Entwicklung eines schnellen Schauererkennungsprozessors als 2.Triggerstufe für TAPS

II.Physikalisches Institut

Justus-Liebig-Universität

Gießen

Juli 1997

Diplomarbeit

von

Stefan Plüschke aus Friedberg / Hessen

Vorwort und Zusammenfassung

In der vorliegenden Arbeit soll die Entwicklung eines Prototyps einer VME-basierten 2. Triggerstufe für den TAPS-Detektor vorgestellt werden, die Experimente in Zusammenarbeit mit dem HADES-Spektrometer erlauben soll.
Dabei werden in einem einleitenden Kapitel zunächst die experimentellen und technischen Rahmenbedingungen erläutert. Dazu werden die beiden Spektrometer HADES und TAPS vorgestellt. Daran anschließend werden die wesentlichen Voraussetzungen, Eckdaten und Richtlinien für die Hardwareentwicklung aus dem bisher dargelegten erarbeitet. Darauf aufbauend lassen sich nun verschiedene Lösungskonzepte diskutieren, welche kurz zusammengefaßt werden sollen.
Im Anschluß an diese Einleitung wird das in dieser Arbeit verwirklichte Konzepte detailliert erläutert und schließlich seine Umsetzung dargestellt. In diesem Zusammenhang wird auch auf einige technische Details der verwendeten Hard- und Software eingegangen. Im besonderen werden die zur Umsetzung des Konzeptes verwendeten programmierbaren Bausteine (PLD), es handelt sich dabei zum einen um FPGAs der Firma XILINX sowie CPLDs der Firma LATTICE, vorgestellt. Außerdem werden die Techniken und Methoden, die bei der Umsetzung verwendet wurden, kurz geschildert. Den Hauptteil dieses Kapitels wird dann die Beschreibung der verwendeten Algorithmen und ihre Implementation in den unterschiedlichen PLDs einnehmen.
Anschließend werden die Ergebnisse der durchgeführten Tests wiedergegeben und erläutert. In diesem abschließenden Kapitel soll letztlich ein Ausblick auf die weitere Entwicklung und Fertigstellung eines endgültigen Schauererkennungsprozessors gegeben werden.
An dieser Stelle soll nicht zuletzt auf den Anhang verwiesen werden. Dort finden sich unter anderem Auszüge aus den Programmquelltexten der im Test verwendeten Programme. Außerdem sind dort einige ABEL-Sourcen für die einzelnen Funktionsgruppen zu finden.

Gießen, den 24.07.97 Stefan Plüschke

Inhaltsverzeichnis

1 Motivation

1.1 Physikalische Ziele des HADES-Spektrometers

Das Dileptonen-Spektrometer HADES[1] ist ein in der Aufbauphase befindlicher Detektor für e^+e^- Paare am Schwerionensynchrotron SIS der GSI in Darmstadt, dabei steht HADES für „**H**igh-**A**cceptance **Di**-**E**lectron **S**pectrometer“. Das HADES-Spektrometer soll dort als „Experiment der 2. Generation“ die „In-Medium-Eigenschaften“ (Modifikationen von Massen und Lebensdauern) leichter Vektormesonen (ρ,ω,ϕ) in Abhängigkeit von der Kernmateriedichte untersuchen. Man hofft, mit den so gewonnen Daten weiteren Aufschluß über die Zustandsgleichung der Kernmaterie zu erhalten. Außerdem sind Untersuchungen zu elektromagnetischen Formfaktoren von Mesonen und Baryonen geplant.

Als einzigen experimentellen Zugang zum Nachweis der „In-Medium-Modifikationen“ bei hohen Baryonendichten bieten sich Schwerionenkollisionen an, da hier wie z.B. bei SIS-Energien Baryonendichten von bis zu 3 ρ_0 ($\rho_0 = 0.17$ fm^{-3}) und Temperaturen von annähernd 100 MeV auf einer Zeitskala von ca. 10 fm/c erreicht werden[2].

Meson	Masse [MeV/c^2]	Breite [MeV/c^2]	c t [fm]	Dominanter Zerfall	Verzweigungs-verhältnis e^+e^-/hadr. Zerfällen
ρ	768	152	1.3	$\pi\pi$	$4.4\ 10^{-5}$
ω	782	8.43	23.4	$\pi^+\pi^-\pi^0$	$7.2\ 10^{-5}$
ϕ	1019	4.43	44.4	K^+K^-	$3.1\ 10^{-4}$

Tabelle 1 fundamentale Eigenschaften der leichten Vektormesonen [HAD94]

Verwendet man nun leichte Vektormesonen als Sonde, so bieten sich aufgrund ihrer Eigenschaften (siehe Tabelle 1) folgende experimentelle Vorteile:

- Kurze Lebensdauer, so daß ein Zerfall mit hoher Wahrscheinlichkeit in der dichten Kollisionszone stattfindet.
- Existenz eines Zerfallskanales, der ausgangsseitig keiner starken Wechselwirkung unterliegt.

Aufgrund seiner sehr kurzen Lebensdauer von ca.1.3 fm/c bietet sich nun im besonderen das ρ-Meson für den Nachweis der „In-Medium-Effekte“ an. Da es mit hoher Wahrscheinlichkeit

[1] HAD94 - HADES Kollaboration „Proposal for a High-Acceptance Di-Electron Spectrometer 1994“

[2] CAS90 - BUU-Modellrechnung: W.Cassing, V.Metag, U.Mosel, K.Niita Phys.Rep. 188 (1990) 363

in der heißen und dichten Reaktionszone zerfällt, sollten sich die erwarteten Effekte direkt in der invarianten Masse des ρ-Mesons nachweisen lassen.

Zu diesen Effekten existieren verschieden Modellrechnungen, die teilweise widersprüchliche Aussagen liefern.

Eine mit einem feldtheoretischen Baryon-Meson-Modell durchgeführte Rechnung[3] sagt voraus, daß die Kopplung der Austausch-Pionen an die Δ-Loch-Zustände der dominierende Faktor für die „In-Medium-Modifikationen" des ρ-Mesons sind. Der Grund hierfür liegt nach diesem Modell in der Veränderung der Breite des ρ-Mesons, welches sich als $\pi^+\pi^-$ Resonanz beschreiben läßt. Eine Verschiebung der Pionen-Massen würde so zu einer Veränderung der Kopplung an die Δ-Loch-Zustände führen, was sich in einer dichteabhängigen Verbreiterung des ρ-Mesons niederschlagen würde.

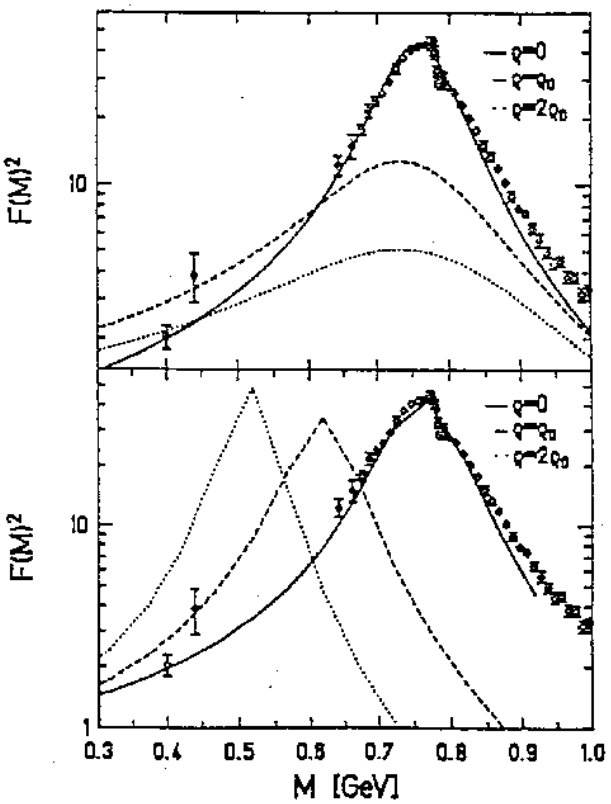


Abbildung 1 Vorhersagen für „In-Medium-Modifikationen" des ρ-Mesons für verschiedene Materiedichten. Bild oben zeigt die Ergebnisse der Rechnung von Herrmann, Bild unten jene von Asakawa

Eine zweite Modellrechnung beschreibt die „In-Medium-Modifikationen" als QCD-Effekte[4]. Nach dieser Berechnung ändert sich in verdichteter Kernmaterie der Erwartungswert des chiralen Kondensates, was zu einer Veränderung der Massen der Konstituenten-Quarks führen muß. In diesem Modell führt gerade die spontane Brechung der chiralen Symmetrie dazu, daß die Konstituenten massebehaftet sind. Eine Änderung der Quark-Massen würde nach diesem Modell aber zwangsläufig zu einer Verschiebung der ρ-Masse führen.

Während also nun das erste Modell eine Verbreiterung des ρ-Mesons mit zunehmender Materiedichte erwarten läßt, schlägt das zweite Modell eine Verschiebung des ρ-Peaks zu kleineren Massen hin vor (siehe Abbildung 1).

[3] HER93 - M.Herrman et al., Nucl. Phys. A560 (1993) 411

[4] ASA92 - Asakawa et al., Phys. Rev. C 46 (1992) R1159

Weitergehende Rechnungen[5] sagen eine allgemeine Restaurierung der chiralen Symmetrie für großen Materiedichten und/oder hohe Temperaturen voraus.
Brown und Rho[6] haben schließlich aus QCD-Summenregeln folgendes Skalenverhalten für die Zerfallskonstante des Pions und damit für die Massen der leichten Vektormesonen abgeleitet:

$$\frac{M_\rho^*}{M_\rho} = \frac{M_\varpi^*}{M_\varpi} = \frac{M_\phi^*}{M_\phi} = \frac{f_\pi^*}{f_\pi}$$

Gl. 1 Gleichung für das Skalenverhalten von Mesonmassen und Pionzerfällen nach Brown und Rho, wobei im Nenner die freien Werte der jeweiligen Größen eingetragen sind. [* - Im Medium]

Das HADES-Spektrometer soll nun mit seinem Physikprogramm dazu beitragen, etwas mehr Licht in die Vielfalt der theoretischen Modelle der Kernmaterie zu bringen.
Wie Abbildung 2 zeigt, gibt es verschiedene Beiträge zum Dileptonenspektrum. Außer den Zerfällen der drei Vektormesonen tragen die Dalitz-Zerfälle des η und des Δ sowie die $\pi^+\pi^-$ Annihilation zum Spektrum bei.

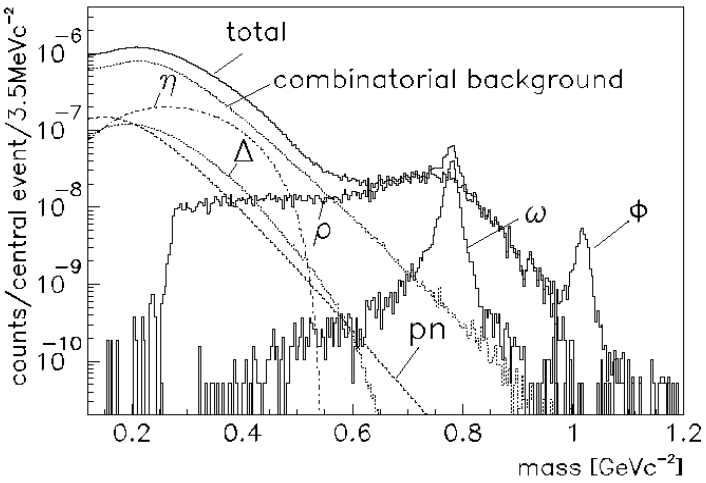


Abbildung 2 simuliertes Dileptonenspektrum 1 AGev Au+Au[1]

Zu einem späteren Zeitpunkt ist ferner geplant, mit dem in der Planung befindlichen Pionenstrahl an der GSI den Zerfall rückstoßfrei erzeugter ω-Mesonen im Kern zu messen[7]. Nur so ist es möglich, „In-Medium-Effekte“ an ω-Mesonen in kalter Kernmaterie zu untersuchen, da diese aufgrund ihrer Lebensdauer von 23.4 fm/c ansonsten außerhalb der Reaktionszone zerfallen würden.
Außerdem sind einige weitere Experimente geplant, auf die ich im Abschnitt 1.4 eingehen werde.

---

[5] WEI94 - W.Weise, Proceedings „Workshop on Dilepton Production in Heavy Ion Reactions“, GSI 1994

[6] BRO91 - G.E.Brown, M.Rho, Phys. Rev. Lett. 66 (1991) 2720

[7] SCH96 - W.Schön et al., Meson 96 (Krakau), Untersuchung rückstoßfrei produzierter ω-Mesonen am Pionenstrahl der GSI

Generelle Probleme der Dileptonenspektroskopie sind die sehr kleinen Produktionsquerschnitte und der zu erwartende starke hadronische Untergrund (siehe Verzweigungsverhältnisse Tabelle 1). Außerdem stellt die benötigte Massenauflösung, die in der Größenordnung der ω-Breite liegen muß, eine große Herausforderung an das geplante Spektrometer.

1.2 Das HADES-Detektorsystem

1.2.1 Aufbau des HADES-Spektrometers

Um den oben aufgeführten Anforderungen gerecht zu werden, wird das HADES-Spektrometer aus verschiedenen einzelnen Detektoren aufgebaut sein (siehe Abbildung 3).

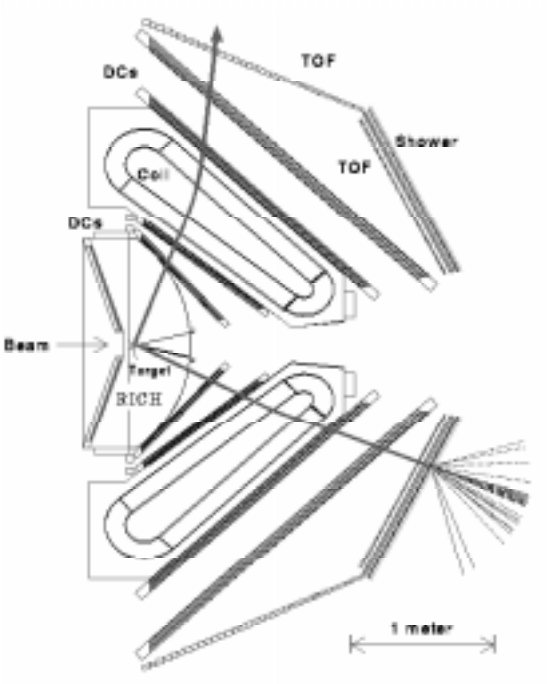
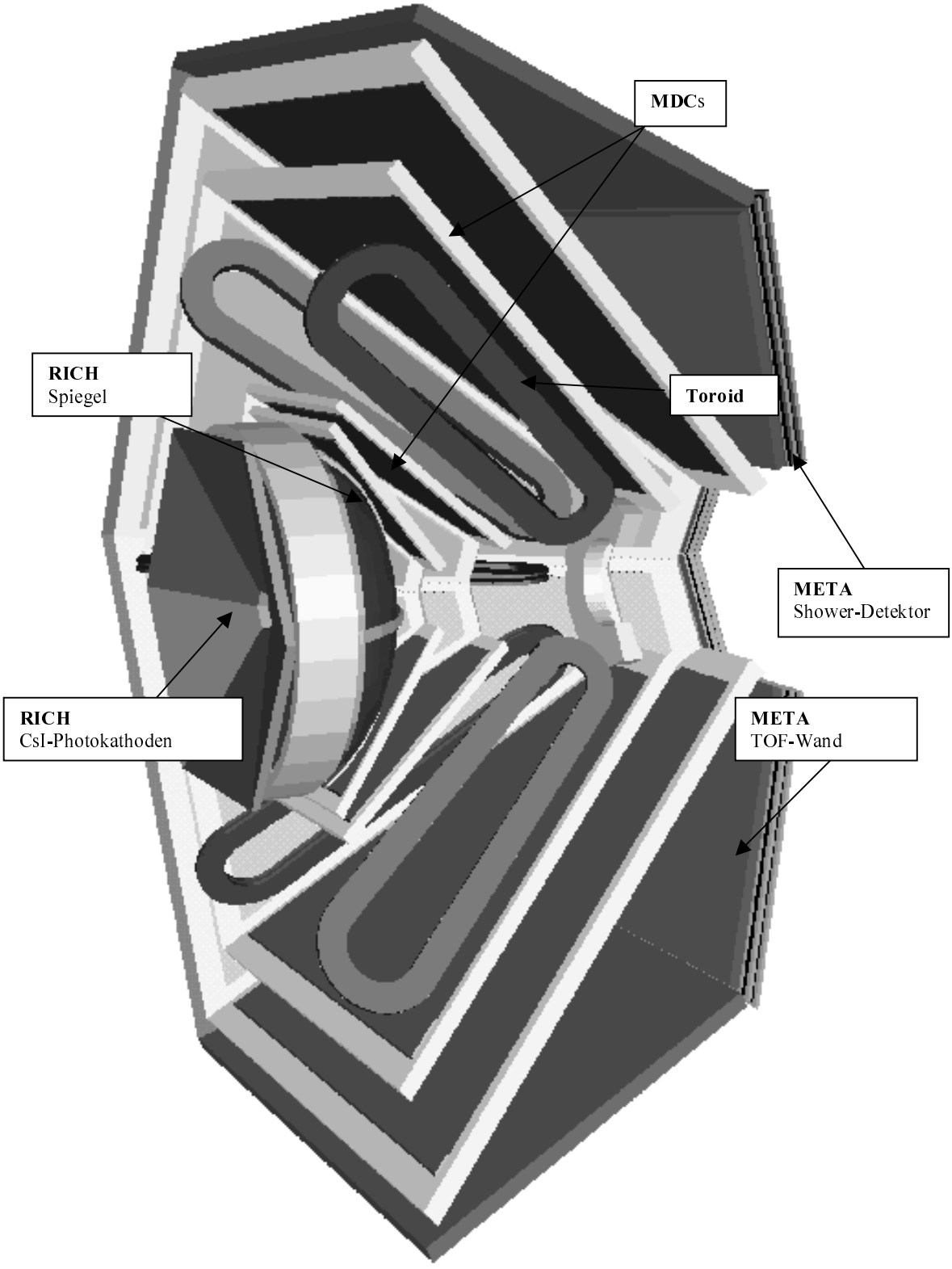


Abbildung 3 Seitenquerschnitt durch HADES

Dabei wird aufgrund der achsensymmetrischen Geometrie eine Winkelakzeptanz von nahezu 2π erreicht (siehe Bild Seite 11).

Herzstück des Spektrometers ist dabei ein Ring Imaging Cherenkov Detektor (**RICH**), der mit einem Gasradiator (C_4F_{10}) gefüllt ist und bei einem Druck von 1.0 – 1.3 bar betrieben werden soll. Der große Vorzug dieser Komponente ist die Tatsache, daß der RICH bei den angestrebten Energien hadronenblind ist, d.h. es werden nur Ringbilder für Elektronen bzw. Positronen erzeugt, da fast alle Hadronen unterhalb der Cherenkov-Schwelle liegen. Dileptonen werden so anhand ihre Ringabbilder identifiziert und lokalisiert. Es ist so möglich, schon im ersten Detektor eine hervorragende Dileptonenidentifikation zu erzielen.

Direkt nach dem RICH folgen die ersten beiden Ebenen von Driftkammern (**MDC**s – Mini Dift Chambers), die zusammen mit einem toroidalen supraleitenden Magneten und den dahinter liegenden beiden Driftkammerebenen eine direkte Impulsmessung der emittierten Teilchen aus der Krümmung der Teilchenspur (siehe Abbildung 4) erlauben. Die angestrebte Impulsauflösung liegt dabei bei 1%, woraus sich eine Massenauflösung von $\Delta m \leq 10$ MeV/c^2 ergibt. Mit dieser Auflösung ist nun auch eine Trennung von ρ und ω Mesonen möglich.



3D-Seitenansicht des HADES-Spektrometers
Deutlich zu sehen ist der HADES-RICH mit seinem Spiegel und den geneigten Fokalebenen. Dahinter (hell) die ersten beiden MDCs, gefolgt vom supraleitenden Toroiden und den verbliebenen zwei MDC-Ebenen. Ganz außen sieht man den META-Detektor aus TOF-Wand und Shower-Detektor.

Im äußersten Bereich des HADES-Detektors befindet sich die Flugzeitwand (**TOF**), welche den vollen Winkelbereich von 22° bis 88° abdeckt. Sie ist aus 996 Szintillatorpanels aufgebaut, die zum einen eine weitere Leptonenidentifikation über Flugzeitmessungen ermöglicht, andererseits aber hauptsächlich der Event-Charakterisierung und als 1st Level Trigger (Teilchen – Multiplizität) dienen wird.
Im Winkelbereich zwischen 22° und 45° wird die Flugzeit zusätzlich durch einen Schauerdetektor ergänzt. Dieser ist neben dem RICH die zweite Detektorkomponente, welche eine hervorragende Leptonenidentifikation ermöglicht. Dabei werden Leptonen anhand elektromagnetischer Schauer lokalisiert. Dazu besteht der Schauerdetektor aus einer Sandwich-Konstruktion aus drei Schichten von Bleikonvertern und Vieldrahtproportional-kammern, die im sogenannten Self Quenching Streamer Mode betrieben werden.
Die Kombination aus Flugzeitwand und Schauerdetektor wird im HADES-Jargon auch **META** (Multiplicity Electron Trigger Array) genannt.
Neben dieser Kombination aus verschiedenen Detektoren bildet der dreistufige Trigger zusammen mit dem Datenaufnahmesystem ein weiteres Herzstück des Detektors.

