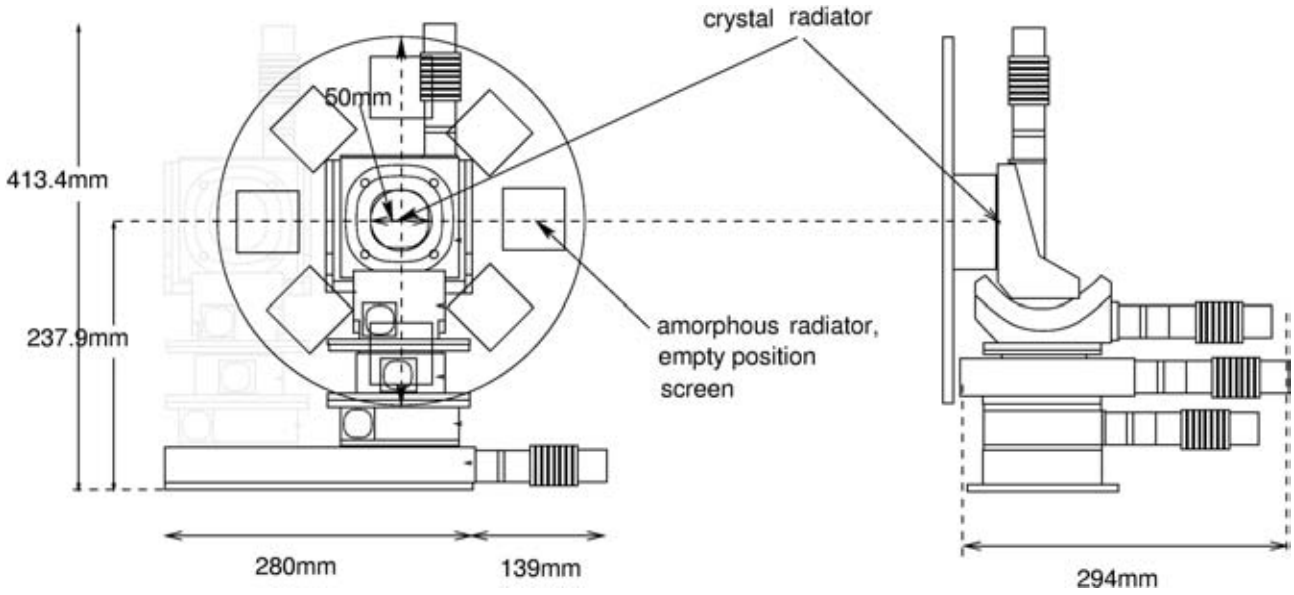


Abbildung 3.4: Frontal- und Seiten-Ansicht des Radiatortargets

Abbildung 3.4 zeigt die Anordnung der möglichen Plätze; hierbei sind acht Plätze im äußeren Teil der rotierbaren Scheibe für amorphe Radiatoren vorgesehen. Es stehen z.B. Targets mit 1/100 und 1/1000 Strahlungslänge zur Verfügung. Im zentralen Platz ist ein Diamantkristall zur Erzeugung von linear polarisierten Photonen durch Braggstreuung angebracht. Hier kann eine lineare Polarisation von bis zu 50% erreicht werden, wobei allerdings der Grad der Polarisation energieabhängig ist. Dieser Kristall kann über fünf Achsen relativ zum Strahl hochpräzise positioniert werden, um die notwendige Orientierung zum Strahl für die Braggstreuung einstellen zu können.

3.4 Das Tagging-System

Informationen über die Energie oder die Erzeugung eines Photons werden durch das Tagging-System (kurz Tagger) ermittelt. Durch den Abbremsprozess im Radiatortarget geben die Elektronen eine gewisse Energie an das Photon ab; sie haben somit weniger Energie als im Primärstrahl. Mit Hilfe eines Dipol-Magneten, welcher ein Feld von ca. $1,5$ Telsa erzeugt, werden die Elektronen aus der Strahlachse herausgelenkt, sodass nur die erzeugten Photonen das Target erreichen. Die Elektronen, die kein Photon über Bremsstrahlung erzeugt haben, werden unter einem festen Winkel von $7,5°$ in einen Beam-Dump geleitet. Die Bremsstrahlelektronen werden im Magnetfeld nach ihrem jeweiligen Impuls unterschiedlich stark abgelenkt und verlassen unter verschiedenen Winkeln zur Strahlachse das Magnetfeld. Aus dem Winkel der Ablenkung und der gemessenen Magnetfeldstärke kann der Impuls des Elektrons und somit aus der Erhaltung der Energie die Energie des Photons bestimmt werden. Die Energie des Photons ergibt sich aus

der Energiedifferenz zwischen der ELSA-Elektronenenergie und dem gemessenen Bremsstrahlelektronenimpuls.

$$E_\gamma = E_0 - E_e$$

E_0 ist die ELSA-Elektronenprimärenergie. E_e ergibt sich aus folgender Beziehung:

$$E_e = e\,B\,r\,c \quad \text{bei} \quad p\,c \gg m_e\,c^2.$$

Der Krümmungsradius r kann aus dem Ablenkwinkel, welcher über das Tagging-System bestimmt wird, ermittelt werden.

Abbildung 3.5 zeigt den Aufbau des Systems. Es besteht aus 14 rechteckigen Szintillationsdetektoren. Zusätzlich befinden sich oberhalb dieser Detektoren zwei Drahtkammern mit 144 Drähten in der oberen Kammer und 208 Drähten in der unteren Kammer mit einer Ortsauflösung von ca. 4 mm. Die Szintillationsdetektoren dienen zur Erzeugung eines schnellen Triggersignals

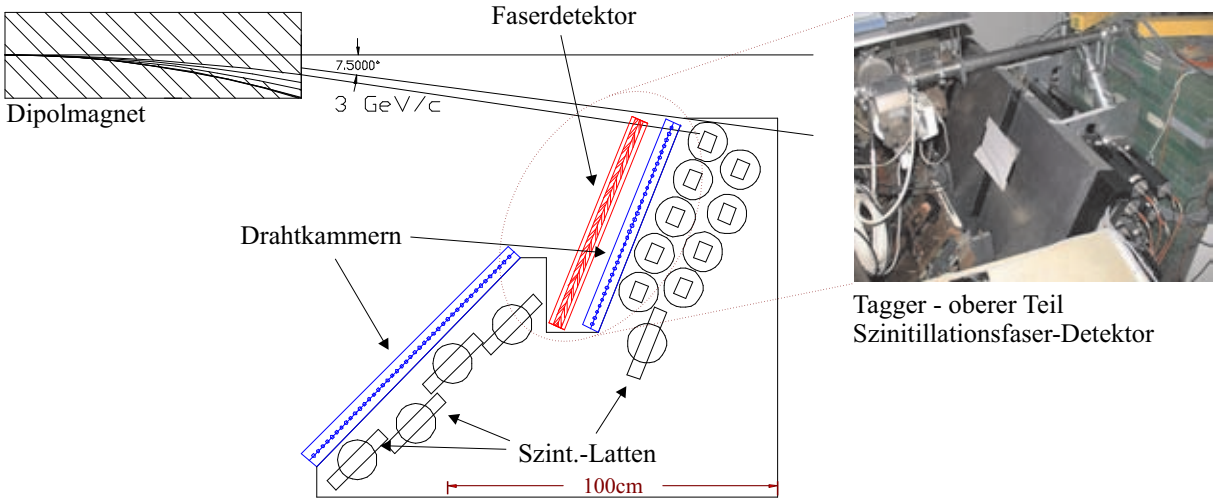


Abbildung 3.5: Schnittbild des Tagging-Systems und Bild des Szintillationsfaserdetektors

und zur Grobbestimmung des Ablenkwinkels. Eine feinere Rasterung des Ablenkwinkels wird mittels der Drahtkammern erreicht. Die Anordnung der Kammern und des Tagging-Systems relativ zum Dipolmagneten erlaubt es, Photonen im Energiebereich von 31,0 % bis 94,4 % der ELSA-Elektronenprimärenergie E_0 zu erfassen.
Betrachtet man die Verteilung der Photonenraten über die beiden Drahtkammern, ergibt sich ein Problem bei höheren Photonenproduktionsraten im Radiatortarget. Da die Anzahl der erzeugten Photonen nicht gleichverteilt über den Energiebereich ist, sondern Photonen mit niedrigen Energien wesentlich häufiger als Photonen mit hohen Energien erzeugt werden ($\sim 1/E$), wird die obere Drahtkammer wesentlich häufiger getroffen als die untere. Die Effizienz dieser

Kammer ist bis zu einer primären Photonenproduktionsrate von $\sim 10^6$ Photonen pro Sekunde zufriedenstellend. Bei dieser Rate wird die Proportionalkammer im oberen Teil mit bis zu 10^5 Treffern je Draht und Sekunde bei einer primären Elektronenenergie von 3,2 GeV belastet. Bei großen Raten setzen Effizienzverluste durch Raumladungseffekte in diesem Teil der Kammer ein.
Um die Effizienz des Tagging-Systems auch bei hohen Produktionsraten von $\geq 10^7$ Photonen pro Sekunde zu gewährleisten, wurde zusätzlich über der oberen Proportionaldrahtkammer ein zweilagiger Szintillationsfaser-Detektor aus 564 Fasern, die eine Dicke von 2 mm haben, installiert, der auch bei hohen Raten eine Effizienz von nahezu 99% aufweist.

3.5 Das Produktionstarget

Die am Radiator erzeugten Photonen treffen im Zentrum des Crystal-Barrel-Detektors auf ein Flüssigwasserstoff-Target [16] (siehe Abbildung 3.6), welches aus einem Zweikreiskühlsystem aufgebaut ist. Im Primärkreis wird verflüssigter Wasserstoff über ein 150 cm langes horizontales

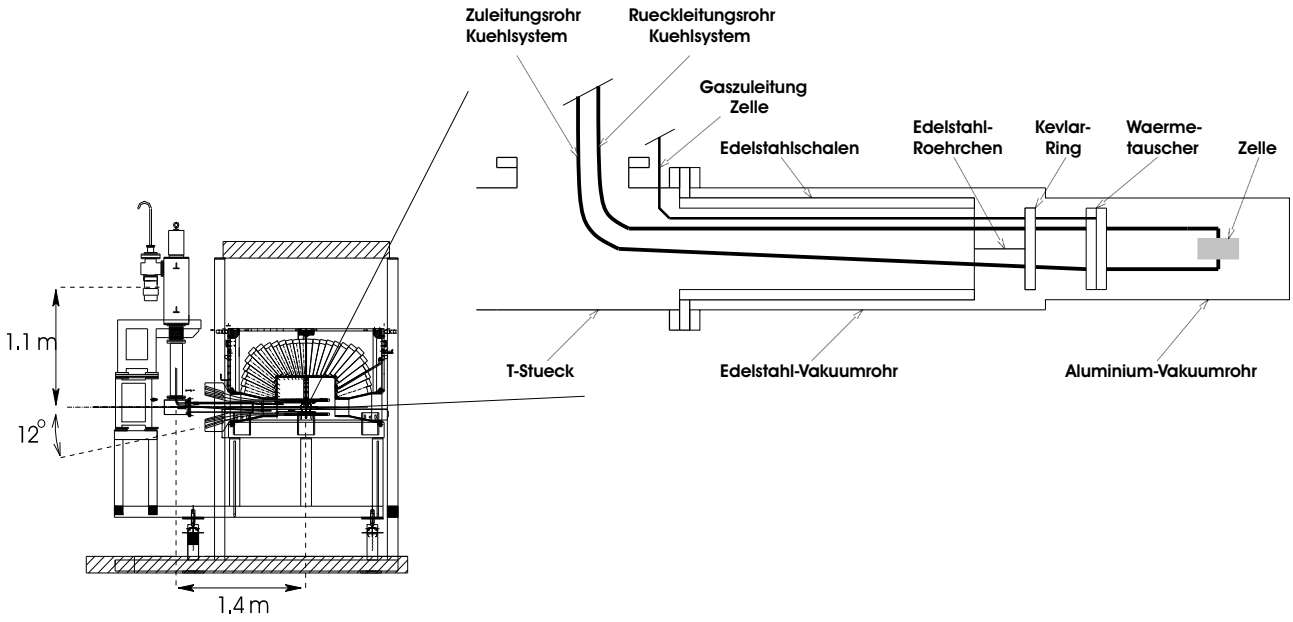


Abbildung 3.6: Aufbau des Flüssigwasserstoff-Target

Rohr zu einem Wärmetauscher geführt. Der Tauscher aus Kupfer befindet sich in unmittelbarer Nähe der Targetzelle. Im Sekundärkreis wird dem Wärmetauscher gasförmiger Wasserstoff zugeführt. Durch ein kurzes Kaptonröhrchen fließt der am Wärmetauscher kondensierte Wasserstoff in die Targetzelle. Die Zelle hat ein Fassungsvermögen von 35 cm^3 bei einem Durchmesser von 3 cm und einer Länge von 5 cm. Sowohl bei der Verlegung der Rohrleitungen für beide Kühlkreisläufe als auch bei der Positionierung des Wärmetauschers wurde darauf geachtet,

dass der Strahlungsuntergrund, der durch den Halo des primären Photonenstrahls an diesen Komponenten erzeugt wird, so klein wie möglich gehalten wird. Der Füllstand der Zelle kann eindeutig über den Druck des Sekundärsystems bestimmt werden, da es sich um ein abgeschlossenes System handelt. Somit kann der Füllstand der Zelle auch im geschlossenen Zustand des Crystal-Barrel-Detektors ständig überwacht und eine konstante Targetmassenbelegung garantiert werden. Ein weiterer Vorteil des Zweikreissystem ist, dass für Messungen an Neutronen als Targetmaterial lediglich der Sekundärkreis mit Deuterium gefüllt werden muss, nicht aber das Gesamtsystem aus Verflüssiger und Kältemittelreservoir.

3.6 Der Innen-Detektor

Der Crystal-Barrel-Detektor ist ein elektromagnetisches Kalorimeter, das auf den Nachweis von Photonen spezialisiert ist. Da aber auch geladene Teilchen, wie zum Beispiel $e^{\pm}$, $\pi^{\pm}$ oder Protonen, Energie nach der Bethe-Bloch-Gleichung im Kristall deponieren und diese nicht ohne weiteres von der Deposition neutraler Teilchen unterschieden werden kann, ist es notwendig, geladene Teilchen unabhängig zu identifizieren.

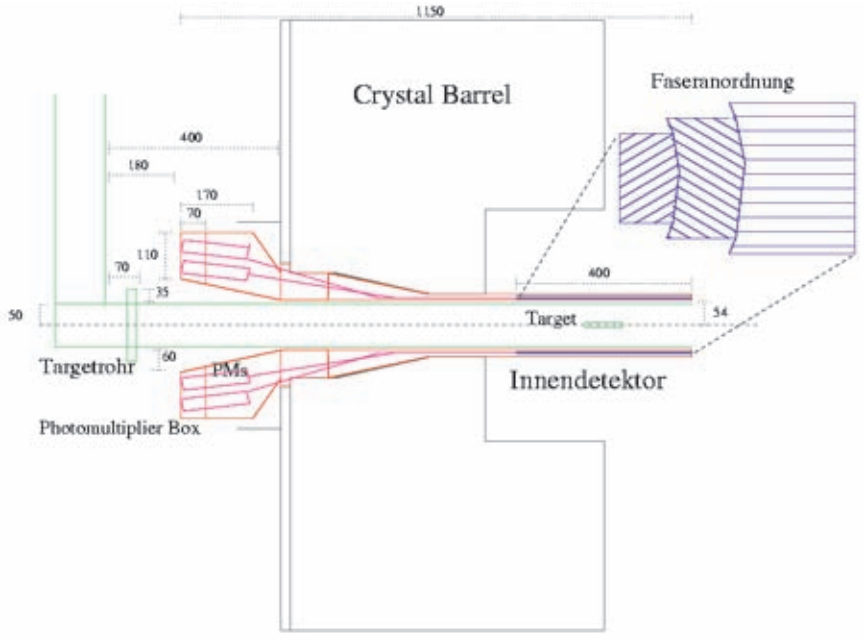


Abbildung 3.7: Schematische Darstellung des Innen-Detektors innerhalb des Crystal-Barrel

Mit Hilfe des Innen-Detektors ist es nun möglich, geladene Teilchen zu detektieren und ihren räumlichen Durchstoßpunkt durch den Innen-Detektor zu bestimmen. Unter Zuhilfenahme der Targetposition kann der Auftrittsort des geladenen Teilchens im Crystal-Barrel ermittelt werden. Somit lassen sich die Energiedepositionen, die von geladenen Teilchen stammen, im Crystal-Barrel-Detektor lokalisieren.

Zur Ortsbestimmung werden szintillierende Fasern eingesetzt, die in drei Lagen auf einer zylindrischen Oberfläche um das Target herum angeordnet sind. Die innere Lage besteht aus 157, die mittlere aus 167 und die äußere Lage aus 191 Fasern, die eine Dicke von 2 mm haben.
Die Ausrichtung der Fasern beträgt zur Strahlachse in der inneren Lage $+25°$, in der mittleren $-25°$ und in der äußeren ist sie parallel zur Achse (siehe Abbildung 3.6). Der Winkel ist so gewählt worden, dass die Fasern nur um 180° um den Zylinder gewickelt werden. Damit ergibt sich bei der Rekonstruktion des Durchstoßpunktes ein eindeutiger Punkt, wenn in jeder der drei Lagen des Detektors mindestens eine Faser angesprochen hat.

Abbildung 3.8: Innen-Detektor vor der Montage im Crystal-Barrel-Detektor

Die Lichtsignale der einzelnen Fasern werden über Lichtleiter aus dem Inneren des Crystal-Barrel-Detektors herausgeführt. Sie führen aus dem Crystal-Barrel strahlaufwärts zu je 16 Vielsegment-Photomultipliern.
In Abbildung 3.6 ist der Innen-Detektor vor dem Einbau in den Crystal-Barrel-Detektor gezeigt; zu sehen ist die Anschlussplatte für die Hochspannungen und die Photomultipliersignale. Detailliertere Informationen über diesen Detektor können der Dissertation von Angela Fösel [15] entnommen werden.

3.7 Der Crystal-Barrel-Detektor

Der Crystal-Barrel-Detektor [14] ist aufgebaut aus 1380 CsI(Tl)-Kristallen, die in einer fassähnlichen Anordnung um den Target-Mittelpunkt herum angeordnet sind. Sie dienen in erster Linie als elektromagnetisches Kalorimeter zum Nachweis von Photonen.
In Abbildung 3.9 sind die Einzelteile eines Kristallmoduls gezeigt. Der CsI(Tl)-Kristall ist zum Schutz vor Feuchtigkeit in eine 100 μm dicke Titanhülle eingepackt. Zusätzlich ist der Kristall von einem dünnen Kaptongehäuse umgeben, um die Module elektrisch gegeneinander zu isolieren. Am Ende des Kristalls befindet sich ein Wellenlängenschieber aus Plexiglas, der die Grundfläche des großen Kristallendes auf

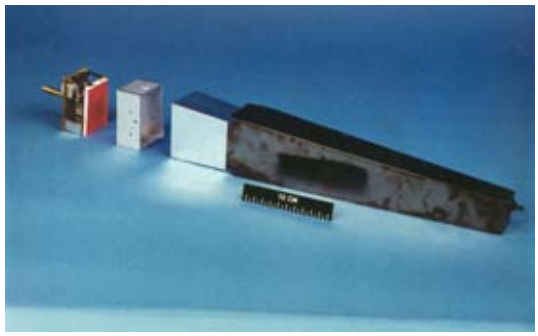


Abbildung 3.9: Einzelteile eines CsI(Tl)-Moduls

die kleine Fläche der Photodiode anpasst. Das abgestrahlte Licht der Kristalle hat ein Emissionsmaximum bei 550 nm und wird mit dem Wellenlängenschieber in den Wellenlängenbereich größerer Empfindlichkeit seitens der Photodiode hin verschoben. Die Photodiode wandelt das einfallende Licht in einen elektrischen Impuls um. Die Pulse am Ausgang des Vorverstärkers haben eine Anstiegszeit von 10-15 μs und eine Abfallzeit von ca. 100 μs. Die Pulshöhe beträgt etwa 1,5 V/GeV. Das elektronische Rauschen liegt im Bereich von $0,4$ mV, welches einer Energie von 250 keV entspricht. Dies erlaubt Messungen von Energiedepositionen bis zu 2 MeV.

Jede Hälfte des Crystal-Barrel besteht aus zehn Ringen mit je 60 Kristallen und einer Endkappe aus drei weiteren Ringen mit je 30 Kristallen. Hieraus ergibt sich, dass jeder Kristall einen Winkel von 6° in Polar- und 6° in Azimutrichtung abdeckt. In den Endkappen ist der Winkel in Azimutrichtung 12°, da hier nur die Hälfte der Kristalle verwendet wird. Der Detektor deckt somit im Laborsystem einen Raumwinkel von nahezu 95% von 4π ab.

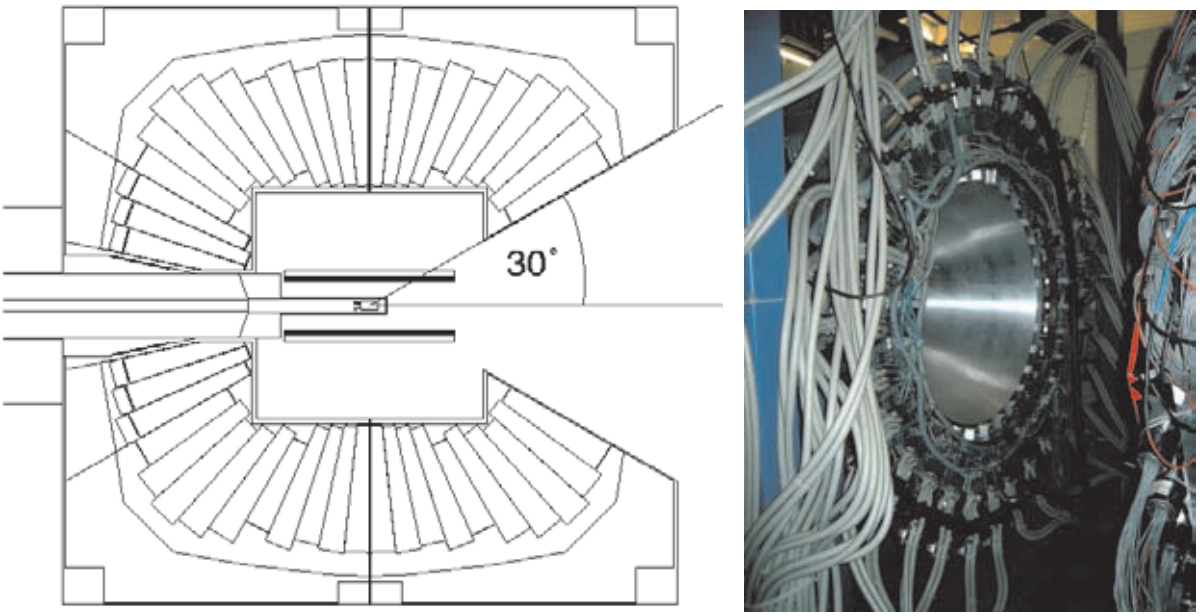


Abbildung 3.10: links: Modifizierte Geometrie des Crystal-Barrel-Detektors; rechts: Blick in die 30°-Öffnung am Strahlplatz

Für den Aufbau mit dem TAPS-Detektor wurde die Geometrie geändert, wie in Abbildung 3.10 gezeigt. Der ursprüngliche Detektor war in zwei spiegelsymmetrische Hälften aufgeteilt. Am ELSA-Strahlplatz aber werden aufgrund des Fixtarget-Experimentes die Zerfallsteilchen stark in Vorwärtsrichtung erzeugt (Lorentz-Boost). Somit stehen für ca. 70% der Ereignisse nur ca. 90 Kristalle der innersten drei Ringe zur Verfügung. Um diesen Bereich mit hochauflösenden Detektoren bestücken zu können, wurde eine Endkappe mit drei Ringen strahlabwärts entfernt und es entstand ein Öffnungswinkel von ±30°, der Platz für andere Vorwärtsdetektoren bietet.

3.8 Der TAPS-Detektor

Aufgrund des Fixtargetexperimentes und dem daraus folgenden Vorwärtsboost der entstehenden Reaktionsprodukte ist es sinnvoll, in Vorwärtsrichtung Detektoren mit ähnlicher Winkelauflösung wie dem zentralen Crystal-Barrel-Detektor zu platzieren.

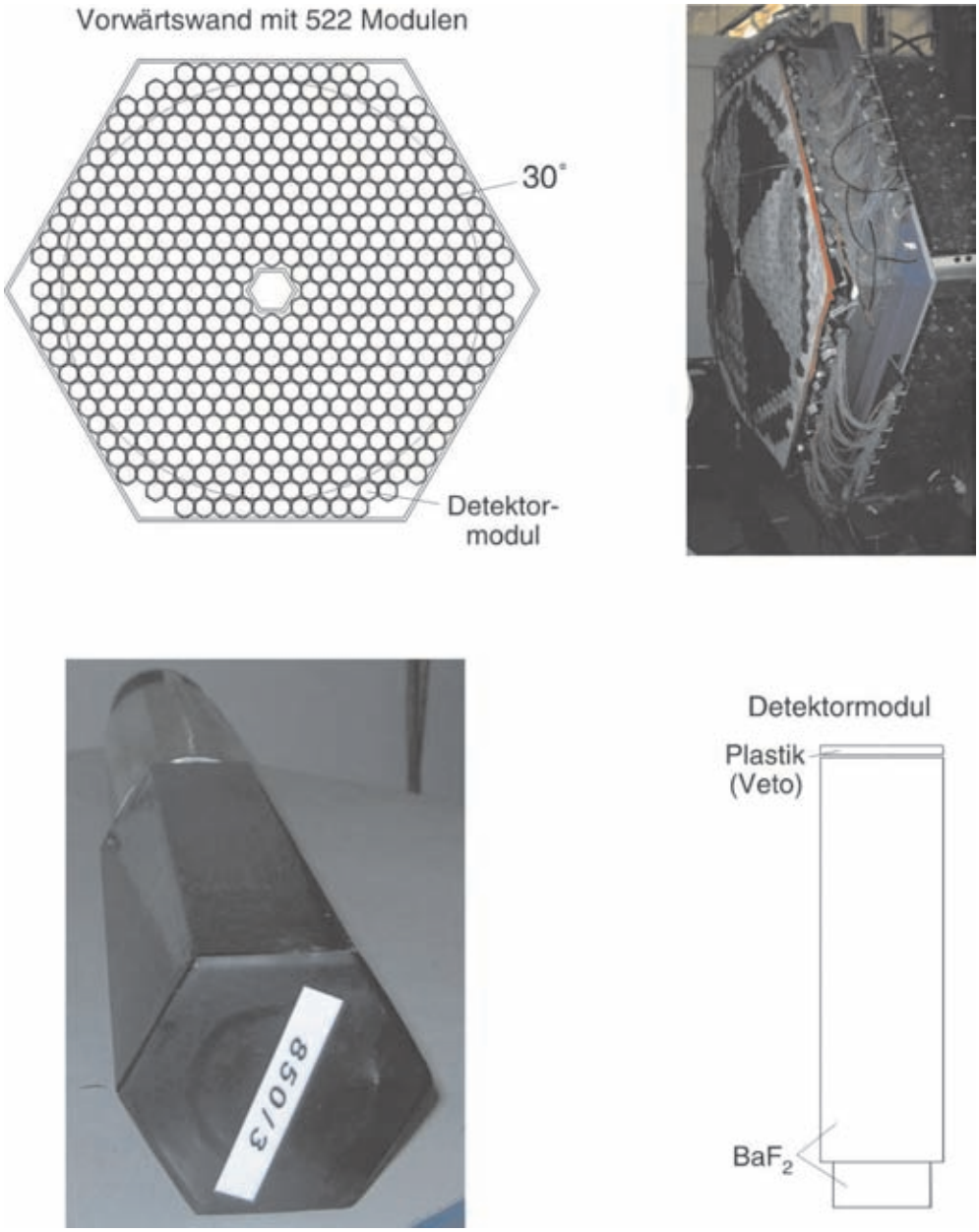


Abbildung 3.11: (oben) TAPS-Detektor; (unten) TAPS-Modul

Ein Detektor, der in den Strahlzeiten der Jahre 2001 bis 2003 zusammen mit dem Crystal-Barrel installiert war, ist das TAPS[2]-Spektrometer der Universität Giessen [18]. Es handelt sich um ein modulares System, das aus mehreren hexagonalen Bariumfluorid-Kristallen zusammengesetzt ist (siehe Abbildung 3.8). Jedes Modul besteht aus einem BaF_2-Kristall mit einer Länge von 25 cm; dies entspricht 12 Strahlungslängen für Photonen.
Vor jedem BaF_2-Kristall ist zusätzlich ein dünner Plastikszintillator angebracht, der eine Trennung von neutralen und geladenen Teilchen erlaubt. Der TAPS-Detektor ist daher in hervorragender Weise dazu geeignet, Photonen oder Protonen, die bei der Reaktion in Vorwärtsrichtung emittiert werden, nachzuweisen; er stellt somit eine ideale Erweiterung zum Crystal-Barrel-Detektor dar. Die Lichtsammlungszeit der BaF_2-Kristalle ist um eine Größenordnung schneller im Vergleich zu den CsI(Tl)-Kristallen des Crystal-Barrel-Detektors, sodass eine deutlich höhere Ratenfestigkeit mit diesem Detektortyp erreicht wird. Der im TAPS verwendete Lichtnachweis durch Photomultiplier ermöglicht zusätzlich die Erzeugung schneller Triggersignale, die zur Eventselektion in der ersten Triggerstufe verwendet werden können.

3.9 Die Flugzeitwand - ToF

Das Flugzeitspektrometer [17] (ToF), das in Abbildung 3.12 dargestellt ist, besteht aus vier Wänden mit jeweils 14 Szintillationsdetektoren. Die durch diese Wände abgedeckte Fläche beträgt 3 x 3 m^2 . Die Szintillatoren haben eine Dicke von 5 cm und werden an beiden Enden von Photomultipliern ausgelesen. Damit ist es möglich, den Durchtrittsort eines geladenen Teilchens

[2]**T**wo **A**rmed **P**hoton **S**pectrometer

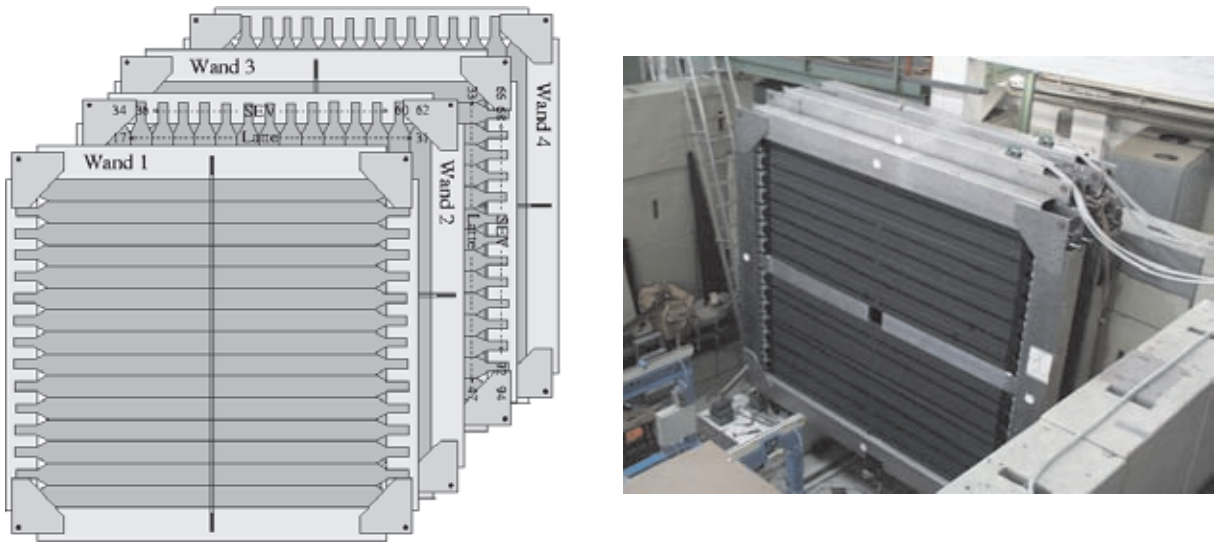


Abbildung 3.12: Aufbau des Flugzeitspektrometers (links); Installation im Experiment (rechts)

durch einen Szintillationsdetektor über die Bestimmung der Zeitdifferenz mit einer Auflösung von weniger als 5 cm zu bestimmen. Die Flugzeit wird aus der Summe beider Zeitmessungen bestimmt, wobei die Zeitauflösung besser als 300 ps ist. Die Szintillatorlatten der vier Wände sind jeweils um 90° gegeneinander verdreht. Auf diese Weise ist eine Bestimmung des Durchstoßpunktes in x- und y-Richtung möglich. In Verbindung mit einem geeigneten Startzeitpunkt aus den anderen Subdetektoren (TAPS, Innen-Detektor) kann dieser Detektor zur Identifikation langsamer Protonen und Neutronen herangezogen werden.

3.10 Der Photonenintensitätsmonitor

Hinter dem Flugzeitspektrometer befindet sich im primären Photonenstrahl ein Detektor [23] zur Messung der Intensität der Photonen, die die Targetzelle durchtreten haben. Aufgrund von Kollimatoren im Strahlrohr zwischen Tagging-System und Targetzelle können manche energiemarkierte Photonen die Targetzelle nicht erreichen; somit registriert der Detektor den Fluss der Photonen durch das Target zum Photonenintensitätsmonitor.
Da der Detektor sich im primären Photonenstrahl befindet, muss eine hohe Nachweiseffizienz auch bei extremen Photonenraten vorhanden sein. Weiterhin ist auch eine sehr gute Ratenfestigkeit erforderlich, damit die Nachweiseffizienz möglichst konstant über die Betriebszeit des Detektors ist.

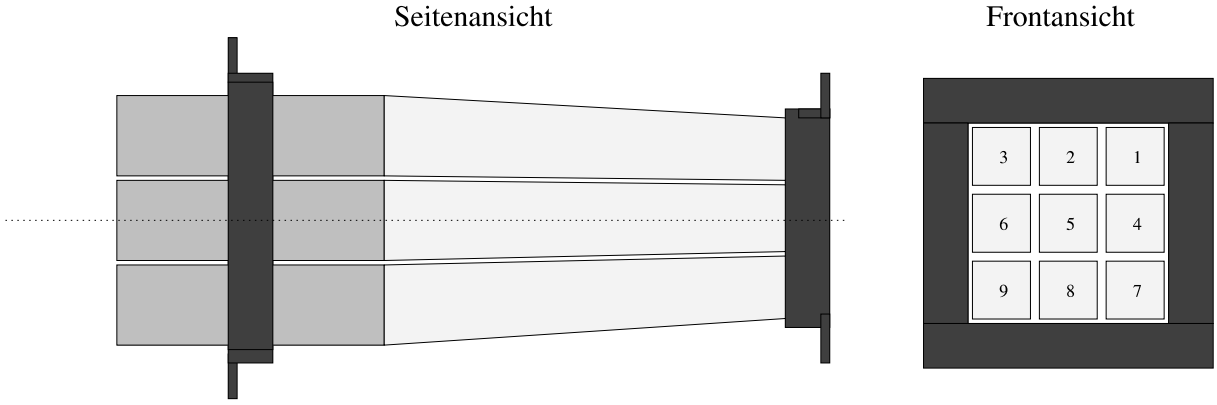


Abbildung 3.13: Aufbau des Photonenintensitätsmonitors

Der Detektor (siehe Abbildung 3.13) besteht aus neun Bleifluorid-Kristallen, die in einer 3×3-Matrix angeordnet sind. Bei jedem Modul (siehe Abbildung 3.14) wird über einen Photomultiplier der Lichtnachweis am hinteren Ende des PbF_2-Kristalls erreicht.
Eine besondere Eigenschaft von PbF_2 ist, dass das Licht nur durch den Čerenkov-Effekt erzeugt wird. Mit dieser Kombination aus Photomultiplier und PbF_2-Kristall können elektronische Signalbreiten von ca. 20 ns erreicht werden, welches einer maximalen Rate von etwa 20 MHz

entspricht.

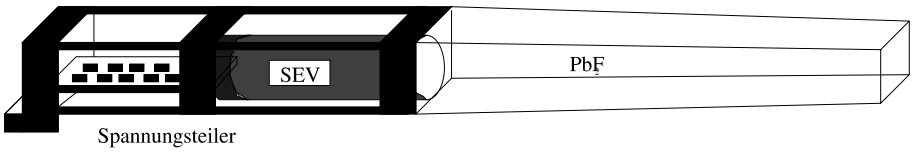


Abbildung 3.14: Bleifluorid-Modul des Photonenintensitätsdetektors

Mit Hilfe der Matrixanordnung der Kristalle kann der Detektor weiterhin zur Bestimmung der Strahlposition genutzt werden.

3.11 Zusammenfassung

Betrachtet man die Summe aller zu erfassenden Signalkanäle, ergibt sich eine Gesamtzahl von 4.699 Informationsquellen, wobei nur die wichtigsten Kanäle der Subdetektoren gezählt wurden. Zu diesen Kanälen kommen noch weitere Informationen hinzu, wie Hitpattern im Crystal-Barrel, Rateninformationen, etc. Dieses große Datenvolumen erfordert eine optimierte Signalaufnahme, eine geschickte Kompaktierung der Daten und ein leistungsfähiges Datenverarbeitungssystem.

Kapitel 4

Das Datenauslesesystem des CB-ELSA-Experimentes

4.1 Rahmenbedingungen - Ausgangspunkt

Ausgangspunkt für die Neuentwicklung war der Stand der Hardwarekomponenten zu Beginn des Jahres 2001.
Grundvoraussetzung eines schnellen Datenakquisitionssystems ist der Parallelismus in der Auslese der einzelnen Subdetektoren (→ Jedem Subdetektor ist ein Ausleseprozessor zugeordnet). Die Frontendkomponenten basieren auf dem VMEbus als Bus zur Kommunikation mit den Frontend-Elektronikmodulen. Die Steuerung der Module erfolgte über Prozessorboards der Firma FORCE mit einem Motorola 680x0 Prozessor und dem Echtzeitbetriebssystem OS-9. Die Kopplung der einzelnen Rechnersysteme erfolgte über einen vertikalen parallelen Bus. Dieser Bus (VIC-Bus) stellte ein Mapping der einzelnen Systeme in einen Adressbereich des globalen Eventbuilders zur Verfügung, sodass der Transfer der Daten über den globalen Eventbuilder per VMEbus-Zugriff durchgeführt wurde. Der Austausch von Buffer-/Synchronisationssemaphoren war auf Basis eines Hardwaremechanismus mittels 32 Bit Wörtern implementiert, die von den Modulen des VIC-Busses zur Verfügung gestellt wurden. Die Daten aus dem globalen Eventbuilder wurden über eine FDDI-Verbindung zu einem Linux Event-Saver geleitet und dort auf Magnetband gespeichert.

Die Konvertierung der einzelnen analogen Signale in digitale Informationen wurde durch diverse Module erledigt. So werden z.B. ADC-/TDC-Modulen des Tagging-Systems über den CAMAC[1]-Bus und durch VMEbus Modulen ausgelesen. Beim Crystal-Barrel-Subdetektor basieren die ADC-Module auf dem FastBus-Standard. Die Verbindung vom VMEbus auf den FastBus ist mit einem speziellen Hardwareprozessor (Sequenzer) mit einem eingebauten drei

[1] **C**omputer-**a**ided **M**easurement **A**nd **C**ontrol

Slot VMEbus-Crate realisiert worden, das dann den Ausleseprozessor des Subdetektors aufnimmt.
Dieser Grundstock an Elektronikkomponenten und die große Zahl der bereits existierenden elektronische Auslesekanäle ließ aus Kostengründen keinen Spielraum für gravierende Änderungen im Hardware-System zu.

4.2 Das Ziel der Neuentwicklung

Aufgrund der neuen Zielrichtung des Experimentes, Baryonenspektroskopie zu betreiben und die Suche nach fehlenden Resonanzen oder von exotischen Teilchen aufzunehmen, benötigt man eine hohe Anzahl von Ereignissen in den gewünschten Kanälen. Dieses Ziel kann auf zwei Wegen erreicht werden. Zum einen können die Strahlzeiten erhöht werden; zum anderen kann die Menge der genommenen Daten während der einzelnen Strahlzeiten gesteigert werden.
Die erste Variante ist ein sehr kostspieliger Weg und in einem vertretbaren Rahmen ließe sich maximal nur eine Verdopplung der aufgezeichneten Ereigniszahl erreichen. Die andere Variante geht in die Richtung der Verbesserung des Datennahmesystems, wobei die Geschwindigkeitssteigerung des Systems direkt als Faktor eingeht.
Glücklicherweise ließ die bisherige Struktur des DAQ-Systems durch die Aufteilung in Subsysteme mit einigermaßen gut definierten Schnittstellen eine Modernisierung durch Einsatz leistungsfähiger Prozessoren zu, ohne eine zeitlich zu große Unterbrechung in den Messphasen zu bewirken. Allerdings war eine komplette Neukonzeptionierung der Auslesesoftware notwendig, da die Rechnerarchitektur und das Betriebssystem drastisch unterschiedlich waren.
Innerhalb dieses neuen Konzeptes sollten folgende Merkmale mit berücksichtigt werden:

1. Verarbeitbarkeit von hohen Datenraten
2. Verwendung von modernen und leistungsfähigen Industriestandards
3. Skalierbarkeit des Gesamtsystems in den Bereichen Hardware und Software
4. Einfache Erweiterbarkeit für die Hinzunahme von neuen Detektorkomponenten
5. Modularer Aufbau des Systems und Abgrenzung der Bereiche durch Schnittstellen

4.3 Die Datenauslese-Prozessoren

Das bisherige Prozessor-System auf Basis des Motorola 68040 mit einer Taktrate von 27 MHz entspricht nicht mehr dem Leistungsstand heutiger Prozessoren mit 700 MHz und größer; der Ersatz dieser Prozessoren ist also zwingend erforderlich.

Es bieten sich hierfür zwei mögliche Prozessor-Familien an: der PowerPC von Motorola oder der Pentium von Intel (Die 680x0-Serie wird von Motorola nicht mehr weiterentwickelt und ist daher keine Alternative). Somit stand nun auch die Betriebssystemplatform zur Wahl.
Da das Open Source Betriebssystem Linux in den letzten Jahren eine rasante Entwicklung erfahren hat, und die Unterstützung des Betriebssystems von Intel-Prozessoren am besten ist, fiel die Entscheidung auf das Linux-Betriebssystem auf Intel-Basis. Gegenüber dem verwandten Betriebssystem OS-9 fehlt jedoch Linux die Echtzeitfähigkeit. In der Zwischenzeit existieren allerdings Erweiterungen auf ein Echtzeit-System für das Linux-System (RT-Linux, etc.).

Ein Nachteil des Intel-Prozessors ist die Inkompatibilität zum VMEbus, der in seiner Grundstruktur auf die Architektur des Motorola 680x0 ausgerichtet ist. Standardmäßig wird in modernen PC-Systemen der PCI-Bus unterstützt. Der Zugriff auf den VMEbus kann somit nur über Bus-Kopplungssysteme erreicht werden.
Im wesentlichen existieren hierfür zwei Formen. Die eine Variante basiert auf zwei Einsteckkarten, die auf der einen Seite in herkömmlichen PCs auf PCI-Basis Platz finden und auf der anderen Seite als VMEbus-Modul in das VMEbus-Crate gesteckt werden. Die Schnittstellenkarten können per Lichtleiter oder über Kabel gekoppelt werden. Die andere Variante sind Prozessorkarten mit einem speziellen Chip, der den Übergang zwischen dem PCI- und VMEbus durchführt. Diese Prozessorkarten sind mechanisch in der VMEbus-Norm ausgeführt und passen daher direkt in ein VMEbus-Auslese-Crate.
Die Wahl fiel auf eine VMEbus Prozessorkarte von der Firma VMIC. Es handelt sich hierbei um ein Modul, das auf einem Intel Pentium III Prozessor mit 733 Mhz oder 1,2 Ghz basiert. Diese Karte ist mit zwei FastEthernet Netzwerkschnittstellen, einem PMC Erweiterungsslot (z.B. Erweiterung auf Gigabit, FPGA-Module, ...), einem Spezialchip zur Konvertierung der unterschiedlichen Byteordnung (little- und big-endian), die bei Intelprozessoren und auf dem VMEbus Verwendung finden, und dem Brückenchip Universe II von der Firma Tundra für die Umsetzung vom PCI-Bus auf den VMEbus ausgestattet. Durch diese Wahl müssen nur die alten Prozessorkarten ersetzt werden, und es sind keine weiteren Hardware-Komponenten erforderlich, die in den bestehenden Racks untergebracht werden müssten. Die neuen VMIC-Prozessoren erhöhen zunächst einmal die Geschwindigkeit der Datenverabeitung vor Ort an den Subdetektoren auf Grund der um den Faktor 30 gesteigerten Rechnerleistung.
Einen weiteren Punkt zur Verbesserung des Datendurchsatzes bot die Ersetzung des parallelen Bus (VIC), über den alle System miteinander gekoppelt sind. Dieser bietet lediglich eine maximale Datenrate von ca. 2 MB/s, die für Hochstatistik-Experimente mit großem Datenaufkommen nicht ausreichend ist. Das Standard-Netzwerksystem auf serieller Basis, die auf den verwendeten neuen Prozessorkarten serienmäßig vorhanden sind, bieten eine wesentliche höhere Leistung von 10 MB/s (siehe Anhang E). Näheres zum benutzten Netzwerksystem wird in einem späteren Abschnitt erläutert.

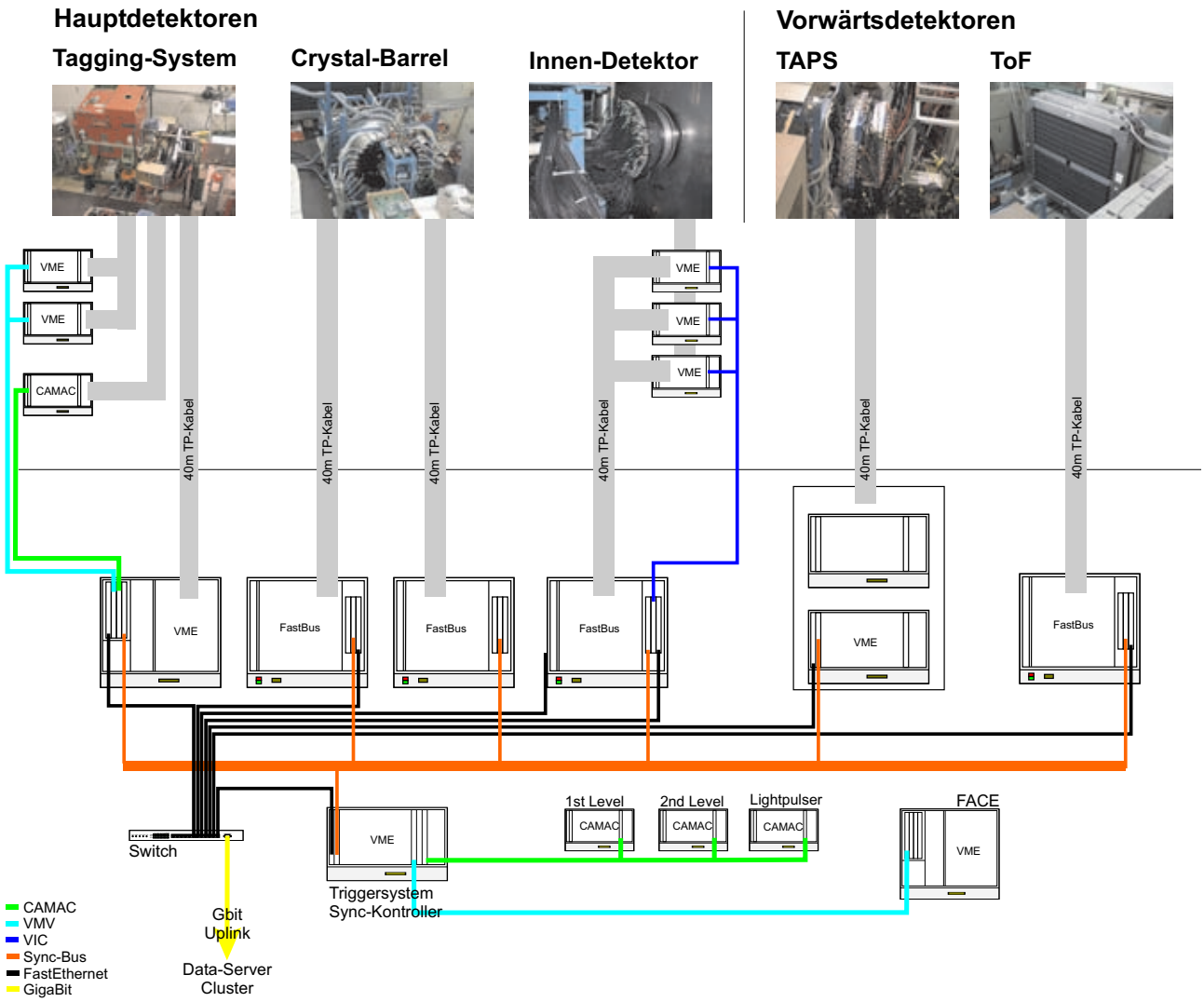


Abbildung 4.1: Schematische Darstellung des neuen Datenauslesesystems am CB-ELSA-Experiment

Mit dem Ersatz des vertikalen parallelen Busses durch ein serielles Netzwerksystem entfiel allerdings die dort integrierte Bufferverwaltungs- und Synchronisationsmöglichkeit. Daher musste diese Funktionalität durch ein Synchronisationssystem auf Basis eines multifunktionalen VMEbus-Moduls realisiert werden. Dieses ist eine Entwicklung des HISKPs. Auf dessen Funktionsweise wird in einem späteren Kapitel eingegangen. In Abbildung 4.1 ist lediglich ein schematischer Überblick über den Aufbau neuen Datenakquisitionssystem des CB-ELSA-Experimentes gezeigt.

4.4 Das Netzwerksystem

Die Parallelität in der Datenverarbeitung der Subdetektoren lässt sich fortsetzen im Datentransfer zu einem zentralen Rechnersystem, das die Informationen zu einem vollständigen Datensatz (Event) zusammenfasst, und durch die Verwendung einer parallelen Netzstruktur, die die einzelnen Datenströme über einen externen schnellen Multiplexer (Switch) zusammenfasst.

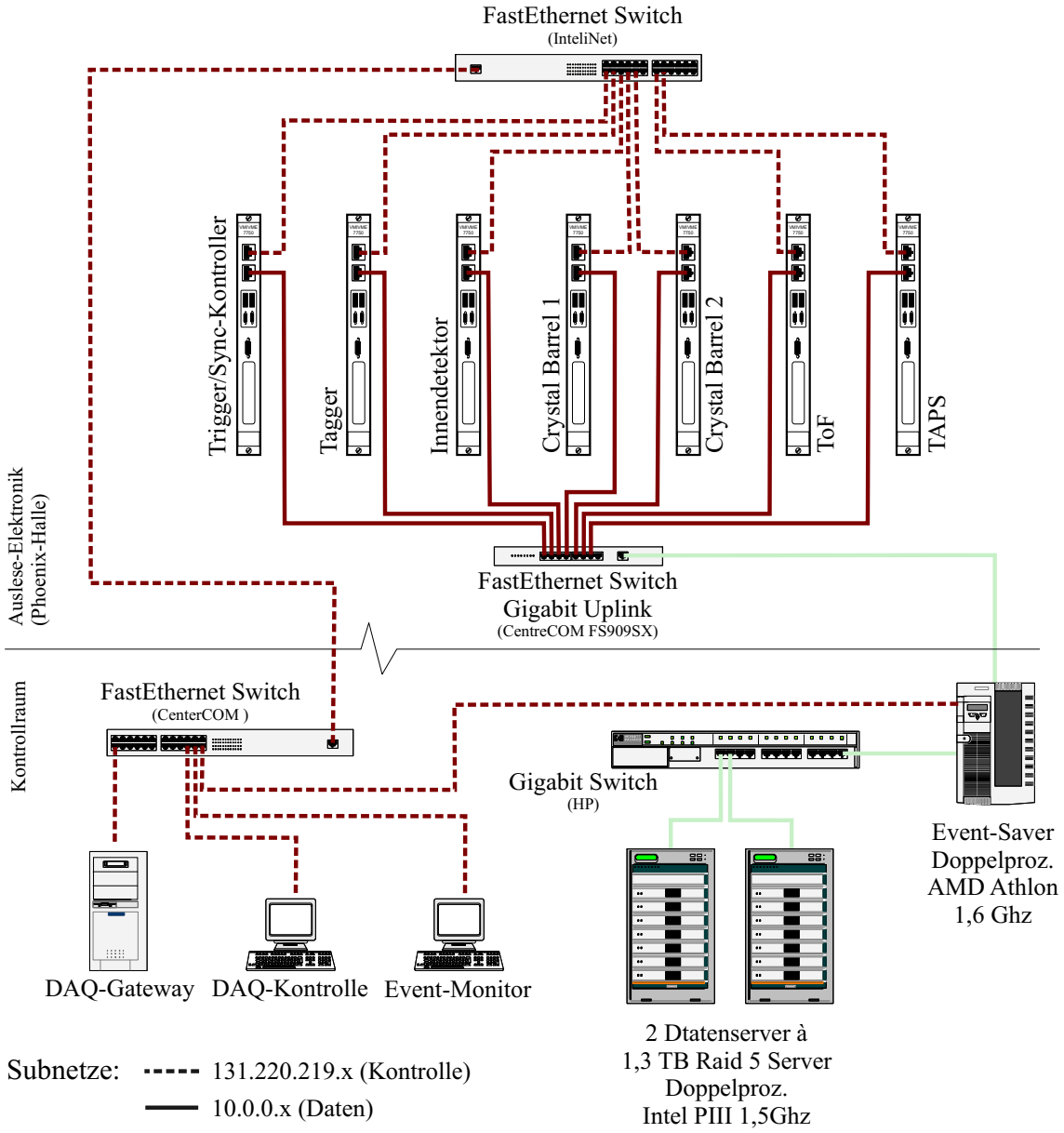


Abbildung 4.2: Struktur der Netzwerkverbindungen im Experiment

Das Transportsystem zwischen den Prozessorkarten und einem zentralen Sammelpunkt, dem Event-Saver, besteht aus zwei Netzwerkstandards, FastEthernet und Gigabit. Die Daten werden über eine 100MBit-Ethernet-Netzwerkschnittstelle übertragen und über einen Layer 2 Switch

gebündelt, die serialisierten Datenpakete werden über eine GigaBit-Verbindung weiterübertragen. Da die Datenrate des GigaBit-Uplinkports eine Zehnerpotenz zu nimmt, bleiben die Daten weiterhin quasi "parallel". Die Kopplung zwischen dem Elektronikbereich (in der Abbildung oberhalb der Linie) und dem Kontrollraum sind über optische Leiter durchgeführt. Um die Parallelität durch andere Datenverbindungen (Kontroll-, Terminalverbindungen, etc.) nicht zu stören, existieren zwei getrennte Netzwerke. Die Frontend-CPUs (siehe Abbildung 4.2) sind daher über zwei FastEthernet Netzwerkschnittstellen vernetzt. Grundsätzlich wird unterschieden in Datenpakete zur Kontrolle und in Pakete zum Transport der anfallenden Daten. Hierzu wird jeweils eine Netzwerkschnittstelle der VMIC-CPUs genutzt.
Die Kontrollverbindungen werden über einen eigenen Switch im Elektronikbereich per Lichtleiter an einen weiteren Switch im Kontrollraum gekoppelt. An diesem Switch befinden sich alle notwendigen Überwachungs- und Steuer-PCs.

4.5 Die Datenkompaktierung und Speicherung

Am Uplink-Port des Switches für die Datenverbindungen befindet sich über eine Punkt-zu-Punkt Verbindung der Event-Saver. Der Event-Saver ist ein Doppelprozessorsystem mit zwei AMD-Athlon 1,6 Ghz Prozessoren und zwei Gigabit-Netzwerkkarten der Firma Intel. Auf diesem Rechner erfolgt die Datensammlung und Kompaktierung bzw. Zusammenfassung der Event-Informationen. Um hier dem hohen Datenaufkommen gerecht zu werden, kommt ein Doppelprozessorsystem für maximale Verarbeitungsleistung zum Einsatz.

Betrachtet man das Datenaufkommen bei Experimenten mit einer Ereignisrate von 300 Hz und einer Ereignisgröße von 2,5 kB, ergibt sich folgendes Datenproduktionsaufkommen:

$$2,5\ kB \cdot 300\ Hz = 975\ kB/s = 3,3\ GB/h = 80\ GB/d$$

Diese Zahlen repräsentieren eine relativ geringe Ereignisrate. Die Datenmenge steigt bei dem Ziel, eine Rate von ca. 1,2 kHz aufzuzeichnen und zu verarbeiten, um einen Faktor vier an (320 GB/d).
Um solche Datenmengen speichern zu können, stehen zwei Datenserver von je 1,3 TB zur Verfügung. Auch diese Server sind mit jeweils einen Gigabit-Netzwerkanschluss versehen und mit dem Event-Saver über einen Gigabit-Switch der Firma HP verbunden. Die 1,3 TB werden durch einen speziellen Festplattenkontroller und zehn 130 GB Festplatten aufgebaut. Da auf Grund der großen Anzahl von Festplatten auch die Ausfallwahrscheinlichkeit immens ansteigt, existiert eine weitere Festplatte in diesem Festplattensystem, sodass über ein spezielles Verteilungsverfahren (RAID 5) der Daten über die einzelnen Festplatten kein Datenverlust eintritt, wenn eine Festplatte ausfällt. Auf diese Weise wird der Verlust von wertvollen Daten durch

einen Hardwaredefekt einer Festplatte minimiert.
Um diese beiden Datenserver in ihrer Speicherkapazität effizient zu nutzen, werden die gespeicherten Daten noch zusätzlich durch einen Kompressionsalgorithmus verkleinert. Mit diesem Verfahren können die Daten um einen Faktor zwei reduziert werden.

Kapitel 5

Der Trigger und das Synchronisationssystem

In der Teilchenphysik liegt das Interesse in der Aufzeichnung von seltenen Reaktionen, wie z.B. dem Zerfall von exotischen Teilchen. Diese Ereignisse befinden sich oft in einer Vielzahl von anderen Reaktionen, welche von untergeordnetem Interesse sind. Die Mechanismen zur Selektion und die Aufzeichnung dieser Informationen haben bei allen Experimenten eine gemeinsame Grundstruktur (siehe Abbildung 5.1).

Sie besteht aus einer Selektioneinheit und einem System zur Erfassung der Ereignisse (Digitalisierung und Speicherung). Diese sind miteinander verkoppelt, d.h. die Selektionseinheit, auch Trigger genannt, kann den Zustand der Datenerfassung ändern und umgekehrt. Die Selektionseinheit übernimmt das Filtern der gewünschten Ereignistopologie aus der großen Anzahl von Reaktionen, d.h. sobald eine Situation vorliegt, in der das gewünschte Ereignis enthalten ist, wird ein Signal an das Datenerfassungssystem gegeben, gleichzeitig wird die Triggerentscheidungslogik geschlossen. Die Erfassung startet die Digitalisierung des Ereignisses und speichert die Daten auf einem Medium ab. Sobald die Daten erfolgreich gespeichert wurden, wird der Zustand des Triggers wieder auf offen gestellt, und der

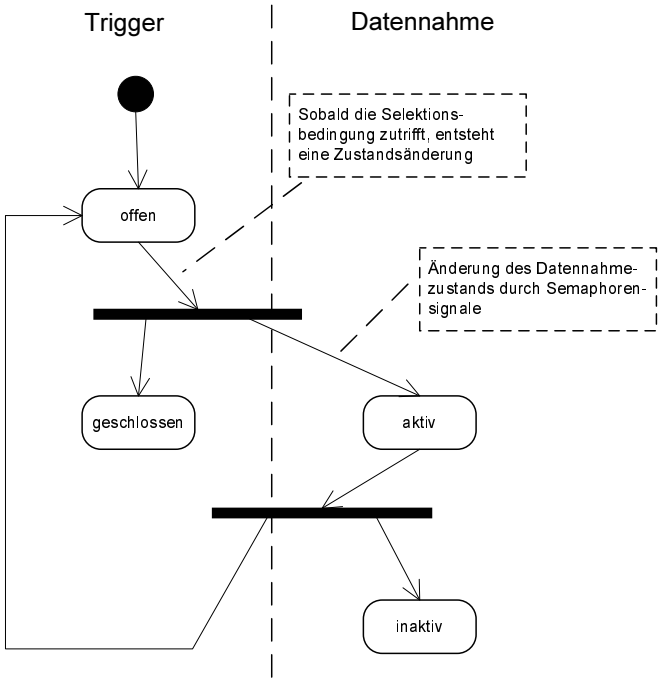


Abbildung 5.1: DAQ-Zustandsdiagramm

Vorgang kann von neuem beginnen.

5.1 Die Selektionseinheit - Trigger

Der Trigger des CB-ELSA-Experimentes basiert auf zwei zeitlich aufeinanderfolgenden Entscheidungsstufen (siehe Abbildung 5.2).
Die Stufe 0 ist die "schnelle" und die Stufe 1 die "langsame" Komponente. Zwischen Stufe 0 und 1 befindet sich ein "gate", das die nächste Stufe aktiviert, sobald ein Ereignis von der Stufe 0 akzeptiert wurde. Mit dem Aktivieren der nächsten Stufe ist auch die Erzeugung der Signale für die Konvertierung der analogen Informationen (Gate der ADCs) und der Startpunkt der Zeitmessung (Start für TDCs) der einzelnen Subdetektorkomponenten verbunden.

