Interner Bericht
DESY F35D–90–02
November 1990

Entwurf, Bau und Erprobung eines Triggermoduls für die Kalibrierung des ZEUS-Kalorimeters

bist auszuleihen

von

L. Hagge

Entwurf, Bau und Erprobung eines Triggermoduls für die Kalibrierung des *ZEUS*-Kalorimeters

Lars Hagge

Diplomarbeit

II. Institut für Experimentalphysik
der Universität Hamburg

Juni 1990

Inhaltsverzeichnis

1 Einleitung 4

2 Die Kalibrierung des *ZEUS*-Kalorimeters 8
- 2.1 Aufbau des *ZEUS*-Präzisionskalorimeters 8
- 2.2 Testprogramm für das *ZEUS*-Kalorimeter 12
- 2.3 Das Experimentierprogramm am CERN-SPS 14
- 2.4 Das Datennahmesystem und die Experimentkontrolle 19

3 Die Triggerbox für die FCAL-Kalibrierung 21
- 3.1 Aufgaben der Triggerbox 21
- 3.2 Planung und Entwurf 24
  - 3.2.1 Anforderungen an den Entwurf 24
  - 3.2.2 Funktionsgruppen der Triggerbox 25
  - 3.2.3 Funktionale Beschreibung der Triggerbox 29
- 3.3 Aufbau der Triggerbox 31
  - 3.3.1 Familien integrierter Schaltkreise 31
  - 3.3.2 Die einzelnen Baugruppen 32
  - 3.3.3 Mechanischer Aufbau 35
- 3.4 Die Testphase 35

4 Das Datennahme- und Kontrollsystem 38
- 4.1 Daten- und Informationsfluß im System 38
- 4.2 Stellung und Struktur des Triggerprozesses 41

5 Erste Ergebnisse von der Strahlkalibrierung der *ZEUS*-FCAL-Module 45
- 5.1 Rekonstruktion der physikalischen Daten 45

5.2 Energieauflösung, Linearität und Uniformität 49

6 Zusammenfassung **58**

A Die Systemumgebung **59**
A.1 Der VMEbus 59
A.1.1 Bussysteme 59
A.1.2 Überblick über den VMEbus 61
A.1.3 Das XVME-085-Prototyping-Modul 62
A.2 Transputer und Occam 63
A.2.1 Das Transputerkonzept 63
A.2.2 Prozeßkontrolle durch Occam 64
A.2.3 Der T800 66
A.3 Das 2TP-VME-Modul 67
A.4 TBRun: Ein Testprogramm für die Triggerbox 69

B Familien integrierter Schaltkreise **75**
B.1 TTL 75
B.2 ECL 77
B.3 CMOS 78

C Technische Dokumentation der Triggerbox **80**
C.1 General Information 81
C.1.1 Introduction 81
C.1.2 Features 81
C.1.3 Overview of the Trigger-Box 81
C.2 Hardware Preparation 84
C.2.1 Introduction 84
C.2.2 Minimum Equipment required for Operation 84
C.2.3 Hardware Configuration Options 84
C.2.4 Connecting to the Trigger-Box 93
C.2.5 Module Installation 93
C.3 Operating Instructions 96
C.3.1 Introduction 96
C.3.2 Operating Controls 96
C.3.3 Status Indicators 96
C.3.4 Operation Procedure 96

C.4 Functional Description 98
C.4.1 Introduction 98
C.4.2 VMEbus Interface Logic 98
C.4.3 Internal Registers 101
C.4.4 I/O and Counter 103
C.4.5 Front Panels 103
C.5 Maintenance Information 106
C.5.1 Introduction 106
C.5.2 Mechanical Structures 106
C.5.3 Circuit Diagrams 106

D Zukünftige Triggerboxen **122**

E Glossar **126**

Kapitel 1

Einleitung

Beim Deutschen Elektronen-Synchrotron *DESY* in Hamburg wird zur Zeit des Entstehens dieser Arbeit die Hadronen-Elektronen-Ringanlage *HERA* gebaut, ein Speicherring, in dem Protonen einer Energie von 820 GeV mit Elektronen von 30 GeV bei einer Schwerpunktsenergie von 314 GeV kollidieren werden. Für die Untersuchung der dabei stattfindenden Reaktionen sind zwei Großdetektoren vorgesehen, die momentan ebenfalls in der Aufbauphase sind.

Das Ziel der Experimente um den *HERA*-Beschleuniger ist die Untersuchung der tief unelastischen ep-Streuung. Sie beruht auf neutralen oder geladenen Strömen. also dem Austausch von $W^{\pm}$- oder γ/Z^0-Bosonen. Die Basisprozesse bei *HERA* sind demnach

$$\begin{aligned} ep &\mapsto \nu_e X \qquad \text{und} \\ ep &\mapsto eX, \end{aligned}$$

wobei X für die Fragmentation des Protons steht.

Die Analyse von *HERA*-Ereignissen erfordert im ersten Schritt eine Klassifizierung nach obigen Prozessen, was große Anforderungen an den Detektor darstellt: im erstgenannten Fall hinterläßt das Neutrino keinerlei Spuren und kann somit nur über Fehlbeträge in der Energiebilanz nachgewiesen werden; im zweiten Fall gilt es, das gestreute Elektron zu identifizieren, was immer dann schwierig ist, wenn es — bedingt durch die Art der Parton-Fragmentation — nicht als isoliertes Teilchen vorliegt.

Ein Detektor für *HERA* muß also, um eine exakte Rekonstruktion von Ereignissen zu ermöglichen, die Identifikation von Jets und isolierten Teilchen — insbesondere isolierten Elektronen in Jets — gewährleisten und

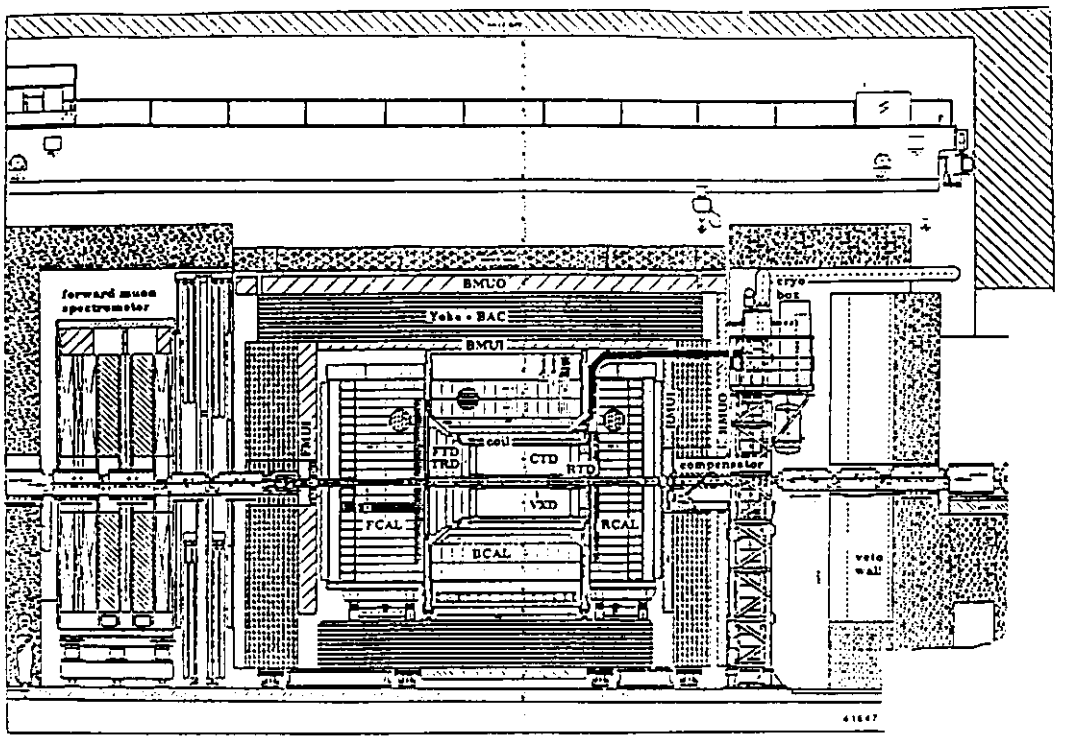

Abbildung 1.1: Schnitt durch den *ZEUS*-Detektor entlang der Strahlachse

über eine exzellente Energieauflösung verfügen. An diesen Anforderungen läßt sich bereits erkennen, daß für den Erfolg eines Experiments bei *HERA* die Kalorimetrie von entscheidender Bedeutung ist.

Die Abbildungen 1.1 und 1.2 zeigen den *ZEUS*-Detektor, einen der beiden für den Einsatz bei *HERA* vorgesehenen Detektoren. Vom Wechselwirkungspunkt auslaufende Teilchen und Jets durchlaufen nach Austritt aus einem Vertexdetektor (VXD[1]) zunächst ein Spurenkammersystem (CTD, FTD oder RTD), das in Vorwärtsrichtung noch durch einen Übergangsstrahlungsdetektor (TRD) ergänzt wird, bevor sie im Kalorimeter (BCAL, FCAL oder RCAL) gestoppt werden. Der ganze Aufbau ist in ein Eisenjoch (Yoke) gekapselt, das zur Ermittlung von Leckenergien aus dem Detektor ein Backing-Kalorimeter (BAC) enthält und auf Innen- und Außenseite mit Drahtkammern zum Nachweis von Myonen (BMUI, BMUO,...) umgeben ist. Zur Krümmung der Teilchenbahnen im Spurenkammersystem ist dieses samt Vertexdetektor von einer supraleitenden Spule (coil) umgeben.

Um *HERA*-Ereignisse mit bestmöglicher Energieauflösung messen zu können, erhält der Detektor ein Präzisionskalorimeter in Uran-Szintillator-Sandwich-Bauweise. Das Kalorimeter wird mit Hilfe von Photoelektronen-Vervielfachern ausgelesen, wobei die hohe *HERA*-Wechselwirkungsrate von

[1] Die verwendeten Abkürzungen entsprechen den in Abb. 1.1 und 1.2 angegebenen.

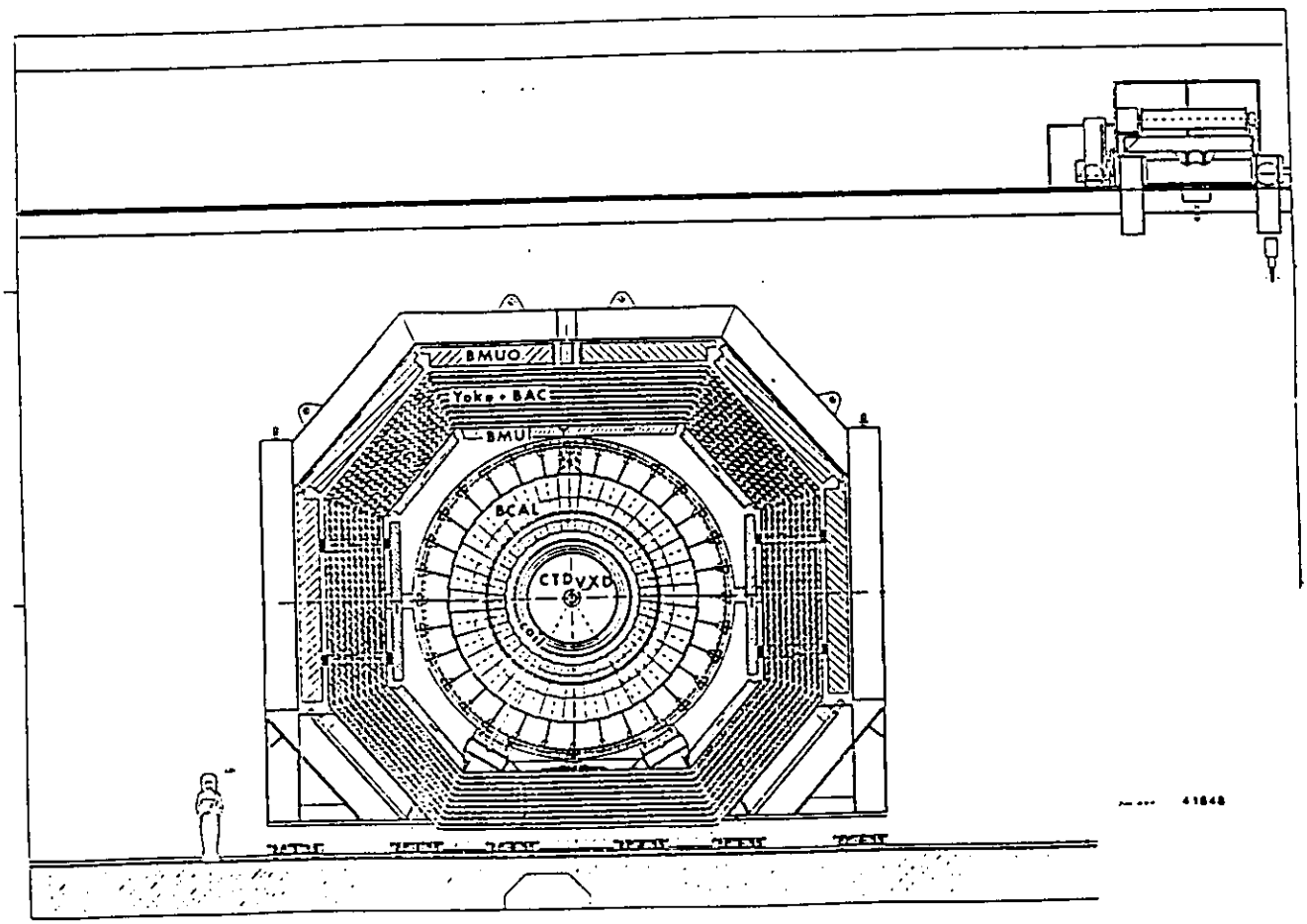

Abbildung 1.2: Schnitt durch den *ZEUS*-Detektor quer zur Strahlachse

nahezu 10.5 *MHz* es erforderlich macht, die Meßwerte bereits vor ihrer Digitalisierung zu filtern. Um den Filteralgorithmen etwa 5 μs Entscheidungszeit zur Verfügung stellen zu können, enthält das Auslesesystem eine Pipeline für analoge Signale.

Die Module des *ZEUS*-Kalorimeters werden vor ihrem Einbau in den Detektor zusammen mit ihrer Ausleseelektronik an einem Teststrahl des europäischen Forschungszentrums *CERN* in Genf kalibriert. Die Beschreibung dieses Testexperiments sowie die Darstellung dort gewonnener Resultate sind das Thema dieser Arbeit. Da der Teststrahl, an dem die Kalibrierung des Kalorimeters stattfindet, über ein Sekundärtarget gewonnen wird, treten die zu detektierenden Ereignisse nicht wie bei *HERA* in einem festen Rhythmus, sondern zeitlich unkorreliert auf. Die daher notwendige Synchronisation des Datenerfassungssystems mit den physikalischen Gegebenheiten sowie deren Realisierung in Form eines Triggermoduls bilden den Schwerpunkt bei der Beschreibung des Experiments.

Das Präzisionskalorimeter für den *ZEUS*-Detektor wird von Physikern und Technikern von über zwanzig Instituten aus acht Nationen unter Beteiligung des II. Instituts für Experimentalphysik der Universität Hamburg, an dem die vorliegende Arbeit durchgeführt wurde, gebaut. Die Inbetriebnahme des *ZEUS*-Detektors wird für Frühjahr 1991 erwartet.

Kapitel 2

Die Kalibrierung des ZEUS-Kalorimeters

2.1 Aufbau des *ZEUS*-Präzisionskalorimeters

Der *ZEUS*-Detektor wird mit einem Uran (DU) - Szintillator (Sci)[1]–Sampling-Kalorimeter ausgestattet, für dessen Entwicklung von der *ZEUS*-Kollaboration die im folgenden ihrer Priorität nach aufgelisteten Kriterien formuliert wurden [1]:

- vollständige Überdeckung des den Wechselwirkungspunkt umgebenden Raumwinkels mit Detektormaterial,
- Messung der in Hadronjets deponierten Energie mit einer Auflösung von $\sigma(E)/E = 35\%/\sqrt{E} \oplus 2\%$, E in GeV,
- Auflösung des Emissionswinkels von Jets von besser als $10\,mrad$ bei gleichzeitigem gutem Trennungsvermögen von Jets
- Separation zwischen Hadronen und Elektronen sowohl für isolierte Elektronen als auch für Elektronen in Jets.

[1] DU: Depleted Uranium (abgereichertes Uran), Sci: Szintillator. Im Falle des *ZEUS*-Kalorimeters enthält das DU 98.4% U238, 0.2% U235 und 1.4% Nb bei $\rho_{DU} = 18.5\,g\,cm^{-2}$.

Bei der Energieauflösung bedeutet "$\oplus$" eine quadratische Summe, die zwei Prozent des zweiten Terms stehen für systematische Fehler. Im ersten Term bedeutet die Skalierung der Auflösung mit $\sqrt{E}^{-1}$ die Verwendung eines Sampling-Kalorimeters. Sowohl durch Monte-Carlo Rechnungen [2] als auch experimentell [3,4,5,6] wurde gezeigt, daß die geforderte Auflösung bei Verwendung eines DU-Sci-Kalorimeters erreicht werden kann. Die Verwendung von Szintillatoren als Auslesemedium hat des weiteren den großen Vorteil, daß die zu verarbeitenden Signale mit etwa $50\,ns$ deutlich kürzer sind als die Wechselwirkungsabstände bei *HERA* ($96\,ns$). Die angestrebte Winkelauflösung läßt sich bei entsprechend feiner Segmentierung des Kalorimeters erreichen, und zur Verbesserung der Hadron-Elektron-Separation sind bei *ZEUS* in das Kalorimeter integrierte Halbleiterdioden vorgesehen.

Das *ZEUS*-Kalorimeter ist modular aufgebaut, wobei Modulgruppen wiederum zu Vorwärts-, Rückwärts- und Zentralkalorimeter, kurz FCAL, RCAL und BCAL[2] zusammengefaßt werden. Die einzelnen Module sind noch in sog. Türme unterteilt, die dann die kleinsten Einheiten des Kalorimeters darstellen. Türme sind in longitudinaler Richtung dreigeteilt (im RCAL lediglich zweigeteilt): vor zwei Sektionen, die nur von hadronischen Schauern erreicht werden (HAC1 und HAC2, im RCAL zu HAC zusammengefaßt), befindet sich ein Abschnitt, in dem alle elektromagnetischen Schauer gestoppt werden (EMC). Um der kleineren Ausdehnung elektromagnetischer Schauer gegenüber hadronischen gerecht zu werden, sind die EMCs ihrerseits transversal nochmals in mehrere Streifen unterteilt. Die Tiefe des Kalorimeters ist überall so gewählt, daß selbst die energiereichsten Jets in mehr als 90% aller Fälle noch immer über 95% ihrer Energie im Kalorimeter deponieren.

Ein Überblick des Kalorimeters ist in Abb. 2.1 gegeben. Man erkennt, daß seine Geometrie — abgesehen vom BCAL-EMC — nicht projektiv ist. Die FCAL- und RCAL-Module sind nahezu identisch aufgebaut und werden im folgenden noch näher beschrieben. Die BCAL-Module, die sich im wesentlichen in ihrer Form von denen des FCAL und RCAL unterscheiden, sollen hier nicht weiter betrachtet werden.

In Abb. 2.2 ist das größte der FCAL-Module dargestellt. Deutlich zu erkennen sind die $20 \times 20\,cm^2$ großen Türme, die im vorderen Abschnitt noch in vier je $5 \times 20\,cm^2$ große Streifen unterteilt sind. Die Unterteilung

[2]Forward, Rear und Barrel CALorimeter

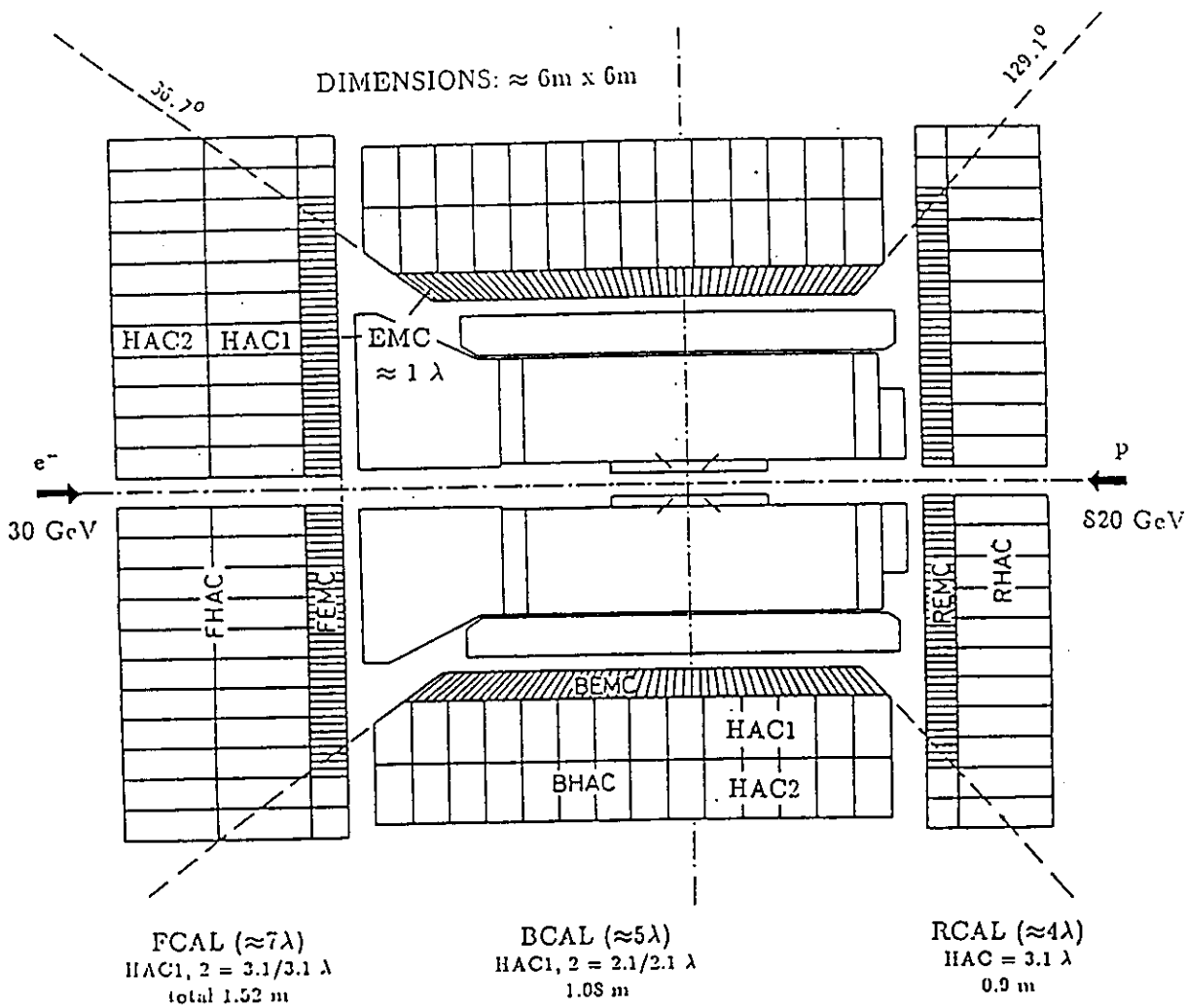

Abbildung 2.1: Komponenten des *ZEUS*-Präzisionskalorimeters

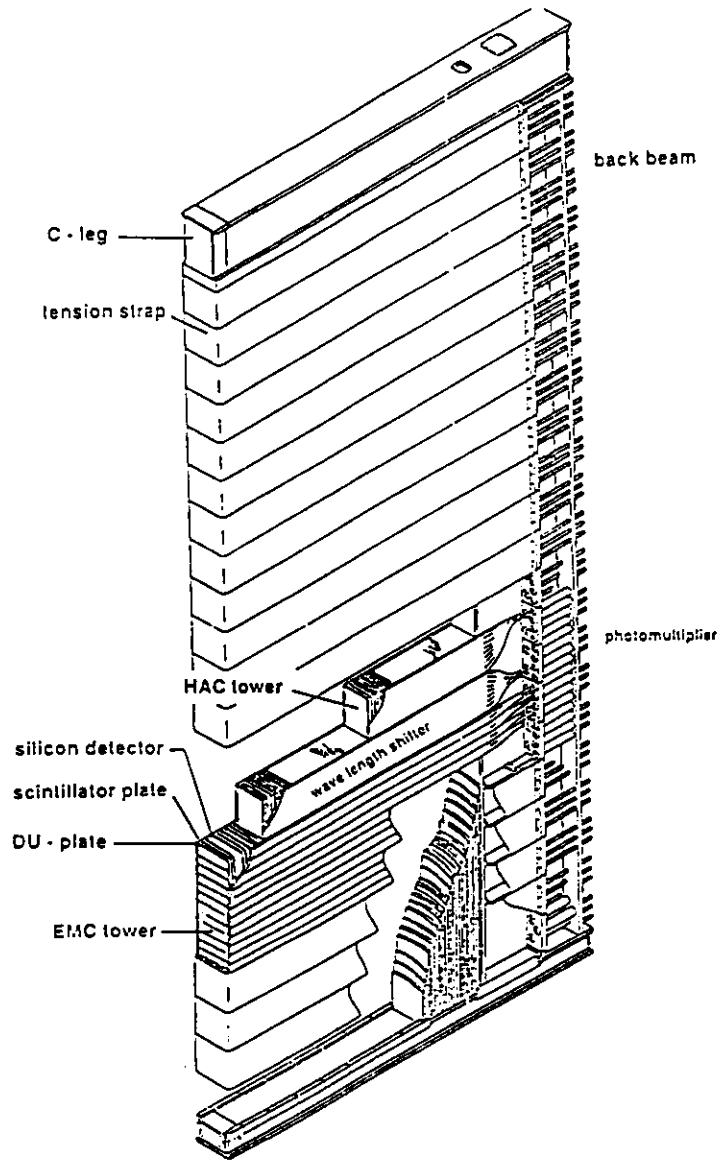

Abbildung 2.2: *ZEUS*-FCAL-Modul

bezieht sich hierbei lediglich auf das Detektormedium, d. h. die Szintillatorplatten, um eine genaue Ortsauflösung bei der Auslese zu ermöglichen. Die DU-Platten sind dagegen über die komplette Länge aus nur einem Stück gefertigt (im Falle des abgebildeten Moduls: $4.60\,m$). Um die Feuergefahr zu verringern, aber auch um eine sichere Handhabbarkeit der DU-Platten zu gewährleisten und gleichzeitig die Szintillatoren etwas gegen die allgegenwärtige Hintergrundstrahlung aus dem Uran abzuschirmen sind die Uranplatten mit einer dünnen Edelstahlfolie umwickelt.

Das im Szintillator erzeugte Licht wird über Wellenlängenschieber (WLS) am Rande des Moduls zu dessen Rückseite geführt. Dort wird es über Photomultiplier (PMTs) in elektrische Signale gewandelt, welche dann auf speziellen, auf der Modulrückseite befestigten "front-end"-Elektronik-Karten aufbereitet und zwischengespeichert werden, bevor sie dann in digitalisierter Form an einen Computer zur weiteren Verarbeitung gelangen. Auf der Abbildung sind die schmalen, außen liegenden Wellenlängenschieber des EMC und die weiter innen befindlichen und entsprechend breiteren und kürzeren HAC-WLS mit den dahinterliegenden PMTs zu erkennen. Die front-end–Elektronik-Karten sind nicht gezeichnet.

Die Module des RCAL gleichen, abgesehen von der Tiefe der Türme und der Segmentierung des EMC, denen des FCAL. Eine detaillierte Beschreibung des *ZEUS*-Kalorimeters kann z. B. in [1] gefunden werden.

2.2 Testprogramm für das *ZEUS*-Kalorimeter

In konventionellen Kalorimetern erzeugt die bei nuklearen Reaktionen zwischen einfallendem Teilchen und Kalorimetermaterial freiwerdende Bindungsenergie kein meßbares Signal. Da diese Größe — bedingt durch Schauerfluktuationen — zwischen verschiedenen Schauern stark variiert, trägt sie in nicht unerheblichem Maße zur Verschlechterung der Energieauflösung bei. Verwendet man dagegen Materialien wie Uran zur Kalorimetrie, so läßt sich ein Teil der verlorenen Bindungsenergie in Form langsamer Neutronen zurückgewinnen und kann somit detektiert werden. Bei DU-Sci-Kalorimetern hängt das von der Neutronenkomponente eines Schauers erzeugte Signal stark vom Sampling-Verhältnis, d. h. dem Verhältnis der Stärken des Absorber- und Detektormaterials, ab. Durch Veränderung die-

ser Größe kann das von Hadronenschauern gelieferte Signal beeinflußt werden, und bei einer geeigneten Wahl ist es somit erreichbar, daß Elektronen und Hadronen gleicher Energie im Mittel gleiche Signale erzeugen. Auf diese Weise wird die Empfindlichkeit des Kalorimeters gegenüber Schauerfluktuationen verringert, und es verkleinert sich der systematische Fehler bei der Energiemessung. Durch diesen "Kompensation" genannten Vorgang erreichen DU-Sci-Kalorimeter ihre einzigartige Auflösung.

Um die Dicken der Absorber- und Detektorplatten zu bestimmen und damit eine optimale Energieauflösung für das *ZEUS*- Kalorimeter zu erreichen, wurden zahlreiche Testexperimente durchgeführt [4,6]. Von dem endgültigen Vorwärts-Kalorimeter wurde schließlich ein kleiner Prototyp gebaut, der 1988/89 am CERN-PS und -SPS[3] untersucht wurde.