1.2.2 Der HADES-Trigger

Um die erwarteten „In-Medium-Modifikationen“ leichter Vektormesonen in einem Dileptonen-Spektrometer nachweisen zu können, ist aufgrund des geringen Verzweigungsverhältnisses der Dileptonenzerfälle eine sehr hohe Primärteilchenrate notwendig.
Dazu wurde das HADES-Spektrometer auf eine Strahlintensität von 10^8 Teilchen pro Sekunde ausgelegt. Unter Verwendung eines Targets mit ca. 1% Wechselwirkungslänge erwartet man so etwa 10^6 Ereignisse pro Sekunde. Mit der Produktionswahrscheinlichkeit von Dileptonen mit invarianten Massen um den ρ - ω - Peak, die bei ca. 10^{-6} pro zentralem Event liegt, und der Erwartung von einem Anteil von 10% zentraler Stöße ist statistisch nur mit etwa 0.1 Dileptonenpaar pro Sekunde zu rechnen. Aus diesen Abschätzung erwächst die Notwendigkeit eines äußerst effizienten Triggersystemes für das HADES-Spektrometer.
Um dieser Aufgabe gerecht zu werden, wurde der HADES-Trigger als dreistufiges Konzept entworfen, welches nun kurz erläutert werden soll:

- Wie in der Vorstellung des HADES-Spektrometers schon angeklungen ist, besteht der **1st Level Trigger** aus einem Multiplizitätstrigger[8], welcher mittels der Szintillatoren der

[8] GSI91 - W.Ahner, GSI Scientific Report 1991 S.49 und dort aufgeführte Referenzen

TOF-Wand zentrale Ereignisse (ca. 10% aller Ereignisse) herausfiltern soll. Dabei wird erwartet, daß bei entsprechender Multiplizitätsschwelle eine Reduktion der Ereignissrate von 10^6 auf 10^5 pro Sekunde erzielt wird.

- Der **2nd Level Trigger** wird aus den beiden Detektorkomponenten RICH und META gewonnen. In der META-Komponente werden Dileptonen einerseits anhand elektromagnetischer Schauer[9] und andererseits über Flugzeitmessungen erkannt. Der RICH-Detektor ermöglicht eine Identifikation der Leptonen anhand ihrer Ringmuster[10]. Aus beiden Subsystemen erhält man so Winkelinformationen bezüglich der gefunden Leptonenkandidaten, welche in zwei Schritten in einer sogenannten Matching Unit zu Dileptonenpaaren kombiniert werden (siehe Abbildung 4).

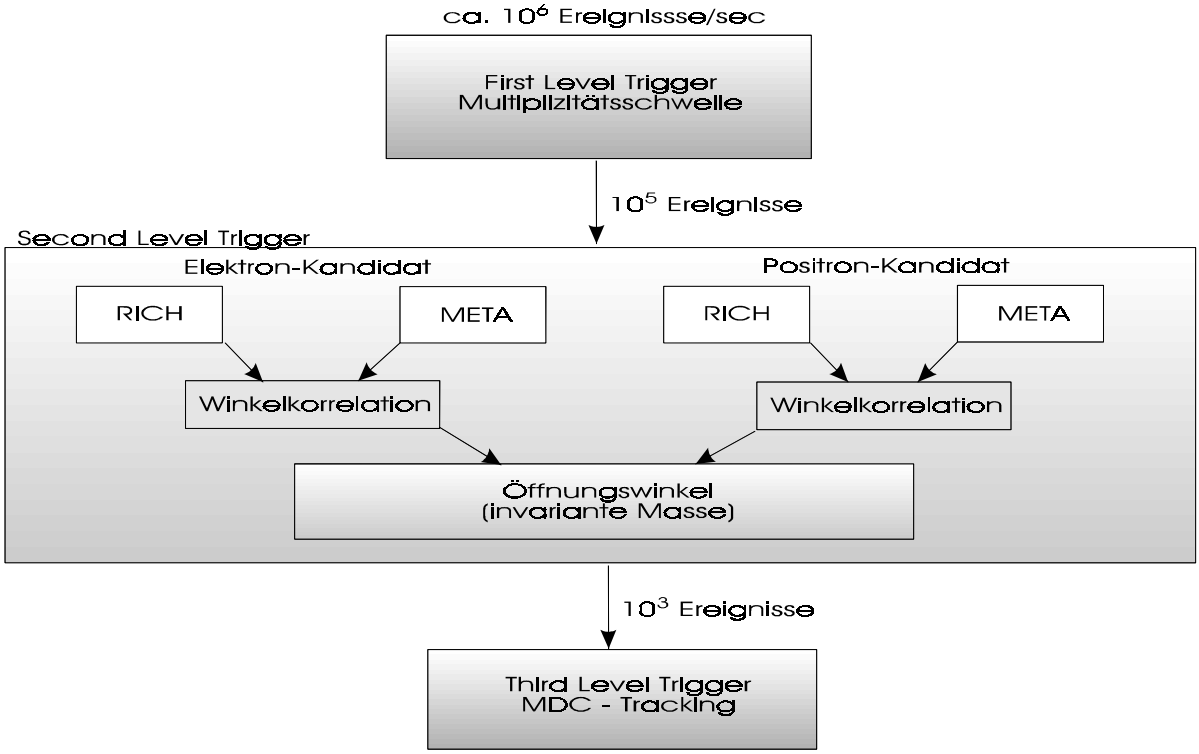


Abbildung 4 Schema des HADES-Triggerkonzeptes

Im ersten Schritt werden mit Hilfe eines Impulsfensters interessante Elektronen- und Positronenkandidaten ausgewählt, welche dann im zweiten Schritt paarweise auf ihren Öffnungswinkel (d.h. ihre invariante Masse[11]) hin kontrolliert werden. Auf diese Weise

[9] PET95 - M.Petri, Diplomarbeit 1995, Schnelle Ladungsmuster-Erkennung mit feldprogrammierbaren Gate-Arrays

[10] KRA96 - H.Kraft, Diplomarbeit 1996, Ein Prozessor zur schnellen Ringerkennung für den HADES RICH

[11] invariante Masse: $m_{inv}c^2 = \sqrt{2E_{e^+}E_{e^-}\left(1-\cos\Phi_{e^+e^-}\right)}$

werden diejenigen Ereignisse ausgewählt, die physikalisch relevante Dileptonenpaare enthalten. Bei geeigneter Wahl der Schwellen, wird so eine Reduktion der Ereignissrate um einen Faktor 100 erreicht.

Aus der 1st Level Triggerrate folgt, daß diese Stufe in der Lage sein muß, alle 10μs ein Event verarbeiten zu können.

- Die letzte Stufe des HADES-Triggersystemes stellt der **3rd Level Trigger** dar. In dieser Stufe werden die gefundenen Dileptonenkandidaten auf das Vorhandensein von entsprechenden Teilchenspuren in den MDCs hin überprüft. Diese Trackingeinheit erlaubt Simulationen zufolge eine weitere Reduktion der Ereignisrate auf schließlich ca. 10^2 pro Sekunde. Während die ersten beiden Stufen des Triggers Hardware-Lösungen darstellen, wird die dritte Stufe eine reine Software-Lösung sein.

Um nun den schon erwähnten Durchsatz von 10^5 pro Sekunde ($\Rightarrow$ erwartete 1st Level Triggerrate) zu erreichen, basiert das gesamte Datenaufnahmesystem des HADES-Spektrometers auf einer Pipeline-Architektur, d.h. z.B. für den 2nd Level Trigger, daß jede einzelne Komponente in der Lage sein muß, ein komplettes Event innerhalb von 10μs abzuarbeiten. Andererseits werden in einer Pipeline-Architektur alle Prozesse in kleine separate Unterprozesse unterteilt, die parallel verschiedene Ereignisse bearbeiten können, so daß die Propagationszeit (man spricht hier von latency) eines einzelnen Events durch die komplette Datenaufnahmen, von der Frontend-Elektronik bis schließlich zum Bandlaufwerk, deutlich größer sein kann. Diese Gesamtzeit ergibt sich aus dem Produkt der Zeit für einen Einzelschritt und der Anzahl der Schritte. Für den 2nd Level Trigger erwartet man bei HADES eine Latenzzeit von ca. 50μs, was einer Anzahl von 5 Events entspräche.

1.3 Das TAPS-Spektrometer – Two Arm Photon Spectrometer

Nachdem im vorangegangenen Abschnitt das Dileptonen-Spektrometer HADES vorgestellt wurde, möchte ich im folgenden kurz auf das Photonen-Spektrometer TAPS eingehen, da dieses eine wesentliche Rolle in HADES-Experimenten spielen wird, auf die ich dann im Abschnitt 1.4 näher eingehen möchte.

1.3.1 Aufbau des Photonen-Spektrometers TAPS

Das „**T**wo **A**rm **P**hoton **S**pectrometer“ TAPS wurde entwickelt, um hochenergetische Photonen, sowie neutrale Mesonen (π^0,η) über ihren Zerfall in 2 Photonen nachzuweisen.
Anders wie der HADES-Detektor weist das TAPS-Spektrometer im wesentlichen nur eine Detektorkomponente auf. Dabei handelt es sich um ein Kalorimeter aus Bariumfluorid-Szintillatoren, welches für spezielle Experimente mit einem zusätzlichen Veto-Detektor für geladenen Teilchen versehen werden kann. In der ersten Ausbaustufe bestand das TAPS-Spektrometer aus 384 einzelnen Bariumfluorid-Szintillatoren, die in Blöcken zu je 64 (8 x 8) gruppiert waren, so daß schließlich sechs Blöcke zur Verfügung standen, welche in zwei Türmen angeordnet waren. In diesem Aufbau bestand die Möglichkeit den Akzeptanzbereich sowohl durch Änderung des Azimutwinkels der TAPS-Türme als auch der Polarwinkel der einzelnen Blöcke zu variieren. Neben diesen Variationsmöglichkeiten der ursprünglichen TAPS-Geometrie besteht ferner die Möglichkeit die Blöcke in anderen Geometrien anzuordnen (siehe z.B. TAPS @ MAMI[12]) oder aber die Blockstruktur komplett aufzulösen und die Detektoren als Supercluster zu konfigurieren. So läßt sich der TAPS-Detektor sehr gezielt an die kinematischen Bedingungen unterschiedlicher Experimente anpassen. Ferner besitzt TAPS aufgrund seines modularen Aufbaus den Vorzug an den unterschiedlichsten Beschleunigeranlagen Experimente durchführen zu können.
Wie zu Beginn schon erwähnt, bilden Szintillatoren aus Bariumfluorid (BaF_2), welches eine gute Zeit-, Energie- und Ortsauflösung ermöglicht, das Herzstück des TAPS-Spektrometers. Die wesentliche Charakteristika von Bariumfluorid sind dabei:

- Dichte : 4.88 g/cm^3
- Strahlungslänge X_0 : 2.05 cm
- Molière-Radius r_M : 4.3 cm
- 2 komponentiges Szintillationslicht mit einer langsamen und einer schnellen Komponente, wobei die schnelle Komponente ihre Maxima bei λ = 195 und 220 nm und eine Lebensdauer von ca. 0.6 ns aufweist. Für die langsame Komponente gilt λ = 320 nm und τ = 620 ns.[13]

[12] GAB93 - A.R.Gabler, Diplomarbeit 1993, Ansprechverhalten des Detektorsystems TAPS für monochromatische Photonen im Energiebereich 50 bis 780 MeV
[13] LAV83 - M.Laval et al., NIM A206 (1983) 169 / KUB87 – S.Kubota et al., Phys.Stat.Sol.(b) 139, 635(1987)

- Teilchenidentifikation durch Pulsformanalyse: Da die beiden Szintillations-komponenten für unterschiedliche Teilchen verschieden stark angeregt werden, ermöglicht Bariumfluorid eine Teilchenidentifikation mittels Pulsformanalyse[14].
- Die schnelle Komponente ermöglicht eine Zeitauflösung Δt besser als 200ps.

Abbildung 5 zeigt eines der TAPS-Detektormodule.

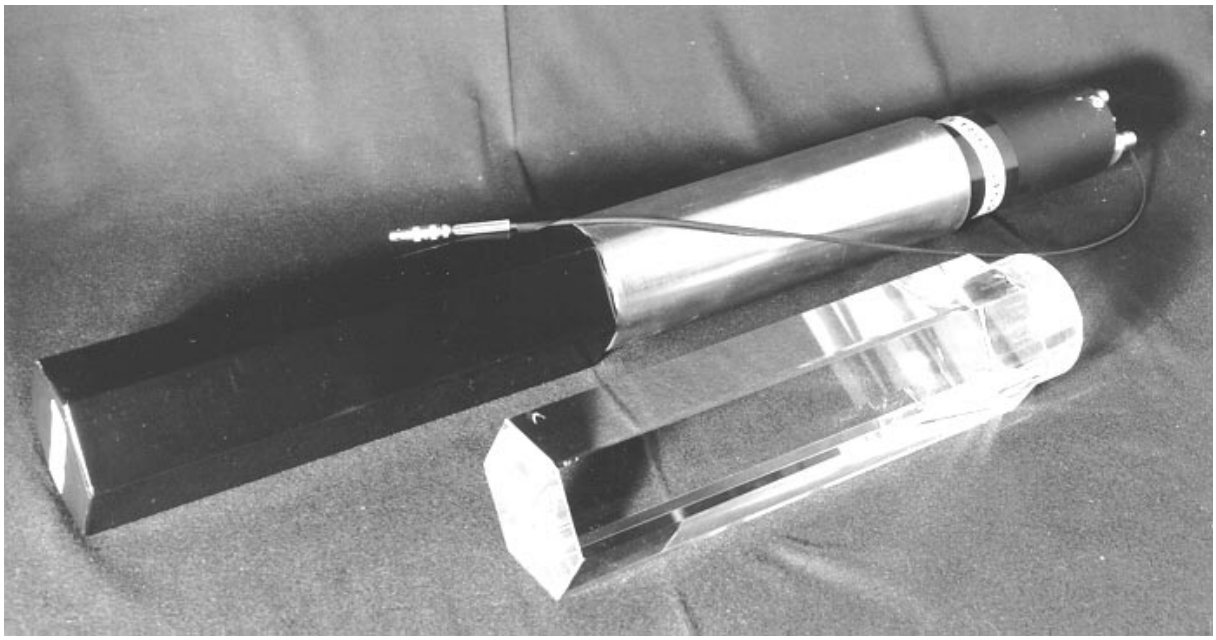


Abbildung 5 Photo eines TAPS-Modules. Im Vordergrund zu sehen, ein BaF_2-Kristall, wie er im TAPS-Detektor zum Einsatz kommt.

Ein einzelnes Detektormodul besteht aus einem BaF_2-Kristall, einem Photomultiplier und einem Spannungsteiler.

Die Bariumfluorid-Kristalle haben dabei einen hexagonalen Querschnitt mit einem Innendurchmesser von 5.4 cm, was ca. 1.26 Molière-Radien[15] entspricht. Ihre Länge beträgt 25 cm, also etwa 12 Strahlungslängen[16]. Die Länge der Kristalle ist so für den Nachweis von Photonen bis zu einigen 100 MeV optimiert. Um die Lichtausbeute zu verbessern, wurden die Kristalle optisch poliert und in 8 Lagen Teflon- sowie eine Lage Aluminiumfolie eingewickelt. Die Lichtdichtigkeit jedes einzelnen Moduls wird durch einen

[14] SCH89 - O.Schwald, Diplomarbeit 1989, Test der ersten TAPS-Untereinheit mit Elektronen

[15] Innerhalb von 2 Molière-Radien sind ca. 90% eines elektromagnetischen Schauers enthalten.

[16] Die Wahrscheinlichkeit für Paarerzeugung innerhalb einer Strahlungslänge liegt bei ca. 54%.

Schrumpfschlauch garantiert. Zur Überwachung der Detektormodule kann zusätzlich Licht eines N_2-Lasers über Glasfasern in die Kristalle eingebracht werden[17].

Wie eingangs schon erwähnt, kann zur Unterdrückung geladener Teilchen zusätzlich ein Veto-Detektor vor jedes einzelne Modul montiert werden. Dabei handelt es sich um einen üblichen Plastikszintillator.

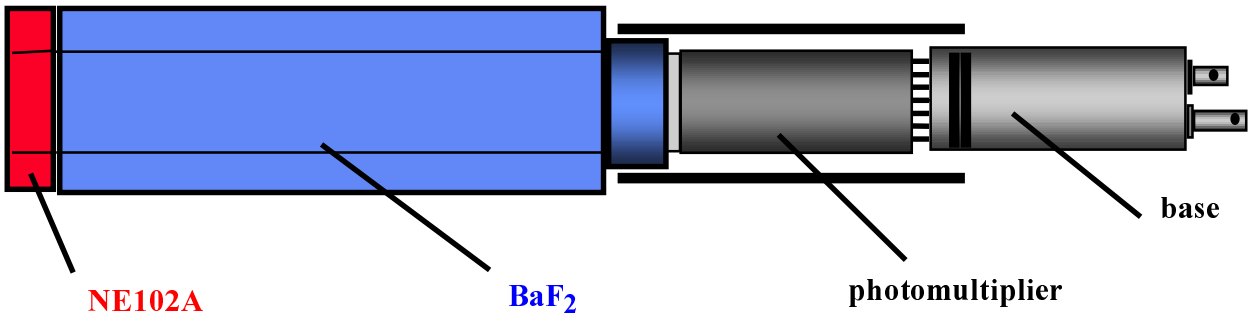


Abbildung 6 Schemazeichnung eines Detektormoduls mit Vetozähler

1.3.2 Nachweis von mittelenergetischen Photonen mit TAPS

Das TAPS-Spektrometer wurde optimiert, Photonen bis zu Energien von einigen 100 MeV nachzuweisen. Dazu bedient es sich, wie schon mehrfach angeklungen ist, des Szintillationseffektes im anorganischen Szintillator Bariumfluorid.

Die Photonen deponieren dabei ihre Energie auf dreierlei Weisen (Photoeffekt,Comptoneffekt und Paarbildung) im Szintillationsmaterial, das durch diese Energiedeposition angeregt wird und diese Energie in Form von Szintillationslicht emittiert. Ein hochenergetisches Photon konvertiert nun zunächst in ein Elektron-Positron-Paar [Die Konversionsrate beträgt ca. 54% innerhalb der 1.Strahlungslänge], welches seinerseits wieder in Photonen zerstrahlt. Diese Photonen wiederum können je nach Energie wieder einen der drei Prozesse durchlaufen, wobei mit zunehmender Energie die Paarbildung überwiegt. Auf diese Weise kommt es zur Ausbildung einer Kaskade aus Photonen, Elektronen und Positronen bis schließlich die Energie der Photonen einer Folgegeneration nicht mehr zur Paarbildung ausreicht. Ist dieser Punkt erreicht, kommt der Kaskadeneffekt zum erliegen. Man nennt die entstandene Kaskade nun auch elektromagnetischen Schauer, da dieser Schauer nur auf Prozessen der QED beruht.

[17] BOO90 - A.L.Boonstra, Optical Gain Monitoring and Calibration System for TAPS, GSI Scien. Rep. 1990

Bei der Ausbildung des Schauers kommt es je nach Zentralität des Treffers zu einer Ausbreitung des Schauers auf die umliegenden Detektormodule[18]. Wie der Molière-Radius von Bariumfluorid erwarten läßt, beträgt dabei der Schauerinhalt eines Arrays aus 7 Detektormodulen ca. 90 % (siehe Abbildung 7). Messungen[18] haben gezeigt, daß eine Verwendung von 19 statt 7 Detektormodulen die Energieauflösung von TAPS von ca. 10% FWHM/E auf ca. 7% FWHM/E verbessert.

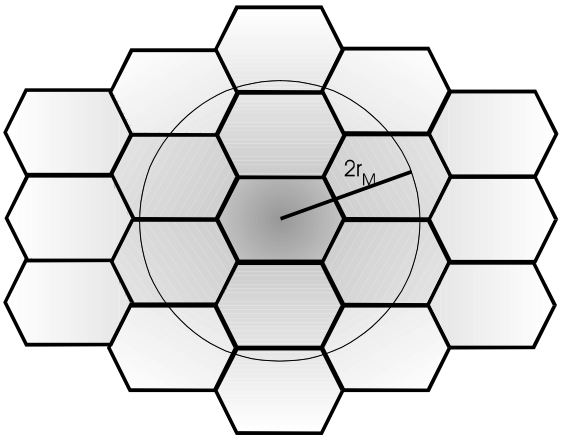


Abbildung 7 Schemazeichnung eines Schauers in 7 (19) TAPS-Modulen

1.4 Gemeinsame Experimente von HADES und TAPS

Neben den in Abschnitt 1.1 bereits angeführten physikalischen Zielen des HADES-Projektes sind weitere Experimente und Studien geplant. So stehen zum Beispiel Messungen zur elektromagnetischen Struktur von Baryonen und Mesonen auf dem Programm. Von besonderem Interesse ist in diesem Programm eine Messung des zeitartigen Formfaktors von Nukleonen im „unphysikalischen“ Bereich. Aus experimenteller Sicht bedeutet dies, daß eine sehr sorgfältig Vermessung und Analyse des kontinuierlichen Spektrums im Massenbereich unterhalb 500 MeV/c^2 durchgeführt werden muß. Dazu könnten Experimente im Bereich sehr niedriger Einschußenergien durchgeführt werden, um die η-Produktion möglichst gering zu halten. Andrerseits wäre eine Ergänzung des HADES-Spektrometers durch ein elektromagnetisches Kalorimeter denkbar, um so die entstandenen neutralen η-Mesonen nachzuweisen..