In der Stufe 0 können nur recht einfache Selektionskriterien angewandt werden, z.B. ob das Tagging-System durch ein Teilchen getroffen oder ob der Innendetektor von einem geladenen Teilchen durchquert wurde. Die Selektion auf bestimmte Ereignistopologien, die im Crystal-Barrel stattfinden, werden in einer nächsten Stufe mittels eines Vetos nachentschieden.
Die Entscheidung der Stufe 1 löst in der anschließenden Trigger-Kontroll-Einheit (TCU) entweder den Start der Datenerfassung ("event") oder einen Abbruch ("fast reset") aus.
Durch den Einsatz von programmierbaren Logikmodulen (u.a. LeCroy PLU 4508) ist das System sehr flexibel bezüglich der Konfiguration von Triggerbedingungen. Diese Module befinden sich in zwei Crates, die an unterschiedlichen Orten im Experiment installiert sind. Durch die Aufteilung in eine schnelle und eine langsame Komponente befindet sich ein Teil sehr nahe bei den Detektorkomponenten im Experiment, um Zeitverluste durch Kabellaufzeiten zu vermeiden. Das andere Crate ist in der Nähe des Prozessor-Crates des Sync-/Trigger-Kontrollers installiert. Die Steuerung und Programmierung erfolgt über einen CAMAC-Bus (siehe Abbildung 4.1).

5.1.1 Die Selektionsstufe 0

Wie schon erwähnt werden im Stufe 0 nur "schnelle" Entscheidungen getroffen. Hier können über zwei PLUs[1] 16 Eingangssignale mit binären Operationen verknüpft werden.
Eine PLU besteht aus acht Eingangs- und acht Ausgangssignalen. Diese können über einen 256 Byte großen Speicher miteinander verknüpft werden. Aus den acht Eingangssignalen wird binär eine Adresse gebildet und der Inhalt an dieser Speicherstelle an den Ausgangsleitungen angelegt. Somit ist es, möglich über die Programmierung des Speichers mit einem bestimmten binären Muster logische Verknüpfungen zwischen den Eingangssignalen zu erzeugen, die auf den frei wählbaren Ausgangsleitungen weiter verwendet werden können.

[1] **P**rogrammable **L**ookup **U**nit

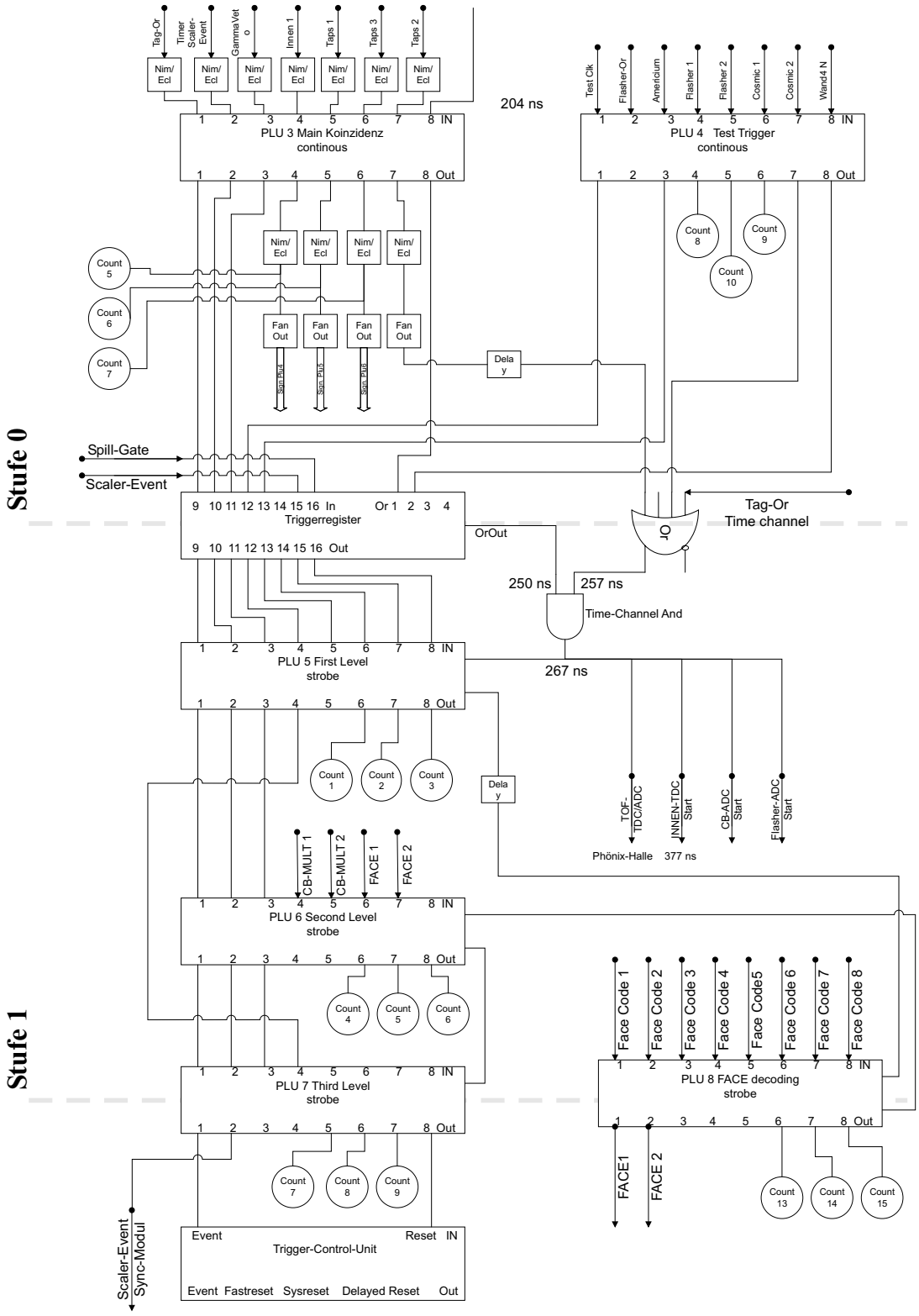


Abbildung 5.2: Schematischer Aufbau des Triggers am CB-ELSA-Experiment

Diese Stufe ist freilaufend, d.h. die Selektionssignale werden kontinuierlich erzeugt. Es ist Aufgabe des nachfolgenden Trigger-Gates, aus diesen Signalen das nächste Ereignis herauszugreifen, um die folgende Triggerstufe zu aktivieren. In dieser Selektionsstufe 0 werden die PLU´s im "overlapping mode" betrieben, wobei die Ausgangspulslänge durch die Zeitdauer des Anliegens der Eingangsbedingung gegeben ist. Sie kann z.B. so lang sein wie der Überlapp zweier Eingangssignale, falls diese durch ein UND miteinander verknüpft sind. Wenn diese Signale auch noch zeitlich jittern, schwankt dann die Breite um den Jitteranteil. Die maximale Schaltrate dieser Einheit beträgt 50 MHz.
Das Triggergate kann zusätzlich noch ein 8 Bit-Eingangsmuster zwischenspeichern. Das gespeicherte Bitmuster kann in den weiteren Stufen des Triggers zur Entscheidungsfindung herangezogen werden. Wird ein Ereignis akzeptiert, verriegelt sich das Trigger-Gate, und das Eingangs-Bit-Muster wird eingefroren. Da diese Triggerstufe die Erzeugung aller ADC-Gate Signale und TDC-Start-Signale vornimmt, müssen alle Detektorsignale um diese Entscheidungszeit verzögert werden, um in der richtigen zeitlichen Reihenfolge an den Digitalisierungseinheiten anzuliegen. Durch die Kabellängen zwischen Detektor und Frontend-Modulen (TDCs, ADCs) ist diese Entscheidungszeit auf maximal 300 ns beschränkt. Bevor aber das Ereignis endgültig akzeptiert wird, durchläuft das Bit-Muster noch eine weitere Stufe, die Selektionsstufe 1.

5.1.2 Die Selektionsstufe 1

In dieser Stufe kann durch externe Signale eine Weiterverabeitung des Ereignisses abgebrochen werden. Diese zusätzlichen Entscheidungsbedingungen können von den Subdetektoren geliefert werden oder von der FACE[2] stammen. Bei der FACE handelt es sich um einen in Hardware implementierten Clustersuchalgorithmus für den Crystal-Barrel-Detektor. Da die Schauer der Photonen sich über mehrere Kristalle verteilen, ist die Bestimmung der Zahl der Photonen über reine Multiplizitäten der getroffenen Kristalle viel zu ungenau. Der Clusteralgorithmus fasst alle zusammenhängenden Kristalltreffer zu einem Cluster zusammen und reduziert somit die einfache Hitmultiplizität auf die richtige Photonenzahl. Diese Logik ist in der Lage, innerhalb von ca. $4\mu s$ die Anzahl der Cluster im Crystal-Barrel zu liefern (Typische Selektionen waren z.B. zwei oder mehr Cluster im Crystal-Barrel).

Wird also in dieser Stufe das Ereignis verworfen, wird von der Trigger-Control-Unit (TCU) ein "fast reset"-Signal erzeugt und die Auslese-Elektronik mit einem "sysreset"-Signal wieder zurückgesetzt. Das "fast reset"-Signal öffnet nach ca. $10\mu s$ automatisch erneut den Trigger. Entsteht kein Veto, startet die Auslese der Elektronik und die Datennahme beginnt. Ist die Auslese beendet, wird das Triggerregister per Software über ein CAMAC-Kommando wieder geöffnet.

[2]Fast Cluster Encoder

Auf Grund einer Eigenschaft der Fastbus-ADC-Module des Crystal-Barrel-Detektors muss dieses Veto innerhalb von $10\mu s$ erfolgen. Diese Zeit ist an den ADC-Modulen fest eingestellt und gibt eine Pause (measuring pause intervall) vor, bevor die Module mit der Digitalisierung beginnen. In diesem Zeitintervall ist ein sofortiges Rücksetzen der Module möglich. Erfolgt keine Unterbrechung ("fast reset"), muss die Konversion von ca. 300 μs abgewartet werden.
Da die Eventinformation schon analog in den Frontendmodulen vollständig gespeichert ist, kann die Entscheidungszeit bis zu $10\mu s$ dauern, was komplexere Algorithmen zulässt. Die sich aus diesen Bedingungen ergebende Fast-Rest-Zeit beträgt maximal $15\mu s$.

5.2 Das Synchronisationssystem

Das Signal (Computer Trigger) zum Starten der Auslese der Frontend-Module (ADCs,TDCs,...) wird von der TCU an alle Prozessoren der Subdetektoren gesendet. Da die Verarbeitungszeit der einzelnen Prozessoren bei der Auslese unterschiedlich ist und starken Schwankungen unterliegt, ist eine Synchronisation dieser Prozessoren notwendig. Für einen korrekten Ablauf der Datennahme muss gewährleistet sein, dass alle Subdetektoren nach einem Auslesevorgang wieder für das nächste Ereignis bereit sind.
Bei einem einzelnen Rechnersystem, welches die Auslese aller Komponenten übernehmen würde, wäre eine Synchronisation nicht erforderlich. Da aber eine performante Auslese nur durch ein paralleles verteiltes Prozessorsystem erreicht wird, muss besonderes Augenmerk auf die Konsistens der Subdetektordaten gelegt werden. Durch eine parallele Auslese der Informationen sind die Daten über mehrere Rechnersysteme verteilt und benötigen zusätzlich durch den unterschiedlichen Umfang der Auslese und des Datenvolumens verschiedene Verarbeitungszeiten. Dies erfordert eine Synchronisation über Statusmeldungen der einzelnen Prozessorsysteme, um das Trigger-Gate im richtigen Zeitpunkt wieder zu öffnen.
Ein solches Handshake-Verfahren lässt sich sowohl mittels spezieller Hardware als auch über Standardverfahren in Software realisieren. Beide haben ihre Vor- und Nachteile, die im folgenden kurz erläutert werden:

- Handshake über spezielle Hardware

  Hierbei werden direkte Busverbindungen von jeder Prozessorkarte zu einem zentralen Punkt aufgebaut. Mittels einfacher Handshake-Leitungen wird der Zustand des jeweiligen Subdetektors vermittelt. Die Kommunikation läuft somit über ein Statusregister in einem Hardwaremodul ab, an dem die Handshake-Leitungen angeschlossen sind. Durch den Zugriff auf ein Register ist der Overhead für das Versenden solcher "Nachrichten" äußerst klein, da ein direkter Schreibzyklus aus dem Programm heraus auf das Modul erfolgt. Der Zyklus besteht aus einem VMEbus-Zugriff, der typischerweise eine Zeit von 500 ns erfordert.

Das Verfahren benötigt keinen Programm-Overhead und ist extrem schnell. Auf der anderen Seite erfordert es den Bau der diversen Hardware-Module für den zentralen Punkt (Master) und die jeweiligen Prozessoren (Slaves) und beinhaltet somit aufwendige Hardware. Ein weiterer Punkt ist, dass die Handshake-Information sehr einfach aufgebaut sein muss, um den betriebenen Hardwareaufwand klein zu halten.

- Handshake über Standardverfahren mittels Software

  Da die Prozessoren über eine serielle Netzwerkschnittstelle (FastEthernet) verfügen, kann eine Synchronisation über Kontrolldaten aufgebaut werden, indem diese per TCP/IP-Verbindung über das Netzwerk verschickt werden. Da Datenpakete versandt werden, kann der Aufbau der Synchronisationsdaten sehr individuell sein, wobei aber auf ein kompaktes Format geachtet werden muss, um die Transferzeit möglichst klein zu halten. Das Verfahren erfordert keinen Neubau von Hardware, da die Kommunikationswege schon standardmäßig vorhanden sind. Auch bezüglich der Software-Entwicklung ist der Aufwand gering, da hier umfangreiche Softwarebibliotheken für eine Interprozessorkommunikation zur Verfügung stehen.

  Durch den verstärkten Einsatz von Softwarelösungen steigt aber auch der Programm-Ooverhead und somit die Transferzeit einer Synchronisationsnachricht. Diese kann durchaus mehrere 100 μs betragen. Da es sich sich um keine dedizierte Verbindung zwischen dem Master und dem Slave handelt, hängt die Verarbeitungszeit stark von der Netzwerklast ab und ist somit nicht deterministisch.

Da ein Datennahmesystem möglichst performant sein und ein deterministisches Verhalten aufweisen soll, wurde eine Hardware-Lösung gewählt, die nun im folgenden diskutiert wird.

Abbildung 5.3 zeigt die Verbindungen und die Bestandteile des Synchronisationssystems. Es besteht aus einem Sync-Branch-Kontroller, an dem alle Statusmeldungen separat zusammenlaufen; dieser übernimmt zusätzlich ein Steuern der einzelnen Sync-Clientmodule. Die Statusmeldungen werden von dem jeweiligen Sync-Clientmodul über zwei Leitungen zum Sync-Branch-Kontroller übertragen. Die Steuerung aller Sync-Clientmodule erfolgt über einen zusätzlich parallelen 16 Bit-breiten ECL-Bus.
Das Synchronisationsystem erfüllt auf zwei Wegen eine Gewährleistung der Datenkonsistenz der Eventinformationen.

Die Prozessor-Synchronisation

Ein Hauptproblem bei der parallelen Verarbeitung der Daten liegt in den unterschiedlichen Verarbeitungsgeschwindigkeit der einzelnen Rechnersysteme. Ohne eine Rückmeldung der jeweiligen Systeme an einen zentralen Punkt würden die einzelnen Systeme bei einer Unregelmäßigkeit

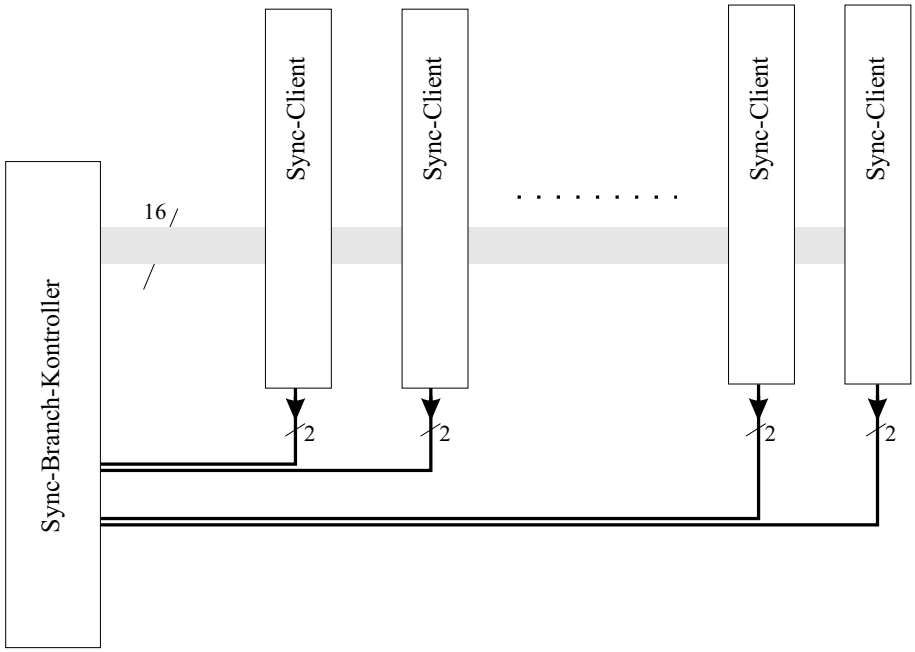


Abbildung 5.3: Sync-Branch-Kontroller und Sync-Clientmodule mit Bus und Statusleitungen

aus der Synchronisierung geraten. In einem Master-Slave-Betrieb kann die Synchronisierung gewährleistet werden, indem der Master in diesem System die Kontrolle über das Trigger-Gate führt und die Slaves überprüft, ob deren Operationen beendet worden sind. Der Master öffnet das Trigger-Gate erst wieder, wenn die Slaves eine Bestätigung zurückliefern.

Das System stellt zwei Statusleitungen zur Abwicklung des Handshakes zwischen Slave und Master zur Verfügung:

1. Die Auslese wird ausgeführt (BUSY).
2. Die Auslese war erfolgreich (OK).

Der Ablauf dieses Synchronisationsprozesse auf Basis dieser beiden Leitungen wird im Abschnitt 5.2.3 an Hand eines Beispiels näher erläutert.

Das Generieren von Synchronisationsdaten

Zur weiteren Überprüfung der Datenintegrität werden noch zusätzliche Informationen bei jedem Auslesevorgang mit den Daten jedes lokalen Eventbuilders aufgezeichnet. Über den Sync-Branch-Kontroller wird eine 4 Bit-breite Buffernummer über den ECL-Bus an die Sync-Clientmodule übertragen. Diese Nummer wird vor jedem Zyklus vom Sync-/Trigger-Kontroller-Prozess erzeugt und an die Sync-Clientmodule der lokalen Eventbuilder transferiert. Bei Beginn der Auslese wird die Nummer durch den Ausleseprozess des jeweiligen lokalen Eventbuilders mit in die

Statusinformationen des Ereignisses übernommen. Die Kontrolle der Buffernummer wird beim Zusammenstellen der einzelnen Daten der lokalen Eventbuilder zu einem vollständigen Ereignis ausgeführt (Näheres zur Überprüfung der Synchronität auf Basis der Buffernummer wird in Kapitel 7.4.1 gegeben).

Die folgenden Abschnitte geben einen Überblick über die technische Umsetzung dieses Synchronisationssystems.

5.2.1 Das Sync-Clientmodul

Als Basis für diese Module dient ein Latch-Counter-Modul, welches im SAPHIR-Experiment zur Auslese der Proportionalkammern des Tagging-System verwendet wurde. Das Design dieses Moduls [25] erlaubt es, durch eine Aufsteckkarte die Ein/Ausgangslogik des Moduls zu verändern, sodass das VMEbus-Interface, die Zähler und das Latchregister wiederverwendet werden können.
Abbildung 5.4 zeigt das Blockdiagramm der Eingangslogik der Aufsteckkarte für dieses VMEbus-Modul.
Die Karte verfügt vier separate NIM-Eingänge und einen 16 Bit breiten ECL-Bus. Bei den NIM-Eingängen existiert ein Readout-Eingang, über den das Computer-Trigger-Signal zugeführt wird, und einen SysReset-Eingang, der zum Rücksetzen des Sync-Moduls verwendet wird. Beide Signale werden von der TCU erzeugt. Der Clock-Eingang dient zur Taktung der Performancezähler (Totzeit, etc.) und wird standardmäßig von einem 20 MHz-Takt gespeist.
Der ECL-Bus wird über den Sync-Branch-Kontroller versorgt, wobei die unteren 3 Bits des ECL-Buses durch externe Signale eingespeist werden. Mit Hilfe dieser `Int_0` bis `Int_2` (siehe Abbildung 5.1) können Informationen zum Triggertyp (interruptqualifier) übermittelt werden und somit zusätzliche Optionen in der Auslese aktiviert werden. Die Aktivierung des Sync-Clientmoduls und das Setzen der Buffernummer erfolgt durch den Sync-Branch-Kontroller.

Eingangsregister des Sync-Clientmoduls:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Frei	Frei	Frei	Frei	Frei	Frei	Addr-Sel	Cmd-Strobe	Buf-Nr Bit 4 / Addr Bit 3	Buf-Nr Bit 3 / Addr Bit 2	Buf-Nr Bit 2 / Addr Bit 1	Buf-Nr Bit 1 / Addr Bit 0	Buf-Nr Bit 0	Int_2	Int_1	Int_0

Tabelle 5.1: Registerstruktur des Sync-Clientmoduls

Jede Client-Karte am Synchronisationsbus kann getrennt aktiv oder inaktiv geschaltet werden. Die Aktivierung bedeutet hierbei, ob der jeweilige lokale Eventbuilder an der Datennahme teilnehmen soll oder nicht. Um zu diesem Zwecke eine Selektion des jeweiligen Sync-Moduls vornehmen zu können, besitzt jede Aufsteckkarte eine eindeutige Nummer, die in einem program-

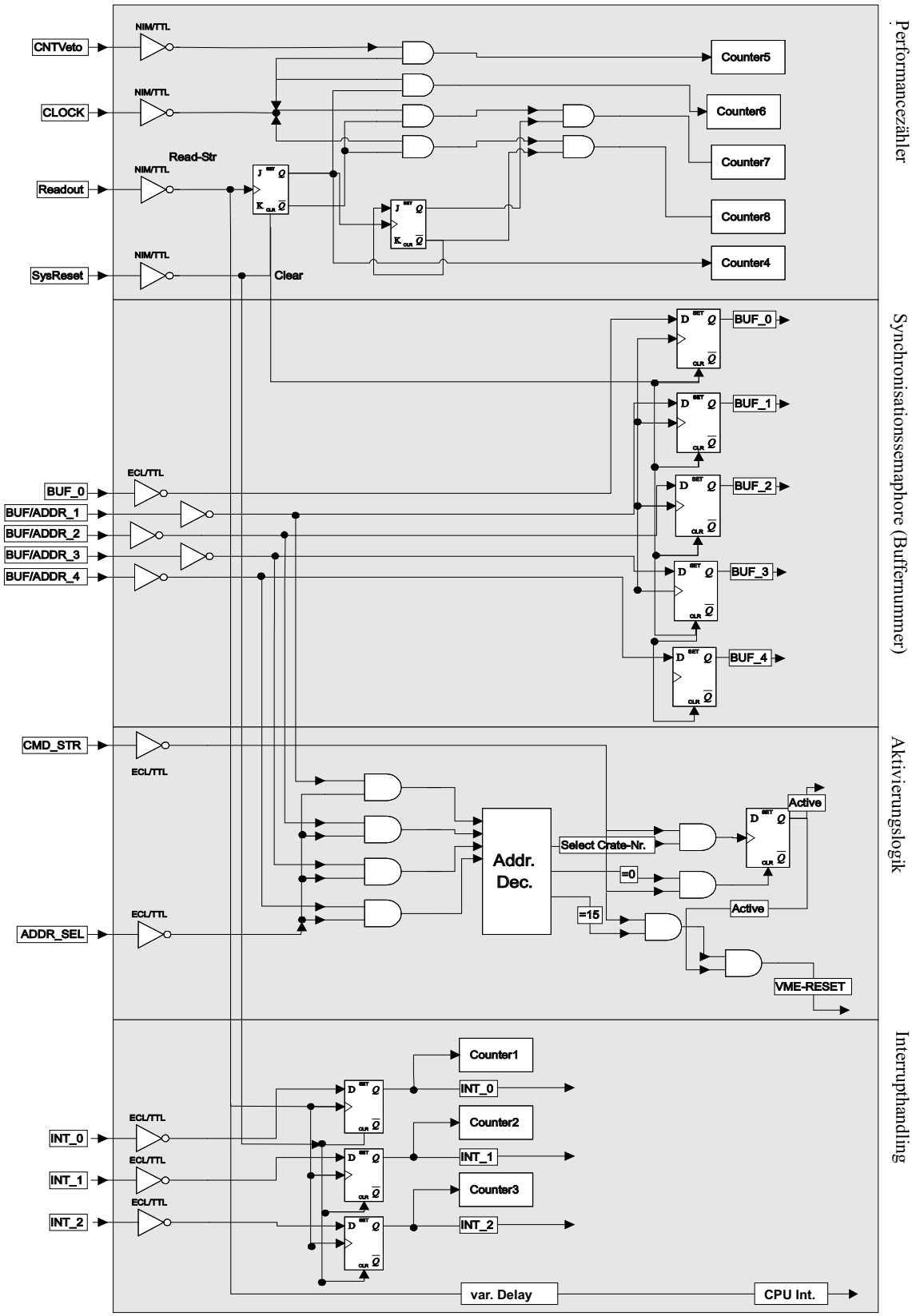


Abbildung 5.4: Eingangslogik des Sync-Client-Moduls

mierbaren Baustein eingebrannt ist. Diese Aktivierungsphase wird vom Sync-Branch-Kontroller durch Setzen des ADDR_SEL-Bit angezeigt und muss dann sequentiell durch Setzen der entsprechenden Identitätsnummer auf den Client-Modulen durchgeführt werden.
Da das Basismodul nicht über eine Interruptmöglichkeit auf dem VMEbus verfügt, wird für die Erzeugung des Auslese-Interrupts der serielle Port auf der jeweiligen Prozessorkarte verwendet. Über den CTS-Eingang der seriellen Schnittstelle wird ein Interrupt für den Ausleseprozess erzeugt. Zwischen dem Readout-Signal und dem Erzeugen des Interrupts über den CTS-Eingang kann eine Verzögerung von $20\mu s$ bis $400\mu s$ eingestellt werden (siehe Abbildung 5.4). Dies ist sehr nützlich, da die durch das Auslesesignal angestoßenen autonomen Ausleseprozesse eine bestimmte Zeit zur Abarbeitung benötigen und erst dann, nach der Bereitstellung der Daten in den Zwischenbuffern, der Datensammelprozess beginnt.

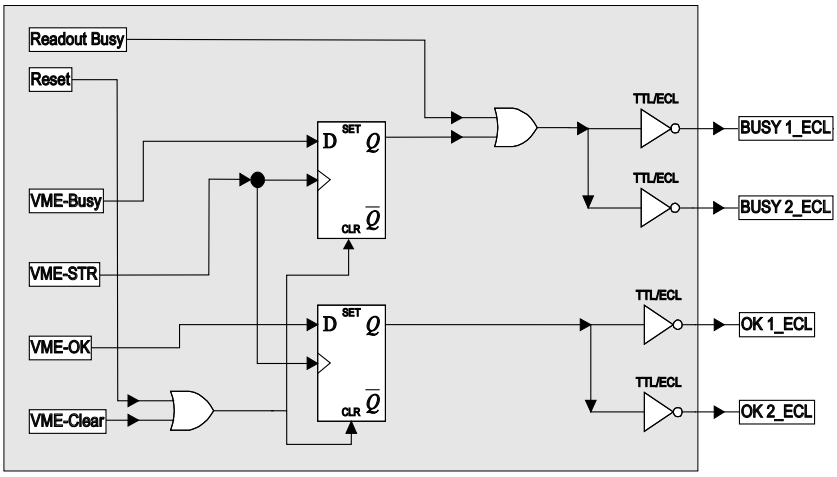


Abbildung 5.5: Ausgangslogik des Client Sync-Moduls

Die Rückmeldungen des Ausleseprozesses zum Sync-Branch-Kontroller werden über dedizierte Leitungen geführt. Die hierfür notwendige Ausgangsstufe ist in Abbildung 5.5 gezeigt. Die Statusinformationen BUSY und OK werden über jeweils ein Flip-Flop zwischengespeichert. Die Leitungen "VME-Busy", "VME-OK" und "VME-Clear" können per Software über ein Register betätigt werden. "VME-STR" (Strobesignal zur Datenübernahme) und "Reset" sind weitergeleitete Signale des VME-Busses.
Eine Besonderheit existiert beim BUSY-Signal: Die Information des BUSY-FlipFlops kann durch ein externes Signal ("Readout-Busy") überlagert werden. Hierdurch ist es auch möglich, ein hardwaremäßig erzeugtes BUSY-Signal in die Sync-Logik mit einfliessen zu lassen.

5.2.2 Der Sync-Branch-Kontroller

Das Mastermodul empfängt die Statusmeldungen der lokalen Eventbuilder und übernimmt weitere Funktionen und Einstellungen über den ECL-Bus, die in folgender Übersicht zusammen-

gefasst sind:

- Aktivierung eines Sync-Client-Moduls
- Deaktivierung aller am Sync-Bus angeschlossenen Sync-Clients
- Ausführen eines VME-Resets an allen aktivierten Sync-Modulen
- Setzen einer Buffernummer in allen Sync-Modulen

Das Mastermodul ist ein 16 Bit Ein-/Ausgaberegister auf Basis einer VMEbus-Einsteckkarte. Das Eingangs- wie das Ausgangsregister verarbeiten ECL-Signale (Tabelle 5.2 zeigt die Struktur der Register).

Eingangsregister des Sync-Kontrollermoduls:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUSY CPU 8	CPU 7	CPU 6	CPU 5	CPU 4	CPU 3	CPU 2	BUSY CPU 1	OK CPU 8	CPU 7	CPU 6	CPU 5	CPU 4	CPU 3	CPU 2	OK CPU 1

Ausgangsregister des Sync-Kontrollermoduls:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Frei	Frei	Frei	Frei	Frei	Frei	Addr-Sel 0		Buf-Nr Bit 4	Buf-Nr Bit 3	Buf-Nr Bit 2	Buf-Nr Bit 1	Buf-Nr Bit 0	reser-viert	reser-viert	reser-viert
Frei	Frei	Frei	Frei	Frei	Frei	Addr-Sel 1	Cmd-Strobe	Addr Bit 3	Addr Bit 2	Addr Bit 1	Addr Bit 0				

Tabelle 5.2: Registerstruktur des Sync-Branch-Kontrollers

Das Ausgangsregister der Karte mündet bis auf die unteren drei Bits direkt in den ECL-Bus, der die einzelnen Sync-Clientmodule parallel miteinander verbindet. Die Aktivierung der einzelnen Sync-Clientmodule und die Verbreitung der Buffernummer erfolgen über diesen ECL-Bus bzw. über das Ausgaberegister.
Es gibt zwei unterschiedliche Belegungsvarianten (Normal- und Selektionsmodus) des Ausgaberegisters, die durch das `ADDR_SEL`-Bit bestimmt werden. Im Normalmodus (`ADDR_SEL`-Bit nicht gesetzt) werden die angelegte Buffernummer und die Triggerinformation in die Flip-Flops der Sync-Clientmodule bei einem Readout-Signal übernommen und können über den VMEbus in dem Eingangsregister des Sync-Clientmoduls gelesen werden. Im Selektionsmodus (Aktivierung des jeweiligen Sync-Clientmoduls) wird `ADDR_SEL` auf 1 gesetzt, und die Bits 5-8 bekommen eine neue Bedeutung. Legt man auf diese Bits eine vorhandene Id an und wird danach das Bit 9 (Kommandostrobe) gesetzt, so wird die entsprechende Karte aktiviert.

Die Ids mit der Nummern 0 und 15 haben eine besondere Bedeutung. Im Falle der 0 werden alle Karten deaktiviert, die an den ECL-Bus angeschlossen sind. Bei Id gleich 15 wird bei den aktivierten Sync-Modulen ein Reset auf dem VMEbus ausgeführt.

Das Eingangsregister der Sync-Branch-Kontroller-Karte wird von den OK- und BUSY-ECL-Leitungen der Sync-Clientmodule bedient (siehe Abbildung 5.2). Dies ergibt eine maximale Anzahl von acht Sync-Clientmodulen, die in diesem System betrieben werden können.

5.2.3 Der Synchronisationsablauf

Der zeitliche Verlauf während der Datennahme ist in Abbildung 5.6 exemplarisch dargestellt. Die relativen Zeitabstände und Pulslängen haben keine realistischen Werte und dienen lediglich der Veranschaulichung. Gezeigt sind die Statusinformationen aller Subdetektorkomponenten und die Signale *Computer-Trigger*, *Gate* und *Hauptkoinzidenz*.

Sobald das Gate geöffnet wird und ein Puls der Hauptkoinzidenz in dieses Fenster fällt, wird ca. 4 μs später ein Signal am Computer-Trigger erzeugt. Durch dieses Signal wird in den Sync-Modulen der Busy-Status auf high gesetzt. Die lokalen Prozessoren beginnen mit der Auslese der Komponenten. Nach dieser Auslese wird die erfolgreiche Auslese über den OK-Status gesetzt und das Busy-Signal zurückgenommen. Der Sync-/Trigger-Kontroller überprüft den Status aller Subdetektoren und öffnet das Gate für ein neues Ereignis, wenn alle aktivierten Sync-Clientmodule dieses OK-Signal übermittelt haben. Der folgende System-Reset setzt dann alle OK-/BUSY-FlipFlops hardwaremäßig zurück. Dieses Handshake-Verfahren und das einmalige Setzen des OK-Signals reduzieren die im allgemeinen langsamen Prozessorinteraktionen der Interruptverarbeitung und gewährleisten einen minimalen Programm-Overhead.

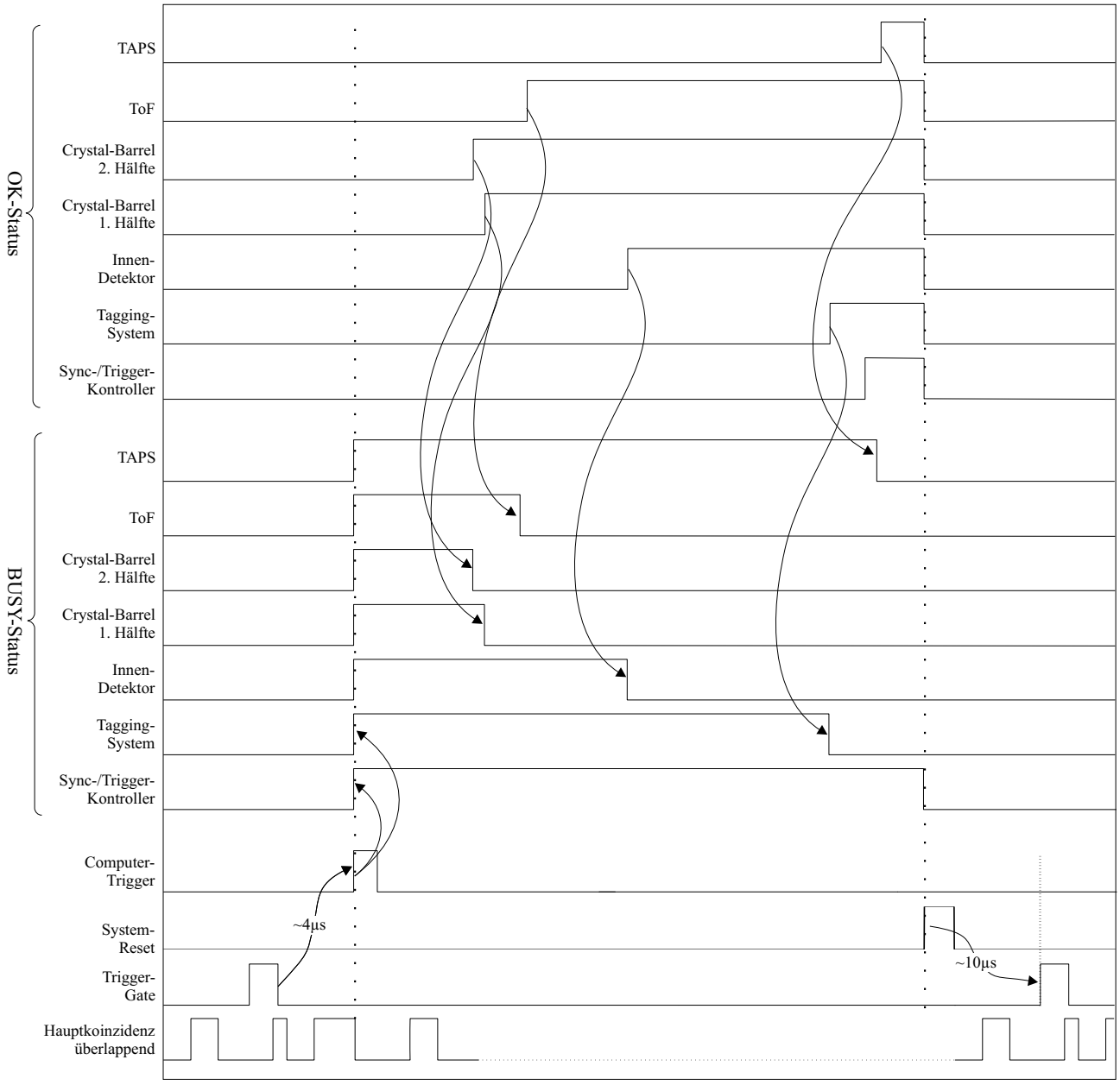


Abbildung 5.6: Zeitlicher Verlauf der Status-Signale während der Datennahme

Kapitel 6

Ausleseelektronik der Subdetektoren

Im folgenden Kapitel wird die Struktur der Ausleseelektronik der einzelnen Subdetektoren beschrieben, und es wird speziell auf die zu erwartenden Datenmengen und die mit den verschiedenen Auslesearchitekturen (VMEbus, Fastbus, CAMAC) verbundenen Auslesezeiten eingegangen. Die hier aufgeführten Zeiten, die einen ganz entscheidenen Einfluss auf den Gesamtdatendurchsatz der Datenakquisition haben, sind ohne den Programmoverhead als reine Zugriffszeiten auf die digitalen Informationen in den Frontend-Modulen zu verstehen. Somit stellen sie die theoretische obere Grenze der Auslesegeschwindigkeit dar, wenn man die entsprechenden Digitalisierungszeiten hinzu rechnet. Durch paralleles Auslesen der Subdetektorkomponenten trägt allerdings nur die langsamste Komponente wesentlich zur Begrenzung der Datenausleserate bei. Somit eröffnet sich die Möglichkeit, durch Neuaufbau dieser langsamsten Auslesekomponente eine Steigerung der Datenakquisitionsrate mit moderatem Aufwand zu erreichen.
Das zeitliche Gesamtverhalten inklusive der Auslese- und Datentransferprogramme wird ausführlich in Kapitel 8 diskutiert.

6.1 Der Sync-/Trigger-Kontroller

Dieser Prozessor spielt unter allen Frontend-Auslese-Systemen eine Sonderrolle. Er liest nicht nur Daten von verschiedenen Frontend-Elektronik-Modulen aus, er steuert auch das Trigger- und das Synchronisationssystem, indem er bei erfolgter Auslese entscheidet, wann das Trigger-Gate wieder geöffnet wird.

Die Steuerung des Triggers und die Auslese von Daten erfolgen hauptsächlich über den CAMAC-Bus. Der CAMAC-Branch-Kontroller CBD 8250 von CES wurde speziell für den Einsatz an dieser Subkomponente modifiziert. Im Gegensatz zur Standard-Version kann der Kontroller sowohl die Daten des gelesenen Moduls als auch das Status-Bit (Q-Bit) in einem 32 Bit-Lese-Zyklus liefern. Die Dauer eines solchen Zyklus beläuft sich auf ca. $2,5\ \mu s$.

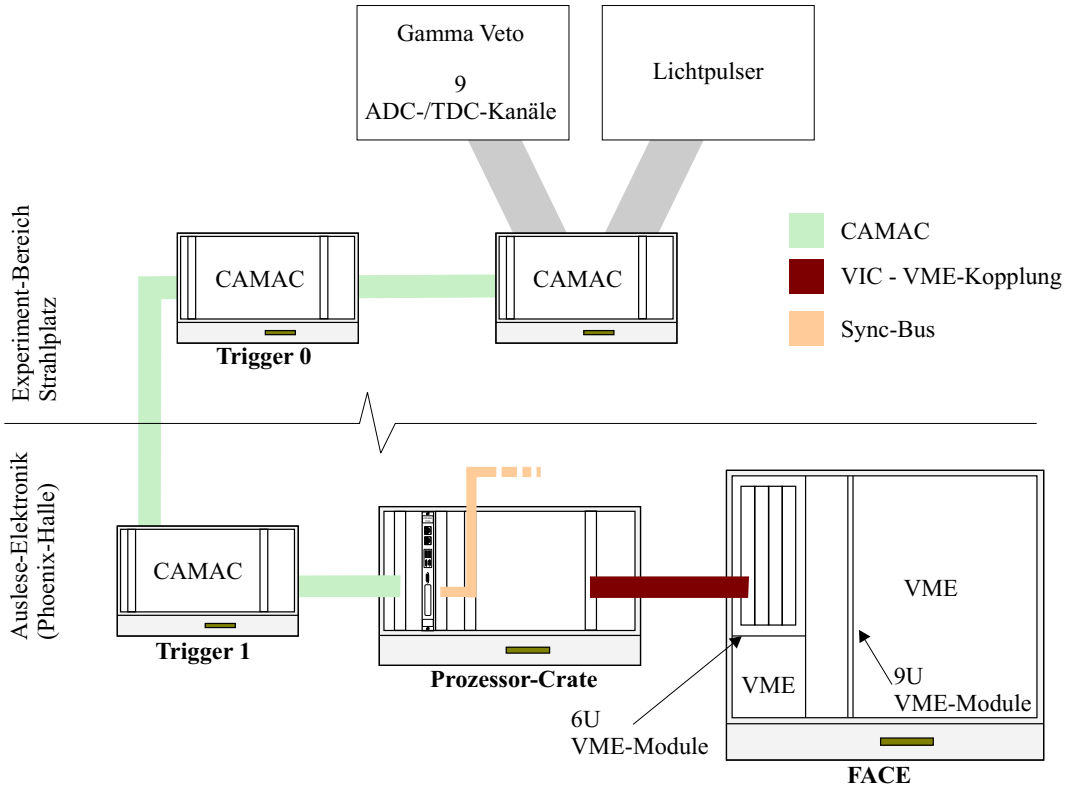


Abbildung 6.1: Komponenten des Sync-/Trigger-Kontroller

Die folgenden Tabellen zeigen den Umfang der über diesen Bus gelesenen Daten:

Trigger-Daten:

Anzahl	Datenbreite	Beschreibung
9	16	Informationen zum Trigger

Zähler-Daten:

Anzahl	Datenbreite	Beschreibung
12	29	Lifezähler
32	16	Spillzähler (32Bit, jeweils 2 kaskadiert)
32	16	Diverse Zähler für Signale innerhalb der Triggerstufen
9	24	Zähler Gamma-Veto (5 μs)
12	24	Zähler Gamma-Veto

Gamma-Veto-Daten:
Der komplette CAMAC-Zugriff zum Lesen der genannten Daten benötigt in der Summe ca. $9 \cdot 2,5\ \mu s + 24 \cdot 2,5\ \mu s + 13 \cdot 3,5\ \mu s + 97 \cdot 2,5\ \mu s = 370,5\ \mu s$.

Anzahl	Datenbreite	Beschreibung
12	12	ADC-Werte
12	12	TDC-Werte
13	16	Multihit-TDC

Die Daten der FACE werden über eine VMEbus-Kopplung mittels eines vertikalen Busses (VIC) per 16Bit VMEbus-Zyklus gelesen. Bei den Daten handelt es sich um die Anzahl der Cluster im Crystal-Barrel und um die Indexnummer der getroffenen Kristalle im Cluster. Der FACE-Kontroller sammelt die Daten der einzelnen Spezialchips des Clustersuchalgorithmus und legt diese in einer onboard First-in-First-Out-Einheit (FiFo) ab. Die Anzahl der Daten sind mindestens drei 16 Bit Werte plus die Anzahl der angesprochenen Kristalle im Crystal-Barrel. Je 16 Bit-Lese-Zyklus benötigt der Prozessor ca. $1,7\ \mu s$.

Zusammenfassend wird die folgende Datenmenge ausgelesen, wobei eine durchschnittliche Anzahl der von der FACE erkannten getroffenen Kristalle auf 50 Stück abgeschätzt wurde:

Daten (DATA_EVENT):

Trigger-Daten	$\sim$ 22,5 μs
+ Zähler	$\sim$ 105,5 μs
+ Gamma-Veto	$\sim$ 242,5 μs
+ FACE ($\sim$50 Kristalle)	$\sim$ 85 μs
	$\sim$ 455,5 μs

6.2 Das Tagging-System

In Abbildung 6.2 ist der Aufbau der beteiligten Bussysteme beim Tagging-System innerhalb der Auslese für die einzelnen Unterkomponenten gezeigt.
Die ADC- und TDC-Module der Szintillationszähler werden über einen CAMAC-Bus gelesen. Die Latch/Scaler-Module für die Proportionalkammer befinden sich in der Nähe des Tagging-Systems in VMEbus-Crates. Diese beiden Crates sind über einen VIC-Bus an den VMEbus des Prozessor-Crates gekoppelt. Die Multi-Hit-TDCs für die Szintillationsfasern sind in direkter Nähe der Prozessorkarte in einem 9 Höheneinheiten großen Crate untergebracht und werden über den VMEbus ausgelesen.

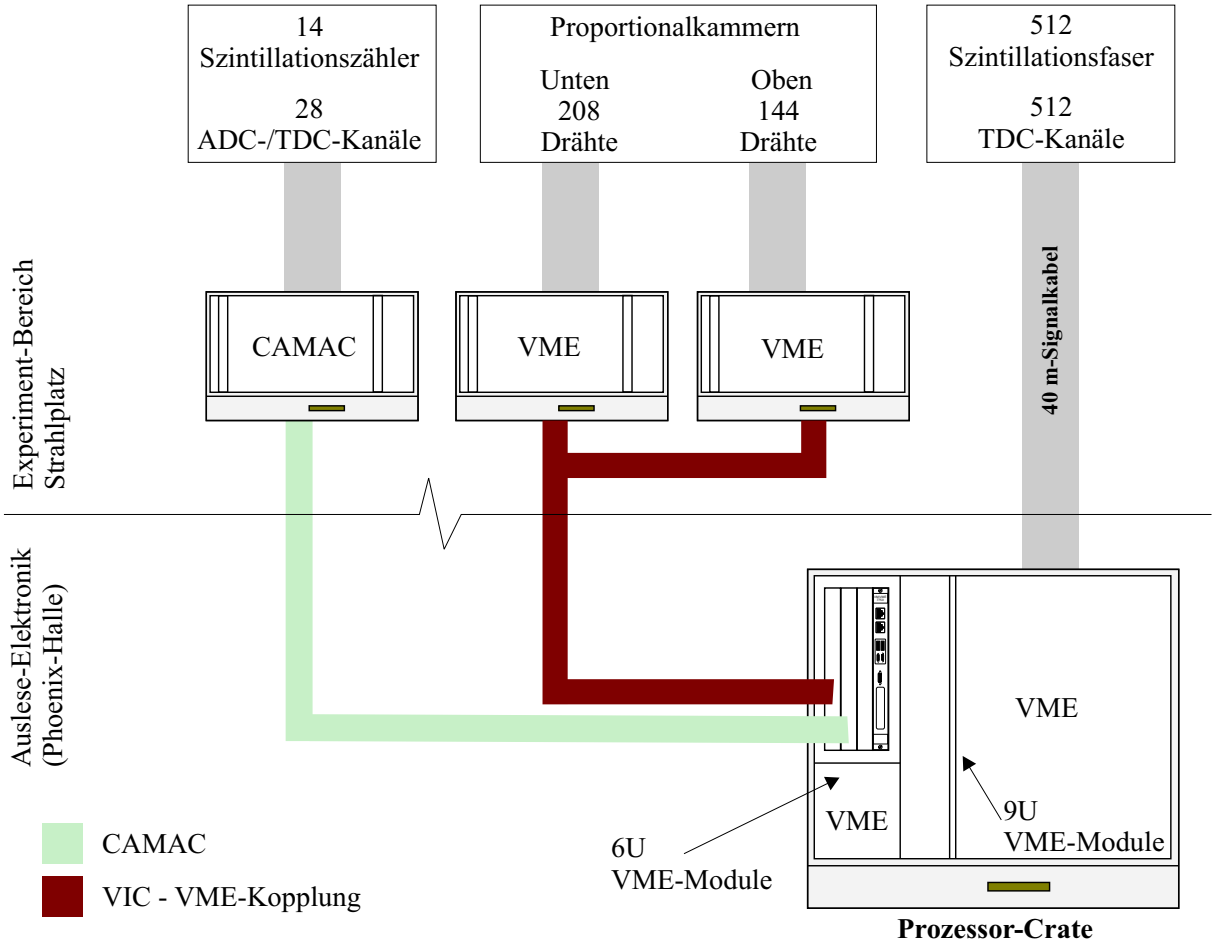


Abbildung 6.2: Komponenten des Tagging-Systems

Die Szintillationszähler

Die Photomultipliersignale der Szintillationszähler werden passiv in zwei Signale gesplittet. Ein Signal wird durch LeCroy 2249A CAMAC ADC-Module digitalisiert, das andere Signal wird über einen LeCroy 3420 CAMAC Diskriminator in ein logisches Signal umgewandelt. Nach einer zeitlichen Verzögerung durch eine programmierbare Delay-Line (LeCroy 4418) wird das Signal auf einen LeCroy 2229 CAMAC-TDC gegeben. Die Kopplung mit dem Bus des Prozessor-Crates wird mit einer Standard-Version des CAMAC-Branch-Kontrollers CBD 8250 von CES durchgeführt.

Der Zugriff auf die CAMAC-Module setzt sich hierbei aus zwei Zyklen zusammen: Lesen der Daten und Abfragen des Status des Zugriffs (Q-Bit). Ist das CAMAC-Modul korrekt angesprochen worden, setzt es das Q-Bit (Q-Response) auf dem CAMAC-Branch. Dieses Q-Bit kann über einen weiteren Zugriff auf das Statusregister des Branch-Kontrollers gelesen und überprüft werden.

Mit Hilfe dieser Abfolge an Zugriffen werden folgende Daten gelesen:

Lese-Zyklus	$\sim$ 2,5 μs
+ Status-Prüfung	$\sim$ 1,0 μs
	$\sim$ 3,5 μs

Anzahl	Datenbreite	Beschreibung
28	12	ADC Wert
28	12	TDC Wert

Die zu erwartende Auslesezeit für die Szintillationszähler ergibt sich zu:

$$28 \cdot 3,5\ \mu s + 28 \cdot 3,5\ \mu s = 196\ \mu s$$

Die Proportionalkammern

Oberhalb der Szintillationszähler sind zwei Proportionaldrahtkammern installiert. Die binären Informationen der insgesamt 352 Drähte werden über ein Scaler/Latch-VMEbus-Modul[25] erfasst. Dieses Modul verfügt über ein 16 Bit-Eingangsregister. Die Eingangssignale können über einen externen Trigger zwischengespeichert werden (Latch). Zusätzlich zu diesem Muster kann die Ansprechhäufigkeit pro Draht in 24 Bit-Zählern erfasst werden.
Für beide Kammern werden 22 Karten eingesetzt, die auf zwei VMEbus-Crates verteilt in unmittelbarer Nähe des Detektors installiert sind. Die Kopplung dieser beiden Crates an das Prozessor-Crate erfolgt über einen vertikalen Bus (CES VIC-8251).
Die Daten sowohl der Zähler als auch der Latches werden über einen Lese-Zyklus des Typs D16 (Latch) und D32 (Zähler) durchgeführt. Der Zugriff benötigt im Schnitt ca. 1 μs.

Anzahl	Datenbreite	Zeit	Beschreibung
22	16	$\sim$ 22 μs	Kammertreffer à 16 Bit
352	24	$\sim$ 352 μs	Zähler; Ansprechhäufigkeit der einzelnen Drähte

Damit wird eine Auslesezeit von 372 μs erwartet.

Die Szintillationsfasern

Die 512 Fasern des Szintillationsfaserdetektors werden ähnlich wie bei den Szintillationszählern mittels Photomultiplier ausgelesen, jedoch werden nur die diskriminierten Signale mittels Multihit-TDCs und Zähler erfasst.
Die Multihit-TDCs basieren auf dem TDC-Chip *F1*, einer Entwicklung der Universität Freiburg. Vier solcher Chips befinden sich auf einer kompakten Aufsteckkarte (CMC[1]) und stellen 32 TDCs mit einer Auflösung von 110 ps oder 16 TDCs mit einer Auflösung von 55 ps zur

[1] **C**ompact **M**ezzaine **C**ard

Verfügung. Die Datenauslese solcher CMC-Karten erfolgt über ein 9U-VMEbus-Modul. Diese sogenannten Catch-Module [27] lesen die Daten der CMCs aus und legen sie in einem Zwischenspeicher (Spy-Buffer) auf dem Catch-Modul ab.
Die Messung der Häufigkeit der Faseransprecher wird auf Basis der gleichen Technik durchgeführt. Die 250 Mhz-Zähler-CMCs [28] sind ebenfalls eine Entwicklung der Universität Freiburg; pro CMC stehen 32 Zähler à 32 Bit zur Verfügung.
Die Catch-Module können mit vier CMCs bestückt werden. Insgesamt werden 12 Catch-Module mit insgesamt 32 TDC-CMCs und 16 Zähler-CMCs eingesetzt. Die TDC-CMCs werden im höher auflösenden Modus (55 ps) betrieben, wodurch die doppelte Menge gegenüber den Zählern benötigt wird.

Die Daten werden nach jedem Trigger in einem Speicherbereich (Spy-Buffer) auf dem Catch-Modul abgelegt. Zusätzlich werden Statusinformationen mit den Daten in ein spezielles Format geschrieben:

0x0
S-Link header 1
S-Link header 2
S-Link header 3
CMC header 1
... n data words ...
CMC trailer 1
...
CMC header 4
... n data words ...
CMC trailer 4
0xcfed1200 (end marker)
31 ... 0

Die Spezifikation der Catch-Module gibt für deren VMEbus-Schnittstelle eine Datentransferrate von ca. 10 MB/s an. Dies entspricht einer Zykluszeit pro 32 Bit-Wort von ca. 400 ns. Die zusätzlichen Status-Wörter im Spy-Buffer erzeugen eine Grundauslesezeit t_{off} pro Catch von

$$t_{off} = 13 \cdot 400\ ns = 5,2\ \mu s$$

Die Multihit-TDC-Chips der CMCs schieben beim Eintreffen eines Pulses auf einem Eingang den Zählerstand einer Referenzuhr in einen Fifo. Wird der TDC für die Auslese getriggert, wird der Inhalt dieses FiFos in den Spy-Buffer kopiert. So können pro TDC mehrere 32Bit Datenwörter zur Verfügung stehen. In der folgende Tabelle sind ein paar Auslesezeiten für verschiedene Anzahlen an Gesamt-TDC-Treffern angegeben:

Anzahl Treffer	Zeit
10	$\sim$ 45,6 μs
20	$\sim$ 49,6 μs
30	$\sim$ 53,6 μs

Bei den CMC-Zählern ergibt sich eine feste Zeit:

$$t_{counter} = 225,6\ \mu s$$

Durch die langen Auslesezeiten der Zähler werden diese nicht mit jedem Ereignis ausgelesen.

Daten (DATA_EVENT):

szint. Latten	$\sim$ 196 μs
+ Drahtkammern	$\sim$ 22 μs
+ szint. Fasern (10 Treffer)	$\sim$ 50 μs
	$\sim$ 268 μs

Daten mit Zählerinformationen (SCALER_EVENT):

szint. Latten	$\sim$ 196 μs
+ Drahtkammern	$\sim$ 22 μs
+ Zähler Drahtkammern	$\sim$ 352 μs
+ szint. Fasern (10 Treffer)	$\sim$ 50 μs
+ Zähler szint. Fasern	$\sim$ 226 μs
	$\sim$ 846 μs

6.3 Der Innen-Detektor

Das Szintillationslicht der 512 Fasern des Innen-Detektors wird über 16fach segmentierte Photomultiplier in ein elektrisches Signal gewandelt und über einen Leading-Edge-Diskriminator der Firma SIS aufbereitet. Jedes Modul umfasst 16 Diskriminatoren, deren Signal an einem 16 Bit breiten ECL-Bus zur Verfügung steht. Diese logischen Informationen werden über 40 m lange TP-Kabel an FastBus-TDCs LeCroy 1875A weitergegeben und in Zeitinformationen umgewandelt. Die hier genutzten TDCs sind keine Multihit-TDCs, da die Erwartung der Raten in den einzelnen Fasern nicht zu hoch ist.
Zusätzlich liefert jedes Diskriminatormodul über jeweils einen NIM-Ausgang eine logische und eine analoge Summe. Aus diesen Informationen wird lagenweise eine analoge Summe gebildet und mit einem weiteren Diskriminator ein Signale für die Triggerlogik erzeugt. Hiermit ist es möglich, beim Erreichen einer gewissen Anzahl von angesprochenen Fasern ein Triggersignal zu erzeugen.
Die VMEbus-Diskriminatoren sind in direkter Nähe zum Innen-Detektor platziert. Über diese drei VMEbus-Crates sind die Diskriminatoren für die einzelnen Fasern verteilt. Die Diskriminatoren können über eine VMEbus-VMEbus Kopplung (CES VIC 8250) initialisiert und

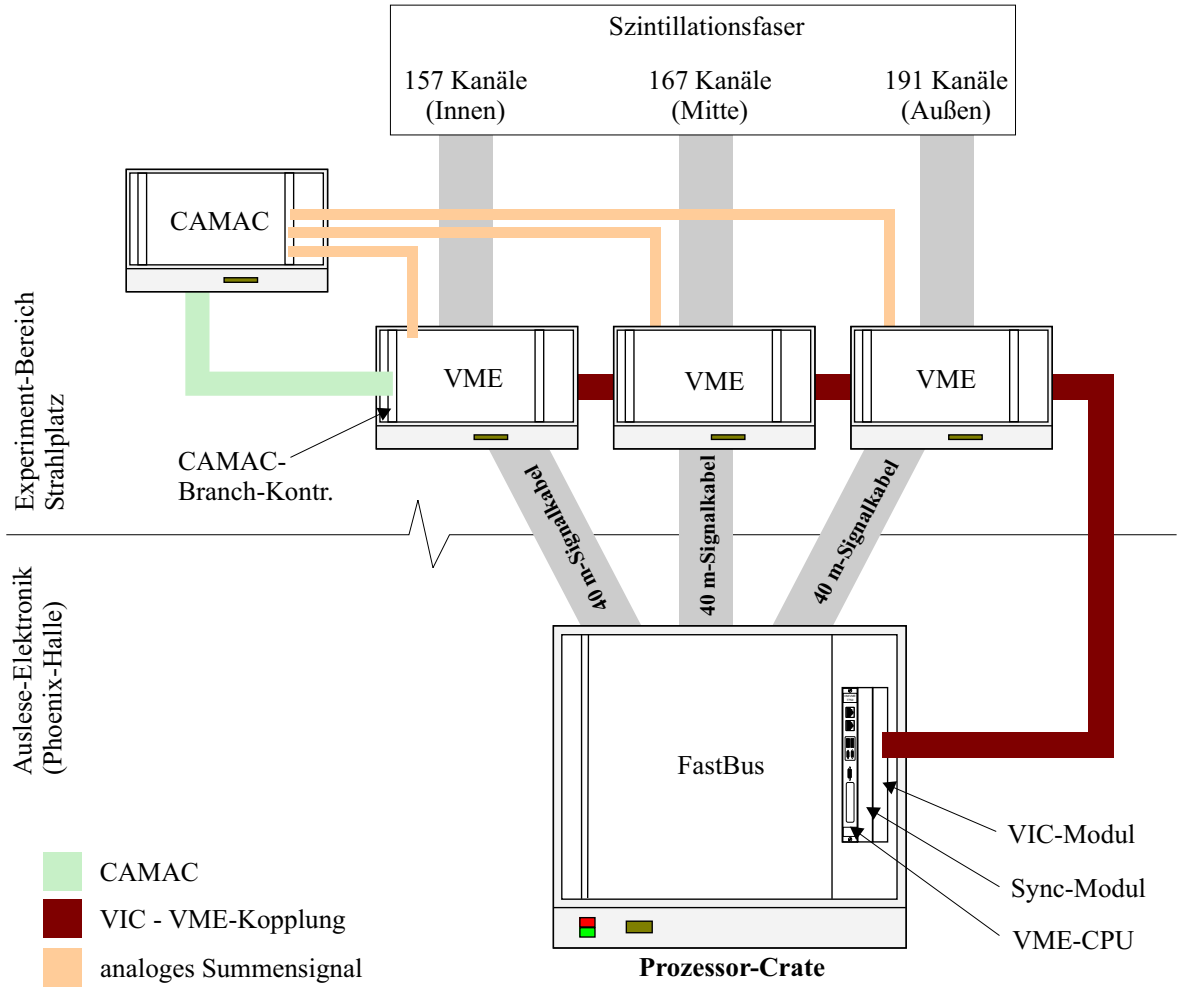


Abbildung 6.3: Komponenten des Innen-Detektors

gesetzt werden. Weiterhin befindet sich in einem der Diskriminatorcrates ein CAMAC-Branch-Kontroller. Über diesen Kontroller kann auf zwei ADC-Module zugegriffen werden, die zur Kontrolle der Photomultiplierverstärkung die analogen Summen digitalisieren.

Abbildung 6.3 zeigt schematisch die hierfür notwendigen Bus-Systeme. Der Übergang zwischen VMEbus und FastBus ist mit einem Sequenzer der Firma Struck STR340/SFI [30] realisiert. Diese "bridge" hat die Eigenschaft, Aktionen, die auf dem FastBus ausgeführt werden sollen, über programmierbare Listen sequentiell autonom abzuarbeiten und die Daten per DMA in einem Speicherbereich abzulegen. Der Sequenzer verfügt zusätzlich noch über einen drei-Slot-breiten VMEbus, welcher die Prozessorkarte, das Sync-Clientmodul und ein VIC-Modul zur VMEbus-VMEbus Kopplung aufnimmt.
Der Transfer der Daten von den FastBus-Modulen in den Speicher der Prozessorkarte wird über den Sequenzer ausgeführt. Es werden acht TDC-Module mit je 64 Kanälen ausgelesen. Nimmt man eine Transferzeit eines Kanals von ca. 200ns an, ergibt diese eine Gesamtzeit von

Anzahl	Bits	Beschreibung
512	12	TDC Wert
33	12	ADC Wert

Tabelle 6.1: Rohdaten des Innen-Detektors (Beispieldaten siehe B.2)

$t = 512 \cdot 0,2\ \mu s = 102,4\ \mu s$.
Für die Transferzeit der CAMAC-ADCs ergibt sich $33 \cdot 3,5\ \mu s = 115,5\ \mu s$.