Zum Experimentierprogramm um den FCAL-Prototypen gehörten vorwiegend Messungen, die das allgemeine Verständnis des Kalorimeters betrafen. Im Brennpunkt des Interesses stand dabei natürlich die Energieauflösung, die in dem für den Einsatz relevanten Energiebereich bestimmt wurde und den bereits angeführten Erwartungen entsprach. Ebenfalls untersucht wurde das Ortsauflösungsvermögen sowie die Uniformität der Signalantwort des Kalorimeters über dessen Oberfläche und für verschiedene Einfallswinkel des Teststrahls. Bei letzterem führte das Einfügen einer $2\,mm$ starken Bleifolie zwischen die Module zu erheblichen Verbesserungen, da diese die verstärkte Erzeugung von Szintillationslicht in den Wellenlängenschiebern verringerte.

Über ein spezielles, eigens dafür entworfenes Lichtleitersystem wurden die Lichtausbeute und Linearität der PMTs gemessen, was Voraussetzung ist, um die Verfälschung eines durch das Kalorimeter erzeugten Signals im Auslesezweig zu bestimmen. Von entscheidender Bedeutung waren schließlich auch Messungen, mit denen die Kalibrierungsmöglichkeiten des Kalorimeters und deren Zuverlässigkeit geprüft wurden [5,7,8].

Im Sommer 1989 ist dann am CERN-SPS ein Teststand aufgebaut worden, auf dem die endgültigen, zum Einbau in den *ZEUS*-Detektor bereiten Module kalibriert werden können. Hier steht natürlich die Qualitätskontrolle der gefertigten Module im Mittelpunkt. Dazu gehören Homogenitätsprüfungen der Kalorimetertürme in longitudinaler Richtung, Einstellung und Stabilitätskontrolle der Hochspannung an den PMTs und Mes-

[3]PS: Protonen Synchrotron; SPS: Super Protonen Synchrotron. Beide Beschleuniger befinden sich am europäischen Kernforschungszentrun CERN in Genf.

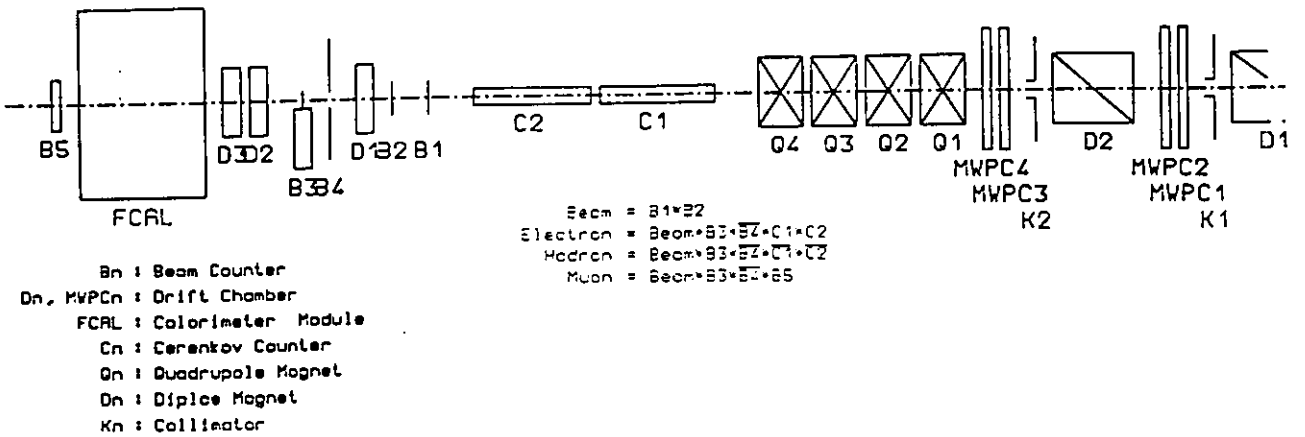

Abbildung 2.3: Das Kalibrierungsexperiment am CERN/SPS

sung des Uranrauschens (UNO[4]) und des Rauschens der Ausleseelektronik genauso wie die Bestimmung des Antwortverhaltens des Kalorimeters auf einfallende Elektronen oder Hadronen eines vorgegebenen Impulses. Ziel des Experiments ist es, für jedes FCAL- und RCAL-Modul einen Satz von Kalibrierungskonstanten zu erstellen, der bei der Ereignisauswertung bei *ZEUS* herangezogen werden kann und gleichzeitig eine Erkennung und Korrektur zeitlicher Veränderungen des Antwortverhaltens des Kalorimeters während seines Einsatzes erlaubt.

2.3 Das Experimentierprogramm am CERN-SPS

Einen Überblick über den Aufbau des Experiments zeigt Abb. 2.3: Die aus dem SPS extrahierten Teilchen werden auf ein Sekundärtarget gelenkt. Dort entstehende Streuteilchen werden durch zwei Dipol- (D1, D2) und vier Quadrupolmagnete (Q1-Q4) zu einem Strahl aus Teilchen definierten Impulses fokussiert. Über ein System aus zwei Kollimatoren (K1, K2) mit

[4]Uranium NOise: Uranrauschen; bezeichnet die allgegenwärtige Hintergrundstrahlung aus dem Uran.

dahinter befindlichen Driftkammern (MWPC1-MWPC4) kann die Impulsbreite Δp des Strahles festgelegt und kontrolliert werden. Der so erzeugte Teilchenstrahl wird auf das zu kalibrierende Kalorimetermodul geleitet. Ein System von Schwellwert-Čerenkov- (C1, C2) und Szintillator-Zählern (B1-B5) liefert Signale, die von einer eigens dafür konstruierten Triggerelektronik ausgewertet und zur Synchronisation mit dem Datenerfassungssystem verwendet werden; dadurch kann die Antwort des Kalorimeters auf das Einfallen eines Teilchens definierter Sorte und definierten Impulses aufgezeichnet werden: Die Schwellwert-Čerenkov-Zähler ermöglichen die Identifikation von Elektronen innerhalb des Strahles, über zwei Szintillator-Zähler (B3, B4) kann der Einfallsort der Teilchen in das Kalorimeter gewählt werden, und ein hinter dem Experimentaufbau befindliche Szintillator-Zähler (B5) erlaubt schließlich die Identifikation von Muonen.

Zur Verdeutlichung der verschiedenen am SPS verwendeten Kalibrierungsmethoden werde zunächst der Weg eines Signals im Kalorimeter vom Erzeugungsprozeß durch ein einfallendes Teilchen bis hin zur Erfassung durch ein digitales Rechnersystem betrachtet (vgl. Abb. 2.4). Die Erzeugung eines Lichtpulses durch ein in das Kalorimeter einfallendes Teilchen sowie dessen Transport im Kalorimeter und Wandlung in ein elektrisches Signal über einen PMT sind bereits im vorigen Abschnitt beschrieben worden. Das vom PMT kommende Signal wird einer front-end–Karte zugeführt, wo es zunächst über ein Widerstandsnetzwerk auf verschiedene Meßzweige verteilt wird. Der Triggerzweig wird für die Kalibrierungsmessungen nicht benötigt und hier daher nicht weiter beschrieben. Der mit UNO bezeichnete Meßzweig wird zur Bestimmung des Uran-Rauschens verwendet und besteht im wesentlichen aus einem Integrator; seine Notwendigkeit wird weiter unten erläutert werden. Zur Erfassung eines Signals durch einen digitalen Rechner wird dieses über den Signalmeßzweig geführt, wo es einen Signalformer (shaper) mit Verstärker durchläuft, um dann in einem Analogspeicher (SCDL[5])festgehalten zu werden. In solchen Speicherbausteinen können bis zu 58 Signale gleichzeitig gespeichert werden. Beim Speichern eines Signals wird dieses achtmal im Abstand von je $96\,ns$ abgetastet, und die dabei ermittelten Pulshöhen werden in analoger Form (kapazitiv) in der SCDL festgehalten. Da ein ankommender PMT-Puls jedoch nur etwa $50\,ns$ breit ist, muß dieser vor der Speicherung durch den Signalformer zeitlich

[5]Switched Capacitor Delay Line, vgl. [9].

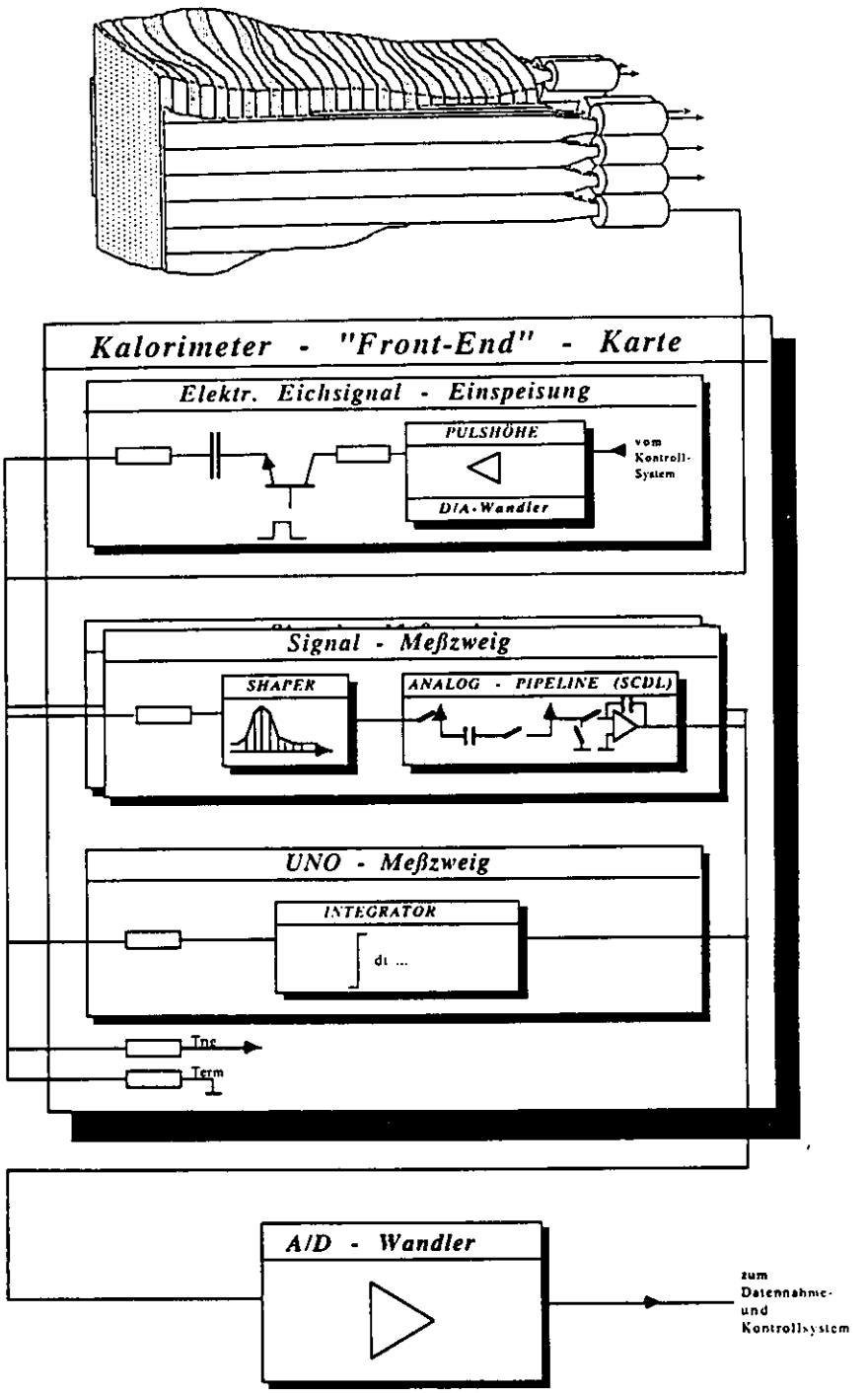

Abbildung 2.4: Signalverlauf bei der Kalorimeterauslese

so gedehnt werden, daß die Abtastung möglich wird. Die Zwischenspeicherung an dieser Stelle ist für die Kalibrierung nicht zwingend notwendig, wird jedoch beim endgültigen Einsatz des Kalorimeters benötigt, um Zeit für Triggeralgorithmen zu gewinnen.

Auf den Analogspeicher kann selektiv zugegriffen werden: ist also ein Ereignis von einem Trigger als verwendbar erkannt worden, so können die zu dem entsprechenden Signal gehörenden Pulshöhen aus der SCDL in einen Pufferspeicher geladen werden, von wo aus sie dann über einen Multiplexer einem Analog-Digital-Wandler (ADC) zugeführt werden. Das digitalisierte Signal wird dann von einem Computer zur weiteren Verarbeitung aufgenommen.

Um den Dynamikbereich der Ausleseelektronik zu vergrößern, ist der Signalmeßzweig für jeden PMT doppelt vorhanden (s. Abb. 2.4), wobei sich die Verstärkungsfaktoren im Signalformer unterscheiden. Eine front-end-Karte enthält die Elektronik, die zur Auslese der Signale von sechs PMTs benötigt wird. Die Auslese eines Kalorimeterturmes erfordert damit zwei front-end- Karten (das gilt nicht für die äußeren Sektionen, die kein EMC enthalten).

Verfolgt man den Signalverlauf, so erkennt man einige mögliche Quellen etwaiger Signalstörungen und -veränderungen. Sie zu erfassen, um dann die Signale dementsprechend zu korrigieren, ist eine wesentliche Aufgabe bei der Kalibrierung und bestimmt zu weiten Teilen das Experimentierprogramm am Kalibrierungsstand. Der folgende Überblick über die am Teststand durchzuführenden Messungen wird dies verdeutlichen:

Das eigentliche Motiv zur Kalibrierung der Kalorimetermodule liegt natürlich darin, ihr Antwortverhalten im Teststrahl zu ermitteln. Von besonderem Interesse ist dabei eine Überprüfung der linearen Abhängigkeit der Signalantwort von der Energie der einfallenden Teilchen und seiner Homogenität über die volle Breite des Moduls. Des weiteren ist zu prüfen, ob die Energieauflösung im Rahmen der Anforderung ist. Um EMC und HAC1 kalibrieren zu können, ist der Beschuß des Moduls sowohl mit Elektronen als auch mit Hadronen notwendig. Zur Kalibrierung der HAC2-Sektion mit Hadronen reichen die am SPS-Teststrahl zur Verfügung stehenden Energien nicht aus, so daß hierfür noch zusätzlich der Beschuß mit Muonen erforderlich ist.

Zur Ermittlung der Anzahl der vom PMT bei Zuführung einer definierten Lichtmenge erzeugten Photoelektronen sowie zur Linearitäts- und

Stabilitätsprüfung der PMTs sind die Kalorimetermodule mit dem bereits in Abschn. 2.2 erwähnten Lichtleitersystem ausgestattet. Dabei handelt es sich um ein Glasfibersystem, das die Zuführung von LED- und Laserlicht bekannter Wellenlänge, Intensität und Pulsform an die PMTs ermöglicht. Auf diese Weise werden die Parameter eines jeden im Kalorimeter eingebauten PMTs bestimmt.

Die Signalverstärkung im PMT ist stark von der Hochspannung abhängig, so daß diese vor Beginn der Messungen festgelegt werden muß. Man bedient sich dabei des Uran-Rauschens: da dieses im gesamten Kalorimeter gleich ist, kann man die Hochspannungen an den PMTs so wählen, daß bei Messung des UNO alle PMTs im Mittel in etwa gleichhohe Signale erzeugen und auf diese Weise Exemplarstreuungen unter den PMTs entgegenwirken[6]. Da das mittlere Uranrauschen keinen Schwankungen unterliegt, hat man so außerdem die Möglichkeit, zeitliche Instabilitäten der PMTs nachzuweisen: wiederholt man in regelmäßigem Abstand die UNO-Messungen, so ergibt sich aus dem Verhältnis der Signalhöhen unterschiedlicher Messungen die Veränderung der Signalverstärkung des PMT.

Neben dem UNO liefert auch die front-end–Elektronik einen Rauschbeitrag (Pedestal). Man kann ihn messen, indem man die Hochspannung an den PMTs so weit herunterregelt, daß diese kein Signal mehr erzeugen können, und dann das Kalorimeter ausliest. Zur Ermittlung von Signalverzerrungen auf den front-end–Karten besteht die Möglichkeit der "Ladungsinjektion" (Q_{inj}): Ein auf einen definierten Wert aufgeladener Kondensator wird über einen Widerstand binnen $20\,ns$ entladen und das entstehende Signal in den Auslesezweig eingespeist. Das so erzeugte Signal ähnelt dem eines PMTs. Sämtliche Daten müssen, bevor sie ausgewertet werden, um die Rauschanteile und Verzerrungen der front-end–Elektronik korrigiert werden. Bei *ZEUS* wird dies später einmal noch vor der Datenspeicherung mit Hilfe eines digitalen Signalprozessors, der Zugang zu den Kalibrierungskonstanten hat, geschehen.

Die letzte Stelle im Auslesezweig, an der ein Signal noch verändert werden könnte, ist bei der Digitalisierung. Die Kalibrierung der ADCs geschieht über Präzisions-Referenzspannungen, hat aber mit der Kalorimeterkalibrierung im eigentlichen Sinne nichts weiter zu tun.

[6] Um Signale verschiedener Kalorimetersektionen absolut miteinander vergleichen zu können, müssen diese im Verhältnis zu ihrem UNO-Anteil ausgedrückt werden, da sie nur in dieser Form unabhängig von Parametern des Auslesezweiges sind (vgl. Kap. 5.3).

2.4 Das Datennahmesystem und die Experimentkontrolle

Ist ein Kalorimetermodul dem Teststrahl ausgesetzt, so wird es mit hoher Frequenz von Teilchen getroffen, d.h. es werden mit ebenso hoher Frequenz Daten erzeugt, die es zu erfassen gilt[7]. Zeichnete man nun alle anfallenden Daten auf, so wäre die Datenmenge sehr schnell so groß, daß sie weder vollständig analysiert noch gespeichert werden könnte. Daher ist es notwendig, vor der Aufzeichnung die Daten auf ihre Verwendbarkeit hin zu untersuchen: will man beispielsweise eine EMC-Sektion kalibrieren, so ist man nicht an hadronischen Schauern interessiert, und es empfiehlt sich, entsprechende Daten gar nicht erst zu erfassen.

Ein weiteres Problem ist die Synchronisation der front-end–Elektronik mit dem Experiment: da die anfallenden Daten auf der front-end–Karte abgetastet werden sollen, muß dort der Zeitpunkt des Eintretens eines Ereignisses bekannt sein, was jedoch insofern recht schwierig ist, als die Strahlteilchen das Kalorimeter völlig unkorreliert treffen. Man ist also gezwungen, diese Information — z. B. durch die Verwendung spezieller Triggerzähler — gesondert zu erzeugen und bei der Datenaufnahme entsprechend einzusetzen.

Die Aufgabe des Datenaufnahmesystems liegt nun darin, bei Eintreten eines verwendbaren Ereignisses die dazugehörenden Daten aufzunehmen, in ein wohldefiniertes Format zu bringen und zusammen mit Informationen über die gewählten Parameter des Experiments zu speichern.

Um das beschriebene Experiment in seiner vollen Komplexität kontrollieren und steuern zu können, ist man selbstverständlich auf die Verwendung eines Computersystems mit einer umfangreichen Steuersoftware angewiesen. Einem Benutzer präsentiert sich das System in Form einer überschaubaren Oberfläche, über die die Steuerparameter des Experiments manipuliert und sein Status abgefragt werden können. Die einzelnen zum Experiment gehörenden Komponenten, ihre Steuerung und ihr Zusammenspiel bleiben dem reinen Anwender darunter verborgen.

Die Synchronisation des Steuer- und Datenaufnahmesystems sowie der Hardware verschiedener Komponenten mit dem Experiment wird von einer

[7]Da der Teststrahl über ein Sekundärtarget vom CERN-SPS gewonnen wird, treffen die Teilchen zeitlich völlig unkorreliert auf das Kalorimeter.

eigens dafür entwickelten Triggerbox geleistet. Ihrem Entwurf und Bau sowie ihrem Einsatz in dem Experiment sind die nächsten Kapitel gewidmet.

Kapitel 3

Die Triggerbox für die FCAL-Kalibrierung

3.1 Aufgaben der Triggerbox

Bei der Beschreibung des Teststandes am CERN-SPS und des dortigen Kontroll- und Datenerfassungssystems im vorigen Kapitel wurde bereits darauf hingewiesen, daß Experiment und Datennahmesystem einer sorgfältigen Synchronisation bedürfen. Der Zweck dieser Synchronisation ist es, eine korrekte Datenerfassung zu ermöglichen und die Aufzeichnung unwesentlicher Daten zu vermeiden.

Das Kalorimeter wird zeitlich unkorreliert von Teilchen getroffen, wodurch dort Daten erzeugt werden. Das Datenerfassungssystem soll diese Daten aufzeichnen, aber außerhalb der Zeiten, in denen Teilchen das Kalorimeter treffen, die Datennahme sperren. Weiterhin muß gewährleistet werden, daß im Falle des Eintreffen eines neuen Teilchens während einer noch andauernden Auslese diese ordnungsgemäß beendet wird[1]. Die Information über den Zeitpunkt des Einfallens eines Teilchens in das Kalorimeter kann über die bereits in Kap. 2.3 beschriebenen Szintillator-Zähler *B1* bis *B4* gewonnen werden. Die Triggerbox hat diese Information so zu verarbeiten, daß das Eintreffen eines jeden Teilchens an das Datennahmesystem gemeldet wird. Da diese Meldung in Echtzeit zu verarbeiten ist, sollte sie in Form eines Systeminterrupts erfolgen.

[1] Dies betrifft insbesondere die Steuerung der Auslesepipeline

Eine weitere Möglichkeit zur Einschränkung der Menge der aufgezeichneten Daten liegt in der Datenfilterung vor der Aufzeichnung: Während einer Messung ist man üblicherweise nur an speziellen Eigenschaften des Kalorimeters interessiert, und daher zumeist auch nur an durch Teilchen einer speziellen Sorte erzeugten Daten. Die Triggerbox sollte also in der Lage sein, nach Angabe einer gewünschten Teilchensorte lediglich beim Einfallen von Teilchen dieser Sorte ins Kalorimeter eine Meldung an das Datennahmesystem abzusetzen. Für die Identifikation der das Kalorimeter treffenden Teilchen stehen die in Kap. 2.3 beschriebenen Čerenkov- und Szintillator-Zähler *C1*, *C2* und *B5* zur Verfügung.

Die bislang beschriebenen Aufgaben der Triggerbox geben die Erfordernisse für Messungen mit dem Kalorimeter im Teststrahl wieder. Die Beschreibung des Kalibrierungs- und Testprogramms (Kap. 2.3) zeigt jedoch, daß auch etliche Messungen außerhalb des Strahles stattfinden werden — so z. B. die Messung des Uranrauschens, die Messungen mit dem LED- und Lasersystem oder aber die Kalibrierung der Ausleseelektronik. Bei diesen Messungen variieren die an der Datenerfassung beteiligten Komponenten je nach Art der Messung und sind dementsprechend zu- oder abzuschalten. Die Komponenten lassen sich über jeweils eine gesonderte Signalleitung schalten, und es obliegt der Triggerbox, die vom Kontrollsystem stammende Information über die Art der Messung entsprechend auf diese Leitungen zu verteilen.

Findet eine Messung mit dem Kalorimeter außerhalb des Teststrahles statt, so liefern die Szintillator- und Čerenkov-Zähler *B1* bis *B5* und *C1*, *C2* keine Signale, so daß deren Verarbeitung und die damit verbundene Interrupt-Generierung ausbleiben. Das Datennahmesystem beginnt die Datenaufzeichnung jedoch nur nach Empfang eines Systeminterrupts, so daß die Triggerbox eine Möglichkeit zur Erzeugung von Interrupts durch das Steuersystem anbieten muß.

Zusammenfassend lassen sich die Aufgaben der Triggerbox durch die Begriffe Synchronisation, Datenfilterung und Koordination beschreiben. Technisch läßt sich dies auf die Verarbeitung und Erzeugung einiger weniger spezieller Signale reduzieren, die im folgenden, ihrer Funktion nach gruppiert, kurz dargestellt werden. Kursiv gedruckte Signalnamen beziehen sich dabei auf Abb. 3.1, die die Stellung der Triggerbox innerhalb des Teststandes am SPS sowie den mit ihr verbundenen Signal- und Datenfluß verdeutlicht.

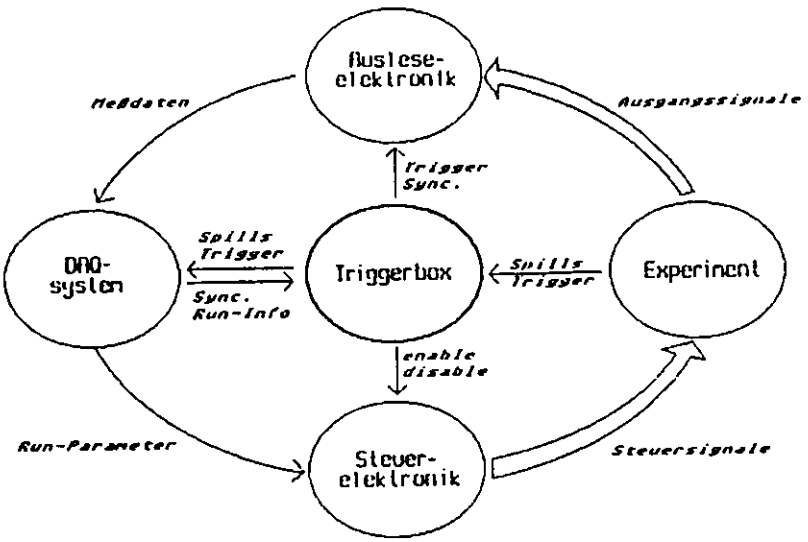

Abbildung 3.1: Stellung der Triggerbox innerhalb des Kalibrierungsaufbaus; die einfachen Pfeile zeigen den für das DAQ-System sichtbaren Daten- und Kontrollfluß.

- Signale zur Teilchenidentifikation *(Trigger)*: Hierunter fallen die Signale der Triggerzähler *B1* bis *B5*, der Čerenkov-Zähler *C1* und *C2* und ihre bereits in Abb. 2.3 dargestellten logischen Verknüpfungen.

- *Enable*- und *Disable*-Signale: Sie dienen dem Zu- oder Abschalten der Komponenten des Datennahmesystems abhängig von der Art der durchgeführten Messung. Die benötigte Information *(Run-Info)* wird vor Beginn einer Messung vom Kontrollsystem an die Triggerbox gesendet, die dann die Verteilung der Information übernimmt. Des weiteren liefert das SPS keinen kontinuierlichen Teststrahl, sondern nur "spills" von etwa $2\,s$ Dauer mit einer Periode von $14.4\,s$. Durch Verarbeitung der von der SPS-Maschine bereitgestellten spill-start- und -stop-Signale *(Spills)* wird die Datennahme an die spill-Struktur angepaßt[2].

- Computererzeugte Trigger *(Sync.)*: Über sie kann durch das Kontrollsystem der für die Datennahme benötigte Systeminterrupt erzeugt werden. Gleichzeitig werden diese Pulse als Ausgangssignale zur Synchronisation externer Hardware (z. B. Zünden eines Lasers,...)

[2]Es ist z. B. sinnlos, außerhalb eines spills Strahldaten nehmen zu wollen; dagegen empfiehlt es sich, das UNO nur zwischen den spills zu messen.

bereitgestellt.

3.2 Planung und Entwurf

3.2.1 Anforderungen an den Entwurf

Aufgrund ihrer zentralen Position innerhalb des Teststandes hat die Triggerbox zu fast allen Komponenten des Systems Schnittstellen. Das bedeutet aber wiederum, daß sich jede Veränderung an einer der Komponenten auch sofort an der Triggerbox bemerkbar macht. Um einen störungsfreien Einsatz der Triggerbox zu garantieren und zu verhindern, daß sie bei jeder Modifikation des Experiments ebenfalls modifiziert werden muß, sind bei ihrem Entwurf einige Kriterien zu berücksichtigen:

- **Unabhängigkeit**, **Einfachheit** und **Transparenz**: Die Triggerbox sollte möglichst wenig spezielle Eigenschaften ihrer Systemumgebung ausnutzen; Zugriffe sollten sich auf einfache Schreib-/Leseoperationen beschränken und die Registerebene ein Abbild des Aufbaus der Triggerbox sein, lediglich für Echtzeit-Funktionen sind Interrupts zu verwenden.

- **Testbarkeit**, **Zuverlässigkeit** und **Wartbarkeit** : Jede Komponente der Triggerbox für sich muß auf Funktionalität prüfbar sein. Um Störungen zu vermeiden, sind Bauteile großzügig zu dimensionieren und genügend Raum für Kühlung vorzusehen. Der Aufbau ist so zu gestalten, daß im Störfall Reparaturen am Einsatzort möglich sind.

- **Kompatibilität**: Die Pegel von Ein- und Ausgangssignalen sollten, soweit dies a-priori bekannt ist, mit ihrer Umgebung kompatibel sein, so daß der Einsatz der Triggerbox möglichst keine externen Pegelkonverter erfordert.

- **Flexibilität**: Unvorhergesehene kleinere Änderungen am Experimentaufbau müssen, ohne die Triggerbox zu modifizieren, bewerkstelligt werden können.