Eine weitere bedeutende Gruppe von Experimenten ergibt sich unter Ausnutzung des an der GSI diskutierten Pionenstrahls. Mit einem derartigen Strahl hochenergetischer Pionen wären Untersuchungen elementarer Wirkungsquerschnitte sowie grundlegender Eigenschaften von Mesonen möglich.

So wären z.B. Messungen zum elektromagnetischen Formfaktor des ω-Mesons von besonderem Interesse, da Messungen[19] der LEPTON-G Kollaboration Anzeichen dafür

[18] APP91 - M.Appenheimer, Diplomarbeit 1991, Test einer TAPS-Untereinheit mit monochromatischen Photonen

[19] LAN85 - Landsberg, Phys.Rep. 128 (1985) 301

liefern, daß der Formfaktor des ω-Mesons durch das **V**ector **D**ominance **M**odel nur unzureichend beschrieben wird. Die Daten der LEPTON-G Kollaboration zeigen eine teilweise um den Faktor 10 überhöhten Wirkungsquerschnitt. Zur Untersuchung des ω-Formfaktors bietet sich der ω-Dalitz-Zerfall an, zumal dieser den Hauptbeitrag zum Dileptonenspektrum im Bereich niedriger invarianter Massen liefern sollte. Eine Vermessung dieses Zerfallskanals des ω-Mesons mit HADES-Spektrometer erfordert aber den zusätzlichen Einbau eines elektromagnetischen Kalorimeters, um das emittierte π^0-Meson durch seinen Zerfall in zwei Photonen nachweisen zu können. Gerade für diese Experimente bietet sich nun aber eine Zusammenarbeit von HADES und TAPS an, da das hohe Auflösungsvermögen von TAPS das HADES-Spektrometer sehr gut ergänzen könnte.

Im Folgenden soll nun der Gedanke gemeinsamer HADES/TAPS – Experimente aufgegriffen und einige fundamentale Richtlinien für Experimente dieser Art entwickelt werden. Ausgehend von diesen Richtlinien lassen sich dann verschieden Ansätze diskutieren.

2 Konsequenzen und Konzepte

2.1 TAPS im HADES-Umfeld

Wie in Abschnitt 1.4 bereits angeklungen ist, ergibt sich für einige interessante Experimente mit dem HADES-Spektrometer die Notwendigkeit der Ergänzung durch ein elektromagnetisches Kalorimeter. Als eine Option bietet sich hier eine Lösung mit einem kostengünstigen Bleiglas-Kalorimeter an. Die relativ niedrigen Kosten einer derartigen Erweiterung würde den Aufbau eines Kalorimeters mit großem Akzeptanzbereich ermöglichen. Eine andere diskutierte Option stellt die Anbindung des TAPS-Detektors an das HADES-Environment dar. Letztere Variante ist insofern von besonderem Reiz, als daß sich die hohe Energie-, Orts- und Zeitauflösung des TAPS-Spektrometers positiv auf die geplanten Experimente auswirken sollte.

Soll der TAPS-Detektor an das HADES-Spektrometer angegliedert werden, so ergeben sich einige grundlegende Rahmenbedingungen für das TAPS-System, welche mit der existierenden Elektronik nicht zu bewältigen sind.

Von fundamentaler Bedeutung hierbei ist die Einbindung des TAPS-Detektors in das HADES-Triggerkonzept, da gerade eine zusätzliche Triggerinformation die Effizienz des HADES-Systems bei den in Abschnitt 1.4 diskutierten Experimenten erhöhen würde. So wäre zum Beispiel eine zusätzliche 2^{nd} Level Trigger – Information bezüglich der invarianten Masse des im TAPS-Detektor nachgewiesenen Photonenpaares von herausragender Bedeutung für die Messung des ω-Formfaktors. Mit der Entwicklung eines derartigen Triggers für den TAPS-Detektor bietet sich außerdem die Möglichkeit, auch mit dem TAPS-Spektrometer alleine bei höheren Datenraten zu messen[20].

Bei der Entwicklung eines „Invarianten Massen-Triggers“ [IMT] für TAPS ist nun auf folgende Punkte besonders zu achten:

- Anpassung an die HADES-Pipeline-Architektur mit einer Latenzzeit von ca. 50μs
- Festigkeit gegenüber hohen 1^{st} Level-Triggerraten von bis zu 10^5 pro Sekunde
- Erhalt der Flexibilität des TAPS-Aufbaus

Aus diesen Rahmenbedingungen lassen sich nun verschiedene Konzepte entwickeln, welche im folgenden Abschnitt kurz andiskutiert werden sollen.

---

[20] Dazu wird augenblicklich eine komplett neue DAQ diskutiert (siehe dazu relevante TAPS-Dokumente)

2.2 Lösungskonzepte für den TAPS 2nd Level Trigger

Aus den in Abschnitt 2.1 aufgelisteten Rahmenbedingungen ergeben sich für die Konzeption eines TAPS 2nd Level Triggers einige wichtige Konsequenzen, die sich direkt in den möglichen Konzepten niederschlagen. Zum einen muß bezüglich der Schauerfindung ein gangbarer Weg zur Lösung des Nachbarproblems gefunden werden, d.h., da jedes TAPS-Modul sowohl Zentrum als auch einer von sechs direkten Nachbarn eines Schauerzentrums sein könnte, muß bei der Schauersuche eine effiziente Behandlung dieses Überlapps gewährleistet sein. Außerdem sollte die Schauersuche unabhängig von der gerade verfügbaren TAPS-Konfiguration anwendbar sein.
Des weiteren erscheint eine Lösung, welche die volle TAPS-Auflösung von 12 bis 14bit abdeckt, aus verschiedenen Gründen nicht anwendbar zu sein. Erstens wäre eine derartige Lösung zu teuer, da hier nur sehr kostenintensive Bauteile in Frage kommen würden, und zweitens ist fraglich ob eine genügend hohe Verarbeitungsgeschwindigkeit bei vertretbarem Kostenaufwand möglich ist. Aus diesem Grund ergibt sich zu dem die Notwendigkeit der Reduktion auf 8bit.
Im Wesentlichen lassen sich so drei Klassen von Lösungsansätzen für die oben gestellte Aufgabe entwickeln.

- Massiv-parallele Verarbeitung als Frontend-Lösung
- 2-teilige Konzepte aus Frontend-QDC (charge to digital converter) und paralleler Triggerstufe mit SRAM-basiertem Eventspeicher[21]
- HADES-konforme parallele Triggerstufe mit Pipeline-Architektur

In dieser Arbeit wurde nun basierend auf dem dritten Lösungsansatz eine Schauererkennungseinheit entwickelt. Bevor ich auf diese genauer eingehe, möchte ich kurz die beiden anderen Ansätze diskutieren.

2.2.1 Massiv-parallele Frontend-Triggereinheit

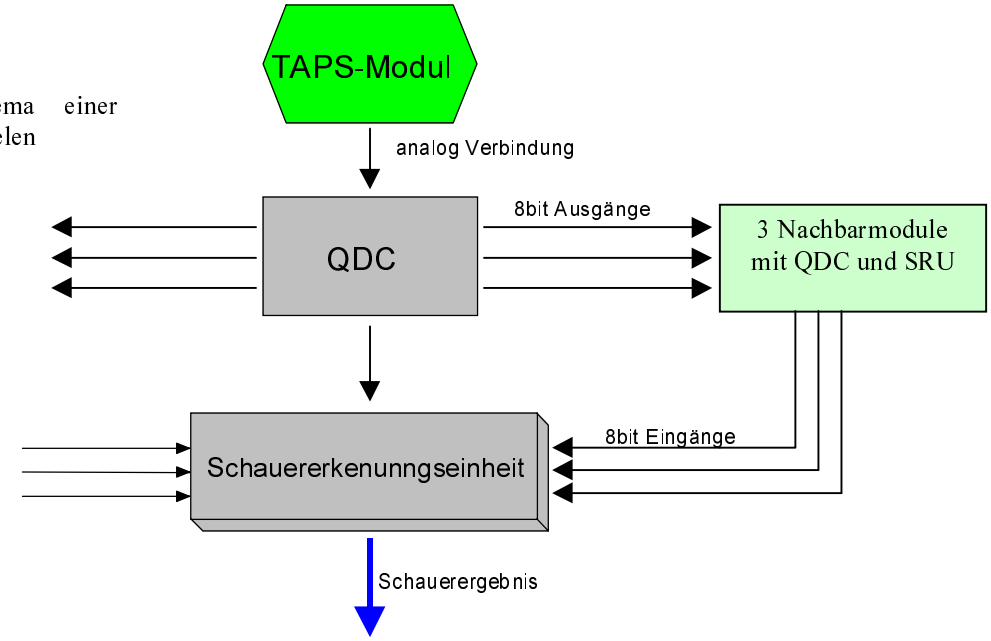
Dieses Konzept basiert auf dem Prinzip des minimalen Datentransfers, d.h. durch Verarbeitung der Eventdaten in Frontendkarten, welche direkt an einen TAPS-Modul gekoppelt werden können, soll der nötige Aufwand für den Transport der Daten minimiert werden. Dazu existiert für jedes TAPS-Detektormodul eine einzelne kleine Frontendkarte, die

[21] GAN96 - S.Plüschke, TAPS-Meeting 01.96 im GANIL / Caen

neben einem „sample and hold“ - QDC (8bit) und einer Schauererkennungseinheit zusätzlich über direkte 8bittige Datenaustauschverbindungen zu den sechs umliegenden Modulen verfügt.
In dieser Schauererkennungseinheit (Implementation in kleinem PAL oder GAL[22]) könnte so direkt hinter jedem Modul die Entscheidung über den Schauerinhalt des Detektormodules erfolgen.

Abbildung 8

Funktionsschema einer massiv-parallelen Frontendkarte



Vorteilhaft erweist sich bei diesem Konzept zum einen seine sehr hohe Verarbeitungsgeschwindigkeit und zum anderen die Unabhängigkeit von jeglichen geometrischen Überlegungen. Andererseits ist der Aufwand beim Aufbau von Experimenten enorm. Ferner stellt sich die Erfassung der Schauerergebnisse und ihre weiter Verarbeitung als Problem dar. So müssen, damit das System als „Invariant Mass Trigger“ verwendbar ist, die gefundenen Schauer in einem zweiten Schritt noch nach Energie und Winkel kombiniert werden. Diese Aufgabe übernimmt im HADES-Trigger die sogenannte Matching Unit (MU).

2.2.2 Speicherbasierte Triggersysteme

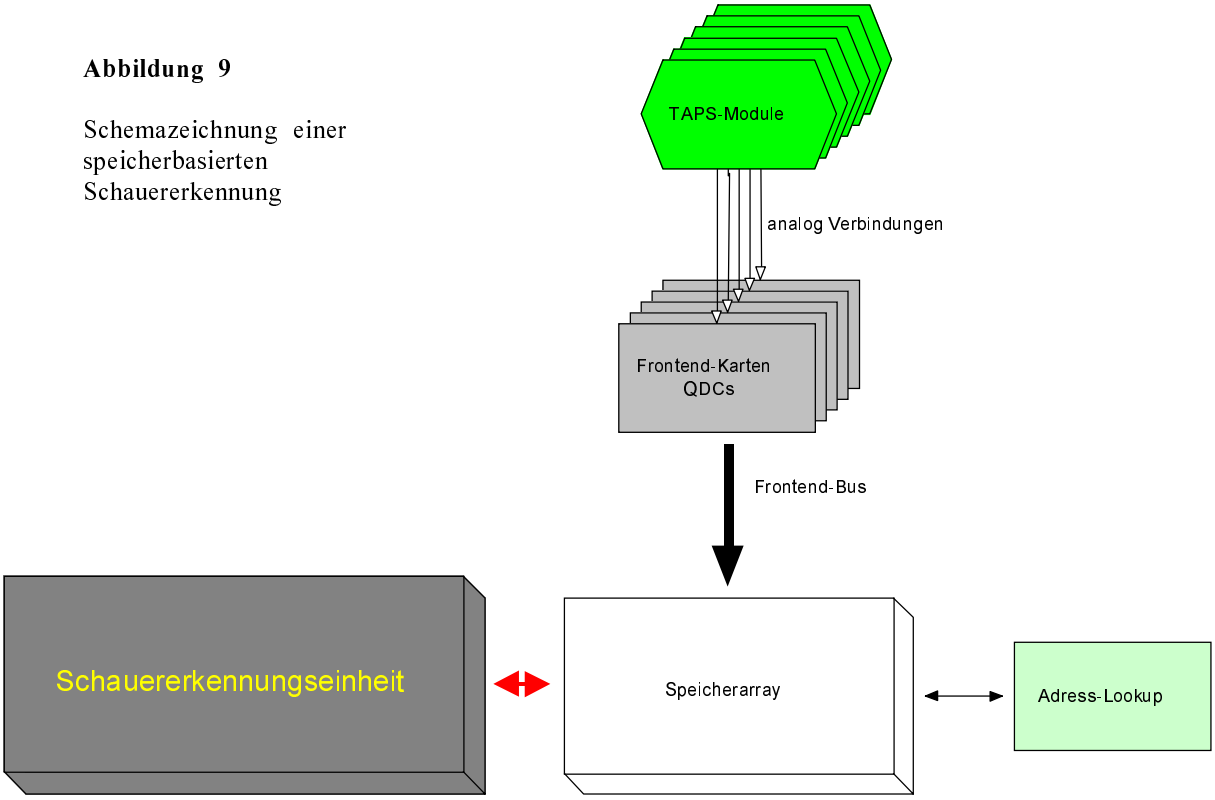
Um den in Abschnitt 2.2.1 aufgeführten Problemen zu begegnen, bietet sich die Auftrennung der Schauererkennungseinheit in zwei Stufen an. In einem Frontend-Modul würde nach diesem Konzept die Digitalisierung der TAPS-Daten erledigt, welche dann über Frontend-

[22] PAL – Programmable Array Logic & GAL – Generic Array Logic

Datenbusse entweder direkt adressiert oder in einer „daisy chain[23]“ ausgelesen würden. Die Schauererkennungseinheit ließe sich dann in einer zweiten Stufe verwirklichen.
In diesem Konzept sind auch komplexerer Logik-Bausteine, welche einen höheren Durchsatz ermöglichen, verwendbar, was zu einer Reduzierung der benötigten Logik-Bausteine führt. Die Methode der „direkt adressierten Auslese“ erweist sich dabei als für unsere Zwecke zu langsam, so daß auf eine „daisy chain[22]“ zurückgegriffen werden muß. Die ausgelesenen Daten werden dann in einen SRAM-Speicher eingetragen, wodurch mittels Adress-Lookup[24] eine Rekonstruktion der TAPS-Geometrie möglich wird.
Um eine genügend hohe Geschwindigkeit zu erzielen, ist allerdings eine mehrfache Spiegelung des Speichers notwendig, was das ganze Konzept eher unelegant wirken läßt.

Abbildung 9

Schemazeichnung einer speicherbasierten Schauererkennung



Im folgenden Abschnitt soll nun eingehend ein Lösungsansatz vorgestellt werden, der sich stark an der Konzeption des 2nd Level Triggers des HADES-Spektrometers orientiert und schließlich zur Verwirklichung dieser Arbeit geführt hat.

---

[23] daisy chain - sequentielle Bearbeitung (Auslese) in einer Kette verschalteter Bauelemente, wobei das „activity strobe“ von einem Baustein zum nächsten durchgereicht wird.
[24] lookup - Rekonstruktion der Geometrie aus einer Verknüpfungstabelle, die einzelne Detektormodule mit ihren geometrischen Positionen mit Speicheradressen verknüpft bzw. umgekehrt.

2.3 HADES-konformer TAPS 2nd Level Trigger

Die Schönheitsfehler des zu letzt vorgestellten Konzeptes lassen sich durch einige neue Konzeptionsmerkmale, wie sie im übrigen schon im HADES 2nd Level Trigger verwirklicht wurden, beheben.

Hauptproblem des speicherbasierten Konzeptes war die Geschwindigkeit der Speicherzugriffe, die letztlich zu einer mehrfach Spiegelung des Speicherinhaltes führen muß. Dieser Effekt läßt sich nun durch die Konzeption einer Pipeline-Architektur umgehen. Während in dem in Abschnitt 2.2.2 diskutierten Ansatz die Schauererkennung im Prinzip in einem Schritt verläuft, kann durch Zerlegung dieses Prozesses in kleine Teilprozesse das Speicherproblem behoben werden. Um gleichzeitig aber weiterhin die Flexibilität der TAPS-Konfiguration beibehalten zu können, muß ein adäquater Ersatz für das Speicherarray und den Adress-Lookup-Mechanismus gefunden werden. Da jeder TAPS-Aufbau sich im Prinzip auf eine segmentierte Ebene abbilden läßt (siehe Abbildung 10), bietet sich hier die Einführung einer Datenmatrix als möglicher Ersatz an. Die Existenz einer derartigen Matrix hätte ferner zur Folge, daß eine Parallelisierung des Schauererkennungsprozesses bei gleichzeitiger Reduktion des Überlapps konzeptionell sehr einfach ist.

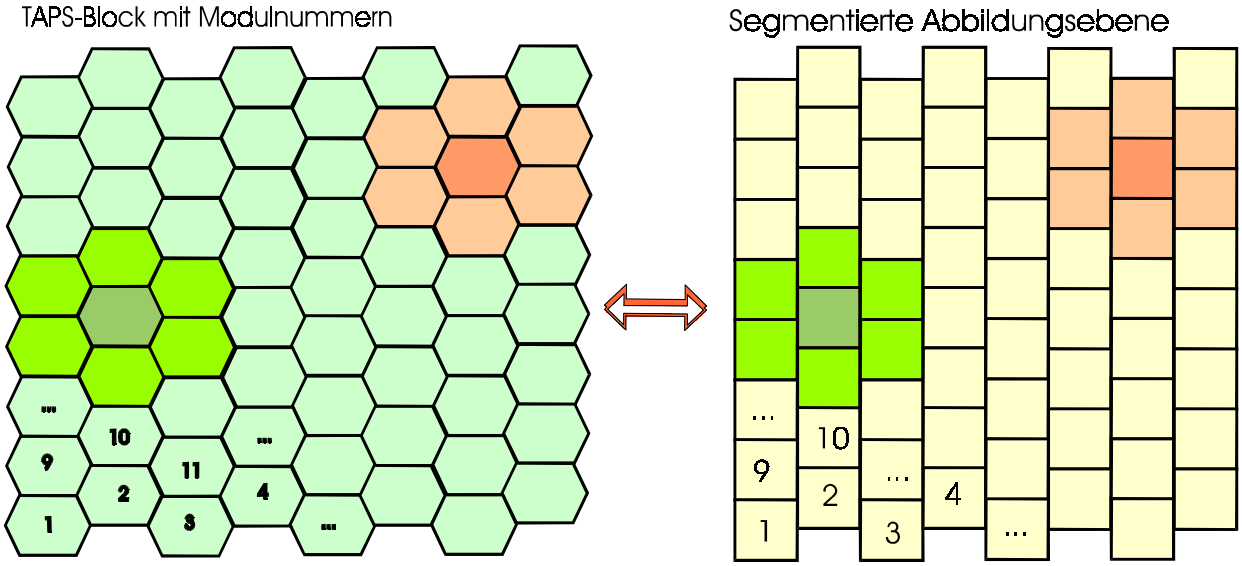


Abbildung 10 Abbildung eines TAPS-Modules aus die segmentierte Ebene

Andererseits führt die Bildung der Datenmatrix aufgrund der hexagonalen Struktur der TAPS-Detektormodule, wie man leicht einsieht, zu zwei unterschiedlichen Abbildern der Schauermuster in der Matrix (siehe Abbildung 11). Die Gestalt der Abbilder bestimmt sich dabei aus der Lage des Schauerzentrums, d.h. der jeweiligen Spaltenzahl in der Matrix. Ist

diese gerade, so liegt das siebte Element eines Kandidatenarrays unterhalb sonst oberhalb des Sechserblocks.

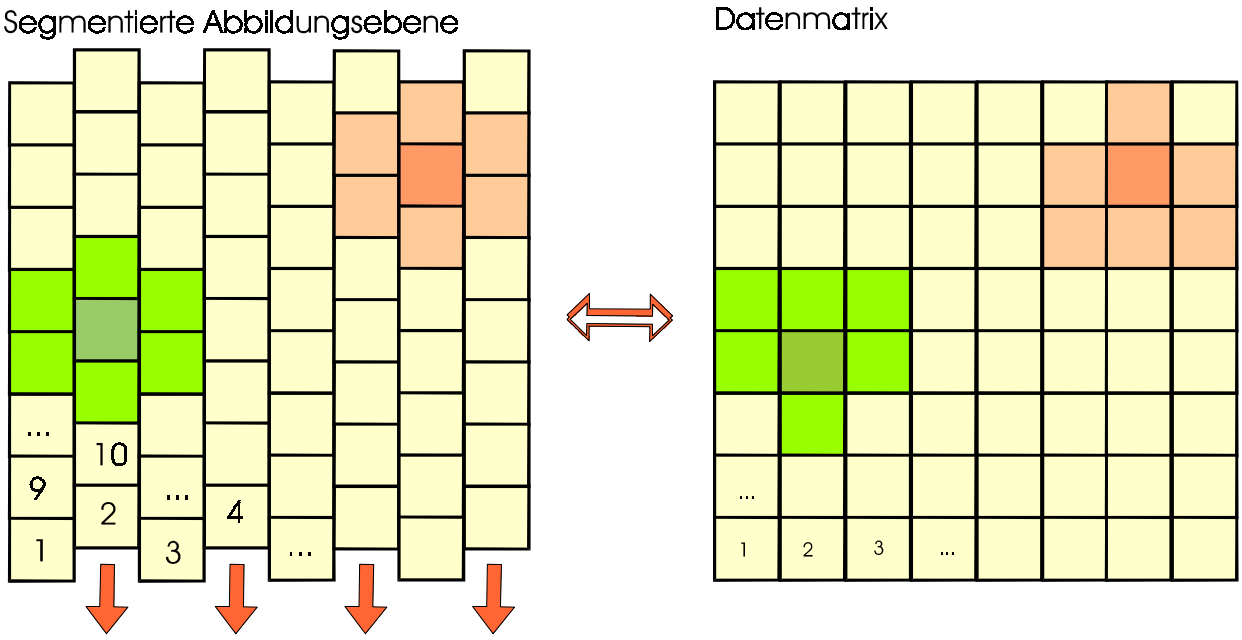


Abbildung 11 Entstehung zweier Datenmuster bei der Matrixbildung

Diese Eigenschaft der Abbildung hexagonaler Strukturen auf einfach quadratische führt nun bei der Umsetzung des neuen Konzeptes dazu, daß die Schauererkennungseinheit in der Lage sein muß, nach zwei Mustern zu suchen.