Daten(DATA_EVENT):

$$\begin{array}{rl} \text{TDCs} & \sim 102{,}4\ \mu s \\ + \text{ ADCs} & \sim 115{,}5\ \mu s \\ \hline & \sim 217{,}9\ \mu s \end{array}$$

6.4 Der Crystal-Barrel-Detektor

Der Crystal-Barrel besteht aus 1380 (1290) CsI(Tl)-Kristallen, die auf zwei Hälften mit jeweils einem eigenen Ausleseprozessor aufgeteilt sind. Das Signal der Photodiode wird über einen Vorverstärker mittels eines Twisted-Pair Kabels über 40 m aus dem Experiment in die Elektronik-Halle zu vier Racks mit Shapermodulen geführt. Diese Module bereiten das Signal für die Auslese auf und splitten das Signal. Eine Abzweigung führt auf zwei Crates mit FAST-Bus ADC-Modulen ([21]), die andere geht auf Diskriminatoren für den Fast-Cluster-Encoder (FACE [20]).

Die Wandlung erfolgt über FASTBus-Module der Firma LeCroy (LeCroy 1882A). Je Einschub stehen 96 Kanäle zur Verfügung. Dies ergibt 8 Module pro Crystal-Barrel-Hälfte und zwei FastBus-Crates. Wie beim Innen-Detektor wird der Zugang zu den FastBus-Modulen über einen FastBus-Sequenzer durchgeführt. Jedoch kommt hier eine erweiterte Version des Sequenzer zum Einsatz. Der FastBus-Sequenzer 4100 NGF der Firma SIS bietet eine weitere Option, der eine Reduzierung der Prozessorlast ermöglicht. Er ist in der Lage, während des Transfers der Daten von den ADC-Modulen in den Speicher des Prozessors nur ADC-Kanäle zu transferieren, die über einer gewissen Schwelle liegen. Diese Schwellen können für jeden ADC-Kanal im Sequenzer gespeichert werden.
Unter der Annahme, dass die Transferzeit im gleichen Rahmen wie beim Innen-Detektor liegt, erhält man eine Transferzeit von

$$t = 765 \cdot 0,2\ \mu s = 153\ \mu s$$

Aufgrund des Design der ADCs müssen allerdings auch nicht belegte Kanäle gelesen werden. Somit ergeben sich feste Auslesezeiten, unabhängig davon, ob die Daten "Nullen"-unterdrückt

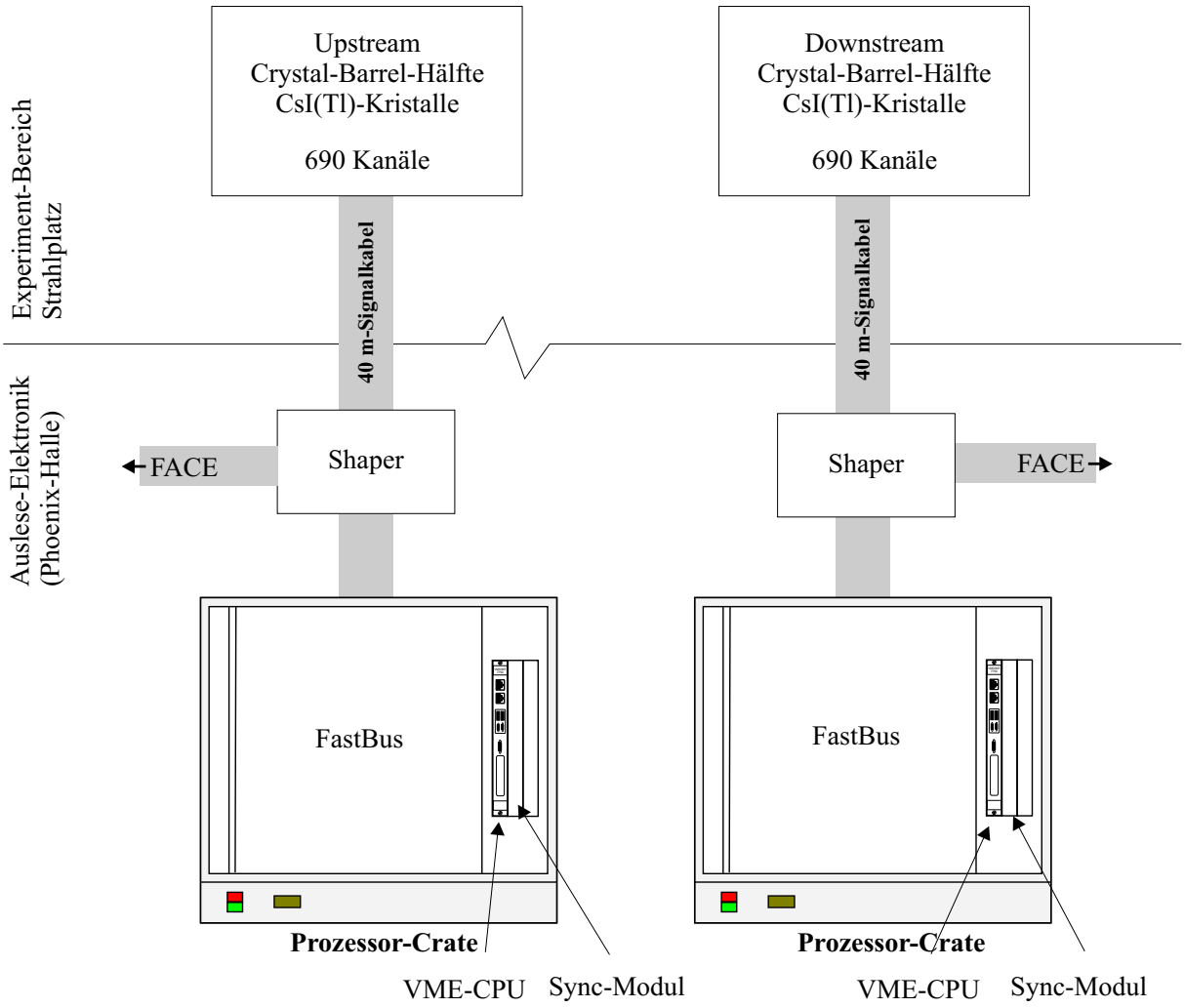


Abbildung 6.4: Komponenten des Crystal-Barrel

weggeschrieben wurden oder nicht.

6.5 Der TAPS-Detektor

Bei der Erweiterung um den TAPS-Detektor handelt es sich um einen temporären Aufbau, sodass hier eine spezielle Situation vorliegt.

Der TAPS-Detektor[18] ist durch den Einsatz an vorherigen anderen Strahlplätzen mit einer eigenen DAQ ausgestattet. Diese besteht aus n CAMAC-Crates. Jedes Crate ist wiederum mit einem aktiven Crate-Kontroller ausgestattet, der die Auslese der Module des jeweiligen Crates übernimmt. Ein VSB-Bus verbindet alle CAMAC-Crate-Kontroller mit einer ELTEC-E6 CPU (siehe Abbildung 6.5). Jeder CAMAC-Crate-Kontroller führt autonom die Auslese seines Crates parallel zu den anderen aus. Zusätzlich wird noch eine Reduktion der Daten durchgeführt, wobei nur Daten übertragen werden, die über einem gewissen Schwellwert liegen. Die Prozessorkarte

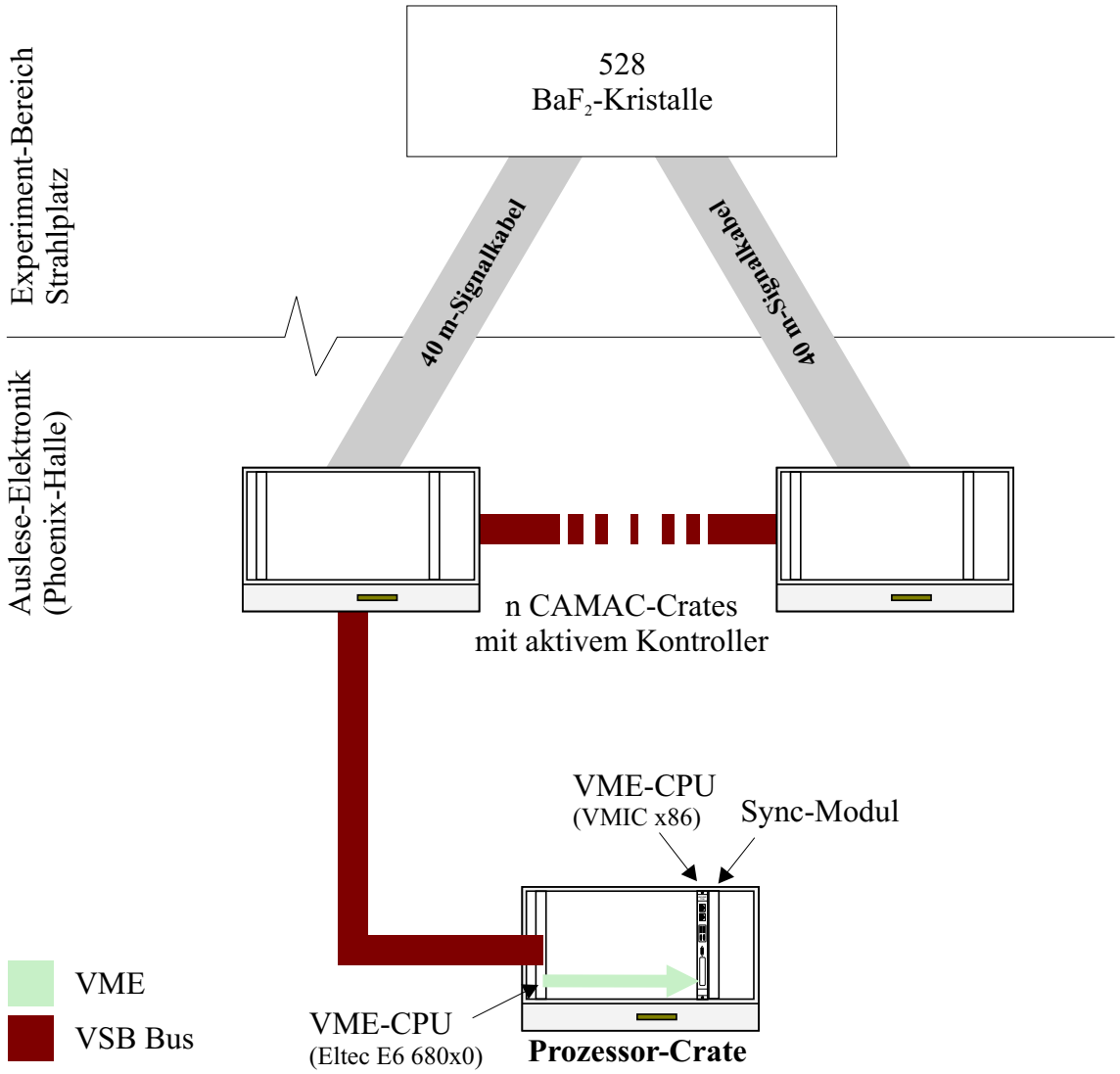


Abbildung 6.5: TAPS-Kopplung

kopiert sequentiell die Daten der Crate-Kontroller über den VSB-Bus in den Speicher.

Die Verwendung dieses kompletten DAQ-Systems im Rahmen des Gesamt-Experiments setzte lediglich die Kopplung der Datenströme aus der CB-DAQ und der TAPS-DAQ voraus, um auf einem gemeinsamen Datenspeicherprozessor die vollständigen Eventinformationen zur Verfügung zu haben.

Im Hauptprozessorcrate der TAPS-DAQ wurde eine weitere CPU (VMIC x86) mit einem Sync-Clientmodul aus der CB-DAQ installiert. Um eine Kopplung der beiden Systeme zu erreichen, wurde ein Modus entwickelt, in dem die TAPS-DAQ von der Crystal-Barrel CPU gesteuert werden kann. Die Daten werden von einem Prozess auf der E6 in einen Speicherbereich der Crystal-Barrel CPU geschrieben, der dann die Weiterverarbeitung der Daten mittels DMA durchführt. Der Zugriff der E6 erfolgt über den VMEbus auf die Crystal-Barrel CPU. Eine

detailliertere Beschreibung des Ablaufs erfolgt in Kapitel 7.3.5.

6.6 Die Flugzeitwand - ToF

Die Flugzeitwand (ToF) besteht aus 4x14 Szintillationslatten. Da die Lichtsignale der Szintillationslatten auf beiden Seiten durch Photomultiplier gewandelt werden, entstehen 112 analoge Signale, welche weiterverarbeitet werden müssen. Die Aufbereitung der Signale erfolgt in der

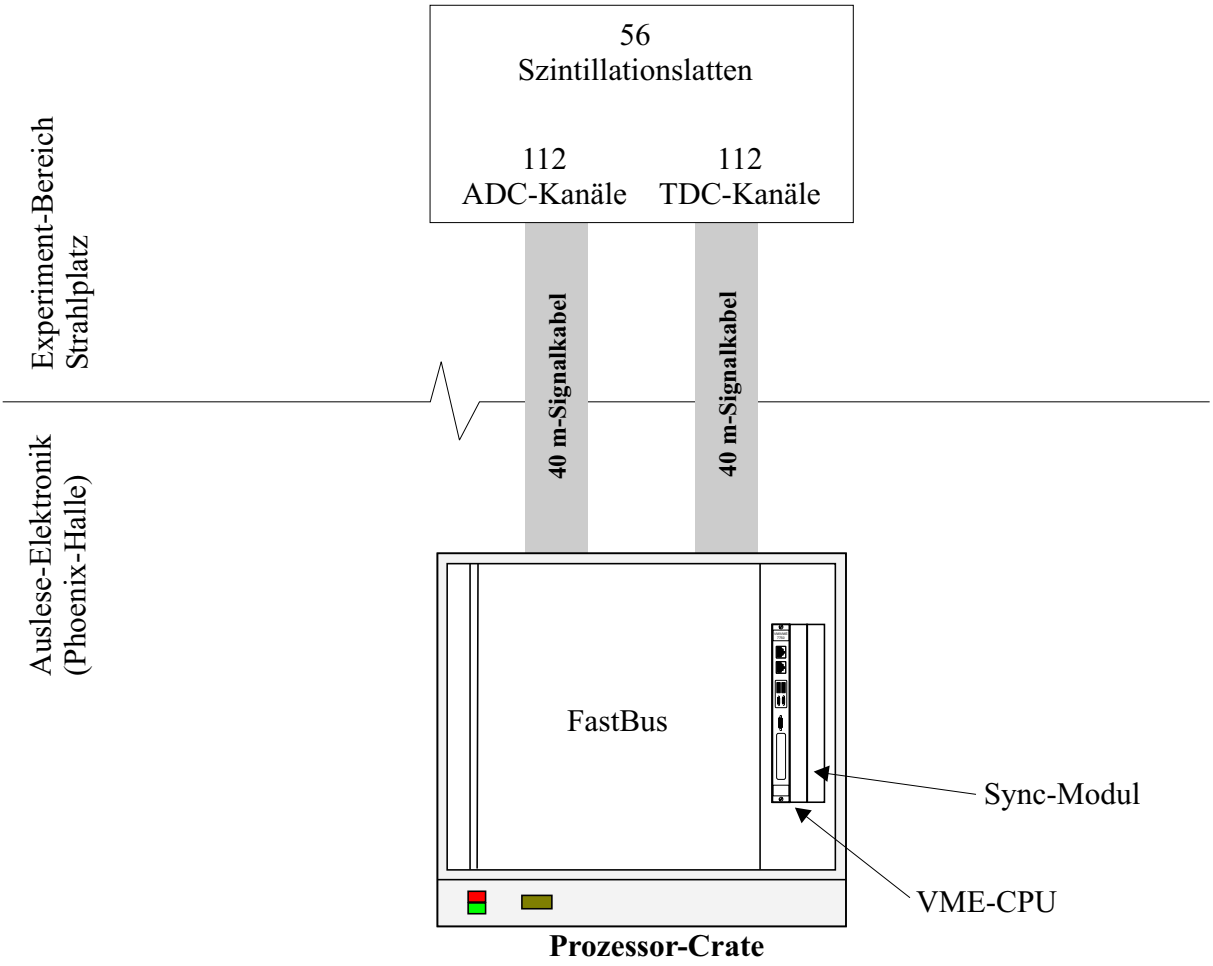


Abbildung 6.6: Komponenten der Flugzeitwand

Nähe der Wand, da auch Trigger-Signale vor Ort erzeugt werden müssen. Zur Bestimmung der Flugzeit und der Identifikation von Teilchen wird jedes analoge Signal in zwei Signale aufgeteilt, die über 40 m-Kabel in die Elektronik-Halle geleitet werden.

Die Signale werden von zwei Fastbus LeCroy 1875A ADCs und LeCroy 1875A TDCs digitalisiert. Pro ADC-Modul stehen 96 Kanäle und pro TDC 64 Kanäle zur Verfügung. Auch hier

sind die ADCs und TDCs nicht voll belegt. Dennoch müssen aber alle Kanäle gelesen werden:

$$t_{ADC} = 192 \cdot 0,2\ \mu s = 38,4\ \mu s$$

$$t_{TDC} = 128 \cdot 0,2\ \mu s = 25,6\ \mu s$$

Durch eine Filterung der vom Sequenzer übertragenen Daten werden nur die TDC-Kanäle mit einem Treffer zur späteren Verarbeitung vom Prozessor kopiert.

Anzahl	Bits	Beschreibung
112	12	ADC Wert
112	12	TDC Wert

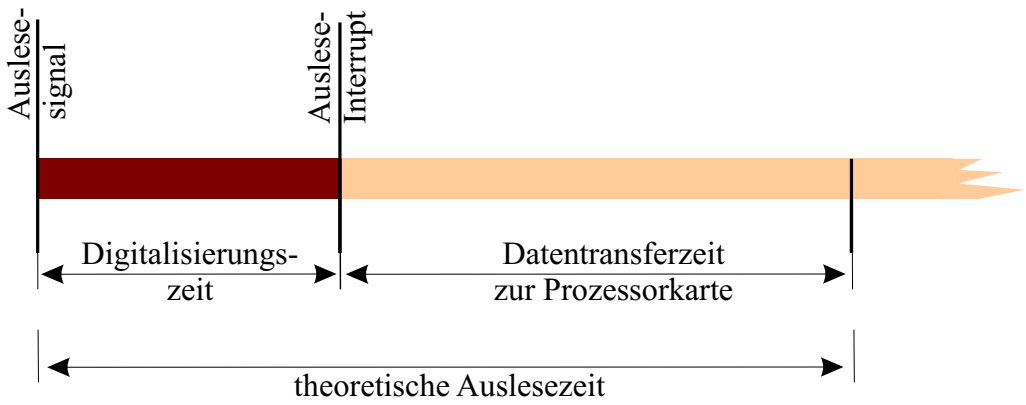
Tabelle 6.2: Rohdaten des ToF (Beispieldaten siehe B.4)

Daten(DATA_EVENT):

$$\begin{aligned} \text{TDCs} &\sim 25{,}6\ \mu s \\ +\ \text{ADCs} &\sim 38{,}4\ \mu s \\ \hline &\sim 64\ \mu s \end{aligned}$$

6.7 Zusammenfassung der Hardwareperformance

Im folgenden sind die theoretisch möglichen Auslesezeiten der Frontend-Elektronik-Module noch einmal zusammenfassend aufgeführt. Diese Zeiten beinhalten den Transfer vom Frontend-Modul bis in den Hauptspeicher der Prozessorkarte und die Digitalisierungszeit der Frontend-Module.



Subdetektor	theo. Auslesezeit
Trigger-/Sync-Kontroller	$\sim 455{,}5 \mu s$
Tagging-System	$\sim 268\ \mu s$
Innen-Detektor	$\sim 217{,}9\ \mu s$
Crystal-Barrel 1	$\sim 453\ \mu s$
Crystal-Barrel 2	$\sim 453\ \mu s$
Flugzeitwand	$\sim 64\ \mu s$
TAPS-Detektor	$\sim 600\ \mu s$

Durch die Parallelität der Auslese aller Subdetektoren gibt die langsamste Komponente die maximal mögliche Ausleserate an. Dies bedeutet, dass das System durch die Auslesegeschwindigkeit des TAPS-Detektors gegeben ist. Somit ergibt sich eine maximale Rate von

$$f_{DAQ} = \frac{1}{600\ \mu s} \approx 1,6\ kHz.$$

Ausgehend von diesen durch die Hardware definierten Auslese- und Datenreduktionszeiten besteht nun die Aufgabe, eine strukturierte Auslesesoftware zu erstellen, die sowohl die Synchronisationsaufgabe als auch den weiteren Datentransfer zu einem Datenprozessor für die Strukturierung und Speicherung der Eventinformationen bewerkstelligt.
Die sich daraus ergebenden zusätzlichen Verarbeitungszeiten sollten auf ein Minimum reduziert werden, um der theoretischen Datenakquisitionsgeschwindigkeit so nah wie möglich zu kommen. Die gewählte Realisierung wird im folgenden Kapitel ausführlich erläutert.

Kapitel 7

Die Software des Datenauslesesystems

Die gestellten Ziele sind nur bei Verwendung moderner Programmierparadigmen zu bewältigen. Dazu zählen

- Objektorientierte Programmierung, Modularität der Software
- Verwendung von Systemstandards im Netzwerkkommunikationsbereich
- Verwendung von Standardbibliotheken in der Interprozesskommunikation
- Ausnutzung der bereitstehenden Funktionalität des Betriebssystems und im Datenbankbereich

Dabei war immer zu prüfen, ob die gewählten Standard-Methoden und die Konzepte der gestellten Aufgabe eines maximalen Durchsatzes gerecht wurden, oder ob durch eine begrenzte Eigenentwicklung bestimmter Funktionalitäten nicht eine Leistungsverbesserung erreicht werden konnte.

Ein Ziel bei der Neuentwicklung war, dass die Software modernen Programmierparadigmen (wie objektorientierte Programmierung OOP, ...) genügt und eine modulare Struktur aufweist, die es ermöglicht, Erweiterungen leicht einzufügen. In der objektorientierten Programmierung wird die Realisierung eines Problemes mittels Objekten durchgeführt, d.h. es werden Einheiten gebildet, die mit bestimmten Eigenschaften versehen sind. Das bedeutet, dass das Problem in Unterbausteine (Blackboxen) zerlegt wird und diese über definierte Steckverbindungen (Schnittstellen) miteinander verbunden werden. Hiermit erreicht man eine leichte Austauschbarkeit von Bausteinen und hat die Möglichkeit, diese in einer Testumgebung auf ihr korrektes Verhalten zu prüfen und zu optimieren. Es sinkt die Fehleranfälligkeit, da die Bausteine vor der Konsolidierung zum Gesamtobjekt geprüft werden. Weiterhin eröffnet es die Möglichkeit, die Entwicklung des Systems in Bereiche aufzuteilen, die von unterschiedlichen Personen parallel

entwickelt und gepflegt werden können. Es ist jedoch bei der Entwicklung von objektorientierten Programmen darauf zu achten, dass bei einer tiefen Schachtelung der Klassenstrukturen ein vergrößerter Software-Overhead (Funktionsaufrufe) entstehen kann, der in zeitkritischen Programmabläufen gegenüber einer "linearen" Programmierung von Nachteil sein kann. Moderne objektorientierte Programmiersprachen stellen hierfür Techniken (Inline-Deklaration,...) zur Verfügung, um diesen Nachteilen vorzubeugen.
Die Verwendung der zahlreichen Standardbibliotheken in den Bereichen Netzwerk, Betriebssystem oder Datenbankfunktionen führt zu einer essentiellen Entlastung im Programmieraufwand, da die dort zur Verfügung stehende Funktionalität Eigenentwicklungen überflüssig macht. Ein durchaus nicht zu unterschätzendes Risiko ist allerdings die begrenzte zeitliche Verfügbarkeit dieser Bibliotheken in ihrem Softwareentwicklungsumfeld (Abhängigkeiten zum Kompiler- oder Betriebssystemversionen), die eine nachträgliche Anpassung unmöglich machen können, wenn sie nicht im Quellcode zur Verfügung stehen. Auch die Möglichkeit der Fehlersuche ist bei ihrer Verwendung auf Grund ihres "black box"-Charakters drastisch eingeschränkt. Dies machte es notwendig im Rahmen der hier vorliegenden Entwicklungsarbeit, die Verwendung einer Kommunikationsbibliothek wieder aufzugeben, da auf Grund von offensichtlichen internen Problemen eine stabile Operation dieses Programmteils nicht zu erreichen war. Ein weiterer Punkt in der Entwicklung des Systems war es, möglichst viele Standardbibliotheken zu nutzen und so weit wie möglich auf Eigenentwicklungen zu verzichten, d.h. in den Bereichen Netzwerk-, Betriebssystem- oder Datenbankfunktionen auf Standards zurückzugreifen. Die verschiedenen Subdetektoren erfordern unterschiedliche Auslesesequenzen, die durch den heterogenen Hardwareaufbau bedingt sind. Eine der ersten Aufgaben war daher, die unterschiedlichen Anforderungen auf ihre strukturelle Modularität hin zu untersuchen, um gemeinsame Funktionalitäten in einem Basisprogramm zusammenzuführen (Synchronisation, Datenbufferhandling,...), und somit den Programmier- und Testaufwand drastisch zu reduzieren.

Im folgenden werden die Programmabläufe und ihre modulare Realisierung im Detail vorgestellt. Im Gegensatz zur Anpassung der Frontend-Hardware wurde die Software für das CB-ELSA-Experiment vollständig neu konzipiert und implementiert, da der Übergang zu einer völlig veränderten Rechnerarchitektur und zu einem modernen Betriebssystem die direkte Anpassung der bisherigen Programme ausschloss. Um die Strukturen und Vorgänge in der Software besser zu erläutern, wird in den weiteren Abschnitten intensiv die Darstellung der Abläufe und Strukturen per Unified Modelling Language genutzt. Die Unified Modelling Language (UML) ist eine Sprache zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme, Geschäftsmodelle und andere Nicht-Softwaresysteme. Sie bietet den Entwicklern die Möglichkeit, den Entwurf und die Entwicklung von Softwaremodellen auf einheitlicher Basis zu diskutieren. Weiterhin ermöglicht es, die Zusammenhänge und Interaktionen zwischen den einzelnen Komponenten grafisch darzustellen.

7.1 Einführung

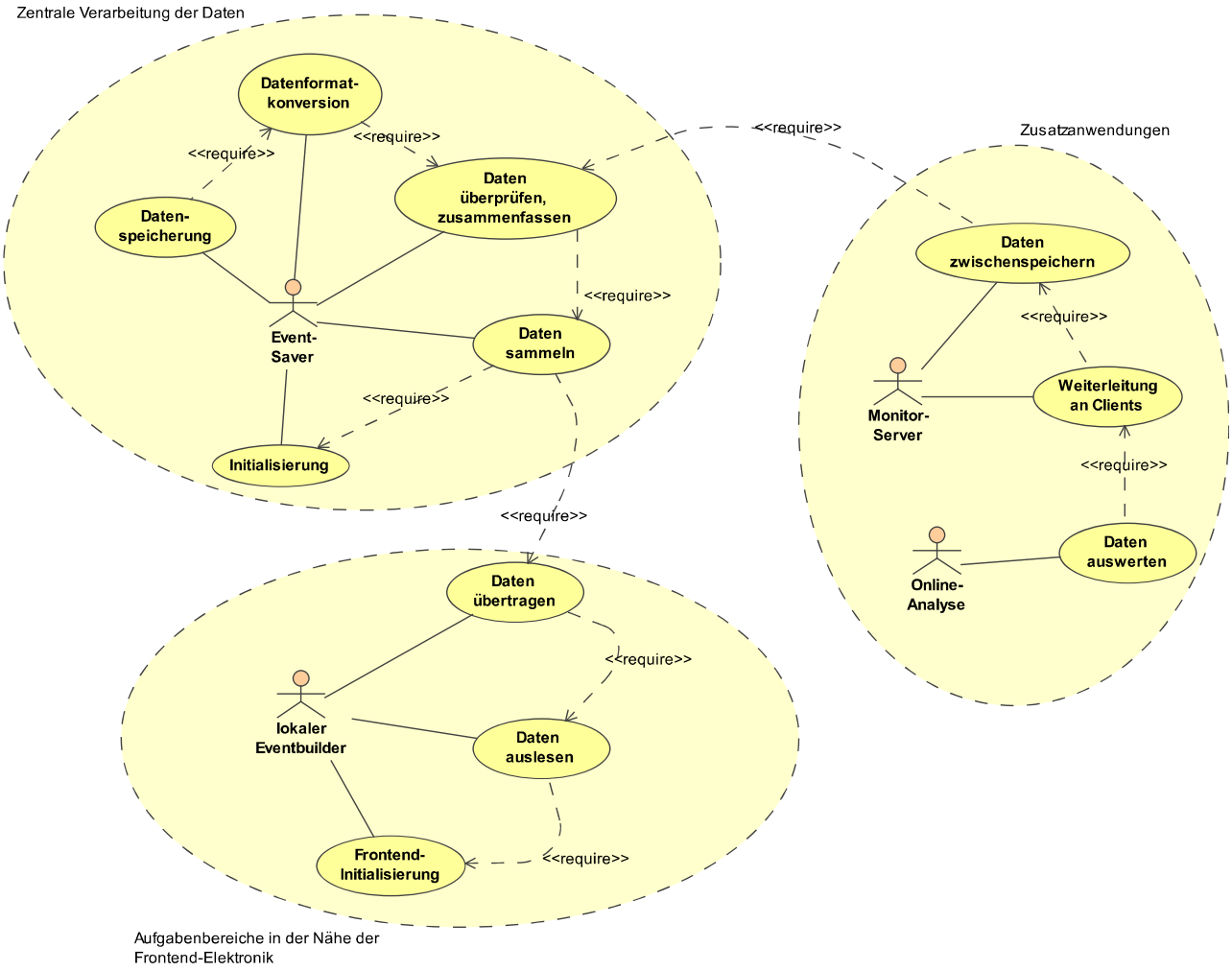


Abbildung 7.1: Use-Case-Diagramm des Datennahmesystems

Im Kapitel 4 wurde bereits die Struktur der Hardware-Komponenten beschrieben. Diese Aufteilung spiegelt sich auch in dem Use-Case-Diagramm aus Abbildung 7.1 wieder. Zur Vereinfachung sind ein Szenario aus einem lokalen Eventbuilder und die wesentlichen Aufgaben zur Ausführung der Datennahme dargestellt.
Die Datennahme teilt sich in drei große logische Bereiche auf:

1. Experimentbereich (Frontend-Elektronik,...)
2. Datenspeicherbereich (Datenverarbeitung, Überprüfung, Speicherung,...)
3. Kontrollbereich (Exp.-Überwachung,...)

Der Experimentbereich umfasst die lokalen Eventbuilder, die die Intialisierung, die Auslese und den Datentransport zum Event-Saver übernehmen. Die Aufgaben des Event-Savers sind

das Zusammenfassen der Daten der lokalen Eventbuilder, die Datenkontrolle und die Speicherung auf einem Medium. Die Datenkontrolle im Event-Saver beinhaltet nur die Gewährleistung der Datensynchronität, d.h. die Sicherstellung, dass die einzelnen Datensegmente der lokalen Eventbuilder zueinander passen. Da eine Kontrolle der Performance des Detektors schon während der Datennahme zwingend notwendig ist, existiert ein weiterer Bereich zur Kontrolle der ausgelesenen Daten. Hier können z.B. die ADC- und TDC-Werte grafisch histogrammiert werden und in Echtzeit aus dem laufenden Experiment heraus betrachtet werden. Dies erlaubt eine sofortige Erkennung von Fehlfunktionen bei der Detektor-Auslese oder in der Elektronik.

Die Abbildung 7.2 zeigt die drei Bereiche aus einem anderen Blickwinkel mit Schwerpunkt auf Daten- und Kontrollverbindungen, da diese Prozesse auf einem verteilten Rechnersystem ablaufen und somit über die inneren Prozesskommunikationsmöglichkeiten eines Rechners hinaus-

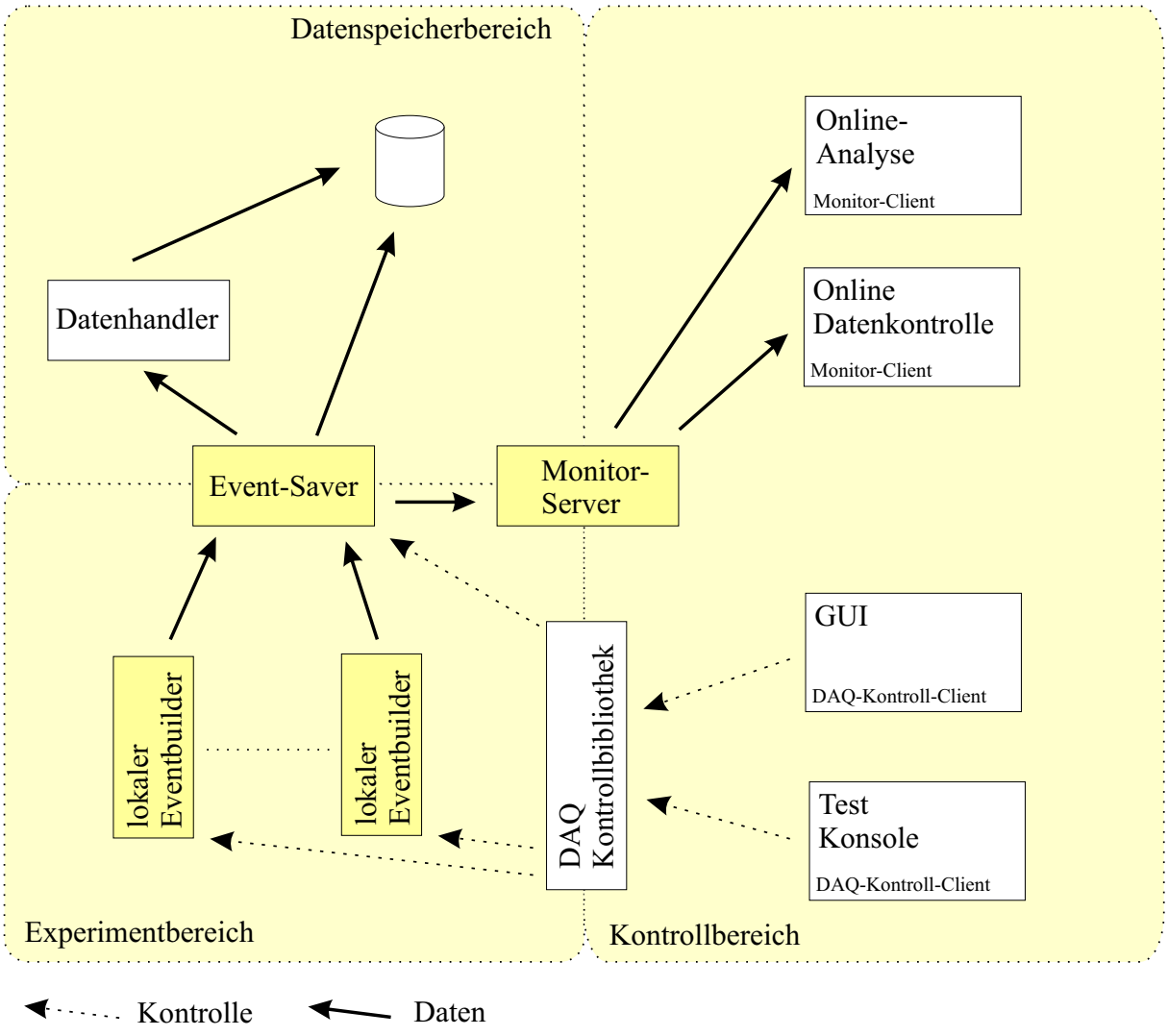


Abbildung 7.2: Softwarestruktur des Datenaquisitionssystems

reichen. Dies erfordert ein hohes Maß an Prozessstabilität und Kommunikation, um in einem solchen verteilten System alle möglichen Fehlerverhalten zu kontrollieren (z.B. Ausfall eines Subprozessors, der zur Auslese aktiviert ist). Die Prozesse in den verschiedenen Bereichen werden von einem zentralen Punkt aus koordiniert. Die DAQ-Kontrollbibliothek übernimmt diese Aufgabe der Kontrolle über die lokalen Eventbuilder und den Event-Saver. Sie übernimmt die Initiierung der externen Datenverbindungen von den lokalen Eventbuildern zum Event-Saver und die Aufgabe, das komplette Datennahme-System mit der Startkonfiguration zu versorgen, zu starten bzw. zu stoppen.

Die nächsten Abschnitte sollen diese drei Bereiche und ihre Einzelheiten näher erläutern, wobei die Reihenfolge der Abschnitte sich am Weg der Daten vom Detektor bis hin zur Datenspeicherung orientiert. Zu Beginn werden der lokale Eventbuilder im allgemeinen (Ausleseschnittstelle, Datentransport,...) sowie die jeweiligen Spezialisierungen der lokalen Eventbuilder für die Frontend-Elektronik erläutert. Im darauffolgenden Abschnitt wird dann die Struktur (Datenhandling, Datenbankanbindung, ...) des Event-Savers dargestellt. In den anschliessenden Abschnitten schließlich werden die DAQ-Kontrollbibliothek (Start/Stop-Abläufe, ...) und die Bibliothek zur Online-Datenüberwachung dargelegt.

7.2 Frontend-Auslese - Der lokale Eventbuilder

Durch den parallelen Aufbau der Frontend-Elektronik ergibt sich eine Aufteilung der Aufgabenlast auf mehrere Rechnersysteme. Die Prozesse auf den Rechnersystemen (lokale Eventbuilder) übernehmen die Steuerung und die Auslese der mit ihnen verbundenen Hardwarekomponenten, wie das Use-Case-Diagramm in Abbildung 7.3 zeigt.

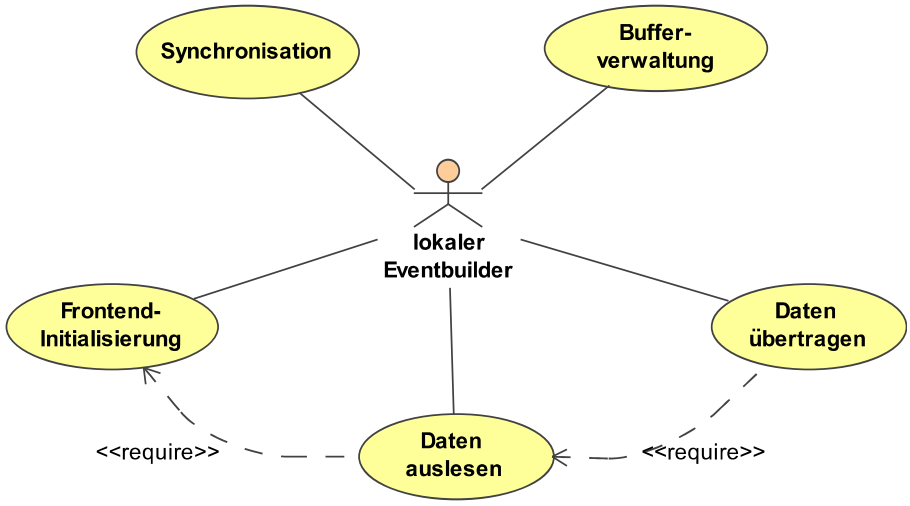


Abbildung 7.3: Use-Case-Diagramm des lokalen Eventbuilder

Die Anforderungen an den Hauptprozess sind als erstes die Steuerung und die Auslese der Frontend-Elektronik. Dies erfordert bei manchen Detektorkomponenten eine auf die Gegebenheiten ausgerichtete Implementation. Es handelt sich hierbei um individuelle Implementierungen, um besondere Eigenschaften der Hardware zur Steigerung der Auslesegeschwindigkeit auszunutzen (siehe Implementation beim Crystal-Barrel).

Eine weitere Aufgabe ist beschränkt auf die Verarbeitung der gewonnenen Daten und ihren schnellen Abtransport. Im allgemeinen wird ein Datenbereich einer gewissen Größe bewegt, wobei der Aufbau der Daten nicht bekannt sein muss. Dies ermöglicht es, den Aufbau einer Grundfunktionalität für alle lokalen Eventbuilder zu erstellen.

In Abbildung 7.4 sind eine Feingliederung der Funktionsbereiche und die Aufteilung in einen allgemeinen und einen speziellen Teil gezeigt. Das Basismodul umfasst die allgemein einsetzbaren Funktionen wie Datentransport, Kontrollfunktionen, usw. Der individuell auf die Frontend-Elektronik angepasste Bereich ist nicht Teil des Basismoduls. Diese Verbindung (Schnittstelle) zwischen dem allgemeinen und spezialisierten Bereich muss die Möglichkeit geben, auch besondere Fähigkeiten der Frontend-Elektronik ausnutzen zu können.

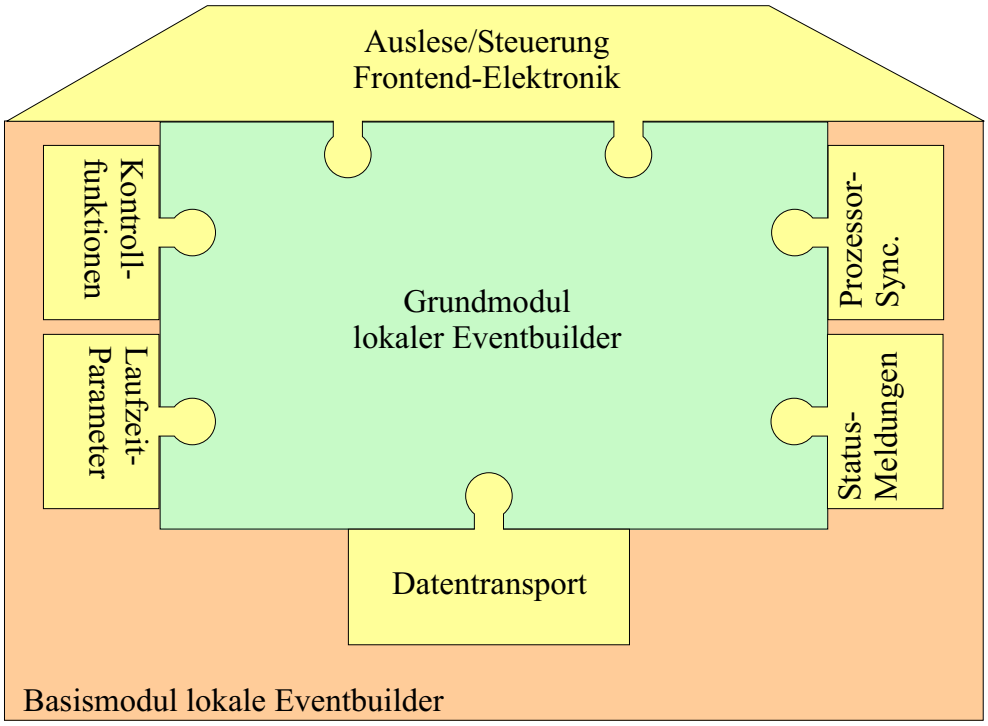


Abbildung 7.4: Puzzle-Darstellung des lokalen Eventbuilder

Die schematische Ansicht in Abbildung 7.4 zeigt noch nicht die im Basismodul verwendeten Klassen und Strukturen. Eine vollständige Übersicht gibt das statische UML-Diagramm in Abbildung 7.5 wieder. Hier werden die Beziehungen und Abhängigkeiten sichtbar.

Zentraler Punkt ist das Hauptobjekt `baselevb` (Es ist im Diagramm als Objekt dargestellt, ist in der Implementation aber die `main()`-Funktion). Dieses Objekt ist zuständig für die Erzeugung und Initialisierung aller Objekte im lokalen Eventbuilder, wie die Beziehungspfeile im Diagramm es zeigen. Beim Start werden alle benötigten Instanzen erzeugt, und die notwendigen Datenstrukturen werden initialisiert. Hierzu gehört ein Objekt für die Synchronisationskontrolle, welches von der Schnittstelle `IControlClient` abgeleitet sein muss. Weiterhin wird ein Objekt angelegt, das die Schnittstelle `CBaseLogger` implementiert. Dieses ermöglicht die Weitergabe von Statusmeldungen in den einzelnen Umgebungen (z.B. Test/Entwicklungsumgebung und Produktivsystem). Ein weiterer Bereich sind die Verwaltung und Erzeugung von `CControlSession`s, die eine Steuerung des lokalen Eventbuilders von außen ermöglichen sollen. Sie dienen zur Kontrolle der Verbindung zwischen lokalem Prozess und der DAQ-Kontrollbibliothek und der Weitergabe von Kommandos an einen Kommandoprozessor. Mittels dieser Kommandos können unter anderem Konfigurationen gesetzt oder Datenverbindungen aufgebaut werden.

Ein weiterer sehr wichtiger Bestandteil ist der Transport der Rohdaten, die von der Auslese-Schnittstelle (`IReadout`) geliefert werden. Die Objekte `CPointerBuffer` und `CRawDatabuffer`

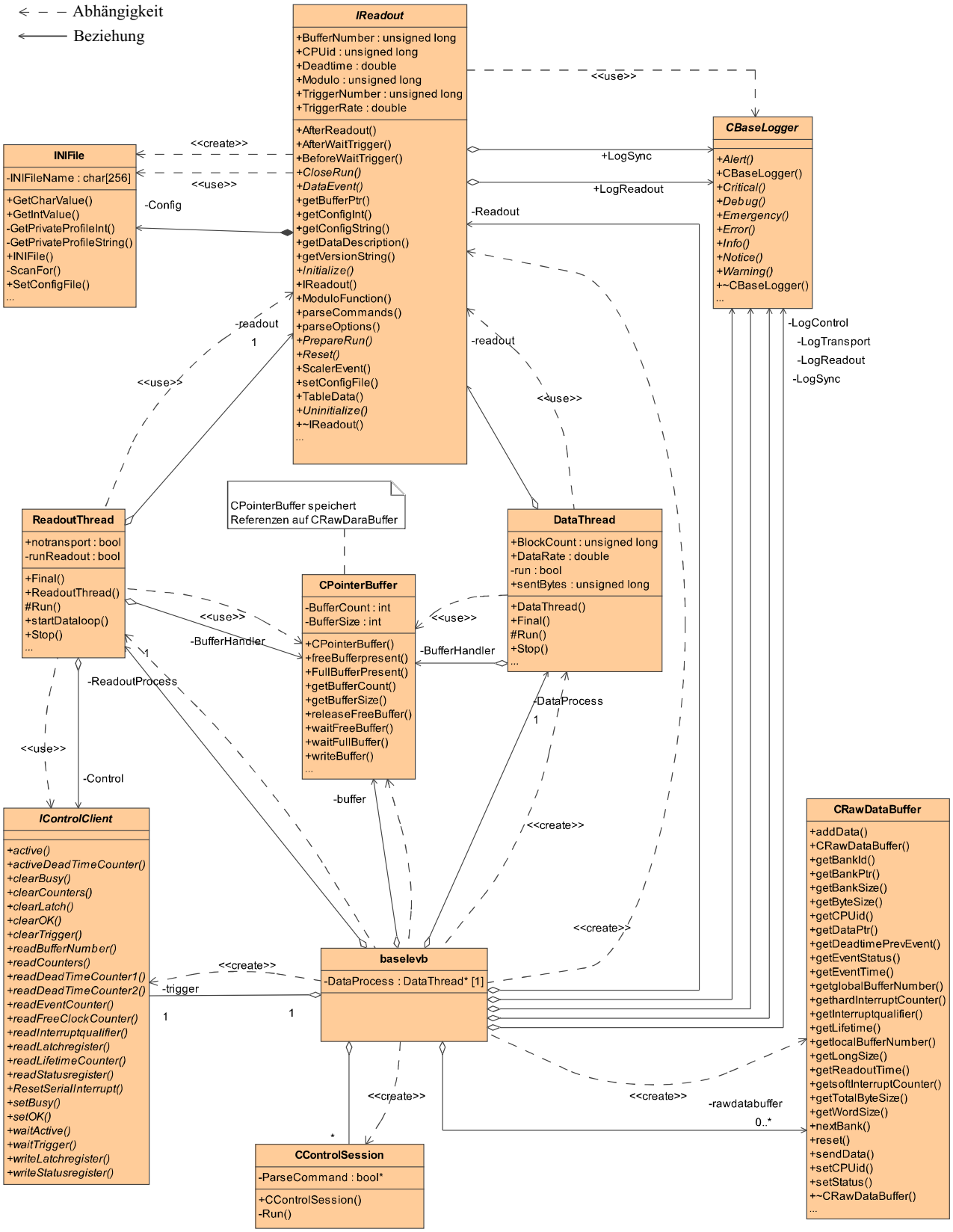


Abbildung 7.5: Statisches UML-Diagramm des lokalen Eventbuilder

bilden die Objekte zur Rohdatenverwaltung. Die Klasse `CPointerBuffer` repräsentiert den Transportkanal zwischen dem Ausleseprozess (`ReadoutThread`) und dem Datentransportprozess (`DataThread`). Die Klasse `CRawDatabuffer` ist der Kontainerdatentyp für die ausgelesenen Rohdaten. Die Instanzen dieser Buffer werden in `baselevb` erzeugt und gespeichert. Im späteren Verlauf werden nur noch Referenzen (Zeiger) an die Klasse `CPointerbuffer` übergeben und verwendet.
Die Klasse `IReadout`, die im oberen Teil des Diagramms zu sehen ist, stellt die Implementierung der Frontend-Elektronik-Auslese dar, d.h. aus dem Grundmodul heraus können nur die Funktionen dieser Klasse ausgeführt werden, was einer Standardisierung der Schnittstelle entspricht. Die Implementation der einzelnen Funktionen ist dann individuell und auslesespezifisch.

In den folgenden Abschnitten werden jetzt die genannten Bereiche und deren Objekte näher erläutert.

7.2.1 Die Ausleseschnittstelle - `IReadout`

Die Klasse `IReadout` umfasst die notwendigen Funktionen für den Betrieb eines lokalen Eventbuilders. Jeder lokale Eventbuilder muss von dieser Schnittstelle abgeleitet sein und die notwendigen Funktionen implementieren. Ein Hauptziel dieser Schnittstelle ist es, dem Programmierer Vorgaben an die Hand zu geben, die eine einfache und schnelle Implementierung eines

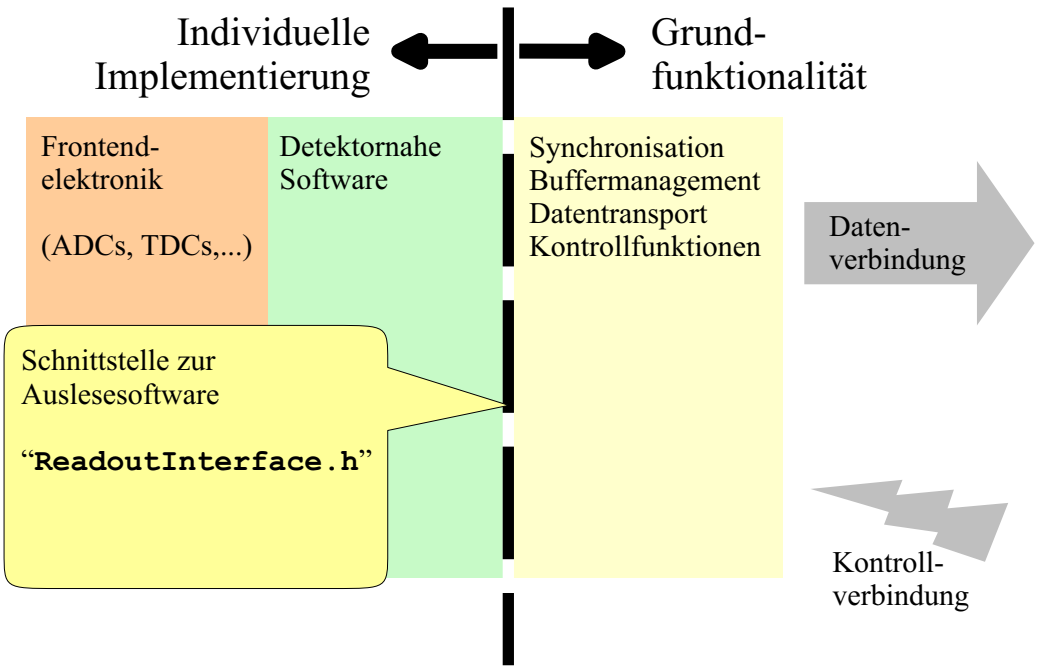


Abbildung 7.6: Grundstruktur der lokalen Eventbuilder

lokalen Eventbuilders ermöglichen sollen. Weiterhin muss sie allgemeingültig genug sein, um auch mögliche Spezialfälle mit einbeziehen zu können. Die Schnittstelle grenzt die Individual-Implementierung von der Grundfunktionalität im Aufbau eines lokalen Eventbuilders ab (siehe Abbildung 7.6). Der Programmier benötigt für eine Implementierung der Auslese nur die Kenntnis der zur Verfügung stehenden Funktionen, ohne die Details in den anderen Bereichen zu kennen.
Diese Trennung zwischen der individuellen Implementierung und der Grundfunktionalität wird in der Definitionsdatei "`ReadoutInterface.h`" festgelegt. Sie definiert zum einen die Funktionen, die zur Ausführung eines lokalen Eventbuilders **notwendig** sind, und zum anderen stellt sie Zusatzfunktionen zu Service-Einrichtungen (z.B. Statusmeldungen, Konfigurationsdateien,...) bereit. So implementiert das Objekt `INIFile` die Funktionalität von Konfigurationsdateien für Daten, die im Bereich der Initialisierung der Frontendelektronik notwendig sein könnten.

Im folgenden Quelltext soll die einfache Anwendung der Schnittstellenklasse `IReadout` verdeutlicht werden:

```
#include "baselevb.h"

class SampleReadout : public IReadout {
public:
   SampleReadout(int id) : IReadout(id) {
   };

   ReadoutException *PrepareRun(unsigned long runNr) {
      return NOEXCEPTION;
   };

   void CloseRun() {
   };

   ReadoutException *DataEvent(CRawDataBuffer *buf) {
      return NOEXCEPTION;
   };
};

INSTANCEREADOUT(SampleReadout,ELBATE)
```

Über die Ableitung "`class SampleReadout : public IReadout ...`" wird es ermöglicht, dass das Grundmodul des lokalen Eventbuilder auf die Auslese zugreifen kann. Auf die De-

finition der Ausleseklasse `SampleReadout` folgt ein Makro, welches ein Objekt der Auslese erzeugt (`INSTANCEREADOUT(SampleReadout,ELBATE)`) und an das Grundmodul zur Verwendung übergibt.
Die Auswahl der notwendigen Funktionen in der Schnittstelle orientiert sich an den jeweiligen Abläufen und Zuständen in einem lokalen Eventbuilder, d.h. es werden bestimmte Funktionen während des Ablaufes in bestimmten zeitlichen Abschnitten aufgerufen. Das Use-Case-Diagramm aus dem Abschnitt zuvor (siehe Abbildung 7.3) gibt eine gewisse Einteilung der Funktionen wieder.

Die Schnittstelle besteht aus zwei Teilen. Ein Teil beinhaltet alle notwendigen Funktionen zur Ausführbarkeit eines lokalen Eventuilders. Der andere Teil stellt Zusatzfunktionen zur Verfügung. Unter anderem können hier Aktionen an bestimmten zeitlichen Stellen eingefügt werden, oder es stehen Servicefunktionen wie z.B. die Ausgabe von Statusmeldungen zur Verfügung.

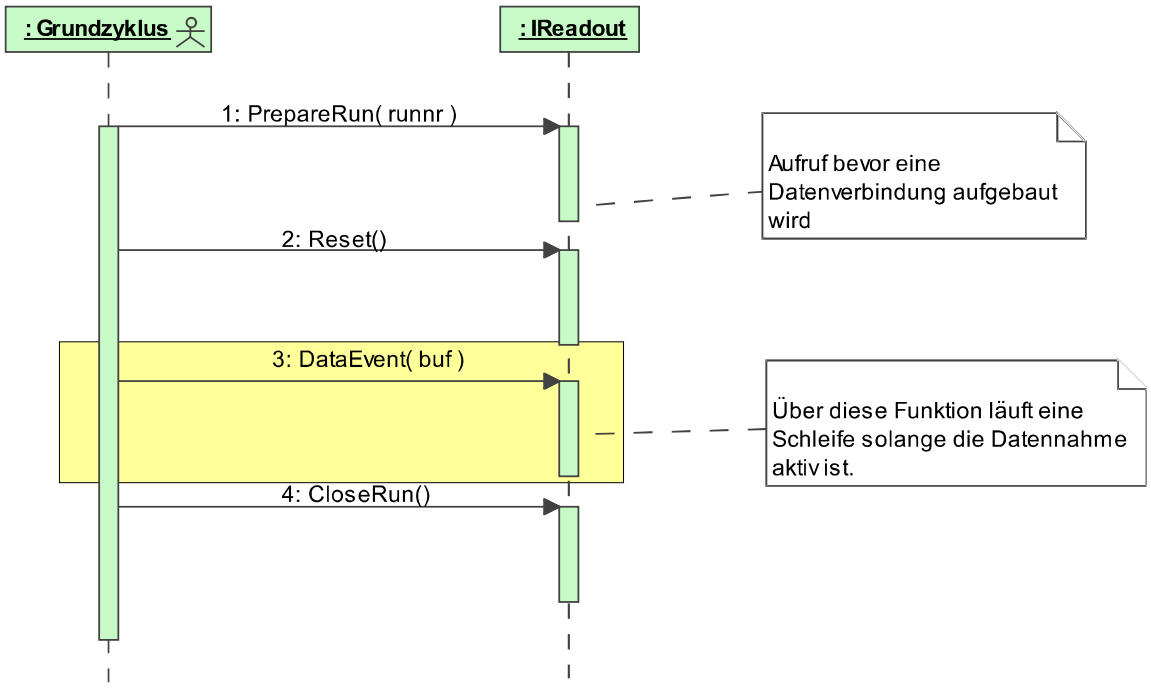


Abbildung 7.7: Vereinfachtes Sequenzdiagramm eines Datennahmezyklus

Abbildung 7.7 zeigt exemplarisch eine Sequenz von Funktionen, die während der Ausführung eines lokalen Eventbuilders aufgerufen werden. Die Auswahl der Funktionen orientiert sich an dem Beispiel-Eventbuilder zuvor, jedoch um die Funktion `Reset` erweitert.

Die Namen der Funktionsaufrufe geben schon grob den Kontext wieder, in dem diese Funktionen

aufgerufen werden. Sie sollen im folgenden aber noch etwas weiter erläutert werden:

`PrepareRun(...)` :

Diese Funktion wird aufgerufen, bevor die Datenverbindung zum Event-Saver aufgebaut wird. Sie dient zur Initialisierung von Variablen, die bei jedem Daten-Run durchgeführt werden müssen. Mit Hilfe des Rückgabewertes kann ein Abbruch des Startens erzeugt werden.

`Reset(...)` :

`Reset(...)` kann zu jedem beliebigen Zeitpunkt während der Ausführung des lokalen Eventbuilders aufgerufen werden. Die Funktion dient dazu, den lokalen Eventbuilder in seinen Ausgangspunkt zurückzusetzen. Sie darf nicht abhängig vom Startzeitpunkt sein, d.h. sie muss so implementiert sein, dass sie in jedem Zustand der Frontend-Elektronik ein Zurücksetzen der Elektronik durchführen kann.

`DataEvent(...)` :

In dieser Funktion wird die Hauptarbeit eines lokalen Eventbuilders geleistet. Sie wird bei jedem Ereignis aufgerufen und ist somit hoch frequentiert. Die maximal mögliche Rate eines lokalen Eventbuilders wird durch die Verarbeitungszeit dieser Funktion bestimmt.

`CloseRun(...)` :

Diese Funktion wird beim Beenden eines Daten-Runs aufgerufen.

Die Implementation oben genannter Funktionen reicht aus, um einen funktionsfähigen lokalen Eventbuilder zu erstellen. Der gesamte Umfang aller Funktionen der Ausleseschnittstelle `IReadout` wird in der DAQ-Referenz-Dokumentation [26] beschrieben.

Im nächsten Abschnitt soll zunächst auf die Verwaltung der Datenbuffer (`CPointerBuffer`) im lokalen Eventbuilder eingegangen werden, da es in den folgenden Abschnitten zum Verständnis der Abläufe im lokalen Eventbuilder beiträgt und diese Klasse universell in verschiedenen Bereichen des Datenakquisitionssystems eingesetzt wird.

7.2.2 Die Datenbufferverwaltung - `CPointerBuffer`

Ein Hauptbestandteil im Datenakquisitionssystem ist der Transport von Datenblöcken. Die Klasse `CPointerBuffer` stellt eine Funktionalität zur Verfügung, die es ermöglicht, Daten ohne Kopieraufwand zwischen zwei Prozessen auszutauschen. Weiterhin übernimmt sie die Verwaltung von Datenbereichen, die zum Datenaustausch benutzt werden können.

Ein Quellprozess liefert Daten, die zu einem Zielprozess transportiert werden sollen. Das Prinzip beruht auf zwei Listen, wobei die eine Liste "volle" Bereiche und die andere "leere" Bereiche enthält. Die Listen enthalten nur Zeiger auf die verwendeten Speicherbereich, sodass der Transport auf der Weiterleitung der Zeiger beruht.