- **Geschwindigkeit**: Die Propagation der Triggersignale ist zeitkritisch. Daher sollte beim Entwurf dieses Teils der Box der Schwerpunkt auf minimale Signaldurchlaufzeiten gelegt werden.
- **Dokumentation**: Der Entwurf, Bau und Test der Triggerbox sowie sämtliche Modifikationen während ihres Einsatzes sind sorgfältig zu dokumentieren.

Der beste Weg, diesen Entwurfskriterien gerecht zu werden, liegt in der Gliederung der Triggerbox in voneinander unabhängige Komponenten, die dann überwiegend aus Standardschaltungen unter der Verwendung von Standardbausteinen aufgebaut werden.

3.2.2 Funktionsgruppen der Triggerbox

Das Herzstück der Triggerbox, die Verarbeitung von Signalen, die das Eintreten eines Ereignisses melden und die daraus hervorgehende Erzeugung eines globalen Triggerpulses, wird in dem Blockschaltbild Abb. 3.2 dargestellt. Darin fällt sofort das große, 16 *bit* breite Oder-Gatter auf: wird einer seiner Eingänge aktiv, so erzeugt es über seinen Ausgang und das dort angeschlossene Monoflop einen globalen Triggerpuls. Gleichzeitig setzt es das "Inhibit-Flip-Flop", das durch Rückkopplung seines Ausgangs das Aktivwerden eines weiteren Eingangs des Oder-Gatters und damit die Erzeugung weiterer globaler Triggerpulse verhindert. Erst, wenn der erzeugte Ausgangspuls vom Kontrollsystem vollständig verarbeitet ist, kann dieses das Inhibit-Flip-Flop löschen und somit erneute Triggerpulse ermöglichen. Um die entscheidende Position des Oder-Gatters innerhalb der Triggerbox zu unterstreichen, wird es üblicherweise "Master-Or" genannt.

Die Logik auf der Eingangsseite des Master-Or ermöglicht es, jeden seiner Kanäle einzeln für externe Signale zu sperren oder transparent zu machen. Außerdem kann über das Trigger-Status-Register jederzeit der Zustand aller Eingänge beim Auftreten des letzten globalen Triggerpulses überprüft werden: bei Erzeugung eines globalen Triggers wird diese Information automatisch festgehalten, so daß die Quelle des Triggers auch im Nachhinein noch ermittelt werden kann. Da der Zustand aller Kanäle festgehalten wird, können dabei auch etwaige Koinzidenzen des auslösenden mit anderen Ereignissen festgestellt werden — auch mit Ereignissen auf Kanälen, deren Signale nicht zum Master-Or propagiert wurden.

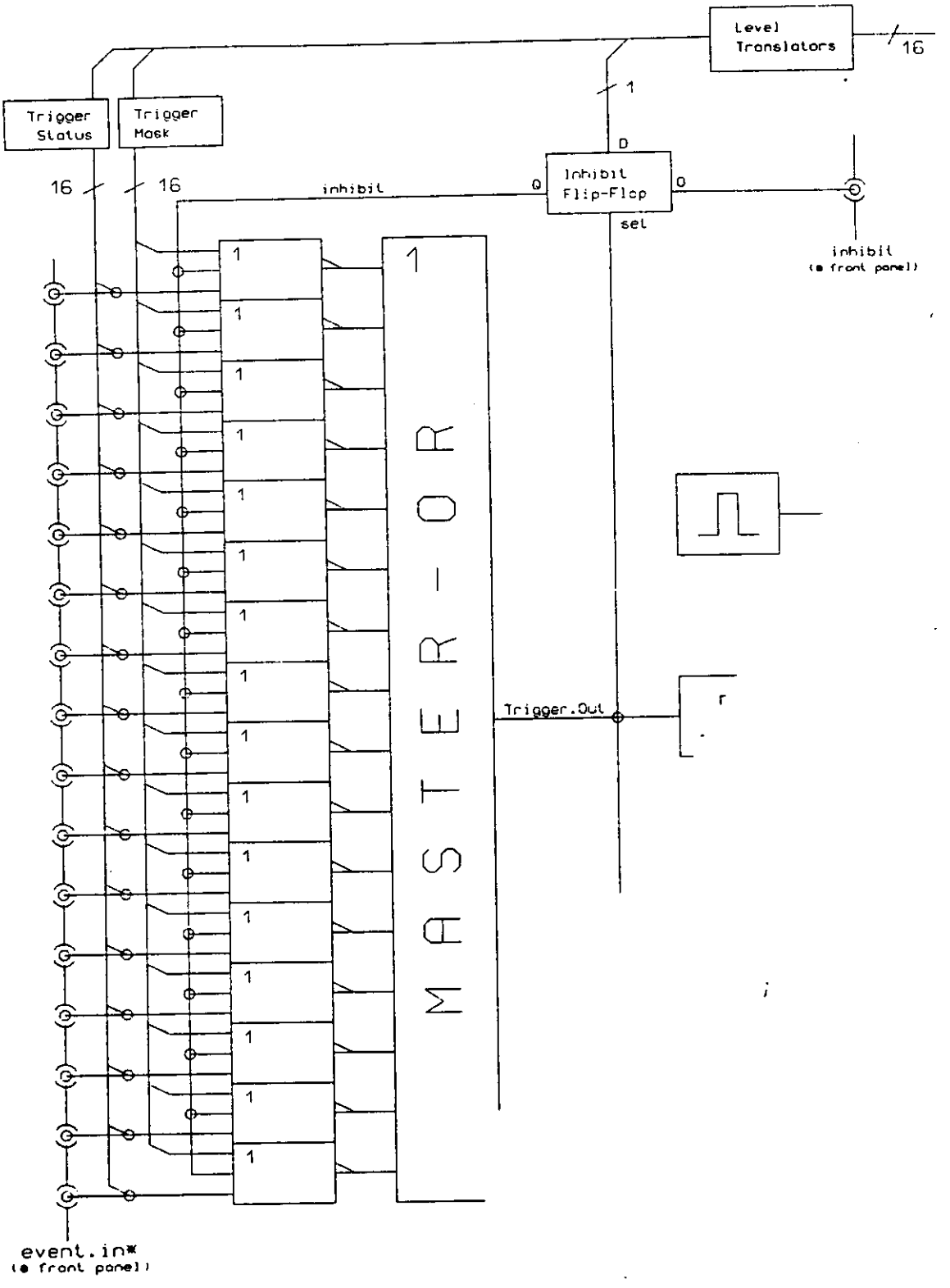

Abbildung 3.2: Blockschaltbild 1: Master-Or und Umgebung

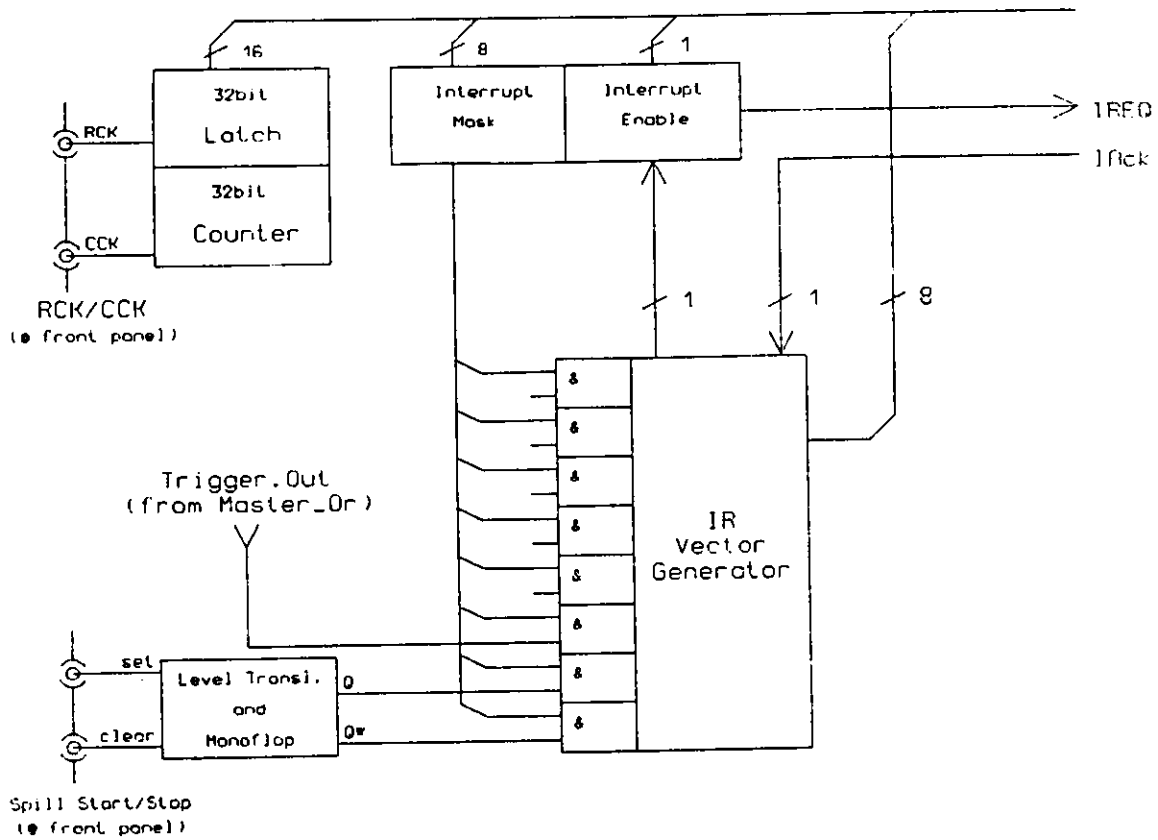

Abbildung 3.3: Blockschaltbild 2: Interrupter und Zähler

Der Computer-Trigger kann vom Kontrollsystem ausgelöst werden. Er findet immer dann Verwendung, wenn, bedingt durch die Art der Messung, keine externen Triggersignale erzeugt werden (beispielsweise bei Messung des Uranrauschens). Um ihn für das Datennahmesystem gegenüber externen Triggern gleichzustellen, kann sein Ausgang mit einem der Eingänge des Master-Or Verbunden werden.

Das zweite Blockschaltbild (Abb. 3.3) zeigt, wie der globale Trigger und die die spill-Struktur beschreibenden Signale dem Kontrollsystem übermittelt werden. Das zentrale Element ist hierbei ein 8-fach priorisierter Interrupter. Ähnlich dem Master-Or sind auch seine Eingänge einzeln (über das Interrupt-Mask-Register) oder global (Interrupt-Enable) zu sperren. Ebenfalls zu erkennen ist in dem Diagramm ein 32 *bit* breiter Zähler, der an einem externen Eingang *(CCK)* auftretende Pulse zählt und seinen Zählerstand auf Anforderung *(RCK)* in ein vom Datenerfassungssystem lesbares Regi-

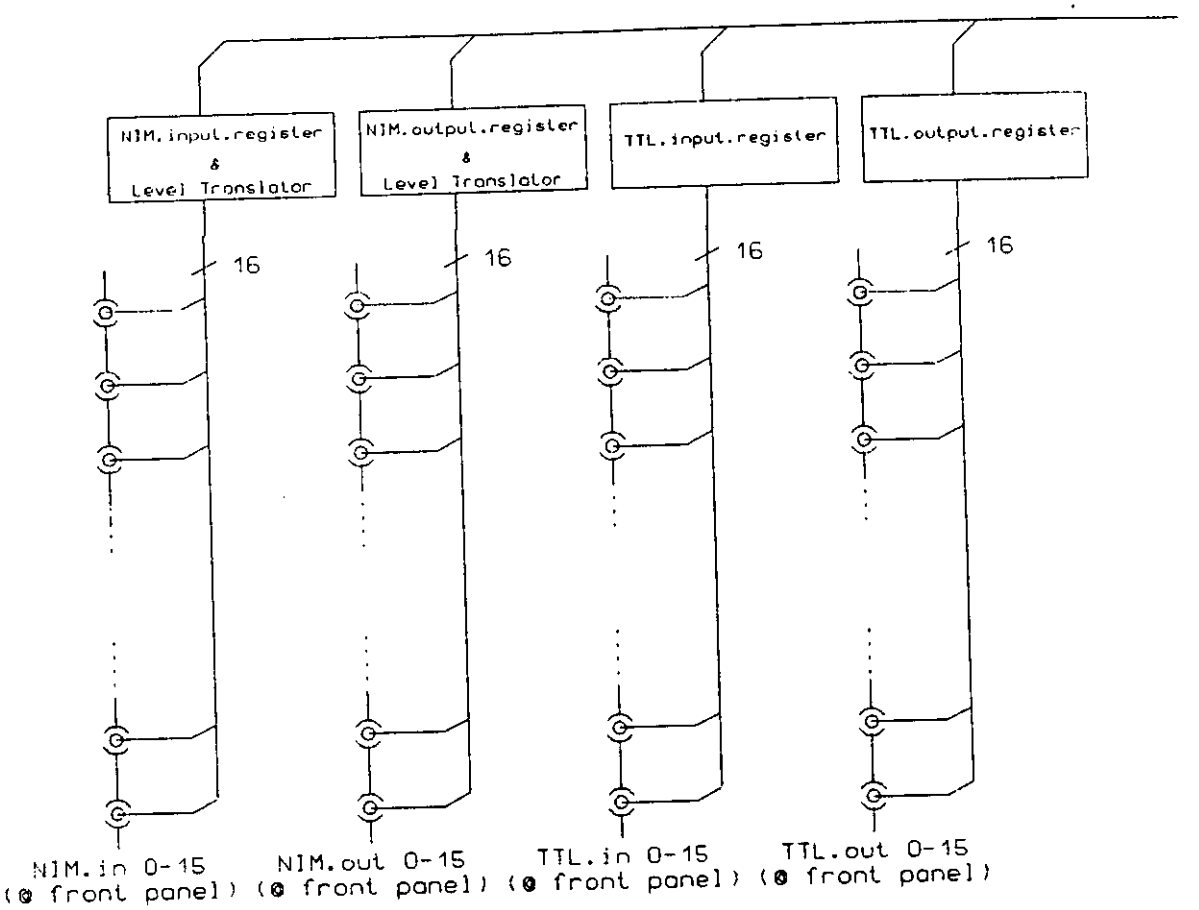

Abbildung 3.4: Blockschaltbild 3: NIM- und TTL-kompatible I/O-Ports

ster kopiert. Für ihn sind verschiedene Anwendungen denkbar: so kann er z. B. die Zahl der an einem Eingang auftretenden Triggerpulse zählen, während dieser über das Inhibit-Flip- Flop gesperrt ist, oder aber er mißt die Zeit, die die Triggerbox während eines spills gesperrt ist. Über diese Zahlen lassen sich Aussagen über die Effizienz des Datennahmesystems gewinnen. Um zu verhindern, daß der Zähler währed der Abarbeitung eines Zählimpulses ausgelesen wird, ist ihm ein Register vorgelagert, das die Daten für die Auslese puffert.

Das letzte Blockschaltbild (Abb. 3.4) zeigt schließlich die in der Triggerbox befindlichen I/O-Baugruppen. Dabei handelt es sich im wesentlichen um einfache Register mit Pegelkonvertern und Ausgangstreibern.

3.2.3 Funktionale Beschreibung der Triggerbox

Die Schnittstellen der Triggerbox zu ihrer Umgebung sind auf Experimentseite ihre Ein- und Ausgänge, auf seiten des Kontrollsystems dagegen ihre Register. Ein Anwender kann die Triggerbox daher auch ohne ausführliche Kenntnisse über ihren Aufbau einsetzen, wenn ihm nur das Zusammenspiel der Ein- und Ausgangssignale untereinander und mit den Registerinhalten bekannt ist. Man nennt diese Abstraktionsebene der Schaltkreisbeschreibung Register-Transfer-Ebene, oder kurz RT-Level [11].

Für die RT-Level-Beschreibung der Triggerbox wird die folgende Terminologie verwendet: Register- und Signalleitungsnamen werden *kursiv* dargestellt. Dabei steht *name* für eine Signalleitung, wohingegen $\langle name \rangle$ den Inhalt eines Registers bedeutet. $name_i$ bedeutet entweder das i-te bit eines Registers oder aber die i-te Leitung eines Busses. Pegel von Signal- und Steuerleitungen sowie Registerinhalte werden durch das Gleichheitszeichen einander zugewiesen[3], wobei $P[n]$ das Auftreten eines Pulses auf Signalleitung n bedeutet. Die verwendeten Namen entsprechen denen aus Abb. 3.2-3.4. Steuersignale, die vom Kontrollsystem erzeugt werden und in den Blockschaltbildern daher nicht zu erkennen sind, werden in Blockschrift und *GENEIGT* dargestellt und im Anhang näher erläutert.

Für das Master-Or mit Umgebung gelten die Beziehungen

$$
\begin{aligned}
P[Trigger.Out] &= \overline{inhibit} \wedge \left(\bigvee_{i=0}^{15} (Trigger.Mask_i \wedge event.in_i) \right), \\
P[Computer.Trigger] &= FIRE.COMPUTER.TRIGGER, \\
inhibit &= \overline{CLEAR.INHIBIT} \wedge \\
&\quad \wedge \ (P[Trigger.Out] \vee \\
&\qquad \vee SET.INHIBIT \vee \\
&\qquad \vee inhibit), \\
Trigger.Status_i &= \left(event.in_i \wedge \overline{P[inhibit]} \right) \vee \\
&\quad \vee \left(Trigger.Status_i \wedge P[inhibit] \right).
\end{aligned}
$$

Die erste Gleichung besagt, daß ein globaler Trigger genau dann erzeugt wird, wenn die Box nicht gesperrt ist und ein für externe Signale transparen-

[3]Die Gleichbehandlung von Registern mit Signal- und Steuerleitungen ist an dieser Stelle zulässig, da sie alle digitale Zustände haben.

ter Eingang des Master-Or aktiv wird. Nach der zweiten Gleichung kann ein Computer-Trigger vom Steuersystem durch Setzen einer Steuerleitung erzeugt werden. Das Inhibit-Flip-Flop kann nur über eine Kontrolleitung gelöscht werden und wird beim Auftreten einer Flanke am globalen Triggerausgang oder durch ein Steuersignal vom Computer gesetzt. Liegt keines dieser Signale an, so hält es seinen Zustand. Die letzte Gleichung zeigt schließlich, daß das Trigger-Status-Register ein Abbild der Event-Eingänge gibt, solange die Triggerbox nicht gesperrt ist, und sonst seinen aktuellen Zustand speichert.

Ein Interrupt Request wird von der Triggerbox generiert, wenn einer der Eingänge des Interrupters aktiv wird. Dabei bedeutet Aktivieren eines Eingangs das Auftreten einer Flanke an besagtem Eingang bei Gesetztsein des korrespondierenden bits im Interrupt-Mask-Register:

$$\begin{aligned} IREQ = {} & Interrupt.Enable \wedge \\ \wedge \; & \big((Int.Mask_8 \wedge P[Spill.Stop]) \vee \\ & \vee (Int.Mask_7 \wedge P[Spill.Start]) \vee \\ & \vee (Int.Mask_6 \wedge P[Trigger.Out]) \big) \end{aligned}$$

Der 32 *bit*-Zähler zählt am CCK-Eingang auftretende Pulse:

$$\langle counter \rangle \;=\; \big((\langle counter \rangle + 1) \wedge P[CCK] \big) \;\vee\; \big(\langle counter \rangle \wedge \overline{P[CCK]} \big)$$

Zur Pufferung des Zählerstandes und Verhinderung von Konflikten beim Lesen des Zählers ist ihm ein Latch vorgeschaltet:

$$\langle latch \rangle \;=\; \big(\langle counter \rangle \wedge P[RCK] \big) \vee \big(\langle latch \rangle \wedge \overline{P[RCK]} \big)$$

Die Ein- und Ausgabe-Ports geben schließlich ein Abbild des Zustandes der mit ihnen verbundenen Signalleitungen. Zu beachten ist lediglich die Richtung des Datenflusses:

$$\begin{aligned} NIM.input.register_i &= NIM.in_i \\ NIM.out_i &= NIM.output.register_i \\ TTL.input.register_i &= TTL.in_i \\ TTL.out_i &= TTL.output.register_i \end{aligned}$$

Familie	Typ	Betriebs-spannung	Verlust-leistung pro Gatter	Signal-laufzeit pro Gatter	Laufzeit-Leistungs-Produkt	
TTL						
standard	7400	$5\,V$	$10\,mW$	$10\,ns$	$100\,pJ$	
LP Schottky	74LS00	$5\,V$	$2\,mW$	$10\,ns$	$20\,pJ$	
Schottky	74S00	$5\,V$	$19\,mW$	$3\,ns$	$57\,pJ$	
advanced LS	74ALS00	$5\,V$	$1\,mW$	$4\,ns$	$4\,pJ$	
fast	74F00	$5\,V$	$5\,mW$	$4\,ns$	$20\,pJ$	
advanced S	74AS00	$5\,V$	$22\,mW$	$1.5\,ns$	$33\,pJ$	
ECL						
standard	10100	$-5.2\,V$	$35\,mW$	$2\,ns$	$70\,pJ$	
	10200	$-5.2\,V$	$35\,mW$	$1.5\,ns$	$53\,pJ$	
high speed	10H100	$-5.2\,V$	$70\,mW$	$1\,ns$	$70\,pJ$	
	100100	$-5.2\,V$	$50\,mW$	$0.75\,ns$	$38\,pJ$	
CMOS						
metal gate	4000	$3-18\,V$	$0.3-3\,\mu W/kHz$	$90-30\,ns$	$0.03-0.09$	pJ/kHz

Tabelle 3.1: Elektrische Parameter der gängigsten Halbleiterfamilien

3.3 Aufbau der Triggerbox

3.3.1 Familien integrierter Schaltkreise

In der Digitaltechnik sind verschiedene Familien integrierter Schaltkreise bekannt. Da ihre entsprechenden Bausteine erhebliche Differenzen in so entscheidenden Parametern wie Signaldurchlaufverzögerung und Verlustleistung aufweisen, ist für jede Schaltung die zugrunde liegende Schaltkreisfamilie sorgfältig auszuwählen.

Eine Beschreibung von handelsüblichen Schaltkreisfamilien anhand ihrer Grundschaltungen und technologischen Besonderheiten befindet sich im Anhang. An dieser Stelle soll Tab. 3.1 genügen, die einen Überblick über die gängigsten Halbleiterfamilien und ihre elektrischen Parameter gibt.

Welche Schlüsse lassen sich daraus für den Aufbau der Triggerbox ziehen? Die TTL-Schaltkreisfamilien sind diejenigen mit dem größten Bauteil-angebot und nach wie vor am weitesten verbreitet. Ihre Pegel sind kompatibel zu allen Rechnersystemen, und außerdem sind sie im Hinblick auf Störanfälligkeit und notwendige Zusatzbeschaltung (wie z. B. Abschluß-

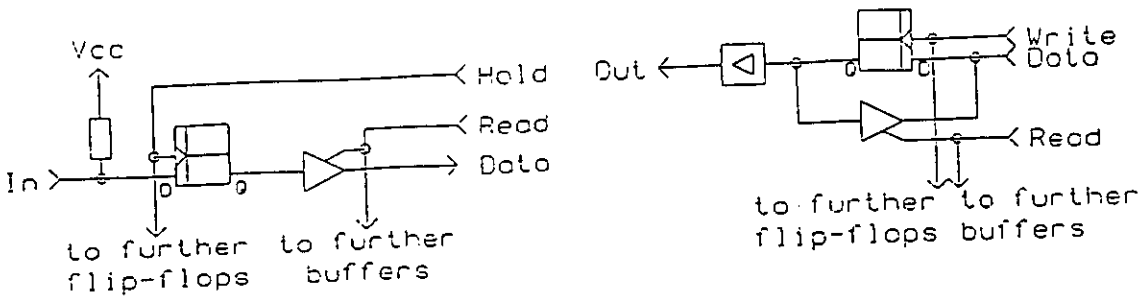

Abbildung 3.5: Input- und Output-Port-bit

widerstände, Ausgangstreiber,...) am robustesten. Daher empfiehlt es sich, soweit wie möglich auf TTL-Bausteine zurückzugreifen.

Im Falle der I/O-Baugruppen ist dies problemlos möglich, und auch für den Interrupter und seine Umgebung ist wegen der Kompatibilität der Signalpegel zum VMEbus die Verwendung von TTL-Bausteinen empfehlenswert. Kritisch wird es jedoch im Falle des Master-Or und seiner umgebenden Logik: sein wesentliches Entwurfskriterium ist die Minimierung der Signaldurchlaufzeit. Daher wird hier auf ECL-Technik zurückgegriffen.

3.3.2 Die einzelnen Baugruppen

In diesem Abschnitt sollen die den einzelnen Baugruppen zugrunde liegenden Schaltungen kurz dargestellt werden. Die detaillierten Schaltpläne der Triggerbox finden sich im Anhang.

Zunächst einmal werden die I/O-Baugruppen betrachtet, da die hier verwendeten Schaltungen in ähnlicher Form auch in allen anderen Funktionseinheiten auftreten. Bei der Vorstellung des Entwurfs der Triggerbox wurde bereits darauf hingewiesen, daß den Kern der I/O-Gruppen einfache Register in Verbindung mit einer Eingangsstufe oder Ausgangstreibern bilden, d.h. sog. Ports. Die Triggerbox verfügt über je zwei 16 *bit* breite Eingabe- und Ausgabe-Ports, wovon je einer NIM- und einer TTL-kompatibel ist.

In Abb. 3.5 ist jeweils ein bit eines Eingabe- und eines Ausgabeports dargestellt. Im Falle des Eingabeports wird der Eingang mit einer exter-

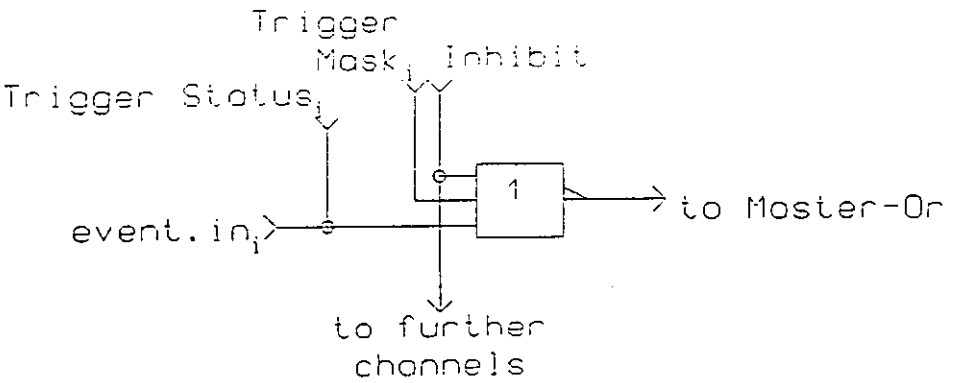

Abbildung 3.6: Eingangstore des Master-Or

nen Signalquelle verbunden, und bei Aktivierung der *Hold*-Leitung wird der momentane Zustand der Eingangsleitung an den Flip-Flop-Ausgang Q weitergeleitet. Dort wird er dann bis zur erneuten Aktivierung der Leitung gespeichert. Der ausgangsseitig angeschlossene Rechner kann nun seinerseits durch Aktivieren der *Read*-Leitung signalisieren, daß er den Wert des Eingangspegels lesen möchte. In diesem Falle wird der Leitungstreiber transparent und ermöglicht somit die Weiterleitung des Signals von Q zum Rechner. Ist man nicht an der Speicherung des Eingangszustands, sondern an dessen Beobachtung interessiert, so ist das Flip-Flop zu entfernen. Dies ist bei den Eingabe-Ports der Triggerbox geschehen.

Im Falle des Ausgabe-Ports kann der Rechner durch Aktivieren der *Write*-Leitung erreichen, daß der an D zuvor angelegte Signalpegel an den Ausgang weitergeführt wird. Soll dagegen der Zustand des Ausgangs überprüft werden, so wird nach Aktivierung der *Read*-Leitung der Leitungstreiber wiederum transparent und leitet die an Q anliegende Information zurück zum Rechner.

Das Master-Or mit seiner Umgebung ist bereits im Zusammenhang mit den Blockschaltplänen näher dargestellt worden. Von besonderem Interesse sind hier seine Eingangstorschaltungen (Abb. 3.6). Jeder Eingang des Master-Or ist mit einem NOR-Gatter mit drei Eingängen versehen, das sich jedoch wegen der negativen Logik der Eingangssignale wie ein logisches UND verhält. An je einen seiner Eingänge wird das Signal einer externen Quelle geführt, die beiden anderen werden mit dem Ausgang des Inhibit-Flip-Flops und einem bit des Trigger-Mask-Registers verbunden

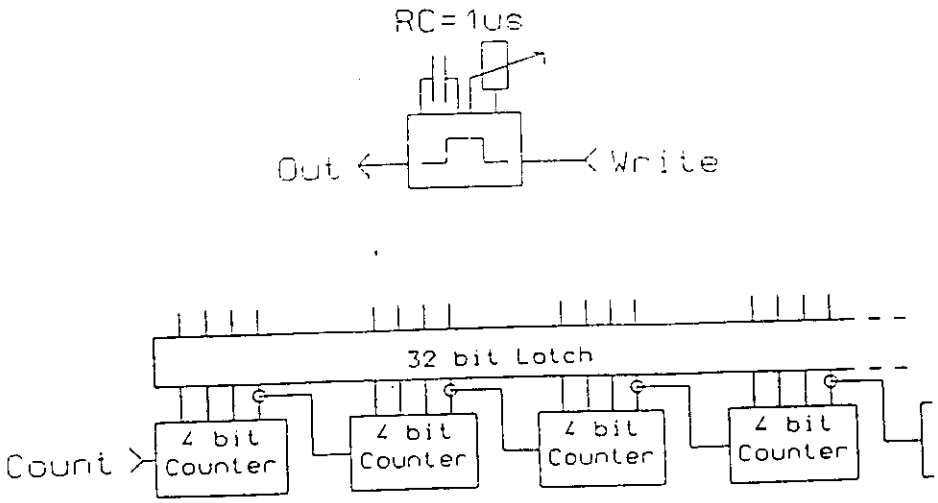

Abbildung 3.7: Computer-Trigger und asynchroner Zähler

(das Trigger-Mask-Register ist — abgesehen von den fehlenden Ausgangstreibern — identisch mit einem Ausgangs-Port). Sind Inhibit-Flip-Flop und das dem Eingang zugeordnete bit des Trigger-Mask-Registers inaktiv, so erscheint am Master-Or der Zustand des *event.in*-Eingangs; ist jedoch eines der beiden aktiv, so ist der Eingang des Master-Or stets inaktiv. Im ersten Fall ist das Tor also transparent, im zweiten gesperrt.