In unserem Konzept wird nun die Arbeit des TAPS 2nd Level Triggers auf drei Schritte aufgeteilt (siehe Abbildung 12). In einem ersten Schritt werden in einer DSP-basierten Einheit die ankommenden TAPS-Daten relativ zu einander kallibriert sowie auf 8bit reduziert. Außerdem wird es Aufgabe dieser Einheit sein, aus den reduzierten Daten eine Matrix aufzubauen, daher wird diese Stufe auch Matrix-Builder genannt.

Um den kompletten TAPS-Detektor in seiner aktuellen Ausbaustufe mit derzeit 6 Blöcken (angestrebt sind demnächst 8) in einer Matrix erfassen zu können, ist eine Matrizengröße von 32 x 32 ausreichend.

Die so entstandene Matrix wird dann an die eigentliche Schauererkennungseinheit übermittelt, deren Aufgabe es ist, die Datenmatrix nach Schauern abzusuchen und diese Schauerkandidaten mit Orts- und Energiesummeninformation an die letzte Stufe, die Matching Unit (vgl. HADES), zu übertragen. In der Matching Unit schließlich werden die Schauerkandidaten zu Paaren kombiniert und ihre invariante Masse bestimmt. Außerdem können hier die Veto-Informationen zur weiteren Datenreduktion eingebracht werden.

Wie oben bereits erwähnt, vereinfacht das Konzept der Datenmatrix zusammen mit einer Pipeline-Architektur die Parallelisierung des Schauererkennungsprozesses. Es ist nun

möglich, die Datenmatrix zeilenweise in ein Array aus Schauererkennungsprozessoren zu laden. Jeder dieser Prozessoren arbeitet dabei im wesentlichen in drei Stufen (Pipeline-Schritten), so daß nach drei Taktzyklen Vorlaufzeit in jedem weiteren Taktschritt eine Entscheidung fallen kann (siehe die folgenden Abschnitte). Einziges Manko dabei bleibt, daß es aufgrund der oben dargelegten Abbildungseigenschaften Prozessoren für Schauerzentren mit gerader und ungerader Spaltenzahl geben muß.

Bei einer angestrebten Taktfrequenz der Schauerprozessoren von 10 MHz, folgt daß die TAPS-Matrix auf eine 16 x 64 Gestalt gebracht werden kann, da bei Berücksichtigung der Vorlaufzyklen eine Bearbeitung eines TAPS-Events so in ca. 7µs gewährleistet ist.

Für die Erkennung elektromagnetischer Schauer in einem derartigen Umfeld bietet sich die Kombination zweier Algorithmen an. Zum einen muß die Summe der Energie eines Schauerkandidaten eine gewisse Schwelle überschreiten (Gruppensummen-Algorithmus), und zum anderen muß das zentrale Element eines Musters im Vergleich zu den Nachbarn den höchsten Energieinhalt haben (Lokale Maximum Bedingung). Sind beide Bedingungen erfüllt, gilt das zentrale Detektor dieser Gruppe als Schauerzentrum.

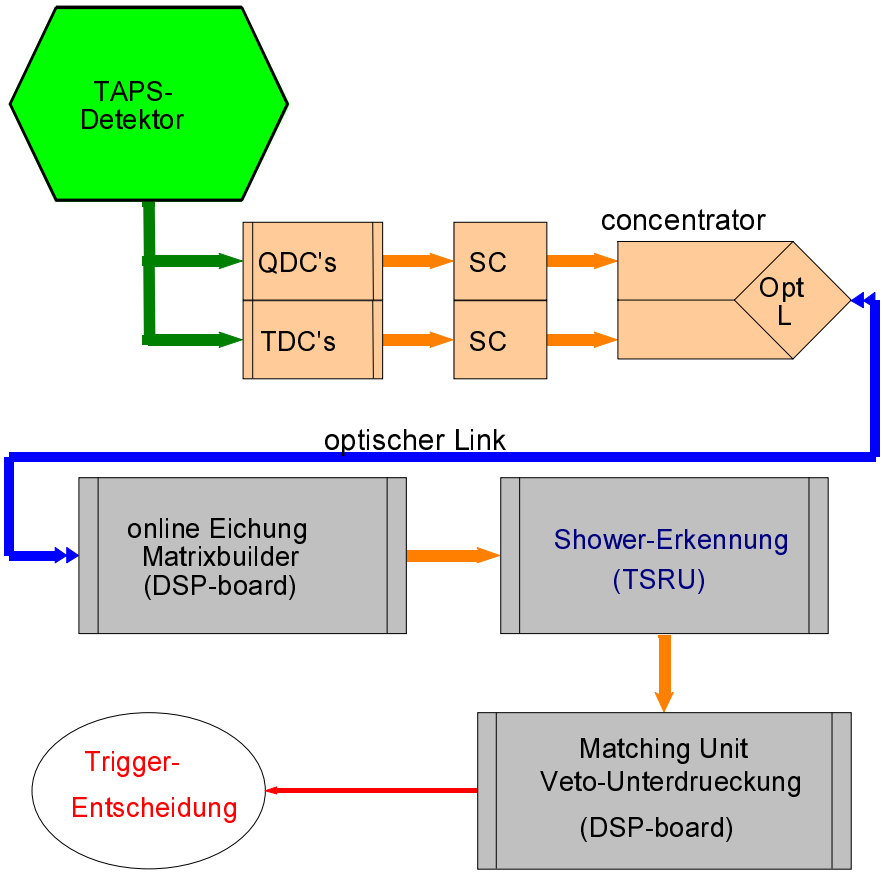


Abbildung 12 Schema des TAPS 2nd Level Triggers mit Frontend-Datenaufnahmesystem (vgl. HADES TOF-DAQ)

2.4 Die TAPS Schauererkennungseinheit (TSRU)

Nachdem nun in Abschnitt 2.3 das angestrebte Gesamtkonzept des TAPS 2nd Level Triggers vorgestellt wurde, möchte ich im folgenden näher auf das Konzept der Schauererkennungs-einheit eingehen. Ziel dieser Untereinheit des Triggers ist auf der übermittelten Datenmatrix Schauerkandidaten zu identifizieren. Wie in Abschnitt 2.3 bereits dargestellt, scheint eine Kombination zweier Algorithmen die geeignete Methode zu sein, wobei aus den oben erwähnten Gründen die Algorithmen je nach Spaltennummer leicht modifiziert werden müssen.

2.4.1 Der Gruppensummen-Algorithmus

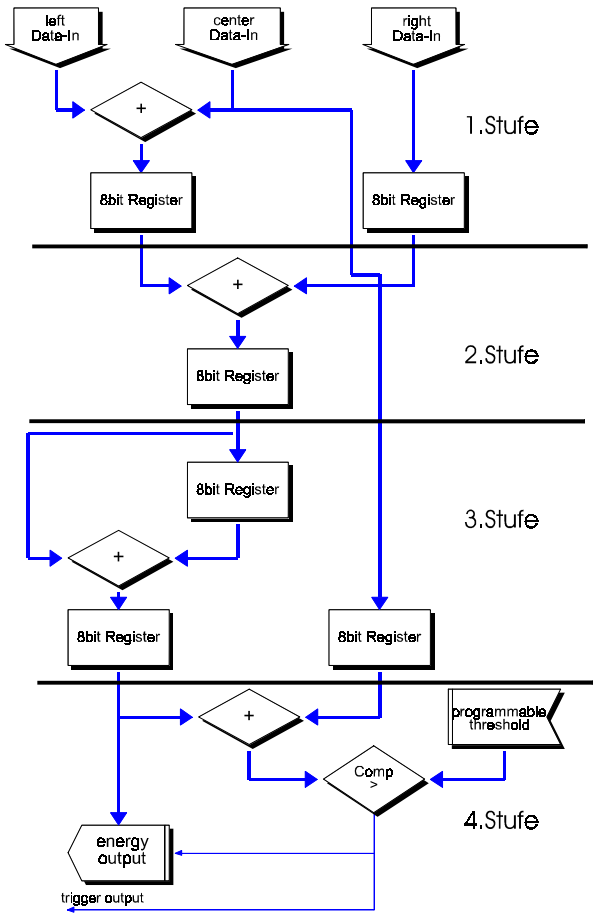


Abbildung 13 möglicher GS-Algorithmus (für gerade Spaltennummern [für ungerade Spaltennummern siehe Anhang]. Besonders sei darauf hingewiesen, daß aufgrund des Pipelinings, in jeder Stufe verschiedene potentielle Schauergruppen gleichzeitig bearbeitet werden können. Jede Stufe entspricht also einem Pipeline-Schritt.

Aufgabe des Gruppensummen-Algorithmus (GS-Alg.) ist die Bestimmung der Energiesumme aller näheren, zu einem möglichen Schauerereignis gehörenden TAPS-Module (siehe 1.3.2). Dazu wird die Summe der Matrixinhalte in vier Additionsschritten gebildet und diese mit einer Schwelle verglichen. Liegt die Summe über dieser Schwelle, so liefert der GS-Algorithmus eine positive Entscheidung.

Ein entscheidender Vorteil des neben dargestellten Algorithmus[25] liegt in der Fortführung der Pipeline-Architektur, die es erlaubt, mit jedem Taktzyklus eine Entscheidung zu treffen.

Neben der eigentlichen Entscheidung liefert dieser Algorithmus auch die Energiesumme im reduzierten 8bit-Format als Ergebnis

[25] Algorithmen für ungerade Spaltennummern siehe Anhang

2.4.2 Ein Algorithmus zur Suche lokaler Maxima

Nachdem im vorangegangenen Abschnitt ein möglicher Algorithmus für die Gruppen-summenbedingung vorgestellt wurde, soll an dieser Stelle noch eine Möglichkeit zur Bestimmung des lokalen Maximums (LM) aufgezeigt werden. Eine zusätzliche Randbedingung für einen LM-Algorithmus wird nun sein, daß er synchron zu dem zugehörigen GS-Algorithmus arbeiten muß. Den nur so kann sicher gestellt werden, daß beide Entscheidungen zu den selben Rohdaten gehört haben. In einem Pipeline-ähnlichen Ansatz werden bei diesem Algorithmus in jedem Taktschritt alle sechs peripheren Energiewerte mit einem möglichen Zentralwert verglichen. Abbildung 14 zeigt eine Möglichkeit der Verwirklichung eines derartigen Algorithmus.

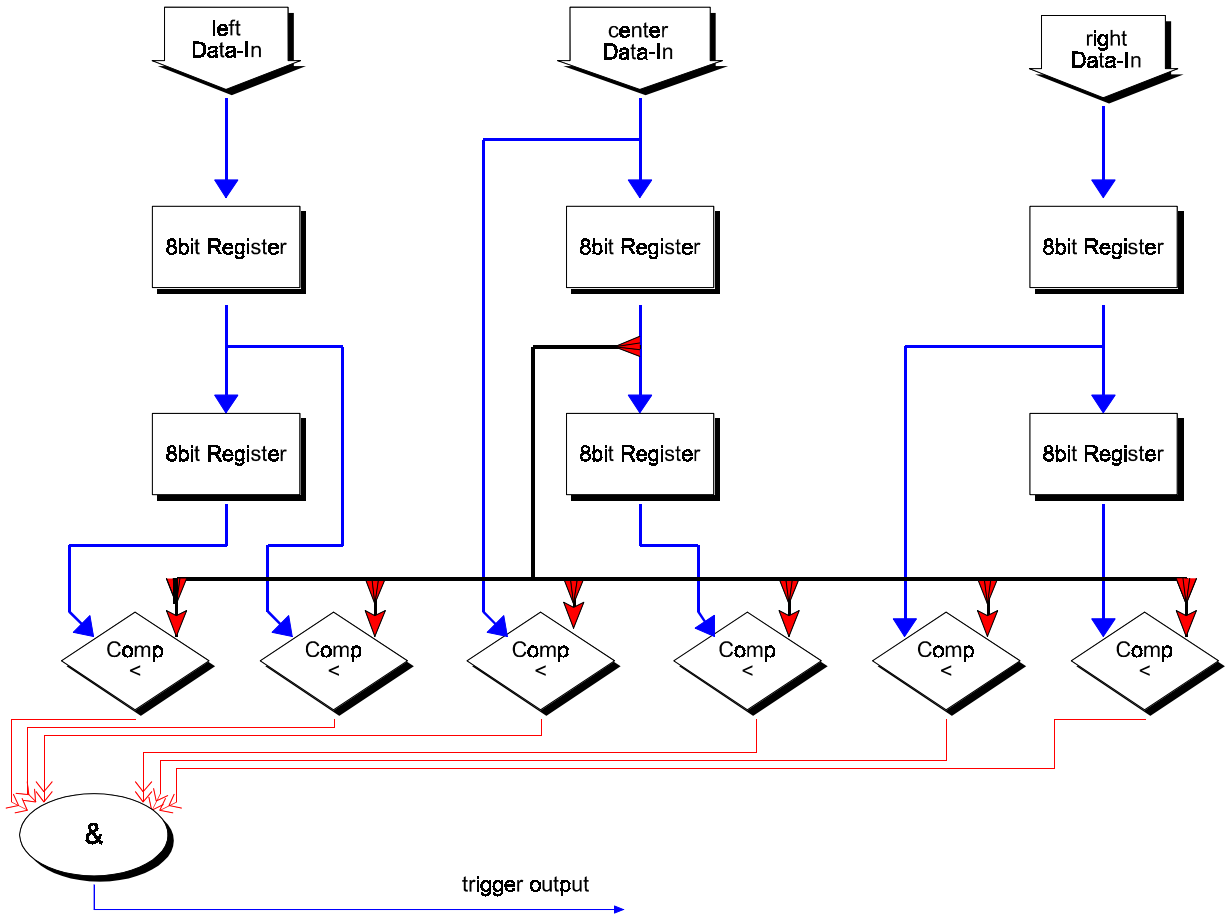


Abbildung 14 Beispiel eines Algorithmus zur „Lokalen-Maximum-Suche“ für gerade Spaltennummern

Diese Algorithmen werden nun in einem Schauererkennungsprozessor parallel betrieben und ihre Ergebnisse über eine UND-Verknüpfung gekoppelt. Als Ergebnis liefert ein Prozessor neben der Schauersumme noch die Position des Schauerzentrums innerhalb der Datenmatrix, so daß eine Rekonstruktion in der Matching Unit möglich wird.

Neben diesen Hauptelementen verfügt eine Schauererkennungseinheit über eine Logik zur Behandlung der Event-ID sowie zur Generierung eines Event-Ende-Zeichens. Abbildung 15 zeigt den schematischen Aufbau eines Schauererkennungsprozessors (SRU).

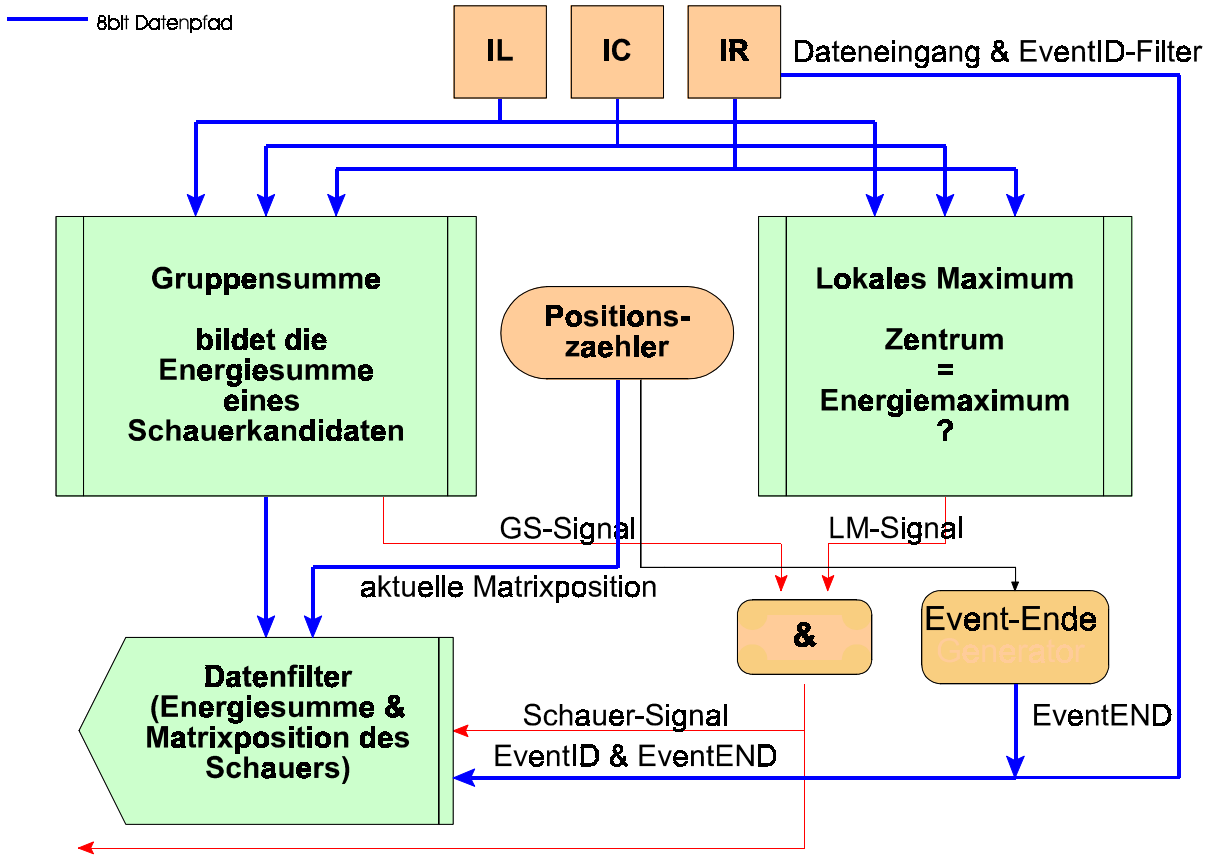


Abbildung 15 Schemazeichnung eines Schauererkennungsprozessors

3 Prototyp einer Schauererkennungseinheit

Nachdem in den beiden vorangegangenen Kapiteln sowohl die physikalischen Zielsetzungen als auch die grundlegenden Konzepte, die zu dieser Arbeit Anlaß gegeben haben, vorgestellt wurden, soll nun die Umsetzung des in Abschnitt 2.4 dargelegten Konzeptes für eine Schauererkennungseinheit eingehend erörtert werden.

3.1 Aufgabenstellung des Prototypen

Ausgehend von den Vorgaben, die sich aus den Kapiteln 1 und 2 ergeben, sowie Meßergebnissen und Simulationen der TAPS-Kollaboration wurden ein schnelle, hardwarebasierte Algorithmen entwickelt (siehe Konzeptalgorithmen in Abschnitt 2.4.1 und 2.4.2). Diese sind so ausgelegt, daß in FPGAs (**F**ield **P**rogrammable **G**ate **A**rrays) implementiert werden können. Außerdem wurde eine VME-Karte (**V**ersa **M**odule **E**urocard) entworfen, die alle wesentlichen Merkmale der zukünftigen Schauererkennungseinheit (TSRU – **T**APS **S**hower **R**ecognition **U**nit) des TAPS 2nd Level Triggers aufweist.
Aufgabe diese Prototypen ist es nun sowohl die Funktionalität der erarbeiteten Algorithmen zu überprüfen als auch eine Performance-Studie der TSRU zu erlauben. Ferner sollten in dem vorliegenden Prototypen folgende Randbedingungen erfüllt sein:

- Das Prototypboard sollte wie die spätere TSRU auf einem VME-Slave-Board verwirklicht werden, da die gesamte HADES-Datenaufnahme auf diesem Busprotokoll beruht (=> optimale Anpassung an HADES).

- Die später im Experiment notwendige Taktfrequenz von 10MHz für die eigentlichen Schauererkennungsprozessoren sollte schon im Prototyp realisiert werden, damit eine genaue Performance-Analyse möglich wird.

- Die Konfiguration der XILINX-FPGA soll schon im Prototyp genauso ablaufen, wie später beim Betrieb der TSRU geplant.

- Die im Prototyp-Test verwendeten Algorithmen sollen auch in einem späteren Einsatz mit nur geringen Modifikationen einsetzbar sein.

Abbildung 16 (umseitig) zeigt ein Funktionsschema des Prototyps mit allen wichtigen Datenpfaden.