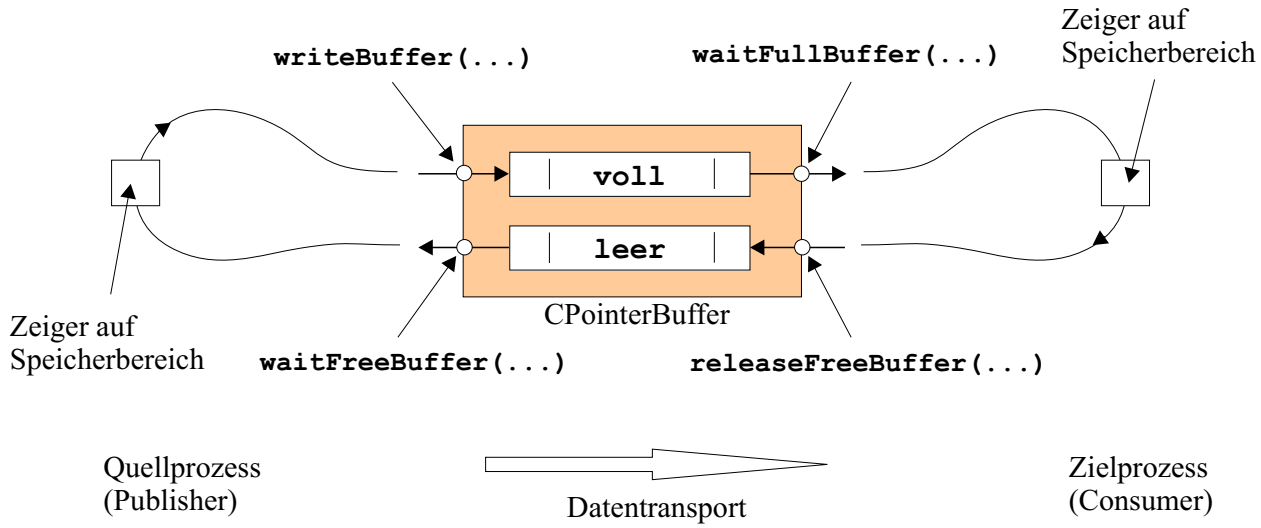


Abbildung 7.8: Anwendungsbereich der Klasse CPointerBuffer

Der Quellprozess entnimmt aus der "leer"-Liste einen Zeiger und füllt den Speicherbereich mit Daten auf. Daraufhin wird der Zeiger an die "voll"-Liste gehängt (siehe Abbildung 7.8). Der Zielprozess entnimmt einen "gefüllten" Zeiger aus der "voll"-Liste, verarbeitet die Daten und hängt den "leeren" Zeiger wieder an die "leer"-Liste an. Das Entnehmen der Zeiger blockiert den jeweiligen Prozess, falls kein Zeiger in der jeweiligen Liste zur Verfügung steht, d.h. im Falle der Verwendung eines Zeigers zirkuliert dieser vom Quellprozess zum Zielprozess und wieder zurück. Dies hat aber den Nachteil, dass die Prozesse sich gegenseitig blockieren. Es kann verhindert werden, indem man mehrere Zeiger verwendet, die vom Quell- zum Zielprozess zirkulieren.

Die Funktionen, beginnend mit `wait` als Namenszusatz blockieren die Weiterverabeitung, falls kein freier bzw. gefüllter Zeiger vorhanden ist. Das Warten auf ein Objekt blockiert zwar den Prozess, gibt aber den Prozessor frei für andere Prozesse. Diese nicht ganz triviale Listenverwaltung inklusive der damit verbundenen Prozesssteuerung konnte mit Standardfunktionen des Betriebssystems realisiert werden. Damit wird erreicht, dass wichtige Teile dieser Verwaltung auf dem Kernelniveau mit hoher Priorität ausgeführt werden und Vorrang vor allen anderen Benutzerprozessen haben.

Die Klasse kann eine beliebige Anzahl von untypisierten Zeigern verwalten, wobei die Zeiger in zwei Kategorien unterteilt werden. Verweist der Zeiger auf einen Buffer mit ausgelesenen

Daten, gilt er als gefüllt, sind die Daten schon weiterverarbeitet, gilt er als leer. Da untypisierte Zeiger (`void *`) verwendet werden, ist diese Klasse universell einsetzbar, wie später noch in anderen Bereichen zu sehen sein wird.

7.2.3 Der Datentransport - `ReadoutThread` → `DataThread`

Die Ausleseschnittstelle beinhaltet zwei Funktionen, welche die Daten aus der Frontend-Elektronik zur Verfügung stellen. Die Funktion `DataEvent(...)` soll die Daten liefern, die bei **jedem** Ereignis gelesen werden. Zusätzlich zu den Daten-Ereignissen existiert ein weiterer Typ die Zähler-Ereignisse, der bewirkt, dass die Zählerinformationen jede Sekunde ausgelesen werden.

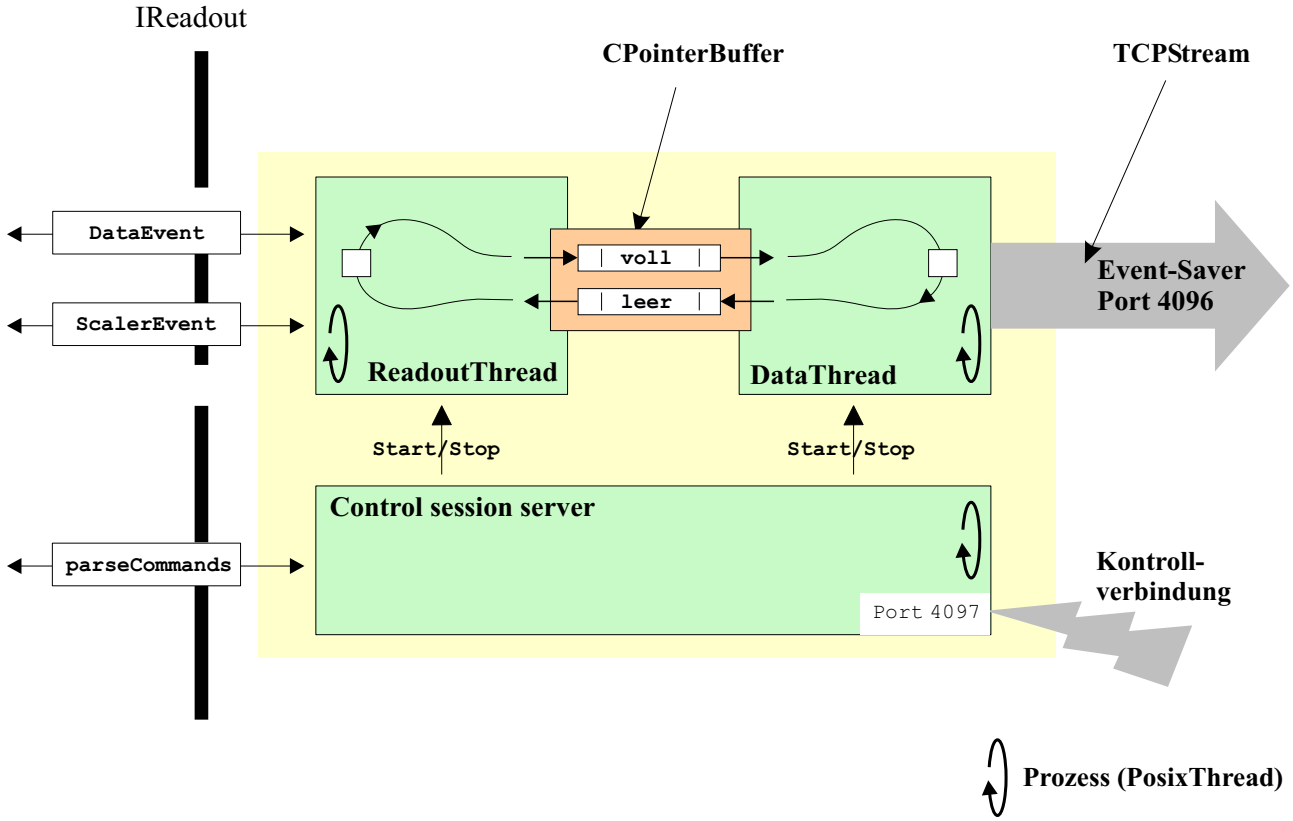


Abbildung 7.9: Schematischer Aufbau im Basismodul

Die Funktion `ScalerEvent(...)` soll diese Daten liefern.
Die Hauptaktivität des Grundmoduls besteht im Übertragen der gelesenen Daten, d.h. der lokale Eventbuilder "schiebt" aktiv die Daten zum Event-Saver. Im alten Datennahmesystem waren die lokalen Eventbuilder in dieser Hinsicht passiv. Sie warteten auf den Abtransport der Daten vom globalen Eventbuilder. Der Vorteil liegt nun in einer parallelen Übertragung der Daten. Dadurch, dass die Daten durch jeden lokalen Eventbuilder zum Event-Saver aktiv kopiert werden, kann hier gegenüber einer sequentiellen Übertragung, wie in der alten Datenakquisition, wertvolle Zeit gewonnen werden.

Makroskopisch gesehen übertragen alle lokalen Eventbuilder ihre Daten parallel zum Event-Saver. Auch im Grundmodul des lokalen Eventbuilder findet sich eine parallele Verarbeitung wieder. Die Aufgaben der Auslese der Frontend-Module und des Transfers der Daten zum Event-Saver werden in zwei eigenständigen Prozessen (threads) abgearbeitet, sodass eine dynamische Verteilung der Prozessorleistung auf diese beiden quasi parallelen Prozesse vom Betriebssystem ermöglicht wird.

In Abbildung 7.9 sind die beiden Prozesse schematisch gezeigt. Der Prozess (`ReadoutThread`) führt die Zugriffe auf die Schnittstelle `IReadout` aus. Hierzu zählen der richtige zeitliche Ablauf der Auslesefunktionen, die Statusmeldungen an den Sync-/Trigger-Kontroller und die Weiterleitung der Daten an die Klasse `CPointerBuffer`. Im Prozess (`DataThread`) werden die anstehenden Daten aus dem `CPointerBuffer` an die Netzwerkkomponenten übergeben. Die Daten werden über einen Multibuffer-Datenkanal (`CPointerBuffer` mit mehreren Datenkontainern) übertragen. Dadurch werden die folgenden Vorteile erreicht:

Ausnutzung von hardwarebedingten Totzeiten

Am Beispiel der Frontend-Module des Crystal-Barrel lässt sich dies anschaulich verdeutlichen. Die Module besitzen eine Konversionszeit von ca. 300 μs, bis die digitalen Informationen vorliegen. Um diese Zeit nutzbar zu machen, muss der Auslesebereich vom Transportbereich abgetrennt sein und jeweils in einem eigenen Prozess ablaufen, sodass jeder einen Zustand einnehmen kann, der keine Prozessorzeit verbraucht.

Entkoppelung vom Transportmechanismus

Ähnlich wie beim Ausleseprozess kann es auch im Transportprozess zu Verzögerungen in der Verarbeitung der Daten kommen. Um den Ausleseteil nicht zu blockieren, wenn es zu einer Verzögerung kommt, befindet sich zwischen den Prozessen ein Buffer. Dieser Buffer speichert bereits gefüllte Datenkontainer bis zu einer gewissen Anzahl zwischen, bevor der Ausleseprozess blockiert wird.

Um die statischen Beziehungen der beteiligten Klassen besser zu erkennen, ist in Abbildung 7.10 ein vergrößerter Teil aus dem UML-Diagramm in Abbildung 7.5 gezeigt. In den beiden Prozessen `ReadoutThread` und `DataThread` werden Referenzen auf den Buffer (`CPointerBuffer`) in der Variablen `BufferHandler` gespeichert. Hierüber erfolgt die Kommunikation zur Instanz des Objekts `CPointerBuffer`. Der Prozess `ReadoutThread` verwendet (<<uses>>) die freien Zeiger auf einen Datenkontainer (`CRawDataBuffer`), um die Daten aus den Frontend-Modulen darin abzulegen. Der `DataThread`-Prozess bezieht diese Zeiger auf einen vollen Datenkontainer aus dem Buffer und schreibt (<<sends>>) den Inhalt über eine Standard-TCP/IP-Verbindung zum Event-Saver. Zur Entkopplung der beiden Prozesse zirkulieren eine gewisse Anzahl von

Zeigern auf `CRawDataBuffer`-Objekten, die im Basismodul erzeugt wurden.

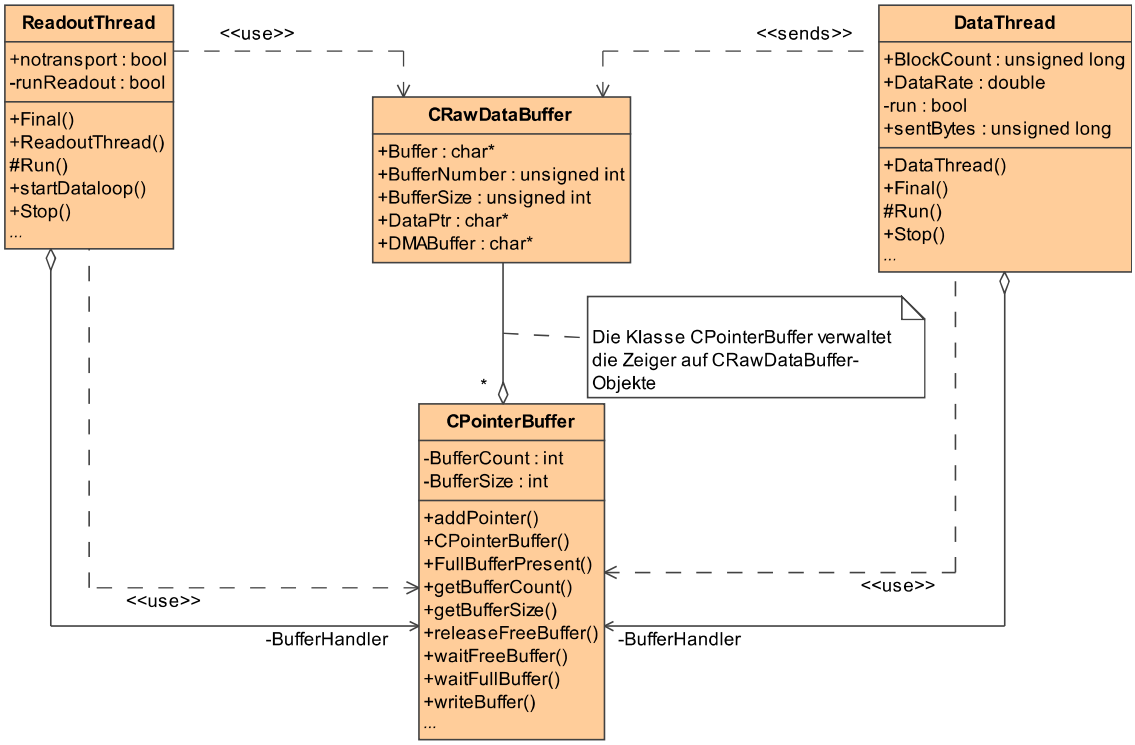


Abbildung 7.10: Statisches UML-Diagramm Klassen im Transportmechanismus

Der Ablauf innerhalb der beiden Prozesse `DataThread` und `ReadoutThread` ist in den UML-Sequenzdiagrammen in Abbildung 7.11 und 7.12 gezeigt. Zur besseren Übersicht ist die Darstellung der Sequenz auf die `Run()`-Funktion beschränkt.
Nach dem Start des `DataThread`-Prozesses wird die Funktion `waitFullBuffer` der `CPointerBuffer`-Klasse aufgerufen. In dieser Funktion wird auf einen Zeiger zu einem gefüllten Zwischenspeicher gewartet. Steht ein gefüllter Buffer zur Verfügung, wird dieser mit der Id des Sync-Moduls markiert (`setCPUid`). Dies erlaubt eine eindeutige Identifikation zu einem späteren Zeitpunkt. Im nächsten Schritt erfolgt die Übertragung der Daten über eine TCP/IP-Netzwerkverbindung zum Event-Saver. Die Klasse `CRawDataBuffer` stellt hierzu eine Funktion (`sendData`) zur Verfügung, die mittels eines Objektes vom Typ `TCPStream` die Daten überträgt. Nach der Übertragung wird der Buffer wieder geleert (`reset`) und mittels `releaseFreeBuffer` als freier Buffer wieder an den `CPointerBuffer` zurückgegeben. Diese Sequenz wird solange durchlaufen, bis der Prozess von außen durch ein Signal gestoppt wird.

Parallel zum `DataThread` läuft der Prozess `ReadoutThread`, der in Abbildung 7.12 gezeigt ist. Zu sehen sind zusätzlich zu den Objekten `CPointerBuffer` und `ReadoutThread` noch die Schnitt-

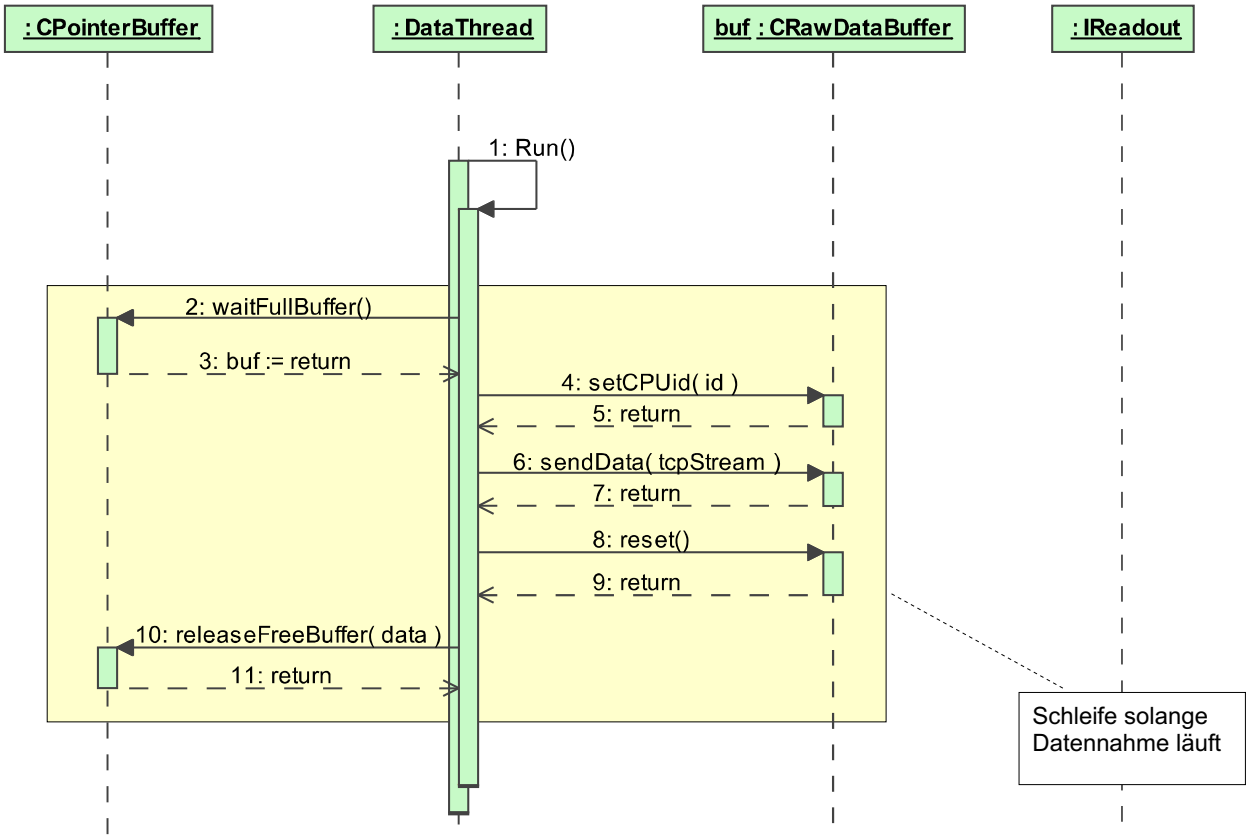


Abbildung 7.11: Darstellung des funktionellen Ablaufs im Datenprozess CDataThread mittels eines UML-Sequenzdiagramms

stellen `IReadout` und `IControlClient`. Auch hier sind zur Vereinfachung nur die wichtigsten Aufrufe dargestellt.

Zu Beginn der Schleife wird auf ein Signal (Interrupt) des Trigger-Systems mit der Funktion `waitTrigger()` aus der Klasse `IControlClient` gewartet. Vor und nach dem Aufruf befinden sich zusätzlich die Callback-Funktionen `BeforeWaitTrigger()` und `AfterWaitTrigger()`. Diese beiden Funktionen erlauben es, zusätzliche Aktionen vor und/oder nach dem Warten auf den Interrupt durchzuführen. Eine solche Erweiterungsmöglichkeit im Ablauf eines lokalen Eventbuilder wird erforderlich für die Kontrolle des Trigger-Gates (im Abschnitt 7.3.1 wird der Einsatz dieser Funktionen deutlicher).

Ist ein Trigger erfolgreich ausgelöst worden, wird über `waitFreeBuffer()` auf einen Zeiger eines leeren Buffers gewartet und an die Funktion `DataEvent()` der Schnittstelle `IReadout` übergeben. Nach erfolgter Auslese wird mittels `readInterruptqualifier()` überprüft, ob zusätzlich die Zähler des Subdetektors mit ausgelesen werden sollen. Bevor wieder auf ein Ereignis gewartet wird, übergibt der `ReadoutThread` den Zeiger des gefüllten Buffers an das `CPointerBuffer`-Objekt mit `writeBuffer()`. Nach dem Abschicken des Buffers erfolgt die Synchronisation mit dem Sync-/Trigger-Kontroller. Hierzu wird mittels `setOK()` der Ausgang des Auslesevorgangs

gesetzt und über `clearBusy()` die Bestätigung gegeben, dass die Auslese beendet wurde. Die Schleife ist somit vollständig und beginnt von neuem.

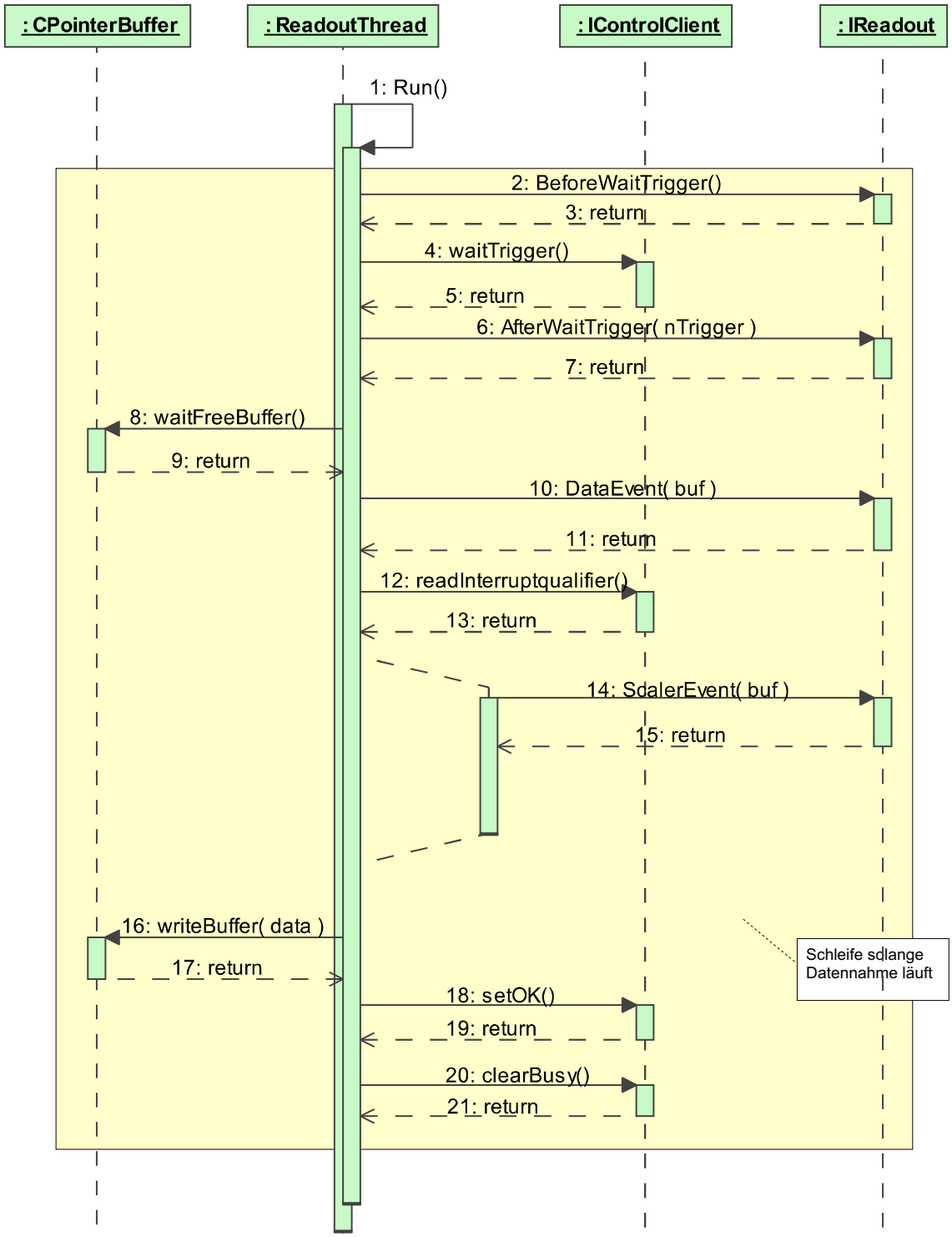


Abbildung 7.12: Das UML-Sequenzdiagramm Ausleseprozess CReadoutThread

7.2.4 Der Rohdatenkontainer - `CRawDataBuffer`

Die Klasse `CRawDataBuffer` beinhaltet die Rohdatenstruktur (`tRawDataBuffer`) und weitere Funktionen zur ihrer Bearbeitung, d.h. es wird nicht direkt auf der Struktur gearbeitet, sondern der Zugriff erfolgt über Ein-/Ausgabefunktionen (getter/setter-Funktionen).

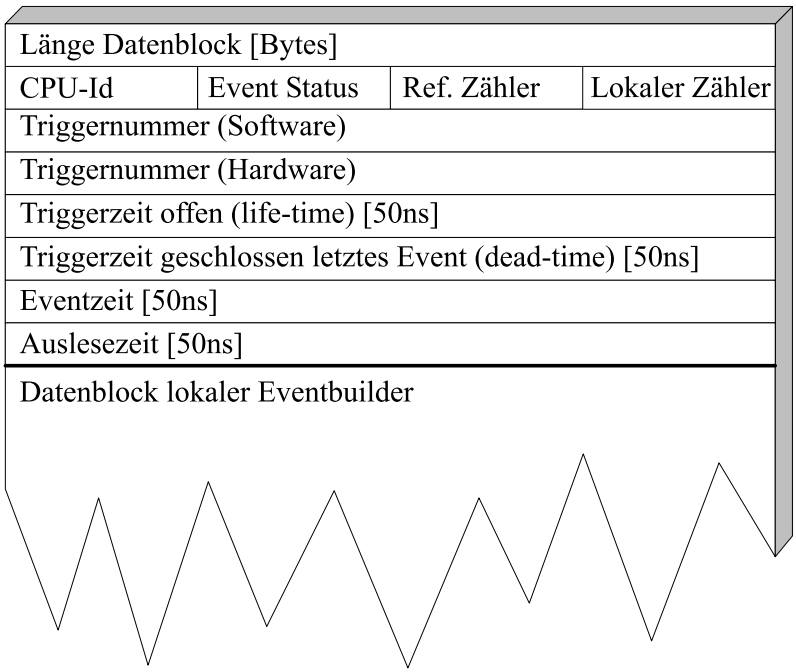


Abbildung 7.13: Der Aufbau der Rohdatenstruktur `tRawDataBuffer`

Neben dem eigentlichen Datenblock des lokalen Eventbuilders sind noch zusätzliche Werte gespeichert, die zur späteren Kontrolle der Synchronisation wichtig sind. Im Kopf der Struktur steht die Länge des gesamten Blocks (inkl. Längenwort) in einem 32 Bit-Wort. Die Länge wird in Bytes angegeben. Die weiteren vier 8 Bit-Werte haben folgende Bedeutung:

CPU-Id :

Sie ist eine Kennung des Quell-Prozesses, um eine korrekte Zuordnung des Datenblocks zu einem lokalen Eventbuilder zu gewährleisten. Mögliche Werte sind von 1-8. Eine Zuordnung CPU↔Id ist in der Definitionsdatei "`baselevb.h`" gegeben. Die eingetragene Id wird aus der Id des Sync-Clientmoduls gebildet.

Event Status :

Diese Variable enthält den Status des Ereignisses und kann folgende Werte annehmen:

`#define RD_DATAOK 0`

Auslese ohne Fehler beendet.

`#define RD_READOUTERR 1`

Bei der Auslese ist ein Fehler aufgetreten.

`#define RD_SIGSEGV 2`

Der Ausleseprozess wurde durch das Signal "Segmentation Violation" beendet.

`#define RD_NTRIGGER 3`

Die Auslese wurde durch mehr als einen Trigger gestartet.

Ref.-Zähler :

Dieses Feld beinhaltet die Buffernummer, die aus dem Sync-Modul bei dem betreffendem Ereignis gelesen wurde. Dieser Wert wird vor dem Öffnen des Triggers vom Sync/Trigger-Kontroller an alle Sync-Module übertragen. Die Zahl hat eine Breite von 4 Bit und wird bei jedem Ereignis um eins erhöht.

Lokaler Zähler :

Dieser Zähler ist auch 4 Bit breit, wird aber vom Grundmodul des lokalen Eventbuilders gezählt. Diese Zahl muss immer mit dem Ref.-Zähler übereinstimmen. Ist dies nicht der Fall, dann hat z.B. der lokale Eventbuilder zwei Triggersignale erfasst, obwohl vielleicht nur ein Signal gesendet wurde.

Triggernummer (Software/Hardware) :

Zur weiteren Bestimmung von Synchronitäten der lokalen Eventbuilder untereinander werden zwei 24 Bit-Zähler zusätzlich gespeichert. Der erste Zähler bestimmt die Anzahl der entstandenen Trigger mittels Software durch einfaches Aufaddieren in einer Variablen, der zweite zählt die eingegangenen Ereignisse über einen Hardwarezähler auf dem Sync-Clientmodul. Auch diese beiden Zähler müssen immer übereinstimmen.

Die nächsten vier 32 Bit-Werte liefern Zeiten, die bei der Ausführung des lokalen Eventbuilders entstehen. Unter anderem wird ein Zähler verwendet, um die Zeit zu bestimmen, die die Auslesefunktion `DataEvent()` benötigt. Diese Zähler werden durch einen 20Mhz Systemtakt gespeist und geben somit die Zeit in 50 ns-Schritten an.

Zeit des offenen Trigger-Gates :

Dieser Zähler gibt die Zeit an, in der das Trigger-Gate vor diesem Ereignis geöffnet war.

Totale Auslesezeit letzte Event :

Auf dem Sync-Modul befinden sich zwei Zähler, die abwechselnd den Systemtakt zählen, um totzeitlos die Zeit aus Auslese und Bufferverwaltung zu bestimmen. Der nicht aktive Zähler des Sync-Clientmoduls wird gelesen, sodass die Totzeit des lokalen Eventbuilders

beim letzten Ereignis erfasst wird. Diese Totzeit rechnet vom Schließen des aktiven Triggers bis zur Wiederöffnung.

Eventzeit :

Zählerstand zum Zeitpunkt, an dem das Interruptsignal empfangen wurde.

Auslesezeit :

Zeit, die zur Ausführung der Funktionen `DataEvent(...)` benötigt wurde. Es wird die Differenz des Zählerstandes eines freilaufenden Zählers vor und nach der Funktion ermittelt und gespeichert.

7.2.5 Das Synchronisationsmodul - `IControlClient,IControlServer`

Dieser Baustein kapselt die Funktionalität des Synchronisationsmechanismus. Er besteht aus einem Master- und mehreren Slave-Modulen.

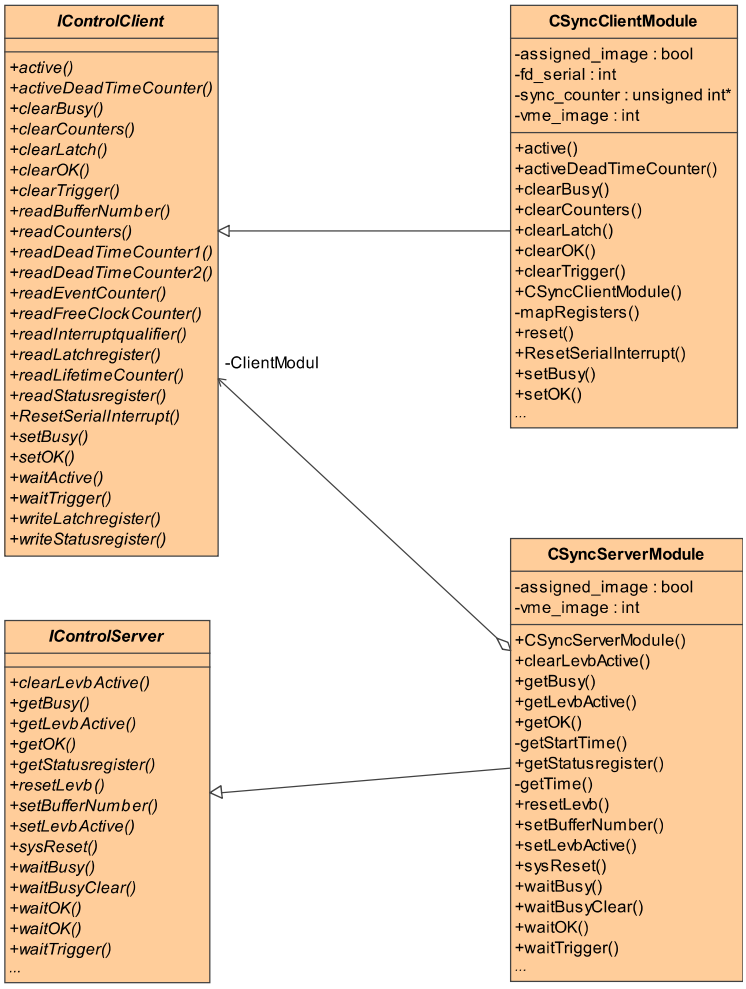


Abbildung 7.14: Die Schnittstellenklasse IControlClient und IControlServer

Das Master-Modul (`IControlServer`) beinhaltet die Aktivierung, die Statusabfragen einzelner lokaler Eventbuilder und die Verteilung von Synchronisationssemaphoren über den Synchronisationsbus.
Im Slave-Modul (`IControlClient`) sind Methoden vorhanden, die es ermöglichen, die Synchronisationsinformationen vom Master-Modul zu lesen, auf externe Triggersignale zu reagieren und zusätzlich auf Zählerinformationen für statistische Zwecke (z.B. Totzeitbestimmungen, etc.) zuzugreifen.

7.2.6 Das Modul für Statusmeldungen - `CBaseLogger`

Die Überwachung der einzelnen lokalen Eventbuilder ist ein wichtiger Punkt bei deren Initialisierung und während der Datennahme. Hierzu wird eine weitere Schnittstelle zur Verfügung gestellt, die eine Unterteilung der zu sendenden Status in verschiedene Wichtigkeitstufen vorsieht. So können später Fehlermeldungen von Statusinformationen gefiltert werden. Die Einteilung orientiert sich an den Stufen des Fehlerberichtssystems des Linux-Betriebssystems. Die Struktur dieses Moduls ist in Abbildung 7.15 dargestellt.

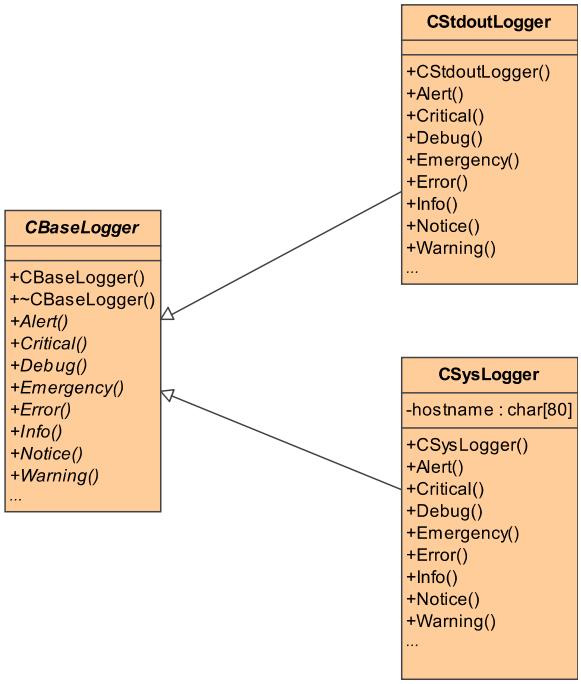


Abbildung 7.15: Die Schnittstellenklasse `CBaseLogger` und abgeleitete Klassen

Es existieren zwei Implementationen der Schnittstelle `CBaseLogger`,eine Kapselung der Ausgabe auf die Standard-Ausgabe (`stdout`) und eine Kapselung, die auf dem System-Logging-Mechanismus (`syslogd`) basiert. Je nach Anwendungsgebiet kann durch Verwenden der einen oder der anderen Instanz die Ausgabe umgeleitet werden.

7.2.7 Das Grundmodul - Die Bibliothek `libbaselevb.a`

Das Grundmodul eines lokalen Eventbuilders mit allen notwendigen Funktionen befindet sich in der Bibliothek `libbaselevb.a`. Diese Bibliothek hat eine Besonderheit, da sie auch die Funktion `main()` enthält, sodass ein ausführbares Programm durch Linken mit dieser Bibliothek erstellt werden kann. Die modulare Struktur erlaubt somit eine einfache Erzeugung des Basiscodes, wobei bei Änderungen in der Programmlogik des Grundmoduls nur eine zentrale Stelle zu modifizieren ist.

In Abbildung 7.9 wurde der grobe interne Aufbau des Grundmoduls eines lokalen Eventbuilders gezeigt.
Von den dort dargestellten drei Prozessen wurden die beiden Prozesse zur Auslese und zum Datentransport bereits in den Abschnitten zuvor erläutert. Der dritte Prozess dient zur Steuerung und Initialisierung aller Grundstrukturen. Beim Start der `main()`-Funktion wird ein Objekt der Ausleseschnittstelle erzeugt und gespeichert. In dieser Phase werden die Rohdatenkontainer `CRawDataBuffer` und der Buffer (`CPointerBuffer`) für den Auslese- und Transportprozess erzeugt. Standardmäßig wird eine Buffer mit 16 Einträgen und einer Buffergröße von 32kByte initialisiert. Falls ein spezieller Speicherbereich verwendet werden soll, wie zum Beispiel bei einem DMA[1]-Transfer, um die Datentransferrate zu steigern, werden über die Funktion `getBufferPtr()` die Adressen der Speicherbereiche abgefragt. Nach der Initialisierung geht der lokale Eventbuilder in einen "bereit"-Status über. Zur Steuerung des lokalen Eventbuilders steht in jedem Zustand ein Kommandointerpreter (Übersicht der Befehle im Anhang C) zur Verfügung, der über eine separate TCP/IP-Verbindung oder über die Konsole erreichbar ist. Je nach Betriebsmodus des lokalen Eventbuilders steht die eine oder die andere Möglichkeit zur Verfügung:

Daemon-Modus :

Dieser Modus wird verwendet, wenn der lokale Eventbuilder sich im Produktionseinsatz befindet, d.h. der Prozess wird automatisch gestartet, die Statusinformationen und Fehlermeldungen werden auf das Standard-Logging-System des Linux-Betriebssystems umgeleitet. Befehle können hierbei über einen TCP-Port abgegeben werden. Dieser ist multisessionfähig und erlaubt den gleichzeitigen Zugriff von verschiedenen Rechnern auf den lokalen Eventbuilder.

Im folgenden ist ein Beispiel für eine Kontrollverbindung gezeigt:

```
control /home/schmidt> telnet velbae1 4097
Trying 131.220.219.231...
```

[1]Direct Memory Access

```
Connected to velbae1.
Escape character is '^]'.
welcome to Trigger readout and control v1.2
status
3
OK
counter
635 5707705 2311049 4739707 15434326
OK
logout
Connection closed by foreign host.
```

Konsolen-Modus :

In diesem Modus ist es möglich, den lokalen Eventbuilder in einer Konsole zu steuern. Die Ausgaben werden direkt auf den Bildschirm umgeleitet. Dieser Modus ist besonders hilfreich für Testzwecke und in der Entwicklungsphase.

```
velbae1 /home/schmidt> evb
initializing trigger
using default config /home/daq/Config/evb/evb.ini
Loading Triggerfile /home/daq/TriggerFiles/taggeror.sts
loading default trigger from /home/daq/TriggerFiles/taggeror.sts
loading trigger readout from /home/daq/CamacFiles/evb/ReadTrigger.csq
loading scaler readout from /home/daq/CamacFiles/evb/ReadScaler.csq
loading gamma veto readout from /home/daq/CamacFiles/evb/ReadGammaV.csq
Trigger readout and control v1.2 started.
data connection: 10.0.0.100:4096, control connection: 0.0.0.0:4097
status
3
OK
counter
635 5707705 2311049 4739707 15434326
OK
logout
velbae1 /home/schmidt>
```

7.3 Die lokalen Eventbuilder - Implementationen

Nachdem zunächst allgemeine Anforderungen/Eigenschaften der lokalen Eventbuilder erläutert wurden, soll nun auf ein paar Besonderheiten bei den jeweiligen Implementationen zu den Subdetektor-Komponenten eingegangen werden, die durch die jeweilige spezielle Hardwareimplementation und Systemaufgabe bedingt sind.

7.3.1 Der Sync-/Trigger-Kontroller - `evb`

Der Sync-/Trigger-Kontroller (Programmname: evb[2]) erweitert das Grundmodul am stärksten, da er sowohl Steuerungsfunktionen über die lokalen Eventbuilder als auch die Auslese von diversen Frontend-Modulen übernimmt. Hierzu zählen das Auslesen von Triggerinformationen, Trigger-Zählern, dem Gamma-Veto-Detektor und der FACE.

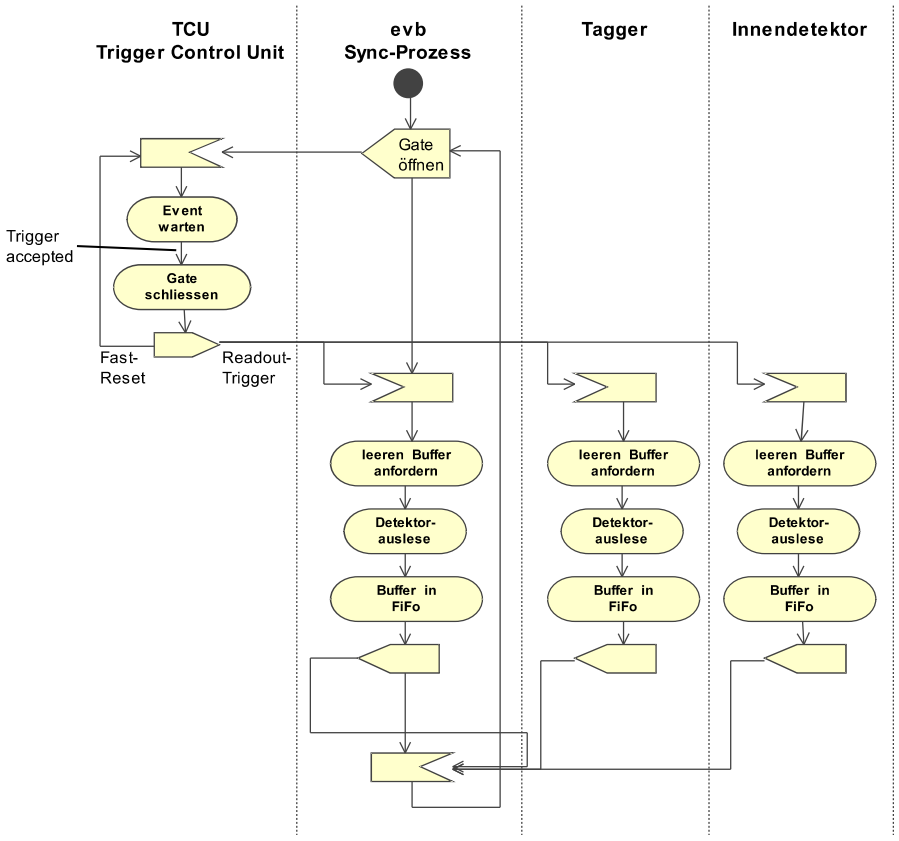


Abbildung 7.16: Aktivitäten-Diagramm des Synchronisationsprozesses

Das Aktivitäten-Diagramm in Abbildung 7.16 zeigt den Ablauf der Triggersteuerung während der Datennahme. Dieser Ablauf wird erreicht, indem noch drei weitere Funktionen im Stan-

[2]Name historisch bedingt

dardverhalten geändert werden:

`BeforeTrigger(...)` :

Bevor der Trigger geöffnet wird, muss die Buffernummer an alle Sync-Module verteilt werden, die zur Überprüfung der Synchronität benutzt werden.
Dies wird mit Hilfe der Instanz `SyncMaster` und der Funktion `setBufferNumber(...)` durchgeführt:

```
SyncMaster.setBufferNumber(BufferNumber);
```

Anschließend wird das Gate des Triggersystems geöffnet:

```
hwt.openTrigger();
```

`AfterTrigger(...)` :

Die Definition der Funktion `AfterTrigger` sieht im Detail folgendermaßen aus:

```
bool AfterWaitTrigger(int nTrigger);
```

Diese Funktion enthält über den Eingangsparameter `nTrigger` die Anzahl der erzeugten Interrupts des Betriebssystems, die vom Interrupttreiber zur Verfügung gestellt wird. Dies dient zur Kontrolle, dass am Triggereingang des Sync-Clientmoduls nur ein Puls die Datennahme ausgelöst hat. Der Rückgabewert gibt an, ob die Datennahme begonnen wird oder das Gate noch einmal geöffnet werden soll.

Zu Beginn der Funktion wird geprüft, ob das Triggersystem ein Ereignis akzeptiert hat. Das System hat die Eigenschaft, das Trigger-Gate selbständig zu schliessen, wenn ein Ereignis eingegangen ist. Dies wird über den Aufruf `hwt.isTriggerClosed()` überprüft. Falls das Gate geschlossen ist, wird im Anschluss ermittelt, ob alle aktiven Sync-Clientmodule als Status "BUSY" melden. In diesem Fall wird mittels Rückgabewert `true` die Datenauslese angestoßen. In allen anderen Fällen erfolgt eine Rückgabe von `false`.

`AfterReadout(...)` :

Die Funktion `AfterReadout` befindet sich am Schluss der Auslese aller Module. Der Sync-/Triggerkontroller wartet zu diesem Zeitpunkt auf das Beenden der Auslese aller lokalen Eventbuilder. Zu Beginn wird mittels `SyncMaster.waitBusyClear(...)` darauf gewartet, dass alle lokalen Eventbuilder die Auslese beendet haben. Melden alle die Fertigstellung,

wird über die OK-Leitung geprüft, ob deren Auslese erfolgreich war. Im Fehlerfall wird eine Meldung über das Statusmodul ausgegeben, und die Datennahme stoppt. Die Fehlerbehebung und der Neustart der Datennahme müssen durch den Anwender erfolgen, da es sich in solchen Situationen um einen schwerwiegenden Fehler handelt, der nicht automatisch behoben werden kann.

Da der Sync-/Trigger-Kontroller den Datennahmeablauf als einziger steuern kann, muss er zusätzlich in allen anderen Aktionen involviert sein, die das Gesamtsystem benötigen, wie Kalibrationsruns und spezielle Scanfunktionen, in denen nur Teile des Detektors benötigt werden.

1. Durchführung eines Lightpulser-Run

   Der Crystal-Barrel verfügt über einen Lichtpulser, der über Lichtleiter Lichtblitze verschiedener Intensität in den Wellenlängenschieber jeden Kristalls einkoppeln kann, um auf diesem Wege die Stabilität des Kristallmesssystems zu kontrollieren. Für jede Crystal-Barrel-Hälfte existiert ein Gerät [22]. Die Steuerung erfolgt über CAMAC-Module, die sich am CAMAC-Bus des Sync-/Trigger-Kontrollers befinden. Über einen speziellen Trigger wird eine Blitzfolge mit definierter Intensität über die ADCs des Crystal-Barrels aufgenommen und somit die Verstärkung des Gesamtsystems gemessen. Die Funktionen und der Ablauf sind in der Klasse `CLightpulser` implementiert.

2. Ausführung eines Scans für das Radiatorsystem

   Das Radiatorsystem verfügt über diverse Achsen, mit denen das Radiatortarget relativ zum Elektronenstrahl positioniert werden kann. Bei einem Goniometer-Run werden nur die Zählerinformationen des Tagging-Systems aufgezeichnet.

   Auf diesem Wege ist es möglich, ein Strahlprofil des primären Elektronenstrahls zu erstellen, da speziell für Messungen mit longitudinalen Photonen die Lage und die Ausdehnung des Strahls kritische Parameter sind. Weiterhin können aus den gemessenen Zählraten Histogramme erstellt werden, die es ermöglichen, einen Kristall so auszurichten, dass polarisierte Photonen bei einer gewünschten Energie erzeugt werden können. Die Klasse `Goniometer` ist verantwortlich für die Bewegung des Goniometers während solcher spezieller Messphasen.

Beide Klassen sind im Sync-/Trigger-Kontroller mit eingebettet und können über spezielle Kommandos gesteuert werden. Hierzu wurde die Funktion `ParseCommands(...)` überladen und leitet je nach Ziel ("GONI" oder "FLASHER") die Kommandos an die entsprechende Klasse weiter (siehe Anhang C.2).

7.3.2 Das Tagging-System - `tagger`

Das Tagging-System besteht aus drei Auslesebereichen (siehe Kap. 6.2): den Szintillationszählern, den Proportionalkammern und dem Faserstreifendetektor. In Abbildung 7.17 ist der Ablauf, wie er in der Funktion `DataEvent(...)` implementiert ist, in groben Zügen dargestellt.

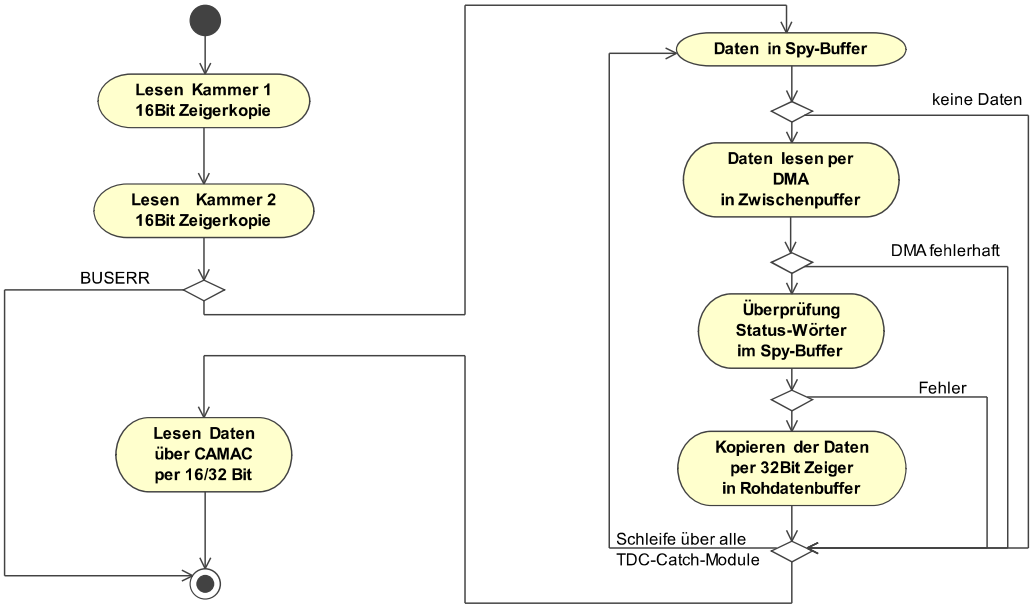


Abbildung 7.17: Aktivitäten-Diagramm des Datennahmenablaufs beim Tagging-System

Zu Beginn werden die Hit-Speicher der oberen und der unteren Proportionalkammer gelesen. Hierzu werden die 16 Bit-breiten Latches über einen D16-Zugriff in einer Schleife in den Rohdatenbuffer kopiert.

Im nächsten Schritt folgt die Auslese der TDC-Catch-Module. Die Funktionen für die Catch-Module sind in der Bibliothek "libcatch.a" ausgelagert, der dargestellte Ablauf ist in der Funktion `readSpyTDC(...)` implementiert. Die acht Catch-Module werden in einer Schleife nacheinander abgearbeitet. Die Daten stehen nach erfolgter Triggerung im Spy-Buffer des Catch-Moduls zur Verfügung. Sobald die Daten im Spy-Buffer vorhanden sind, werden die Werte mittels DMA in einen Zwischenspeicher kopiert. Dann werden die Status-Wörter im Spy-Buffer auf Fehlermeldungen geprüft; und die Daten werden per 32 Bit-Zugriff in einer Schleife in den Rohdatenbuffer kopiert.

Zum Ende der Auslese werden noch die ADC und TDC-Werte der Szintillationszähler über einen 16 Bit-Zugriff aus dem CAMAC-System in den Rohdatenbuffer kopiert.

7.3.3 Der Innen-Detektor - `scifi`

Die Daten des Innendetektors teilen sich auf in 30 12 Bit-ADC-Werte, die über CAMAC-ADC-Module bestimmt werden, und in 512 12 Bit-TDC-Werte, die über FastBus ausgelesen werden.

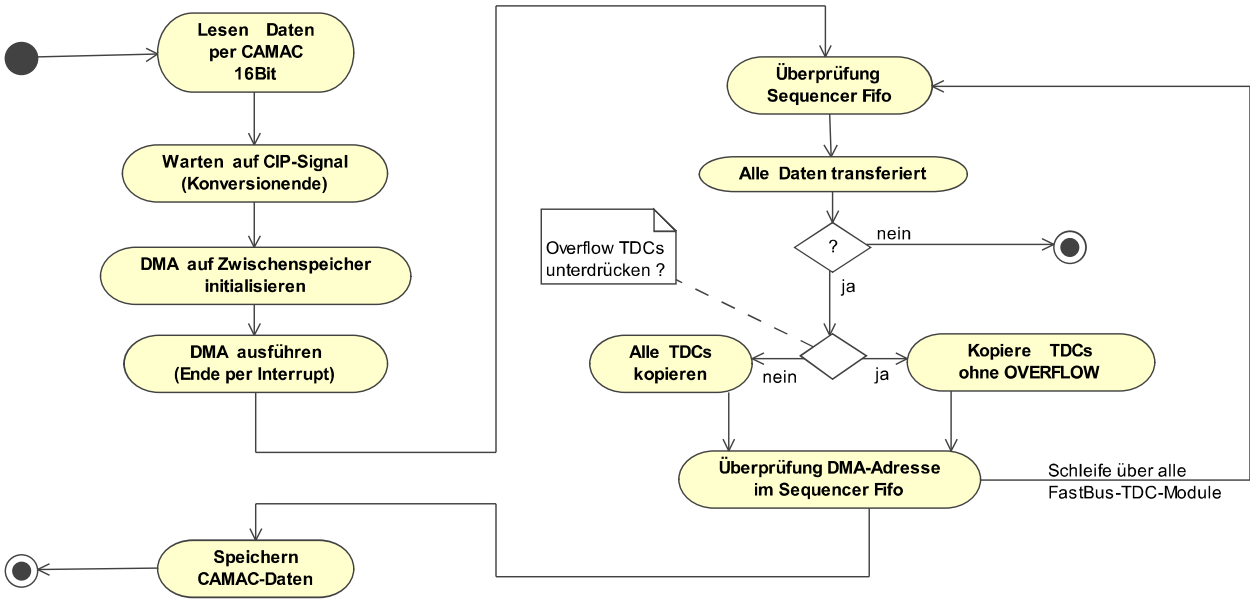


Abbildung 7.18: Aktivitäten-Diagramm des Datennahmeablaufs beim Innen-Detektor

Durch die geringe Anzahl von VMEbus-Slots im FastBus-Sequenzer ist der Zugriff auf den CAMAC-Branch-Kontroller über einen VIC-Bus realisiert. Der Kontroller des VIC-Busses spiegelt das entfernte Crate, in dem sich der CAMAC-Branch-Kontroller befindet, in einen gewissen Speicherbereich, sodass der Zugriff identisch abläuft, so als befände sich der Kontroller auf dem gleichen VMEbus. Nachdem die ADC-Werte per 16Bit-Zugriff gelesen wurden, wird auf das Konversionsende (CIP-Signal) der FastBus-TDC-Module gewartet. Ist das Konversionsende erreicht wird die Liste, die im FastBus-Sequenzer zur Auslese der TDC-Kanäle abgelegt ist, gestartet. Der Sequenzer führt nun autonom die Liste aus und legt die gelesenen Werte per DMA-Transfer direkt im Speicher der Prozessorkarte ab. Der FastBus-Sequenzer gibt das Ende des Transfers mittels eines Interrupts über den VMEBus an den Prozessor weiter. In einem speziellen Fifo des Sequenzers werden Informationen zu diesem DMA-Transfer abgelegt. Dadurch kann der erfolgreiche Transfer überprüft werden. Die TDC-Werte werden im Anschluss an den DMA-Transfer per 32 Bit-Zugriff Wort für Wort kopiert. Hierbei können noch zusätzlich durch den Prozessor die TDC-Kanäle unterdrückt werden, die sich im Überlauf befinden.

7.3.4 Der Crystal-Barrel - `cb1,cb2`

Die Auslese des Crystal-Barrel baut auf der gleichen Bus-Technik auf, die beim Innen-Detektor verwendet wurde. Jedoch kommt bei der Auslese des Crystal-Barrel eine erweiterte Version des FastBus-Sequenzers zum Einsatz.
Wie beim Innen-Detektor ist im Sequenzer eine Liste zur Auslese der ADC-Module abgelegt. Zusätzlich zu dieser Liste kann ein Schwellwert zu jedem ADC-Kanal im Sequenzer gespeichert werden. Die erweiterte Version des Sequenzers verfügt über einen speziellen Signalprozessor, der den gelesenen Wert eines ADC-Kanals mit dem zugehörigen Wert in der Schwellwertliste vergleichen kann. Liegt der Wert oberhalb, wird der Kanal per DMA in den Speicher kopiert, falls nicht, wird der nächste Kanal des ADC-Moduls überprüft. Diese Eigenschaft erlaubt es, das Datenaufkommen drastisch zu reduzieren (Reduktion der Daten um ca. einen Faktor 6). Die Erstellung der Schwellwerttabelle beruht auf einer Bestimmung des Pedestal-Wertes pro ADC zu Beginn jeden Runs. Somit werden nur ADC-Kanäle gespeichert, die zur späteren Analyse wesentlich sind.
Diese Besonderheit ist im einfachen Aufbau des Ablauf-Diagramms (siehe Abbildung 7.19) zu bemerken.

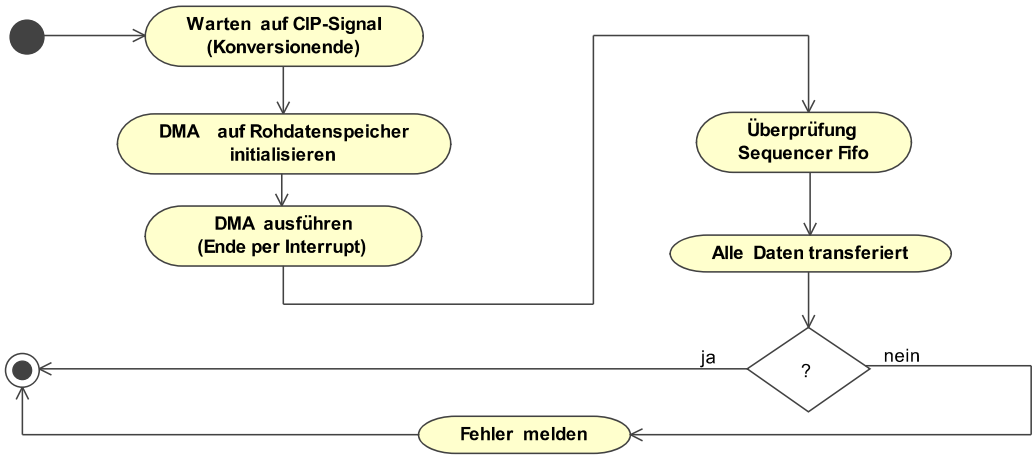


Abbildung 7.19: Aktivitäten-Diagramm des Datennahmenablaufs beim Crystal-Barrel

Da beim DMA-Transfer die Daten durch den Filter behandelt werden, ist für die übertragenen Daten keine weitere Überarbeitung mehr erforderlich, sie müssen nur noch in den Speicher des Buffers zur Datenübertragung kopiert werden. Dieser Kopiervorgang kann allerdings mit Hilfe der Funktion `getBufferPtr()` aus der Schnittstelle `IReadout` vermieden werden.
Standardmäßig werden im Buffersystem des lokalen Eventbuilders Speicherbereiche einer Standardgröße im Benutzerspeicher angelegt. Sollen die Elektronik-Module aber über einen DMA ausgelesen werden, so benötigt man einen speziellen Speicherbereich, der vom VMEBus-Treiber zur Verfügung gestellt wird. Die Zeiger auf diese Speicherbereiche können nun über die Funkti-

on `getBufferPtr()` an das Buffersystem übergeben werden. Hierzu wird die Funktion während der Erzeugung der Buffer aufgerufen und der Rückgabewert als Zeiger ins Buffersystem eingetragen.

Während der Datennahme wird vor dem Start der Liste die Startadresse des DMA-Transfers auf die Adresse des Buffers gesetzt, d.h. der Sequenzer kopiert die Daten der ADC-Module direkt an die richtige Stelle; es wird kein weiterer Kopiervorgang notwendig.

Im Abschnitt 8.3, wo die Leistungsdaten und der Zeitverbrauch der lokalen Eventbuilder diskutiert werden, zeigt sich deutlich eine Zeitersparnis gegenüber z.B. dem Innen-Detektor.

7.3.5 Die TAPS-Kopplung - `taps`

Die TAPS-Datenakquisition wird im Kontext der Crystal-Barrel-DAQ als Subdetektor angesehen, d.h. die TAPS-DAQ wird in einem Modus betrieben, in dem das Triggersystem des Crystal-Barrel-Detektors aktiv ist und die Ablaufsteuerung übernimmt. Daher muss ein Handshake zwischen dem lokalen Eventbuilder (Ausleseprozessor und Sync-Clientmodul) und der TAPS-DAQ erfolgen. Realisiert ist dieses über einen Speicherbereich, der sowohl von der TAPS-DAQ als auch vom lokalen Eventbuilder im Haupt VME-Crate der TAPS-DAQ erreichbar ist.