Ebenfalls mit den *event.in*-Eingängen verbunden ist das Trigger-Status-Register. Es ist entsprechend den oben beschriebenen Eingabe-Ports aufgebaut, wobei hier jedoch der Zustandsspeicher realisiert ist, wodurch bei jedem auftretenden globalen Triggerpuls der Zustand aller Eingangsleitungen festgehalten wird und daher das Kontrollprogramm durch Lesen des Registers auch im nachhinein noch dessen Auslöser ermitteln kann. Das *Hold*-Signal wird aus dem Ausgangssignal des Inhibit-Flip-Flops gewonnen.

Der auf dem Blockschaltbild zu sehende "one-shot"–Computer-Trigger ist ein einfaches Monoflop, das durch ein von einem Rechner kommendes *Write*-Signal angestoßen wird (Abb. 3.7). Um ihn einem vom Master-Or erzeugten Trigger gleichzustellen, ist sein Ausgang mit einem Eingang des Master-Or zu verbinden.

Die aufwendigste Einheit der Triggerbox ist der achtfach priorisierte Interrupter. Er muß die ankommenden Signale möglichst zügig entschlüsseln und sich beim Generieren der Unterbrechungsanforderungen genau an das Protokoll des VMEbus halten. Eine Beschreibung seiner Schaltung würde den Rahmen dieser Arbeit bei weitem übersteigen, kann jedoch in [10] ge-

funden werden.

Das Interrupt-Mask-Register ist gleich dem Trigger-Mask-Register aufgebaut, und bei dem 32 *bit*-Zähler handelt es sich um einen einfachen asynchronen Zähler für Frequenzen unter 25 *MHz* (Abb. 3.7), dessen Ausgänge über ein dem Status-Register gleichendes latch geführt werden.

3.3.3 Mechanischer Aufbau

Die Triggerbox ist in zwei Module unterteilt: Das eine ist ein doppeltbreites VME-Modul und trägt den Interrupter, den Zähler, den Computer-Trigger und die TTL-kompatiblen I/O-Ports [10], bei dem anderen handelt es sich um ein NIM-Modul dreifacher Breite, das das Master-Or mit Trigger-Masken- und -Status-Register trägt sowie die NIM-kompatiblen Ein-/Ausgabeports zusammen mit einigen Pegelwandlern. Die Module werden über ein etwa 3 *m* langes 64-pol. Flachbandkabel miteinander verbunden.

Als Basis des VME-Moduls wird ein Prototyping-Modul verwendet, das neben etwa 200 cm^2 Lochrasterfläche bereits ein VME-Interface nach IEEE 1014, Rev. C1 enthält. Eine kurze Beschreibung des Moduls befindet sich im Anhang. Das NIM-Modul baut auf einer Grundplatte auf, die vollständig mit IC-Sockeln versehen ist, die bereits eine der ECL-Familie entsprechende Spannungsversorgung haben.

Um einen Austausch defekter Schaltkreise nicht von vornherein zu behindern, werden alle verwendeten Bauteile gesockelt. Die Schaltung selbst wird in wire-wrap Technik aufgebaut, so daß sie — zumindest in der Entwicklungsphase — noch in kleinem Rahmen modifiziert werden kann.

3.4 Die Testphase

Ein Ausfall der Triggerbox während des Experimentierbetriebs hätte fatale Konsequenzen: nicht nur, daß das Experiment ohne die Triggerbox nicht fortgesetzt werden könnte, auch eine Reparatur der Box wäre, wenn diese erst einmal eingesetzt ist, äußerst schwierig und zeitaufwendig. Zum einen behindert die Vielzahl der mit der Triggerbox verbundenen Signalleitungen den Zugang zur Box in ihrem Überrahmen, zum anderen fürt die Mehrheit der Leitungen zeitkritische Signale, die teilweise auf Bruchteile von Nano-

sekunden genau aufeinander abgestimmt sind; da Reparaturen an der Box immer mit einer geringen Veränderung ihres Zeitverhaltens verbunden sind, müßten derartige Leitungen sämtlichst neu vermessen und verlegt werden.

Auf diese Art entstünden durch einen Triggerboxausfall Komplikationen und Verzögerungen im Experiment; sie würden schließlich in der Notwendigkeit, den Teststand länger als geplant aufrecht zu erhalten oder aber das Experimentierprogramm zu straffen, enden und damit enorme Zusatzkosten verursachen. Der weitestmögliche Ausschluß von Störungen im Einsatz muß daher der wichtigste Aspekt der Triggerbox sein. Die Testphase ist somit einer der Schwerpunkte bei der Erstellung der Triggerbox.

Ein sorgfältiger Test der Triggerbox, der sowohl Aussagen über ihre Korrektheit als auch über ihre Belastbarkeit ermöglicht, geht in mehreren Schritten vonstatten:

- Bevor in irgendeiner Form das Verhalten der Triggerbox untersucht werden kann, muß die Korrektheit ihres Aufbaus getestet werden. Mittelpunkt ist hierbei die Frage, ob der Entwurf korrekt in eine funktionsfähige elektrische Schaltung umgesetzt wurde. Um sie zu klären, bedarf es eines Vergleichs der im Entwurf erwähnten und in der Schaltung lokalisierten Bauelemente auf Vollständigkeit sowie einer Überprüfung der Verdrahtung der Elemente untereinander. Letzteres geschieht, bevor die Bauelemente eingesetzt sind, in Form eines Tests auf Unterbrechung eines gezogenen Drahtes und einer Kurzschlußprüfung zwischen benachbarten Drähten. Die korrekte Polung der Versorgungsspannung an allen Bauelementen sollte in einem gesonderten Test nochmals bestätigt werden.

- Danach wird zunächst in einem statischen Test das Verhalten der Bauelemente und -gruppen untersucht. Hierfür werden die Schaltkreise mit ihrer Zusatzbeschaltung jeweils funktionsgruppenweise eingesetzt und ihre Antworten auf extern anliegende Pegel auf Korrektheit überprüft. Zu diesem Test gehört bereits eine Überprüfung, ob alle in der Triggerbox vorhandenen Register schreib- und lesbar sind. Arbeitet die Schaltung soweit einwandfrei, so kann bei Bereitstellen der entsprechenden Pegel an den Steuerleitungen die Transparenz der Ein- und Ausgabeports untersucht werden: die in die Ausgabe-Register geschriebenen Werte müssen an den Ausgängen erkennbar

sein, und andersherum der Zustand der Eingänge aus den Eingabe-Registern rekonstruierbar. Besonderes Augenmerk ist hierbei auf die Dämpfung und Verzerrung ankommender Signale zu werfen.

- Arbeitet die Triggerbox statisch einwandfrei, so ist ihr dynamisches Verhalten zu testen. Grundlage hierfür bildet die im vorigen Abschnitt gegebene RT-Level-Beschreibung: die dort auf den rechten Seiten der Gleichungen angegebenen Zustände von Signalleitungen sind der Reihe nach einzustellen, die daraus resultierenden Zustände der Triggerbox dann mit den linken Seiten der entsprechenden Gleichung zu vergleichen. Nur, wenn alle Zustände der RT-Level-Beschreibung verifiziert wurden, kann eine Funktion der Triggerbox garantiert werden.

- Der letzte Schritt der Testphase ist schließlich die Systemintegration. Dabei handelt es sich im wesentlichen um einen Test des Zusammenspiels aller am Experiment beteiligten Komponenten. Sind diese zu Testzwecken nicht verfügbar — wie z. B. die Signale der Strahl- und Čerenkovzähler — so sind sie sorgfältig zu simulieren. Nur, wenn alle Komponenten zusammen eingesetzt werden, kann die Korrektheit des Daten- und Kontrollflusses im Experiment überprüft werden. Wichtig ist in dieser Phase ein Dauertest des gesamten Systems, sowohl zur Feststellung dessen Belastbarkeit, als auch als Grundlage einer Optimierung des Systems.

Ein Stand-Alone-Testprogramm, das den statischen und dynamischen Test der Triggerbox ermöglicht und auch als "debugging-tool" wertvolle Dienste leistet, wird im Anhang vorgestellt. Es ist so gestaltet, daß es in Verbindung mit der auf dem Teststand vorhandenen Hardware einwandfrei funktioniert, aber auch ohne große Probleme auf andere Systeme übertragen werden kann.

Kapitel 4

Das Datennahme- und Kontrollsystem

Im vorigen Kapitel wurde die Triggerbox vor allem von ihrer technischen Seite her beschrieben. Bei der gewählten Betrachtungsweise wurde die Triggerbox mehr als ein "stand-alone"-Projekt als vor dem Hintergrund des Experiments gesehen, und der Schwerpunkt der Darstellung lag auf einer Beschreibung ihres internen Aufbaus. Im hier folgenden wird nun einmal kurz das Datennahme- und Kontrollsystem des Experiments vorgestellt. Ein besonderes Augenmerk richtet sich dabei natürlich wieder auf die Triggerbox, insbesondere auf Art und Abwicklung des Daten- und Kontrollflusses von und zu der Box. Die natürliche Schnittstelle der Triggerbox zur Systemsoftware ist dabei das schon in Abschn. 3.2.2 beschriebene RT-Level.

4.1 Daten- und Informationsfluß im System

In Abb. 4.1 sind die wesentlichen an der Datennahme und Experimentkontrolle beteiligten Prozesse und ihre Verbindungen untereinander dargestellt. Die Graphik ist in Anlehnung an die Datenfluß-Diagramme der *SASD*[1]-Methode erstellt worden, d. h. Kreise stehen für Prozesse, Kästen

[1] Structured Analysis and Structured Design; eine Abstraktionsmethode in der Softwareentwicklung, mit der nach dem Prinzip der schrittweisen Verfeinerung aus der Beschreibung einer Systemkonfiguration eine Darstellung der am System beteiligten Prozesse, ihrer Daten- und Kontrollverbindungen untereinander sowie ihrer Anbindung an

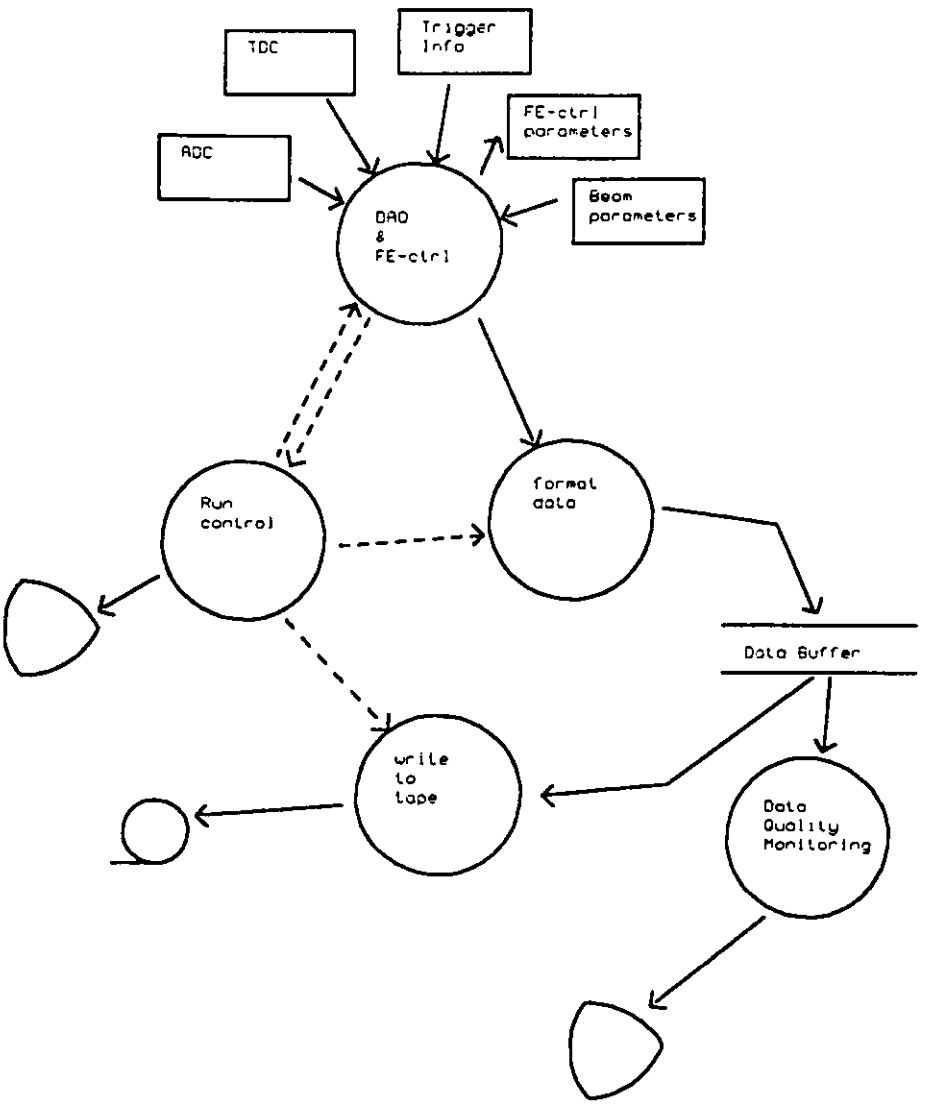

Abbildung 4.1: Vereinfachte Darstellung des Datenerfassungs- und Kontrollsystems

für externe Hardware und Doppellinien für Speicherbereiche. Der Datenfluß zwischen den Komponenten wird durch solide Pfeile, der Kontrollfluß durch gestrichelte dargestellt.

An dem gezeigten System fällt sofort auf, daß Experimentkontrolle und Datennahme von zwei unabhängigen Prozessen durchgeführt werden! An der Schnittstelle zum Experiment befindet sich der "Data AcQuistion and Front-End control"-Prozeß, der sowohl die vom Kontrollprozeß kommenden Informationen an die Komponenten des Experiments verteilt und die dort anfallenden Statusinformationen zurückleitet, als auch die im Experiment anfallenden Daten an einen Prozeß übergibt, der sie zur weiteren Verarbeitung vorbereitet und zwischenspeichert. Es ist dies also diejenige Stelle im System, an der die vom Experiment kommenden Signale erfaßt und klassifiziert werden, und es ist auch die Stelle, an der die Kommunikation mit der Triggerbox abgewickelt wird. Über den mit "Run Control" bezeichneten Prozeß laufen alle Information, die mit der Kontrolle des Experiments und Synchronisation seiner Komponenten zusammenhängen. Über das gezeigte Terminal besteht die Möglichkeit, den Experimentstatus einzusehen und seinen Ablauf zu manipulieren. Der "format data" benannte Prozeß sammelt die vom Experiment einlaufenden Daten ein und stellt sie, nachdem sie geordnet wurden, in einem Pufferspeicher zur Verfügung. Von dort aus können sie über den "Data Qualitiy Monitoring"-Prozeß an einem Terminal begutachtet werden, bevor sie schließlich der "write to tape"-Prozeß auf Magnetband schreibt.

Eine detaillierte Beschreibung des gesamten Systems würde den Rahmen dieser Arbeit bei weitem überschreiten. Daher soll im folgenden lediglich die Verteilung der Komponenten auf verschiedene Hardware und die Lokalisierung des Triggerprozesses im System noch weiter erläutert werden.

---

ihre Umgebung gewonnen werden kann. Bei Erstellung dieser Systembeschreibung wird gleichzeitig Buch geführt über verwendete Datenformate und Kommunikationsprotokolle sowie mögliche Zustände einzelner Prozesse, so daß die Fehleranfälligkeit während der Systementwicklungsphase drastisch verringert wird.

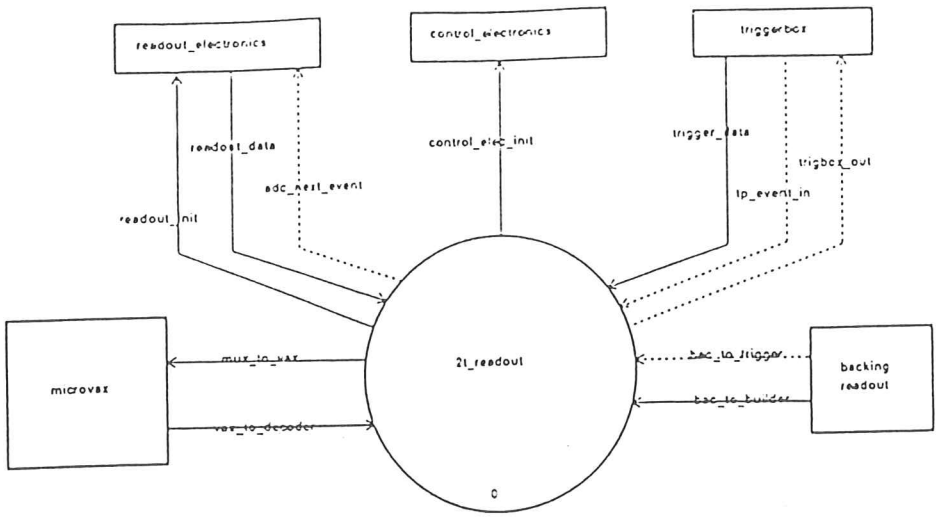

Abbildung 4.2: Transputer-Readout-Prozeß mit Umgebung (entspricht "DAQ & FE-ctrl" aus dem Übersichtsdiagramm

4.2 Stellung und Struktur des Triggerprozesses

Es ist bereits an Abb. 4.2 zu erkennen, daß der "DAQ & FE-ctrl"-Prozeß eng an den Aufbau des Experiments gekoppelt ist, wohingegen die anderen Prozesse eher mit der Aufbereitung der vom Experiment einlaufenden Informationen in Verbindung stehen. An diesem logischen Übergang befindet sich auch in der Hardware ein Schnitt: während Experimentkontrolle und Datenverarbeitung auf einer VAX abgewickelt werden, ist der "DAQ & FE-ctrl"-Prozeß auf einem eigens für die Datenerfassung beim *ZEUS*-Experiment entwickelten Transputer-Board[2] plaziert.

Führt man den Abstraktionsprozeß aus Abb. 4.2 weiter, so hat man die dort gezeigten Prozesse in Subprozesse aufzulösen. Die ersten Schritte dieser Verfeinerung sind für den "DAQ & FE-ctrl"-Prozeß in den Abbildungen 4.2 bis 4.4 gezeigt, wobei die verwendete Symbolik mit der aus dem

[2]Transputer, die ihnen eigene Programmiersprache occam und das oben erwähnte "2Tp-Modul" werden im Anhang im Zusammenhang mit dem Testprogramm für die Triggerbox kurz vorgestellt.

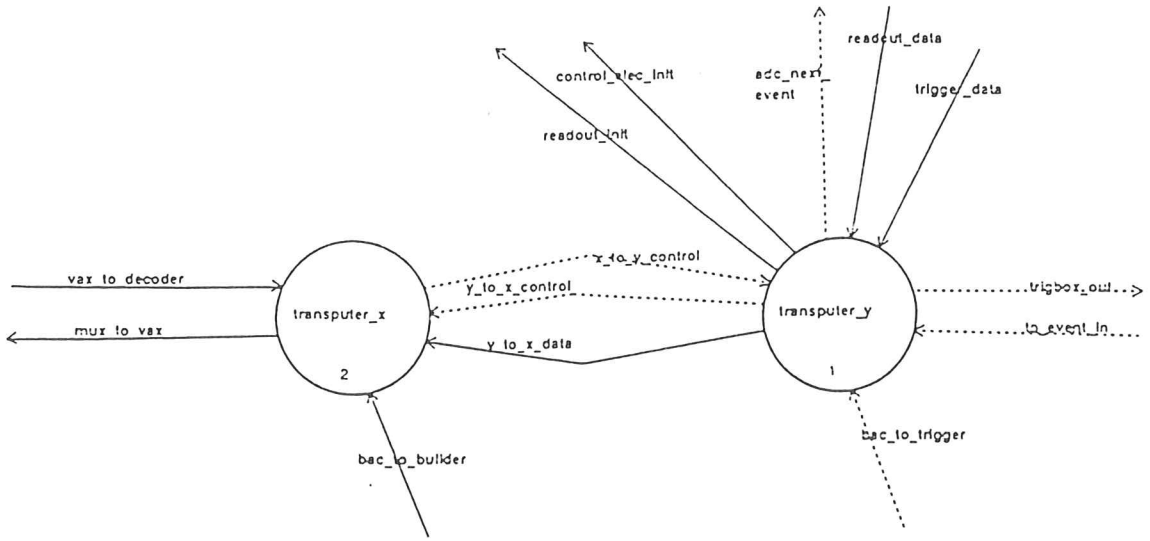

Abbildung 4.3: Verteilung der Prozesse auf dem Transputermodul

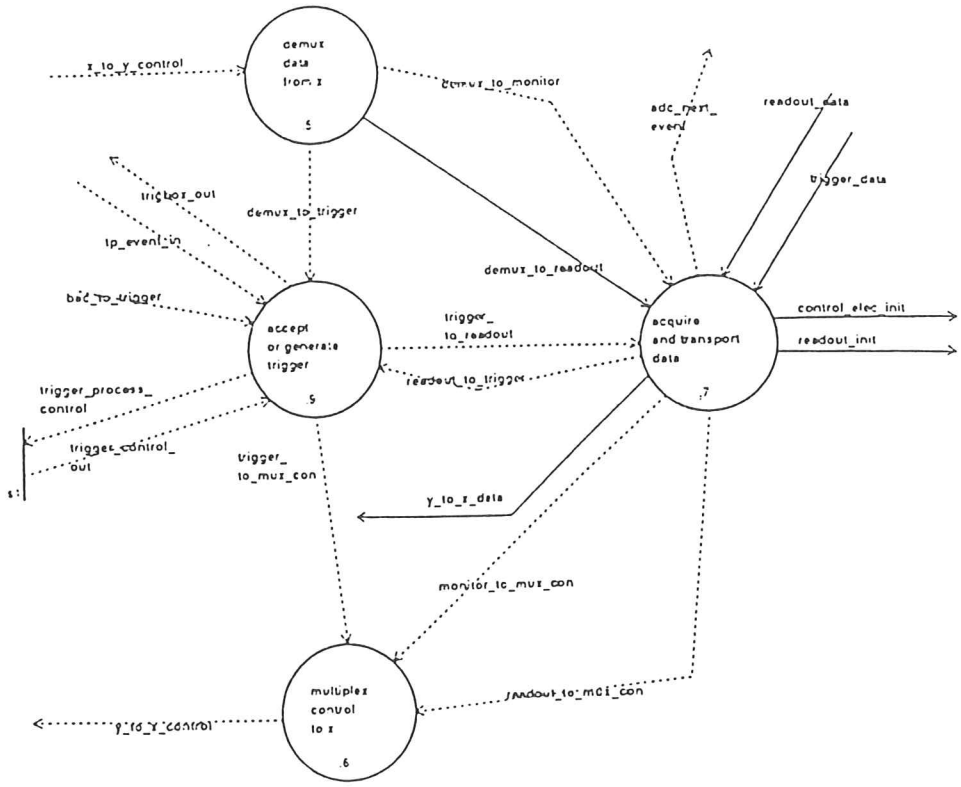

Abbildung 4.4: Trigger- und Datenerfassungsprozess auf Transputer Y

vorigen Abschnitt identisch ist. Abb. 4.2 zeigt dabei das Transputermodul in seiner Systemumgebung, Abb. 4.3 die Gliederung der Software auf dem Modul. An letzterem ist wesentlich, daß sich auf einem Transputermodul zwei mit X und Y bezeichnete Transputer befinden und daher zwei voneinander unabhängige Aufgaben erfüllen können. Im Falle des beschriebenen Systems wickelt Transputer X die Strukturierung der Daten und Kommunikation mit der VAX ab, während Transputer Y die Kommunikation mit den Komponenten des Experiments erledigt. Die auf Transputer Y befindlichen Prozesse sind in Abb. 4.4 spezifiziert.

Die Funktion der einzelnen Elemente der Abbildungen kann hier nicht weiter erläutert werden, als sie aus den Bezeichnungen in den Abbildungen hervorgeht, da dies zu weit in den Bereich der Softwareentwicklung führen würde; so müßten für ein tieferes Verständnis des Systems auch Dinge wie Protokolle der Prozeßkommunikation oder Prozeß- und Systemzustände berücksichtigt werden[3]. Zu erkennen ist jedoch auch auf diesem Niveau schon, daß die Stellung des Triggerprozesses innerhalb des Kontroll- und Datenerfassungssystems genauso zentral ist wie die Stellung der Triggerbox innerhalb des Experimentaufbaus. Insbesondere Abb. 4.4 zeigt, daß fast der gesamte Kontrollfluß zum und vom Experiment über den Triggerprozeß und die Triggerbox abgewickelt wird.

Wie geht dies nun in der Praxis vor sich? Zu Beginn einer Messung[4] werden die benötigten Komponenten dem Experiment zugeschaltet. Dies geschieht durch Beschreiben der Ausgangs-Ports der Triggerbox mit entsprechenden Werten, so daß die dort über Signalleitungen angeschlossenen Komponenten über ihre Aufgabe informiert werden. Alsdann wird durch Setzen der Triggermaske die Art der aufzuzeichnenden Daten bestimmt, bevor schließlich das Inhibit-Flip-Flop gelöscht und somit die Triggerbox "scharf gemacht" wird. Während eines Runs kann über die Eingangs-Ports der Status der Komponenten erfaßt werden, sofern diese dort angeschlossen sind; von eigentlichem Interesse sind hier jedoch die von der Triggerbox erzeugten Interrupts, die, je nach ihrer Quelle, ein Aktivieren oder Deaktivieren der Triggerbox oder aber die Erfassung von Daten von den zuvor selektierten Komponenten nach sich ziehen. Soll ein Run unterbrochen oder beendet werde, so genügt es, das Inhibit-Flip-Flop zu setzen.

[3] Eine vollständige *SASD*-Beschreibung des Datenerfassungs- und Experimentkontrollsystems findet sich z. B. in [12]

[4] Üblicherweise wird statt *Messung* das engl. *Run* verwendet

Diese kurze Beschreibung von der Bedienung der Triggerbox kann nur einen groben Eindruck vom Aufbau des Triggerprozesses geben, da Aspekte wie z. B. die Kommunikation des Prozesses mit anderen Teilen des Systems überhaupt nicht erwähnt wurden. Dennoch zeigt sie, daß die RT-Level-Beschreibung der Triggerbox Grundlage für die Entwicklung ihres Software-Pendants ist.

Kapitel 5

Erste Ergebnisse von der Strahlkalibrierung der ZEUS-FCAL-Module

In den bisherigen Kapiteln wurde das Kalibrierungsprogramm der FCAL-Module[1] zusammen mit seinem experimentellen Aufbau vorgestellt. Einen wesentlichen Aspekt stellte dabei die elektronische Datenerfassung und ihre Synchronisation mit dem Experiment dar. In diesem abschließenden Kapitel wird nun gezeigt, wie aus den für die Erfassung aufbereiteten Daten wieder ihre ursprüngliche physikalische Bedeutung gewonnen wird, und wie sich aus dieser schließlich Aussagen über die Qualität der untersuchten FCAL-Module gewinnen lassen. Letzteres wird am Beispiel der Kalibrierung eines FEMC geschehen.

5.1 Rekonstruktion der physikalischen Daten

Der erste Schritt bei der Datenrekonstruktion besteht in der Auswahl der zu rekonstruierenden, d. h. für die weitere Analyse relevanten Daten. Durch den Einsatz der Triggerbox innerhalb des Datenerfassungssystems ist es möglich gewesen, schon während der Datenaufnahme über die Verwend-

[1] Die verschiedenen Sektionen des Kalorimeters sind in Kapitel 2.1 beschrieben.

barkeit von Daten zu entscheiden und dementsprechend nicht verwendbare Daten auch gar nicht erst aufzuzeichnen. Da diese Entscheidung jedoch nicht für alle Daten getroffen werden kann, befinden sich auch unter den aufgezeichneten Daten noch einige wenige, die vor der eigentlichen Analyse auszusortieren sind.

Daten werden immer dann für die weitere Analyse aufgezeichnet, wenn sie durch das Einfallen eines Teilchens einer zuvor gewählten Sorte in das Kalorimeter erzeugt werden. Dadurch ist jedoch nicht ausgeschlossen, daß zusammen mit dem selektierten noch weitere Teilchen das Kalorimeter treffen und so möglicherweise die Meßergebnisse verfälschen. Um derartige Daten zu erkennen und aussortieren zu können, ist ihr zugehöriger Triggerstatus auszuwerten. Hierfür enthält die Triggerbox neben dem Triggermasken-Register, in das die gewünschte Teilchensorte eingetragen wird, das gesonderte Triggerstatus-Register (vgl. Kap. 3.2.2), in dem die Sorten aller Teilchen registriert sind, die zusammen mit dem selektierten in das Kalorimeter einfallen. Nur wenn Triggermaske und Triggerstatus übereinstimmen sind die entsprechenden Daten für die weitere Analyse verwendbar.