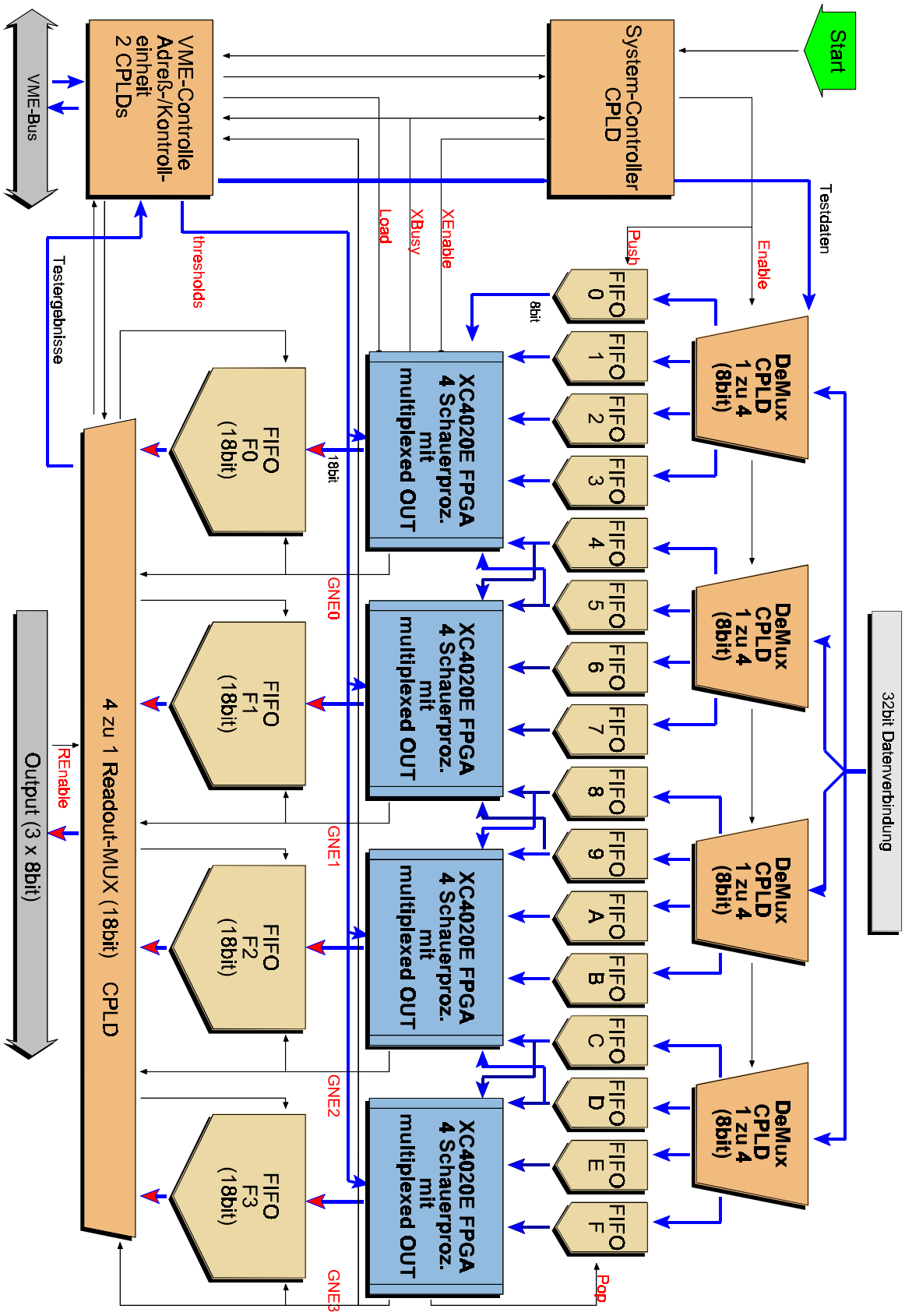


Abbildung 16 Funktionsschema der Prototyp-VME-Karte mit allen wichtigen Datenpfaden. Neben dem Hauptdatenpfad mit den beiden Frontsteckern ist auch der Testdatenpfad gezeigt, der einen Test der Algorithmen über VME-Zugriffe erlaubt.

Bevor ich nun im Detail auf den Prototypen und seine Funktionsweise eingehen werde, möchte ich kurz einige Worte über die verwendeten Hardwarekomponenten verlieren. Auch soll das VME-Busprotokoll und das VME-Interface kurz vorgestellt werden.

3.2 Der VMEbus

Der VMEbus[26] wurde 1981 von den Firmen Motorola, Mostek und Signetics als Interface-Standard für die Verbindung verschiedener DV-Module über ein gemeinsames, mikroprozessorunabhängiges Bussystem geschaffen. Dieser Standard umfaßt neben der Spezifikation des Busprotokolls[27] auch mechanische Richtlinien über die Beschaffenheit von VME-Modulen (Größe des PCB[28], Art und Geometrie der Steckverbindungen etc.).
Die grundlegende Architektur des VME-Standards folgt dabei dem Master-Slave-Prinzip, d.h. eine eigenständiges Modul, Master genannt, kontrolliert den Datentransfer von und zu den Slave-Modulen. Das für diesen Transfer verwendete Busprotokoll ist dabei vollständig asynchron, d.h. das langsamste Modul in einem jeweiligen Zyklus bestimmt die Datentransfergeschwindigkeit. Dabei stehen hier vier verschiedene Modi für den Datentransfer zur Verfügung:

- Read/Write
- Block-Transfer
- Read-Modify-Write
- Adress-only

Damit erlaubt der 32bit breite Adress- und Datenbus im schnellsten Modus (Block-Transfer) eine maximale Übertragungsrate von 40Mbyte/s. Eine Erhöhung der Übertragungs-geschwindigkeit ist mit Einführung des 64bit breiten VME64-Standard vorgesehen.
Das Busprotokoll des Read/Write-Modus, welcher als Standardmodus für den Prototypen (Slave-Modul) Verwendung findet, ist in Abbildung 17 zu sehen. In diesem Modus wird in jedem Zyklus der Empfänger explizit durch Treiben des Adreßbusses (A01-A31) mit der entsprechenden Adresse des Slave-Moduls angesprochen. Dazu werden neben der Adresse zusätzlich zwei Kontrollsignale (IACK* = Interrupt Acknowledge / LWORD* = Long Word[29]) durch das Master-Modul generiert. Das AS*-Strobe (Adress-Strobe) markiert die

[26] VME93 - W.D.Peterson, The VMEbus Handbook, VITA Verlag (1993)

[27] Busprotokoll – spezifiziert den Datentransfer auf dem Bus

[28] PCB – Printed Circuit Board => Platine

[29] ‚xxx*' – markiert ‚active low'-Signale (inverse Logik)

Gültigkeit der anliegenden Adresse. Die Gültigkeit des eigentlichen Datenwortes wird schließlich durch beiden Daten-Strobes (DSA* und DSB*) gekennzeichnet. Ferner wird mittels des behaupteten WRITE*-Signals schließlich festgelegt, ob eine Lese- oder ein Schreibzugriff erfolgt.

Sollen nun in einem Zyklus Daten in ein Slave-Modul geschrieben werden, so muß dieses nach Empfang der Daten, diesen mit einem DTACK*-Strobe (Data Acknowledge) quittieren. Erscheint DTACK*-Signal auf dem Bus, wird ein Bus-Error ausgelöst (BERR*).

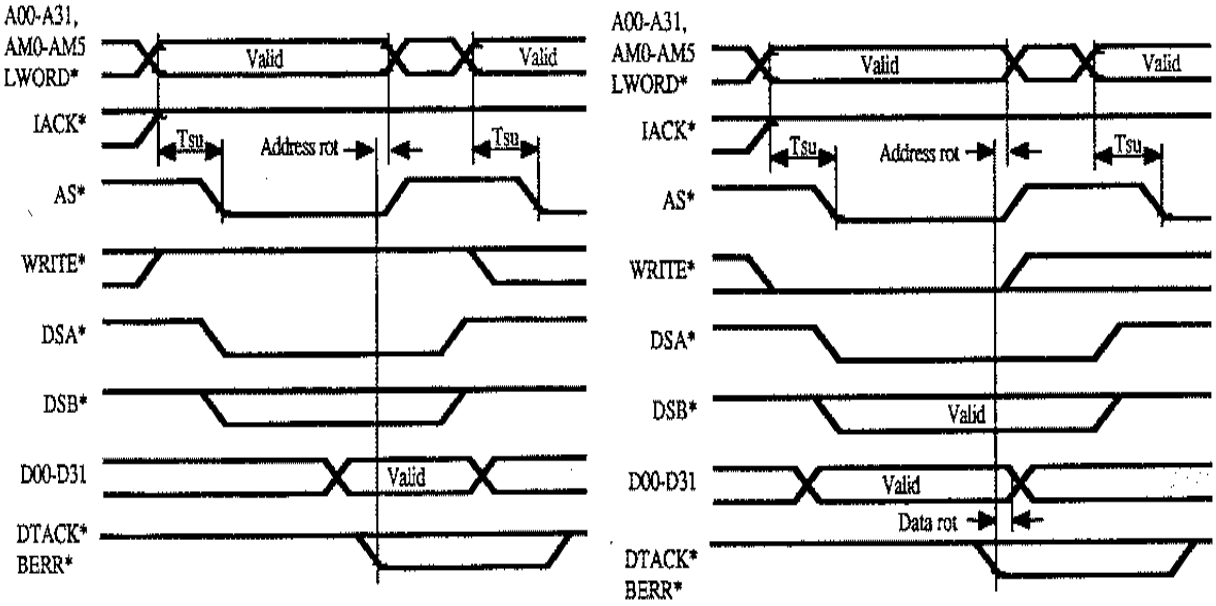


Abbildung 17 Die 1. Abbildung zeigt das Timingverhalten der wichtigsten Signale während eines typischen VME-Lesezuriff. In der 2. Abbildung ist ein typischer Schreibzugriff dargestellt. Die mittlere Zugriffszeit beträgt in beiden Fällen ca. 1µs (siehe VME93).

Auf dem Prototyp-Board ist nun das VME-Interface in zwei getrennten CPLD-Bausteinen der Firma LATTICE implementiert, wobei der eine einen Adress-Decoder/Konverter beherbergt, während der andere die eigentlichen Kontrollaufgaben, wie z.B. Steuerung der Schreib/Lese-Zugriffe, Konfiguration der FPGAs etc., übernimmt (siehe Abschnitt 3.4).

3.3 FPGA- und CPLD-Bausteine

Aufgrund der hohen Komplexität moderner, logischer Schaltungen scheint es inzwischen unmöglich geworden zu sein, diese aus Standard-Bauelementen der TTL- oder CMOS-Familien aufzubauen. Zum einen bieten derartige Bausteine meist eine für den Anwender zu sehr eingeschränkte Funktionalität mit sich, und zu anderen werden die eigentlichen logischen Verknüpfungen erst auf der Platine hergestellt, was eine sehr geringe Fehlerredundanz mit

sich bringt. Diese Nachteile herkömmlicher Standard-Bauteile können durch den Einsatz komplexer, programmierbarer Bausteine (PLDs) weitgehend umgangen werden. Zu diesen Bausteinen gehören neben PAL- und GAL-Bausteinen auch die sogenannten CPLD- (Complex Programmable Logic Device) und FPGA-Bausteine (Field Programmable Gate Array). Diese beiden Typen der Familie der PLDs unterscheiden sich grundlegend in ihrer inneren Architektur.

Abbildung 18 zeigt eine schematische Darstellung eines CPLD-Bausteins. Es handelt sich dabei um einen Baustein vom Typ 1032[30] der Firma LATTICE, wie auf dem Prototyp-Board mehrfach Verwendung (Demultiplexer, Multiplexer & VME-Kontroller) findet. Außer diesem Typ werden ferner zwei Bausteine vom Typ 1024 verwendet, die aber im wesentlichen die gleiche Architektur aufweisen.

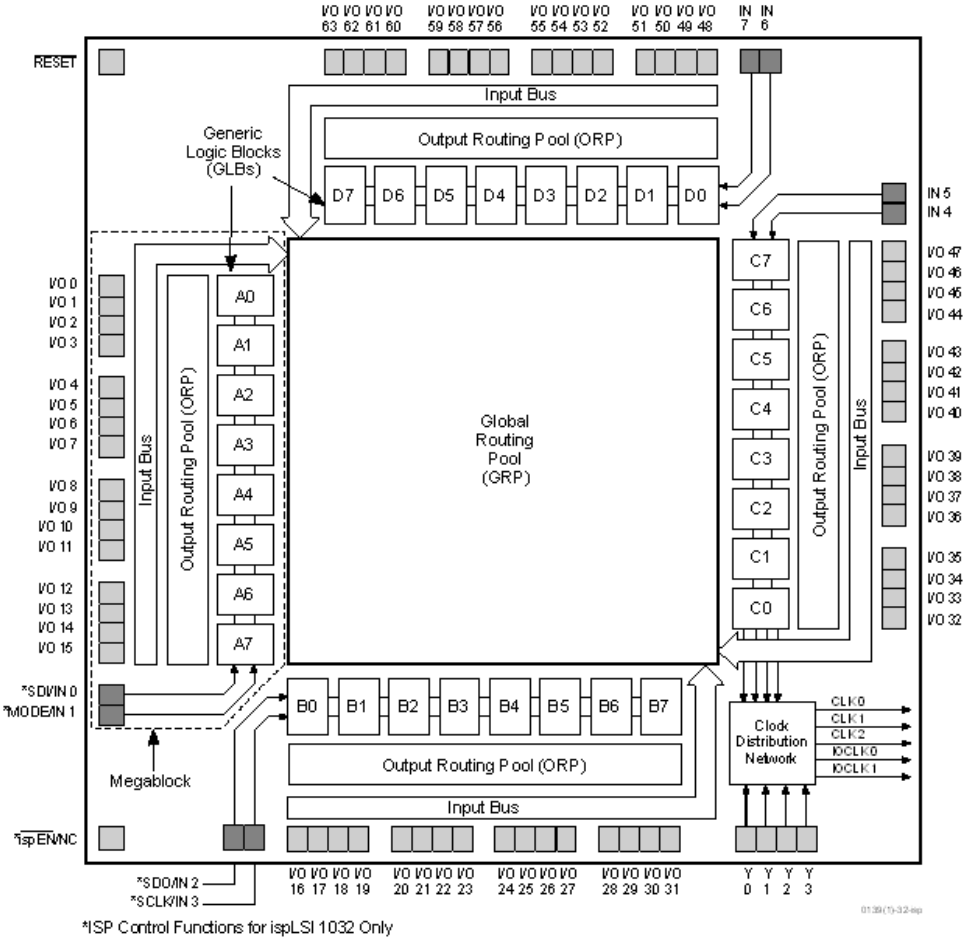


Abbildung 18 innere Architektur eines CPLD am Beispiel eines LATTICE 1032

[30] LAT94 - Lattice Datenbuch 1994

Wie man deutlich erkennen kann, basiert die Architektur eines CPLD-Bausteines aus vier grundlegenden Elementen. Auf der einen Seite finden sich zwei Gruppen von sogenannten Routing-Resourcen (4 x Output Routing Pool / 1 x Global Routing Pool), mit denen die verschiedenen Elemente miteinander verbunden werden können. Wesentliches Merkmal ist dabei die genau definierte Signallaufzeit durch diese Routing-Blocks (Pin-to-Pin-Zeit). Auf der anderen Seite findet man noch IO-Makrozellen und sogenannte Generic Logic Blocks (GLB), welche die eigentliche programmierbare Logik-Funktionen enthalten.
Die komplette Architektur eines derartigen CPLD-Bausteines ist dabei auf der E^2CMOS-Technologie aufgebaut, so daß diese Bausteine einerseits mehrfach rekonfiguriert werden können und andererseits ihr Programm auch bei Spannungsverlust beibehalten. Ausgehend von dieser Technologie sind inzwischen Bausteine verfügbar, welche sich „im System“,d.h. auf der Platine, programmieren lassen.
Für die CPLD-Programmierung stehen prinzipiell mehrere Wege offen, so reicht die Spannweite der verfügbaren Designmethoden von Tools mit schematischen Designentry mit Makrobibliotheken bis hin zu Hardware-Beschreibungssprachen wie z.B. ABEL-HDL der Firma DATA-IO. Für die Programmierung des Prototypen wurde aus dieser Vielfalt der Weg über die Beschreibungssprache ABEL (siehe Abbildung 19) gewählt [Im Anhang finden sich einige ABEL-Sources der Peripherie-Chips des Prototyp-Boards.].

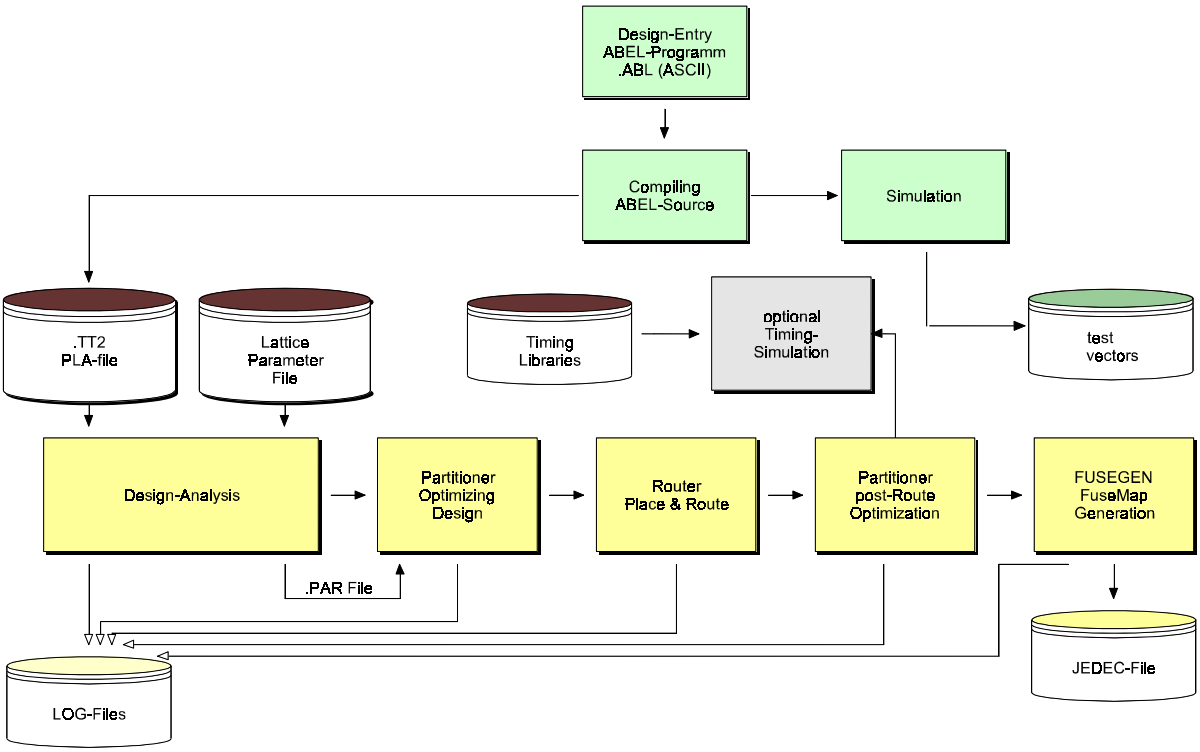


Abbildung 19 ABEL-Design-Flow für CPLDs der Firma LATTICE, wobei der Kasten (pDS+ Fitter) hierbei den hardwareabhängigen Teil markiert.

Im Gegensatz zu den CPLD-Bausteinen sind FPGAs auf SRAM-Strukturen aufgebaut, woraus sofort die Flüchtigkeit der Programmierung bei Spannungsverlust als ein grundlegender Unterschied folgt. Andererseits lassen sich mit dieser Technologie deutlich komplexere Bausteine entwickeln, die gegenüber den CPLDs einen deutlichen Gewinn an Flexibilität und Einsatzspektrum verbuchen können. Wie in den CPLD-Bausteinen finden sich auch in den FPGAs zwei Gruppen von aktiven Elementen – sogenannte IOBs (IO-Blöcke) und CLBs (konfigurierbare Logikblöcke). Diese Gruppen können nun über Netze von Verbindungen [Verbindungsmatrizen]. Neben diesen konventionellen Verbindungen zwischen verschiedenen aktiven Elemente existieren zusätzlichen besonders schnelle Verbindungsnetze für die Verteilung von Clock-Signalen etc.

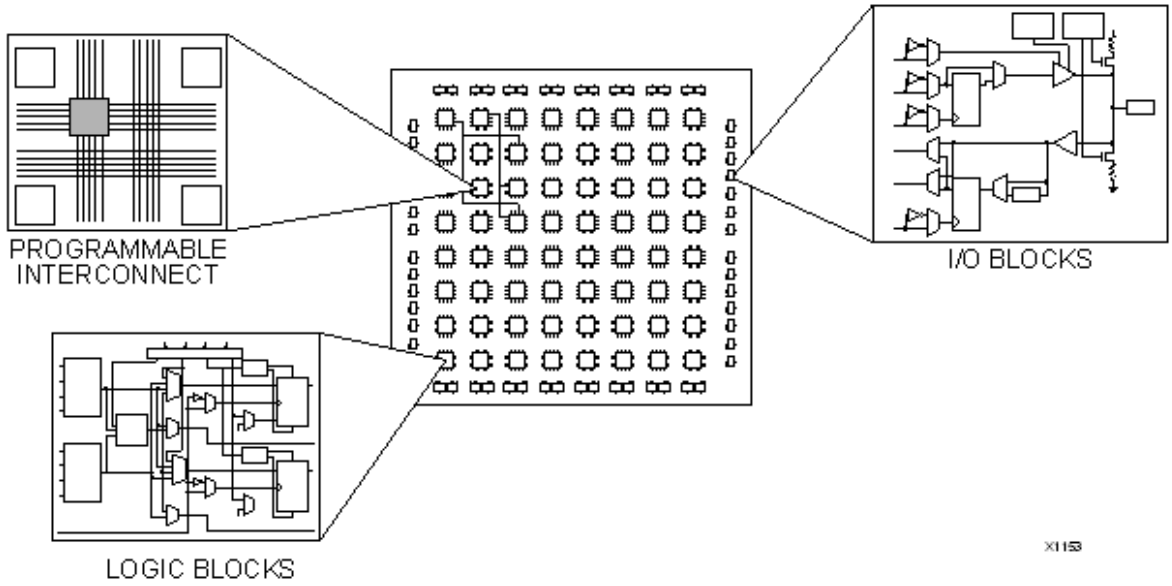


Abbildung 20 grundlegender Aufbau eines XILINX-FPGAs

Aufgrund ihrer höheren Komplexität, die im wesentlichen auf der Möglichkeit beruht, mehrere CLBs nacheinander in einer Folgekette zu koppeln, ist es bei FPGAs nicht möglich von vornherein ein genaues Zeitverhalten festzulegen, wie dies bei CPLDs aufgrund der festen Pin-to-Pin-Zeit möglich war. Das Design einer logischen Schaltung mit FPGAs ist daher bedeutend aufwendiger und bedarf ausgeklügelter Simulationstechniken zur Verifikation. Dabei kann man zwei Simulationsarten unterscheiden. Zum einen ist es möglich das Design direkt nach dem Entry mit Funktionalsimulation auf seine Brauchbarkeit hin zu überprüfen. Für eine genaue Aussage über die Einsetzbarkeit ist letztlich allerdings eine Timingsimulation von Nöten, welche nach der eigentlichen Übersetzung für einen bestimmten FPGA-Typ durchgeführt werden kann.