Abbildung 7.20 zeigt den implementierten Ablauf bei dieser DAQ-Kopplung.

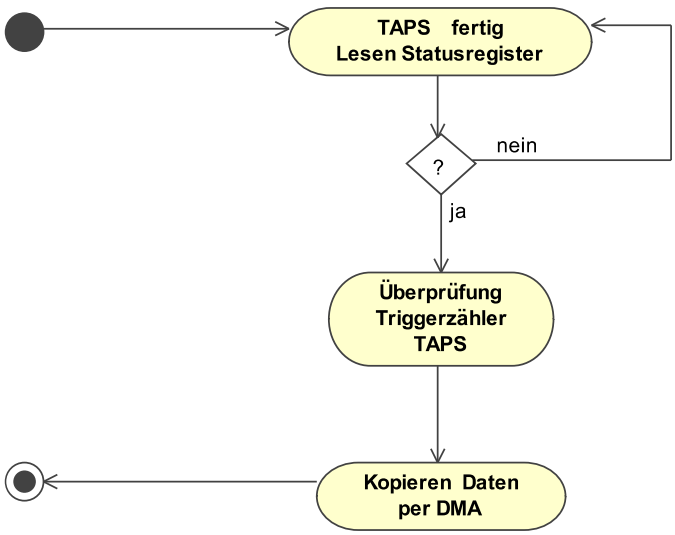


Abbildung 7.20: Aktivitäten-Diagramm des Datennahmenablaufs bei TAPS

7.3.6 Die Flugzeitwand - `tof`

Beim ToF-Detektor werden die Informationen der Szintillationszähler über FastBus-ADC- und TDC-Module gelesen. Wie bei den beiden Subdetektoren zuvor kommt der Struck FastBus-Sequenzer zum Einsatz, d.h. es werden die gleichen Subroutinen für die Auslese der Module verwendet.
Abbildung 7.21 zeigt den Ablauf der Auslese.

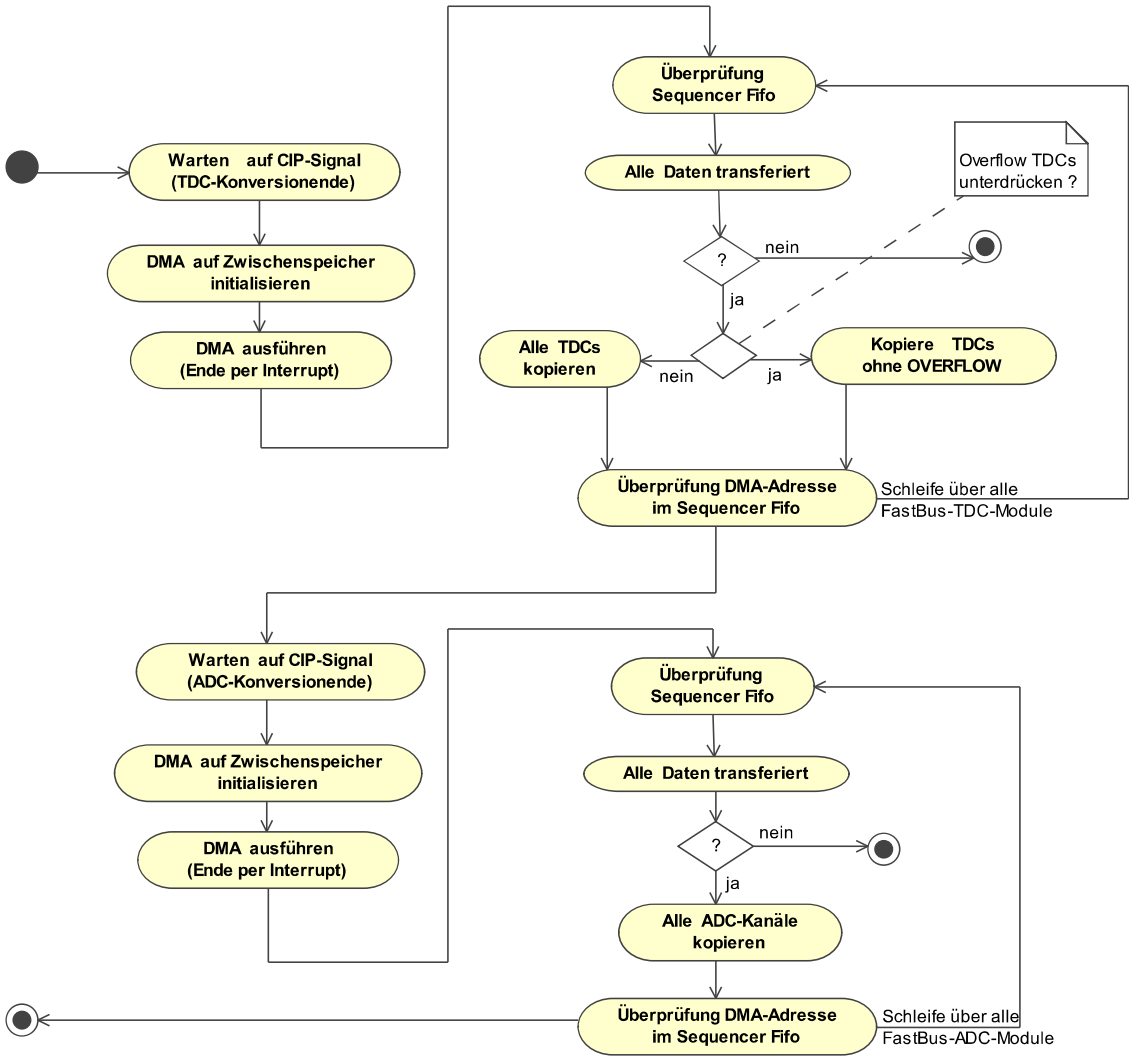


Abbildung 7.21: Aktivitäten-Diagramm des Datennahmenablaufs beim ToF

7.4 Der Event-Saver - evs

Zentraler Punkt im Datenakquisitionssystem ist nach den lokalen Eventbuildern der Event-Saver. Hier laufen alle Datenströme der Rohevents zusammen und müssen zu einer vollständigen Dateninformation (Event) der einzelnen Subdetektorereignisse unter Wahrung der Synchronität zusammengefasst werden.
Abbildung 7.22 zeigt in einer Übersicht die Anwendungsfälle des Event-Saver.

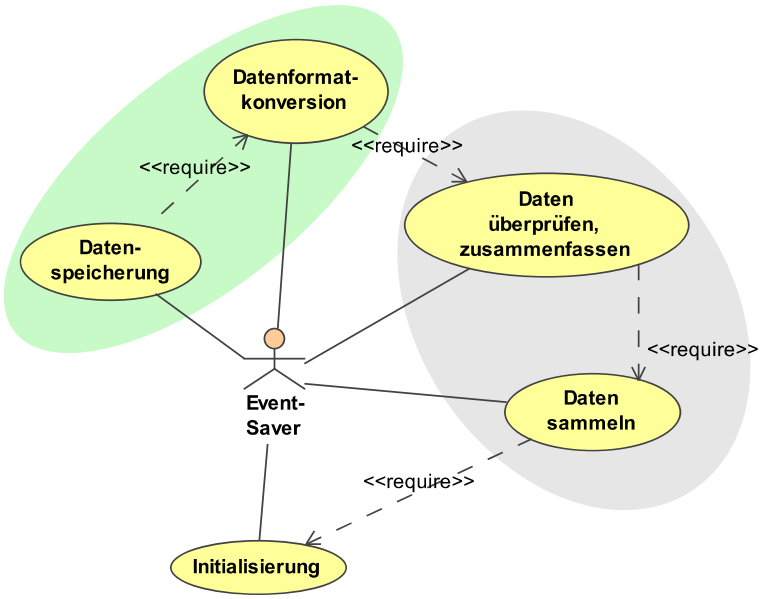


Abbildung 7.22: Use-Case-Diagramm Event-Saver

Im Event-Saver laufen die Daten der einzelnen lokalen Eventbuilder zusammen, d.h. die Daten treffen parallel über eine Standard-TCP/IP-Netzwerkverbindung am Event-Saver ein. Aus diesem parallelen Datenstrom wird nun aus den Daten der einzelnen Subdetektoren ein Ereignis zusammengefasst. Die Ereignisse werden dann mittels Prüfmechanismen auf die Synchronität der einzelnen Subdaten hin überprüft. Diese Aufgaben sind im Use-Case-Diagramm (Abbildung 7.22) durch eine graue Ellipse unterlegt. Die so zusammengefassten Rohdaten werden in einem weiteren Schritt in ein definiertes Datenformat geschrieben (grüne Ellipse), um sie zu einem späteren Zeitpunkt in Analyse-Programmen zu verarbeiten. Diese grobe Einteilung in den Aufgabengebieten findet sich auch in der Struktur der einzelnen Klassen später wieder. Die farblich markierten Anwendungsbereiche aus dem Use-Case-Diagramm (siehe Abbildung 7.22) sind auch in Abbildung 7.23 mit den zugehörigen erstellten Klassen und ihrer Beziehungen gezeigt. Abbildung 7.24 zeigt die gleichen Klassen wie aus dem UML-Diagramm, wobei hier die Datenwege und Datentypen in den Vordergrund gerückt wurden.
Die Daten der lokalen Eventbuilder werden mittels zweier Objekte am Event-Saver empfangen und bearbeitet. Die Betriebssystemfunktionen für die Bedienung der Netzwerkschnittstelle wer-

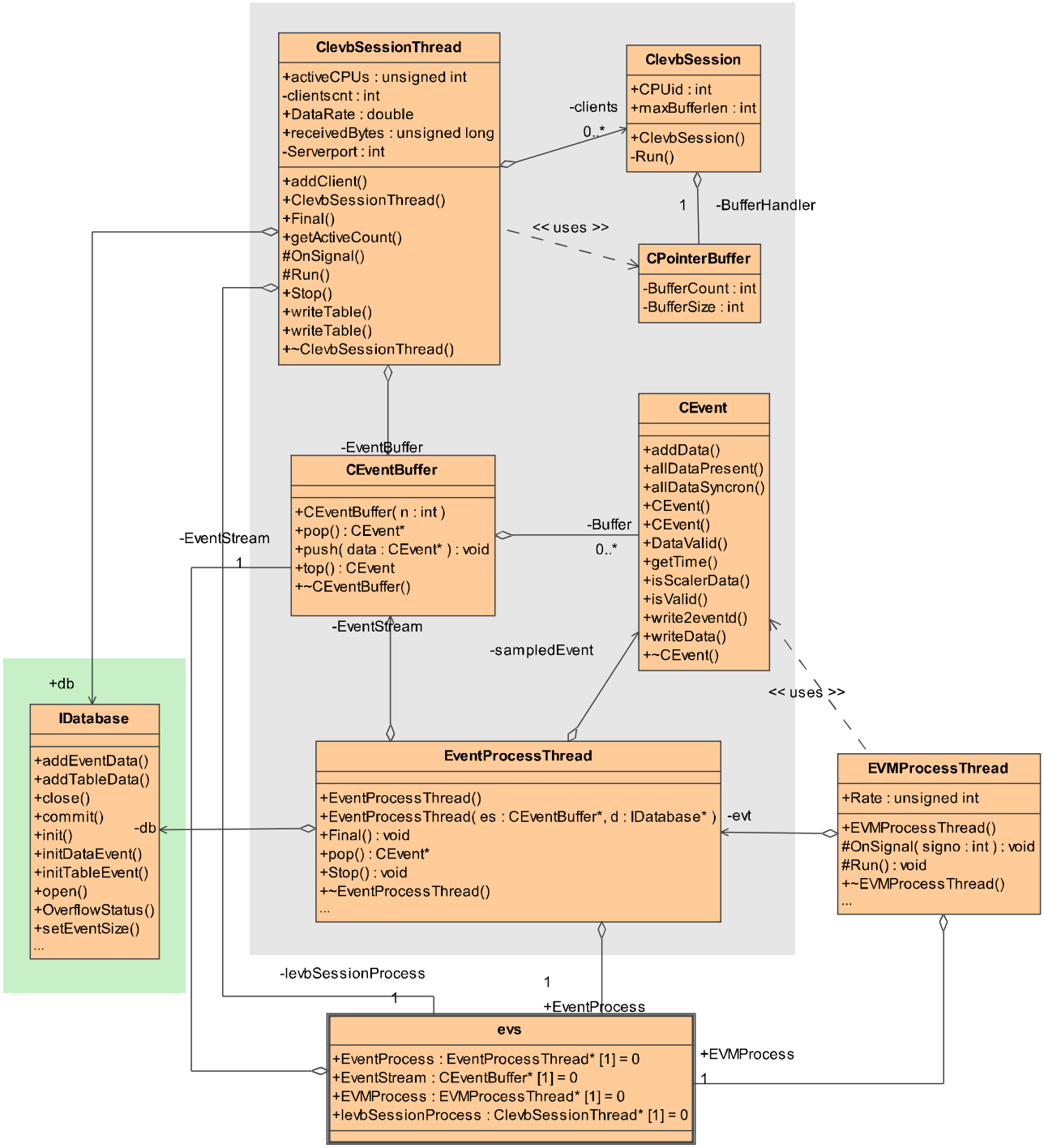


Abbildung 7.23: Statisches UML-Diagramm Event-Saver

den über die Klasse `TCPStream` gekapselt, die von einer Standardbibliothek (Common-C++) zur Verfügung gestellt wird. Das Objekt `ClevbSession` empfängt die Datenblöcke des lokalen Eventbuilders, die die Struktur `tRawDataBuffer` besitzen, und reicht sie im Datenkontainer `CRawDataBuffer` über einen Multibuffer (`CPointerBuffer`) an den Verwaltungsprozess `ClevbSessionThread` weiter.

Das Objekt `ClevbSession` wird dynamisch während der Laufzeit des Event-Savers beim Verbindungsaufbau von einem lokalen Eventbuilder aus erzeugt und nach Beendigung der Verbindung wieder gelöscht. Die hiermit verbundene Ablaufsteuerung wird wie beim lokalen Eventbuilder durch eine Kontrollsession-Klasse durchgeführt, die aus Übersichtlichkeit in der Abbildung nicht dargestellt ist (Dieses dynamische Verhalten wird im Kapitel 7.5 näher erläutert).

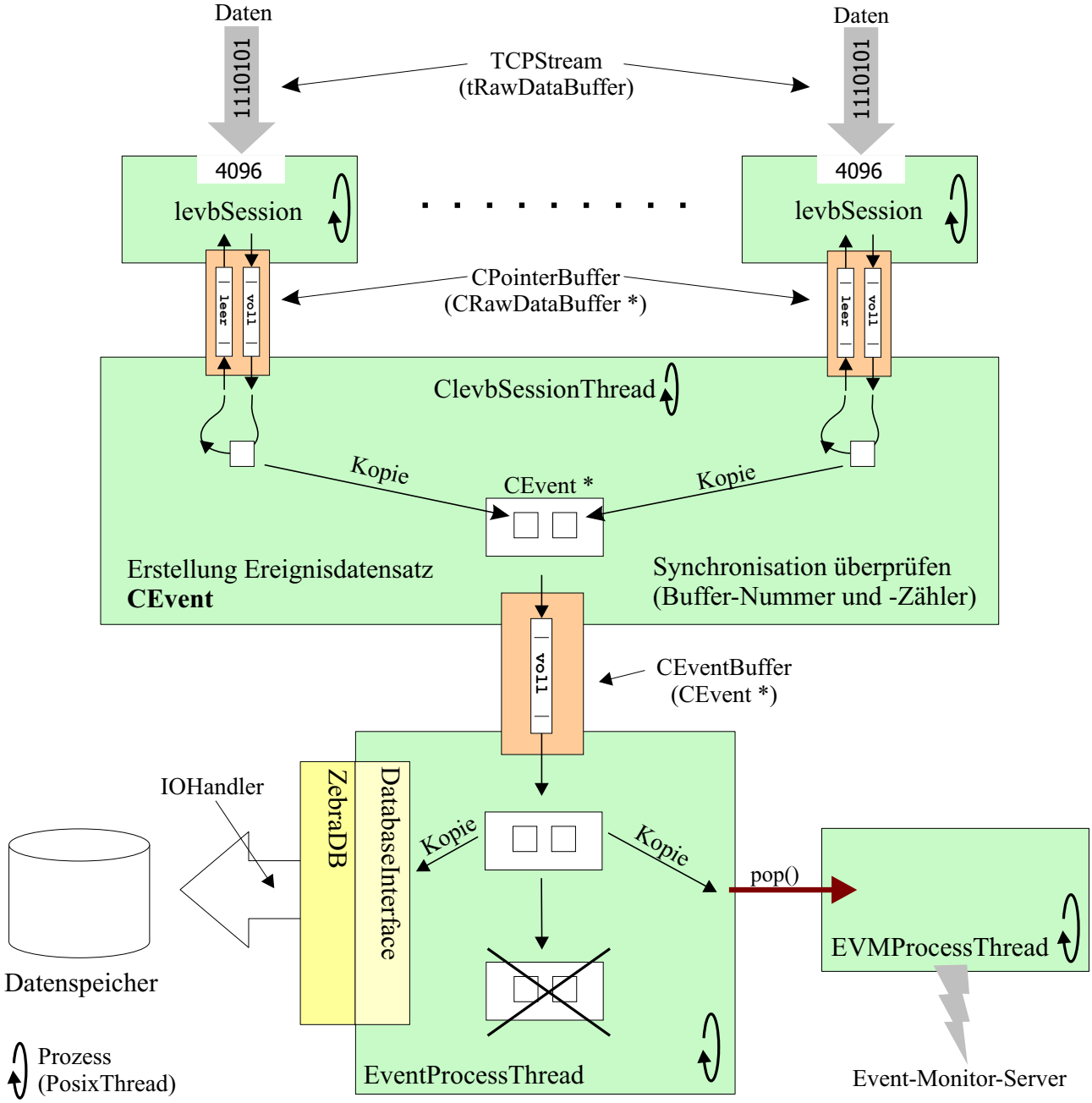


Abbildung 7.24: Schematische Darstellung der Prozesse und Datenwege im Event-Saver

Für jeden aktivierten lokalen Eventbuilder existiert ein `ClevbSession`-Objekt mit zugehörigem `TCPStream`. Der `ClevbSessionThread` verwaltet diese `ClevbSession` Objekte und sammelt die

Daten aus den einzelnen Multibuffern in einem CEvent-Objekt zusammen.

Der Weitertransport der erzeugten Ereignisse erfolgt über die Klasse EventProcessThread. Die CEvent-Objekte gelangen über einen weiteren Buffer (CEventBuffer) vom ClevbSessionThread in den EventProcessThread. Dieser übernimmt die Speicherung der Daten mittels der Schnittstelle IDatabase in ein Datenformat.
Für Monitorfunktionen existiert der Prozess EVMProcessThread. Er verfügt über eine Referenz auf den EventProcessThread und kann eine Kopie eines CEvent-Objektes aus dem Datenstrom herausfiltern und an den Event-Monitor-Server (Programm eventd) weitergeben. Alle aktiven Aufgaben, wie z.B. Daten empfangen oder Daten speichern, sind als Prozesse realisiert, um einen maximalen Grad an Parallelität in der Verarbeitung zu erreichen.

Die Kontrollsteuerung der Prozesse und die Parametereinstellungen werden wie beim lokalen Eventbuilder von der DAQ-Kontrollbibliothek (siehe Kapitel 7.5) übernommen.

7.4.1 Der Datentransport - ClevbSessionThread

Die Klasse ClevbSessionThread ist der Sammelpunkt für alle lokalen Eventbuilder. Bevor jedoch die Funktionsweise von ClevbSessionThread erläutert wird, soll zunächst kurz auf der Verbindungsaufbau (siehe 7.32) der einzelnen lokalen Eventbuilder mit den Server-Session-Prozessen (ClevbSession) eingegangen werden.

Die Verbindungsaufnahme wird angestoßen über die DAQ-Kontrollbibliothek. Diese führt hierzu auf dem Kommandointerpreter des Event-Savers den Befehl "CONNECT" aus, der im Event-Saver im wesentlichen die Funktion ClevbSessionThread::addClient() aufruft. Die Funktion bindet sich zu Beginn an den Port 4096 des Event-Savers und wartet fünf Sekunden auf eine eingehende Verbindung eines lokalen Eventbuilders. Wird in dieser Zeitspanne über den Port 4097 eine TCP/IP-Verbindung aufgebaut, wird ein ClevbSession-Objekt für die Verbindung erzeugt.
Somit existiert für jeden lokalen Eventbuilder ein eindeutiges ClevbSession-Objekt. Diese Klasse ist von von PosixThread [31] abgeleitet und kann als Unterprozess des Event-Savers ausgeführt werden. In der Run() Funktion (siehe Abbildung 7.26) ist das Warten auf Daten des verbunden lokalen Eventbuilders implementiert. Die Übergabe der Daten an den Server-Prozess (ClevbSessionThread) erfolgt über einen Multibuffer auf der Basis von CPointerBuffer, der auch im Datentransport vom lokalen Eventbuilder eingesetzt wird. Wie Abbildung 7.23 zeigt, haben beide Objekte eine Referenz auf diesen Buffer, der von ClevbSessionThread verwaltet und gespeichert wird.
Alle Verbindungen werden so Schritt für Schritt von den einzelnen lokalen Eventbuildern zum

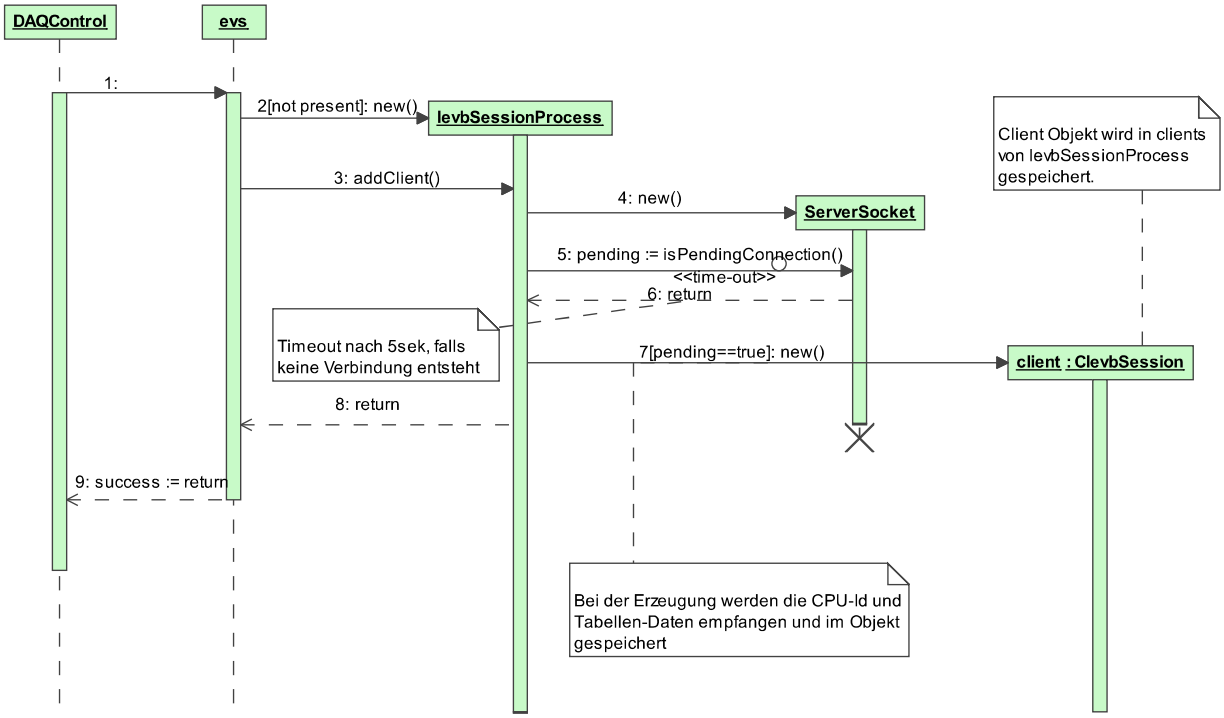


Abbildung 7.25: Sequenz-Diagramm zum Aufbau der Datenverbindung `addClient()`

Event-Saver aufgebaut. Es werden nur Verbindungen von den aktivierten lokalen Evenbuildern erzeugt, die in der DAQ-Kontrolle ausgewählt sind (siehe DAQ-Kontroll-Bibliothek). Nach erfolgreicher Verbindung aller aktivierten lokalen Eventbuilder wird die Datennahme durch die DAQ-Kontrolle gestartet. Dies bewirkt, dass der `ClevbSessionThread` gestartet und somit dessen `Run()`-Funktion ausgeführt wird.

Abbildung 7.27 zeigt den Ablauf in dieser Funktion. Zu Beginn werden die `CPointerBuffer`-Objekte für die einzelnen `ClevbSession`-Objekte erzeugt und die Referenz an das Objekt übergeben. Danach werden ebenfalls die einzelnen `ClevbSession`-Prozesse gestartet: sie beginnen mit dem Ausführen ihrer `Run()`-Funktion (siehe Abbildung 7.26).

Folgende Aktionen werden in einer Endlosschleife[3] ausgeführt:

In der ersten Schleife aus Abbildung 7.27 wird über alle Buffer der lokalen Eventbuilder iteriert und es wird sequentiell jeweils ein Datenkontainer aus dem Multibuffer entnommen.

Erhalten alle lokale Eventbuilder ein Auslesesignal, so steht der Datenblock des jeweiligen Eventbuilders an erster Position im Multibuffer zur Verfügung. Die Abfrage erfolgt über die Funktion `waitFullBuffer()`(5).

Im nächste Schritt (8) wird ein `CEvent`-Objekt erzeugt. Die empfangenen Datenkontainer (`CRawDataBuffer`) werden über die Funktion `addData()` zum `CEvent`-Objekt hinzugefügt (Ko-

[3]Die Schleife bzw. der Prozess kann über die geerbte Funktion Stop() der Basisklasse PosixThread beendet werden.

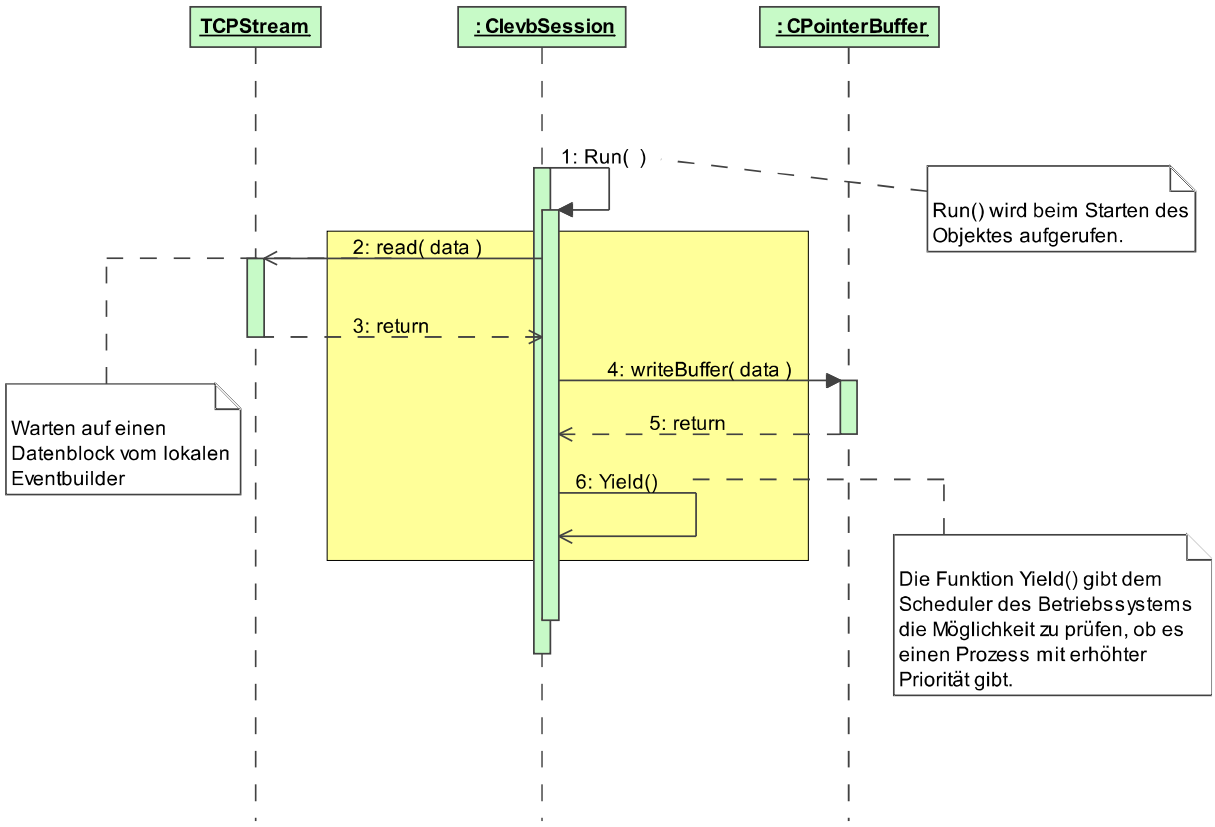


Abbildung 7.26: Sequenz-Diagramm ClevbSession (Funktion Run())

pien der Datenkontainer werden angelegt). Anschließend werden mittels der Funktion `isValid()` (11) die Daten auf Synchronität geprüft. Zur Überprüfung werden die zusätzlichen Informationen herangezogen, die im Datenkontainer (siehe tRawDataBuffer) enthalten sind. Sind alle Daten synchron, d.h. zum gleichen Computer-Trigger-Signal ausgelesen worden, müssen die Zählerstände des Trigger-Signals "Software" und "Hardware" jeweils über alle lokalen Eventbuilder übereinstimmen. Weiterhin muss die vom Sync-/Trigger-Kontroller übertragene Buffernummer in allen Datenblöcken übereinstimmen.
Nach erfolgreicher Kontrolle (isValid()==true) (13) übergibt der `ClevbSessionThread` mit der Funktion `push()` das Ereignis an den `CEventBuffer` zur weiteren Verarbeitung. Von diesem Punkt an wird die Kontrolle (Speicherfreigabe, etc.) über das Objekt CEvent an den Prozess `EventProcessThread` übergeben.
Bei fehlerhafter Überprüfung wird das `CEvent`-Objekt innerhalb von `ClevbSessionThread` gelöscht (15) und eine Pause von einer Sekunde eingelegt. Dies soll das System so verlangsamen, dass zum einen das Logging nicht überfordert wird und zum anderen der Benutzer die Möglichkeit hat, auf den Fehler zu reagieren.

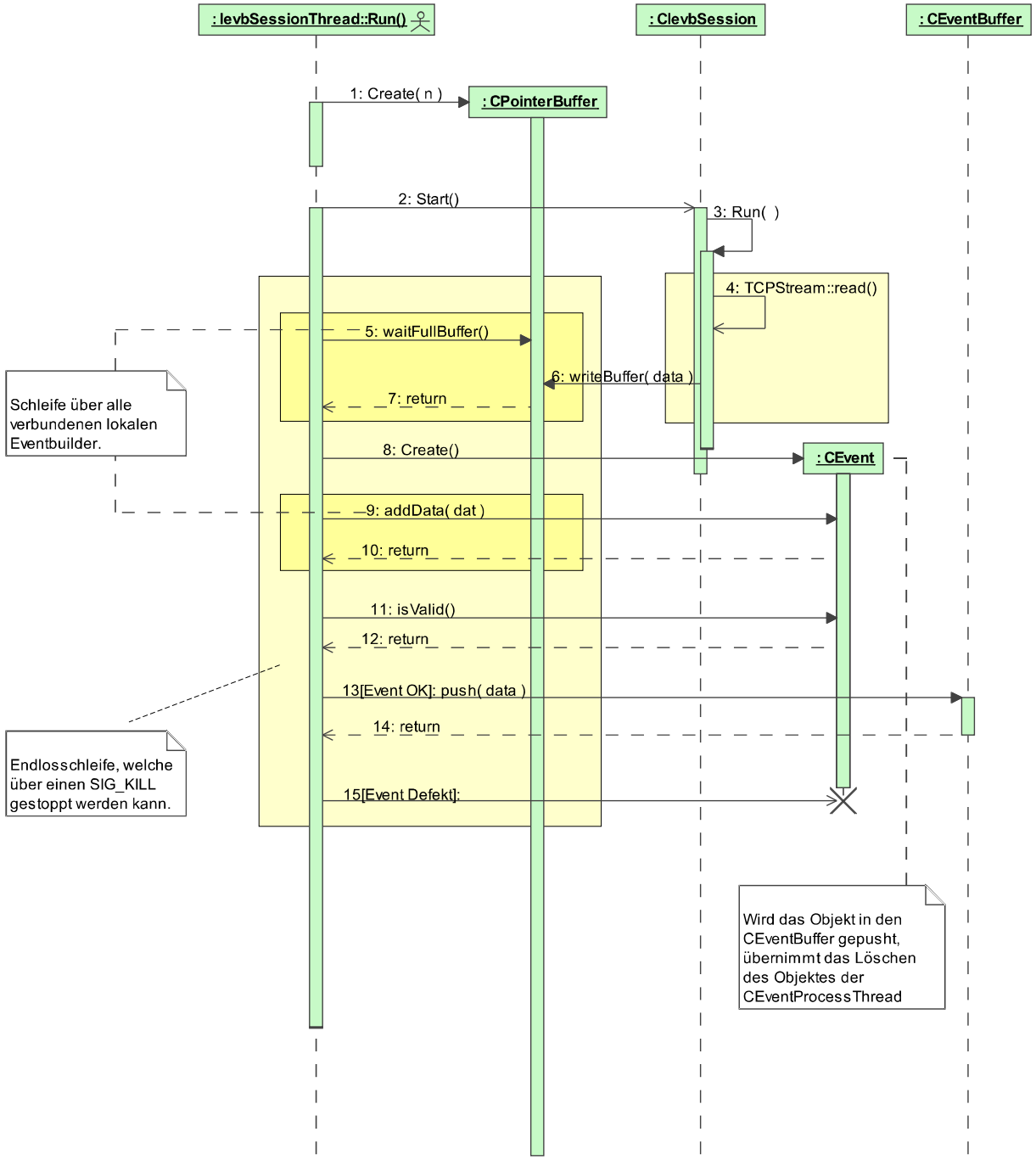


Abbildung 7.27: Sequenz-Diagramm ClevbSessionThread (Funktion Run())

7.4.2 Die Datenbankanbindung - `EventProcessThread,IDatabase`

Die Ereignisse, die im `ClevbSessionThread` erzeugt wurden, werden in einem eigenen Prozess weiterverarbeitet, dem EventProcessThread. Hier wird eine Verteilung der Ereignisse an unterschiedliche Ziele vorgenommen. Zum einen werden die Daten in ein Datenformat über die Schnittstelle `IDatabase` geschrieben, zum anderen kann eine Kopie eines Ereignisses an den

Event-Monitor-Server geschickt werden.
Abbildung 7.28 zeigt den Ablauf innerhalb der Klasse `EventProcessThread`.

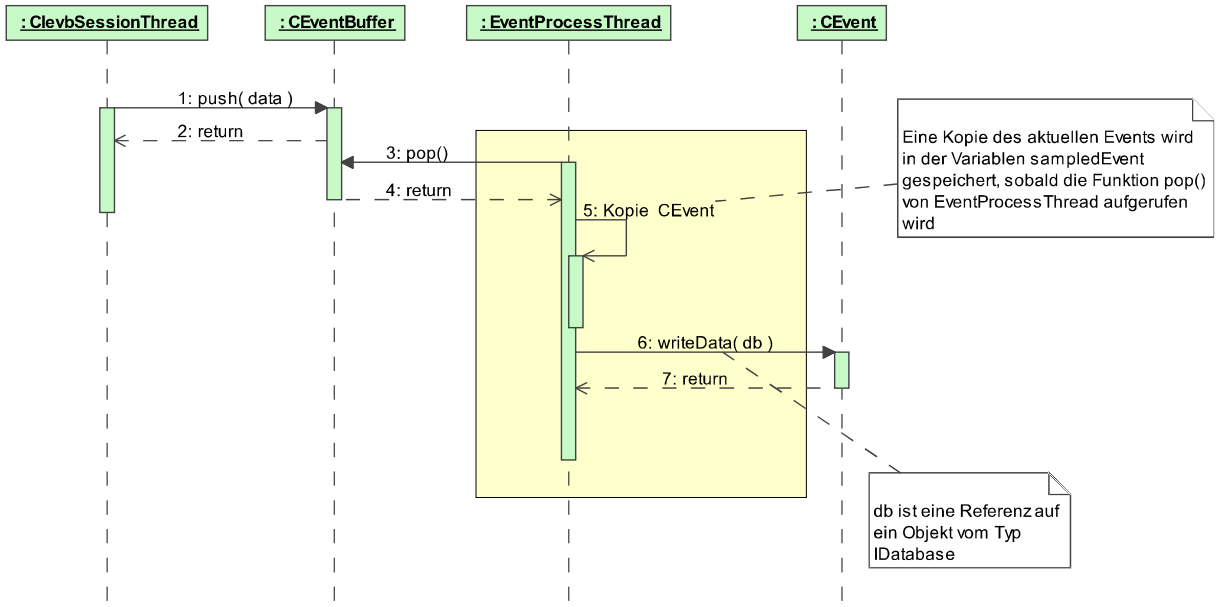


Abbildung 7.28: Sequenz-Diagramm EventProcessThread (Funktion Run())

Die Daten fliessen im Diagramm von links nach rechts. Der Prozess `ClevbSessionThread` übergibt die Ereignisse über die Funktion `push(...)` in den Buffer `CEventBuffer`. Der hierzu parallel laufende Prozess `EventProcessThread` bezieht in einer Schleife über `pop(...)` die zu speichernden Events. Im Anschluss wird, falls notwendig, eine Kopie (5) für den `EVMProcessThread` gemacht und das Ereignis über `writeData(...)` gespeichert (6).

Die Klasse `EventProcessThread` übernimmt die Aufgabe, parallel zur Erstellung der `CEvent`-Objekte die Rohdateninformationen über die Schnittstelle `IDatabase` in ein Datenbankformat zu konvertieren. Bis zu diesem Punkt wurde am Datenformat, so wie es die Auslese der Frontend-Elektronik vorgibt, nichts geändert, denn der Schwerpunkt liegt hier auf einer optimalen und schnellen Ausleseprozedur. Für eine spätere Analyse durch umfangreiche Softwarepakete muss jedoch dann ein auf diese Anforderungen angepasstes Datenformat verwendet werden.
Die Anforderungen ergeben sich aus Sicht des Anwenders der Analyse-Programme:

- Standardisierter Zugriff auf die Daten
- Strukturierbarkeit der Informationen in logische Blöcke (Subdetektoren)
- Verarbeitbarkeit von variablen Datenblocklängen

- Konsistenzprüfung der Daten bei der Speicherung
- Kompressionsalgorithmen zur Minimierung des Speicherbedarfs
- Automatische Erkennung der Strukturänderungen der Informationen eines Subdetektors

Bei klassischen Anwendungen wie Kundenverwaltungen o.ä. wird zur Speicherung der Daten in einer Datenbank eine standardisierte Schnittstelle zum Datenbanksystem verwendet. In diesem Bereich existieren Standards wie ODBC[4] oder JDBC[5]. Diese ermöglichen den erstellten Anwendungen, vom verwendeten Datenbanksystem unabhängig zu werden, sodass je nach den Anforderungen das entsprechende System gewählt werden kann. Diese Möglichkeit ist in der Datenakquisition durch die Schnittstelle `IDatabase` realisiert worden.
Abbildung 7.29 zeigt die Funktionen in der Schnittstelle, sowie zwei Implementationen von Datenformaten (ZEBRA und ROOT).

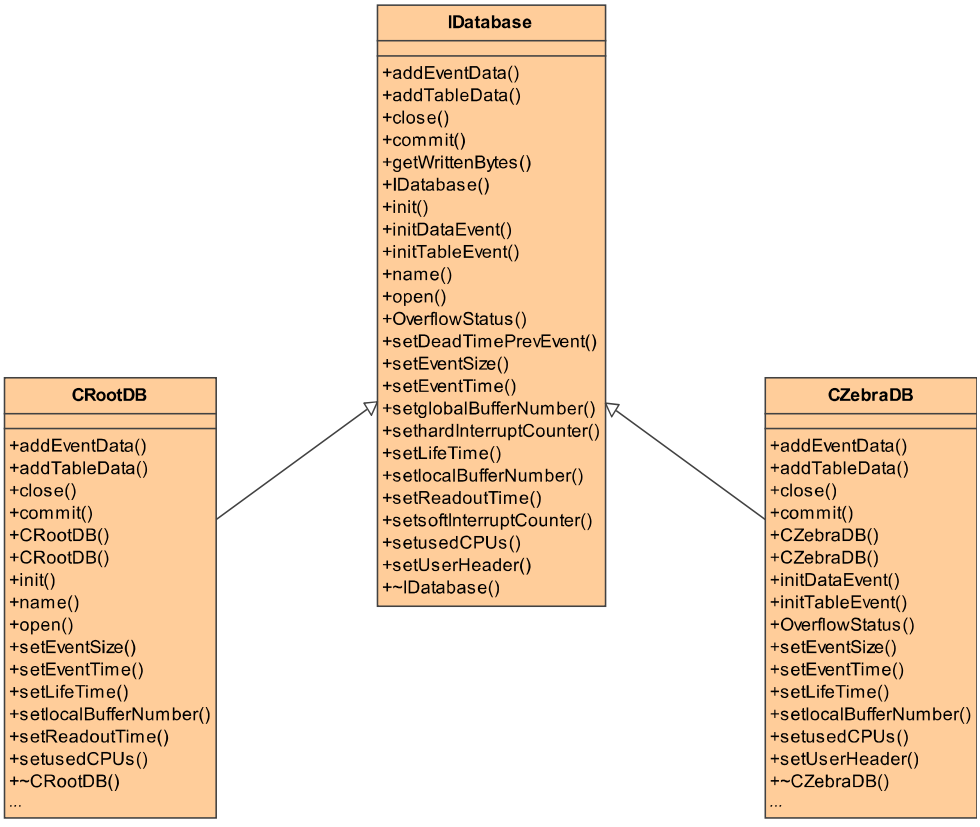


Abbildung 7.29: UML-Vererbungsdiagramm bei der Datenbankanbindung `IDatabase`

Die Schnittstelle teilt sich in drei Bereiche auf, die hier kurz mit ihren Funktionen erwähnt werden (Eine ausführlichere Beschreibung befindet sich in der CB-DAQ-Referenzdokumentation [26]).

[4] **O**pen **D**ata**B**ase **C**onnectivity
[5] **J**ava **D**ata**B**ase **C**onnector

1. Initialisierung und Öffnen der Datenbank

   `init(...)` :

   Diese Funktion dient zur Vorbereitung der Datenbankschnittstelle; es darf kein Öffnen einer Datei enthalten sein.

   `open(...)` :

   Öffnen der entsprechenden Datei auf dem Speichermedium

   `close(...)` :

   Schließen der Datenbank

2. Hinzufügen und Schreiben von Datenbänken

   `initDataEvent(...), initTableEvent(...)` :

   Initialisieren und Hinzufügen eines neuen Datensatzes an die Datenbank

   `addEventData(...), addTableData(...)` :

   Hinzufügen von Datenblöcken an den aktuellen Satz der Datenbank

   `commit()` :

   Sind alle Daten über `addEventData` hinzugefügt worden, werden diese per `commit()` in die Datenbank geschrieben.

3. Setzen von DAQ-Zusatzinformationen

   Für die Zusatzinformationen wie Totzeitzähler, Buffernummer, etc. existieren zusätzliche Funktionen für jeden Typus (z.B. `setEventTime(...)`,...).

Wie Abbildung 7.29 zeigt, existieren zwei Implementationen für diese Schnittstelle:

1. ZEBRA

   Dieses Datenformat wird in der derzeitigen Auswertungssoftware eingesetzt. Das hat historische Gründe. Da eine umfangreiche Auswertungssoftware des vorherigen Experimentes am LEAR existierte, wurde diese auf die neue Situation an ELSA angepasst und das Dateiformat ZEBRA somit weiterverwendet (siehe Anhang A.1). Im wesentlichen werden dabei die Daten in einer definierten Struktur abgelegt und in Blöcken zu je 23050 Bytes in einer Datei gespeichert, wobei sowohl die Rohdaten als auch die Resultate der Analysesoftware im gleichen Format strukturiert werden, um die Einheitlichkeit des Zugriffs auf die Daten zu gewährleisten. Es wird eine neue Kodierung in C++ verwendet, um unnötige Abhängigkeiten von betriebssystemfremder Software (Kompilation und Linken) zu vermeiden, da die CERN-Bibliotheken in FORTRAN vorliegen. Diese Implementation war Bestandteil der vorherigen DAQ [19] auf Basis von OS9 und konnte daher übernommen und mit Hilfe der Schnittstelle wieder verwendet werden.

2. ROOT

   ROOT ist eine Software-Entwicklungsumgebung zur Erstellung von Histogrammen, Analysen, etc.; sie ist vollständig in C++ objektorientiert entwickelt worden. Dieses Programmpaket bietet zusätzlich ein neues Datenformat an. Dieses enthält noch weit reichendere Möglichkeiten als das ZEBRA-Format. So können z.B. selbst definierte Objekte in diesem Datenformat gespeichert werden. Weiterhin erlaubt es eine Kompression der Daten während des Schreibvorgangs ("on the fly").

Zur Datennahme wird zur Zeit das ZEBRA-Format (`CZebra`) verwendet, da (wie schon erwähnt) die komplette Analyse-Software dieses Datenformat verwendet. In der Klasse `CRootDB` sind zum Test der Realisierbarkeit die Daten des Datenakquisitionssystems (Totzeitzähler, etc.) implementiert worden. Eine Erweiterung auf alle Rohdatenbänke sollte relativ leicht möglich sein, wobei hier noch Entscheidungen über die Art der Struktur der Daten im ROOT-Dateiformat getroffen werden müssten.
Die hier gewählte Softwarerealisation zeigt somit eine hohe Flexibilität in der Wahl des Datenformats, das mit relativ geringem Aufwand an neue Erfordernisse des Experimentes angepasst werden kann.
Nach der Formatierung der Datensätze verbleibt nur noch die Aufgabe, die Daten auf einem geeigneten Medium zu speichern. Die Daten in der Vorgänger-DAQ des Experimentes wurden auf DLT-Magnetbändern gespeichert, für Testzwecke konnte auch auf Festplatte geschrieben werden. Um dabei ebenfalls unabhängig von der unterschiedlichen Ansteuerung des verwendeten Mediums zu werden, wurde die Schnittstelle `IOHandler` eingeführt. Dabei sind deren Bestandteile lediglich die Grundfunktionen open(), read(), write() und close(). Hiermit ist es möglich, jede Art von Schreibvorgang abzubilden.

Drei Implementationen auf der Basis dieser Schnittstelle existieren:

1. CFileHandler

   Diese Klasse kapselt die Ein-/Ausgabe-Funktionen des Betriebssystems. Es handelt sich hierbei um eine "eins-zu-eins"-Abbildung, d.h. die Funktionen bestehen lediglich aus den Aufrufen der zugehörigen Betriebssystemfunktionen.

2. CTapeHandler

   Da Bandlaufwerke als hochdichtes Langzeitspeichermedium auf Grund ihres sequentiellen Charakters einen völlig anderen Zugriffsablauf haben, müssen zu ihrer Bedienung zusätzliche Funktionen eingeführt werden. So sind z.B. Funktionen notwendig zur Identifizierung der Magnetbänder, um ein ungewolltes Überschreiben von bereits gespeicherten Informationen zu vermeiden. Zu diesem Zweck tragen die Bänder sogenannte Labels, die durch eine spezielle Initialisierung auf das Band geschrieben werden.

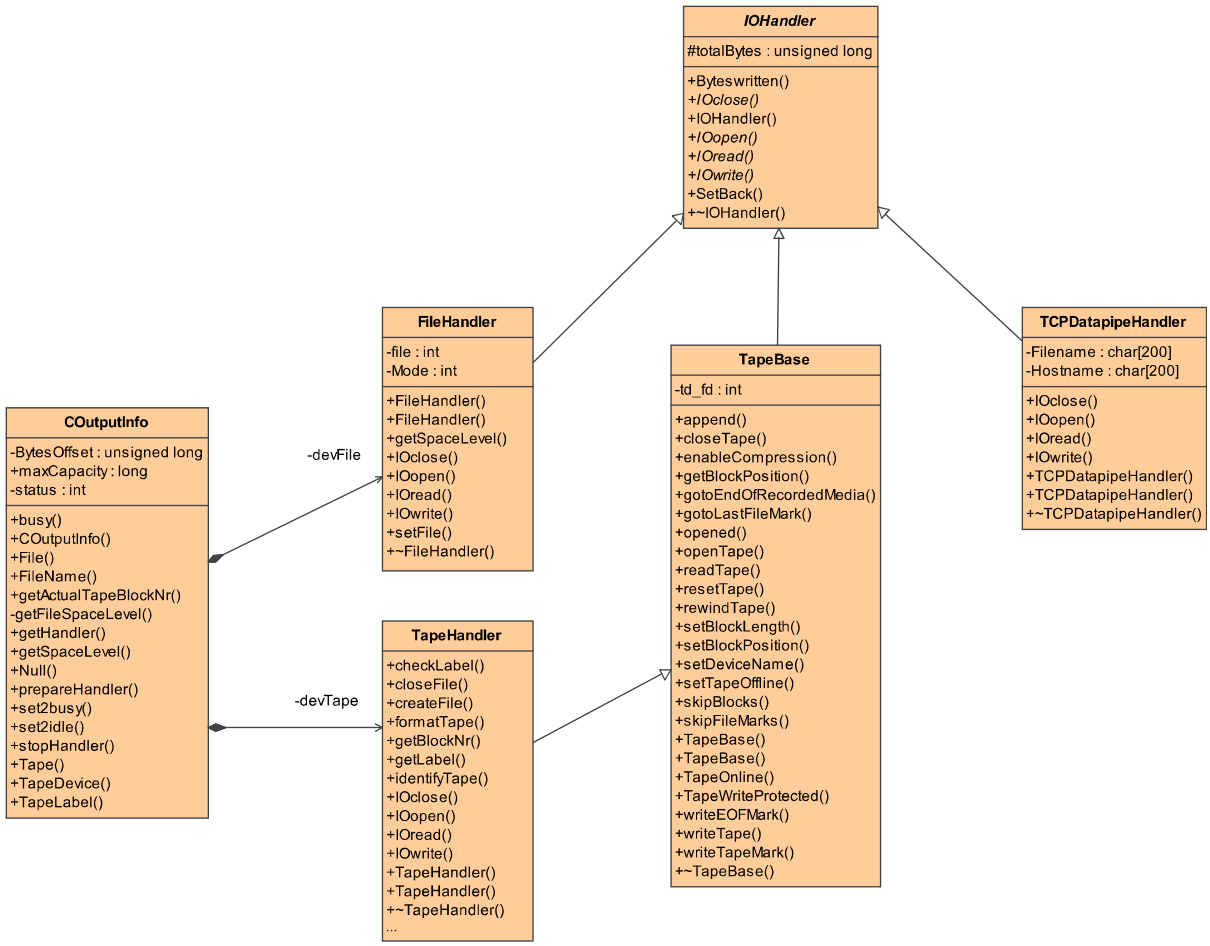


Abbildung 7.30: UML-Vererbungsdiagramm für die Schnittstelle `IOHandler`

3. TCPDatapipeHandler

   Dieser Handler erlaubt es, die Daten über das Standard-Netzwerk (100MBit oder 1GBit) zu einem speziellen Server zu senden, der sie dort lokal auf Festplatte speichern kann. Der Zweck dieses Transportkanals ist es, weitere Softwareschichten zu umgehen, wenn die Daten auf entfernten Rechnern mittels der Standardverfahren gespeichert werden sollen (z.B. importierte Festplatten über NFS[6]). Da im Falle des Datenakquisitionssystems nur eine Punkt-zu-Punkt-Verbindung innerhalb einer festgelegten Clusterarchitektur notwendig ist (Daten-Server und Event-Saver liegen fest), gewinnt man deutlich an Übertragungsgeschwindigkeit.

Die Gesamttransferleistung des vorgestellten modularen Systems wird in Abschnitt 8.2 ausführlich diskutiert werden.

[6]Network File System

7.5 Die DAQ-Kontrolle - Die Bibliothek `libdaqctrl.a`

Die Abschnitte zuvor ergeben, dass das Gesamt-Datennahmesystem aus einer großen Anzahl von individuellen Rechnersystemen und den dazugehörigen Prozessen besteht. Ihre Kontrolle und Überwachung ist ein besonders wichtiger Punkt. Die Reihenfolge des Starts der Prozesse ist dabei ein komplexes Gebilde, dass von einem zentralen Punkt aus gesteuert werden muss. Die definierten Sequenzen können nicht manuell ausgeführt werden und sind mit geeigneten Status- und Fehlerprozeduren zu versehen, um das verteilte Rechnersystem stabil zu betreiben. Zu ihrer Durchführung existiert eine Bibliothek, die die notwendigen Funktionen zur Steuerung und Konfiguration des Gesamtsystems zur Verfügung stellt. Sie kann auf einem dedizierten Rechner in einem Kontrollprogramm ausgeführt werden.
Bevor aber auf Abläufe und mögliche Zustände im System eingegangen wird, soll kurz die technische Realisierung erläutert werden. Bezüglich des Designs der Kommunikation in verteilten Rechnersystemen existieren Standards, die es ermöglichen, über Definitionen von Schnittstellen (sogenannte IDLs[7]) Funktionen auf andern Rechnersystemen auszuführen. Zum einen gibt es die Technik der **R**emote **P**rocedure **C**alls (RPCs), ein anderer Standard ist CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture), der einen objektorientierten Ansatz enthält. Leider lassen beide Standards Interpretationsvarianten zu, sodass Inkompatibilitäten entstanden sind, die ein Mischen von Programmbibliotheken verschiedener Softwarehersteller nicht möglich machen.

Die Erfahrungen mit dem vorherigen Datenahmesystem und ein erster Einsatz dieser Methoden zeigten, dass sie sehr umständlich und fehleranfällig in der Ausführung sind, sodass ein stabiler Langzeitbetrieb des vernetzten Systems nicht erreicht werden konnte. Um dies zu realisieren, wird eine einfachere Technik verwendet. Diese wird bei vielen TCP/IP-Netzwerk-Diensten (z.B. HTTP,Mail,...) eingesetzt und beruht auf einer TCP/IP-Verbindung, über die der Datenaustausch auf Basis von ASCII-Kommandos realisiert ist.
Der Aufruf von Funktionen auf den verteilten Rechnern wird mittels einer einfachen TCP/IP-Verbindung und durch das Senden von ASCII-Kommandos durchgeführt (In Kapitel 7.2.7 wurde bereits ein Beispiel für eine solche Verbindung gezeigt). Die ASCII-Kommandos werden in einem einfachen Kommandointerpreter verarbeitet, d.h. diese Funktion analysiert eine Zeichenkette, die in einem Zwischenspeicher abgelegt ist. Ein solcher Übergabemodus erlaubt es, mittels einfacher Methoden mit dem Benutzer zu kommunizieren. Die ASCII-Kommandos können z.B. über die Tastatur dem Programm übergeben werden. Dadurch ist es leicht möglich den lokalen Eventbuilders zu testen. Ein anderer Weg ist es, über ein Terminalprogramm (z.B. telnet) diese Kommandos mittels einer TCP/IP-Verbindung an den lokalen Eventbuilder zu senden.

Abbildung 7.31 zeigt schematisch die möglichen Verbindungen der implementierten Bibliothek

[7] **I**nterface **D**efinition **L**anguage

zu den einzelnen Rechnersystemen.

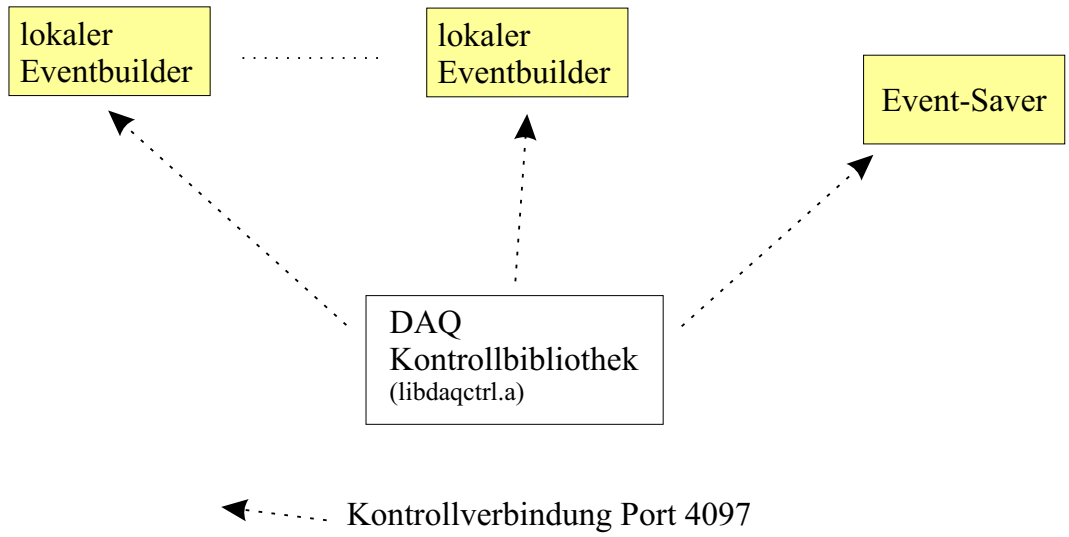


Abbildung 7.31: Kontrollverbindungen der DAQ-Kontrollbibliothek

Die Kontrolle über z.B. acht lokale Eventbuilder und den Event-Saver kann nicht durch den Benutzer des Datennahmesystems erledigt werden. Ihre Aufgaben und Abläufe sind in der Programmbibliothek `libdaqctrl.a` zusammengefasst, und die Klasse `CDAQControlCenter` stellt die Funktionalitäten zur Verfügung.
Auch hier wurden die auszuführenden Funktionen, die zur Steuerung des kompletten Datenakquisitionssystems notwendig sind, in einer Schnittstelle `IDAQControl` definiert, um unabhängig von der jeweiligen Implementation des DAQ-Kontroll-Mechnismuses zu sein. Dies erlaubt es, eine Änderung in dem Kontroll-Mechanismus ohne Anpassung der jeweiligen Programme, die diese Schnittstelle nutzen, durchführen zu müssen. Die Verbindung zu den Rechnersystemen wird im Konstruktor der Klasse `CDAQControlCenter` durchgeführt.
Die Bedienung dieser Klasse wird über die Schnittstelle `IDAQControl` definiert. Hier ein kleiner Ausschnitt aus der Schnittstelle `IDAQControl` über mögliche Funktionen:

```
class IDAQControl {
public:
...
   //!< Aktivieren eines lokalen Eventbuilder für die Datennahme
   virtual void activateLEVB(int id) = 0;
   virtual void StartDAQ() = 0; //!< Starten DAQ
   virtual void StopDAQ() = 0; //!< Stoppen DAQ
...
};
```

Fehlersituationen, die während der Ausführung der Funktionen auftreten können, werden über Fehlerausnahmen (Exceptions) mitgeteilt. Hierbei handelt es sich um ein Standardverfahren der Sprache C++, um Fehlerbehandlungen an einer definierten Stelle im Quelltext auszuführen. Die Schnittstelle wird in den Programmen `daqcontrol` und `runcontrol` (siehe Abbildung 7.36) angewendet, die die Benutzerschnittstelle des Experimentes für den Experimentator darstellt. Voraussetzung für die Verbindungsaufnahme ist, dass die notwendigen Programme gestartet sind. Hierzu existiert ein Skript (`daq`) auf Kommandozeilenebene, das die Programme über eine sichere Netzwerkverbindung (ssh) zu den jeweiligen Prozessorkarten der Subdetektoren automatisch ausführt. Nach dem Start des Grundsystems befinden sich alle Programme im Ruhezustand und sind bereit für Kontroll- oder Datenverbindungen. Die möglichen Zustandsänderungen der einzelnen Systeme und die jeweiligen Befehle oder Aktionen werden in Abbildung 7.32 und 7.33 gezeigt.

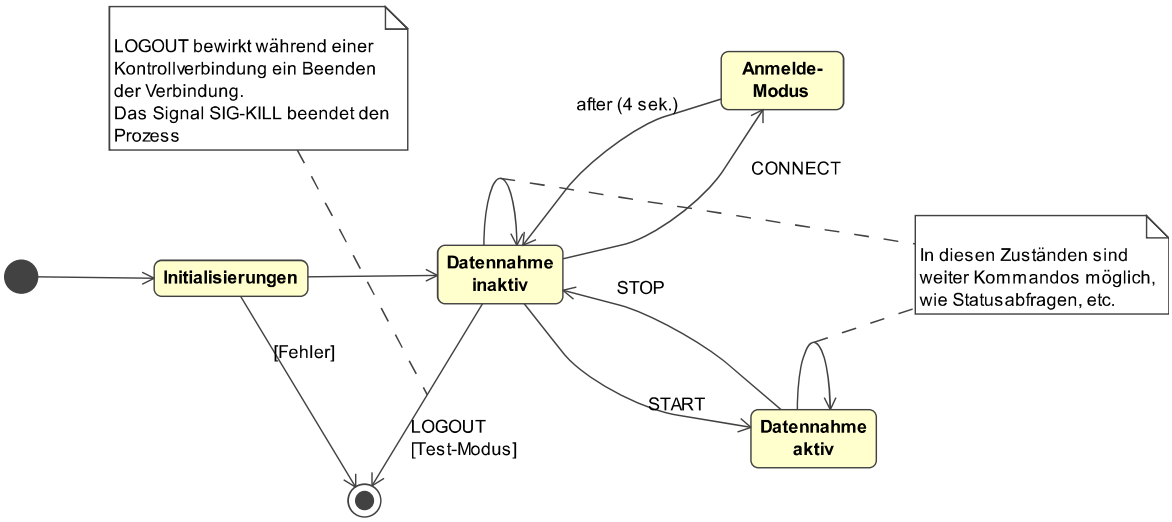


Abbildung 7.32: UML-Zustands-Diagramm des Event-Savers

Die Zustandsdiagramme des lokalen Eventbuilders und des Event-Savers ähneln sich im Grundaufbau, unterscheiden sich aber in einigen Details. Diese sollen im folgenden kurz erläutert werden.