Der Verlauf eines Signals im Kalorimeter von seiner Erzeugung bis zur Erfassung durch das Datennahmesystem wurde bereits in Kap. 2.3 erläutert. Die dabei für die weitere Verarbeitung der Daten entscheidende Tatsache war das "samplen" der Daten, d. h. das Abtasten eines Signals und Speichern der dabei ermittelten Pulshöhen, denn dadurch wird die Rekonstruktion des ursprünglichen Signals aus diesen samples zu einer wesentlichen Aufgabe bei der Datenanalyse.

Bevor man an diese Aufgabe herangehen kann, sind jedoch zunächst die einzelnen samples um die von der Ausleseelektronik ausgeübten Effekte zu korrigieren: nach der Abtastung sind die Pulshöhen sowohl in einer Analog-Pipeline als auch in einem Pufferspeicher zwischengespeichert und jeweils geringfügig verstärkt und mit einem kleinen zusätzlichen Rauschanteil versehen worden. Daher ist ein jedes sample vor seiner Auswertung mit entsprechenden Korrekturfaktoren[2] zu verrechnen, wobei besonders auf die Reihenfolge der Korrekturen zu achten ist: da die Rauschanteile der Pipeline in dem dahinterliegenden Pufferspeicher bereits mit verstärkt worden sind, ist sicherzustellen, daß die Korrektur in gegenüber dem Auftreten der

[2]Diese Korrekturfaktoren sind für jede front-end-Elektronik-Karte gesondert zu ermitteln. Im Falle der Pipeline und des Pufferspeichers sind für jede Speicherzelle Rauschen und Verstärkungsfaktor zu ermitteln.

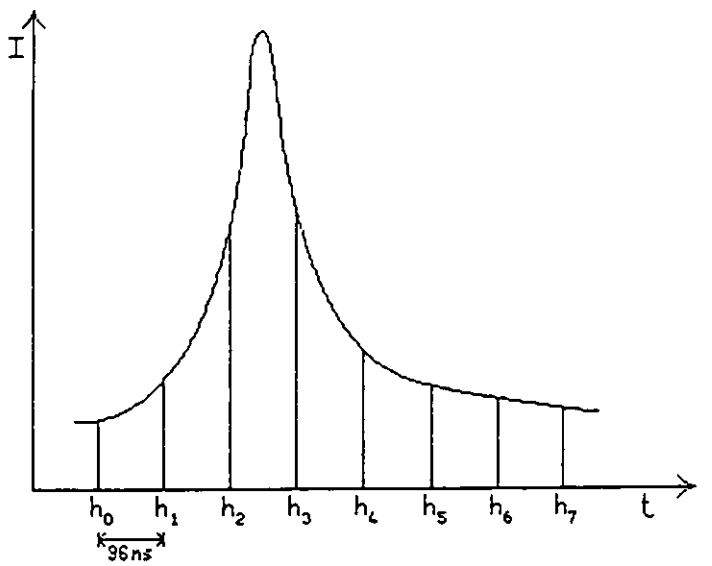

Abbildung 5.1: Form eines PMT-Pulses nach Durchlaufen des shapers und Lage der Abtastpunkte

Störungen umgekehrter Reihenfolge erfolgt. Demnach sind die samples zuerst von dem Rauschanteil des Pufferspeichers zu befreien und anschließend durch dessen Verstärkungsfaktor zu dividieren, bevor sie entsprechend um Rauschen und Verstärkung der Pipeline korrigiert werden.

Die so korrigierten samples sind nun die Ausgangsinformation für die Rekonstruktion der von einem PMT erzeugten Ladung Q. Ihre Höhen hängen dabei von der Form des abgetasteten Pulses ab, die wiederum durch den shaper festgelegt ist. Ein Puls mit seinen acht Abtastpunkten h_i ist in Abb. 5.1 gezeigt: die Abtastung erfolgt in Abständen von $96\,ns$ (entsprechend dem Wechselwirkungsintervall von *HERA*), und die shaper sind so konstruiert, daß mit Sicherheit ein sample auf der steigenden Flanke des Pulses und eins auf der fallenden genommen wird. Der Abbildung ist zu entnehmen, daß es sich bei diesen samples um h_2 und h_3 handelt, während h_0 ein Maß für das Rauschen des Kalorimeters gibt, da es deutlich vor dem Eintreffen des Pulses genommen wird. Die gesuchte Ladung Q läßt sich demnach aus einer gewichteten Summe der um das "physikalische Rauschen h_0" korrigierten samples h_2 und h_3 gewinnen.

Als Ansatz für die Ladungsrekonstruktion wird

$$Q' = (h_2 - h_0) + C \times (h_3 - h_0).$$

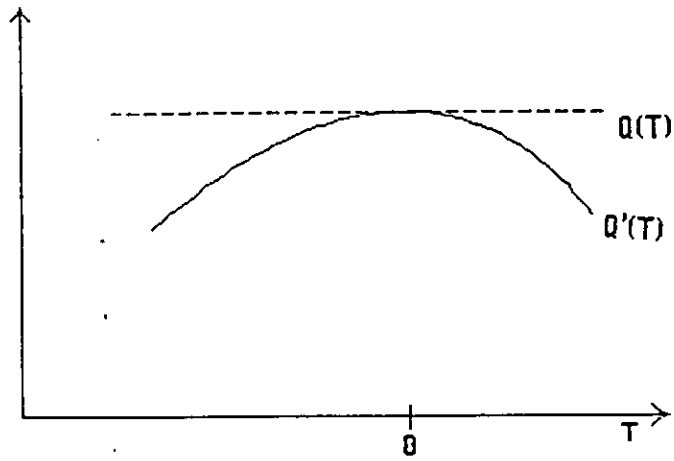

Abbildung 5.2: Abhängigkeit der rekonstruierten Ladung von der Lage des Pulses relativ zu den Abtastpunkten.

gewählt. Exemplarstreuungen der shaper, die sich in entsprechenden Veränderungen der Pulsform äußern, gehen dabei in die Konstante C ein, die u. a. die Flankensteilheit des geformten Pulses bei h_2 und h_3 enthält.

Offensichtlich hängt Q' von der Lage des abgetasteten Pulses relativ zu den Abtastzeitpunkten ab und wird maximal, wenn h_2 und h_3 gleichhoch sind. Um dies zu sehen, definiert man eine Größe T gemäß

$$T = h_2 - h_3$$

und betrachtet $Q'(T)$, das konstant sein müßte, falls Q' unabhängig von T ist. Die Abhängigkeit des Ansatzes Q' von T ist in Abb. 5.2 gezeigt. Da $Q'(T)$ nicht konstant ist, die gesuchte Ladung Q aber nicht von T abhängt, ist der Ansatz mit einer von T abhängigen Korrekturfunktion $f(T)$ zu verrechnen, um die gesuchte Ladung Q zu erhalten[3]. Zieht man alle erforderlichen Proportionalitätskonstanten in $f(T)$ hinein, so ist schließlich

$$Q(T) = Q' \times f(T).$$

[3]Diese Funktion wird numerisch bestimmt und ist in guter Näherung ein Polynom vierten Grades.

die um die Eigenschaften des Auslesezweiges korrigierte, tatsächlich vom PMT erzeugte Ladung. Sie ist diejenige Größe, die die objektive Antwort des Kalorimeters auf ein eingefallenes Teilchen beschreibt.

5.2 Energieauflösung, Linearität und Uniformität

Eine Messung mit dem Kalorimeter im Teststrahl bedeutet im wesentlichen die Bestimmung des im Kalorimeter erzeugten Signales[4] Q als Antwort auf ein an einer bestimmten Position einfallendes Teilchen einer definierten Energie E. Dementsprechend erhält man für jede Meßposition eine Meßkurve $Q(E)$. Bei der Auswertung der genommenen Daten sind diese dann in eine Form zu bringen, aus der Aussagen über die Qualität des untersuchten Kalorimetermoduls gewonnen werden können. In diesem Abschnitt sollen einige während der ersten Phase des Experiments gewonnene Ergebnisse vorgestellt werden. Es handelt sich dabei um Messungen an dem elektromagnetischen Kalorimeter eines zentralen FCAL-Moduls (interne Bezeichnung: NL-1).

Die im vorigen Abschnitt angegebenen Korrekturmethoden erlauben es, die Meßdaten von allen Eigenheiten der Auslesekanäle, über die sie jeweils erfaßt wurden, zu befreien. Um aus ihnen physikalische Aussagen gewinnen zu können, müssen sie jedoch noch weiter aufbereitet werden. Zwei wesentliche Aspekte treten dabei schon bei der Datenselektion, also noch vor ihrer statistischen Verarbeitung, auf:

- Der Impuls der Strahlteilchen wird über einen Dipolmagneten mit dahinter befindlichem Kollimator festgelegt. Demnach ist er nicht etwa konstant, sondern variiert innerhalb durch die Kollimatoröffnung festgelegter Grenzen, so daß auch bei "fester Strahlenergie" die Messung nicht an einem Punkt p_0, sondern in einem kleinen Intervall $[p_0 - \Delta p/2; p_0 + \Delta p/2]$ der Impulsachse stattfindet. Um die Auswertung einer Messung nicht zu verfälschen, ist daher für jedes Teilchen, welches das Kalorimeter trifft, nicht nur die von ihm vom PMT

[4] Der Buchstabe Q deutet an dieser Stelle an, daß das Kalorimetersignal in Form einer von einem PMT erzeugten Ladung ausgelesen wird.

erzeugte Ladung, sondern auch seine exakte Energie[5] aufzunehmen. Verhält sich das Kalorimeter im Meßintervall linear, so können die Daten auf den Schwerpunkt des Intervalls extrapoliert werden, um so für die Datenpräsentation doch einen Meßpunkt zu erhalten. Wählt man für die Analyse diejenigen Daten aus, die von Teilchen mit einem nur wenig von p_0 verschiedenen Impuls[6] erzeugt wurden, so ist dies i. allg. zulässig.

- Elektromagnetische Schauer sind in ihrer Ausdehnung so gering, daß sie sich, falls sie durch ein zentral in einen strip einfallendes Elektron entstanden sind, in der überwiegenden Mehrheit aller Fälle nur in diesem entwickeln. Liest man nun das gesamte Modul aus, um die durch sie vom Kalorimeter erzeugte Ladung zu bestimmen, so liest man überwiegend Kanäle, in denen nur das UNO ein Signal erzeugt hat. Das Uranrauschen ist zwar eine über das gesamte Kalorimeter konstante Größe, jedoch gilt dies nur im zeitlichen Mittel, weshalb das UNO auch in einem vom Signalmeßzweig getrennten Weg über einen Integrator gemessen wird. Wird es gesamplet, was im Signalmeßzweig zwangsläufig der Fall ist, so fluktuiert es auch. Bildet man nun über alle Kalorimeterkanäle die Summe über die dort jeweils erzeugten Signale, um so die insgesamt erzeugte Ladung zu erhalten, so enthält diese Größe also nicht nur die Fluktuationen des durch das eingefallene Teilchen erzeugten Signales, sondern auch durch das Uranrauschen bedingte. Da diese jedoch zu einem schlechteren Signalauflösungsvermögen führen, gilt es, sie weitestmöglich zu vermeiden. Bei der Datenauswertung hat es sich gezeigt, daß ein elektromagnetischer Schauer vollständig und mit optimaler Auflösung erfaßt wird, falls der zentral getroffene strip über den Meßzweig niedriger und die sich links und rechts direkt neben ihm befindlichen über den Meßzweig hoher Signalverstärkung ausgelesen werden.

Zur Überprüfung der Linearität des Antwortverhaltens eines Kalorimetermoduls und Ermittlung seines Energieauflösungsvermögens wurde ein

[5] Die Energiemessung der Strahlteilchen erfolgt über die in Abb. 2.3 dargestellten Driftkammern *MWPC 1* bis *MWPC 4*

[6] In der Praxis haben sich Impulsschwankungen von bis zu einem Prozent als tolerierbar erwiesen.

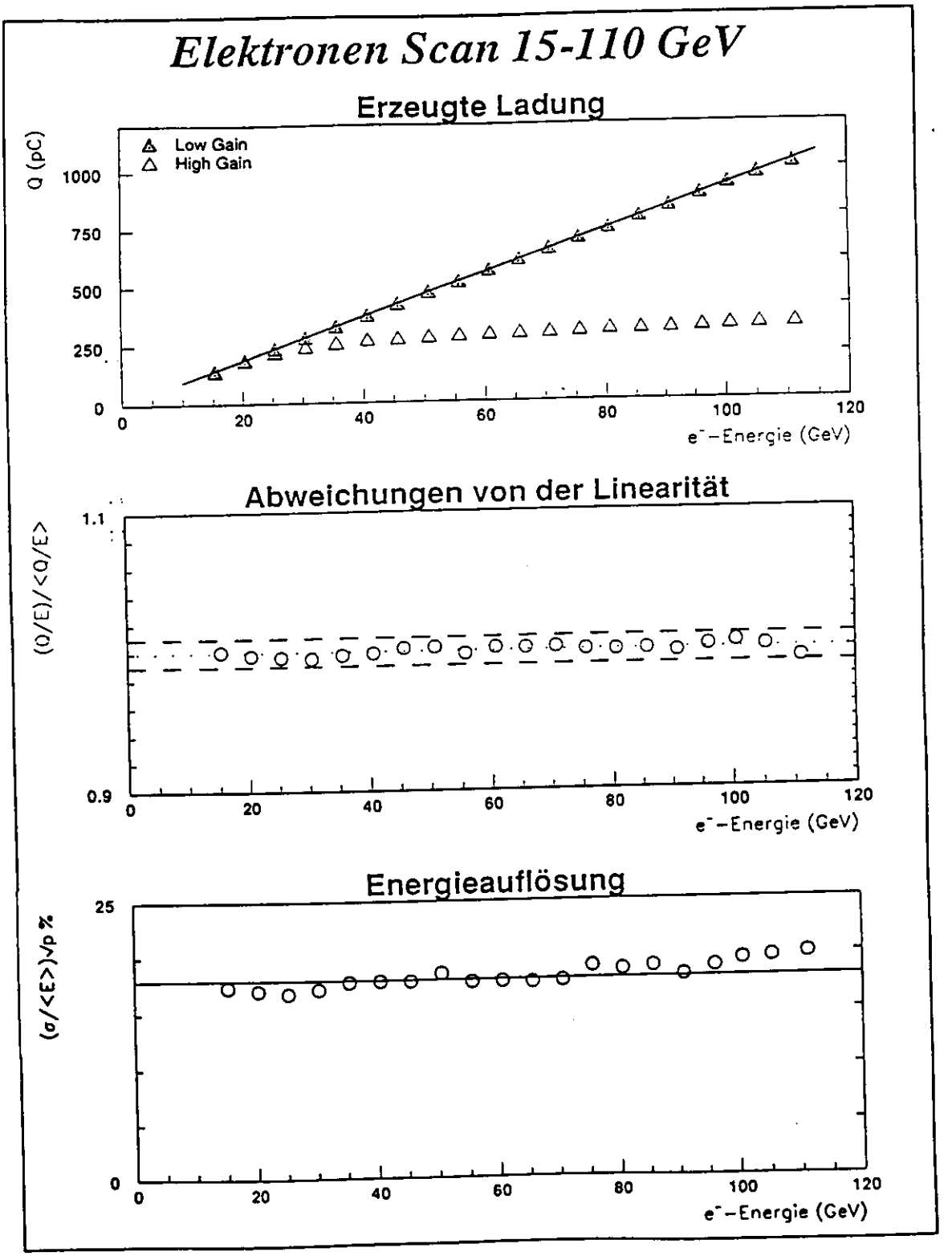

Abbildung 5.3: Linearität und Energieauflösungsvermögen des EMC aus Modul NL-1. Die statistischen Fehler sind so klein, daß sie in den Graphiken nicht gezeigt werden können.

strip des EMC mit Elektronen einer Energie von $15\,GeV$, die in $5\,GeV$-Schritten bis auf $110\,GeV$ erhöht wurde, beschossen. Der Mittelwert der bei jeder Einschußenergie vom Kalorimeter erzeugten, nach den eben bestimmten Kriterien ermittelten Ladung ist in der oberen Graphik der Abb. 5.3 dargestellt. Es ist zu erkennen, daß der Meßzweig hoher Signalverstärkung (Hi-Gain) bei etwa $25\,GeV$ Einschußenergie in Sättigung geht, während sein Pendant (Lo-Gain) über den gesamten untersuchten Energiebereich linear ist, was durch die gezeigte Gerade verdeutlicht wird.

Um die Linearität des Moduls quantitativ fassen zu können, kann der Abstand der einzelnen Meßpunkte von der Näherungsgeraden betrachtet werden. Dazu ist aus den Daten die mittlere im Kalorimeter erzeugte Ladung pro GeV dort deponierter Energie zu bestimmen[7]: wird sie mit der mittleren Energie E eines Meßpunkts multipliziert und dann ins Verhältnis zu der tatsächlich für die Energie ermittelten Ladung $Q(E)$ gesetzt, so erhält man ein Maß für die relative Abweichung der Kalorimeterantwort vom linearen Verhalten bei einer Energie E. In der mittleren Graphik in Abb. 5.3 sind die entsprechenden Daten dargestellt. Die ermittelte Abweichung der Kalorimeterantwort von der Linearität liegt im Mittel bei weniger als 0.3 %:

$$\frac{Q/E}{< Q/E >} = 1.0000 \pm 0.0028$$

Bislang ist lediglich gezeigt, daß ein in das Kalorimetermodul einfallendes Teilchen dort ein zu seiner Energie E proportionales Signal Q erzeugt. Wichtig zu wissen ist aber ebenfalls, mit welchem Auflösungsvermögen die Energie des Teilchens rekonstruiert werden kann. Um dies für eine Einschußenergie E zu bestimmen, ermittelt man die Standardabweichung der Verteilung der mittleren vom Kalorimeter erzeugten Ladung $Q(E)$. Sie sollte mit $1/\sqrt{E}$ skalieren, was sich unter Ausnutzung der Proportionalität von Q und E als

$$\frac{\sigma}{< Q >}\sqrt{E} = const.$$

schreiben läßt. Die Konstante sollte dabei nach den in Kap. 2.1 genannten Anforderungen 18 % betragen. Die Größe $\frac{\sigma}{<Q>}\sqrt{E}$ wurde für die untersuchten Energien bestimmt und die erhaltenen Werte in der unteren Graphik

[7] also die Steigung der Näherungsgeraden

von Abb. 5.3 dargestellt. Es wurde gefunden

$$\frac{\sigma}{< Q >}\sqrt{E} = 17.9\%, \qquad 15\,GeV < E < 110\,GeV.$$

Die bisher präsentierten Daten wurden alle in nur einem strip des Kalorimeters gewonnen. Um feststellen zu können, ob sie repräsentativ für das Modul[8] sind und gleichzeitig einen Eindruck von der Uniformität der Signalantwort über das gesamte FEMC zu gewinnen, wurden alle Sektionen des Kalorimeters mit Elektronen fester Energie beschossen. Als Einschußenergien wurden dabei $20\,GeV$, $75\,GeV$ und $110\,GeV$ gewählt, und im Gegensatz zu der zuvor beschriebenen Messung wurden hier nicht nur drei strips des EMC, sondern drei komplette Türme des Moduls ausgewertet. Dabei wurden die jeweils im Lo-Gain-Kanal erzeugten Signale zugrunde gelegt[9]. Abb. 5.4 zeigt die erhaltenen Resultate, wobei die rekonstruierten Ladungen zuvor ins Verhältnis zum im jeweiligen Kanal gemessenen Uranrauschen gesetzt wurden, um so den absoluten Vergleich der Meßwerte zu ermöglichen: wie bereits erwähnt, hängt das von einem PMT bei Einfall Lichtes fester Wellenlänge erzeugte Signal von der ihm zugeführten Hochspannung ab. Mißt man nun in allen FEMC-Sektionen das UNO, so spiegeln die Meßwertschwankungen genau die relativen Schwankungen der Signalverstärkung der Auslesekanäle zueinander wider, da das mittlere Uranrauschen eine im gesamten Kalorimeter konstante Größe ist. Division der Meßwerte durch die Höhe des ihnen unterliegenden Uranrauschens führt diese also gerade auf eine einheitliche Skala zurück.

Anhand von Abb. 5.4 läßt sich bereits erkennen, daß das Kalorimeter über seine volle Breite nahezu uniform ist. Um dies quantitativ fassen zu können, ist in Abb. 5.5 ein jeder Meßpunkt im Verhältnis zum mittleren, bei der entsprechenden Einschußenergie im Kalorimeter erzeugten Signal dargestellt. Man erkennt, daß die relativen Signalhöhen im Mittel um weniger als zwei Prozent variieren.

Nähert man die für die drei Energien gewonnenen Meßpunkte jeder EMC-Sektion jeweils durch eine Gerade an und untersucht dann — analog

[8] Der Zeitaufwand der zuvor vorgestellten Messung ist zu hoch, als daß diese für jeden EMC-strip jedes Moduls durchgeführt werden könnte

[9] Dadurch wird zwar das Signalauflösungsvermögen verschlechtert, jedoch erlaubt es einen objektiveren Vergleich der verschiedenen EMC-Sektionen, da die ausgewerteten Signale sich immer nur auf einen, und nicht auf drei strips beziehen.

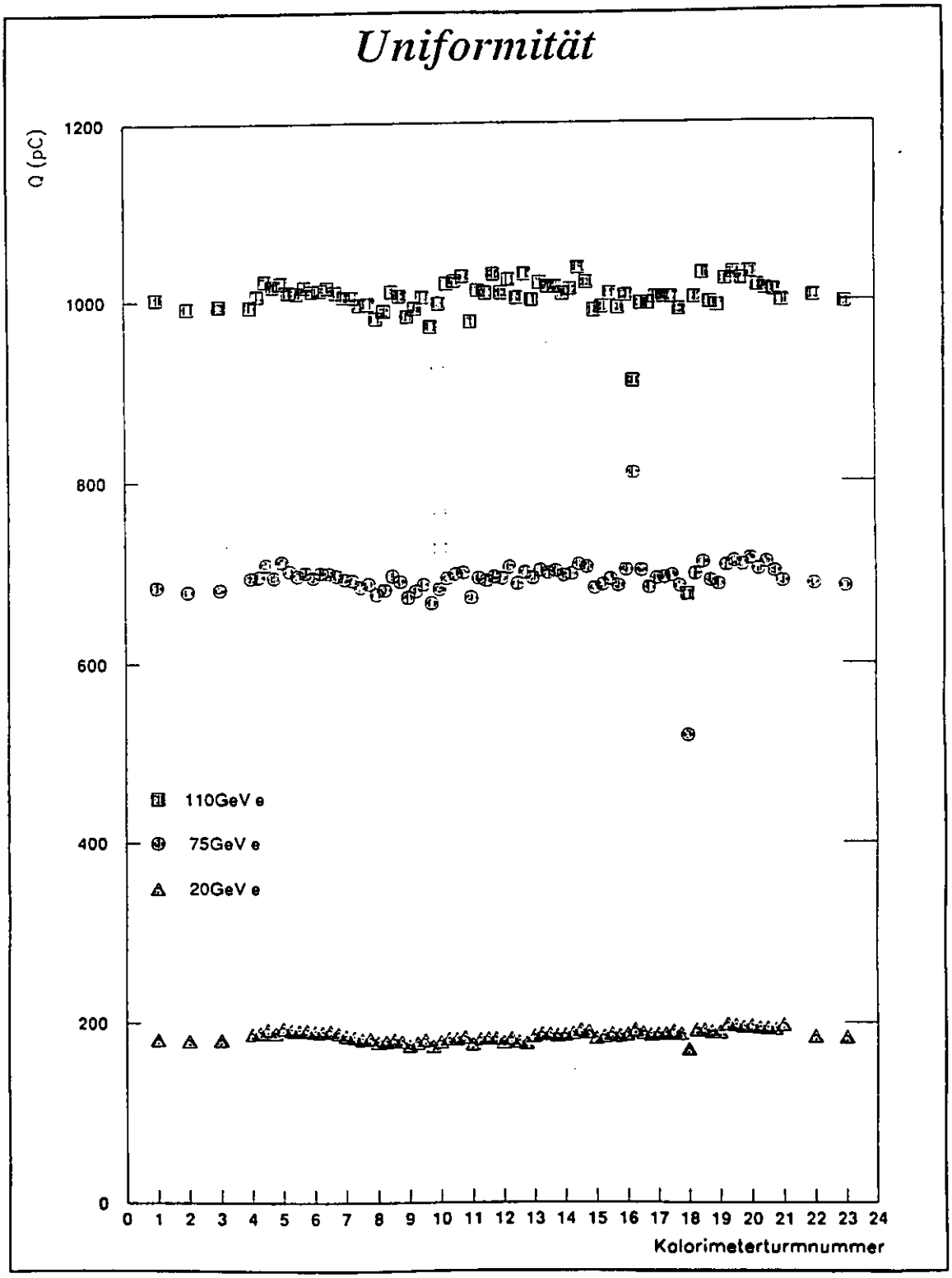

Abbildung 5.4: Signalantwort des Kalorimeters auf zentral in einen strip geschossene Elektronen einer Energie von $20\,GeV$, $75\,GeV$ und $110\,GeV$. Die Ausreißer sind auf Instabilitäten der an die jeweiligen PMTs angelegten Hochspannung zurückzuführen.