Auch für FPGAs gibt es eine Fülle unterschiedlicher Designentry-Methoden. So ist eine Eingabe sowohl in einer ABEL-verwandten Hochsprache oder aber in VHDL möglich als

auch auf graphischem Wege. Bei der Entwicklung der Schauererkennungseinheit kam hierbei ein graphisches Tool zum Tragen, da es mehr Überblick über das gesamte Design gewährt als ein Programmquelltext. Außerdem bestand die Möglichkeit der Kommunikation zwischen Schematic-Tool und Simulationsprogramm, so daß alle Zustände während der Simulation direkt sichtbar gemacht werden konnten.
Nach eingehenden Studien zur Performance der Algorithmen in verschiedenen Simulationen, wurde schließlich ein FPGA vom Typ 4020E[31] für das Prototyp-Board ausgewählt, da dieser sowohl die geforderte Performance gewährleistet als auch über zusätzliche Resourcen für etwaige Änderungen und Ergänzungen im Design verfügt.

3.4 Die Prototyp-Platine

Wie Abbildung 16 bereits gezeigt hat, besteht die als VME-Slave-Modul konzipierte Prototyp-Platine im wesentlichen aus den fünf Funktionseinheiten VME-Interface, Demultiplexer, Schauererkennungsprozessoren, Systemcontroller und Multiplexer sowie zwei in FIFO-Gruppen organisierten Speicherbereichen, wobei die Schauererkennungseinheiten in vier FPGAs implementiert sind, während alle anderen Funktionselemente in CPLDs verwirklicht wurden. Es seien nun kurz die einzelnen Aufgaben der Funktionsgruppen näher erläutert.
Aufgabe des **VME-Interfaces**, welches in zwei getrennten CPLD-Bausteinen verwirklicht wurde, ist zum einen die Herstellung der Bus-Kommunikation und zum anderen die Steuerung aller für den Testbetrieb notwendigen Funktionen des Prototyp-Boards. Eine weitere fundamentale Funktion dieser Einheit liegt in der Steuerung des Konfigurationsprozesses der FPGAs, da diese, wie bereits erwähnt, grundsätzlich bei jedem Neustart konfiguriert werden müssen.
Um diese Aufgaben zu erfüllen, werden zunächst in einem Adress-Decoder auf dem VME-Bus anliegende Adressen (es wird nur ein Adreßraum von 24bit verwendet) decodiert und gegebenenfalls in Befehlscodes des Prototypen-Boards übersetzt (siehe Tabelle 2). Dabei wird auf das Konzept der Offset-Adressierung zurückgegriffen, so daß es eine Boardadresse gibt, zu der alle anderen Adressen einfach addiert werden. So deckt die TSRU in unserem Testbetrieb einen Adressraum von 800000h bis 80008fh ab.

[31] XIL96 - XILINX Datenbuch 1996

Der eigentliche VME-Controller führt bei entsprechender Adressierung dann eine der folgenden Funktionen aus:

- Set/Reset-Steuerung der unterschiedlichen Funktionsgruppen
- Steuerung der FPGA-Konfiguration
- Bereitstellung eines Statusregisters

Tabelle 2 Adressen des TSRU-Prototypen im Testbetrieb

Adresse (24bit [hex])	**Zusätzlich nötiges Datenwort (8bit)**	**Funktion der TSRU**
800000	-	Boardbasis-Adresse (ohne Funktion)
Offset-Adresse (8bit [hex])		**Funktionsbeschreibung**
02	-	RESET des Eingabespeicherbereichs
04	-	RESET des Ausgabespeicherbereichs
06	-	RESET des Demultiplexers
08	-	RESET des Multiplexers
0a	-	RESET der Schauererkennnungseinheiten
0c	-	RESET des Systemsontrollers
0e	-	SYNCHRONISATION der Boardclock (10/20 MHz)
10	01	Setzen des PROG-Pins für die XILINX-Konfiguration
	02	Rücksetzen des PROG-Pins
	04	Setzt DIN auf HIGH (Konfigurationsdatenbit)
	08	Setzt DIN auf LOW
	10	Setzt CCLK auf HIGH (Konfigurationsclock)
	20	Setzt CCLK auf LOW
12	-	Auslese des XILINX-Statusregisters
14	XX	Setzt unteren Energiesummenschwelle
16	XX	Setzt obere Energiesummenschwelle
20,22,..,3e	XX	Schreibt Datenwort XX in InputFIFO 0..15
40	-	Führt Lesezugriff auf den Ausgangsspeicherbereich aus
42	-	Übermittelt obere 8bit des gelesenen Datenwortes
44	-	Übermittelt mittlere 8bit des Datenwortes
46	-	Übermittelt untere 8bit des Datenwortes
50	-	Startet die Schauererkennungseinheit
52	-	Hält Schauererkennnungsprozessoren an
80	01	Schaltet den Prototypen in den TESTMODE
	02	Rücksetzen des TESTMODE

Die **Demultiplexer-Einheit** (4 CPLDs vom Typ 1032) hat zwei verschiedene Aufgaben. Im späteren Normalbetrieb soll diese Einheit das Verteilen der über den Front-Stecker (40polig) empfangenen 32bit breiten Datenpakete des Matrixbuilders auf die Eingangsmatrixspeicher (1. FIFO-Gruppe) übernehmen. Während des Testbetriebs ist diese Aufgabe leicht verändert, da im Testmodus nur 8bit breite Datenworte angeliefert werden können, so daß die FIFOs gezielt über diese CPLDs adressiert werden können. Außerdem markiert der Demultiplexer das erste Datenwort in jedem FIFO als EventID. Das Konzept der EventIDs wurde in der HADES-Datenaufnahme zu verbesserten Kontrolle eingeführt, um eine Vermischung verschiedener Events in der Pipeline zu verhindern.
Ist nun ein komplettes Event im Eingangsspeicherbereich (Matrixspeicher) abgelegt worden, so ist es Aufgabe des **Systemcontrollers** die Schauererkennungseinheiten zu starten. Außerdem kontrolliert diese Einheit den Status der TSRU und löst zum Beispiel bei Überfüllung des Ausgangsspeichers ein entsprechendes Signal (TSRUBUSY) aus. Im Testbetrieb ist auch hier ein direkter Eingriff über einen entsprechenden Befehl möglich.
Der **Multiplexer** hat schließlich zur Aufgabe die gefundenen Schauerkandidaten, die von den Schauererkennungsprozessoren im Ausgangsspeicherbereich abgelegt wurden, zu sammeln und weiterzuleiten. Dabei muß zusätzlich eine Formatkonversion in drei 8bit Worte erfolgen, da die Schauererkennungseinheiten 18bit breite Ergebnisse liefern, wobei die obersten 10bit die jeweilige Position des Schauerzentrums in der Matrix und die untersten 8bit die reduzierte Energiesumme beinhalten. Der Ausgangsspeicher wird dabei im sogenannten „Polling-Mode“ betrieben, d.h. der Multiplexer liest solange aus einer festen Ausgangsfifogruppe bis er dort auf ein Event-Ende-Wort stößt, welches die jeweilige Schauererkennungseinheit zum Abschluß einer jeden Eventbearbeitung in diese Speicher schreibt. Stößt der Multiplexer nun im Verlauf des Ausleseprozesses auf eine derartige Kennung, wechselt er die FIFO-Gruppe und der Prozeß beginnt von vorn (Format der Ausgabedaten siehe Anhang).
Aufgabe der FPGA-basierten **Schauererkennungseinheiten** ist nun die eigentliche Suche Schauerkandidaten auf der übergebenen Datenmatrix. Während der Bearbeitung eines Events wird nun zunächst die EventID ausmaskiert und unbearbeitet in den Ausgangsspeicher geschrieben. Danach beginnt mit der zweiten Matrixzeile die eigentliche Verarbeitung des Events. Ist die letzte Matrixzeile bearbeitet und ein mögliches Ergebnis gespeichert, generiert die Schauererkennungseinheit, wie schon erwähnt, ein Event-Ende-Wort und geht in den Ruhezustand zurück.
Die Verarbeitung der Eventdaten in der Schauererkennungseinheit geschieht, wie in Abschnitt 2.4 eingeführt, in zwei separaten Algorithmen, wobei diese wiederum aus den bereits

erwähnten Gründen abwechselnd in zwei Gruppen zusammen gefaßt wurden. Die Abbildungen 21 und 22 zeigen die beiden hardwareimplementierfähigen Algorithmen für die Schauersuche auf geradzahligen Matrixspalten (vgl. Abbildungen 13 & 14), wie sie schematischen Designentry entwickelt wurden. Die beiden Algorithmen bilden dabei bei der Designeingabe mit dem am Institut vorhandenen Viewlogic-Paket die unterste Schicht eines mehrlagigen Designs. Es handelt sich also um sogenannte Softmakros, welche selbst aus sogenannten Fixedmakros aus Firmenbibliotheken ausgebaut sind, welche wiederum auf im FPGA implementierte Hardmakros und Strukturen zurückgreifen.
Um Randeffekte durch das Auftreten von Überträgen bei der Summation von 8bit-Werten in 8bittigen Addierern zu verhindern, wurden beim Gruppensummen-Algorithmus 16bit breite Addierer (Makros der Xilinx-Bibliothek) verwendet. Am Ende der Summation werden diese Daten dann je nach Bedarf auf 8bit Breite reduziert.
Falls beide Algorithmen zu einer positiven Entscheidung führen, wird die Energiesumme (8bit) zusammen mit der Position des Zentrums (10bit = 4bit Matrixspalte + 6bit Matrixzeile) als Ergebnis in den Ausgangsfifos gespeichert.
Wie schon erwähnt, wird die Konfiguration dieser FPGAs vom VME-Kontroller aus gesteuert. Die FPGAs kennen dabei verschiedene Modi der Konfiguration (siehe XILINX Datenbuch 1996). Bei der Entwicklung der Prototyp-Platine wurde nun der sogenannte Serial-Slave-Mode gewählt, bei dem die Xilinx FPGAs die Konfigurationsdaten in einem seriellen Bitmuster empfangen, daß ihnen von einem Master, in dem Fall dem VME-Kontroller, übermittelt wird[32]. Außerdem bieten die FPGAs die Möglichkeit, eine Konfigurations-daisy-chain zu entwerfen, so daß alle vier FPGAs in einem Zyklus nacheinander konfiguriert werden können. Dazu müssen jedoch die Konfiguartionsdateien derart aneinandergefügt werden, daß alle Headerzeilen entfernt werden und ein neuer Header mit einer aktualisierten Dateilänge (Anzahl der clockcycles bis zur vollständigen Konfiguration) generiert wird (siehe Quelltext zum Merging-Programm im Anhang).
Die **Eingabe- und Ausgabespeicher** wurden in FIFOs vom Typ 7C421 der Firma CYPRESS realisiert, welche eine Breite von 9bit bei einer Tiefe von 512k Worten bereitstellen. Im Ausgangsspeicherbereich wurden zusätzlich immer zwei dieser FIFOs zu Gruppen zusammengefaßt, was eine Breite der Ausgabedaten von 18bit erlaubte.

[32] KRA96 - H.Kraft, Diplomarbeit 1996, Ein Prozessor zur schnellen Ringerkennung für den HADES RICH

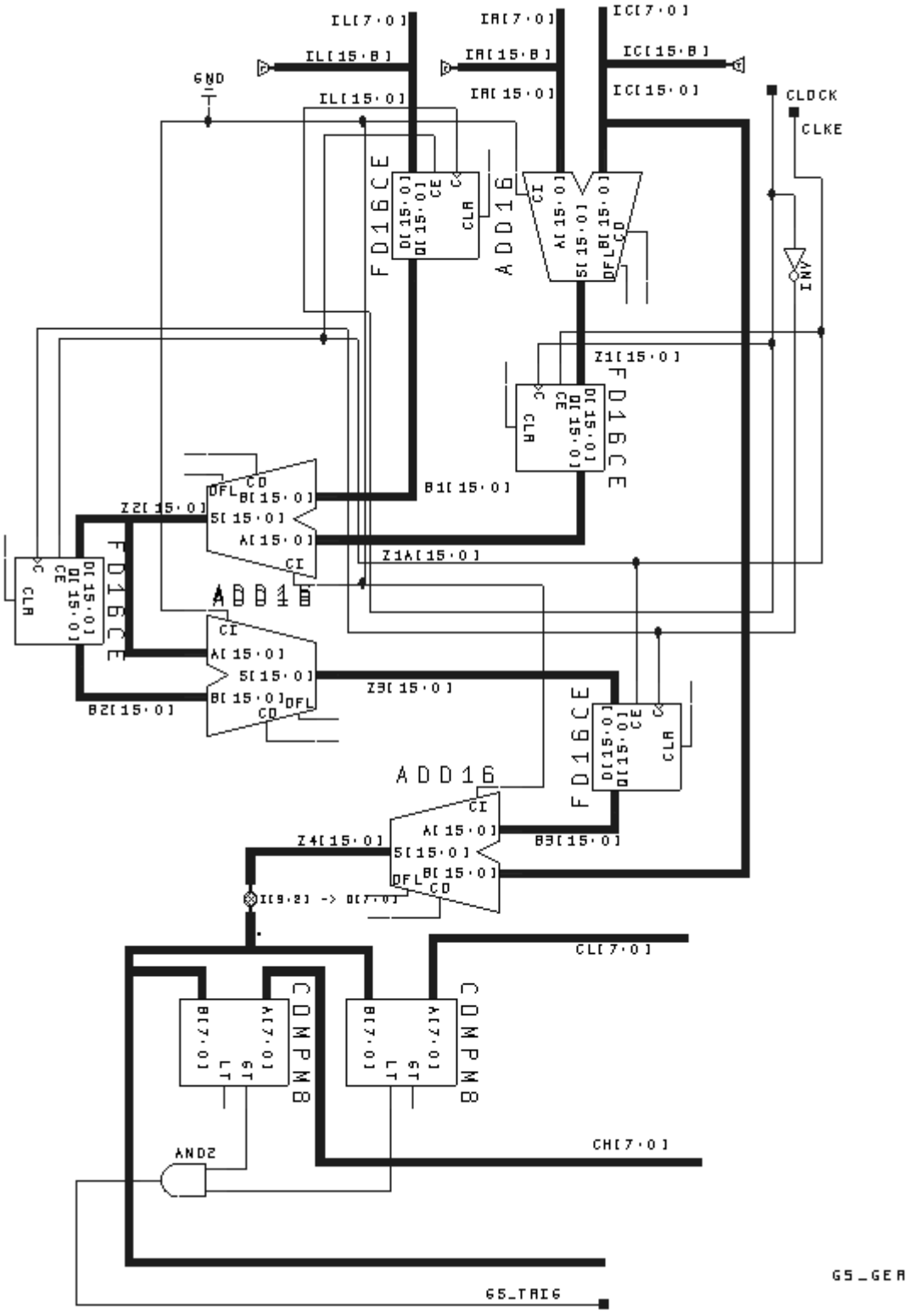


Abbildung 21 Gruppensummen-Algorithmus für Implementation in einen XILINX FPGA. Diese Darstellung zeigt das typische Format der Designeingabe mit dem Viewlogic-Paket, welches zur Entwicklung der Schauererkennungseinheit verwendet wurde.

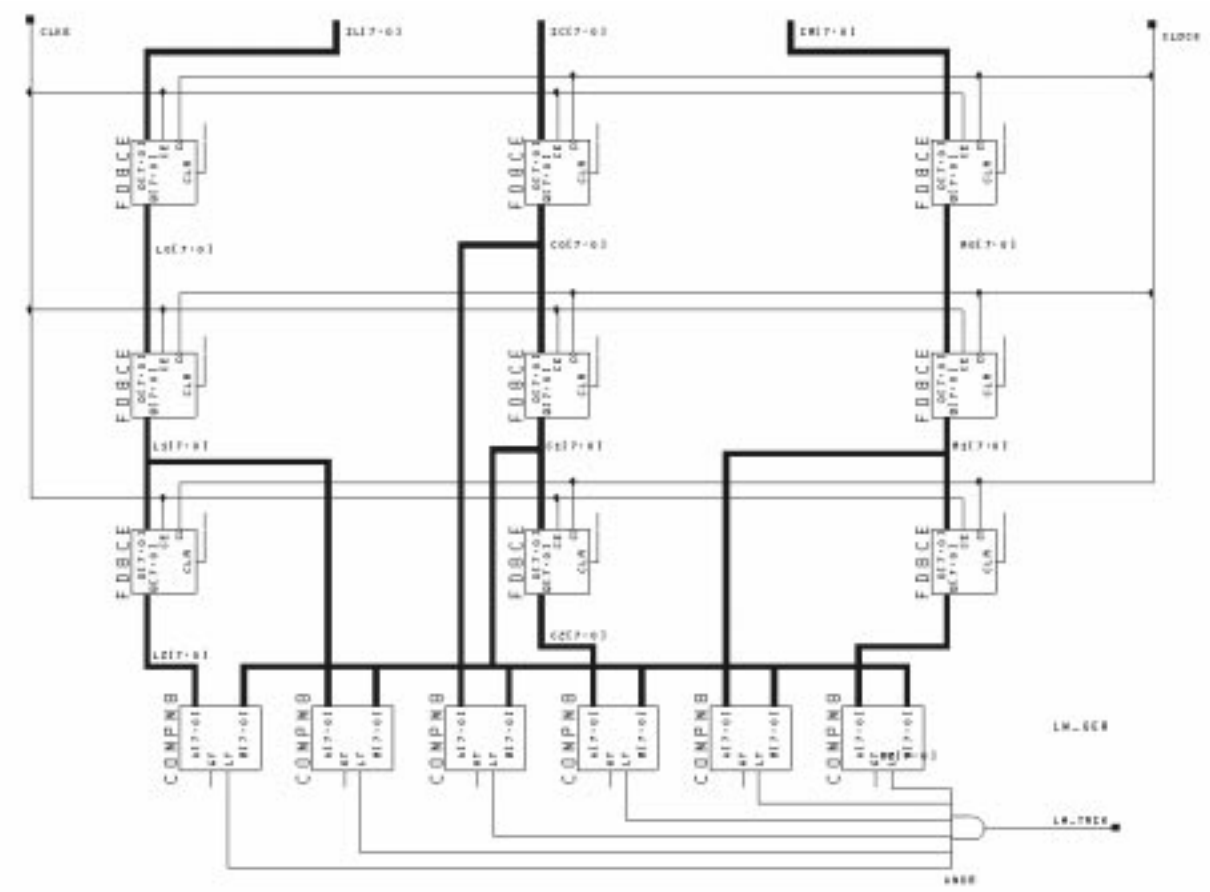


Abbildung 22 Prototyp-Algorithmus für die Lokale Maximum Suche basieren auf sechs synchronen Vergleichen

Timingsimulationen (siehe Abbildung 23 Seite 44) haben gezeigt, daß ein FPGA vom Typ 4020E-2 (Speedgrade 2) mit einem derartigen Design, welches vier Schauererkennungsprozessoren in einem FPGA verwirklicht, bei einer Frequenz von 14MHz betrieben werden kann. Höhere Frequenzen sind aufgrund der Verwendung von FIFO-Makros, welche interne Strukturen als Speicher verwenden, nicht erreichbar[33]. Die Verwendung dieser Makros erschien aus zwei Gründen ratsam. Einerseits spart die Verwendung von vier dieser FIFOs, eines pro Schauererkennungs-prozessor, IO-Resourcen am FPGA, und andererseits werden außerhalb der FPGAs keine zusätzlichen Puffer benötigt, welche ein gleichzeitiges Schreiben in ein und dasselbe Ausgangsfifo verhindern.

Auf dem Prototypen-Board wird jede Schauererkennungseinheit mit der für das Konzept angestrebten Frequenz von 10MHz betrieben.