Abbildung 7.32 zeigt die Zustände des Event-Savers. Zu Beginn des Prozesses wird eine Initialisierung durchgeführt. Ist diese beendet, wechselt der Event-Saver in einen Bereit-Modus, in dem die Datennahme in einem inaktiven Zustand ist. Es gibt im wesentlichen zwei Zustände: den"Anmelde-Modus" zur Initialisierung der Datenverbindungen zwischen Event-Saver und lokalem Eventbuilder und den "Datennahme-aktiv"-Zustand. Der Anmelde-Modus wird über den Befehl `CONNECT` erreicht und wird automatisch nach vier Sekunden wieder verlassen. Der Datennahme-Zustand kann über `START` und `STOP` gewechselt werden. Weitere Zusammenhänge mit den lokalen Eventbuildern werden in den nächsten Abschnitten erläutert.

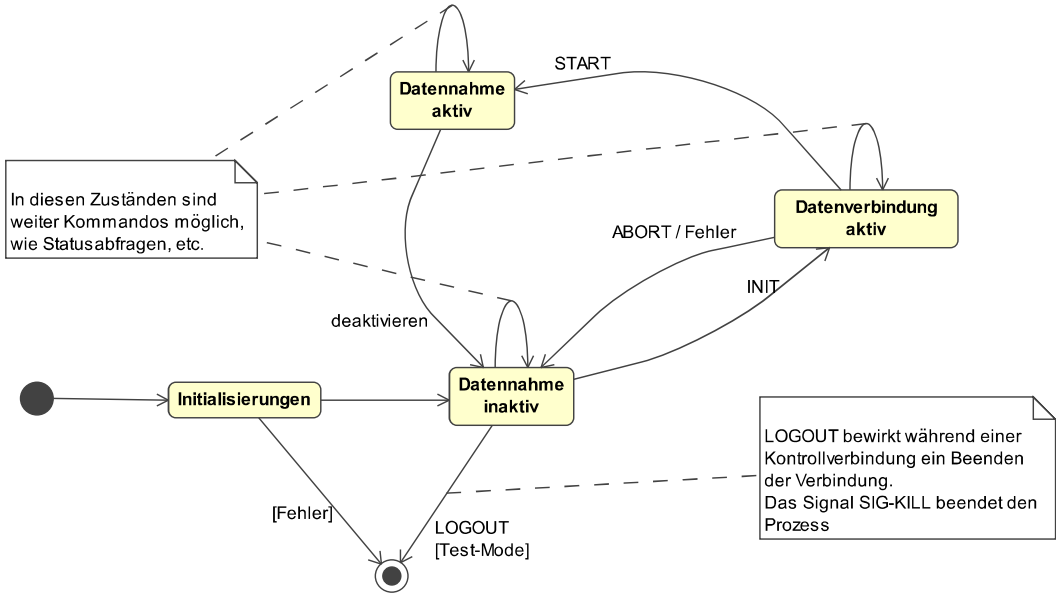


Abbildung 7.33: UML-Zustands-Diagramm des lokalen Eventbuilders

Beim lokalen Eventbuilder in Abbildung 7.33 existieren auch die Zustände "Datennahme aktiv" und "Datennahme inaktiv", jedoch unterscheiden sich die Übergänge in einigen Punkten. Die Datennahme kann nur in den "aktiv"-Zustand übergehen, wenn vorher die Datenverbindungen zum Event-Saver initialisiert sind. Die Initialisierung erfolgt über den Befehl `INIT`. Befindet sich der lokale Eventbuilder im Zustand "Datenverbindung aktiv", kann mittels des Befehls `START` die Datennahme aktiviert oder mittels `ABORT` abgebrochen werden. Dieses Verhalten ist im Grundmodul des lokalen Eventbuilders und im Event-Saver implementiert.

Im folgenden sollen nun die wesentlichsten und komplexesten Funktionen `Start()` und `Stop()`.

Die Funktion StartDAQ()

In Abbildung 7.34 ist das Sequenzdiagramm für die Funktion StartDAQ() gezeigt. Zu sehen ist ein Beispiel aus einem lokalen Eventbuilder und der zeitliche Verlauf der jeweiligen Objekte. Zu Beginn der Funktion befinden sich alle Programme im Zustand "Datennahme inaktiv".
Über die Befehle USER SETACTIVE (1) und DB (2) werden Initialisierungseinstellungen am System vorgenommen. `USER SETACTIVE` veranlasst den Sync-/Trigger-Kontroller (evb), die Sync-Module der gewünschten lokalen Eventbuilder zu aktivieren. Über das Kommando `DB` wird der Name der Datendatei für die Speicherung der Eventdaten der einzelnen lokalen Eventbuilder im Event-Saver (evs) gesetzt.

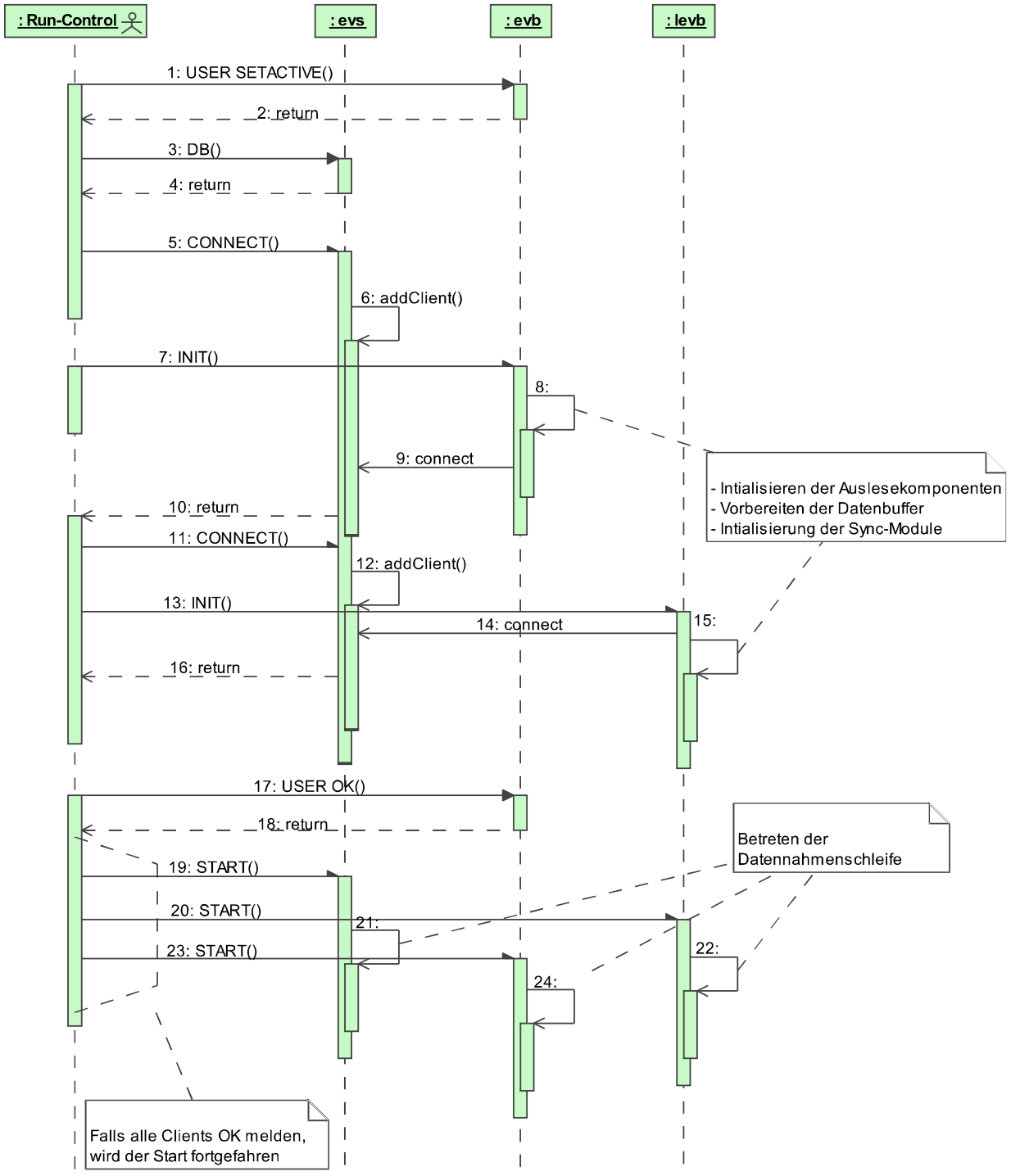


Abbildung 7.34: Sequenzdiagramm zur Funktion `StartDAQ()`

Die Sequenz von (5) bis (17) dient zum Aufbau der Datenverbindungen vom Sync-/Trigger–Kontroller (evb) und einem lokalen Eventbuilder (levb) zum Event-Saver (evs). Über den Befehl `CONNECT` (5) wechselt zunächst der Event-Saver in den Zustand des "Anmelde-Modus" (siehe Abbildung 7.32). Nun kann mittels `INIT` (7) beim Sync-/Tigger-Kontroller der Aufbau der Da-

tenverbindung veranlasst werden (siehe Abbildung 7.33). Ist die Verbindung aufgebaut worden oder hat eine Zeitüberschreitung stattgefunden, wechselt der Event-Saver wieder in den Grundzustand über. Bei erfolgreicher Verbindung befindet sich der lokale Eventbuilder im Zustand "Datenverbindung aktiv" und ist bereit zum Start des Datennahmeprozesses. Diese Abfolgen werden bei den weiteren lokalen Eventbuildern durchgeführt (siehe 11-16). Sind alle Datenverbindungen erfolgreich hergestellt, wird über das Kommando `USER OK` (17) die Bereitschaft der lokalen Eventbuilder kontrolliert. Melden alle den Status OK, werden zuerst der Event-Saver, dann alle lokalen Eventbuilder und zum Schluss der Sync-/Trigger-Kontroller (evb) mit dem Befehl "START" gestartet. Mit diesem letzten Start-Aufruf beginnt die Datennahme.

Die Funktion StopDAQ()

Das Stoppen der Datennahme ist einfacher aufgebaut (siehe Abbildung 7.35), hat aber eine wesentliche Besonderheit.

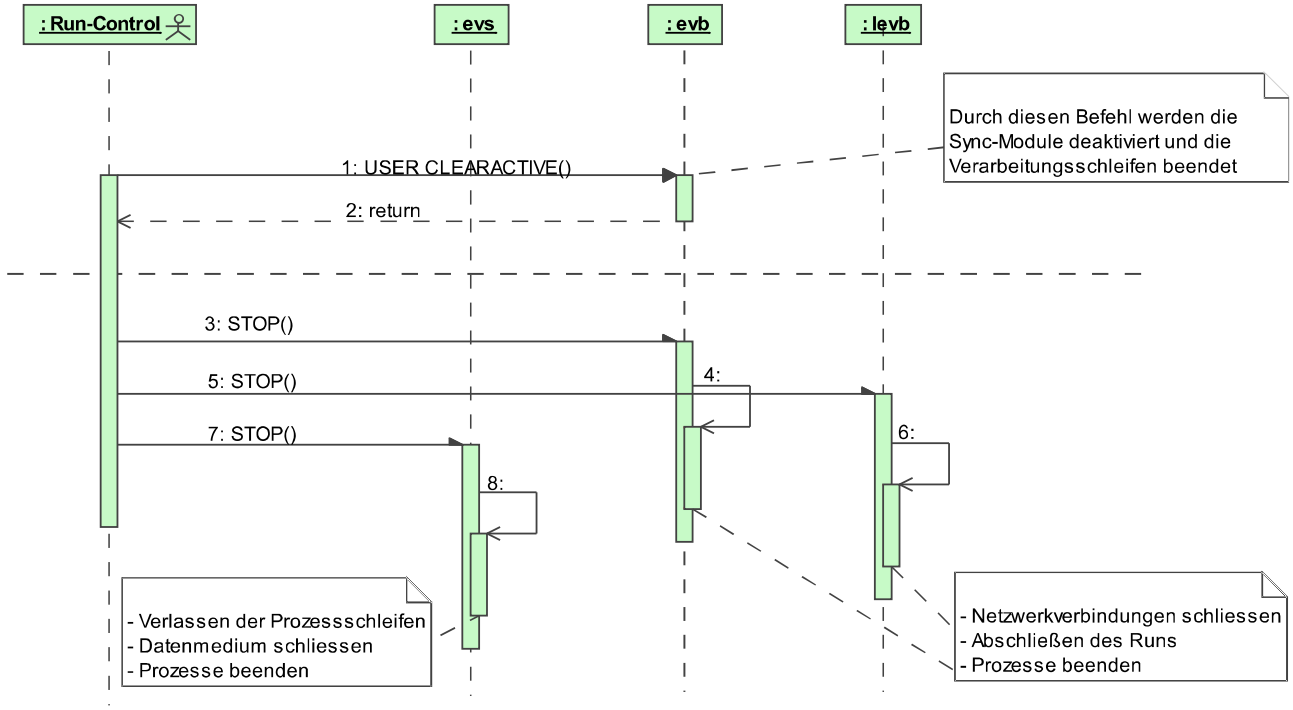


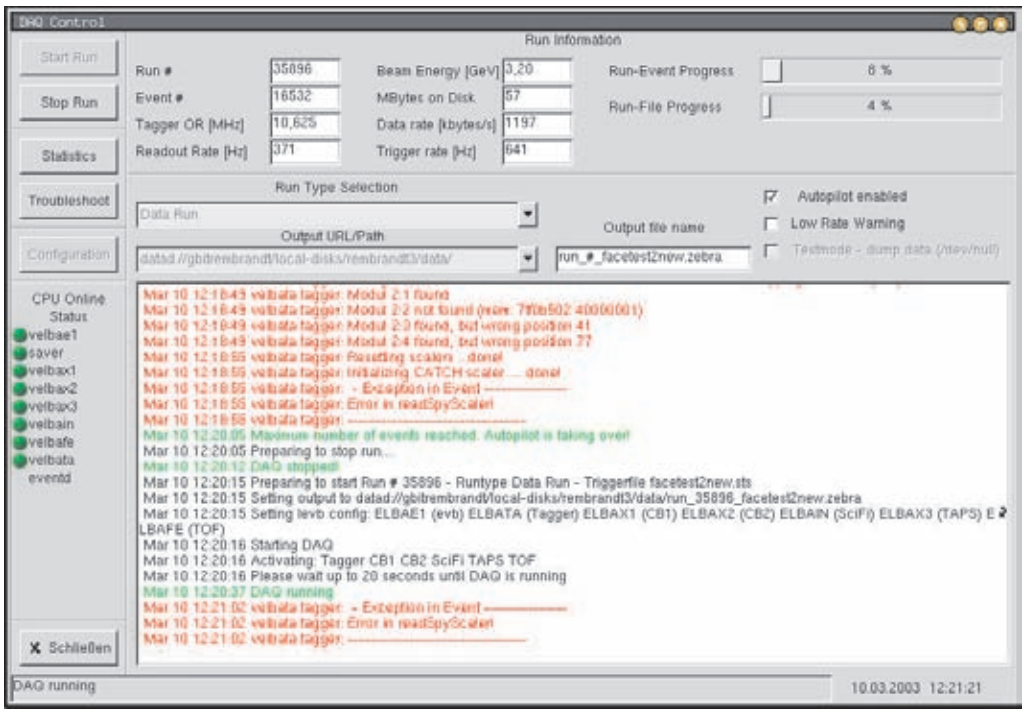
Abbildung 7.35: Sequenzdiagramm zur Funktion `StopDAQ()`

Das Stoppen, bzw. das Verlassen der Datennahmeschleifen in den einzelnen lokalen Eventbuildern wird durch den Befehl `USER CLEARACTIVE` (1) über den Sync-/Trigger-Kontroller hervorgerufen. Durch das Löschen der Aktivierungen der Sync-Module der einzelnen lokalen Eventbuilder wird der `ReadoutThread` (siehe Abschnitt 7.2.3) unterbrochen, da zu Beginn jeden Ereignisses überprüft wird, ob das jeweilige Client-Syncmodule noch aktiviert ist. Ist der Befehl `USER CLEARACTIVE` an dem Sync-/Trigger-Kontroller erfolgt, wird von diesem der Trigger nicht

mehr geöffnet und eine Pause von zwei Sekunden durchgeführt, um den jeweiligen `DataThread`s auf den lokalen Eventprozessoren Zeit zu geben, ihre Datenpuffer zu leeren.
Im Anschluss werden die noch laufenden Prozesse auf den lokalen Eventbuildern mit STOP (3),(5) beendet. Unter anderem wird der Prozess `DataThread` verlassen, der daraufhin die Datenverbindung zum Event-Saver schließt. Am Schluss der Sequenz werden alle Prozesse des Event-Savers beendet. Nun befinden sich alle Programme im Grundzustand "Datennahme inaktiv". Den Abschluss bildet die Rückmeldung an das DAQ-Kontrollprogramm, dass die Datennahme abgschlossen ist.

Trotz der Komplexität der zeitlichen Abläufe der zahlreichen Prozesse ist es mit der gewählten Hard- und Software-Implementation gelungen, ein stabiles und funktionstüchtiges Kontroll- und Steuersystem auch für den Experimentbenutzer zu erstellen, der kein "DAQ-Experte" ist. Die vergangenen Datennahmenperioden in den Jahren 2002 und 2003 ($\sim$ 3000 Strahlstunden) haben die Richtigkeit des gewählten Konzepts gezeigt.

Programm "daqcontrol":



Programm "runcontrol":

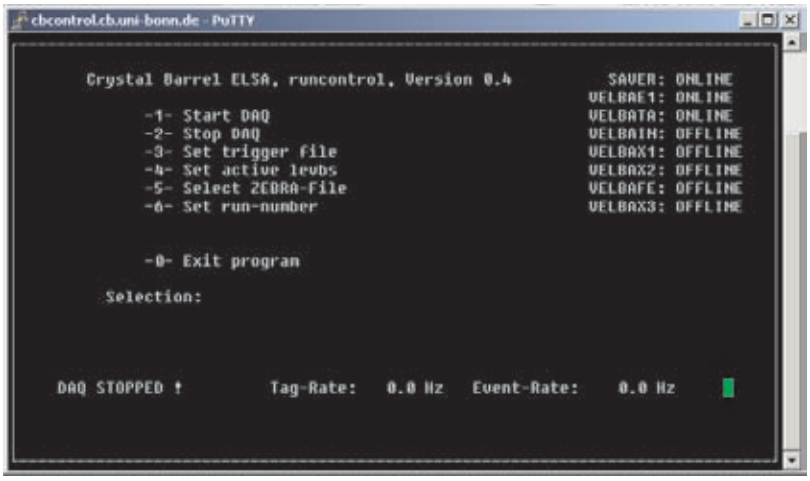


Abbildung 7.36: Programme zur Steuerung des Datennahmesystems

7.6 Die Monitor-Server-Anbindung - Die Bibliothek `libevm.a`

Es soll noch auf eine zusätzliche Option im Datenakquisitionssystem eingegangen werden, die speziell bei Experimenten mit langen Datennahmeperioden von großer Bedeutung ist. Die Überwachung der Funktionsweise der einzelnen Subdetektoren schon während der Messphasen ist eine zwingende Voraussetzung für eine erfolgreiche Datennahme, da der elektronische Ausfall von Teilen eines Subdetektors entscheidend die Qualität der aufgezeichneten Eventinformation verschlechtern kann, wenn nicht gar diese Eventinformation unbrauchbar wird. Daher muss die Möglichkeit bestehen, mit geeigneten Analyse- und Histogrammprogrammen "online" die Performance der Subdetektoren zu überwachen.

Dies erfolgt über einen weiteren Serverprozess (eventd), der über den Event-Saver mit Evnet-Informationen versorgt wird. Dieser sogenannte Monitor-Server übernimmt die Replizierung der Datenblöcke an die jeweiligen Monitor-Clients (siehe Abbildung 7.37), die die Online-Analyse der Eventdaten durchführen. Um eine flexible Verteilung solcher Eventdaten auf mehrere Analyseprogramme zu gewährleisten, wurde eine TCP/IP-Netzwerkverbindung gewählt, die auf jedem Rechner des Institutsnetzes etabliert werden kann.

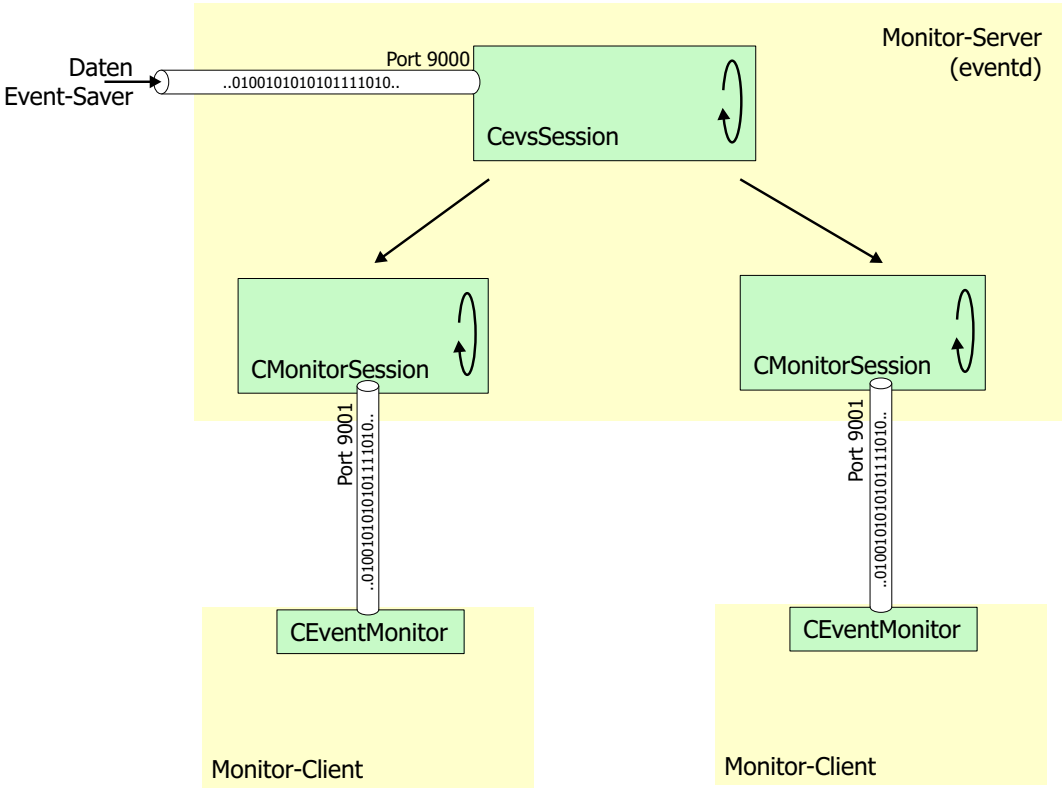


Abbildung 7.37: Aufbau des Monitor-Servers `eventd`

Die Anbindung der Client-Programme an den Monitor-Server ist über eine einfache Schnittstelle realisiert. Diese Programme verwenden die Klasse `CEventMonitor`, die sich in der Bibliothek `libevm.a` befindet, und erhalten hierdurch einen Online-Zugriff auf die Daten des Experimentes. In dieser Schnittstelle existieren u.a. Funktionen zum Abrufen von kompletten Ereignissen aus dem Datenstrom (`getEvent()`).
Auf Basis dieser Bibliothek sind ein grafisches Frontend (siehe Abbildung 7.38) mit der ROOT-Softwarebibliothek und ein Konsolenprogramm (`dumpevent`) entwickelt worden, wobei letzteres die Daten auf dem Niveau der einzelnen Rohdatenwörter der Frontend-Module darstellt, um eine schnelle Fehlersuche in der Hardware möglich zu machen.
Für alle Subdetektoren wurden in der ROOT-Umgebung Kontrollhistogramme entwickelt, die während der einzelnen Messperioden regelmäßig gefüllt und überwacht wurden, um eine gleichbleibende Qualität der Experimentdaten zu gewährleisten (siehe Abbildung 7.38).

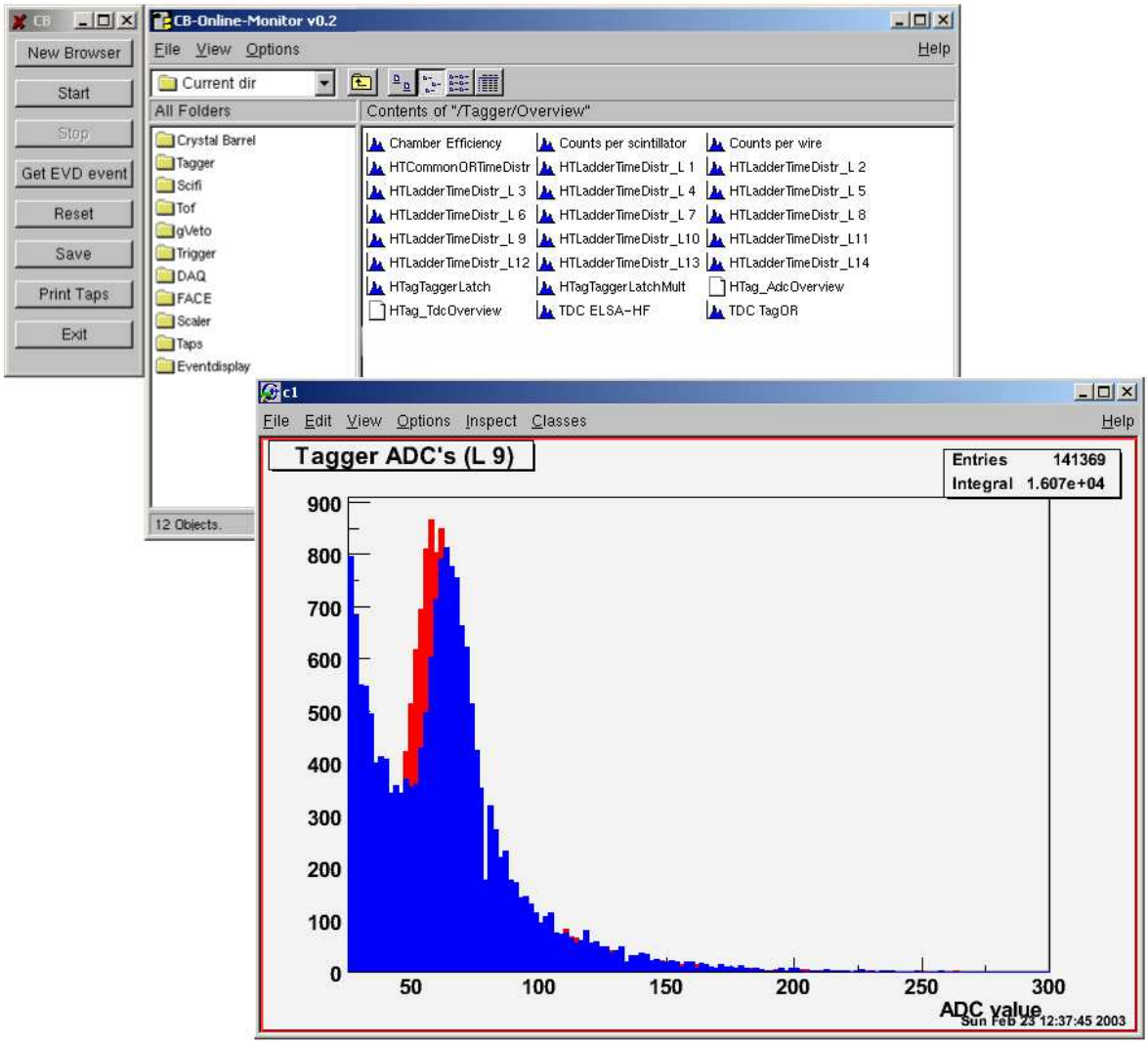


Abbildung 7.38: Kontrollprogramm zur Histogrammierung von Rohdaten auf Basis der ROOT-Entwicklungsumgebung. Gezeigt ist ein typisches ADC-Spektrum des Szintillatorzählers Nr. 9, mit dem die Verstärkung und Ratenabhängigkeit dieses Teildetektors kontrolliert werden kann.

Kapitel 8

Leistungsdaten des Datenauslesesystems

Nachdem in den Kapiteln zuvor die Strukturen und die Implementation in den jeweiligen Bereichen diskutiert wurden, steht nun das zeitliche Verhalten der einzelnen Teilsysteme und des Gesamt-Systems im Blickpunkt, das eine direkte Aussage über die Leistungsfähigkeit des Gesamtsystems und damit über die notwendigen Strahlzeitdauern für spezielle Fragestellungen zulässt.

Zu diesem Zwecke wurden Tests zur Bestimmung der Performance der jeweiligen Komponenten durchgeführt. Die Tests bzw. Leistungen teilen sich in zwei Bereiche auf. Das Basissystem beinhaltet den Transport vom Punkt der Auslese bis zur Speicherung der Daten. Hier ist also die Höhe des Datendurchsatzes maßgebend. Der andere Bereich ist die Geschwindigkeit der Auslese der Elektronik-Module, die im wesentlichen durch die einzelnen Hardwarekomponenten bestimmt wird.

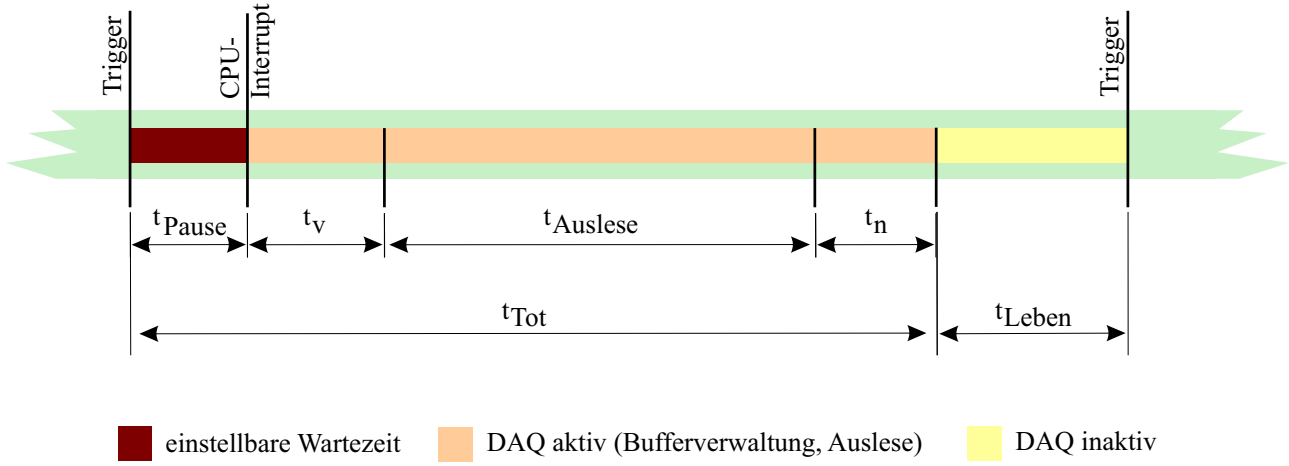


Abbildung 8.1: Zeitspannen zwischen zwei Ereignisse aus Sicht der Datennahme

Die Zeitspanne zwischen zwei Triggerzeitpunkten (Event-Zyklus) während der Datennahme besteht aus zwei Komponenten, der Totzeit t_{Tot} und der Lebendzeit t_{Leben}. Die Totzeit t_{Tot} lässt sich noch in weitere vier Abschnitte unterteilen (Abbildung 8.1):

$$t_{Tot} = t_{Pause} + t_v + t_{Auslese} + t_n$$

t_{Pause} :

Die Totzeit beginnt mit einer einstellbaren Pause, die am Sync-Clientmodul festgelegt werden kann. Diese Pause verzögert die Weitergabe des eingehenden Trigger-Signals an den Interrupt der Prozessorkarte. Da die Daten vor Ende der Digitalisierungsphase nicht zur Verfügung stehen, kann durch die Wahl des Zeitpunktes des Interrupts der Prozessor entlastet werden. Die eingestellten Zeiten bei den Sync-Clientmodulen der einzelnen Prozessoren sind in folgender Tabelle 8.1 zusammengefasst. Sie wurden zuvor mittels eines Oszilloskops am jeweiligen Sync-Clientmodul gemessen.

Name	S/T-Kontroller	Tagging-System	Innen-Detektor	Crystal-Barrel 1,2	ToF	TAPS
Zeit	$24\mu s$	24 μs	180 μs	300 μs	180 μs	28 μs

Tabelle 8.1: Eingestellte Verzögerungszeiten zwischen Trigger und Prozessor-Interrupt

t_v, $t_{Auslese}$, t_n :

Der Interrupt löst im Auslese-Prozess die Vorbereitung für den Aufruf der Auslesefunktionen des lokalen Eventbuilders aus. Diese Zeit t_v besteht aus der Interrupt-Latenzzeit und der Wartezeit auf einen freien Datenbuffer für die Auslesefunktion. Sie sollte auf Grund der hinreichenden Tiefe des Buffersystems auf den lokalen Eventbuildern sehr kurz sein und kaum zur Gesamttotzeit beitragen. Steht ein freier Buffer zur Verfügung, beginnt die Auslese der Daten. Die Restzeit t_n bis zum Ende der Totzeit enthält die Übergabe der Daten an den Transportprozess und die Statusmeldungen an den Sync-/Trigger-Kontroller.

Die Restzeit bis zum Beginn des nächsten Ereignisses wird Lebendzeit genannt. In dieser Spanne wartet der Ausleseprozess auf ein weiteres Event.
Die Zeiten $t_{Auslese}$, t_{Tot} und t_{Lebend} werden vom Sync-Clientmodul separat erfasst und gespeichert (siehe Kapitel 5.2.1). Sie werden über separate Zähler ermittelt, die mit einer 20 MHz-Clock getaktet sind.
In den folgenden Abschnitten werden nun zunächst die Resultate der Tests geschildert, bei denen keine Auslese stattgefunden hat, da ein spezieller lokaler Test-Eventbuilder mit einer Implementierung der Auslesefunktion eingesetzt wurde, der nur einen Datenbuffer variabler

Größe zurückliefert. Somit ist die Zeit $t_{Auslese}$ vernachlässigbar bzw. Null. Weiterhin werden die Zeiten t_v und t_n nur in der Summe erfasst, $t_{Daten} = t_v + t_n$.

$$t_{Tot} = t_{Daten} + t_{Pause}$$

Dies gibt einen Aufschluss über die maximal erreichbare Datenakquisitionsrate, die durch den Datentransport limitiert wird.

8.1 Der lokale Eventbuilder - Basismodul

Die erste Testreihe bestand in der Bestimmung der Zeit t_{Daten} für die verschiedenen Prozessorkarten im Bereich der Frontendelektronik. Getestet wurde mit einem Zufallstrigger, der mit einer Rate von ca. 100 kHz lief. Getestet wurde immer ein Paar aus Sync-/Trigger-Kontroller und jeweils einem Prozessor eines Subdetektors. Das Datenakquisitionssystem lief in einem automatischen Modus, wobei mit einer Ereignisgröße von 10 Byte begonnen wurde und nach 20.000 Ereignisse die Ereignisgröße um 40 Byte erhöht wurde. Um den Einfluss des restlichen Datentransportsystems (Event-Saver, Daten-Server) abzukoppeln, wurden die Daten allerdings nicht im Event-Saver synchronisiert, sondern nur in die Netzwerkschnittstelle des Event-Savers transferiert.

Aus den Testläufen für die verschiedenen Subdetektoren wurden die Totzeit-Verteilungen für ansteigende Buffergrößen ermittelt. Sie sind in Abbildung 8.2 dargestellt. In jeder Verteilung fallen drei Merkmale auf:

1. eine Grundtotzeit, unabhängig von der Ereignisgröße
2. eine Datenbuffergröße, ab der die durchschnittliche Totzeit sich erhöht
3. eine Totzeit, die in gewissen Fällen von der Übertragungsgeschwindigkeit der Netzwerkschnittstelle des lokalen Prozessors abhängt

Zu 1:

Die Grundtotzeit setzt sich zusammen aus:

$$t_{Tot} = t_{Daten} + t_{Pause}$$

Die Zeiten t_{Pause} sind eine feste Größe und können der Tabelle 8.1 entnommen werden. Der Wert t_{Daten} ergibt sich dann aus der Differenz zwischen dem Wert, welcher aus dem Histogramm abgelesen werden kann, und der gemessenen Zeit t_{Pause}.

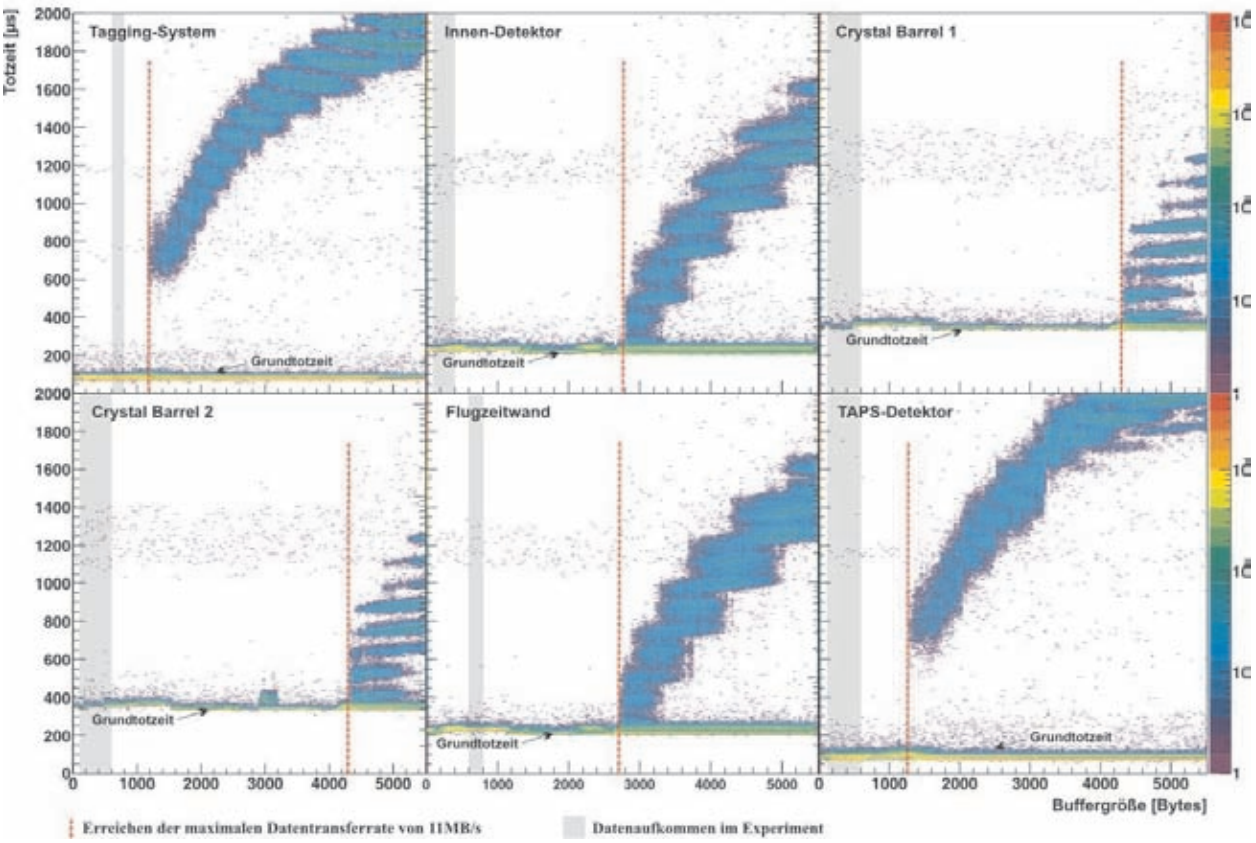


Abbildung 8.2: Abhängigkeit der Totzeit pro CPU von der Größe des erzeugten Datenbuffers

Für gegen Null gehende Buffergrößen erhält man die Zeit des Overheads (Bufferhandling, Interrupt-Latenzzeit, Prozess-Umschaltzeit, ...) die pro Ereignis immer benötigt wird. Folgende Tabelle gibt eine Übersicht über die ermittelten Werte:

Name	Tagging-System	Innen-Detektor	Crystal-Barrel 1,2	ToF	TAPS
t_{Daten}	37 μs	30 μs	30 μs	30 μs	35 μs

Die Ergebnisse liegen, wie zu erwarten, alle in der gleichen Größenordnung. Sie zeigen einen sehr geringen Overhead von Seiten der Grundmodulsoftware, da schon die Prozess-Umschaltzeit beim Linux-Kernel 2.4.x im Bereich von ca. 13 μs [32] liegt.

Eine weitere Eigenschaft der Grundtotzeit ist die Begrenzung des Datenakquisitionssystems auf eine maximale Rate, mit der das System laufen kann. In den Testläufen bestimmte jeweils die Pausezeit t_{Pause} des jeweiligen lokalen Eventbuilders diese maximal mögliche Rate.

Name	Tagging-System	Innen-Detektor	Crystal-Barrel 1,2	ToF	TAPS
max. Rate	10 kHz	4 kHz	3,5 kHz	4 kHz	10 kHz

Zu 2:

Die Größe des Datenbuffers, ab der oberhalb der Grundtotzeit (rote Linie in Abbildung 8.2) weitere Zeit benötigt wird, wird zum einen durch die maximale Datentransferrate der Netzwerkschnittstelle (FastEthernet 100MBit, 11 MB/s) bestimmt und zum anderen durch die Rate, mit der das Datenakquisitionssystem maximal laufen kann.
Die Tests wurde immer mit dem Sync-/Trigger-Kontroller in Verbindung mit einem lokalen Eventbuilder durchgeführt. Hierdurch ergibt sich die maximal mögliche Rate aus der eingestellten Pausezeit für die Konversionszeit des jeweiligen getesteten lokalen Eventbuilders.
Bevor die maximale Datentransferrate erreicht ist, wird die Prozessierung mit nahezu konstanter Rate ausgeführt. Durch die steigende Größe des Datenbuffers ergibt sich eine linear mit der Buffergröße ansteigende Datenrate. Wird die maximale Datenrate der Netzwerkschnittstelle erreicht, kann eine Verlängerung der Totzeit auftreten.
Die Zeit, die ein Transfer eines Datenbuffers maximal benötigen darf, ergibt sich aus der Zeit t_{Pause} und dem Overhead pro Ereignis (t_{Daten} für kleine Datenbuffergrößen).

$$t_{max} = t_{Daten} + t_{Pause}$$

Benutzt man diese Zeit t_{max} und die maximal mögliche Datentransferrate der Netzwerkschnittstelle (100MBit) von ca. 11 MB/s, so ergeben sich die maximalen Datenbuffergrößen, bei der noch kein Ratenabfall auftritt, über folgende Formel:

$$s_{max} \approx t_{max} \cdot 11\ MB/s$$

Für die einzelnen Subdetektoren ergeben sich unterschiedliche kritische Buffergrößen s_{max}, bei der der Ratenabfall beginnt:

Name	Tagging-System	Innen-Detektor	Crystal-Barrel 1	Crystal-Barrel 2	ToF	TAPS
$s_{max}[kB]$	1	2,5	4,1	4,1	2,5	1

Die Zeiten stimmen mit den Einsatzpunkten aus Abbildung 8.2 überein, d.h. der Datentransport wird ausschließlich durch die Datentransferleistung der Netzwerkschnittstelle bestimmt.

Ein weiteres Merkmal ist, dass mit dem Einsetzen der maximalen Datentransferrate die Grundtotzeit weiterhin vorhanden bleibt. Dies ist eine Folge der Bufferverwaltung durch die Übergabe

der Datenblöcke vom Auslese-Prozess zum Datentransport-Prozess. Dieses Verhalten soll noch etwas genauer untersucht werden.

Zu 3:

Bei kleinen Datenbuffergrößen ist der Transportprozess in der Lage, die anfallenden Blöcke schnell genug zu transferieren. Ist die Transportzeit aber größer als die maximale Transferzeit, die ohne einen Ratenabfall möglich ist, wird der Buffer zwischen den Prozessen `ReadoutThread` und `DataThread` langsamer geleert als er gefüllt wird. Da die Bufferverwaltung nur über eine begrenzte Anzahl an Buffern verfügt, kommt es nach einer gewissen Anzahl von Ereignissen zu einer längeren Wartezeit auf einen freien Buffer und somit zu einer erhöhten Totzeit.Diese Auswirkungen sollen im folgenden an den Tests des Prozessors des Tagging-Systems verdeutlicht werden.

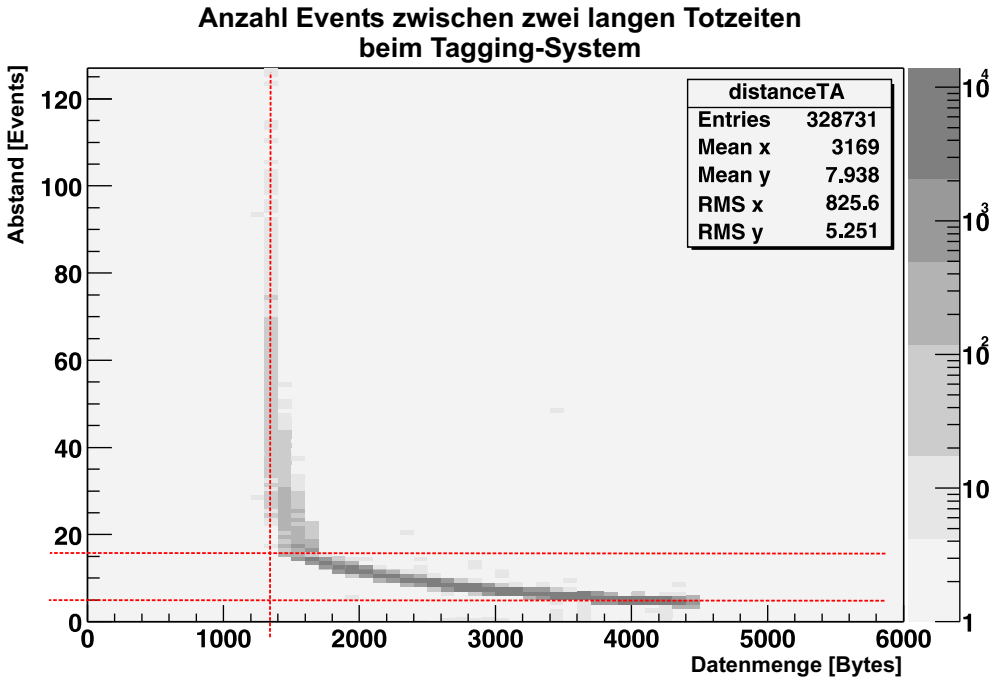


Abbildung 8.3: Abstand zwischen zwei Ereignissen, die zu einer verlängerten Totzeit führen

Die Abbildung 8.3 zeigt für das Tagging-System die Verteilung der Anzahl an Ereignissen zwischen zwei Ereignissen, die zu einer verlängerten Totzeit gegenüber der Grundtotzeit führten, für verschiedene Datenbuffergrößen. Die senkrechte rote Linie gibt die Datenbuffergröße an, bei der die maximale Datentransferrate erreicht ist. Die waagerechte Linie zeigt die Anzahl der Datenbuffer im Buffersystem zwischen dem Auslese- und dem Transport-Prozess.
Bis zum Eintreten der kritischen Datenbuffergröße ist keine erhöhte Totzeit zu finden. Erst nach der Linie bei ca. 1 kB Größe beginnt ein Bereich, in dem erhöhte Totzeiten mit einem

wiederkehrenden Abstand zwischen zwei Ereignissen auftritt. Erst nach ca. 400 Bytes hinter dem Einsatzpunkt beginnt unterhalb 16 Ereignissen ein regelmäßiger und häufigerer Abstand aufzutauchen. Dies bedeutet, dass z.B. bei einer Datenbuffergröße von ca. 3000 Bytes jedes achte Ereignis (entspricht 12,5 % der Gesamtzahl der Events) zu einer erhöhten Totzeit führen.

Die Abbildung 8.4 zeigt dies noch einmal anhand von Projektionen des zwei dimensionalen Histogramms aus Abbildung 8.2. Auch hier ergeben sich ähnliche Verhältnisse zwischen Grundtotzeit und erhöhter Totzeit wie die aus Abbildung 8.3 ermittelt. Der prozentuale Anteil mit erhöhter Totzeit an der Gesamtzahl der Ereignisse ist in den kleinen Histogrammen aufgeführt. Bis zu einem Datenaufkommen von ca. 3 kB pro Ereignis beim Tagging-Systems ist dieser Anteil in einem vertretbaren Rahmen.

Zu größer werdenden Datenbuffern entsteht eine Struktur bei den erhöhten Totzeiten. In diesem Bereich konkurrieren die Prozesse Auslese, Datentransfer und System-I/O um die vorhandene Prozessorzeit, zusätzlich beeinflusst vom Scheduler des Linux-Betriebssystems. Dies ist eine Betriebssystemfunktion, die die vorhandene Prozessorzeit in Zeitscheiben an die potentiellen aktiven Prozesse und an den Kernelprozess selbst verteilt. Eine theoretische Vorhersage ist bei diesem gekoppelten System schwierig, da die Art der Implementation des Schedulers einen großen Einfluss auf dieses Zeitverhalten hat.

Die Verlängerungen in der Totzeit zeigen sich auch in Abbildung 8.5, in der nun der resultierende Ratenverlauf der einzelnen lokalen Eventbuilder dargestellt ist. Die rote Linie gibt den Verlauf bei einer maximalen Datenrate der Netzwerkschnittstelle von 11 MB/s an. Hier zeigt sich, dass die gemessene Ereignisrate dem Kurvenverlauf sehr schön folgt. Somit wird die Ereignisrate beim Erreichen der maximal möglichen Datentransferrate allein durch die Netzwerkschnittstelle bestimmt.

Vor dem Erreichen der maximalen Datentransferrate (Übergang zwischen dem flachen und dem abfallenden Bereich) sind kleine Stufen erkennbar. Dieser Unterschied in der Ereignisrate entspricht einer Schwankung der Totzeit von ca. 10 μs. Da der Umschaltvorgang zwischen zwei Prozessen in ähnlicher Größenordnung liegt, könnte dies vom dynamischen Wechsel zwischen den konkurierenden Prozessen (`ReadoutThread` $\leftrightarrow$ `DataThread`) stammen, der durch den Scheduler[1] im Linux-Betriebssystem verursacht werden kann.

Zusammenfassend zeigt sich, dass aufgrund der eingestellten Pausenzeiten am Sync-Clientmodul (bedingt durch die Digitalisierungszeit der Frontendmodule) und der Transferleistung der Netzwerkschnittstelle der VMIC-Prozessorkarten keine zusätzlichen Totzeiten durch das entstehende Datenvolumen an den Subdetekoren erzeugt werden.

[1] Hierbei handelt es sich um eine Verwaltungsfunktion, die entscheidet wann welcher Prozess den Prozessor ausgeführt wird

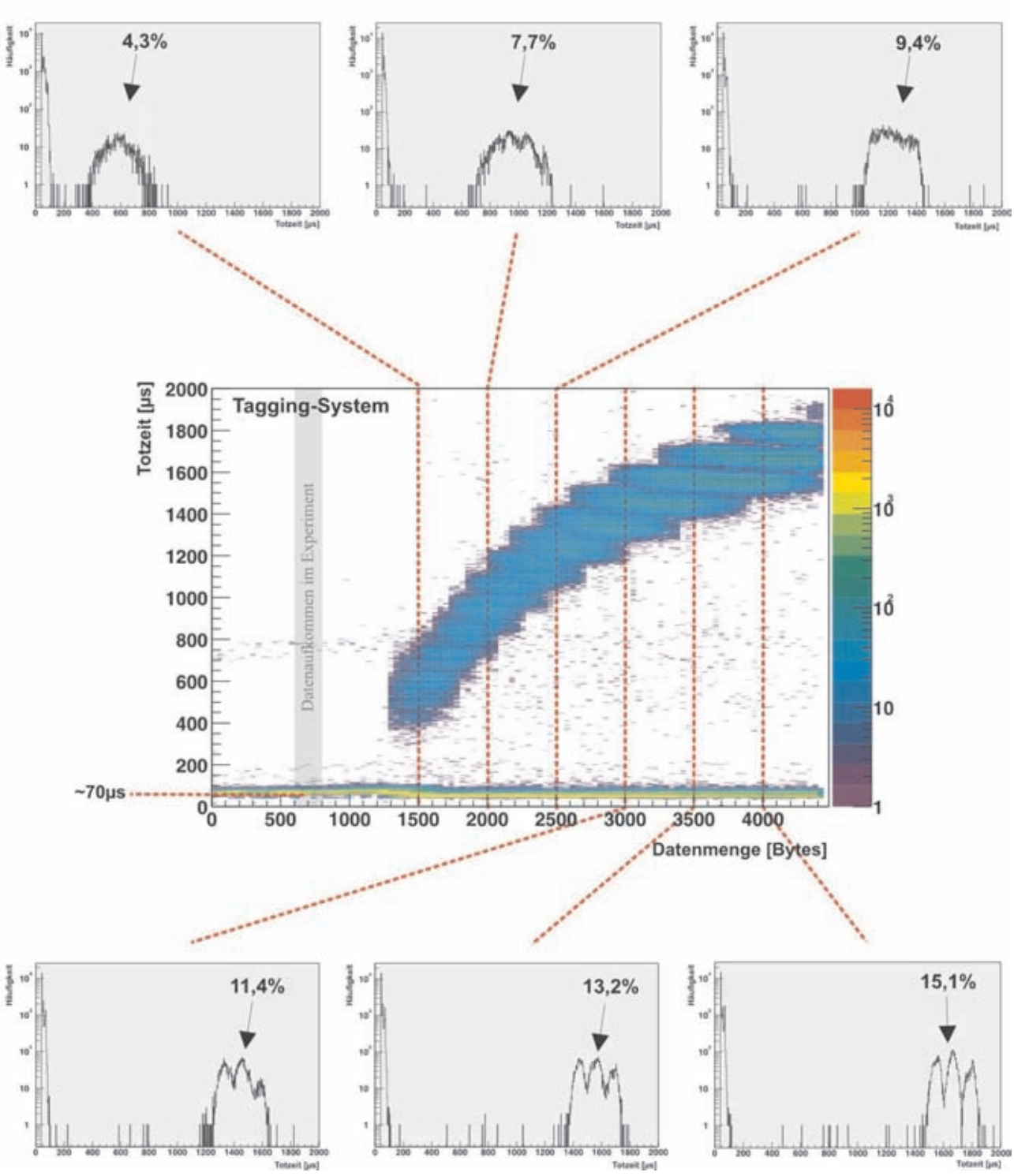


Abbildung 8.4: Abhängigkeit der Totzeit beim Tagging-System von der Größe der erzeugten Datenbuffer (Oberhalb und Unterhalb sind die Projektionen auf die Y-Achse zu verschiedenen Buffergrößen gezeigt. Der graue Bereich repräsentiert das Datenaufkommen im Experiment).

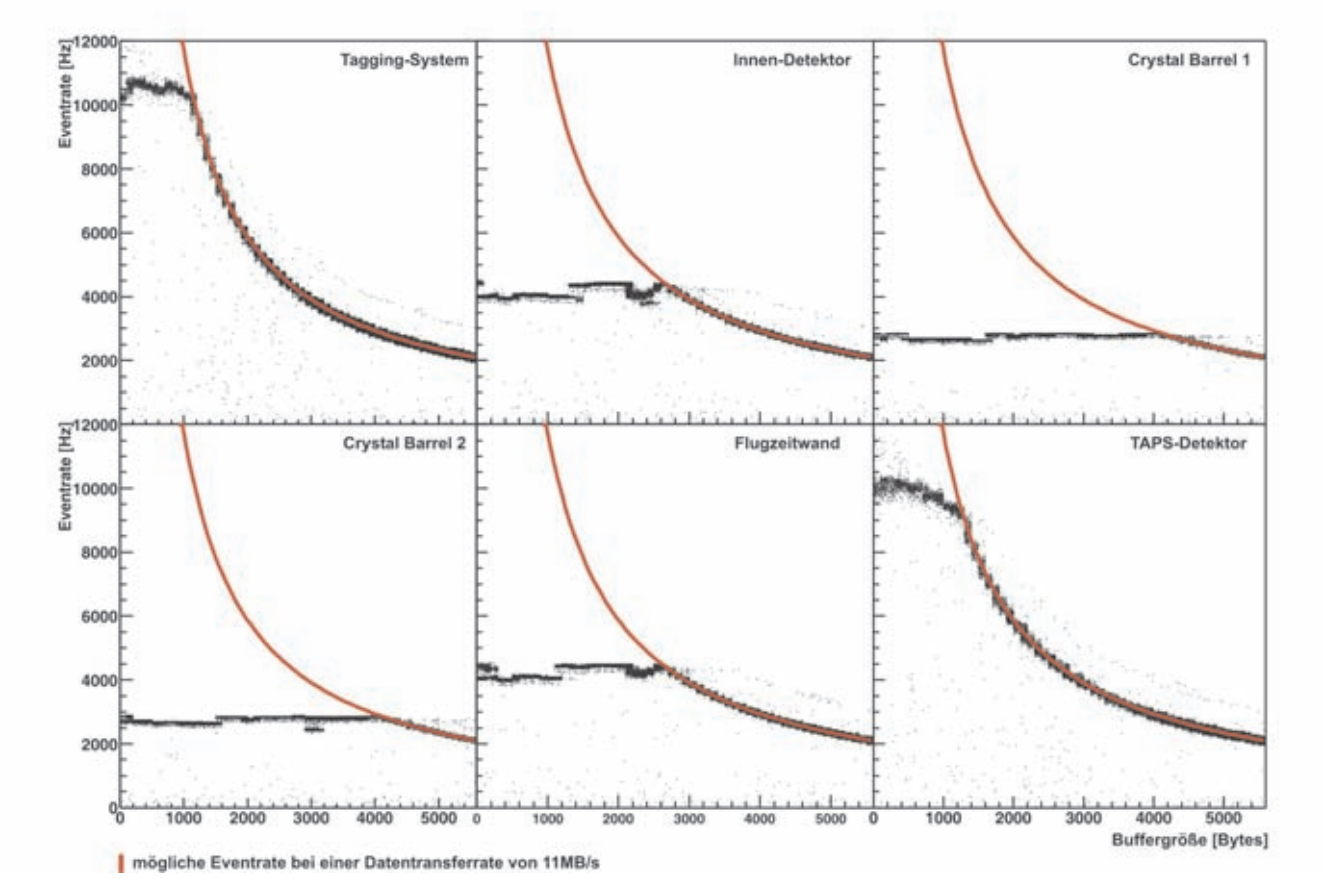


Abbildung 8.5: Resultierende Eventraten auf Grund der Totzeit in Abhängigkeit von der gesendeten Datenmenge

Subdetektor	Tagging-System	Innen-Detektor	Crystal-Barrel 1,2	ToF	TAPS
ø Datenaufkommen [kB]	0,56	0,1	0,23	0,76	0,23
krit. Datengröße [kB]	1	2,5	4,1	2,5	1

Tabelle 8.2: Gegenüberstellung zwischen durchschnittlichem Datenaufkommen und maximaler totzeitfreier Datenbuffergröße

Tabelle 8.2 gibt einen Überblick über das durchschnittliche Datenaufkommen bei den jeweiligen Subdetektoren. In allen Fällen liegt das Aufkommen um mindestens einen Faktor 2 unterhalb der kritischen Datengröße.

In diesen Tests wurde ausschließlich das Zeitverhalten des Grundmoduls vom lokalen Eventbuilder (siehe Kapitel 7.2) betrachtet. Im folgenden Abschnitt soll nun das Verhalten der Datentransferrate, bezogen auf das Basissystem aus den lokalen Eventbuildern und dem Event-Saver, näher erläutert werden.

8.2 Das Datentransportsystem

Das Basistransportsystem ist verantwortlich für den Datendurchsatz von der Frontend-Elektronik bis hin zum Datenspeicher. Abbildung 8.6 zeigt die einzelnen Komponenten, die an diesem Prozess beteiligt sind. Wie bei den Tests im Abschnitt zuvor wird der gleiche lokale Eventbuilder verwendet, der nur einen Datenbuffer gewisser Größe liefert. Hierdurch lässt sich die Leistungsfähigkeit, bezogen auf den Datentransfer des Basissystems, beurteilen.

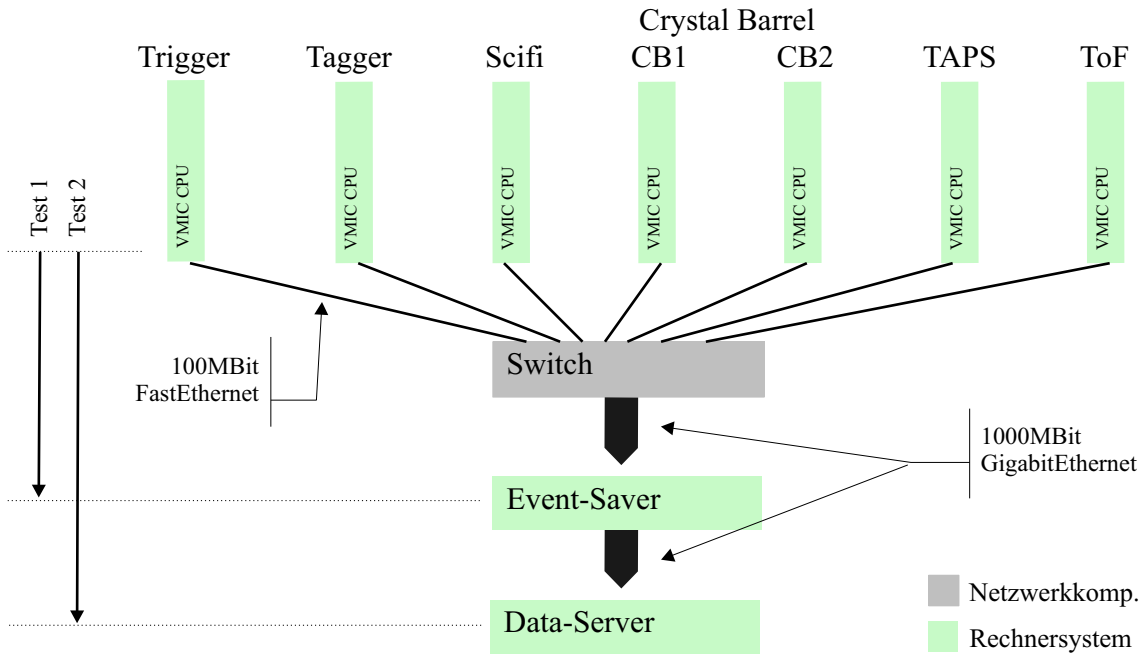


Abbildung 8.6: Datenfluss von der Frontendelektronik bis zum Daten-Server

Die Testumgebung aus dem Abschnitt zuvor wird um eine Weiterverarbeitung im Event-Saver und um die Anzahl der lokalen Eventbuilder erweitert. Vereinfacht gesehen läuft der Prozess folgendermaßen ab:
Ein Datenblock wird auf dem Prozessor des Subdetektors erzeugt und danach über eine 100MBit-Netzwerkverbindung an den Event-Saver weitergeleitet. Es werden jedoch nicht nur von einem Prozessor Daten gesendet, sondern von mehreren parallel. Die einzelnen Prozessoren sind physikalisch nicht direkt mit dem Event-Saver verbunden. Sie werden an einem Layer 2 Switch gebündelt, welcher über eine 1GigaBit-Ethernet Netzwerkverbindung mit dem Event-Saver gekoppelt ist. Der Event-Saver ist wiederum über eine weitere 1GigaBit-Ethernet-Leitung mit den Daten-Servern gekoppelt, der die ausreichende Festplattenkapazität für die Speicherung zur Verfügung stellt.