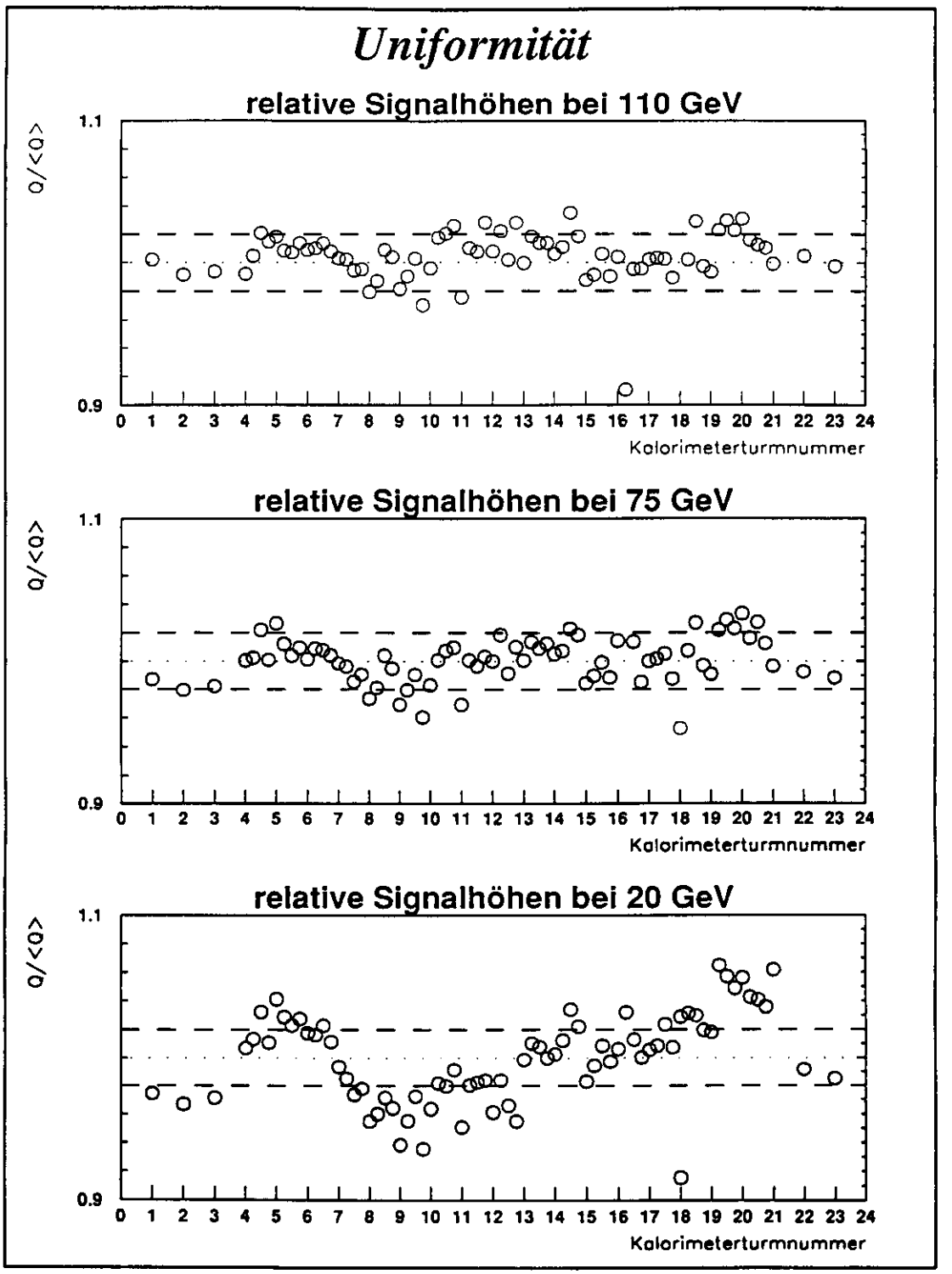

Abbildung 5.5: Quantitative Darstellung der Signalhöhenschwankungen über das Kalorimetermodul

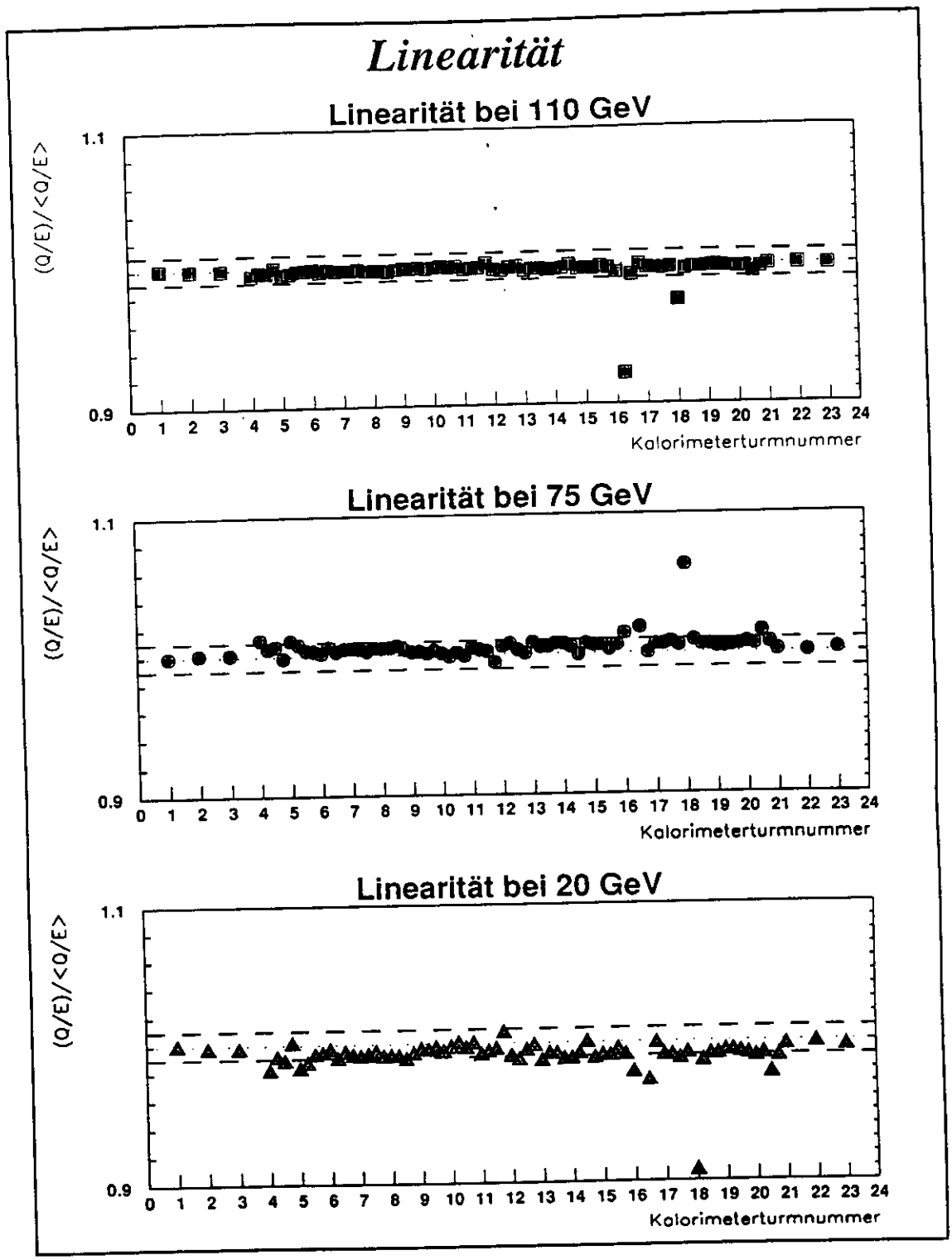

Abbildung 5.6: Linearität der Signalantwort über das gesamte Kalorimeter bei $20\,GeV$, $75\,GeV$ und $110\,GeV$

dem eingangs bereits dargestellten Verfahren — den relativen Abstand eines Meßpunktes von der Näherungsgeraden, so läßt sich die Linearität des Antwortverhaltens des Kalorimeters in allen Kanälen gesondert überprüfen. In Abb. 5.6 sind die Meßwerte entsprechend dargestellt: obwohl die Datenselektion hier nicht nach so strengen Kriterien durchgeführt wurde wie bei der eingangs betrachteten Meßreihe, liegen die Abweichungen des Kalorimeterverhaltens von der Linearität dennoch bei weniger als einem Prozent. Es kann also davon ausgegangen werden, daß die oben für einen strip des Moduls gewonnenen Ergebnisse durchaus repräsentativ für das gesamte Kalorimeter sind.

Abschließend sei noch kurz eine Frage aufgeworfen, die sich im Zusammenhang mit Kalibrierungsmessungen immer stellt: Unterliegen die ermittelten Daten zeitlichen Schwankungen, oder sind sie stabil? Eine Möglichkeit, diese Frage zu klären, besteht in dem Versuch, einmal gewonnene Ergebnisse zu reproduzieren. Für das untersuchte *ZEUS*-FCAL-Modul ist dies im Falle eines Scans des Kalorimeters mit 20 GeV Elektronen geschehen, der nach einer Zeitspanne von einem Monat ein zweites Mal durchgeführt wurde. Die ursprünglich erhaltenen Ergebnisse konnten dabei im Low-Gain-Kanal bis auf 0.7 % reproduziert werden [13].

Kapitel 6

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Kalibrierung der Module des *ZEUS*-Vorwärts- und Rückwärtskalorimeters. Nach einer kurzen Beschreibung des Aufbaus und der Funktionsweise des *ZEUS*-Kalorimeters wird ein für seine Kalibrierung am CERN-SPS errichteter Teststand vorgestellt. Neben seinem physikalischen Aufbau und den dort durchgeführten Kalibrierungsmessungen steht dabei vor allem die Synchronisation des Datenerfassungs- und Kontrollsystems mit dem Experiment im Zentrum der Darstellung. Die Beschreibung einer eigens für Synchronisationszwecke entwickelten Hardware, ihrer Integration in den Experimentaufbau und ihrer Inbetriebnahme bilden den Kern des ersten Teils der Arbeit.

Im zweiten Teil der Arbeit werden erste in dem Experiment gewonnene Ergebnisse vorgestellt. Es wird die Kalibrierung einer elektromagnetischen Sektion eines Vorwärts-Kalorimetermoduls im Teststrahl erläutert. Die dabei gewonnenen Resultate erfüllen sämtlich die an das Kalorimeter vor seiner Entwicklung gestellten Leistungsansprüche: die Linearität des Kalorimeters ist besser als 0.28%, die Uniformität der Signalantwort besser als 2%, und das relative Energieauflösungsvermögen für Elektron-induzierte Schauer beträgt $17.9\%/\sqrt{E}$, wobei E in GeV gemessen wird.

Anhang A

Die Systemumgebung

A.1 Der VMEbus

A.1.1 Bussysteme

Innerhalb einer komplexen Computeranlage gibt es etliche Komponenten, die für eine sichere und effektive Betriebsabwicklung regelmäßig Informationen austauschen müssen. Da Art und Umfang dieser Informationen dabei genauso vom Systemzustand abhängen wie die Informationsquellen und -anforderer, sind die Informationswege nicht a-priori festlegbar, so daß der Datenaustausch zwischen irgendzwei Komponenten des Systems jederzeit möglich sein muß.

Ein Weg, dieser Anforderung gerecht zu werden, besteht in der Schaffung direkter "Punkt-zu-Punkt"-Verbindungen von einer jeden zu einer jeden Komponente des Systems. Dies ist jedoch nicht nur so aufwendig und teuer, daß es zumindest für große Rechnersysteme nicht realisiert werden kann, es hat außerdem den großen Nachteil, daß jede noch so kleine Modifikation des Systems sofort eine Anpassung aller am System beteiligten Komponenten nach sich zieht.

Eine weitaus billigere und obendrein flexiblere und praktikable Methode zur Bereitstellung der benötigten Datenpfade ist die Schaffung eines sogenannten Bussystems. Es besagt, daß für jeden innerhalb des Systems zu übertragenden Informationstyp[1] eine Sammelleitung geschaffen wird, an die

[1] Typ bezieht sich hierbei nur auf die Erscheinungsform der Information — Datum, Adresse, Kontrolleitung,... — und nicht auf deren Inhalt!

dann alle Komponenten des Systems angeschlossen werden. Während des Betriebs werden dann für zwei Komponenten, die miteinander kommunizieren müssen, jeweils für die Dauer ihrer Kommunikation alle Sammelleitungen — der Bus — reserviert.

Um einen reibungslosen Datenübertragungsablauf auf einem Bus zu gewährleisten, muß dieser während des Betriebs überwacht werden. Dies wird von einem speziellen "Bus-Controller" erledigt, dem Komponenten ihren Nutzungsbedarf des Busses mitteilen, und der Komponenten über ihre Nutzungsbefugnis informiert. Für die Abwicklung dieser Synchronisation sind in einen Bus spezielle Steuerleitungen integriert.

Aus Sicht des Bus-Controllers gibt es lediglich Komponenten von zwei verschiedenen Sorten: entweder treten sie gelegentlich mit der Forderung, den Bus zur Verfügung gestellt zu bekommen, an ihn heran, woraufhin sie dann Herr über den Bus *(master)* wären, oder aber sie verhalten sich während des gesamten Systembetriebes passiv, da sie nie selbständig Informationen vermöge VMEbus-Zugriffen anfordern oder abgeben — in diesem Fall wären sie *slaves*[2]. Die in dieser Arbeit beschriebene Triggerbox ist ein typisches Beispiel für einen slave.

Technisch gesehen unterscheidet man grob zwei Typen von Bussen: parallele und serielle. Erstgenannte stellen für jeden Signaltyp eine eigene Leitung zur Verfügung, wobei diese teilweise noch mehrfach vorhanden sind, um so eine gleichzeitige Übertragung mehrerer Informationen zu ermöglichen[3], während letztere (neben den Steuerleitungen für die Bus-Kontrolle) nur eine einzige Busleitung besitzen, über die dann alle Informationen der Reihe nach übermittelt werden können. Derartige Busse sind zwar deutlich langsamer als ihre Gegenstücke, dafür jedoch aber auch mit wesentlich geringerem technischen und materiellen Aufwand verbunden und werden daher vor allem für lange Übertragungswege bevorzugt.

In praxi gibt es mehrere etablierte Bussysteme, die sich in ihren Zielgruppen und Leistungsdaten teils erheblich unterscheiden. Kennzeichen eines Busses, die über seine Verwendbarkeit in einem System entscheiden, sind neben den oben genannten vor allem sein Datendurchsatz, der von

[2]Mehrere Bus-Spezifikationen definieren außerdem noch sog. *location monitors*, die allerdings für einen Bus-Controller unsichtbar sind.

[3]Üblicherweise realisiert man die Daten- und Adreßleitungen eines Systems 8- bis 32mal, so daß entsprechend viele Stellen eines Datenwortes gleichzeitig übertragen werden können

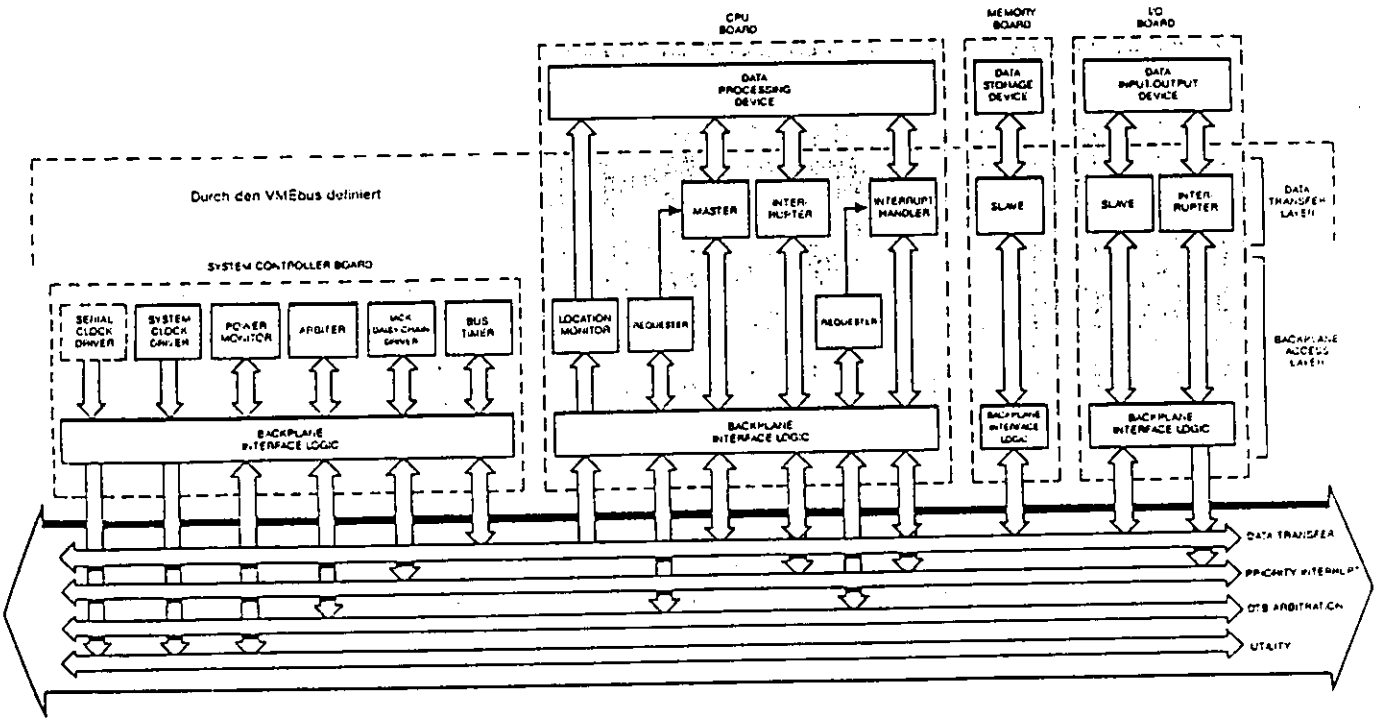

Abbildung A.1: Aufbau eines VMEbus-Systems

ihm unterstützte Funktionsumfang[4] und seine Betriebssicherheit, nicht zu vergessen jedoch auch sein mechanischer Aufbau.

Ein in der Industrie inzwischen weit verbreiteter und bewährter Bus, der auch in dem beschriebenen Teststand für die *ZEUS*-FCAL-Module zur Anwendung kam, ist der VMEbus, der hier daher kurz vorgestellt werden soll.

A.1.2 Überblick über den VMEbus

Der VMEbus ist ein 1981 von den Firmen Motorola, Mostek, Philips/Signetics und Thomson CSF vorgestelltes Bussystem, das besonders für die Eigenschaften der von Motorola entwickelten 68 000er Mikroprozessorserie ausgelegt ist. Es handelt sich um ein Backplane-Bussystem mit wahlweise

[4] Es gibt z. B Bussysteme, die mehrere Master gleichzeitig erlauben, falls die von ihnen benötigten Busabschnitte sich nicht überlappen

16 oder 32 Daten- und 24 oder 32 Adreßleitungen, wodurch es an den Betrieb von 16- und 32-bit Mikroprozessoren angepaßt ist. Das System ist modular aufgebaut, läßt mehrere Master zu und unterstützt dadurch Multiprozessorsysteme. Es ist für Datendurchsätze bis zu $20\,MByte/s$ ausgelegt[5]. Die Basis des Systems bilden die weit verbreiteten Europakarten (nach DIN 41 494) und die als zuverlässig bekannten 96-pol. DIN-Steckverbinder (41 612).

Die Busleitungen sind ihren Funktionen entsprechend zu vier Teilbussen zusammengefaßt. Es sind dies

- der **Daten-Transfer-Bus (DTB)**, der die schnelle parallele Datenübertragung zwischen verschiedenen Funktionsmoduln leistet; sein Übertragungsprotokoll ist asynchron mit Quittungsbetrieb, wobei der Master Zyklusart und Transferrichtung bestimmt,
- der **Arbitration-Bus**, der Arbiter und Requester koordiniert und Busanforderungen entsprechend ihrer Priorität berücksichtigt,
- der **Priority-Interrupt-Bus (PIB)**, über den die Programmunterbrechungsanforderungen der Interrupter an den Handler verarbeitet werden,
- der **Utility-Bus**, der neben Taktsignalen noch Signale für die Koordination beim Ein- und Ausschalten der Stromversorgung enthält.

Die Spezifikation des VMEbus [15] war von Anfang an frei von Lizenzen und Copyrights, so daß nie die Gefahr einer kurzfristigen Änderung seitens eines Buseigentümers bestand. Des weiteren ist sie in allen für Kompatibilitätsfragen wichtigen technischen Details sehr präzise, so daß in praxi eine hohe Betriebszuverlässigkeit erreicht wird — auch bei Kombination von Moduln verschiedener Hersteller miteinander. In Forschung und Industrie ist der VMEbus heute sehr weit verbreitet [16].

A.1.3 Das XVME-085-Prototyping-Modul

Das XVME-085 ist ein VME-Prototyping-Modul der Firma *XYCOM*. Es stellt auf einer Doppeleuropakarte ein VME-Interface für 16 *bit* breite slaves

[5] Neueste Entwicklungen, die die Spezifikation voll einhalten, erlauben sogar Datentransfers mit bis zu 80 *MByte/s* [14].

sowie etwa 200 cm^2 Lochrasterfläche zur Nutzung durch den Anwender zur Verfügung.

Das Modul kann wahlweise über 16 *(short I/O)* oder 24 *(memory mapped I/O)* Adreßleitungen angesprochen werden. Seine Basisadresse ist in 1 kB-Schritten innerhalb eines Bereiches von 64 kB wählbar, wobei sie im Falle des Boardzugriffs über 24 Adreßleitungen an die oberste Speicherseite bei $(FF0000)_{16}$ gebunden ist. Weiterhin kann das Modul neben dem Standard- auch noch in einem privilegierten Modus angesprochen werden, wobei die Wahl sowohl der Basisadresse als auch der Adreßbusbreite und des Betriebsmodus über Steckbrücken ("Jumper") auf dem Modul geschieht.

An seiner Basisadresse enthält das Modul ein ID-Prom, in dem ein Kürzel für seinen Herstellernamen und seine Typbezeichnung steht. Für Testzwecke kann dieses Prom ausgelesen werden, um so zu überprüfen, ob das Modul überhaupt auf Anforderungen reagiert. Des weiteren enthält es ein Status- und Kontrollregister, über das zwei an der Frontseite des Moduls befindliche Leuchtdioden und sein Interruptverhalten gesteuert werden können: das Modul ermöglicht die Generierung von Interruptanforderungen auf einer der Prioritäten 1-7, die Wahl geschieht wiederum über Jumper.

Als Schnittstelle zum Entwickler stellt das Interface einen eigenen, in den Schaltplänen mit *XYCOM* bezeichneten Bus. Er enthält die gepufferten Daten- und unteren zehn Adreßleitungen des VMEbus und Steuerleitungen für einen synchronen Betrieb von I/O-Bausteinen. Außerdem stellt er noch einige eigene Signale (z. B. zur Vereinfachung der Adreßdekodierung) bereit.

A.2 Transputer und Occam

A.2.1 Das Transputerkonzept

Transputer sind eine von der britischen Firma *INMOS Ltd.* entwickelte und vertriebene Familie von VLSI-Schaltkreisen, die eine CPU, internen Speicher und verschiedene anwendungsspezifische Interfaces enthalten. Sie zeichnen sich durch einen in die Hardware integrierten Scheduler aus, wodurch sie zum Einsatz bei der Parallelverarbeitung von Prozessen prädestiniert sind. Dies wird noch durch die den Transputern eigenen Links verstärkt, einer speziell für die Verbindung von Transputern untereinander entwickelten seriellen Schnittstelle. Links ermöglichen es, Transputer ohne zusätzliche

Hardware miteinander zu verbinden und schaffen dadurch eine Kommunikationsmöglichkeit zwischen auf verschiedenen Prozessoren laufenden Prozessen.

Für die Programmierung von Transputern wurde mit occam eine eigene Programmiersprache entwickelt. Sie basiert auf der Theorie der *"Communicating Sequential Processes"* von *C. A. R. Hoare* [17], in der ein Formalismus zur Beschreibung parallel ausgeführter sequentieller Prozesse, die nur über direkte Punkt-zu-Punkt-Verbindungen miteinander kommunizieren können, entwickelt worden ist. Diese Art von Verbindung, kurz Kanal genannt, wird bei Transputern entweder virtuell oder aber durch eine physikalische Verbindung zwischen zwei Links realisiert — ersteres, falls beide Prozesse auf einem Prozessor laufen, letzteres im Falle von auf zwei Prozessoren verteilten Prozessen.

Die große Stärke von occam ist, daß es nicht zwischen virtuellen und physikalischen Kanälen unterscheidet, was für den Programmentwickler den Vorteil hat, daß er sich bei der Erstellung seiner Programme nicht um die Rechnerkonfiguration zu kümmern braucht. Bei einem geeigneten Betriebssystem erfährt er nicht einmal mehr, ob von ihm parallel gestartete Prozesse wirklich parallel oder aber "nur" im Zeitscheibenverfahren laufen — wobei er jedoch bei expliziter Angabe der Rechnerkonfiguration über occam stets festlegen kann, welcher Prozeß auf welchem Prozessor gestartet werden soll[6].

A.2.2 Prozeßkontrolle durch Occam

Mit occam wurde eine Programmiersprache geschaffen, die speziell für den Einsatz auf Transputernetzwerken ausgelegt ist. Sie enthält alle für die Kontrolle von und Kommunikation zwischen auf einem Netzwerk laufenden Prozessen notwendigen Instruktionen.

In occam wird nicht zwischen Anweisungen und Prozessen unterschieden: auf einem Transputernetzwerk wird ein Programm ausgeführt, indem auf jedem Transputer des Netzwerks ein Prozeß gestartet wird. Diese Prozesse können über Kanäle miteinander kommunizieren und bestehen ihrer-

[6]Bei der Programmentwicklung mit dem occam-Entwicklungssystem von INMOS Ltd. ist der Programmierer gezwungen, die Rechnerkonfiguration und Prozeßverteilung auf die Konfiguration anzugeben. Ein Betriebssystem, das die Plazierung von Prozessen in einem Transputernetzwerk übernimmt, soll mit *HELIOS* [18] geschaffen worden sein.

seits wieder aus Prozessen, die sequentiell und / oder parallel ablaufen und ggf. über Kanäle miteinander kommunizieren. Die Bausteine eines occam-Programms sind also elementare Prozesse.

Für die Formulierung von sequentiellen Prozessen, die als Programm im Sinne einer herkömmlichen Programmiersprache wie z. B. *PASCAL* oder *C* verstanden werden können, sind in occam die für die strukturierte Programmierung notwendigen Konstrukte realisiert. Dazu zählen

- **Sequenz**,
- **Repetition**,
- **Alternative** und
- **Blockbildung**.

Darüber hinaus enthält occam weitere, für die Synchronisation mehrerer solcher Prozesse notwendige Instruktionen. Sie regeln die Prozeßkommunikation und den Kontrollfluß. Im einzelnen sind dies

- die Schreib- und Leseanweisung für einen Kanal, ! und ?, die unabhängig von der Art des verwendeten Kanals sind,
- die Aufforderung, zwei Prozesse nacheinander (**SEQ**) bzw. gleichzeitig (**PAR**) zu starten,
- die Aufforderung, den ersten ausführbaren Prozeß einer Menge von Prozessen zu starten (**ALT**),
- die Anweisung, einen Prozeß zu einem bestimmten Zeitpunkt abzubrechen oder ihn bis dahin zurückzuhalten (**TIMER** in Verbindung mit ? **AFTER**),
- die Möglichkeit, Prozesse zu priorisieren (**PRI ALT** und **PRI PAR**).

Während occam damit die Programmierung von Netzwerken sehr komfortabel macht, hat es jedoch gerade im Kleinen einige ärgerliche Schwächen. So unterstützt es keine strukturierten Datentypen, und weder Verzeigerung von Variablen noch dynamische Speicherverwaltung sind unter occam möglich. Des weiteren erweist sich auch das Konzept, jegliche Anweisung

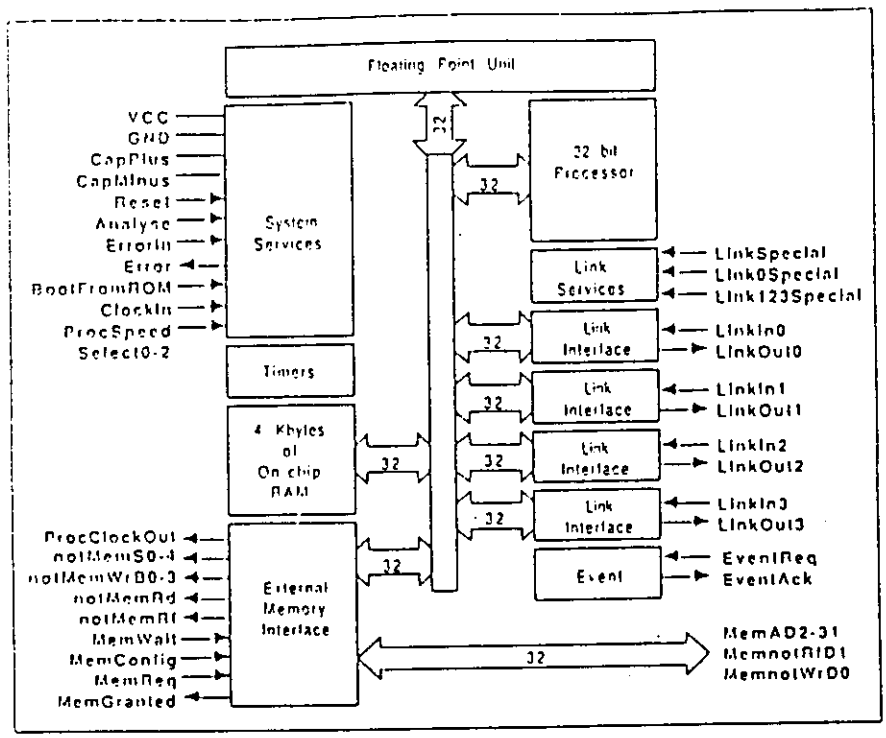

Abbildung A.2: Funktionsgruppen im T800

einem Prozeß gleichzusetzen, gelegentlich als Hindernis. So ist beispielsweise die Prüfung auf das Nichteingetretensein eines Ereignisses nur über Umwege zu realisieren, da sie i. allg. auf einen nicht-terminierenden Leseprozeß führt. Schließlich ist in **occam** — allerdings konzeptbedingt, da in diesem Fall die Anzahl der zu startenden Prozesse nicht a priori bekannt ist — keine rekursive Programmierung möglich [19,20,22].

A.2.3 Der T800

Zur Zeit sind fünf verschiedene Typen von Transputern von *INMOS Ltd.* erhältlich. Sie sind im Befehlscode aufwärts kompatibel und unterscheiden sich im wesentlichen in der Datenpfadbreite, der Arbeitsgeschwindigkeit sowie in Anzahl und Art der auf dem Chip befindlichen Interfaces.

Der IMS T800 ist ein 32-bit CMOS-Mikroprozessor mit 4 kB on-chip RAM, dessen Arbeitsfrequenz über externe Signale zwischen 17.5 MHz und

Typ	CPU-Breite	Interner Speicher	Adreßraum	Links	max. Takt-frequenz	Besonderheiten
T222	16 bit	2 kB	64 kB	4	22.5 MHz	
M212	16 bit	2 kB	64 kB	2	17.5 MHz	Disk-Interface
T414	32 bit	2 kB	4GB	4	20.0 MHz	
T425	32 bit	4 kB	4GB	4	30.0 MHz	
T800	32 bit	4 kB	4GB	4	30.0 MHz	FPU

Tabelle A.1: Derzeit erhältliche Transputertypen

30 MHz wählbar ist. Damit wird eine Rechenleistung von bis zu 15 MIPS[7] erreicht. Der IMS T800-20 ist pinkompatibel zum IMS T414-20 und kann direkt in für diesen entwickelte Schaltungen eingesetzt werden, wo er mit einer Taktfrequenz von 20 MHz läuft.

Auf dem IMS T800 befindet sich eine 64-bit Fließkommaeinheit, die bis zu 1.5 Mflops bei 20 MHz Taktfrequenz leistet (2.25 Mflops bei 30 MHz). Der IMS T800 kann auf einen linearen Adreßraum von 4 GB direkt zugreifen und erreicht bei entsprechendem Speicher Datendurchsatzraten von bis zu 40 MB/s. Neben den einfachen Speicherzugriffsmöglichkeiten verfügt er über mikrocodierte Befehle für Speicher-Blocktransfers.

Für Verbindungen zu anderen Transputern enthält der IMS T800 vier Links, über die mit 5, 10 oder 20 Mbit/s Daten übertragen werden können. Bei Datenübertragung zwischen zwei IMS T800 ist die Wahl der Übertragungsgeschwindigkeit nur durch die Länge des Übertragungsweges begrenzt. Bei bidirektionaler Datenübertragung beträgt der maximale Datendurchsatz 2.35 MB/s [21,22].

A.3 Das 2TP-VME-Modul

Das 2TP-VME-Modul ist ein von NIKHEF-H entwickeltes VME-Modul, das in seiner Struktur den Aufgaben der Datenaufnahme bei wissenschaftlichen Experimenten angepaßt ist. Diese sind

- **Datentransport** von den Meßwertaufnehmern
- **Puffern** von aufgenommenen Daten

[7] Alle in diesem Abschnitt angegebenen Daten sind dem "IMS T800 engineering data"-Buch [21] entnommen.