[33] XILINX – Readme.txt zu den FIFO-Makros für FPGAs der 4000E Familie

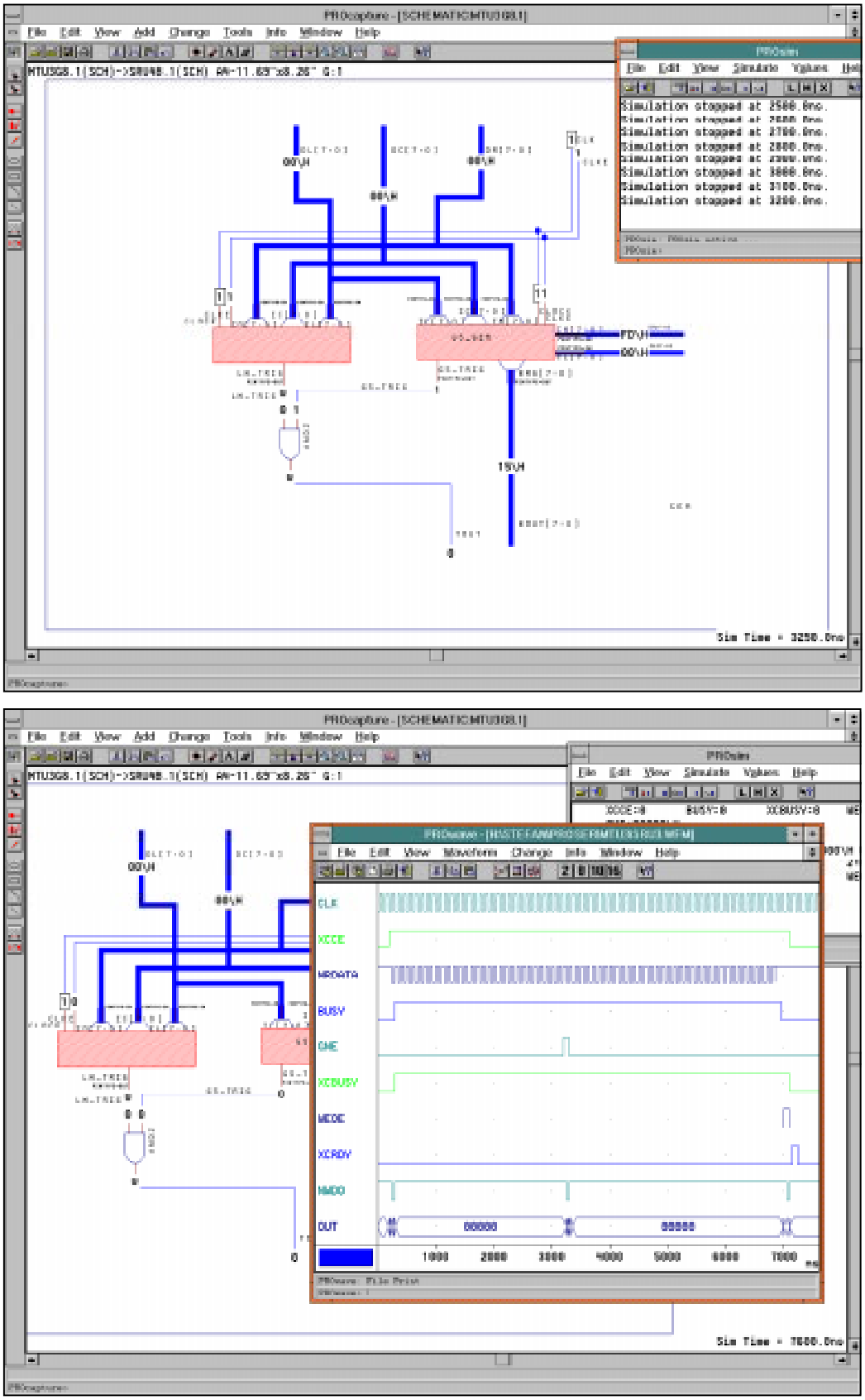


Abbildung 23 Screenshots der ViewLogic-Entwicklungsumgebung während einer Simulation zum TSRU-Projekt. Wie schön erkennen ist, ermöglicht diese Entwicklungsumgebung das zeitschrittweise Anzeigen aller Signalzustände innerhalb eines Designs. Diese Funktion erlaubt es, schon vor der eigentlichen Realisierung eines Projektes genaue Aussagen über die inneren Abläufe im jeweiligen FPGA zu treffen.

3.4 Die Prototyp-Platine

Abbildung 24 zeigt das Prototypen-Board, wie es in den Tests, welche im folgenden Kapitel kurz geschildert werden sollen, eingesetzt wurden. Beim Layout der Platine wurde eine symetrische Form gewählt, da so am besten geometrischen Bedingungen der einzelnen Stufen Rechnung getragen werden konnte. Während die Eingangsseite mit den vier Demultiplexern und dem sehr breiten Matrixspeicher (16 FIFOs) sehr viel Platz benötigt, ist ausgangsseitig deutlich weniger Platz von Nöten. Es ergibt sich also streng genommen das Bild eines umgedrehten Pyramindenstumpfes, welches sich am besten durch eine Aufteilung in der Mitte und eine Drehung jeweils um 90° plazieren läßt (siehe Abbildung 25).

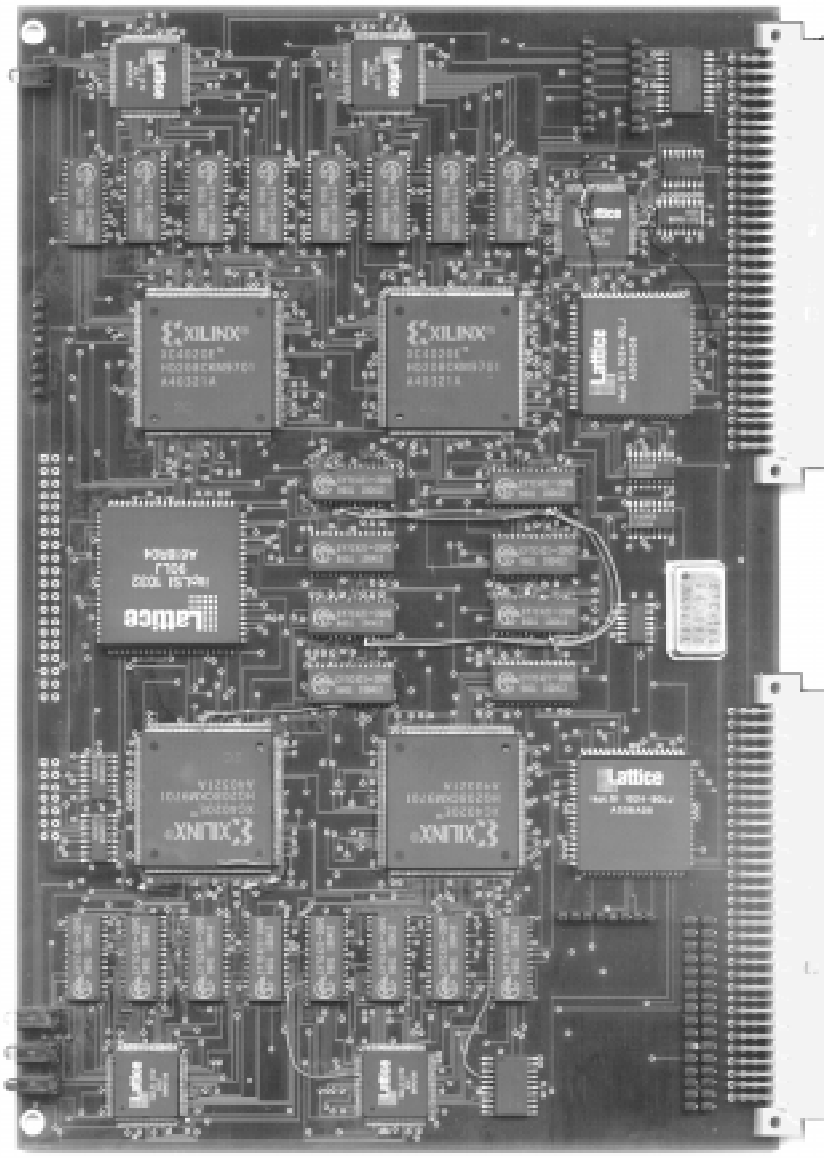


Abbildung 24 Photo des TSRU-Prototypen. Deutlich sind am oberen und unteren Rand die vier Demultiplexer zuerkennen, auf die jeweils 8 Inputfifos folgen. Im Zentrum des Boards befinden sich die 4 FPGAs und die zugehörigen Ausgabespeicher. Außerdem sind oben rechts die beiden CPLDs des VME-Interfaces zu sehen. Der Multiplexer sitzt zentral direkt hinter den IO-Steckern (nicht bestückt).

(Photo vom 10.07.197)

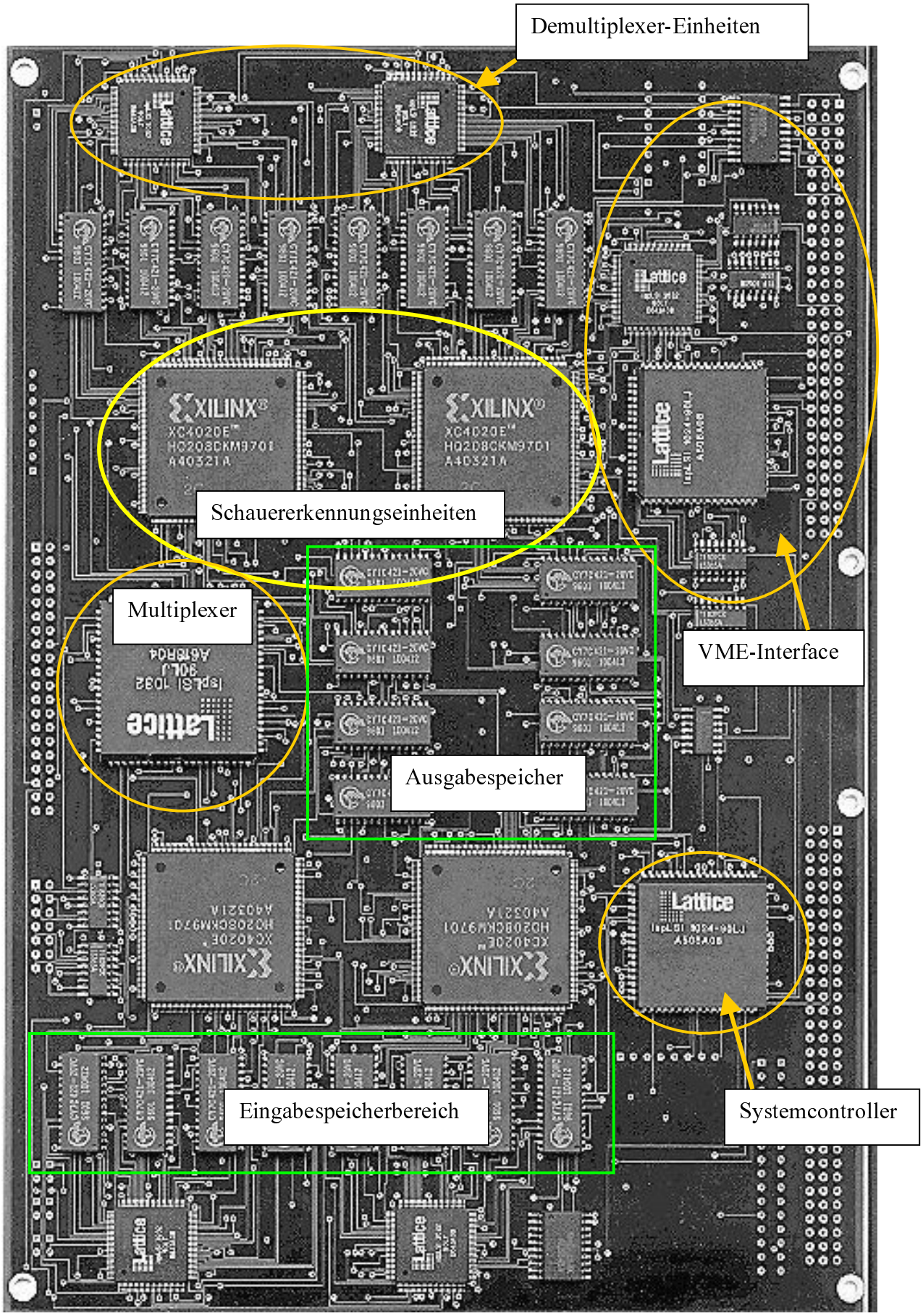


Abbildung 25 zeigt die einzelnen Funktionsgruppen des TSRU-Prototypen. Diese Aufnahme entstand direkt nach dem Löten im Reflow-Ofen.

4 Test und Ergebnisse

4.1 Bau des Prototypen

Für den TSRU-Prototypen wurde eine Multilayer-Platine mit vier Signal- und zwei Power-Lagen entworfen, welche außer Haus von der Firma ILFA in Hannover gefertigt wurde. Bestückt und gelötet wurde der Prototyp allerdings im institutseigenen SMD-Labor. Dort stehen neben einem automatischen Dispenser[34], einem Bestückungshalbautomat und einem Reflow-Lötofen[35] auch verschiedene Vorrichtungen zur Löt- und Logikkontrolle zur Verfügung. So ist es z.B. mit einem 3D-Mikroskop der Firma Vision möglich, die verlötete Platine optisch auf sogenannte „kalte Lötstellen“ oder Kurzschlüsse hin zu überprüfen. Auch ist ein komplettes SMD-Reparatur-Kit vorhanden. Nur mit dieser Infrastruktur ist es möglich, effizient derartig komplexe Elektronikentwicklung anzugehen.

4.2 Test des Prototypen

Durch den grundlegenden Aufbau des Prototypen aus programmierbaren Logikbausteinen war es im Test möglich, die TSRU schrittweise, d.h. funktionsgruppenweise, in Betrieb zu nehmen und auf seine Funktionstüchtigkeit hin zu untersuchen.

Alle Tests wurden mit Hilfe eines Crate-Controllers (ELTEC E7) in der Betriebssystem-umgebung OS9 durchgeführt (siehe Abbildung 26). Außer den boardeigenen Kontroll-mechanismen (Kontroll-LEDs) wurden für die Tests eine Logikanalysestation der Firma HP und ein LogicScope der Firma TEKTRONIX eingesetzt.

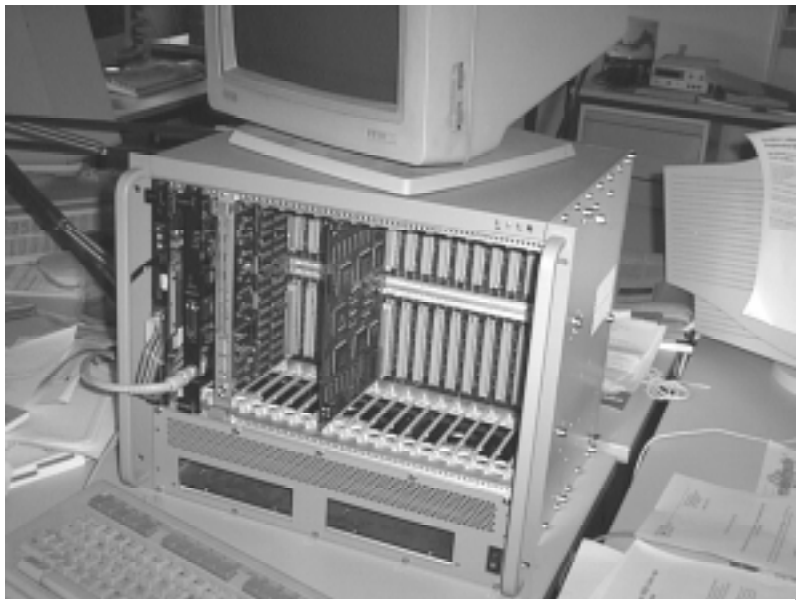


Abbildung 26 Testaufbau, wie er beim Prototypentest zum Einsatz kam.

[34] Unter „Dispensing“ versteht man das Aufbringen von Lötpaste auf den für SMD-Bauteile vorgesehenen Pads.

[35] Bei der Reflow-Löttechnik wird die bestückte Platine durch einen Mehrzonenofen gefahren, wo sie schrittweise erhitzt und abgekühlt wird.

Der Test des Prototypen gliedert sich nun in mehrere Stufen, in denen zunächst schrittweise ein immer größerer Teil des TSRU-Prototypen in Betrieb genommen wurde.

4.2.1 Inbetriebnahme des Prototypen

Zu Beginn dieser Stufe des Testprogramms wurde verifiziert, daß sich alle CPLD-Bausteine konfigurieren lassen. Da die Download-Software für die LATTICE-CPLDs zu Beginn jedes Konfigurationszyklus einen ‚Handshake' mit den zu konfigurierenden Bausteinen vollzieht, ist dieses Vorgehen ein zuverlässiger Test, ob alle CPLD-Bausteine den Lötvorgang überstanden haben.

Nachdem alle Bausteine diesen Test mit Erfolg bestanden hatten, wurde in einem nächsten Schritt mit der Programmierung des VME-Interfaces begonnen. Dabei wurde der Adress-Decoder zunächst auf solche Adressen beschränkt, die für eine Konfiguration der XILINX-FPGAs notwendig sind. Der VME-Kontroller beinhaltete in diesem Stadium der Inbetriebnahme lediglich die für die VME-Zugriffsteuerung und für FPGA-Konfiguration notwendigen Routinen. Am Abschluß dieses Schrittes mußten nun die vier FPGAs zeigen, ob auch sie den Lötprozeß gut überstanden hatten. Dabei erwies sich die Konfiguration dieser Bausteine in einer daisy-chain als nicht ganz trivial. So war ein spezielles Merging-Verfahren[36] notwendig, um eine korrekte Konfiguration mit der ebenfalls im Anhang abgedruckten Konfigurationsroutine zu ermöglichen. Wie Abschnitt 3.4 bereits beschrieben, müssen dabei die Datenbereiche der vier Programme derart aneinandergefügt, daß hinterher ein neuer Programmheader generiert wird.

Im anschließenden Schritt wurden nun die Datenpfade beginnend von Ausgangsseite her auf ihre Funktionstüchtigkeit überprüft. Dazu wurde ein Multiplexer-Programm entworfen, welches eine getrennte Auslese der Ausgabefifos gestattete. Die FPGAs wurden so konfiguriert, daß sie die Werte zweier implementierter 8bit-Counter in diese Speicher schreiben würden. So konnte sowohl die Funktionstüchtigkeit des Multiplexer-Konzeptes, welches auf einem Polling-Prinzip[37] beruht, überprüft, als auch das Verhalten der entsprechenden Signale nachgemessen werden.

Im zweiten Teilschritt dieser Datenpfadüberprüfung wurde schließlich die Dateneingangsmultiplexer so konfiguriert, daß über definierte VME-Adressen gezielt Datenworte in das entsprechende Eingangsfifo geschrieben werden konnten. Die FPGAs

---

[36] Siehe Programmquelletext „merge_rbt.c" im Anhang

[37] Der Multiplexer beginnt mit der Auslese der ersten Ausgabefifogruppe, bis er auf ein spezielles Steuerwort stößt, worauf er die zweite Gruppe adressiert, und so fort.

dienten in diesem Schritt nur als durchschreibende Instanz, als Verbindungsglied zwischen Eingangs- und Ausgangsspeicherbereich.

Im abschließenden Schritt dieses Teils des Prototypentests wurde schließlich der gesamte TSRU-Prototyp so konfiguriert, daß er prinzipiell die für einen Datentest notwendige Konfiguration umfaßte. Es sollte also nun möglich sein, mit den in Kapitel 3 beschriebenen Programmkonzepten, einen Test mit simulierten Schauerdaten durchzuführen. Ziel dieses Tests war es im wesentlichen, das Timingverhalten der verschiedenen Funktionsgruppen für den eigentlichen Test mit echten TAPS-Daten einzurichten und zu verifizieren.

Diese Tests haben nun das erste Mal gezeigt, daß das in Kapitel 2 und 3 entwickelte Konzept eines schnellen Schauererkennungsprozessors für das TAPS-Spektrometer tatsächlich den Erwartungen entspricht.

4.2.2 Test des Prototypen mit Meßdaten

Nachdem der Prototyp im letzten Schritt der Inbetriebnahme gezeigt hatte, daß sein Konzept die geforderten Kriterien (siehe Abschnitte 2.1 & 2.3) erfüllen kann, sollte nun ein intensiver Test mit Meßdaten eines TAPS-Experimentes, welches am MAMI in Mainz stattgefunden hat, die endgültige Verifikation des Konzeptes und seiner Umsetzung liefern. Wie Abbildung 27 zeigt, ist das TAPS-Spektrometer bei diesen Experimenten in einer ringförmigen Geometrie aus 6 Einzelblöcken und einer Vorwärtswand zum Einsatz gekommen, wobei besonders zu beachten ist, das der 6.Block (Block F) aus alten, kürzeren Kristallen aufgebaut war, so daß seine Auflösung deutlich unter der der anderen Blöcke gelegen hat. Die Fullrange betrug bei diesen Experimenten ca. 820 MeV.