Die Untersuchungen teilen sich in zwei Tests auf. In Abbildung 8.6 sind die Tests zur Be-

stimmung der Datentransferraten durch zwei Pfeile gekennzeichnet. Im folgenden werden diese Pfeile erläutert. Sie erlauben einen Schluss über die möglichen Datenraten von den Prozessoren bis zum Daten-Server:

Test 1 Es werden die Daten bis zum Event-Saver transportiert, aber nicht an den Daten-Server weitergeleitet.

Test 2 Es handelt sich um die komplette Kette von den Frontendprozessoren bis auf den Datenträger, inklusive der Wandlung in das Datenformat ZEBRA.

Zur Vereinfachung wird die Ereignisrate bestimmt, indem die Ereignisse gezählt werden, die innerhalb von 25 ms auftreten. Die Datenrate ergibt sich dann durch Mittelung der Ereignisgröße über die gezählten Ereignisse, geteilt durch 25 ms.

Test 1

Bei diesem Test wird der parallele Datenstrom von den einzelnen lokalen Eventbuildern durch den Layer-2 Switch in eine Datenleitung hinein gemultiplext, d.h. die Datenströme werden hierdurch serialisiert. Diese quasi "parallelen" Daten werden dann im Event-Saver zu einem Ereignis zusammengefasst und verworfen. Betrachtet man die Verbindung auf der TCP/IP-Softwareschicht, handelt es sich um mehrere parallele Punkt-zu-Punkt Verbindungen. Der Switch ist hier nicht sichtbar, könnte aber die Datentransferraten negativ beeinflussen.

Die Abbildung 8.7 zeigt die Verläufe der Transferraten für 1 bis 4 lokale Eventbuilder. Bei diesem Test wurde der gleiche Test-Eventbuilder wie zuvor benutzt, die Größe der gesendeten Datenbuffer wurde bei allen Eventbuildern gleichzeitig erhöht. Das Datenakquisitionssystem wurde bei den jeweiligen Konstellationen aus 1,2,3 und 4 lokalen Eventbuildern mit einer gleichen Ereignisrate von ca. 4 kHz betrieben. Die rote durchgezogene Linie in den Diagrammen kennzeichnet die maximal möglichen Datentansferraten eingezeichnet, wenn alle lokalen Eventbuilder mit der ihnen maximalen Datenrate von 11 MB/s übertragen. Die gepunktete Line zeigt den linearen Anstieg der Datenrate, bezogen auf die Gesamtdatengröße (Summe über alle beteiligten lokalen Eventbuilder).
In jedem Test wird jeweils die maximale Datentransferrate erreicht. Es ergibt sich somit, dass das System einen parallelen Datenstrom von vier mal 11 MB/s pro lokalem Eventbuilder bis zur Erzeugung der Eventinformation ohne Probleme verarbeiten kann. Eine Überprüfung der Datenrate über eine Punkt-zu-Punkt Gigabit-Verbindung zwischen zwei PCs ergibt eine maximal mögliche Transferrate von 58 MB/s. Der aus den Spezifikationen ersichtliche Wert wird nicht erreicht. Die vorliegende Begrenzung erfolgt durch die beteiligten PCI-Busse (32/64Bit) und die zur Verfügung stehende jeweilige Prozessorleistung.

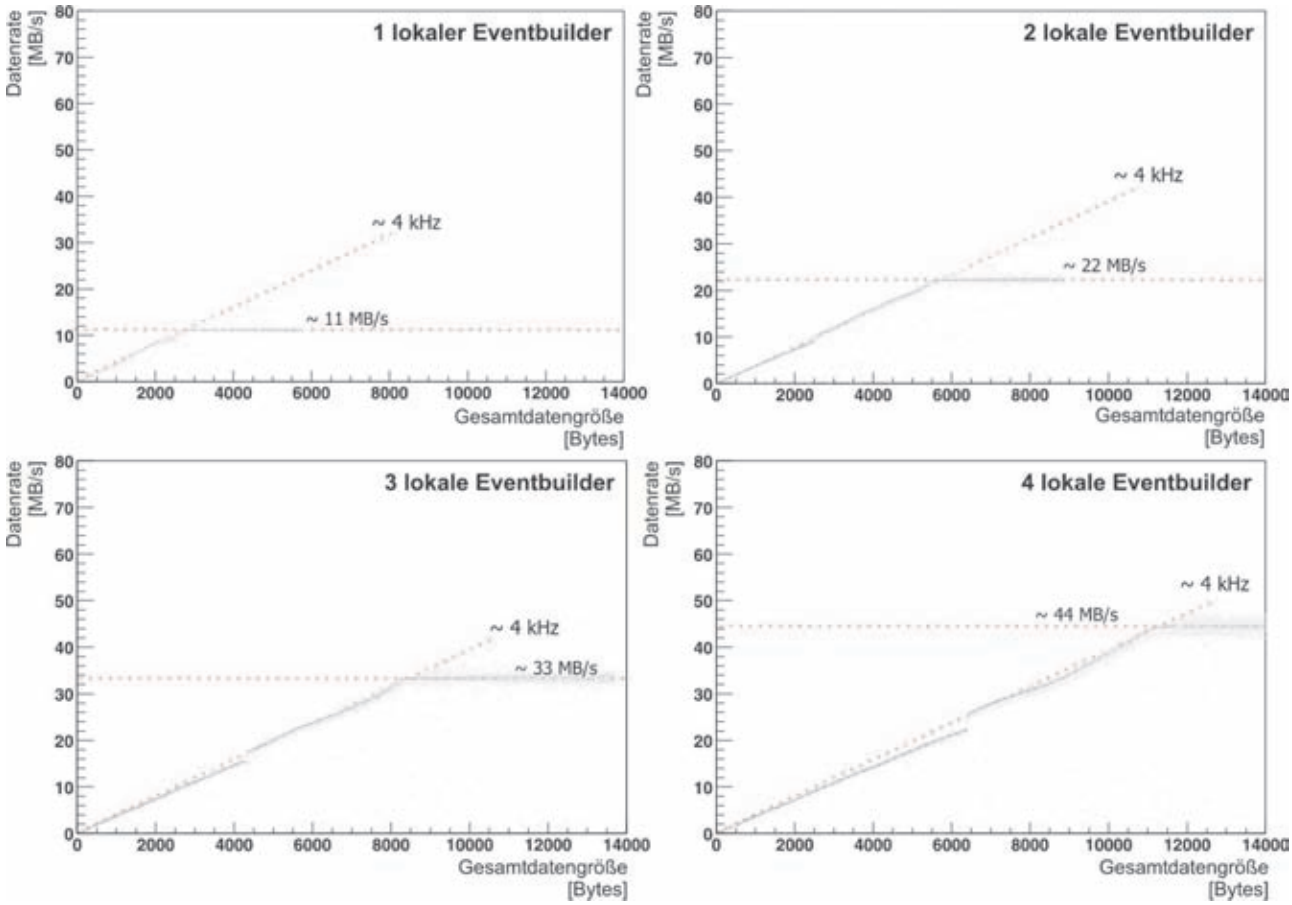


Abbildung 8.7: Datenrate an der Netzwerkschnittstelle des Event-Savers in Abhängigkeit von der Anzahl der lokalen Eventbuilder

Das System aus der Datenübertragung (inkl. "parallelisieren" durch den Switch) und der Zusammenfassung in ein Ereignis erlaubt eine Datentransferrate von 44 MB/s. Wird im Experiment eine Ereignisgröße von durchschnittlich 2 kB erwartet und unter der Annahme, dass das Datenakquisitionssystem (alle Subdetektoren wie im Experiment-Setup) mit maximaler Ereignisrate von 1,4 kHz läuft, so ergibt sich eine Gesamtdatenrate von 2,7 MB/s, wobei die Ereignisrate durch den TAPS-Detektor begrenzt ist. Durch die Konversionszeit der ADC-Module beim Crystal-Barrel ist eine maximale Rate gegeben, die nicht überschritten werden kann. Nimmt man diese theoretische Auslesezeit an, ergibt sich eine maximale Datenrate von 4,3 MB/s. Somit ist das Datentransportsystem sehr gut dimensioniert und stellt keine Begrenzung dar.

Test 2

Im Test zuvor sind die Daten auf dem Event-Saver nicht weiterverarbeitet worden, d.h. es erfolgte kein Transfer zum Daten-Server. In Test 2 ist diese Option nun aktiviert worden. Zur Verdeutlichung des Einflusses der Speicherung der Daten auf ein Medium wurden drei

verschiedene Varianten von Speicher- und Übertragungsmedien getestet.

1. Transfer über eine 1000MBit Verbindung (GigabitEthernet) auf den Daten-Server
2. Speicherung auf einer lokalen Festplatte
3. Transfer über eine 100MBit Verbindung (FastEthernet) auf den Daten-Server

Zuvor soll für den 1. Fall eine Abschätzung der zu erwartenden Datenrate gegeben werden. Die Datentransferrate der Gigabit-Verbindung (1000MBit) zwischen den beiden beteiligten Rechnern wurde auf $D_{Netz} = 58$ MB/s bestimmt. Auf dem Daten-Server ergibt sich eine maximale Datentransferrate ins Festplatten-System von ca. D_{HDD}=42 MB/s[2].

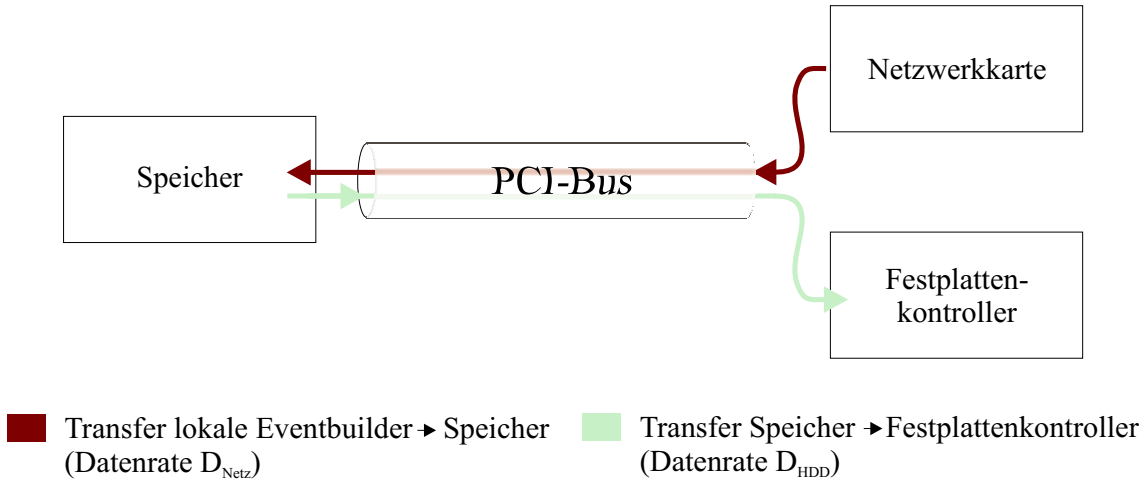


Abbildung 8.8: Transfer der Daten von der Netzwerkkarte zum Festplattenkontroller

In Abbildung 8.8 ist die Situation verdeutlicht. Der Transfer erfolgt in zwei Schritten. Zuerst werden die Daten mit einer Datenrate D_{Netz} in den Speicher übertragen, anschliessend erfolgt der Transfer zum Festplattenkontroller mit einer Rate von D_{HDD}. Aufgrund dieses sequentiellen Ablaufs ergibt sich, dass die beiden Zeiten pro Transfer addiert werden müssen. Für die resultierende Datenrate D_{tot} ergibt sich also:

$$\frac{1}{D_{tot}} = \frac{1}{D_{Netz}} + \frac{1}{D_{HDD}}$$

$$D_{tot} = \frac{D_{Netz} + D_{HDD}}{D_{Netz} \cdot D_{HDD}} = 24,36\ MB/s$$

[2]Ermittelt mit dem Befehl `hdparm -t`

Ein separater Test für die Konversion ins ZEBRA-Datenformat zeigte eine Datenkonversionsrate von nahezu 200 MB/s, sodass diese ZEBRA-Formatierung zunächst vernachlässigt werden kann.

Test 2 wurde durchgeführt wie Test 1, jedoch mit aktivierter Datenspeicherungsoption, vier lokalen Eventbuildern und einer Eergenisrate von ca. 4 kHz. Die gesendeten Daten wurden schrittweise nach 1000 Ereignisse um 40 Bytes erhöht. Diese Art der Erhöhung der Datenmenge durch die ZEBRA-Konversion erzeugt aber einen Effekt bei großen Datenmengen pro Event. Durch das Speichern in das ZEBRA-Format entsteht ein Effekt, der bei der Interpretation der entstehenden Verteilungen zu berücksichtigen ist. Betrachtet man das Verhältnis der Datenmenge vor der Konversion zur Größe der erzeugten Datendatei nach der Konversion ins ZEBRA-Format, ergibt sich eine Zunahme der Daten, wie in Abbildung 8.9 gezeigt. Dieser Faktor kann die Datentransferrate ab einer gewissen Ereignisgröße (Datenmenge) nahezu verdoppeln. Durch das ZEBRA-Format kommen zusätzlich zu den Eventdaten auch Steuer- und Markeninformationen hinzu, die zusätzlich mit den Daten auf Festplatte gespeichert werden müssen und somit eine Erhöhung verursachen.

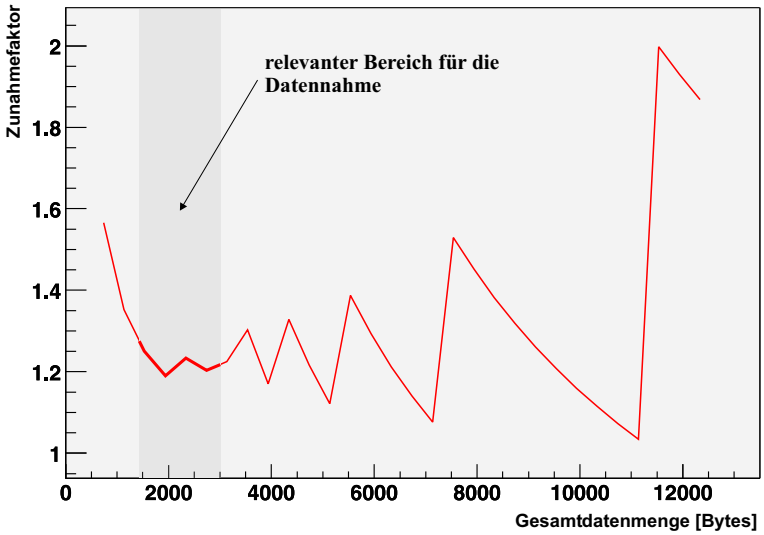


Abbildung 8.9: Zunahmefaktor $fak_{ZEBRA}(s)$ in Abhängigkeit von der Ereignisgröße bei Konversion in das ZEBRA-Datenformat

Unterhalb von einer 4 kB-Ereignisgröße ergibt sich eine Zunahme von ca. 25 % zur ursprünglichen Eventgröße. Für sehr kleine Ereignisse (<1 kB) steigt die Zunahme auf ca. 50 % an. Oberhalb von 4 kB entsteht ein Zackenmuster, das durch starke Schwankungen in der resultierenden Datentransferrate verursacht ist, die nicht vernachlässigbar sind. Verursacht durch diese relativ großen Ereignisse passen auf Grund der ZEBRA-Blocklänge von 23040 Bytes immer weniger Ereignisse in einen Block; es entstehen immer mehr ungenutzte Bereiche durch den

Überhang von Ereignissen über die Blocklänge hinaus. Der Restbereich nach dem Überhang wird mit Null-Informationen bis zur nächsten Blockgrenze aufgefüllt. Diese Extremsituation tritt im Experiment aber nicht auf.

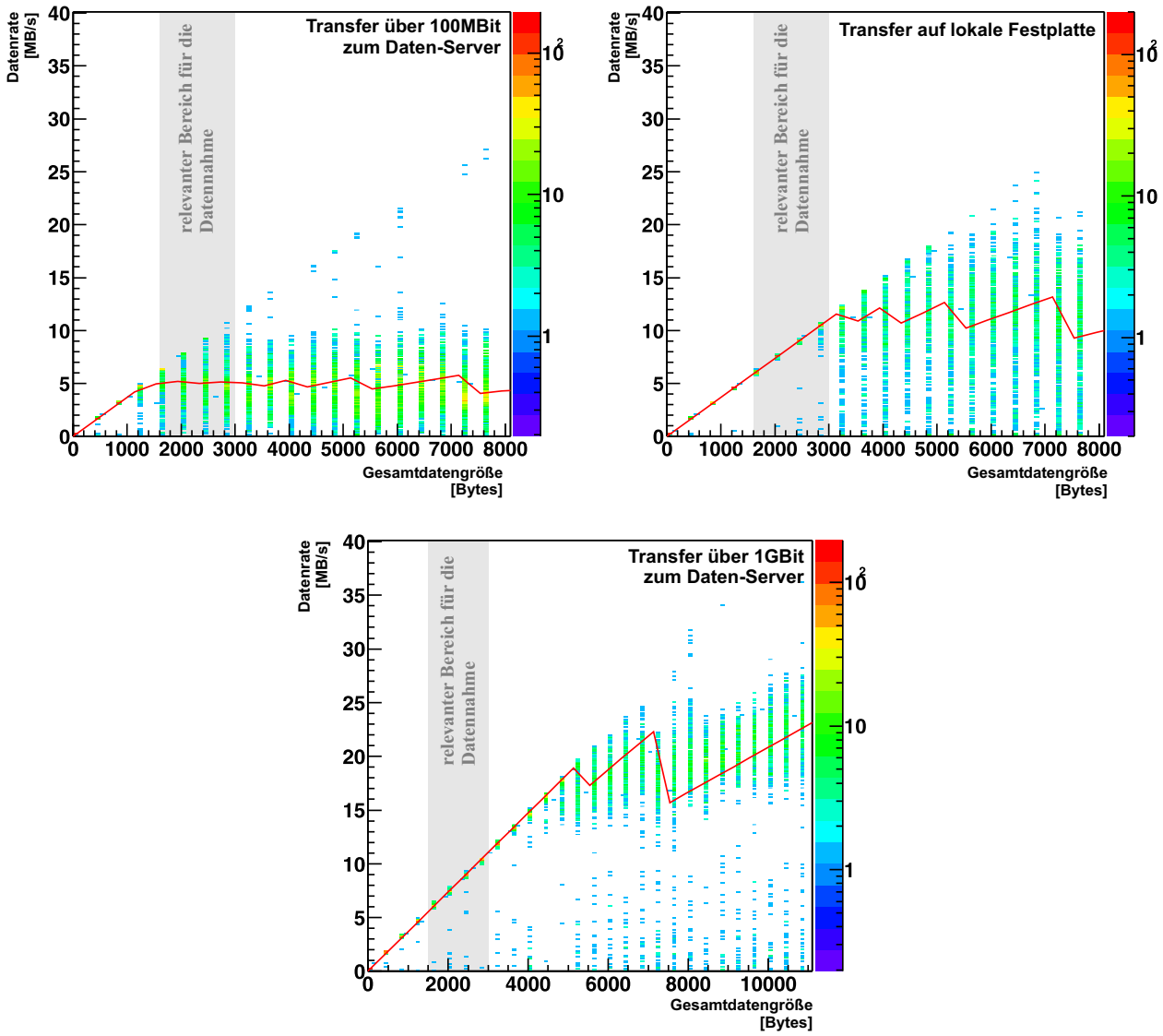


Abbildung 8.10: Datenrate für verschiedene Transfermethoden zum Daten-Server und auf eine Festplatte

In Abbildung 8.10 sind die Resultate der Testläufe für die verschiedenen Übertragungsmedien auf den Daten-Server und auf eine lokal angeschlossene Festplatte dargestellt. Die eingezeichnete rote Linie soll den theoretischen Verlauf der Datentransferrate darstellen. Dieser Verlauf beinhaltet den Zunahmefaktor durch das ZEBRA-Format und den Einfluss der verschiedenen maximalen Datentransferraten.

Der resultierende unstetige Verlauf der gesamten Datentransferrate $d_g(s)$, welche den Verlauf repräsentieren soll, ist durch zwei Stufen mit einer Begrenzung auf die jeweilige maximale Datentransferrate aufgebaut.
Die lokalen Eventbuilder transportieren die Datenmenge s mit einer konstanten Ereignisrate von 3.900 Hz über das Netzwerk zum Event-Saver (d_{Netz}). Diese Eingangsdatenrate $D_{Netz}(s)$ für den Transfer von den lokalen Eventbuildern zum Event-Saver wird mit einer maximal möglichen Rate von D_{Netz} =44 MB/s begrenzt:

$$d_{Netz}(s) = s \cdot 3.900\ Hz$$

$$D_{Netz}(s) = \begin{cases} D_{Netz} & : \quad d_{Netz} \geq D_{Netz} \\ d_{Netz}(s) & : \quad d_{Netz}(s) < D_{Netz} \end{cases}$$

Auf dem Event-Saver wird die einkommende Datenrate D_{Netz} auf Grund der Konversion ins ZEBRA-Datenformat durch den Faktor $fak_{ZEBRA}(s)$ erhöht. Diese erhöhte Datenrate $d_g(s)$ kann nun noch durch die maximale Transferrate auf das Speichermedium D_{HDD} begrenzt werden :

$$d_g(s) = fak_{ZEBRA}(s) \cdot D_{Netz}(s)$$

$$D_g(s) = \begin{cases} D_{HDD} & : \quad d_g(s) \geq D_{HDD} \\ d_g(s) & : \quad d_g(s) < D_{HDD} \end{cases}$$

Der Wert $D_g(s)$ gibt die Datenrate am Ende des Transfers auf das Speichermedium an. Die gemessenen Zeiten, bzw. die Datentransferraten $D'_g(s)$ in Abbildung 8.10 beziehen sich jedoch auf die Datenrate in Abbhängigkeit der Eventgröße, d.h. gemessen wurde der Einfluss einer Datenratenbeschränkung durch das Speichermedium und die ZEBRA-Konversion auf die eingehende Datenrate der lokalen Eventbuilder. Die Zunahme durch die ZEBRA-Konversion muss also wieder herausgerechnet werden:

$$D'_g(s) = \frac{D_g(s)}{fak_{ZEBRA}(s)}$$

Diese Funktion wurde an die jeweiligen Verteilungen angepasst, indem die maximale Datentransferrate D_{HDD} variiert wurde. Die resultierenden Funktionen für die jeweilige Speicherart sind in den Verteilungen mit einer roten Linie gekennzeichnet.
Für die Übertragung über eine 1Gigabit-Ethernet-Verbindung (Fall 1), die zwischen Event-Saver und den Daten-Servern eingesetzt wird, ergibt sich eine Datentransferrate von ca. 24 MB/s. Die Verteilungen in Abbildung 8.10 zeigen auch den relevanten Datentransferratenbereich für das Experiment. Bei einer Eereignisrate von ca. 4 kHz ist die Dimensionierung über eine 1Gigabit-Verbindung zu den Daten-Servern sehr gut gewählt, sodass ein Spielraum zu höheren

Eventgrößen offen bleibt, während beim Transfer auf Festplatte der Bereich sich kurz vor der Begrenzung befindet. Die anderen beiden Varianten sind nicht zu empfehlen, sogar im Falle der 100MBit-Verbindung würde sich die Datentransferrate deutlich auf die Ereignisrate auswirken. Bei einer Ereignisrate von 2 kHz tritt eine Auswirkung auf die Ereignisrate durch die Speicherung auf die Daten-Server sogar erst um einen Faktor zwei später ein (ab Eventgrößen von ca. 8 kB).

Mit den zu erwartenden Eventgrößen und Eventraten ist das Datentransportsystem sehr gut dimensioniert und ermöglicht somit, dass bei Änderungen in der Frontend-Auslese (Zeitgewinn) eine direkte Erhöhung der Eventraten erfolgt!

8.3 Zeitverhalten in der Datennahme

Im vorherigen Abschnitt wurde die Datenakquisition unter Testbedingungen und ohne reale Auslese der Frontend-Elektronik betrachtet. Im nun folgenden Abschnitt wird ein repräsentativer Run aus einer der Strahlzeiten ausgewählt und es wurden die Auslesezeiten ermittelt.
Die zu erwartenden Auslesezeiten aus Kapitel 6 und die gemessenen Zeiten sind in folgender Tabelle gegenübergestellt:

Subdetektor	theo. Auslesezeit	reale Auslesezeit
Trigger-/Sync-Kontroller	$\sim 380\ \mu s$	$\sim 440\ \mu s$
Tagging-System	$\sim 268\ \mu s$	$\sim 440\ \mu s$
Innen-Detektor	$\sim 217{,}9\ \mu s$	$\sim 350\ \mu s$
Crystal-Barrel 1	$\sim 153\ \mu s$	$\sim 150\ \mu s$
Crystal-Barrel 2	$\sim 153\ \mu s$	$\sim 150\ \mu s$
Flugzeitwand	$\sim 64\ \mu s$	$\sim 120\ \mu s$

Bei der Implementation der Auslese für den Crystal-Barrel-Detektor wird die zu erwartetende Auslesezeit fast exakt erreicht. Der Grund dafür liegt in der Verwendung eines modernen FastBus-Sequenzers, der die Aufgabe der Datenkompaktierung (Pedestalunterdrückung) während des Datentransfers in den lokalen Eventbuilder autonom erledigt. Der Prozessor übernimmt hier nur noch das weitere Buffermanagement. Die anderen Subdetektoren benötigen für diese zusätzliche Aufgaben den lokalen Prozessor. So muss z.B. beim Innen-Detektor die Unterdrückung von leeren TDC-Kanälen durch den Prozessor durchgeführt werden.
Die Beschränkungen für die einzelnen lokalen Eventbuilder sind in den folgenden Abschnitten erläutert.

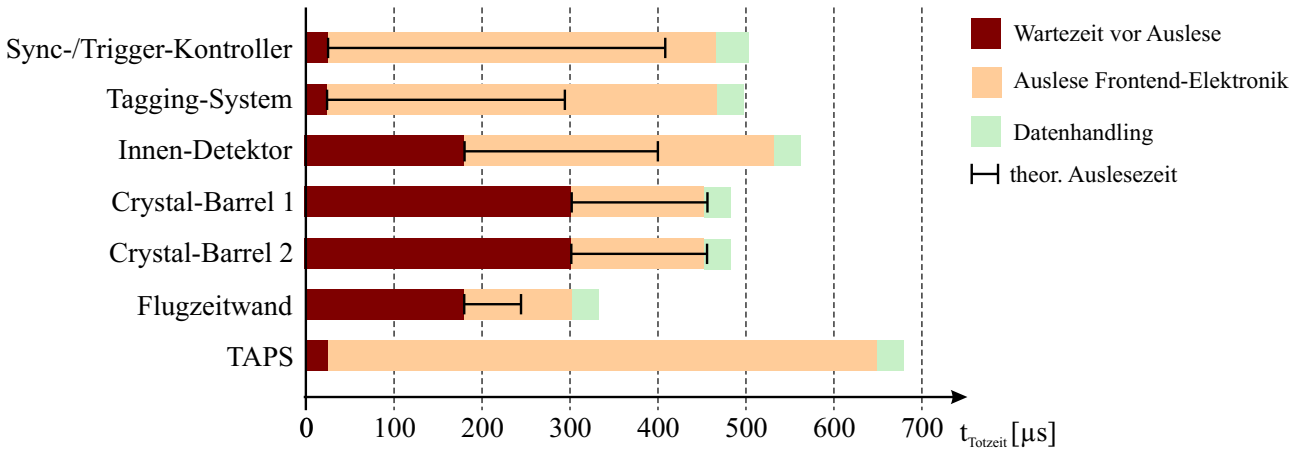
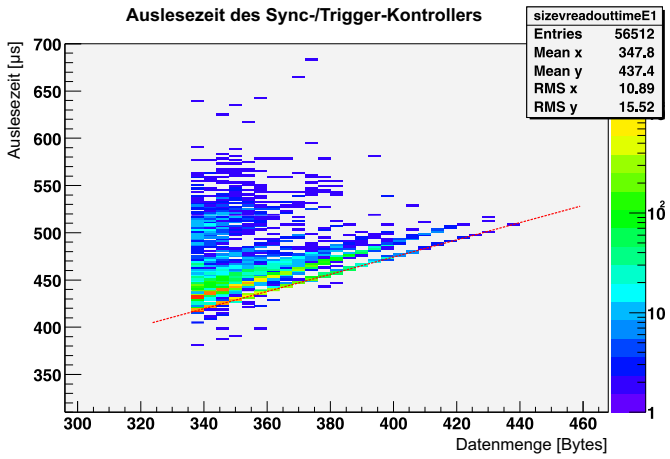


Abbildung 8.11: Aufstellung der benötigten Zeiten pro lokalem Eventbuilder

In Abbildung 8.11 sind die Totzeiten (Wartezeit+Auslesezeit+Bufferhandling) im Vergleich zu

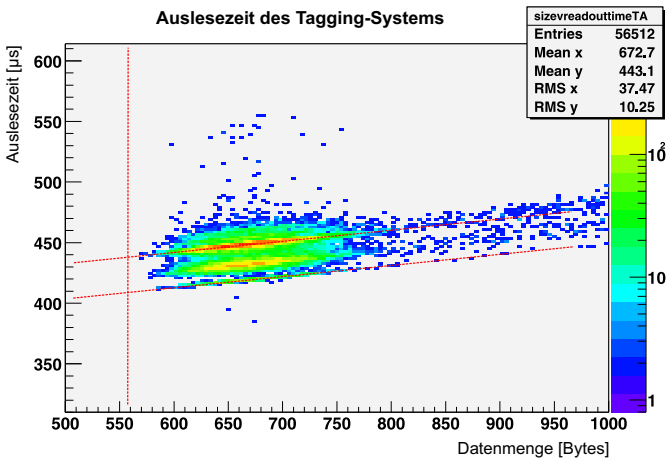
allen lokalen Eventbuildern und der theor. Auslese dargestellt. Nach dieser Übersicht ergibt sich somit eine maximale Datenakquisitionsrate von ca. 1,4 kHz.

8.3.1 Sync-/Trigger-Kontroller



Der Sync-/Trigger-Kontroller führt die Auslese von Triggerinformationen, Zählern, dem Gammaveto-Detektor und der FACE aus. Im Mittel benötigt dieser Prozess $\sim 440\ \mu s$. Alle gelesenen Informationen haben eine feste Länge. Nur die Daten der FACE variieren, da die Zahl der Cluster von Ereignis zu Ereignis schwankt. Deutlich sieht man einen linearen Verlauf zwischen Größe und Auslesezeit (rote Linie). Die Steigung entspricht einer Rate von ca. 1 MB/s und gibt die maximale Zugriffsgeschwindigkeit über den VIC-Bus wieder. Weiterhin sieht man eine Verlängerung der Auslesezeit, die nicht mit der Größe korreliert, d.h. der Ausleseprozess wird durch äußere Einflüsse verzögert. Da auf diesem Prozessor zusätzlich noch Zählerstände für eine Strahldiagnose über den CAMAC-Bus abgefragt werden, ist dies durchaus plausibel.

8.3.2 Das Tagging-System

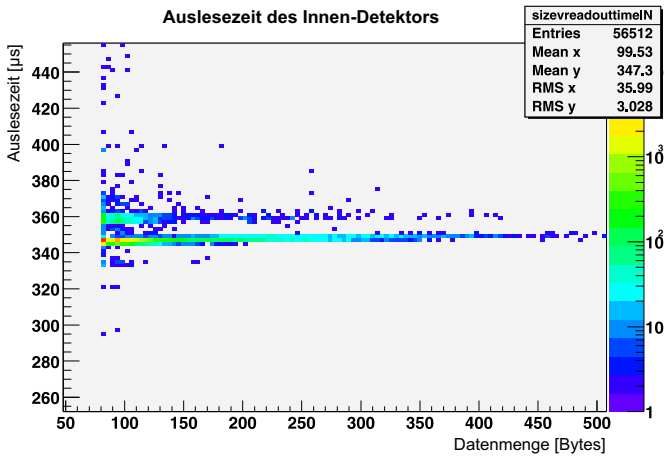


Beim Tagging-System werden die Informationen auf drei unterschiedliche Weisen ausgelesen. Die Szintillationszähler werden über CAMAC, die Proportionalkammer über eine VMEbus-VMEbus Kopplung und die Catch-TDCs über einen VMEbus-Zugriff gelesen.

Dieser Prozess benötigt während der Datennahme $\sim 440\ \mu s$. Die Daten der Catch-TDCs können in der Länge variieren, da es sich hier um einen Multi-Hit-TDC handelt, somit also eine Abhängigkeit von der markierten Photonenrate auftritt. Die Steigung der eingezeichneten Geraden entspricht einer Datentransferleistung von ca. 12 MB/s. Dieses Ergebnis ist im Einklang mit der Spezifikation der VMEbus-Schnittstelle des Catch-Moduls [27]. Es fällt jedoch auf, dass die Zeit um ca. 150 μs über der Schätzung aus dem Abschnitt 6.2 liegt.

Bei der Schätzung ist die Annahme gemacht worden, dass die Daten direkt in den Datenbuffer der DAQ kopiert werden. Dies ist in der Auslese nicht der Fall. Die Daten werden durch den Prozessor kopiert, die einzelnen Statuswörter werden entfernt. Des weiteren wurden bei der Abschätzung die interne Transferzeit von den TDC-Buffern in den durch den VMEbus-Zugriff zugänglichen Spy-Buffer vernachlässigt.

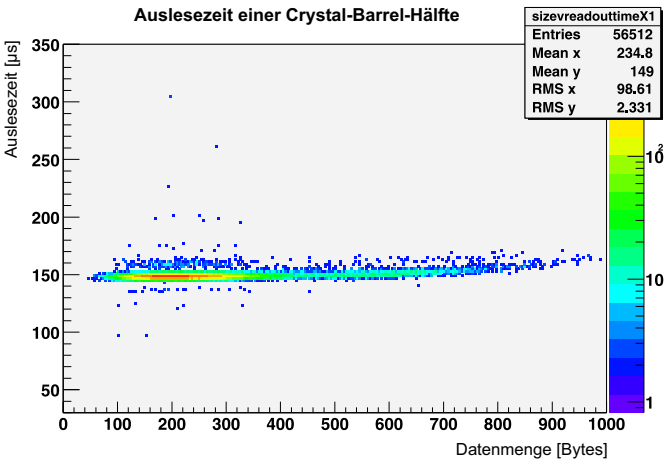
8.3.3 Der Innen-Detektor



Die TDC-Werte des Innen-Detektors werden über FastBus ausgelesen. Zusätzlich werden noch 33 ADC-Kanäle per CAMAC an die Daten angefügt. Der Prozess benötigt während der Datennahme ca. 350 μs. Auffällig ist, dass erstens die Auslesezeit konstant ist, obwohl die Datenlänge sich ändert, und dass zweitens die Zeit zur Abschätzung ca. 130 μs zu lang ist.

Die 512 TDC-Kanäle werden mittels eines DMA-Transfers vom FastBus Sequenzer in den Speicher kopiert. Sie werden aber nicht direkt in den Datenpuffer geschrieben, sondern in einen Zwischenspeicher. Durch einen zusätzlichen Kopiervorgang werden nur die Kanäle aus diesem Bereich in den Datenpuffer transferiert, die einen Treffer hatten. Dieses benötigt die zusätzliche Zeit.

8.3.4 Der Crystal-Barrel-Detektor

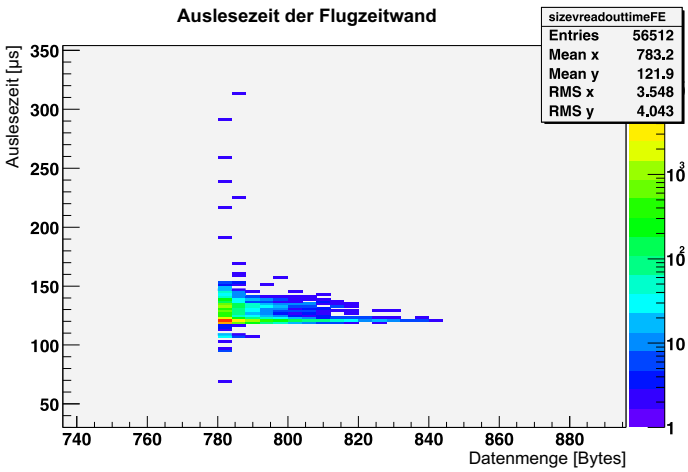


Die ADC-Kanäle des Crystal-Barrel werden über FastBus-ADCs ausgelesen. Die durchschnittliche Auslesezeit aus dem Histogramm ergibt sich zu 149 μs. Gezeigt ist nur die Zeit für die eine Crystal-Barrel-Hälfte. Die andere zeigt das gleiche Zeitverhalten. Die Abschätzung von 153 μs aus Kapitel 6 ist in hervorragender Übereinstimmung mit dem gemessenen Wert.

Dies kann durch die hardwaremäßige Datenunterdrückung erreicht werden, da die Daten nicht noch einmal vom Prozessor bearbeitet werden müssen. Der DMA des Sequenzers

kopiert die ADC-Kanäle direkt in den Datenpuffer, sodass keine zusätzlichen Kopieroperationen notwendig sind.

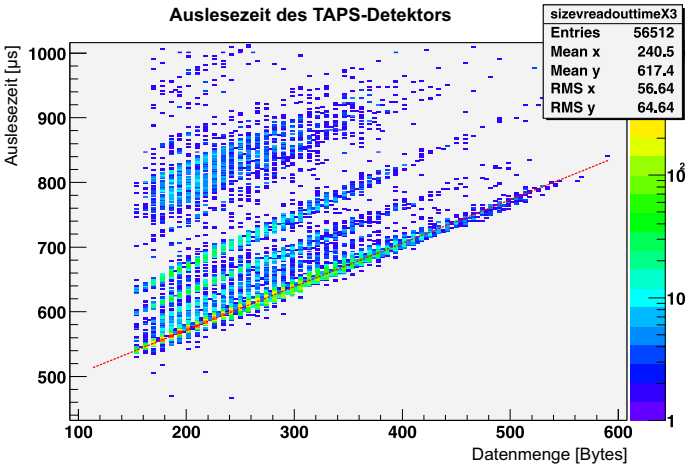
8.3.5 Die Flugzeitwand - ToF



Die Flugzeitwand wird ebenfalls mittels eines FastBus Sequenzers ausgelesen. Die benötigte Zeit der Auslese beträgt ca. 120 μs. Auch hier stimmt der Wert nicht mit der Abschätzung von 64 μs überein.

Die Daten der TDC- und ADC-Module werden nacheinander vom Sequenzer per DMA in einen Zwischenspeicher kopiert und im nächsten Schritt vom Prozessor in den Datenpuffer kopiert, um auch hier eine "Null"- bzw. eine Pedestalunterdrückung, ähnlich wie beim Innen-Detektor, durchführen zu können.

8.3.6 Der TAPS-Detektor



Bei der Auslese des TAPS-Detektors ist eine Kopplung beider Datenakquisitionssysteme vorgenommen worden. Deutlich ist eine große zeitliche Streuung der Auslesezeiten zu erkennen.

Das zeitliche Verhalten der ebenso komplexen Datenakquisition des TAPS-Detektors konnte nicht im Rahmen dieser Arbeit analysiert werden, sodass hier nur die gemessenen Zeiten zur Kenntnis genommen werden konnten. Auf Grund der Benutzung des CAMAC-Standards für die meisten Frontendmodule ergeben sich für kleine Datenmengen etwa die Zeiten, die auch beim Sync-/Tigger-Kontroller gemessen wurden. Die breite Streuung und die starken Bandstrukturen spiegeln das Zeitverhalten verteilter Prozessorsysteme wieder.

8.4 Zusammenfassung

Die in den beiden vorhergehenden Abschnitten durchgeführten Messungen haben gezeigt, dass die realisierte Hard- und Softwarekonfiguration auch im realen Betrieb die theoretisch erwarteten Spezifikationen erreicht und keine nicht verstandenen Faktoren vorhanden sind, die die Leistungsfähigkeit des Gesamtsystems einschränken. Die typische Datenakquisitionsrate von 1 kHz ist im wesentlichen durch die verwendete Auslesehardware bestimmt und nur unter erheblichem finanziellen Aufwand zu verbessern. Trotzdem haben die Untersuchungen bei einzelnen Subdetektoren (z.B. Innen-Detektor) Möglichkeiten aufgezeigt, die Auslese durch den Einbau von Auslese-Sequenzern zu beschleunigen. Die modulare Programmstruktur unterstützt solche Modifikationen in optimaler Weise, da nur wohldefinierte Submodule zu ändern sind.

Kapitel 9

Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Datennahmesystem neu entwickelt und erfolgreich in Betrieb genommen. Die Software des Systems wurde auf Basis von C++ vollständig neu implementiert und die Anzahl der Quellkodezeilen beläuft sich auf 82.494 Zeilen.

Es wurde ein System geschaffen, das nach "oben hin" offen ist, d.h. es wurden Hardwarekomponenten verwendet, die über standardisierte Verfahren (TCP/IP,Linux,C++,...) gekoppelt werden. Somit ist eine Leistungssteigerung des Systems durch Austausch von Hardwarekomponenten möglich. Die Datenakquisitionsrate des aktuellen Systems ist entscheidend durch die Performance des Frontend-Systems (Auslese) bestimmt. Jede Verbesserung der Geschwindigkeit der langsamsten Auslese spiegelt sich wieder in einer Steigerung der effektiven Ereignisrate oder in der Reduktion des Totzeitfaktors.

Die Modularität des Softwarekonzepts eröffnet die Möglichkeit, das bestehende System auf die zukünftigen Bedingungen im Rahmen des angelaufenen Transregio-Projektes mit einem moderaten Programmieraufwand zu modifizieren. Die geplante Verbesserung der Auslese von Sync-/Trigger-Kontroller, Innen-Detektor und Flugzeitwand-Auslese lässt eine Steigerung der Akquisitionsrate um den Faktor zwei erwarten. Die Einbindung neuer Subdetektoren wird durch das Softwarekonzept optimal unterstützt, da für die Basisaufgaben die Hard- und Software bereitsteht, und nur der direkte Ausleseprozess optimiert werden muss.

Eventuelle Engpässe im Netzwerkdurchsatz könnten durch den Übergang vom 100MBit-Ethernet zu 1 Gigabit fast "plug and play"-mäßig realisiert werden kann, da die Hardwareschnittstellen schon vorhanden wären und auch die Software auf den Standard-Netzwerkfunktionen aufsetzt.

Ein möglicher Übergang von dem derzeitigen ZEBRA-Datenformat auf eine neue Datenstruktur, die von dem Geant4-Programmpaket (Monte Carlo Detektor Simulation) direkt unterstützt

wird, ist durch Anpassung einer einzigen Klassenbibliothek mühelos zu vollziehen.

Auch die Einführung eines Online-Datenfilters auf Software-Basis stellt durch das modulare Design kein Problem dar. Somit ist davon auszugehen, dass das bisherige System den Anforderungen der kommenden Experimentphase entspricht und dort, wo es notwendig, leicht angepasst und nachgerüstet werden kann.

Das Datenakquisitionssystem ist seit dem 1.8.2002 in Produktion. Es hat seit diesem Zeitpunkt ca. 6 Mrd. Ereignisse genommen (siehe Abbildung 9.1) und ca. 14 Terabyte Daten zur weiteren Analyse aufgezeichnet. Das alte Datenakquisitionssystem hat im Zeitraum vom 11.04.2000 bis zum 1.8.2002 eine Gesamtzahl von ca. 1 Mrd. Ereignisse auf Band aufgezeichnet. Das neue System hat somit in einem kürzeren Zeitraum einen Faktor sechs mehr Daten geliefert.

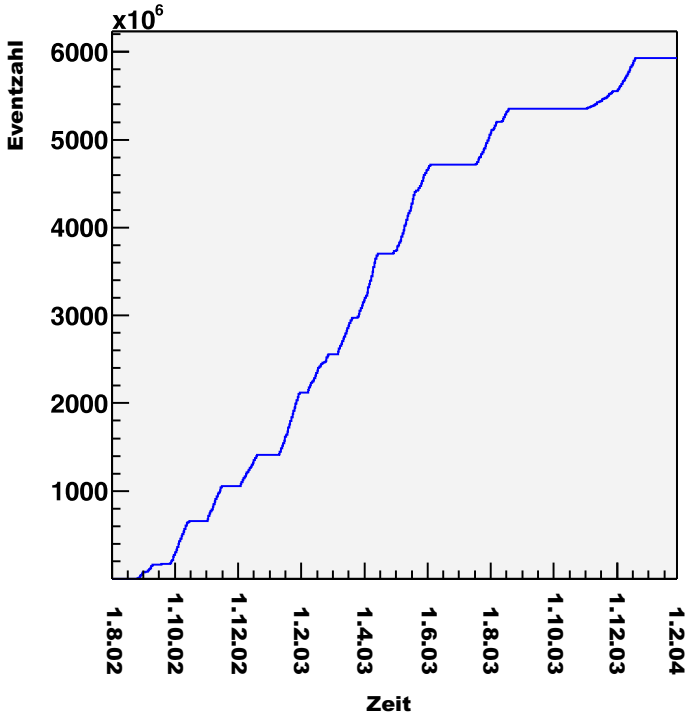


Abbildung 9.1: Verlauf der integrierten Eventzahl seit Beginn der Produktivphase mit dem neuen Datenakquisitionssystem

Anhang A

Datenstrukturen

Auf den folgenden Seiten sind die Strukturen aufgeführt, die in den einzelnen ZERBA-Bänken verwendet werden.

Folgende Banknamen werden pro Rekord ins ZEBRA-Format geschrieben:

Detektor	Banknamen
Sync-/Trigger-Kontroller	RTRG,RSCL,RGVT,
	RLPR,RFAC
Tagging-System	RTAC,RTSC
Innen-Detektor	RZAC, TZAC
Crystal-Barrel 1	RCB1, TCB1
Crystal-Barrel 2	RCB2, TCB2
Flugzeitwand	RTOF
TAPS	RTAP,RTAS
DAQ	RDAQ
Slowcontrol	RSLC

A.1 Bank RTAC - Daten Tagger

bank size in words (16 bit) incl. header	
ADC Ladder 1, PM left	ADC Ladder 1, PM right
...	...
ADC Ladder 14, PM left	ADC Ladder 14, PM right
TDC Ladder 1, PM left	TDC Ladder 1, PM right
...	...
TDC Ladder 14, PM left	TDC Ladder 14, PM right
ELSA-HF, 33 1/3 MHz	TDC tagger-or
CAMAC Latch 2	CAMAC Latch 1
free	CAMAC Latch 3
Latch 2	Latch 1
Latch 4	Latch 3
...	...
Latch 22	Latch 21
Catch header word 1, module 1	
Catch header word 2, module 1	
Catch header word 3, module 1	
TDC entry 1, module 1	
...	
TDC entry N1, module 1	
0xcfed1200 (end marker)	
...	
Catch header word 1, module 8	
Catch header word 2, module 8	
Catch header word 3, module 8	
TDC entry 1, module 8	
...	
TDC entry N8, module 8	
0xcfed1200 (end marker)	

31 ... 16 15 ... 0

ADC value (10 bit):

free	ADC value

15 ... 10 9 ... 0

TDC value (11 bit):

free	TDC value

15 ... 11 10 ... 0

CAMAC Latch and Latch (16 bit):

Latch value

15 ... 0

TDC entry (32 bit):

Fibre index	TDC value

31 ... 16 15 ... 0

For every fibre multiple entries are possible (multihit-TDC)
Fibre index 0 is the reference channel.
Error words begin with 0xFFFxxxxx.

A.2 Bank RTSC - Zähler-Daten Tagger

bank size in words (16 bit) 32 * 2 + 2
Scaler Ladder 1
...
Scaler Ladder 14
Scaler Tagger-Or
Scaler Gamma-Veto
Scaler Ladder 1 * GV
...
Scaler Ladder 14 * GV
Scaler Taggeror * GV
Scaler GV * GV
Chamber Scaler 1
...
Chamber Scaler 352
Catch header word 1, scaler module 1
Catch header word 2, scaler module 1
Catch header word 3, scaler module 1
Header CMC 0
Scaler 0
...
Scaler 31
Time and hit pattern 1
Time and hit pattern 2
Trailer CMC 0
...
Header CMC 3
Scaler 96
...
Scaler 127
Time and hit pattern 1
Time and hit pattern 2
Trailer CMC 3
0xcfed1200 (end marker)

31 ... 0

Scaler Ladder (32 bit):

32 bit scaler value

31 ... 0

Chamber Scaler entry (24 bit):

free	Scaler value

31 ... 24 23 ... 0

Catch Scaler (32 bit):

1	Scaler value

31 30 ... 0

The first 32 channels are unused, so fibre index 1 corresponds to the scaler entry number 33!
Scaler entries are sorted by ascending fibre index 1 .. 480
Error words begin with 0xFFFxxxxx.

A.3 Bank RZAC - Daten Innen-Detektor

bank size in words (16 bit) 2N + 2 + 34	
TDC entry Nr. 1	
TDC entry Nr. 2	
...	
TDC entry Nr. N	
ADC discr. sum 2	ADC discr. sum 1
ADC discr. sum 4	ADC discr. sum 3
...	...
ADC discr. sum 32	ADC discr. sum 31
0xFFFF	ADC discr. sum 33
31 ... 16	15 ... 0

TDC entry (32 bit):

free	AR	index	TDC value
31 ... 24	23	22 ... 12	11 ... 0

index: fibre index
AR: Autorange bit to indicate low/high range

ADC value (10 bit):

free	ADC value
15 ... 10	9 ... 0

A.4 Bank TZAC - Tabellen-Daten Innen-Detektor

bank size in words (16 bit) 576 + 2	
TDC table entry Nr. 2	TDC table entry Nr. 1
TDC table entry Nr. 4	TDC table entry Nr. 3
...	...
TDC table entry Nr. 576	TDC table entry Nr. 575
31 ... 16	15 ... 0

Table entry (16 bit):

Fibre index for this TDC channel
15 ... 0

A.5 Bank RZSC - Zähler-Daten Innen-Detektor

bank size in words (16 bit) 48 * 2 + 2
Scaler 1, channel 1
...
Scaler 1 channel 16
Scaler 2, channel 1
...
Scaler 2 channel 16
Scaler 3, channel 1
...
Scaler 3 channel 16
31 ... 0

Scaler entry (24 bit):

free	scaler value
31 ... 24	23 ... 0

A.6 Bank RCB1,RCB2 - Daten Crystal-Barrel

bank size in words (16 bit) 2N + 2
ADC entry Nr. 1
ADC entry Nr. 2
...
ADC entry Nr. N

31 ... 0

ADC entry (32 bit):

If bit 30 is zero: (No NGF functions are used, data is suppressed by software)

Ped	0	free	AR	index	ADC value
31	30	29 ... 24	23	22 ... 12	11 ... 0

If bit 30 is one: (The data is pedestal suppressed by NGF)

0	1	AR	free	index	free	ADC value
31	30	29	28,27	26 ... 16	15 ... 12	11 ... 0

index: crystal index
AR: Autorange bit 0: low range, 1: high range
ped: Datatype, 0: Data is pedestal suppressed, 1: not suppressed

A.7 Bank TCB1,TCB2 - Tabellen-Daten Crystal-Barrel

bank size in words (16 bit) 3074
ADC table entry Nr. 1 (low range)
ADC table entry Nr. 2 (low range)
...
ADC table entry Nr. 768 (low range)
ADC table entry Nr. 1 (high range)
ADC table entry Nr. 2 (high range)
...
ADC table entry Nr. 768 (high range)

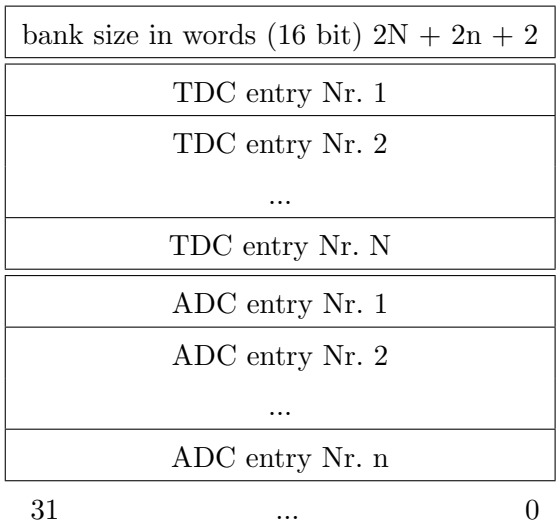
31 ... 0

Table entry (32 bit):

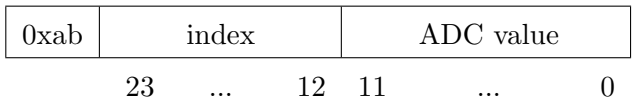
free	Index	Pedestal	Threshold
31	30 ... 20	19 ... 10	9 ... 0

Since aug. 2002 there are no longer pedestals for high range, so the bank size is only 1537 words.