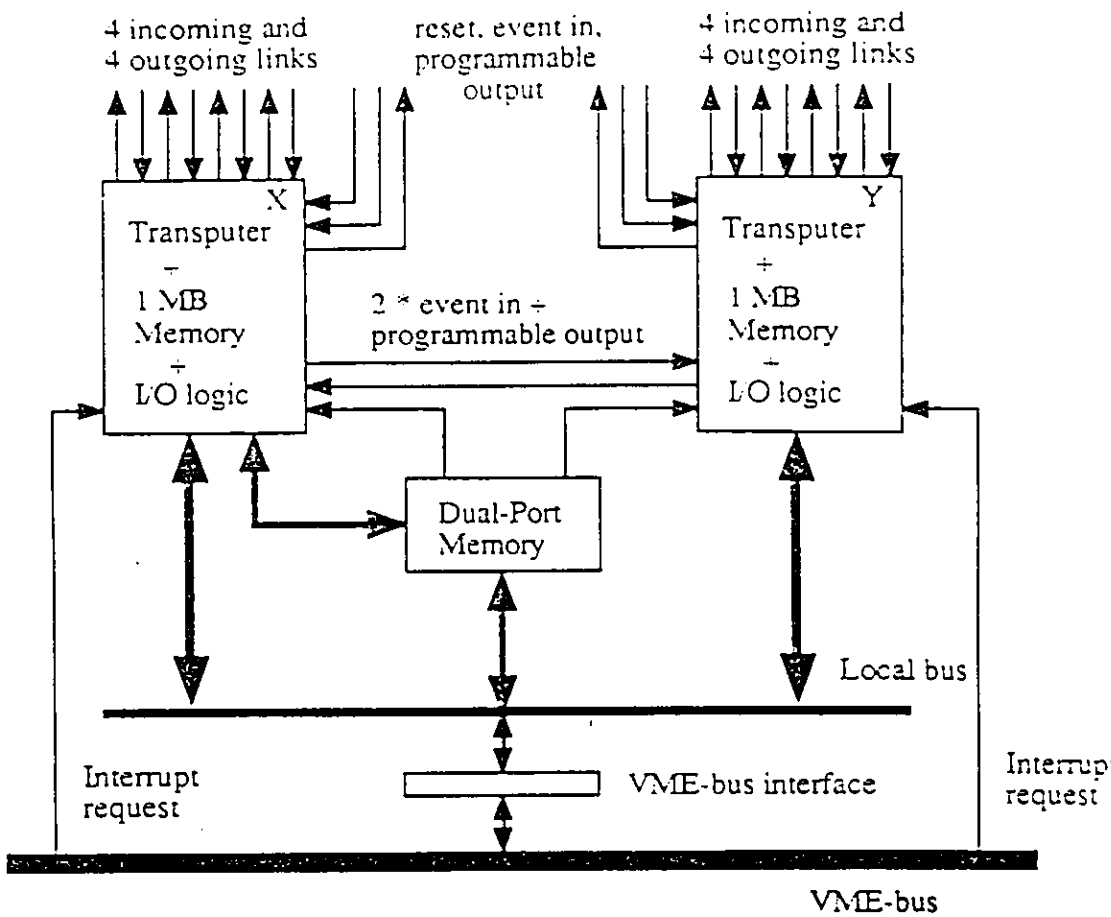

Abbildung A.3: Blockdiagramm des 2TP-VME-Moduls

- Ausführung von **Entscheidungsalgorithmen** über die Verwendbarkeit der gesammelten Daten
- ggf. **Weiterleitung** der gesammelten Daten zur Speicherung und weiteren Verarbeitung

Das Modul beherbergt zwei IMS T800-Transputer, denen bis zu 8 MB lokales DRAM zur Verfügung steht, ein Dual-Ported Memory von 128 kB SRAM und ein Interface zum VMEbus. Die I/O-Leitungen der Transputer sind nach außen geführt und können vom Anwender frei verwendet werden.

Die beiden Transputerumgebungen (X und Y genannt) und das DPM sind über einen lokalen Bus miteinander verbunden. Die Idee bei der Verwendung zweier Transputer ist, einen den Datentransport und die Kommunikation mit der Peripherie abwickeln zu lassen, während der andere Entscheidungsalgorithmen ausführt und somit über die Weiterleitung der Daten bestimmt. In praxi sollen anfallende Daten von Transputer Y aufgenommen und in das DPM geschrieben werden, von wo sie dann von Transputer X zur Verarbeitung gelesen werden. Um einen schnellen und ungestörten Zugriff auf die Daten durch die Entscheidungsalgorithmen sicherzustellen, wurde der Transputerumgebung X eine eigene Schnittstelle zum DPM geschaffen [23].

A.4 TBRun: Ein Testprogramm für die Triggerbox

In diesem Abschnitt ist das bereits in Kap. 3 erwähnte Testprogramm für die Triggerbox abgedruckt. Es läuft auf einem T800-Transputer eines 2Tp-Moduls und wurde vollständig in occam geschrieben. Die von dem Programm aufgerufenen Bibliotheken sind allesamt Bestandteil des Transputer-Development-Systems von *INMOS*.

Die Funktion des Programms hier im einzelnen zu erläutern, würde den Rahmen dieses Anhangs übersteigen. Stattdessen sei zur Illustration ein Bildschirmabdruck des Programms gezeigt: im linken oberen Fenster wird ein Status Report der Box gegeben. Er wird regelmäßig in Abständen von $1\,s$ aufgefrischt, kann aber auch jederzeit über Tastendruck an der Konsole erneuert werden. Die gezeigten Registerinhalte lassen sich über das Terminal manipulieren.

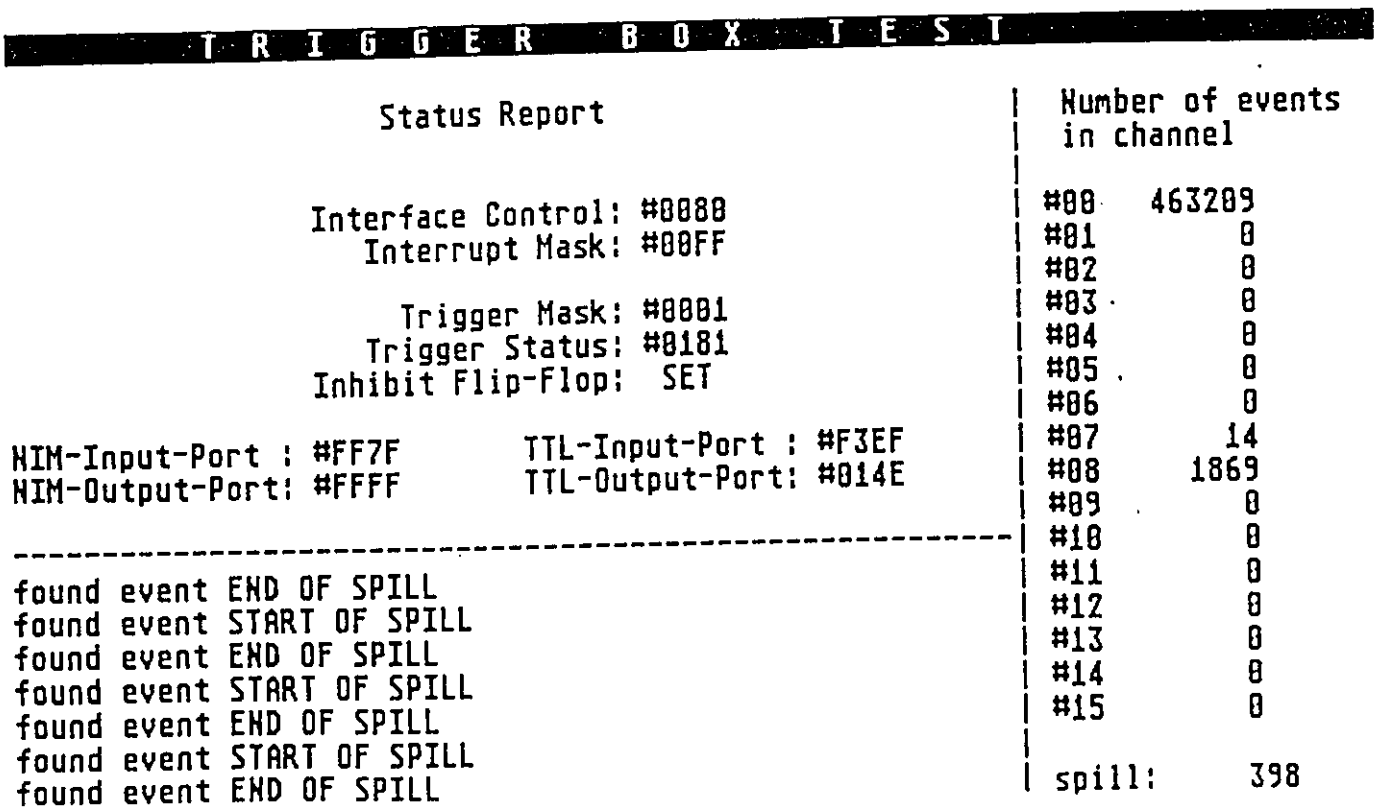

Abbildung A.4: Bildschirmausdruck des Testprogramms TBRun

Das rechte Fenster zeigt die in den Kanälen null bis fünfzehn seit Programmstart erzeugten Trigger und gibt in der untersten Zeile Auskunft über die Anzahl der insgesamt verfolgten spills. Das linke untere Fenster ist schließlich das Meldefenster des Programms: Hier werden alle aufgetretenen spill-Signal, Bus-Fehler und sonstige Fehler angezeigt.

Abbildung A.5: Programmtext des Triggerbox-Testprogramms TBRun

```
PROC tbrun (CHAN OF ANY keyboard, screen)

  #USE "QToccam$LIB:terminal"
  #USE "QToccam$OIR:flibs"

  #INCLUDE "QToccam$LIB:userhdr"

--------------------------------------------------------------------------------
  -- constant definitions: descriptors of trigger-box
  #INCLUDE "SYS$$SYSDEVICE:[HAGGE.INCLUDE]tbval"

  -- constant definitions: descriptors of 2TP-board
  VAL INT TP.STATREG.ADDRESS      IS ((#88000000 >< MOSTNEG INT) >> 2):
  VAL INT TP.INTVECT.ADDRESS      IS ((#5C000000 >< MOSTNEG INT) >> 2):
  VAL INT TP.AMDOREG.ADDRESS      IS ((#94000000 >< MOSTNEG INT) >> 2):

  -- constant definitions: masks for 2TP-board's status-register
  VAL INT TP.STAT.VME.INTERRUPT   IS 1:
  VAL INT TP.STAT.BUS.ERROR       IS 2:

  -- constant definitions: speed up the message-output-process
  VAL [50]BYTE BLANKS             IS
                                                                    ":

  -- make 2TP-board's memory accesible through variables
  CHAN OF ANY event:
  INT   tp.statreg,
        amrO:
  BYTE  tp.intvect.reg:

  PLACE event                     AT 8:
  PLACE tp.statreg                AT TP.STATREG.ADDRESS:
  PLACE tp.intvect.reg            AT TP.INTVECT.ADDRESS:
  PLACE amrO                      AT TP.AMDOREG.ADDRESS:

  -- make trigger box accessible through variables
  [#400]BYTE tb.byte.address.space:
  PLACE      tb.byte.address.space    AT TRIGGERBOX.BASE.ADDRESS:

  BYTE  tb.ctrl.reg      RETYPES tb.byte.address.space [TB.STATREG.INDEX]:
  BYTE  tb.one.shot      RETYPES tb.byte.address.space [TB.GENTRIGGER.INDEX]:
  BYTE  tb.intmask.reg   RETYPES tb.byte.address.space [TB.INTMSKREG.LOW.INDEX]:
  BYTE  tb.trmask.reg.lo RETYPES tb.byte.address.space [TB.TRMASKREG.LOW.INDEX]:
  BYTE  tb.trmask.reg.hi RETYPES tb.byte.address.space [TB.TRMASKREG.HIGH.INDEX]:
  BYTE  tb.trstat.reg.lo RETYPES tb.byte.address.space [TB.TRIGSTAT.LOW.INDEX]:
  BYTE  tb.trstat.reg.hi RETYPES tb.byte.address.space [TB.TRIGSTAT.HIGH.INDEX]:
  BYTE  tb.inhibit.ff    RETYPES tb.byte.address.space [TB.CLRINHIBIT.INDEX]:
  BYTE  tb.nimin.reg.lo  RETYPES tb.byte.address.space [TB.NIMINREG.LOW.INDEX]:
  BYTE  tb.nimin.reg.hi  RETYPES tb.byte.address.space [TB.NIMINREG.HIGH.INDEX]:
  BYTE  tb.nimout.reg.lo RETYPES tb.byte.address.space [TB.NIMOUTREG.LOW.INDEX]:
  BYTE  tb.nimout.reg.hi RETYPES tb.byte.address.space [TB.NIMOUTREG.HIGH.INDEX]:
  BYTE  tb.ttlin.reg.lo  RETYPES tb.byte.address.space [TB.TTLINREG.LOW.INDEX]:
  BYTE  tb.ttlin.reg.hi  RETYPES tb.byte.address.space [TB.TTLINREG.HIGH.INDEX]:
  BYTE  tb.ttlout.reg.lo RETYPES tb.byte.address.space [TB.TTLOUTREG.LOW.INDEX]:
  BYTE  tb.ttlout.reg.hi RETYPES tb.byte.address.space [TB.TTLOUTREG.HIGH.INDEX]:

  -- constant definitions: global variables
  INT   clock.now,                      -- enforce a status-report every second
        tp.status.now,                  -- 2TP's status register
        spill.count,                    -- ...
        count:                          -- dummy for initialization, ...
  BYTE  tp.last.intvect,                -- buffer for VME-interrupt-vector
                                        -- WARNING: reading tp.intvect.reg en-
                                        -- forces an IAck-Cycle on the VMEbus
        dummy.b,                        -- just in case...
        tp.event.in,                    -- dummy for event-pin-readout
        key.pressed:                    -- dummy for keyboard-readout
  BOOL  go,                             -- ensure a proper termination
        spill:                          -- no status reports during spill
  TIMER clock:                          -- enforce a status-report every second

  [8][50] BYTE message:                 -- buffer for message display
  [16]    INT  channel.count:           -- counter for events on channels
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
  PROC display.setup ()

    SEQ
      goto.x.y (screen, 0, 0)
      clr.eos  (screen)

      -- produce reversed headline
      type (screen, "7m          T R I G G E R    B O X    ")
      type (screen, "T  E  S  T                        Om*c*n")

      crlf (screen)
      type (screen, "                      Status Report    ")
      type (screen, "                   ¦  Number of events  ")
      goto.x.y (screen, 59, 3)
      type (screen, "¦  in channel       ")

      -- draw window frames
      SEQ i = 1 FOR 23
        SEQ
          goto.x.y (screen, 59, i)
          type (screen, "¦")
      goto.x.y (screen, 0, 15)
      type (screen, "---------------------------------------")
      type (screen, "-------------------")

      -- create status display mask
      goto.x.y (screen, 16,  5)
      type     (screen, " Interface Control: ")
      goto.x.y (screen, 16,  6)
      type     (screen, "    Interrupt Mask: ")
      goto.x.y (screen, 16,  8)
      type     (screen, "       Trigger Mask: ")
      goto.x.y (screen, 16,  9)
      type     (screen, "    Trigger Status: ")
      goto.x.y (screen, 16, 10)
      type     (screen, " Inhibit Flip-Flop: ")
      goto.x.y (screen,  0, 12)
      type     (screen, "NIM-Input-Port : ")
      goto.x.y (screen, 30, 12)
      type     (screen, "TTL-Input-Port : ")
      goto.x.y (screen,  0, 13)
      type     (screen, "NIM-Output-Port: ")
      goto.x.y (screen, 30, 13)
      type     (screen, "TTL-Output-Port: ")
```

Abbildung A.6: Programmtext des Triggerbox-Testprogramms TBRun (Forts.)

```
    SEQ i = 1 FOR 7
      message [i-1] := message [i]
    message [7] := BLANKS
    [message [7] FROM 0 FOR SIZE new.message] := new.message

    -- refresh display
    SEQ i =  0 FOR 8
      SEQ
        goto.x.y (screen, 0, i+16)
        type     (screen, message [i])
:
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
PROC display.event.count ()

  SEQ
    SEQ i = 0 FOR 16
      SEQ
        goto.x.y (screen, 66, i+5)
        put.int  (screen, channel.count [i], 7)
    goto.x.y (screen, 68, 23)
    put.int (screen, spill.count, 7)
:
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
SEQ
  -- welcome message
  print (screen, "*c*n************ Triggerbox Test **************c*n*n")
  print (screen, "press   O   to fire the oneshot")
  print (screen, "        S   to update the status display")
  print (screen, "        E   to edit the register-settings")
  print (screen, "        C   to clear the inhibit-flipflop")
  print (screen, "        A   to abort the program*c*n*n")

  -- initialize variables
  amr0             := TB.AMR.CODE
  spill.count      := 0
  count            := 0
  go               := TRUE
  spill            := FALSE
  tp.last.intvect  := BYTE #00
  tp.status.now    := (tp.statreg /\ #007F)

  SEQ i = 0 FOR 8
    message [i] := BLANKS
  SEQ i = 0 FOR 16
    channel.count [i] := 0

  type (screen, "Trying to initialize Transputer...*c*n*n")
  type (screen, "Initial TP-Status: ")
  put.int (screen, tp.status.now, 5)
  crlf (screen)

  -- clear up status-register
  -- create channel-count display mask
  SEQ i = 5 FOR 16
    SEQ
      goto.x.y (screen, 61, i)
      type (screen, "#")
      put.int (screen, i-5, 2)
  goto.x.y (screen, 61, 23)
  type (screen, "spill: ")
:
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
PROC display.status ()

  INT dummy.i:                          -- dummy for constructing integers out
                                        -- of two bytes

  SEQ
    goto.x.y    (screen, 36,  5)
    put.int.hex (screen, INT tb.ctrl.reg)
    goto.x.y    (screen, 36,  6)
    put.int.hex (screen, INT tb.intmask.reg)
    dummy.i :=  ( ((INT tb.trmask.reg.hi) << 8) \/ (INT tb.trmask.reg.lo) )
    goto.x.y    (screen, 36,  8)
    put.int.hex (screen, dummy.i)
    dummy.i :=  ( ((INT tb.trstat.reg.hi) << 8) \/ (INT tb.trstat.reg.lo) )
    goto.x.y    (screen, 36,  9)
    put.int.hex (screen, dummy.i)
    goto.x.y    (screen, 36, 10)
    IF
      ( ((INT tb.inhibit.ff) /\ #0001) > 0 )
        type (screen, " SET    ")
      TRUE
        type (screen, " CLEARED")
    dummy.i :=  ( ((INT tb.nimin.reg.hi) << 8) \/ (INT tb.nimin.reg.lo) )
    goto.x.y    (screen, 18, 12)
    put.int.hex (screen, dummy.i)
    dummy.i :=  ( ((INT tb.ttlin.reg.hi) << 8) \/ (INT tb.ttlin.reg.lo) )
    goto.x.y    (screen, 48, 12)
    put.int.hex (screen, dummy.i)
    dummy.i :=  ( ((INT tb.nimout.reg.hi) << 8) \/ (INT tb.nimout.reg.lo) )
    goto.x.y    (screen, 18, 13)
    put.int.hex (screen, dummy.i)
    dummy.i :=  ( ((INT tb.ttlout.reg.hi) << 8) \/ (INT tb.ttlout.reg.lo) )
    goto.x.y    (screen, 48, 13)
    put.int.hex (screen, dummy.i)
:
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
PROC display.message (VAL []BYTE new.message)

  SEQ

    -- scroll up
```

Abbildung A.7: Programmtext des Triggerbox-Testprogramms TBRun (Forts.)

```
WHILE ( ((tp.status.now /\ TP.STAT.VME.INTERRUPT) > 0) AND (count < 42) )
  SEQ
    tp.last.intvect := tp.intvect.reg -- interrupt pending? => !Ack-Cycle
    tp.statreg      := 0              -- SR=0  <=>  still IREQ => next event
    tp.status.now   := (tp.statreg /\ #007F)
                                      -- no more events on the VMEbus?
    count           := count + 1

type    (screen, "*c*nHad to clear Transputer*'s status register ")
put.int (screen, count, 2)
type    (screen, " times...*c*n*n")

-- status register should be cleared now, so deal with power-on interrupts
PRI ALT
  event  ?  tp.event.in               -- IREQ while booting Transputer?
    SEQ
      tp.status.now := (tp.statreg /\ #007F)
                                      -- serve it, but ignore it!
      type    (screen, "===>There was an event pending!!!   (status: ")
      put.int (screen, tp.status.now, 3)
      type    (screen, ", vector: ")
      put.int (screen, INT tp.last.intvect, 3)
      type    (screen, ")*c*n*n")
  TRUE & SKIP
    SKIP

-- general trigger box enable
dummy.b          := tb.ctrl.reg       -- enable interrupter
tb.ctrl.reg      := BYTE ( (INT dummy.b) \/ #0008 )
tb.intmask.reg   := BYTE #E0          -- Spill-Start/-Stop- & Beam-Events
tb.trmask.reg.lo := BYTE #FF          -- enable all channels
tb.trmask.reg.hi := BYTE #FF
tb.inhibit.ff    := BYTE #00          -- enable Master-Trigger

-- wait for key to be pressed, then start test-program
keyboard  ?  key.pressed

-- and here we go!
display.setup ()
display.status ()
display.event.count ()
display.message ("Transputer initialized --- Testprogram running")
```

```
WHILE go
  SEQ
    clock  ?  clock.now

    PRI ALT

      -- keyboard redout is running at high priority
      keyboard  ?  key.pressed
        IF

          -- ABORT TEST?
          (key.pressed = 'a') OR (key.pressed = 'A')
            SEQ
              -- general trigger box disable
              dummy.b          := tb.ctrl.reg
              tb.ctrl.reg      := BYTE ( (INT dummy.b) \/ #0008 )
              tb.intmask.reg   := BYTE #00
              tb.trmask.reg.lo := BYTE #00
              tb.trmask.reg.hi := BYTE #00
              tb.inhibit.ff    := BYTE #01

              -- terminate program
              display.status ()
              display.message (" ")
              goto.x.y (screen, 0, 22)
              go := FALSE

          -- FIRE ONE SHOT?
          (key.pressed = 'o') OR (key.pressed = 'O')
            SEQ
              tb.one.shot := BYTE #01     -- write access => fire monoflop
              display.message ("Computer-Trigger generated...")

          -- UPDATE STATUS-REPORT?
          (key.pressed = 's') OR (key.pressed = 'S')
            display.status ()

          -- CLEAR INHIBIT-FLIPFLOP?
          (key.pressed = 'c') OR (key.pressed = 'C')
            tb.inhibit.ff := BYTE #00

          -- EDIT REGISTER SETTINGS?
          (key.pressed = 'e') OR (key.pressed = 'E')
            INT which.register,
                new.value:
            SEQ
              display.message ("EDIT REGISTER SETTING:")
              display.message (" ")
              display.message ("(1) interrupt mask register")
              display.message ("(2) control register")
              display.message ("(3) trigger mask register")
              display.message ("(4) NIM output port register")
              display.message ("(5) TTL output port register")
              get.int (keyboard, screen, which.register)

              IF
                (which.register > 0) AND (which.register < 6)
                  SEQ
                    display.message (" ")
                    display.message ("Enter new value (hex): ")
                    get.int (keyboard, screen, new.value)

                    IF
                      (which.register = 1)
                        tb.intmask.reg   := BYTE  (new.value /\ #00FF)
                      (which.register = 2)
                        tb.ctrl.reg      := BYTE  (new.value /\ #00FF)
                      (which.register = 3)
                        SEQ
                          tb.trmask.reg.lo := BYTE  (new.value /\ #00FF)
                          tb.trmask.reg.hi :=
                                      BYTE ((new.value /\ #FF00) >> 8)
                      (which.register = 4)
                        SEQ
                          tb.nimout.reg.lo := BYTE  (new.value /\ #00FF)
                          tb.nimout.reg.hi :=
                                      BYTE ((new.value /\ #FF00) >> 8)
                      (which.register = 5)
                        SEQ
                          tb.ttlout.reg.lo := BYTE  (new.value /\ #00FF)
                          tb.ttlout.reg.hi :=
                                      BYTE ((new.value /\ #FF00) >> 8)
                      TRUE            -- you never know...
```

```
                SKIP

          TRUE
            SKIP

        SEQ j = 0 FOR 7          -- brute force to clear display...
          message [j] := BLANKS
        display.message (" ")
        display.status ()

    -- ignore any other key
    TRUE
      SKIP

-- here start the low-priority-processes
ALT

  -- interrupt, bus error or anything like that?
  event  ?  tp.event.in
    SEQ

      -- check status register
      tp.status.now := (tp.statreg /\ #7F)
      IF
        ((tp.status.now /\ TP.STAT.VME.INTERRUPT) > 0)
        -- make sure that you read the interrupt vector only, when
        -- there really was an Interrupt Request, 'cause otherwise
        -- you'll get a bus error which causes another event, but
        -- again no Interrupt Request on the bus...
          tp.last.intvect := tp.intvect.reg
        TRUE
          SKIP
      tp.statreg := 0

      -- evaluate event
      IF
        ((tp.status.now /\ TP.STAT.BUS.ERROR) > 0)
          display.message ("found event BUS ERROR...")


        ((tp.status.now /\ TP.STAT.VME.INTERRUPT) > 0)

          IF
            (tp.last.intvect = (BYTE TB.START.OF.SPILL))
              SEQ
                spill := TRUE
                -- set green LED
                dummy.b := tb.ctrl.reg
                tb.ctrl.reg := (BYTE ((INT dummy.b) \/ #0002))
                display.message ("found event START OF SPILL")
                spill.count := spill.count + 1

            (tp.last.intvect = (BYTE TB.END.OF.SPILL))
              SEQ
                spill := FALSE
                -- clear green LED
                dummy.b := tb.ctrl.reg
                tb.ctrl.reg := (BYTE ((INT dummy.b) /\ #FFFD))
                display.message ("found event END OF SPILL")
                display.event.count ()

            -- this doesn't look very nice, but it has to be both
            -- fast and comfortable - which is sort of excluding
            -- each other, but at least structured programming
            (tp.last.intvect = (BYTE TB.TRIGGER))
              SEQ
                SEQ i = 0 FOR 8
                  SEQ
                    IF
                      ( ((INT tb.trstat.reg.lo) /\ (1 << i)) > 0 )
                        SEQ
                          channel.count [i] := channel.count [i] + 1
                          IF
                            spill
                              SKIP
                            TRUE
                              SEQ
                                goto.x.y (screen, 66, i+5)
                                put.int (screen,
                                         channel.count [i], 7)
                      TRUE
                        SKIP
                    IF
                      ( ((INT tb.trstat.reg.hi) /\ (1 << i)) > 0 )
                        SEQ
                          channel.count [i+8] := channel.count [i+8]+1
                          IF
                            spill
                              SKIP
                            TRUE
                              SEQ
                                goto.x.y (screen, 66, i+13)
                                put.int (screen,
                                         channel.count [i+8], 7)
                      TRUE
                        SKIP

                -- rearm trigger box
                tb.inhibit.ff := BYTE #00

            -- tp.last.intvect-evaluation
            TRUE
              SKIP

        --tp.status-evaluation
        TRUE
          SEQ
            display.message ("Found unknown event...")
            clock  ?  clock.now

    (spill = FALSE) & clock  ?  AFTER clock.now + 15625
      display.status ()
```

Abbildung A.8: Programmtext des Triggerbox-Testprogramms TBRun (Forts.)

Anhang B

Familien integrierter Schaltkreise

Bei der Diskussion des Entwurfs der Triggerbox ist bereits darauf hingewiesen worden, daß es bei der einer Schaltung zugrunde liegenden Schaltkreistechnologie einer sorgfältigen Wahl bedarf. Die dort angegebene Tabelle 3.3.1 stellte die wichtigsten elektrischen Parameter der drei gängigsten Schaltkreisfamilien einander gegenüber. An dieser Stelle soll nun zur Erläuterung dieser Werte ein kurzer Überblick über die Funktionsweise dieser Schaltkreise gegeben werden.

B.1 TTL

Die TTL-Technik ist die älteste Technik standardisierter integrierter Schaltkreise überhaupt. Sie basiert auf dem 1959/60 von J. Kilby und J. Hoerni entwickelten Planarprozeß. Serienmäßig gefertigte TTL-ICs sind seit 1963 auf dem Markt. Heute ist TTL die am weitesten verbreitete Schaltkreisfamilie, und ihre Signalpegel sind Standard in der Computertechnik. Ihr Charakteristikum ist, daß sowohl die Eingangs- als auch Ausgangsstufen der einzelnen Elemente transistorisiert sind, eine Tatsache, die dieser Familie letztlich auch ihren Namen gab (TTL: Transistor-Transistor-Logic). Abb. B.1 zeigt die TTL-Grundschaltung. ein NAND-Gatter.

In der TTL-Technik werden die Transistoren als Schalter verwendet, was bedeutet, daß sie im Übersteuerungsbereich ihrer Kennlinie arbeiten. Übersteuerung eines Transistors tritt immer dann ein, wenn seine

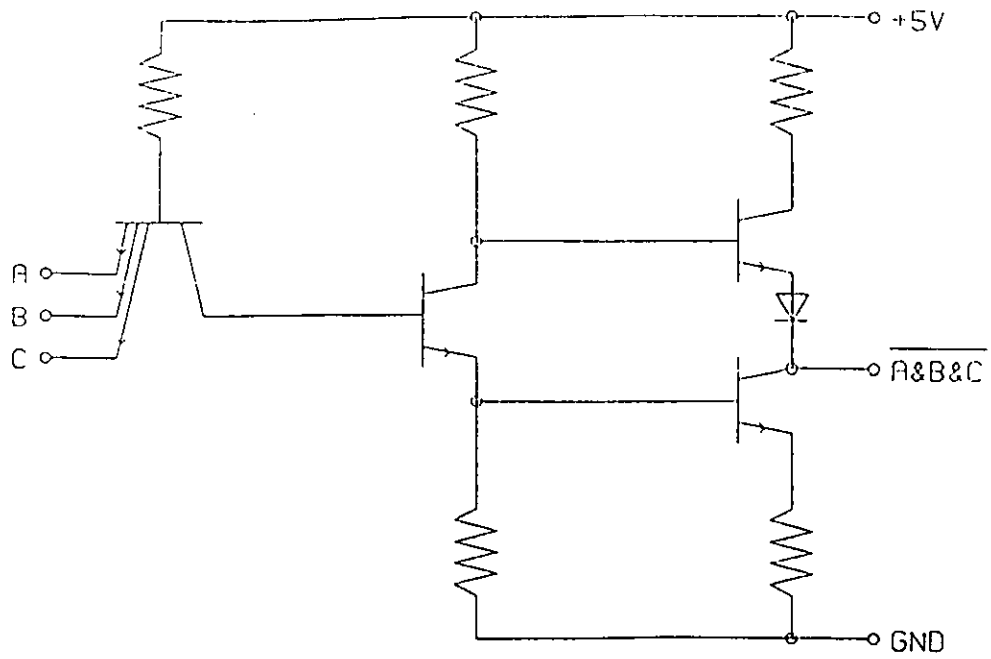

Abbildung B.1: NAND-Gatter mit drei Eingängen in integrierter TTL-Technik

Kollektordiode leitend wird. In diesem Falle liefert — anders, als im Normalbetrieb — auch der Kollektor Ladungsträger in die Basis. Dadurch entsteht in der Basis ein "Speicherladung" genannter zusätzlicher Ladungsträgerüberschuß. Dieser Effekt wirkt sich störend auf den Umschaltvorgang aus: da die Speicherladung dabei jedesmal auf- oder abgebaut werden muß, verzögert sich die Umschaltzeit entsprechend.

Im Laufe der Jahre wurde die TTL-Technik weiterentwickelt und entscheidend verbessert. Die beiden wichtigsten Errungenschaften sind dabei die Vergrößerung der internen Widerstände und die Verwendung von Schottky-Transistoren. Erstgenanntes reduziert den Leistungsbedarf der Schaltkreise, allerdings auf Kosten erhöhter Schaltverzögerungen. In Schottky-Transistoren kann dagegen die Kollektordiode nicht leitend werden, so daß hier eine Verkürzung der Schaltzeiten erreicht werden kann. Kombiniert man beide Effekte (Low-power-Schottky–TTL), so erreicht man die Schaltzeiten von Standard-TTL bei nur etwa 20% des dortigen Leistungsaufwands.

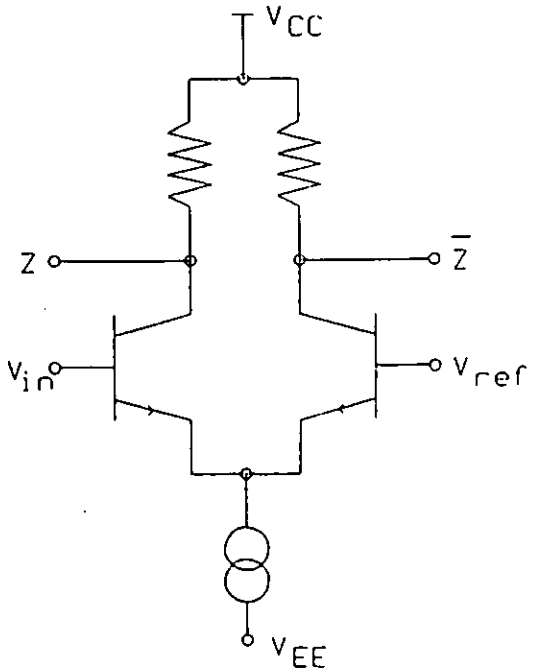

Abbildung B.2: Differenzverstärker als Grundschaltung der ECL- Schaltkreise

B.2 ECL

Der TTL-Technik entgegen steht die fast ebenso alte ECL-Technik (ECL: Emitter Coupled Logic). Sie basiert ebenfalls auf dem Planarprozeß. verwendet die Transistoren jedoch im linearen Bereich ihrer Kennlinie. Grundschaltung der ECL ist der in Abb. B.2 dargestellte Stromschalter: je nachdem, ob die Eingangsspannung V_{in} größer oder kleiner als die interne Referenzspannung V_{ref} ist, wird der von der Konstantstromquelle erzeugte Strom durch den linken oder rechten Zweig der Schaltung geleitet. Die Signalpegel der ECL-Technik sind mit den TTL-Pegeln nicht kompatibel.

Da die Transistoren nicht in den Übersteuerungsbereich ihrer Kennlinie kommen, treten bei ECL keine Speicherladungseffekte auf. und die ECL-Technik ist erheblich schneller als ihr TTL-Pendant. Dieser Geschwindigkeitsvorteil läßt sich allerdings, da er auf dem permanenten Stromfluß in der Schaltung beruht, nur auf Kosten eines erhöhten Leistungsbedarfs erreichen.

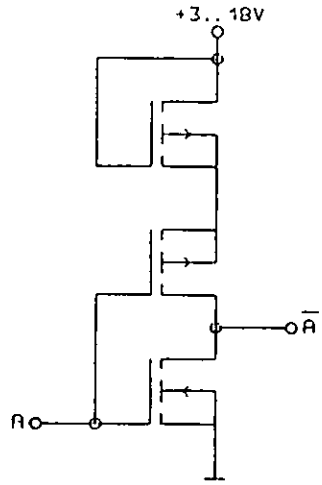

Abbildung B.3: Inverter in CMOS-Technik

B.3 CMOS

Bei den Anfang der 70er Jahre auf dem Markt erschienenen CMOS[1]-Bausteinen handelt es sich um Produkte einer gänzlich anders gearteten Technologie. Ihnen liegt nicht mehr der Planarprozeß zugrunde, sondern vielmehr eine im Namen bereits ausgedrückte Schichtung von Metall, Siliziumoxid und Halbleitermaterial, in die entsprechend vorher gefertigter Masken Leiterbahnen geätzt sind. Basiselement der CMOS-Technologie ist der Feldeffekt-Transistor, die CMOS-Grundschaltung — ein Inverter — ist in Abb. B.3 gezeigt.

Feldeffekttransistoren schalten durch die Wirkung eines durch eine externe Spannung entstehenden Feldes. Dementsprechend sind CMOS-Bausteine sehr träge, was Schaltzeiten anbelangt. Da der Schaltvorgang jedoch mit nahezu keinem Stromfluß verbunden ist, ist die CMOS-Technologie dieje-

[1] CMOS steht für "Complementary Metal Oxide Semiconductor"; dabei geben die letzten drei Worte die Materialschichtenfolge der Bauelemente an, während das erste darauf hinweist, daß die Ausgangsstufe eines Elements FETs beider Polaritäten enthält, die so geschaltet sind, daß immer einer von ihnen leitet (dadurch werden Ausgangskapazitäten stets von einem leitenden Transistor umgeladen und somit die Schaltgeschwindigkeiten erhöht).

nige mit der mit Abstand geringsten Verlustleistung[2] (Dies gilt nicht mehr bei hohen Schaltfrequenzen, da die Verlustleistung proportional mit der Schaltfrequenz wächst — vgl. Tab. 3.1). Die Signalpegel der CMOS-Technik sind nicht in so engem Rahmen wie bei TTL oder ECL festgelegt, sondern in einem großen Bereich wählbar. Sie sind TTL-aufwärtskompatibel, nicht aber ECL-kompatibel.

[2] Da in der CMOS-Technologie Widerstände, Kondensatoren und Transistoren identisch aufgebaut sind — ein Widerstand wird als FET mit konstant vorgespanntem Gate gebaut, bei Kondensatoren werden Source und Drain eines FET verbunden und seine Sperrschichtkapazität ausgenutzt — ist es auch die mit Abstand billigste Schaltkreistechnologie.

Anhang C

Technische Dokumentation der Triggerbox

In der Arbeit wurde an mehreren Stellen darauf hingewiesen, daß die Schaltpläne der Triggerbox im Anhang zu finden wären. Im Original der Arbeit befindet sich an dieser Stelle eine 42-seitige technische Dokumentation, die sowohl die Schaltpläne als auch detaillierte Hinweise zum Betrieb der Triggerbox enthält. Aus Kostengründen ist die Dokumentation in diesem Nachdruck nicht enthalten. Sie kann allerdings im Original der Arbeit in der Bibliothek des Deutschen Elektronen-Synchrotrons eingesehen werden.

Anhang D

Zukünftige Triggerboxen

Die in dieser Arbeit beschriebene Triggerbox ist zum Zeitpunkt des Entstehens der Arbeit bereits seit einem halben Jahr im Einsatz. Ihr Konzept hat sich bewährt, und das Ziel, experimentelle Gegebenheiten mit einem Datennahme- und Kontrollsystem zu synchronisieren und dabei gleichzeitig verschiedenartige Messungen an einem Objekt mit nur einem System zu ermöglichen, ist erreicht worden. Ein in Kürze anlaufendes, ähnlich dem in dieser Arbeit beschriebenen Teststand angelegtes Experiment wird mit einer baugleichen Triggerbox ausgestattet werden.

Betrachtet man die Integration der Triggerbox in das System, so fällt ein Punkt auf, an dem sich die Triggerbox noch sinnvoll erweitern ließe: die zwischen Triggerzählern und Triggerbox befindliche Logik, über die in dem in dieser Arbeit beschriebenen Experiment die Teilchenidentifikation durchgeführt wurde. Wie aus Abb. 2.3 hervorgeht, ist für jede Triggerbedingung $(\epsilon, \mu, \ldots)$ die entsprechende logische Schaltung aufzubauen, und die gesamte Schaltung ist starr, d.h. unflexibel und für jede zusätzliche Bedingung zu erweitern.

Ließe man diese "Identifikationslogik" einfach entfallen und führte stattdessen die von den Triggerzählern kommenden Signale direkt an die Triggerbox, um dann per Software über Analyse des Statusregisters die Verknüpfung der von den Zählern kommenden Signale durchzuführen, so würde der angeschlossene Computer u. U. mit Interrupt-Anforderungsraten von einigen hundert kHz konfrontiert werden[1], einer Rate, die gängige Computer nicht

[1] Dies wäre z. B. der Fall, wenn *B1* oder *B2* mit transparenten Eingängen der Triggerbox verbunden wären.

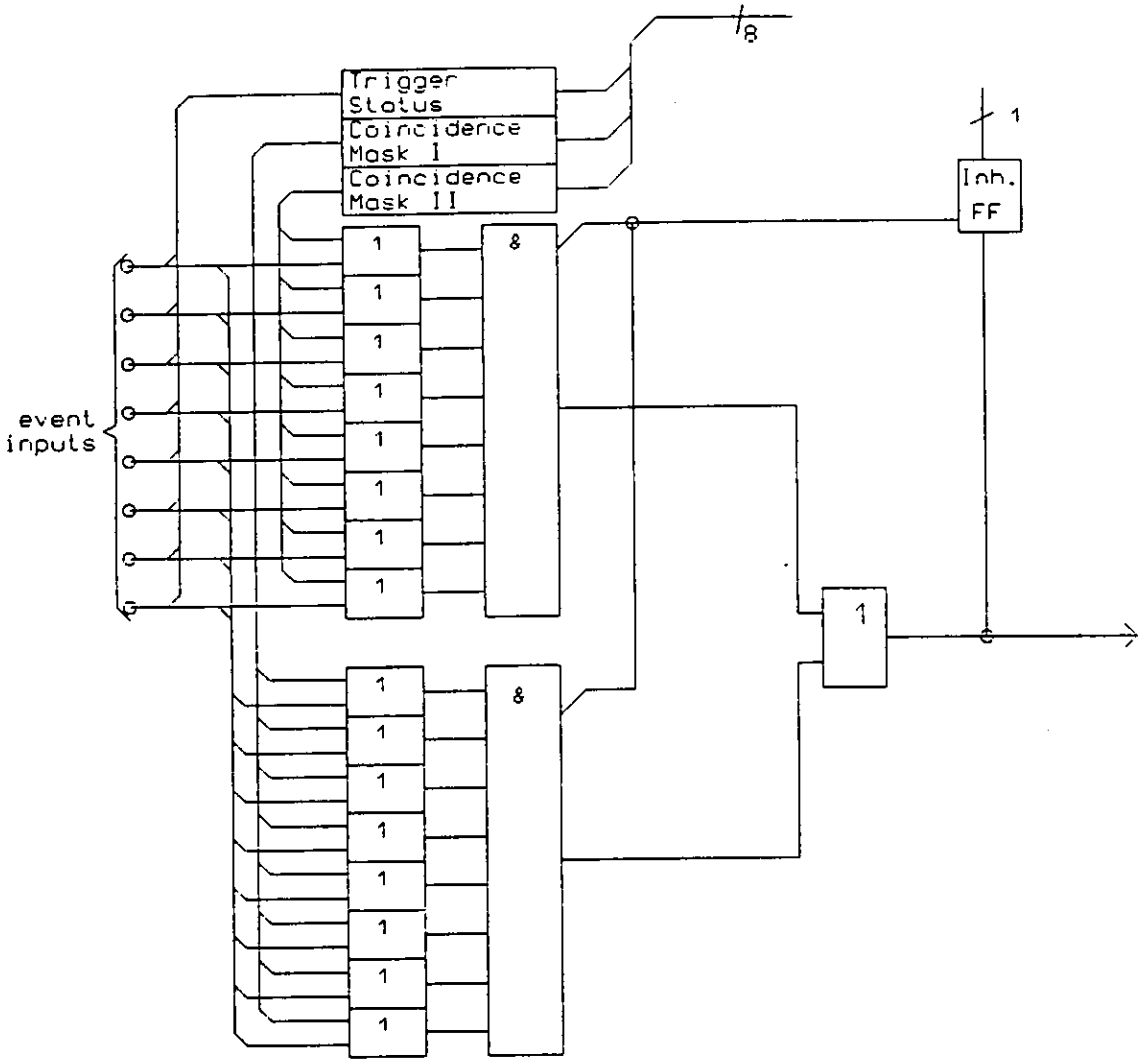

Abbildung D.1: Möglicher Aufbau einer zukünftigen Triggerbox

sinnvoll verarbeiten können. Selbst wenn nur der Eingang der Triggerbox transparent gemacht würde, an dem die geringste Triggerrate erwartet wird, führte der erforderliche zusätzliche Rechenaufwand für die Analyse des Statusregisters zu einer Senkung der maximal vom System verarbeitbaren Triggerrate.

Eine Möglichkeit, durch eine kleine Erweiterung der Triggerbox die starre, externe Identifikationslogik zu vermeiden, ist in Abb. D.1 angegeben. Sie ist als Ersatz für das in Abb. 3.2 gezeigte Blockschaltbild des Master-Or und seiner Umgebung gedacht[2]. Das Trigger-Status-Register,

[2] Allerdings ist die Darstellung hier aus Gründen der Übersichtlichkeit in positiver Logik

das Inhibit-Flip-Flop und das Master-Or selbst sind dabei unverändert geblieben, lediglich die Eingangstore sind um eine Koinzidenzmaske erweitert worden. Da diese die Funktion des entsprechenden bits der Triggermaske mit übernehmen kann, ist das Triggermaskenregister entfallen. Außerdem reduziert sich wegen der größeren Flexibilität der Eingangsstufe die Breite des Master-Or auf die maximale Anzahl unabhängiger Ereignisse, auf die getriggert werden soll (im Falle der Abb. ist diese zu zwei angenommen).

Was leisten nun die Koinzidenzmasken? Löscht man eines ihrer bits, so erzeugt ein an dem entsprechenden Eingang einlaufendes Signal eine logische "1" an dem den Eingangstoren folgenden UND-Gatter, setzt man es dagegen, so liegt der zugehörige Eingang des UND immer auf "1". Nur, wenn alle Eingänge des UND-Gatters auf "1" liegen, d.h. wenn an allen gelöschten bits der Koinzidenzmaske entsprechenden Eingängen gleichzeitig Signale einlaufen, wird der Ausgang des UND aktiv und erzeugt somit über das Master-Or einen globalen Trigger-Puls. Dies entspricht gerade der Funktion, die die "Teilchenidentifikationslogik" des beschriebenen Experiments ausübt. Im Gegensatz zu ihr können hier jedoch die Verknüpfungen der Triggersignale einfach durch entsprechendes Überschreiben des Wertes der Koinzidenzmaske verändert werden[3]!

Verwendet man diese Methode zur Verknüpfung der Triggersignale, so erhält man zwei wesentliche Vorteile: zum einen verschwindet die Identifikationslogik aus dem Experimentaufbau, wodurch einige Elektronikmodule eingespart und die Übersichtlichkeit der Verkabelung erhöht werden können, zum anderen ist die Propagation der Triggersignale zeitkritisch und muß auf Bruchteile von Nanosekunden genau abgeglichen werden[4]. Im Falle der diskret aufgebauten Identifikationslogik ist dieser Abgleich für jede realisierte Verknüpfung erneut durchzuführen, bei der hier vorgestellten Version dagegen für jeden Triggerzähler nur genau einmal. Integriert man sog. "programmable delay chips" in die Pfade der von den Triggerzählern

---

und für nur acht Eingangskanäle erfolgt

[3]Signale wie die des "Veto-Counters" $B4$ würden dabei zuvor invertiert werden müssen. Ob dies in einem externen Inverter, oder aber in einer speziellen Eingangsstufe der Triggerbox geschieht, sei hier dahingestellt

[4]Insbesondere muß dabei sichergestellt sein, daß die von den einzelnen Triggerzählern kommenden Signale GLEICHZEITIG an den jeweiligen sie verknüpfenden Bauteilen ankommen.

herrührenden Signale, so wird dieser Abgleich noch wesentlich vereinfacht, da er dadurch in die Software ausgelagert ist und somit keinerlei Hardware-Modifikationen (wie z. B. einfügen von Verzögerungsleitungen,...) mehr erfordert.

Anhang E

Glossar

Adreßbus — Gesamtheit der Adreßleitungen eines Systems. Über Adreßleitungen können verschiedene Komponenten eines Systems selektiert werden.

asynchroner Zähler — Grundschaltung der Digitalelektronik, die an einem Eingang auftretende Flanken zählt und ihrern Zählerstand in binärer Form zur Verfügung stellt. "Asynchron" bedeutet dabei, daß eine jede Stelle des Zählers erst dann ihren endgültigen Wert annimmt, wenn die nächst-niederwertige dies getan hat. Die unterste Stelle erhält ihren Wert bei Eintreffen des Zählimpulses, die folgenden Stellen um die Signaldurchlaufzeiten ihrer Vorgänger später. Im Jargon werden asynchrone Zähler auch "ripple-counter" genannt.

bit — Abk. f. "binary digit", eine "Binärziffer".

Bus — Sammlung von Signalleitungen.

CMOS — Abk. f. "Complementary Metal Oxide Semidonductor". Bezeichnet eine Schaltkreisfamilie, vgl. Anh. B.

CPU — Abk. f. "Central Processing Unit"

DAQ Abk. f. "Data AcQuisition"

Datenbus Gesamtheit der Datenleitungen eines Systems.

debugging Jargon für Fehlerbeseitigung. "Bug" ist im Zusammenhang mit Computern inzwischen ein gängiger Ausdruck für Fehler geworden.

DQM Abk. f. "Data Quality Monitoring"

DRAM Abk. f. "Dynamic Random Access Memory"; dynamische RAMs unterscheiden in ihrem internen Aufbau, ihrer Ansteuerung durch externe Bauelemente, ihrer Speicherkapazität pro Chip und vor allem ihrer Geschwindigkeit von statischen RAMs.

DPM Abk. f. "Dual Ported Memory". Speicherbausteine haben üblicherweise ein Interface, über das ihre Speicherinhalte gelesen und modifiziert werden können. DPMs haben dagegen zwei voneinander vollständig unabhängige Schnittstellen, so daß zwei Systeme gleichzeitig, d. h. ohne sich gegenseitig zu stören und ohne synchronisiert werden zu müssen, Zugriff auf dessen Daten haben.

ECL Abk. f. "Emitter Coupled Logic". Bezeichnet eine Schaltkreisfamilie, vgl. Anh. B.

Flip-Flop Grundschaltung der Digitalelektronik, die zwei verschiedene Zustände annehmen und speichern kann. Flip-Flops können entweder durch Aktivieren von Steuereingängen (asynchron) oder durch Erzeugen einer Flanke an einem speziellen Takteingang unter zeitgleicher Auswertung des Pegels eines Dateneinganges (synchron) in einen Zustand gezwungen werden.

Hardware Gesamtheit der von einem Benutzer nicht veränderbaren Teile eines Systems (Bauelemente, Grundplatten....).

IAck Signal, mit dem ein Rechner einem Interrupt-Requester die Bearbeitung dessen Unterbrechungsanforderung anzeigt (vgl. Interrupt)

Interface s. Schnittstelle

Interrupt Interrupts bezeichnen die Möglichkeit, einen Rechner bei der Ausführung eines Programms zu unterbrechen, um zwischendurch die Abarbeitung einer anderen Routine zu fordern. Die auszuführende "Interrupt-Routine" wird i. allg. vom Anforderer (Interrupt REQuester) spezifiziert, muß aber dem unterbrochenen Rechner bekannt sein. Sind Interrupts priorisiert, so wird bei der Aufforderung zur Programmunterbrechung dem Rechner eine Information über die Dringlichkeit der Anforderung übermittelt. Nur, wenn das gerade in Ausführung befindliche Programm von geringerer Wichtigkeit ist als die Unterbrechungsanforderung, wird dieser sofort nachgekommen.

I/O Abkürzung für "Input/Output"; wird zumeist als Bezeichnung für die Gesamtheit der Eingangs- und Ausgangsports eines Systems verwendet.

IRQ Signal, das eine Unterbrechungsanforderung einem Rechner übermittelt (vgl. Interrupt).

message Bestandteil eines Protokolls

MFlops Abk. f. "Million Floating-Point operations per second"

Mips Abk. f. "Million instructions per second"

negative Logik — Umkehrung der Bedeutung der Signalpegel (vgl. TTL und NIM)

Kanal — Verbindung zwischen zwei Systemen, wobei lediglich ihre Existenz angenommen ist, aber nichts über die Realisierung der Verbindung ausgesagt wird (sowohl eine von zwei Programmen gemeinsam benutzte Variable als auch ein Kabel zwischen zwei Elektronik-Moduln kann als Kanal bezeichnet werden).

NIM — Abk. f. "Nuclear Instrumentation Module", ein in der Instrumentierung kern- und hochenergiephysikalischer Experimente verbreiteter mechanischer und elektrischer Standard (AEC-Report Nr. TID-20893). Signalpegel: 0 V entsprechen inaktiven Signalen, 0.8 V an 50 Ω aktiven.

NOR — Grundschaltung der Digitalelektronik, die ihren Ausgang nur dann aktiviert, wenn ihre beiden Eingänge inaktiv sind.

ODER — Grundschaltung der Digitalelektronik, die bei Aktivierung eines ihrer Eingänge ihren Ausgang aktiviert.

Pegelkonverter — Schaltung, die die Pegel einer Schaltkreisfamilie unter Erhaltung ihrer logischen Bedeutung in die einer anderen wandelt.

Port — Register mit vorgeschalteter Eingangsstufe (Eingangsport) oder nachgeschalteten Ausgangstreibern (Ausgangsport).

Protokoll — Format, in dem Daten zwischen zwei Systemen ausgetauscht werden. Nur, wenn beide Systeme sich an das gleiche Protokoll halten, kann gewährleistet werden, daß die von einer

Seite gesendeten Daten auf der anderen Seite auch empfangen und verstanden werden können.

Pufferspeicher Will ein System Daten an einen Kommunikationspartner übertragen, ohne daß sichergestellt ist, daß dieser schon empfangsbereit ist, so kann der Sender die Daten zunächst in einen "Pufferspeicher" genannten, zwischen Sender und Empfänger liegenden Speicher schreiben, welcher die Daten dann bis zur Abholung durch den Empfangsprozeß festhält. Auf diese Weise wird verhindert, daß ein Prozeß bei der Datenübertragung auf seinen Kommunikationspartner warten muß.

Register Typischerweise 8 *bit* (Byte), 16 *bit* (Wort) oder 32 *bit* (Langwort) breite Ansammlung von Flip-Flops, die die elementare Speicherzelle eines Computersystems darstellt. Die Flip-Flops eines Registers können nur in ihrer Gesamtheit manipuliert werden.

Registerebene Abstraktionsebene der Schaltkreisbeschreibung (vgl. Kap. 3.2.2).

RT-Level s. Registerebene

Schnittstelle Physikalischer und/oder logischer Übergang zwischen zwei Komponenten.

Signallaufzeit Zeitspanne zwischen dem Eintreffen eines Signals an einem Eingang einer Schaltung und dem Erscheinen der Signalantwort am entsprechenden Schaltungsausgang.

Software Gesamtheit der vom Benutzer an einem System während seines Einsatzes veränderbaren Teile (Computerprogramme,...).

Systemumgebung Gesamtheit der Systeme und Komponenten, mit denen ein System in Verbindung steht. Im Falle einer Schaltung können dies genauso weitere, mit ihr verbundene Schaltungen sein wie auch Programme, die ihre Steuerung übernehmen...

Trigger Funktionseinheit, die den Zustand eines Systems verfolgt und bei Eintreten eines definierten, vorher festgelegten Zustandes ein Signal erzeugt.

TTL Abk. f. "Transistor-Transistor-Logik" (vgl. Anh. B). Aktive Signalleitungen führen einen Pegel von $5\,V$, inaktive liegen auf Masse *(GND)*.

UND Grundschaltung der Digitalelektronik, die nur bei Aktivierung aller ihrer Eingänge ihren Ausgang aktiviert.

VMEbus In der Industrie gebräuchliche und inzwischen bewährte Norm für ein Bussystem, das die Kommunikation mehrerer Rechner untereinander regelt (vgl. Kap. A.1).

VLSI Abk. f. "Very Large Scale Integration"; bezeichnet die vierte Generation integrierter Schaltkreise.

Danksagung

Während meiner Diplomarbeit habe ich mit vielen Kollegen der *ZEUS*-Kollaboration und des DESY zusammengearbeitet. Für das angenehme Betriebsklima und die erhaltene gute Unterstützung möchte ich mich an dieser Stelle bedanken.

Mein besonderer Dank gilt dabei Herrn Prof. Dr. E. Lohrmann, der mir eine überaus abwechslungsreiche und interessante Aufgabe übertrug und deren zügige Bewältigung ermöglichte.

Herrn Dr. W. Vogel danke ich für eine hervorragende Betreuung und viele nützliche Hinweise in punkto Schaltungsentwurf und -test. Herrn Dr. R. Klanner danke ich für die umfassende Einführung in die Funktionsweise des Kalorimeters und des Kalibrationsaufbaus. Schließliche danke ich T. Woeniger für seine Geduld bei den Versuchen, mich mit den Geheimnissen der DESY-IBM und ihrer Programmierung in FORTRAN vertraut zu machen.

Erklärung

Hierdurch versichere ich, die vorliegende Arbeit selbständig unter Verwendung der angegebenen Literatur angefertigt zu haben.

Hamburg, im April 1990, Lars Hagge.

Literaturverzeichnis

[1] The *ZEUS* Detector, Status Report 1989
ZEUS Kollaboration, März 1989

[2] R. Wigmans, NIM A262(1987)243

[3] T. Akesson et al., NIM A262(1987)243

[4] G. d'Agostini et al., NIM A274(1989)134

[5] U. Behrens et al., DESY 89-128(1989)

[6] R. Klanner, NIM A265(1988)200

[7] A. Andresen et al., DESY 89-149(1989)

[8] K. Dierks, DESY F35-90-01(1990)

[9] W. Buttler et al., NIM A277(1989)217

[10] L. Hagge, Studienarbeit am FB Informatik (ANT)
Universität Hamburg 1989

[11] R. W. Hartenstein: Fundamentals of Structured Hardware Design
North-Holland Publ. Co., Amsteram, New York, Oxford 1977

[12] H. Boterenbrood et al., *ZEUS*-Note 90-10

[13] E. Ros, T. Tsurugai, interne Mitteilung

[14] H. Zeltwanger, VMEbus 1(1990)6

[15] The VMEbus Specification, ANSI/IEEE Std. 1014-1987, IEC 82)

[16] H. Strass, Design&Elektronik 14(1986)98

[17] C. A. R. Hoare: Communicating Sequential Processes
Prentice Hall International, London 1985

[18] Proc. 7. Entwicklerforum "Transputer" (1989)143
München 1989

[19] R. Steinmetz: Occam2 - Die Programmiersprache für parallele Verarbeitung
Dr. Alfred Hüthig, Heidelberg 1988

[20] A. Schütte: Occam2 Handbuch
Teubner, Stuttgart 1988

[21] The Transputer Data Book, 2nd. Edition
Inmos Ltd., Bristol 1989

[22] J. C. Vermeulen: Transputer - Experience and Practice I
NIKHEF-H/88, Amsterdam 1988

[23] H. Boterenbrood: The ZEUS Two-Transputer-VME-Module - An Overview
NIKHEF-H/88, Amsterdam 1988