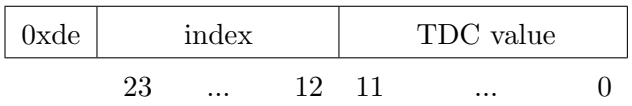
A.8 Bank RTOF - Daten Flugzeitwand



ADC entry (32 bit):



TDC entry (32 bit):



A.9 Bank RGVT - Daten Gamma Veto Detektor

bank size in words (16 bit) incl. header	
ADC value module 2	ADC value module 1
ADC value module 4	ADC value module 3
ADC value module 6	ADC value module 5
ADC value module 8	ADC value module 7
ADC analog sum	ADC value module 9
ADC Scintillator 2	ADC Scintillator 1
TDC value module 2	TDC value module 1
TDC value module 4	TDC value module 3
TDC value module 6	TDC value module 5
TDC value module 8	TDC value module 7
TDC value sum	TDC value module 9
TDC Scintillator 2	TDC Scintillator 1
Multihit TDC Hit 2	Multihit TDC Hit 1
Multihit TDC Hit 4	Multihit TDC Hit 3
Multihit TDC Hit 6	Multihit TDC Hit 5
Multihit TDC Hit 8	Multihit TDC Hit 7
31 ... 16	15 ... 0

ADC value (10 bit):

free	ADC value
15 ... 10	9 ... 0

TDC value (11 bit):

free	TDC value
15 ... 11	10 ... 0

Multihit TDC value (16 bit):

TDC value
15 ... 0

A.10 Bank RTRG - Daten Sync-/Trigger-Kontroller

bank size in words (16 bit) incl. header	
Input PLU 5	Triggerregister
Input PLU 7	Input PLU 6
DAQ Busy 1 MHz	Input PLU 8
TCU Status	Life Clock 4 MHz
free	Input PLU 9
31 ... 16	15 ... 0

```
Triggerregister (16 bit):

15                                  0
 X X X X X X X Sp T2 T1 I2 I1 C3 C2 C1
                |  |  |  |  |  |  |  '->Triggercode 1
                |  |  |  |  |  |  '->Triggercode 2
                |  |  |  |  |  '->Triggercode 3
                |  |  |  |  '->Innencode 1
                |  |  |  '->Innencode 2
                |  |  '->Tofcode 1
                |  '->Tofcode 2
                '->Spill on
```

```
Input PLU 5 (16 Bit):

15                                  0
 X X X X X X X Sp T2 T1 I2 I1 C3 C2 C1
                |  |  |  |  |  |  |  '->Triggercode 1
                |  |  |  |  |  |  '->Triggercode 2
                |  |  |  |  |  '->Triggercode 3
                |  |  |  |  '->Innencode 1
                |  |  |  '->Innencode 2
                |  |  '->Tofcode 1
                |  '->Tofcode 2
                '->Spill on
```

```
Input PLU 6 (16 Bit):

15                                  0
 X X X X X X X x F2 F1 C2 C1 T3 T2 T1
                |  |  |  |  |  |  |  '->Triggercode 1
                |  |  |  |  |  |  '->Triggercode 2
                |  |  |  |  |  '->Triggercode 3
                |  |  |  |  '->CB-Multipl. 1
                |  |  |  '->CB-Multipl. 2
                |  |  '->Face 1
                |  '->Face 2
                '->NC
```

```
Input PLU 7 (16 Bit):

15                                  0
 X X X X X X X x  x  x  x T3 T2 T1
                |  |  |  |  |  |  |  '->Triggercode 1
                |  |  |  |  |  |  '->Triggercode 2
                |  |  |  |  |  '->Triggercode 3
                |  |  |  |  '->NC
                |  |  |  '->NC
                |  |  '->NC
                |  '->NC
                '->NC
```

```
Input PLU 8 (16 Bit):

15                                  0
 X X X X X X X F8 F7 F6 F5 F4 F3 F2 F1
                |  |  |  |  |  |  |  '->FACE-Mult 1
                |  |  |  |  |  |  '->FACE-Mult 2
                |  |  |  |  |  '->FACE-Mult 3
                |  |  |  |  '->FACE-Mult4
                |  |  |  '->FACE-Mult 5
                |  |  '->FACE-Mult 6
                |  '->FACE-Mult 7
                '->FACE-Mult 8
```

```
DAQ Busy 1 MHz (16 Bit):

  C16 C15 C14 C13 C12 C11 C10 C9 C8 C7 C6 C5 C4 C3 C2 C1
```

```
Life Clock 4 MHz (16 Bit):

  C16 C15 C14 C13 C12 C11 C10 C9 C8 C7 C6 C5 C4 C3 C2 C1
```

A.11 Bank RSCL - Zähler-Daten Sync-/Trigger-Kontroller

bank size in words (16 bit) incl. header	
Life TaggerOr 2	Life TaggerOr 1
Life GammaVeto 2	Life GammaVeto 1
Life Tag * GamV 2	Life Tag * GamV 1
Life Innen 2	Life Innen 1
Life TOF 2	Life TOF 1
Life Clock 20MHz*) 2	Life Clock 20MHz*) 1
Spill TaggerOr Lsb	Spill TaggerOr Msb
Spill GammaVeto Lsb	Spill GammVeto Msb
Spill Tag*GamV Lsb	Spill Tag*GamV Msb
Spill Innen1 Lsb	Spill Innen1 Msb
Spill Innen2 Lsb	Spill Innen2 Msb
Spill TOF Lsb	Spill TOF Msb
Spill Plu3out4 Lsb	Spill Plu3out4 Msb
Spill Plu3out5 Lsb	Spill Plu3out5 Msb
Spill Plu3out6 Lsb	Spill Plu3out6 Msb
Spill Plu4out6 Lsb	Spill Plu4out6 Msb
Spill Plu4out7 Lsb	Spill Plu4out7 Msb
Spill Lsb 1	Spill Msb 2
...	...
Spill Lsb 7	Spill Msb 8
Spill Clk 20MHz*) Lsb	Spill Clk 20MHz*) Msb
31 ... 16	15 ... 0

*): before run 2409 this was a 1MHz clock

```
Life TaggerOr 1 (16 Bit):
             X X X C13 C12 C11 C10 C9 C8 C7 C6 C5 C4 C3 C2 C1

Life TaggerOr 2 (16 bit):
             C29 C28 ------------------------------  C15 C14

Spill TaggerOr Msb (16 Bit):
             C32 C31 C30 C29 -------------------- C19 C18 C17

Spill TaggerOr Lsb (16 Bit):
             C16 C15 C14 ------------------------ C3  C2  C1
```

Plu5out7	Plu5out6
Plu6out6	Plu5out8
Plu6out8	Plu6out7
Plu7out6	Plu7out5
Fast Reset	Plu7out7
Event	Sys Reset
Plu8out7	Plu8out6
1 MHz Clock 2. level	Plu8out8
CB Upstream Up	CB Upstream Left
CB Upstream Down	CB Upstream Right
CB Downstream Up	CB Downstream Left
CB Downstream Down	CB Downstream Right
nc 2	nc 1
...	...
nc 8	nc 7
Life Tag*Gamma (Del.) 2	Life Gamma-Veto (Del.) 1
...	...
nc	nc
nc	nc
nc	nc
31 ... 16	15 ... 0

A.12 Bank RFAC - Daten FACE

bank size in words (16 bit): n + 3	
Face entry 1	Number of clusters
Face entry 3	Face entry 2
...	...
Face entry n	Face entry n-1

31 ... 16 15 ... 0

If the number of face entries is even, the size is n+4 and the last zebra entries are:

0xFFFF	Face entry n

31 ... 16 15 ... 0

```
Face entry (16 Bit):

15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
       |  |  |                 |              |
       |  |  |                 |              +--
       |  |  |                 |                . column-address
       |  |  |                 +-----------------
       |  |  |                                  . row-address
       |  |  +-----------------------------------
       |  |
       |  +--------------------------------------- New Cluster Flag
       |
       +------------------------------------------ New Event Flag
```

A.13 Bank RLPR - Daten Lichtpulser

bank size in words (16 bit) incl. header	
Flasher Status Down	Flasher Status Up
Flasher Status Down	Flasher Status Up
Flasher Down Ref ADC 2	Flasher Up Ref ADC 1
Flasher Up Sig ADC 4	Flasher Up Sig ADC 3
Flasher Down Sig ADC 6	Flasher Up Sig ADC 5
Flasher Down Sig ADC 8	Flasher Down Sig ADC 7
ADC 10, nc	ADC 9, nc
31 ... 16	15 ... 0

ADC value (12 bit):

free	ADC value
15 ... 12	11 ... 0

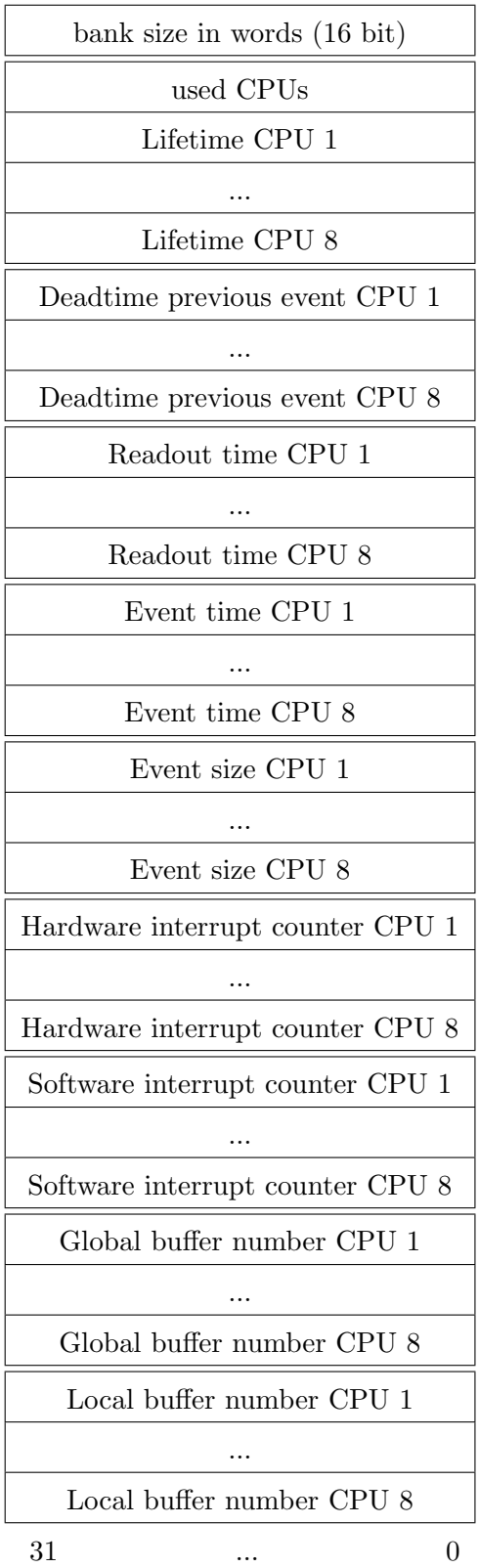
```
Flasher Status (16 Bit):

 X X X X X X X X Sp T2 T1 I2 I1 C3 C2 C1
                 |  |  |  |  |  |  |  '->Filtercode 1
                 |  |  |  |  |  |  '->Filtercode 2
                 |  |  |  |  |  '->Filtercode 3
                 |  |  |  |  '->Filtercode 4
                 |  |  |  '->Filtercode 5
                 |  |  '->Filtercode 6
                 |  '->Online
                 '->Flasher on
```

A.14 Bank RSLC - Daten Slowcontrol

bank size in words (16 bit) incl. header			
Crate Status (Crate 16-31)		Crate Status (Crate 0-15)	
Crate Status (Crate 48-63)		Crate Status (Crate 32-47)	
RESERVED			
RESERVED			
Profibus Node 3	Profibus Node 2	Profibus Node 1	Profibus Node 0
...	...	...	...
Profibus Node 31	Profibus Node 30	Profibus Node 29	Profibus Node 28
Target cell pressure		Target level diode	
RESERVED			
RESERVED			
RESERVED			
Reserved		Tagger field (mT)	
RESERVED		Tagger temperature ($10^{-}1C$	
RESERVED		Tagger chamber flow	
RESERVED		RESERVED	
Beam scan rate 0		Beam scan position 0	
...		...	
Beam scan rate 199		Beam scan position 199	
Gamma Veto HV demand volt Ch 0		Gamma Veto HV current volt Ch 0	
Reserved Ch 0		Gamma Veto HV status Ch 0	
...		...	
Gamma Veto HV demand volt Ch 31		Gamma Veto HV current volt Ch 31	
Reserved Ch 31		Gamma Veto HV status Ch 31	
Tagger HV demand volt Ch 0		Tagger HV current volt Ch 0	
Reserved Ch 0		Tagger HV status Ch 0	
...		...	
Tagger HV demand volt Ch 31		Tagger HV current volt Ch 31	
Reserved Ch 31		Tagger HV status Ch 31	
31 ... 16		15 ... 0	

TOF HV demand volt Ch 1	TOF HV current volt Ch 1
Reserved Ch 1	TOF HV status Ch 1
...	...
TOF HV demand volt Ch 255	TOF HV current volt Ch 255
Reserved Ch 255	TOF HV status Ch 255
Scifi HV demand volt Ch 1	Scifi HV current volt Ch 1
Reserved Ch 1	Scifi HV status Ch 1
...	...
Scifi HV demand volt Ch 40	Scifi HV current volt Ch 40
Reserved Ch 40	Scifi HV status Ch 40
Tagger top chamber HV demand volt	Tagger top chamber HV current volt
Reserved top chamber	Tagger top chamber HV current
Tagger bottom chamber HV demand volt	Tagger bottom chamber HV current volt
Reserved bottom chamber	Tagger bottom chamber HV current
Goniometer demand position axis 1	Goniometer actual position axis 1
...	...
Goniometer demand position axis 5	Goniometer actual position axis 5
Goniometer status	
Scifi Tagger HV demand volt Ch 0	Scifi Tagger HV current volt Ch 0
Reserved Ch 0	Scifi Tagger HV status Ch 0
...	...
Scifi Tagger HV demand volt Ch 31	Scifi Tagger HV current volt Ch 31
Reserved Ch 31	Scifi Tagger HV status Ch 31
Scifi Tagger Booster HV demand volt Ch 0	Scifi Tagger Booster HV current volt Ch 0
Reserved Ch 0	Scifi Tagger Booster HV status Ch 0
...	...
Scifi Tagger Booster HV demand volt Ch 31	Scifi Tagger Booster HV current volt Ch 31
Reserved Ch 31	Scifi Tagger Booster HV status Ch 31
31 ... 16	15 ... 0

A.15 Bank RDAQ - Daten DAQ



A.16 Das ZEBRA-Datenformat

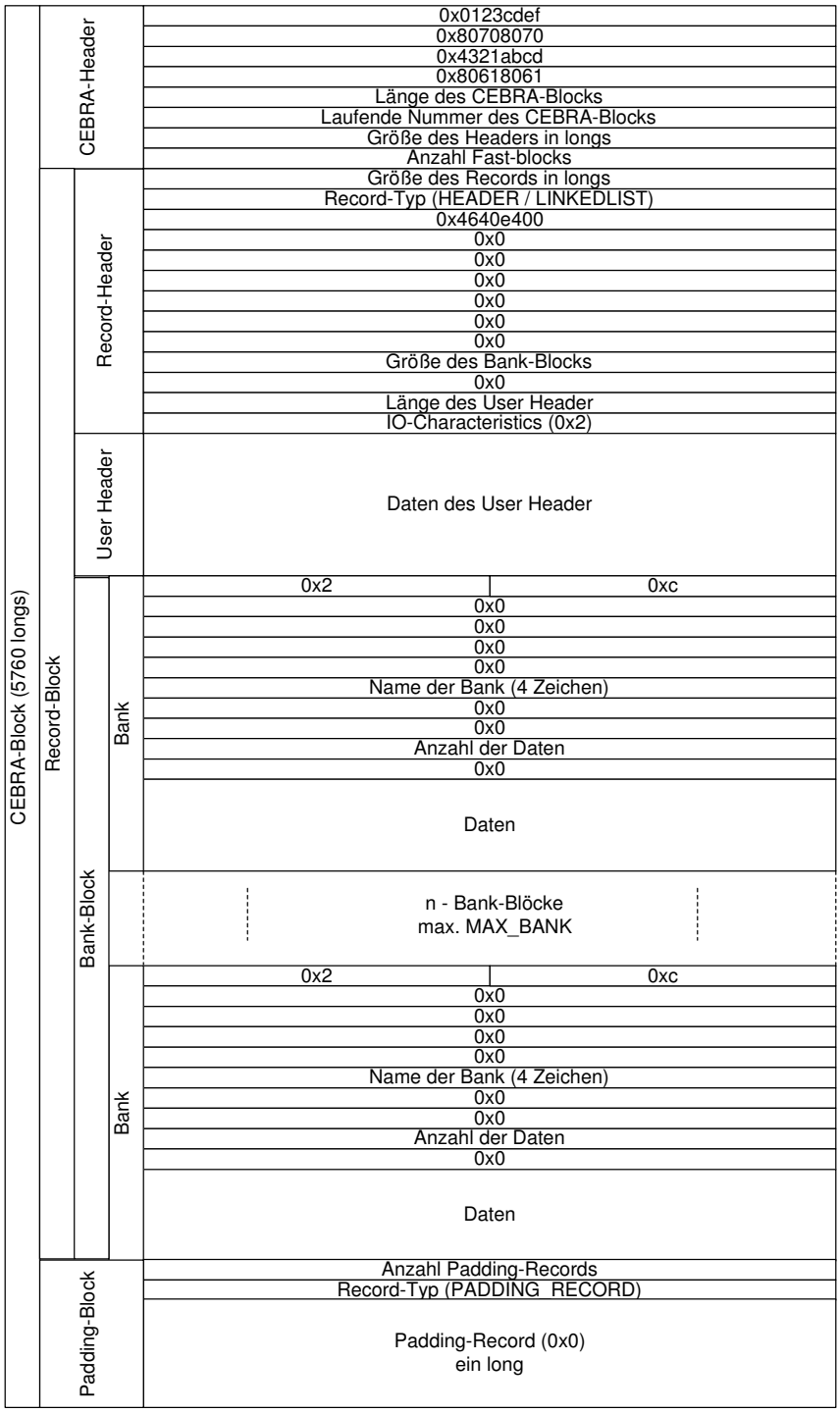


Abbildung A.1: Datenstruktur des ZEBRA-Datenformats

Anhang B

Beispieldaten

B.1 Das Tagging-System

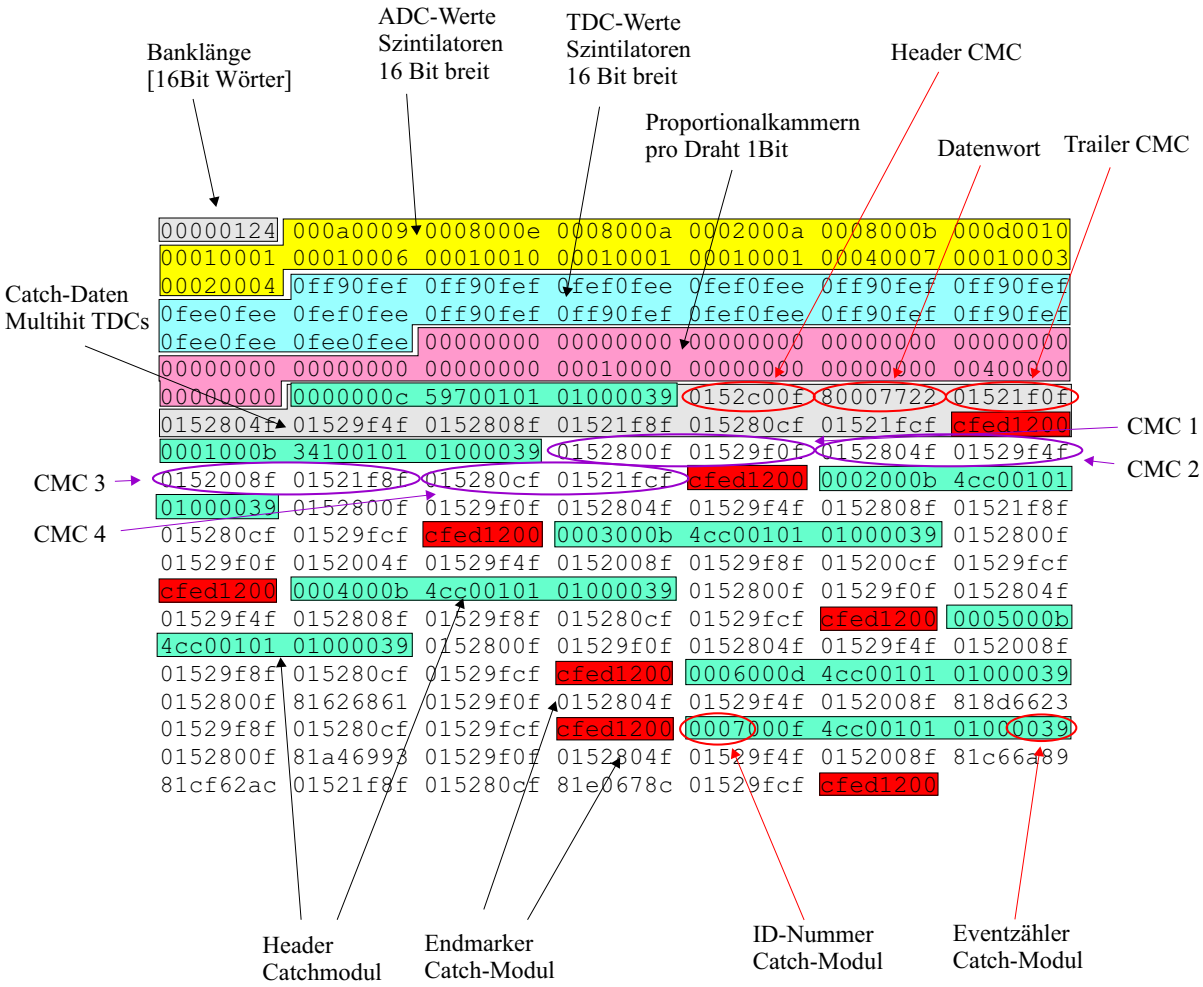


Abbildung B.1: Beispieldatensatz Tagger

B.2 Der Innen-Detektor

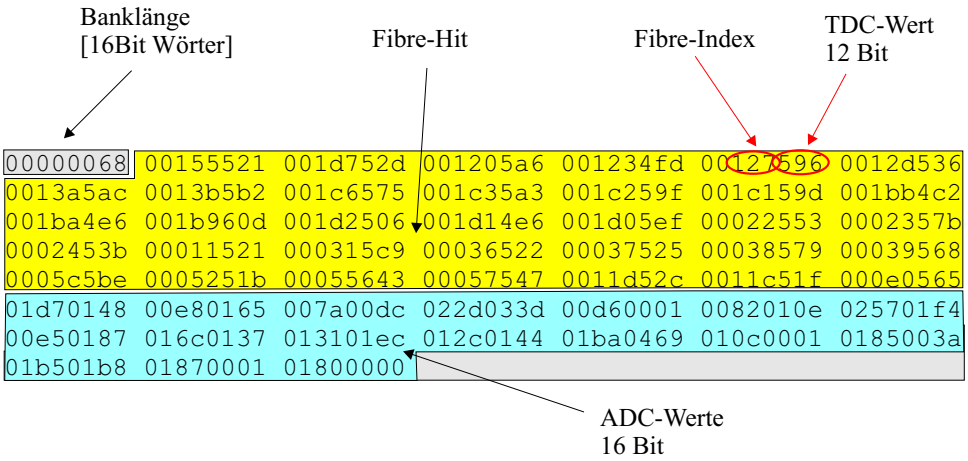


Abbildung B.2: Beispieldatensatz Innen-Detektor

B.3 Der Crystal-Barrel-Detektor

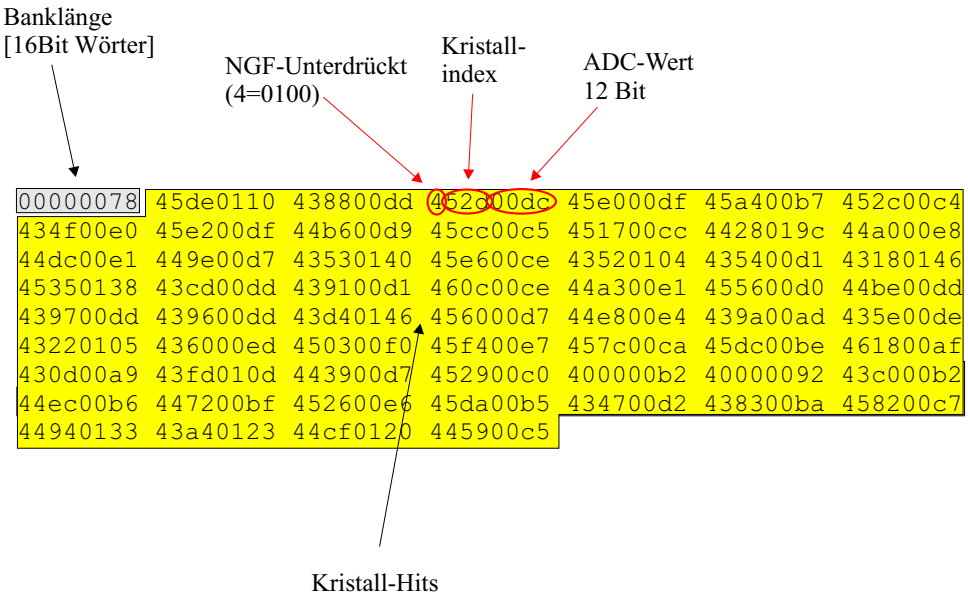


Abbildung B.3: Beispieldatensatz Crystal-Barrel-Detektor

B.4 Die Flugzeitwand - ToF

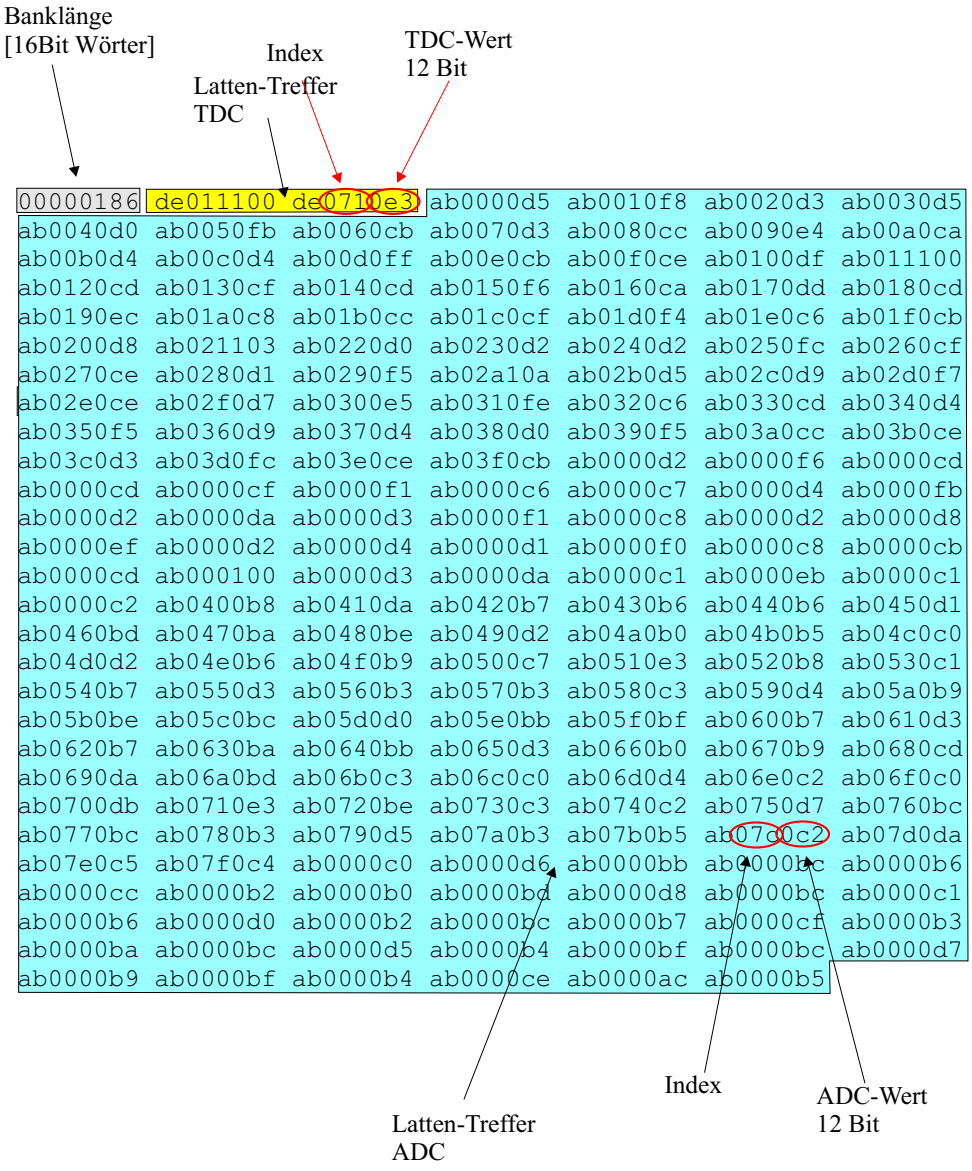


Abbildung B.4: Beispieldatensatz ToF-Detektor

Anhang C

Befehle im Kommandointerpreter

C.1 Befehle Grundmodul

`INIT` :

Initialisierung der Auslese und Aufbau der Datenverbindung zum Event-Saver

`START` :

Starten der Datennahmeschleife

`STOP` :

Stoppen der Datennahmeschleife

`ABORT` :

Abbrechen eines Startvorgangs nachdem die Datenverbindung bereits aufgebaut gewesen ist

`COUNTER` :

Ausgabe der Zählerstände des Sync-Moduls

Format: Trigger,life-time,deadtime1,deadtime2,clock

`DATARATE` :

Ausgabe der aktuellen Datenrate über einen Zeitraum von 1s gemittelt

`DEADTIME` :

Ausgabe des aktuellen life-time zu dead-time Verhältnisses über eine 1s gemittelt

`RESET` :

Ausführen eines Sys-Resets im Triggersystem

`RUNNR` :

Setzen der Run-Nummer

`STATUS` :

Abfrage des aktuellen Zustands: 0=bereit,1=Datenverbindung aufgebaut, 2=Datenschleife betreten

`KILLPROG` :

Alle Prozesse werden beendet, auch der Grundprozess

`CMDLOGON` :

Aktivieren des Sendens aller Kommandos an das Logging-System

`CMDLOGOFF` :

Deaktivieren des Sendens aller Kommandos an das Logging-System

`USER` :

Aufrufen eines Benutzerkommandos. Hierbei wird das Wort "USER" abgeschnitten und der Text dahinter wird an die Funktion `parseCommand(...)` übergeben

C.1.1 Zustand des Sync-Client-Moduls

SYNC

`SB` :

Setzen des BUSY-Signals am Sync-Modul

`CB` :

Löschen des BUSY-Signals am Sync-Modul

`SO` :

Setzen des OK-Signals am Sync-Modul

`CO` :

Löschen des OK-Signals am Sync-Modul

`CS` :

Löschen aller Zähler des Sync-Moduls

`AC` :

Aktivierung des Sync-Moduls ausgeben

`STATUS` :

Ausgabe des Status-Register des Sync-Moduls

`WRITESTATUS` :

"WRITESTATUS xx": Wert xx ins Status-Register des Sync-Moduls schreiben

`LATCH` :

Ausgabe des Latch auf dem Sync-Modul

`WRITELATCH` :

"WRITELATCH xx": Wert xx ins Latch des Sync-Moduls schreiben

`CLEARLATCH` :

Das Latch-Register auf dem Sync-Modul löschen

`BUFNUM` :

Ausgabe der auf dem Sync-Bus angelegten Buffernummer

C.1.2 Zähler des Sync-Client-Moduls

SCAL

`BUF` :

Ausgabe der geschriebenen Datenbuffer

`TRIG` :

Ausgabe der Anzahl der per Software gezählten Trigger

`RATE` :

Ausgabe der Trigger-Rate

C.1.3 Prozessstatus

PROG

`DT` :

Überprüfung, ob der DataThread ausgeführt wird

`RT` :

Überprüfung, ob der ReadoutThread ausgeführt wird

C.2 Sync-/Trigger-Kontroller

C.2.1 Sync-Master-Modul

`SETCPU` *`Ids`* :

Die Sync-Clientmodule, die durch Komma separierte Id-Nummer angegeben sind, werden aktiviert

Beispiel: SETCPU 1,2,3

`OK,BUSY` :

Ausgabe des jeweilgen Teils des Eingangsregister für OK und BUSY in der dezimaler Form (Maximal 8Bit breit)

`TEST` :

Überprüfung, ob der ReadoutThread ausgeführt wird

`CLEARACTIVE` :

Deaktivieren aller Sync-Clientmodule, die am Sync-Bus angeschlossen sind

`BUFNUM` :

Setzen der Buffernummer am Sync-Bus

`SYSRESET` :

Ausführen eines System-Resets durch Öffnen des Triggers

`TRIGGER` [***Dateiname***] :

Setzen der Triggerkonfigurationsdatei

Beispiel: TRIGGER /home/daq/TriggerFiles/clock.sts

`STATUSREG` :

Ausgabe des Statusregisters

`BEAMDIAG` :

Ausführen einer speziellen CAMAC-Sequenz zur Bestimmung von Zählraten für die Beamdiagnose

C.2.2 Lichtpulsersteuerung

`START` :

Starten der Blitzsequenz, welche über die Konfigurationsdatei angegeben wurde

`STOP` :

Stoppen des Lichtpulsers

`ON,OFF` :

Ansteuern des Lichtpulser durch den Sync-/Trigger-Kontroller aktivieren bzw. deaktivieren

`FILE` [***Dateiname***] :

Setzen der Steuerdatei für den Lichtpulser

Beispiel: FILE

Dateiname

`STATUS` :

Abfrage des Zustands des Lichtpulsers

0=ENDE; 1=AKTIV

`WAITFINISHED` :

Diese Kommando wird erst beendet, wenn der Lichtpulser mit dem Ausführen der Blitzsequenz fertig ist

`PERCENTAGE` :

Prozentsatz der Fertigstellung des Lichtpulserruns

`TESTADC` :

Testen des ADC-Moduls für die Referenz-Quelle

C.2.3 Radiatorsteuerung

`ON,OFF` :

Ansteuern der Radiatorsteuerung durch den Sync-/Trigger-Kontroller aktivieren bzw. deaktivieren

`FILE` [***Dateiname***] :

Setzen der Steuerdatei für mit Radiatorpositionen

Beispiel: FILE [Dateiname]

`NEXTPOS` :

Anfahren der nächsten Position aus der Positionsdatei

`PERCENTAGE` :

Prozentsatz der Fertigstellung der Positionsdatei

`STATUS` :

Abfrage des Zustands des Lichtpulsers

0=Position erreicht; 1=Positionierung läuft; 2=Fehler

C.3 Event-Saver

`CONNECT` :

Verbindungsaufbau für lokalen Eventbuilder vorbereiten und 4 Sekunden auf eine Verbindung warten. Bei erfolgreicher Verbindung wird OK zurückgegeben, ansonsten FAILED.

`RUN` :

Setzen von Runnummer, etc.

`DB` :

Ausführen von Datenbankfunktionen, wie z.B. Dateinamen setzen, Datenbank schliessen, etc.

`START` :

Starten aller Prozesse für die Verwaltung der Verbindungen der loaklen Eventbuilder und Verarbeitungsprozesse der Daten

`ABORT` :

Abbrechen der Initialisierungsphase.

`STOP` :

Stoppen der Datennahmeprozesse.

`STATUS` :

Status des Zustands des Event-Savers abfragen. 0,1,2,3

`EVENTD` :

Überprüfung, ob der ReadoutThread ausgeführt wird

`DATARATE` :

Ausgabe der Datenrate in MB/s

`KILL` :

Allen Prozessen das Signal SIG_KILL senden und den Event-Saver stoppen

`SCON,SCOFF` :

Aktivieren bzw. Deaktivieren der Auslese der Slow-Control-Informationen

`KILLPROG` :

Überprüfung, ob der ReadoutThread ausgeführt wird

`CMDLOGON,CMDLOGOFF` :

Aktivieren bzw. Deaktiviern der Statusausgaben für gesendete Kommandos des Kommandointerpreters

`LOGOUT` :

Kontrollsession beenden

Anhang D

Bus-Systeme

Die gebräuchlichsten und von der Industrie unterstützten Systeme sind:

- PCI-Bus
  Hauptanwendungsgebiet sind Geräte im Consumer- und IT-Bereich, wie z.B. Workstations, Server , Home-PC, etc.
- CompactPCI ist die Industrienorm des PCI-Bus, d.h. höhere Anforderungen an EMV, ...
- VMEbus ist eine weitere Industrienorm, die durch den 68000er Prozessor von Motorola geprägt wurde.

Außerdem existieren Busse, die physikalisch motiviert sind und hauptsächlich auch dort Anwendung finden:

- CAMAC oder die modernere Variante FastCAMAC
- FastBus

In einem Rechnersystem sind nicht nur der Prozessor (CPU) und das Speichersystem (RAM) von entscheidender Bedeutung, auch die Verbindungsstrukturen sind von großer Wichtigkeit. In einem kompletten System exisistieren eine Vielzahl von Verbindungselementen.
In einem konventionellen PC treten spezialisierte Verbindungen (Busse) im mikroskopischen wie im makroskopischen Bereich auf.

- Prozessorinterne Busse: Verbindungen von Leitwerk,Registern, Rechenwerken,....
- Prozessor/Speicherbus: Kopplung des Prozessors mit dem Speicher
- Systembus: Verbindung von Prozessor mit Peripherie-Elementen (Grafikkarte, E/A-Karten, ...)
- Peripherie-Busse: Anbindung von externen Geräten mit dem Rechnersystem (Festplatten, Bandlaufwerke, ...)

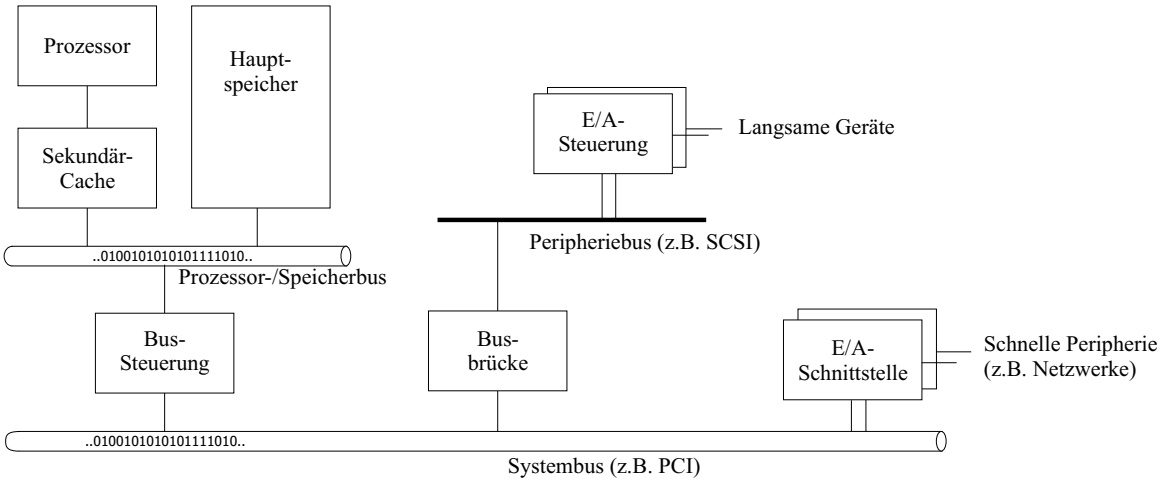


Abbildung D.1: Aufbau eines Busses in einem konventionellen PC

Abbildung D.1 zeigt schematisch diesen Aufbau. Es ergibt sich eine Hierarchie in den Buss-Systemen in einem Rechner. Die Struktur und Leistungsfähigkeit ist nach den Anforderungen im jeweiligen Bereich angepasst. Hier einige Beispiele:

- Speicherbusse sind auf hohe Leistung (Datendurchsatz) optimiert. Dies hat zur Folge, dass sie sehr kurz sein müssen, und möglichst wenig Teilnehmer angeschlossen sind.
- Peripheriebusse sind mehr auf die Anwendung optimiert, d.h. auf eine große Anzahl von Teilnehmern, aufwendigere Steuerelemente, Überbrückung von großen Distanzen, etc. .

Die Übergänge zwischen diesen Bussen werden von sogenannten Brücken (engl.: Bridges) übernommen. Diese entkoppeln die Busse voneinander und stellen weitere Kontroller zur Verfügung, wie z.B. einen DMA-Kontroller für einen schnellen Datentransport.

Ein weiteres Merkmal von Bussen ist, dass es sich um einen von den Teilnehmern gemeinsam genutzen Datenweg handelt. Es existiert im Bussystem eine Institution, die die Zugriffe der Teilnehmer steuert. Existierte nur ein aktives Element (auch Master genannt) auf dem Bus, so wäre dieses Element nicht nötig. Da heutige Systeme aber mehrere aktive Teilnehmer erfordern (Multi-Master-Betrieb), existiert eine sogenannte Arbitrierungseinheit[1], die über die Zugriffe auf dem Bus "richtet". Die Grundstruktur der Bus-Systeme ist verdeutlicht in Abbildung D.2. Er lässt sich in Steuer-, Adress- und Datenleitungen unterteilen. Diese Merkmale variieren je nach Anwendungsgebiet. Wesentliche Merkmale sind:

- Die Breite der Daten- und Adressleitungen.

  Es existieren hierbei zwei Möglichkeiten. Die binären Informationen werden entweder alle parallel über mehrere Leitungen (paralleler Bus) oder nacheinander über eine Leitung (serieller Bus) übertragen. Die serielle Form wird meist zur Übertragung über große Strecken verwendet. Im

[1] vom engl. arbiter = Richter

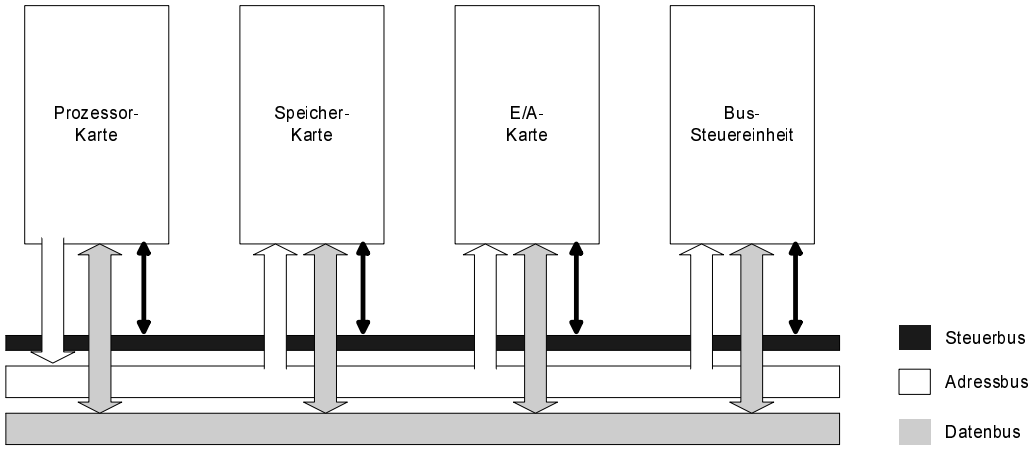


Abbildung D.2: Grundstruktur eines Busses mit Teilnehmern

Parallel-Betrieb sind die gebräuchlichsten Breiten 16, 32 oder 64 Leitungen, der PCI-Bus, z.B. besitzt 32 Adress- und Datenleitungen.

- Der Betrieb von Adress- und Datenleitungen

  Diese werden häufig aufgrund von Engpässen in der Anzahl verfügbarer Leitungen mehrdeutig benutzt (Multiplex-Betrieb). Hierbei werden z.B. auch die Adressleitungen als Datenleitungen verwendet, um den Datendurchsatz zu verdoppeln.

- Die zeitliche Abfolge der Signale auf den Leitungen.

  In einem synchronen Bus wird durch einen externen Taktgeber der Zeitpunkt, wann ein Zugriff auf den Bus möglich ist, vorgegeben. Häufig handelt es sich hierbei um einfache Abläufe, die durch Erhöhung der externen Taktfrequenz beschleunigt werden können. Nachteil ist, dass die Teilnehmer an die vorgegebene Busgeschwindigkeit angepasst werden müssen und diese bei hohen Taktfrequenzen wegen Signallaufzeiten und Verzerrungen (clock skew) nur sehr kurz sein können. Einsatzgebiete sind meist Prozessor- oder Speicherbusse. Der asynchrone Bus besitzt spezielle Leitungen zur Steuerung der Bustransaktionen, d.h. jeder Zugriff wird durch eine Bestätigung quittiert (handshaking). Vorteil bei diesem System ist, dass auch unterschiedlich schnelle Komponenten an diesem Bus teilnehmen können. Dies erfordert jedoch eine zeitlich genaue Abfolge der relativ komplexen Quittierungszyklen.

Technisch gibt es zwei Realisierungsformen von Bussen. Die erste Variante ist eine Grundplatine (Motherboard), auf der sich alle Komponenten und Busse schon befinden. Die Erweiterung erfolgt über eine geringe Zahl von Stecksockeln. Haupteinsatzgebiet sind Workstations, Server und PCs im Consumerbereich.
Die zweite Variante sind sogenannte Rückwandbusse (Backplanebusse). Hierbei handelt es sich um einen Rahmen, in dem sich **ein** Bus befindet. Diese Rahmen (engl. crates) können eine relativ hohe Anzahl an Zusatzmodulen in sich aufnehmen. Einsatzgebiet sind industrielle Anwendungen, wie Ro-

botersteuerungen, Fertigungsstraßen, etc. .

Es ergibt sich aus diesem kleinen Exkurs, dass die Anwendungsgebiete einen Einfluss auf die zu verwendenden Busse haben. Weiterhin existieren aus der zeitlichen Entwicklung der Experimente heraus Bussysteme, die für den Einsatz in physikalischen Experimenten entwickelt wurden.
Im folgenden werden diese Busse und ihre Eigenschaften nun kurz erläutert.
Im Bereich Front-End Elektronik (Konversion Analog→Digital) sind folgende Busse in physikalischen Anwendungen gebräuchlich:

- VMEbus
- CAMAC (Computer Automated Measurement And Control)
- FastBus

Im Bereich Back-End (Server, Workstation,...) ist der PCI-Bus etabliert.
Aus der Vielzahl von Prozessoren und Bussystemen in der Anfangszeit der Computer haben zwei Hersteller Standards gesetzt. Die Firma Motorola prägte mit ihrer 68000er-Familie den VMEbus-Standard und Intel den PCI-Bus.

D.1 Der VMEbus

1979 definierte die Firma Motorola als erste den VERSAbus für ihre 68000er-Prozessorfamilie. In dieser Zeit existierten Busse wie Multibus^{TM}, STD Bus, S-100 und Q-bus. Diese Busse haben sich jedoch nicht durchgesetzt und finden kaum noch Anwendung.

1980 wurde der VMEbus-Standard auf Basis des VERSAbuses zu einem prozessorunabhängigen Bus entwickelt, der sowohl mechanische (Größe der Module,Anzahl der Module,...), wie elektrische (Anzahl Daten- und Adressleitungen,...) Richtlinien festlegt. Die Anzahl der Module wurde auf 21 Stück festgesetzt; die mechanischen Dimensionen wurden so gewählt, dass dieser Rahmen (Crate) in ein Standard 19" Industrierack passt.

Tabelle D.1 zeigt einen Überblick der VMEbus-Normen und deren Leistungsmerkmale.

D.2 Der PCI-Bus

In der Anfangszeit der Prozessorfamilie von Intel existierten unterschiedliche Bussysteme. Begonnen wurde mit dem ISA-Bus, welcher eine 16 Bit-Datenbreite hatte. Eine schnellere Variante folgte mit dem VL-Bus und den EISA-Bus. Diese wurden aufgrund schlechter Kompatibilität wieder eingestellt. Infolge der Bus-Vielfalt entwickelte Intel den PCI-Bus und setzte hiermit einen Standard für ihre Prozessor-Familie. Die aktuelle Version des Standards ist die Version 2.2.

	VMEbus (IEEE-1014-1987)	VME64	VME64x	VME320
Busart	asynchron	asynchron	asynchron	asynchron
Adressbreite	32	64 (6U) 32/40 (3U)	64	64
Datenbreite	32	64	64	64
Takt	keiner	keiner	keiner	keiner
Transferraten	MByte/s			
BLT-Modus	40			
MBLT-Modus		80		
2eVME-Modus			160	
2eSST-Modus				320-500

Tabelle D.1: Allgemeine VMEbus Eigenschaften

Tabelle D.2 zeigt eine Übersicht der Eigenschaften. Hauptunterschiede des PCI-Busses zum VMEbus sind, dass der PCI-Bus synchron betrieben wird und die Leitungen für Daten und Adresse ein und dieselbe sind (zeitlich gemultiplext).
Um diesen Bus prozessorunabhängig zu machen, wurde beim Design vorgesehen, den Bus und den Prozessor über eine Host-Bridge miteinander zu verbinden. Des weiteren setzt diese "Brücke" PCI-Zyklen in CPU-Zyklen um und umgekehrt. Für die Anbindung des PCI-Busses an den jeweiligen Prozessortyp (INTEL, Alpha, AMD usw.) müssen nur unterschiedliche Host-Bridges verwendet werden. Diese Eigenschaft macht das PCI-Bussystem auch für zukünftige Prozessorgenerationen relativ unabhängig und erweiterbar. Mittels solcher Bridges ist auch eine Kopplung verschiedener Bussysteme möglich, wie z.B. dem VMEbus.

D.3 Der CAMAC-Bus

CAMAC[2]-Bus basiert auf dem CERN NP Standard von 1962 und wurde 1971 zu einem Standard erhoben, um eine einheitliches System in der Experimentalphysik zu bekommen und somit einen besseren Austausch innerhalb der Institutionen und preiswertere Module zu erhalten. CAMAC basiert auf einem Crate mit einer Backplane (Dataway),das 25 Module aufnehmen kann. Der Bus unterscheidet sich in seiner Struktur von anderen Standardbussen. So exisitieren auf der Backplane folgende Leitungen:

- 24 Datenleitungen für Lesezyklen
- 24 Datenleitungen für Schreibzyklen

[2] **C**omputer-**a**ided **M**easurement **A**nd **C**ontrol

	PCI-Bus 32 Ver. 1.0	PCI-Bus 32 Ver. 2.1	PCI-Bus 64 Ver. 2.0	PCI-Bus 64 Ver. 2.1
Busart	synchron	synchron	synchron	synchron
Adressbreite	32	32	32	32
Datenbreite	32	64	32	64
Bustakt [Mhz]	33	33/66	33	33/66
Transferraten 32Bit				
Non-Burst (Read/Write)	MBytes/s			
33Mhz	44/66	44/66	44/66	44/66
66Mhz	-	88/132	-	88/132
Burst (Read/Write)	MBytes/s			
33Mhz	106/117	106/117	106/117	106/117
66Mhz	-	211/234	-	211/234
Transferraten 64Bit				
Non-Burst (Read/Write)	MBytes/s			
33Mhz	-	88/132	-	88/132
66Mhz	-	-	-	172/264
Burst (Read/Write)	MBytes/s			
33Mhz	-	211/234	-	211/234
66Mhz	-	-	-	423/468

Tabelle D.2: Spezifikationen PCI-Bus

- 24 Adressleitungen, wobei jedes Modul durch eine einzige Leitung angesprochen wird
- 5 Funktionsleitungen
- 4 Unteradressleitungen

Auf Grund der Art der Adressierung von Modulen gibt es im Crate eine ausgezeichnete Position (Modul 25), in dem ein Mastermodul stecken kann. Nur dieses Modul kann auf die einzelnen Module zugreifen, d.h. die restlichen Module im Crate können keine Aktion auf dem Bus ausführen. Der Bus wird über das Mastermodul (Cratekontroller) synchron betrieben. Die mittlere Zugriffszeit über diesen Bus beträgt ca. 1,5 μs, was einer Datenrate bei Verwendung aller 24 Datenleitungen von ca. 2 MB/s entspricht.
Diese Kontroller können nun autonom sein, d.h. in ihnen befindet sich ein Prozessor, der über einen weiteren Bus an ein Rechnersystem gekoppelt ist. Eine andere Variante ist, die Cratekontroller über einen Bus (Branch) mit einem Branchkontroller zu verbinden, der die jeweiligen Aktionen über diese

Cratekontroller ausführt. Dieses System erlaubt es, sieben Crates an einem Branchkontroller zu betreiben. Das Rechnersystem, in dem sich der Branchkontroller befindet, kann auf diesem Wege eine große Anzahl von CAMAC-Modulen ansprechen. Kontroller diesen Typs existieren auf einer VMEbus-Basis.

D.4 Der FastBus

FastBus ist eine Entwicklung des U.S. NIM Commitee in Zusammenarbeit mit dem Europäischen ESONE Kommitee im Jahre 1986. Er vereinigt die Einfachheit und die logischen Strukturen von CAMAC mit den Eigenschaften eines VME-Busses. Mechanisch basiert der Standard wieder auf einem Crate mit einer Backplane, das 26 Module aufnehmen kann, und folgende Eigenschaften besitzt:

- 32 Adress- und Datenleitungen
- Die Leitungen basieren auf dem ECL-Standard und nicht TTL wie bei VMEbus. Hieraus ergibt sich eine Bandbreite von bis zhu 150MB/s
- Die Adressierung kann wie bei CAMAC geographisch erfolgen, also durch die Positionsnr. im Crate, oder über eine Adresse definiert im 32 Bit-Adressraum.
- Ein Multiprozessorsystem mit Zuweisung von Segmenten, d.h. die Backplane kann in Bereiche aufgeteilt werden, sodass in diesen die volle Bandbreite verfügbar ist.
- Daten können synchron und asynchron übertragen werden.
- Standard definiert Crate-Kopplungen

Anhang E

Netzwerksysteme

Die Nachfrage nach Standards für lokale Netzwerke (LAN - Local Area Network) ließ die Organisation IEEE (Institute of Electrical and Electronics Engineers) eine Arbeitsgruppe einrichten. Seitdem steht der Name Ethernet als Synonym für alle unter der Arbeitsgruppe 802.3 vorgeschlagenen und standardisierten Spezifikationen. Angefangen hat diese Entwicklung in den 80er Jahren beim 10-MBit-Ethernet über Koaxialkabel, dann Fast-Ethernet mit 100 MBit/s und Gigabit-Ethernet mit 1000 MBit/s und 10 GBit/s. Alle Ethernet-Varianten basieren auf denselben Prinzipien.

E.1 Das Zugriffsverfahren

Das Zugriffsverfahren basiert auf dem CSMA/CD Prinzip (Carrier Sense Multiple Access with Collision Detection). Als Mehrfachzugriffsnetz (Multiple Access) können mehrere Ethernet-Stationen unabhängig voneinander auf das Übertragungsmedium zugreifen. Alle Stationen hören permanent das Übertragungsmedium ab (Carrier Sense) und können zwischen einer freien und besetzten Leitung unterscheiden. Bei einer freien Leitung kann gesendet werden. Während der Datenübertragung wird überprüft, ob eine andere Station gleichzeitig gesendet hat und eine Kollision der Daten aufgetreten ist (Collision Detection). Ist keine Kollision aufgetreten, ist die Übertragung erfolgreich abgeschlossen. Verloren gegangene Pakete müssen durch Protokolle, wie z. B. durch TCP, neu angefordert werden. Tritt dies häufiger auf, drückt das auf die Performance des Netzwerkes.

E.2 Das Ethernet-Datenpaket

Ethernet ist ein paketvermittelndes Netzwerk. Die Daten werden in mehrere kleine Pakete, sogenannte Frames, aufgeteilt. In einem Frame werden neben den Daten auch die Zieladresse, die Quelladresse und Steuerinformationen verpackt. Als Adressen dienen die MAC-Adressen (**M**edium **A**ccess **C**ontrol). Diese Adresse ist hardwareseitig vom Hersteller eingestellt und muss eindeutig sein. Sie läßt sich im Regelfall nicht verändern.

Die maximale Länge/Größe eines Ethernet-Paketes beträgt 1538 Byte. Davon sind 1500 Byte Daten enthalten. Die minimale Länge beträgt 84 Byte, mit 46 Byte Daten. Ist die Datenmenge größer als 1500 Byte, so werden die Daten voneinander getrennt, und in 1500 Byte-Blöcken übertragen.

Präambel	Zieladresse	Quelladresse	Typfeld	Datenfeld	CRC	Inter Frame GAP
8 Byte	6 Byte	6 Byte	2 Byte	46-1500 Byte	4 Byte	12 Byte

Präambel Zeigt den Start des Ethernet-Paketes an.

Zieladresse Adresse des Empfängers.

Quelladresse Adresse des Senders.

Typfeld Gibt den Typ des Protokolls (z. B. TCP/IP) oder die Länge an.

Datenfeld Hier stehen die zu übertragenen Daten.

CRC CRC-Prüfsumme.

Inter Frame GAP Eine Pause, die 12 Byte entspricht.

Tabelle E.1: Aufbau eines Ethernet-Frames nach IEEE 802.3

Am Ende jeden Paketes erfolgt eine Pause von 12 Byte, dem Inter Frame GAP. Bei einem 10MBit Netzwerk entspricht dies $9,6\mu s$. Die Länge der Pause ändert sich bei höheren Datenraten.

E.3 Das Übertragungsmedium und Netzwerk-Topologie

Das ursprüngliche Ethernet nutzte ein Koaxialkabel als Übertragungsmedium. Dabei wurde mit einem Kabel jeweils eine Station mit mehreren anderen Stationen verbunden. Das Netzwerk wurde dann als sogenannter Bus aufgebaut. Jeweils am Kabelende wurde die Kabelstrecke mit einem Widerstand abgeschlossen.

Auf Grund der Nachteile von Netzwerken mit der Bus-Topologie und dem Koaxialkabel wurde Ethernet um den Einsatz von Twisted-Pair-Kabel der Kategorie 3 und 5 erweitert. Es handelt sich dabei um 8-adrige-Kabel, deren Adern jeweils paarweise verdrillt sind. Die Leitungsführung ist als Stern-Topologie mit Switches oder Hubs als Verteilstationen aufgebaut. Mit Switches kommt man ohne Kollisionserkennung aus und kann die Vollduplex-Übertragung nutzen. Twisted-Pair-Kabel haben allerdings eine Reichweite von nur 100 Metern, was sie für die Vernetzung von Gebäuden oder als Backbone ungeeignet macht. Aus diesem Grund wurde Ethernet auch für Glasfaserkabel standardisiert.

Heute spielt das Koaxialkabel keine Rolle mehr. Für Neuinstallationen werden generell Twisted-Pair-Kabel nach Kategorie 5, 5e oder besser 6 eingesetzt. Zur Überbrückung von längeren Strecken wird Glasfaserkabel verwendet.

	802.3		802.3z	
IEEE-Standard	(aij)	802.3u	802.3ab	802.3ae
Datenrate	10MBit/s	100MBit/s	1GBit/s	10GBit/s
Framerate	1/s			
min.	812	8.127	81.274	812.743
max.	14.880	148.809	1.488.095	14.880.952
eff. Datenrate	MByte/s			
min.	0,57	6,53	65,28	652,81
max.	1,16	11,63	116,26	1.162,64

Tabelle E.2: Übersicht IEEE 802.3

Fast-Ethernet Fast Ethernet ist die Weiterentwicklung des Ethernet-Standards mit 100 MBit/s über Twisted-Pair-Kabel. Um die Übertragungsrate von 10 MBit/s auf 100 MBit/s anzuheben wurde der Leitungscode 4B5B eingesetzt. Dabei werden 4-Bit binäre Dateninformationen in 5-Bit binäre Übertragungsinformationen codiert. Die Reichweite blieb auf nur 100 Meter beschränkt.

Gigabit-Ethernet Die hohe Netzlast, verursacht durch vielerlei Anwendungen (z. B. Internet, Multimedia, Messdatenerfassung,...) macht es notwendig, zentrale Ethernet-Stationen, wie z. B. Server und Switches mit mehr Bandbreite zu verbinden als die übrigen Stationen. Gigabit-Ethernet wurde auf der Grundlage der ursprünglichen Norm entwickelt. Erst für Glasfaserkabel, später auch für Twisted-Pair-Kabel der Kategorie 5. Beide Varianten erlauben die Übertragung von Daten mit 1000 MBit/s.

Aufgrund der geringeren Störanfälligkeit von Glasfaserverbindungen ist das Medium Glasfaser für schnelle Übertragungen nach oben hin offen. Bei Twisted-Pair-Kabeln müssen mehrere Tricks angewendet werden, um auf diese hohe Geschwindigkeit zu kommen. Grundsätzlich nutzt Gigabit-Ethernet über Twisted-Pair-Kabel alle 4 Adernpaare. Der Datenstrom wird mit PAM5x5 (Pulse-Amplituden-Modulation mit fünf verschiedenen Regeln) codiert. Ausserdem kommt eine neue Fehlerkorrektur zum Einsatz.

10-Gigabit-Ethernet Beim 10-Gigabit-Ethernet wird kein Twisted-Pair-Kabel mehr berücksichtigt. Für 10-Gigabit-Ethernet kommt nur noch Glasfaserkabel zum Einsatz. Hier wurde auf das Zugriffsverfahren CSMA/CD verzichtet. Der Betrieb erfolgt ausschließlich im Vollduplex-Modus.

Glossar

ADC

Analog to **D**igital **C**onverter

CPU

Central **P**rocessing **U**nit. Ein Chip in einem Computer, der für das Interpretieren von Befehlen und zum Ausführen von Programmen verantwortlich ist. Die CPU ist die wichtigste Komponente eines Computersystems.

DAQ

Data **A**c**Q**uisition. Die englische Abkürzung für Datennahmesystem.

DMA

Direct **M**emory **A**ccess. Direkter Zugriff, z.B. durch eine Einsteckkarte, auf den Datenspeicher ohne Verwendung des Prozessors.

GUI

Graphical **U**ser **I**nterface. Allgemeine Bezeichnung einer grafischen Oberflächengestaltung zur Bedienung von Programmen.

PCI

Peripheral **C**omponent **I**nterconnect-Bus. Von Intel entwickelter Bus zum Datenaustausch für Steckkarten in kommerziellen PC-Systemen (1992 vorgestellt).

OMG

Object **M**anagement **G**roup. 1989 gegründeter internationaler Zusammenschluss von (inzwischen) über 800 Herstellern und Anwendern mit dem Ziel, einen offenen Standard für verteilte Objektsysteme einzuführen.

Socket

Software-Schnittstelle zur Kommunikation zwischen verschiedenen Prozessen über das TCP-Protokoll.

TDC

Time to **D**igital **C**onverter

Thread

Ein Thread ist eine vom Betriebssystem autonom verwaltete Ablaufinstanz für einen Prozess bzw. Prozess-Schritt. Bei einem Prozess handelt es sich um eine Folge von Aktionen, die auf einem Prozessor in einem Programm ablaufen (meist unter Regie eines Betriebssystems).

UML

Unified **M**odeling **L**anguage. 1998 von der OMG herausgegebene Objekt-Meta-Modellierungssprache. Aus UML-Modellen kann man u.a. (mit entsprechenden Klassenbibliotheken) C++ und Java generieren. UML 2.0 derzeit in Arbeit (Revision 1 am 3. Juni 2002 freigegeben).

VMEbus

Versa **M**odule **E**uropean-Bus - Industriestandard; 2x96-pin-Europastecker, 16/32Bit Daten, 24/32Bit Adress parallel, asynchron, 7 Interruptebenen, 4 Arbitrierungsebenen, max. 24 MByte/s.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] M.E. Mermikides, "Data analysis for bubble chambre and hybrid systems", 6th CERN School of Computing.

[2] N. Isgur and G. Karl, "Hyperfine Interactions In Negative Parity Baryons," Phys. Lett. B **72** (1977) 109.

[3] T. D. Cohen and L. Y. Glozman, "Does one observe chiral symmetry restoration in baryon spectrum?," Int. J. Mod. Phys. A **17** (2002) 1327.

[4] D. B. Lichtenberg, "Baryon Supermultiplets of $SU(6) \times O(3)$ in a Quark-Diquark Model", Phys. Rev. **178** (1969) 2197–2200.

[5] M. Chemtob, "Skyrme Model Of Baryon Octet And Decuplet," Nucl. Phys. B **256** (1985) 600.

[6] S. Godfrey and N. Isgur, "Mesons In A Relativized Quark Model With Chromodynamics," Phys. Rev. D **32** (1985) 189.

[7] S. Capstick and N. Isgur, "Baryons In A Relativized Quark Model With Chromodynamics," Phys. Rev. D **34** (1986) 2809.

[8] L. Y. Glozman, W. Plessas, K. Varga and R. F. Wagenbrunn, "Unified description of light- and strange-baryon spectra," Phys. Rev. D **58** (1998) 094030.

[9] U. Loring, B. C. Metsch and H. R. Petry, "The light baryon spectrum in a relativistic quark model with instanton-induced quark forces: The strange baryon spectrum," Eur. Phys. J. A **10** (2001) 447.

[10] U. Loring, B. C. Metsch and H. R. Petry, "The light baryon spectrum in a relativistic quark model with instanton-induced quark forces: The non-strange baryon spectrum and ground-states," Eur. Phys. J. A **10** (2001) 395.

[11] G. 't Hooft, "Computation of the quantum effects due to a four-dimensional pseudoparticle", Phys. Rev. D **14** (1976) 3432.

[12] T. Nakano *et al.* [LEPS Collaboration], "Observation of S = +1 baryon resonance in photo-production from neutron," Phys. Rev. Lett. **91** (2003) 012002.

[13] B. Krusche and S. Schadmand, Study of Non-Strange Baryon Resonances with Meson Photoproduction, 17 Juni 2003, Universität Basel und Giessen.

[14] E. Aker *et al.* [Crystal-Barrel Collaboration], "The Crystal Barrel spectrometer at LEAR," Nucl. Instrum. Meth. A **321** (1992) 69.

[15] Angela Fösel, Entwicklung und Bau eines Innendetektors für das Crystal-Barrel-Experiment an ELSA/Bonn, Dissertation, Universität Erlangen, 2000.

[16] Bertram Kopf, Untersuchung der photoinduzierten Reaktion $\gamma p \to p\pi^0\pi^0$ und $\gamma p \to p\pi^0\eta$ an einem Flüssig-Wasserstoff-Target, Dissertation, Universität Dresden, 2002.

[17] Stefan Höffgen, Einbindung eines großflächigen Flugzeitspektrometers als Vorwärtsdetektor für Experimente mit CB-ELSA, Diplomarbeit, Physikalisches Institut, Universität Bonn, 2000.

[18] R. Novotny [TAPS Collaboration], "The BaF_2 photon spectrometer TAPS," IEEE Trans. Nucl. Sci. **38** (1991) 379.

[19] Christoph Schmidt, Optimierung des Datenakquisitions-Systems des Crystal Barrel Experimentes an ELSA, Diplomarbeit, Institut für Strahlen- und Kernphysik, Universität Bonn, 1999.

[20] Holger Flemming, Entwurf und Aufbau eines Zellularlogik-Triggers für das Crystal-Barrel-Experiment an der Elektronenbeschleunigeranlage ELSA, Dissertation, Ruhr-Universität Bochum, 2001.

[21] Andreas Ehmanns, Entwicklung, Aufbau und Test eines neuen Auslesesystems für den Crystal-Barrel-Detektor zur Messung photoinduzierter Reaktionen an ELSA, Dissertation, Institut für Strahlen- und Kernphysik, Universität Bonn, 2000.

[22] O. Bartholomy, Test und Modifikation des Lichtpulsersystems für den CB-ELSA-Detektor Institut für Strahlen- und Kernphysik, Diplomarbeit, Universität Bonn, 2000.

[23] M. Konrad, Ortssensitiver Detektor für hochenergetische Photonen bei höchsten Raten Physikalisches Institut, Diplomarbeit, Universität Bonn, 2001.

[24] Y. Goldschmidt-Clermont, The analysis of nuclear particle tracks by digital computer CERN 57-29 Scientific and Technical Services Division.

[25] Karsten Wittmack, Entwicklung, Bau und Test eines VME-Moduls zur Auslese des SAPHIR-Taggingsystems, Diplomarbeit, Universität Bonn, 1995.

[26] Christoph Schmidt, Referenz-Dokumentation der CB-DAQ, CB-Note 008, 2003.

[27] F.H. Heinsius *et al.*, CATCH Users Manual, COMPASS-Note-2001, Fakultät für Physik, Universität Freiburg.

[28] Marc Niebuhr, Entwicklung eines 250 Mhz-Zählers mit totfreier Auslese für das COMPASS-Experiment, Diplomarbeit, Universität Freiburg, 2000.

[29] Dokumentation Tundra Universe II, Juni 2002.

[30] Dokumentation FastBus Struck Sequenzer STR340/SFI.

[31] Dokumentation Common-C++ 1.9.6.

[32] `http://cs.nmu.edu/~benchmark/index.php?page=context`

Danksagung

Zum Abschluss möchte ich mich bei all denen bedanken, die mit zum Gelingen dieser Arbeit beigetragen haben.
Mein besonderer Dank gilt Herrn Prof. Dr. Eberhard Klempt für die interessante Aufgabenstellung und die ausgezeichnete Betreuung während meiner Promotion. Seine vielen Anregungen, die Diskussionen mit ihm und sein stetes Interesse an den Fortschritten meiner Arbeit ermöglichten wesentlich das Gelingen dieser Dissertation.
Herrn Dr. Hartmut Kalinowsky danke ich für seine ständige Bereitschaft und Unterstützung bei allen auftretenden Fragen. Mit seinem Fachwissen und seiner Kompetenz hat er in erheblichem Maße zur Lösung vieler Hard– und Softwareprobleme beigetragen.
Weiterhin geht mein Dank an die gesamte CB–ELSA–Kollaboration für die freundliche Zusammenarbeit und für die mannigfaltige Hilfe.
Meinen Eltern danke ich für ihre Unterstützung während meines gesamten Studiums und meiner Frau Lydia für ihr Verständnis und ihre Geduld während meiner Promotion.