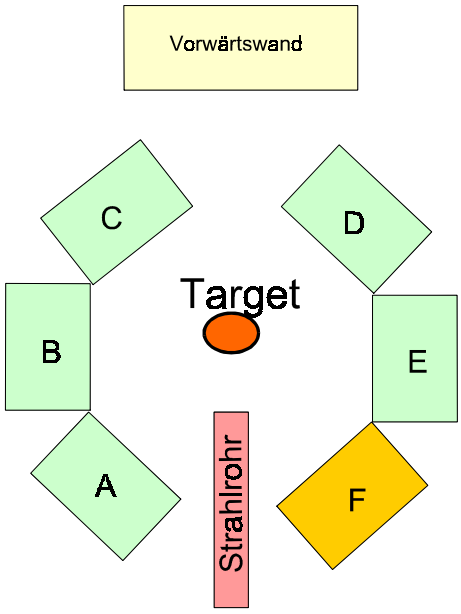


Abbildung 27 Schematische Darstellung der Mainzer TAPS-Experimente

Für den die geplanten Testläufe des TSRU-Prototypen stand ein Datensatz[38] mit ca. 45000 Einzelevents aus den 6 Blöcken eines MAMI-Experimentes bereit, wobei der Energieinhalt pro Detektormodul in Einheit von 100KeV notiert war. Mit diesem Datensatz war es nun möglich, verschiedene Testläufe der TSRU unter Wahl verschiedener Parameter durchzuführen. Als Parameter bietet sich zum einen die Energiesummenschwelle in den

[38] Auszüge aus den Meßdaten zur photoninduzierten η-Produktion am ^{4}He zwischen 560 und 820 MeV

Gruppensummenalgorithmen und zum anderen die Variation der Auflösung an. Wenn man berücksichtigt, daß das Konzept der TSRU auf 8bit-breite Datenworte angelegt ist, wird klar, daß zur Durchführung der Tests eine Normierung der Energieinhalte auf 8bit-Werte notwendig wird. Außerdem kann abhängig von der Wahl dieser Normierung zusätzlich die Reduktion der Summen mit einer Maximalbreite von 11bit auf 8bit-Ergebnisse variiert werden (z.B. A[9:2] → ES[7:0] o. A[11:4] → ES[7:0]).
Diese Variation kann allerdings nur durch direkte Änderung der FPGA-Programme erfolgen.

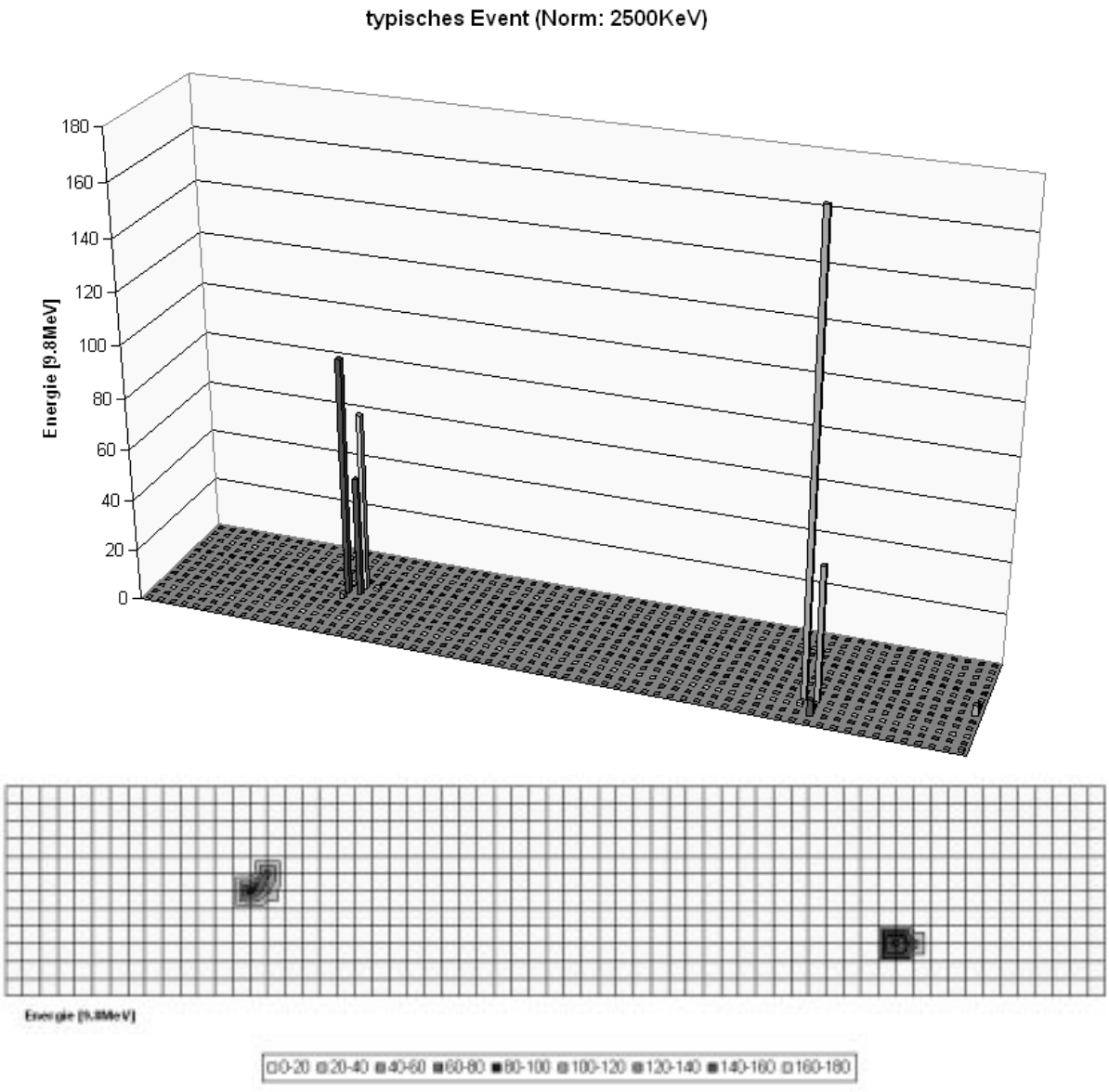


Abbildung 28 Ein typisches TAPS-Event bei einer Normierung auf 2500KeV als Eintrag in eine Eventdatenmatrix für die Verarbeitung mit dem Prototypen. In der oberen Abbildung sind deutlich zwei mögliche Schauerkandidaten zu erkennen. Die untere Grafik zeigt eine Aufsicht der Matrix (16 x 64).

Eine Analyse der Eventstruktur hat gezeigt, daß eine Beschränkung der maximalen Energiesummen auf 9bit-Werte möglich ist, daher wurden keine weiteren Tests mit anderen Reduktionsverhältnissen (z.B. 10 zu 8 oder gar 11 zu 8) durchgeführt.
Bevor nun die einzelnen Testläufe mit verschiedenen Schwellen und Normen durchgeführt wurden, ist zunächst ein Test mit einer niedrigen Schwelle und einer Maximalnorm von 6614KeV (maximaler Energieinhalt eines Moduls in diesem Datensatz) durchgeführt worden. Abbildung 29 zeigt den integrierten Schauerinhalt pro Detektormodul nach ca. 45000 Events.

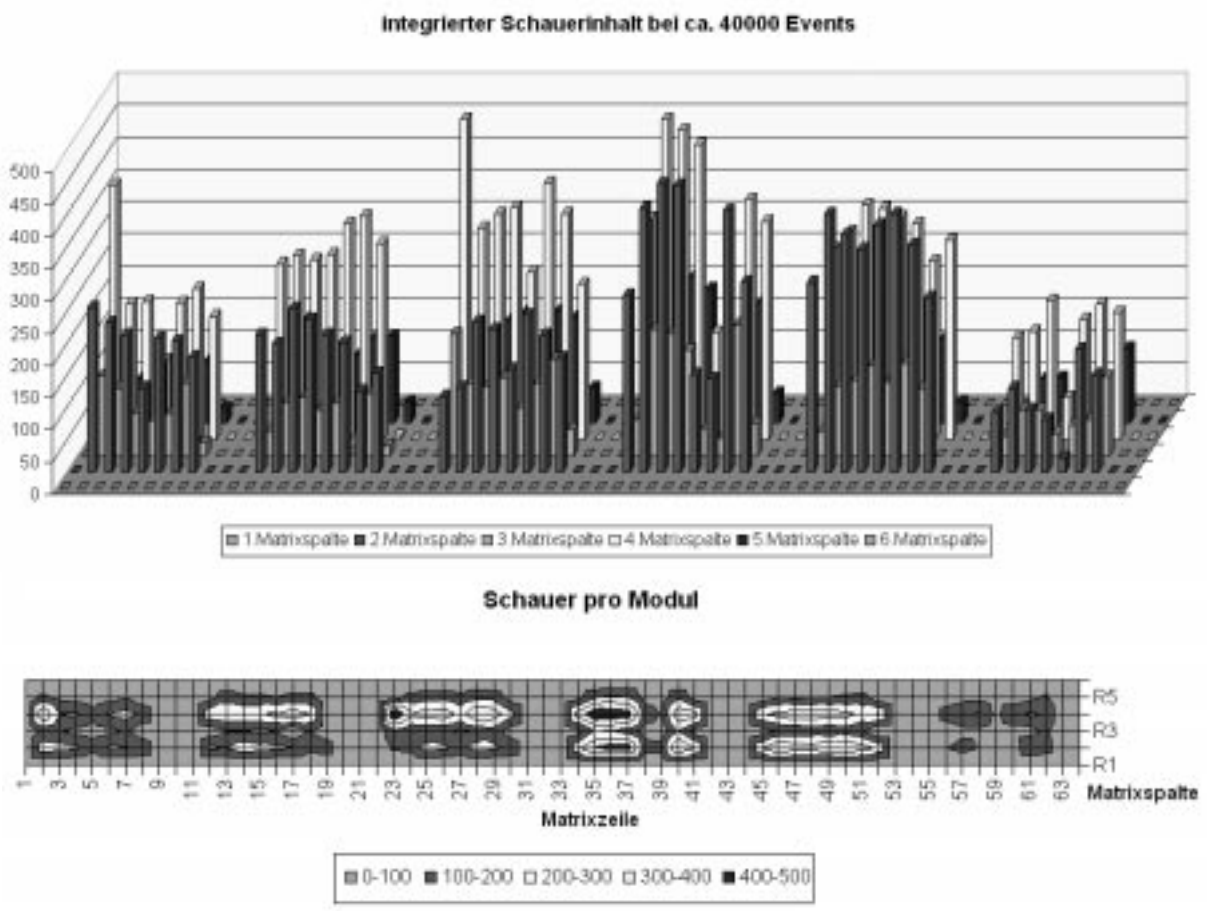


Abbildung 29 In diesen beiden Grafiken ist totale Schaueranzahl pro Detektormodul bei ca. 45000 Events aufgetragen. Dazu ist zu bemerken, daß bei diesem Testlauf eine sehr niedrige Energiesummenschwelle angewendet wurde, so daß auch sehr niederenergetische Ereignisse ihren Eingang in diese Diagramme gefunden haben.

Im wesentlichen erkennt man in diesen Diagrammen die Struktur des Mainzer TAPS-Aufbaus wieder. Dazu ist zu sagen, daß jede Säulengruppe einem TAPS-Block im Mainzer-Aufbau entspricht, so daß, beginnend mit Block A alle sechs Blöcke zu sehen sind. Deutlich ist zu erkennen, daß die Blöcke C und D die meisten Schauerevents beinhalten, was dadurch zu erklären ist, daß diese in Vorwärtsrichtung montiert waren. Außerdem ist deutlich zu sehen, daß der Block F eine schlechtere Auflösung hatte.

In den sich nun anschließenden Testläufen wurde eingehend die Abhängigkeit der Schauererkennungseigenschaften des TSRU-Prototypen von den beiden variablen Parametern untersucht. Dabei hat sich gezeigt, daß beide Parameter großen Einfluß auf die Effizienz der Schauererkennung haben. Auch ist erwartungsgemäß eine direkte Abhängigkeit der beiden Parameter voneinander deutlich nachzuweisen (siehe Energiespektren in Abbildung 30).

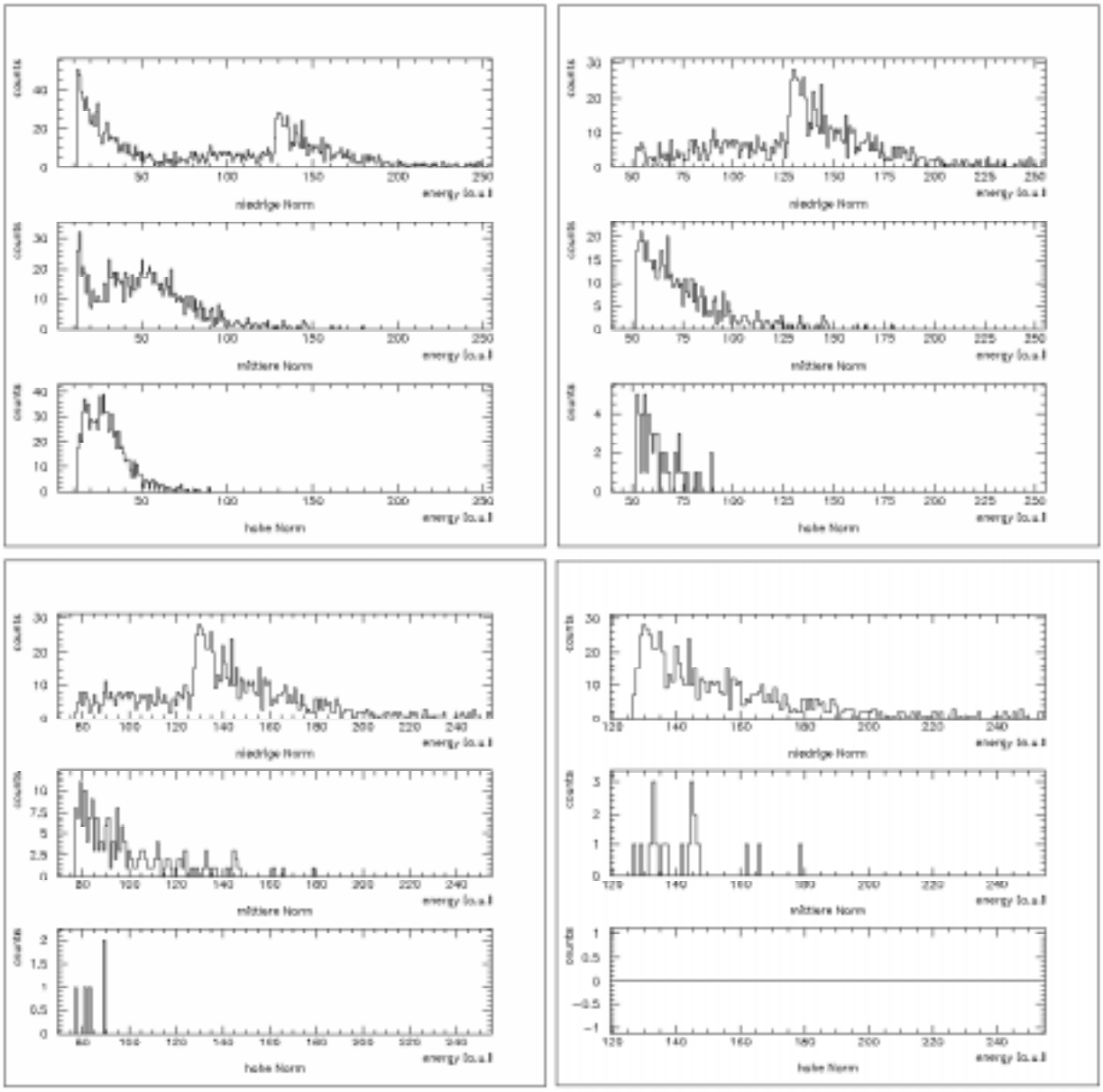


Abbildung 30 Energiespektren der registrierten Schauer, wie sie während des Testläufe vom TSRU-Prototypen aufgenommen wurden. Zu sehen ist sowohl die Abhängigkeit von der eingestellten Summenschwelle (10, 50, 75 & 125) als auch von den verwendeten Normen (1000, 2500 & 6000 KeV).

Diese Tests haben unter anderem gezeigt, daß für eine gute Effizienz sowohl Schwellen als auch Normen im mittleren Wertebereich sinnvoll sind. Außerdem stellen diese Testergebnisse einen guten Beweis für die Anwendbarkeit unseres Konzeptes und seiner Umsetzung im

Schauererkennungsprototypen dar. In Abbildung 31 schließlich sind einige rekonstruierte Schauer bei unterschiedlicher Wahl der Parameter zu sehen.

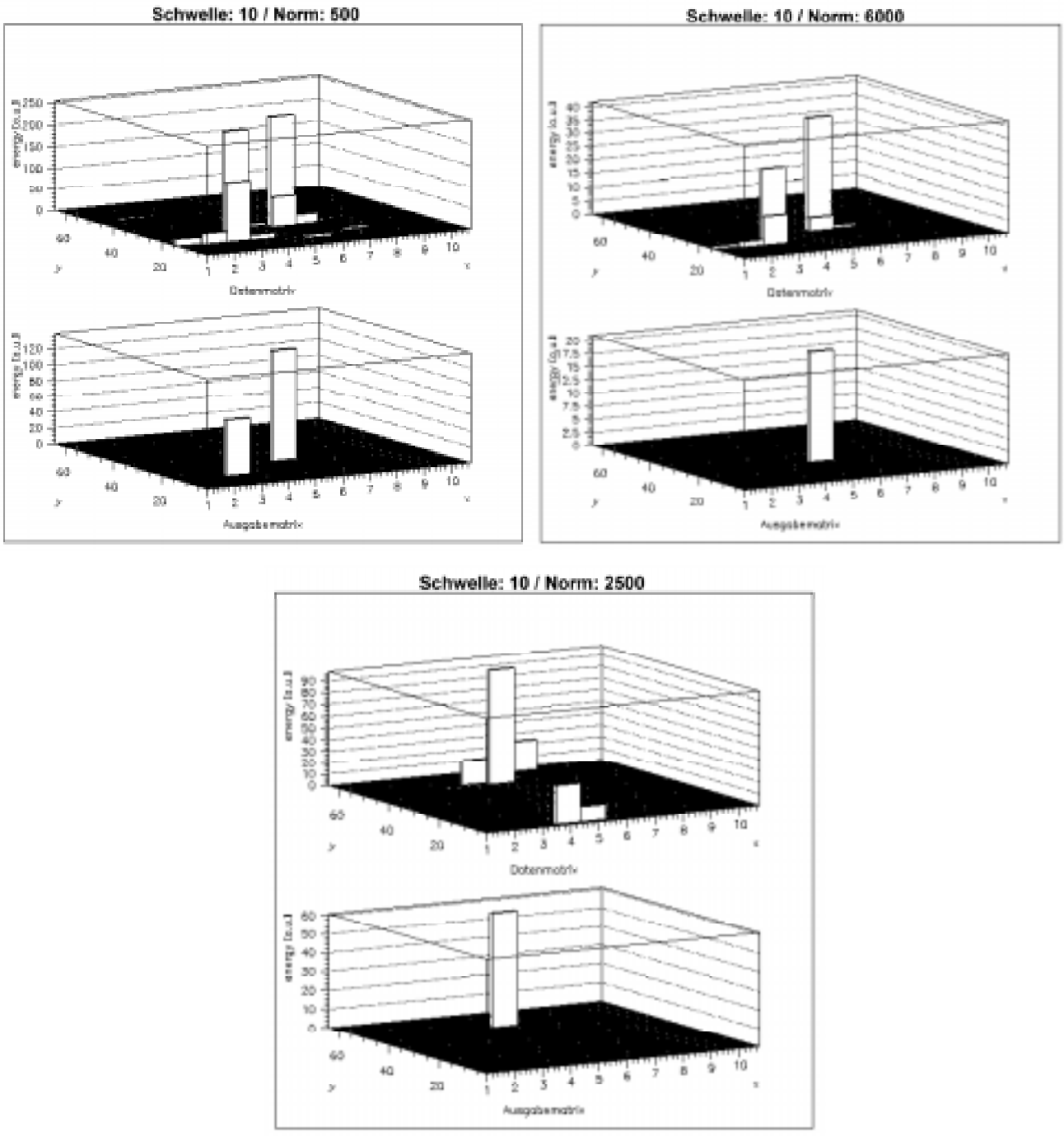


Abbildung 31 Die beiden ersten Grafikblöcke zeigen dasselbe Event bei Verwendung unterschiedlicher Normen. Es ist deutlich zu erkennen, daß bei Wahl einer höheren Normierung die Energiesumme der vorderen Einträge die gewählte Summenschwelle nicht mehr überschreitet. Die untere Grafik zeigt ein weiteres Event, bei dem erwartungsgemäß nur ein Schauer erkannt wurde.

4.3 Bewertung und Ausblick

Wie die durchgeführten Tests gezeigt haben, erfüllt das in Abschnitt 2.4 erarbeitete Konzept die an einen schnellen Schauererkennungsprozessor gestellten Anforderungen. So erbringt die auf dem Prototypen verwirklichte Schauererkennungseinheit bei Normalbetrieb eine äquivalente Rechenleistung von ca. 14.000 MIPS. Es hat sich ferner gezeigt, daß für einen „invarianten Massen Trigger“ eines Standard-TAPS-Aufbaus mit maximal 8 Blöcken eine einzige Schauererkennungseinheit vom oben beschriebenen Typ ausreichen sollte. Lediglich bei Aufbauten, wie sie im Mainzer-Experiment realisiert wurden, sind aufgrund der Geometrie zwei TSRU-Einheiten notwendig, da ansonsten die Datenmatrix verlängert werden müßte, was direkt zu einer längeren Verarbeitungszeit pro Event führen würde. Um eine Eventrate von 100KHz noch verarbeiten zu können, dürfen die Datenmatrizen nicht größer als 16 x 94 werden. Bei dieser Größe ist eine Verarbeitungszeit von 10µs pro Event mit dem vorliegenden Prototypen gerade noch gewährleistet. Steigert man die Taktfrequenz der TSRU-Einheit auf die nach der Timingsimulation möglichen 14MHz, so steigt die maximale Matrixgröße auf 16 x 134, was maximal 2144 Detektormodulen in einer einfachen Wand-Konfiguration entspräche. Die aktuelle Ausbaustufe des TAPS-Spektrometers umfaßt zur Zeit nur 384 Module, in nächster Zeit angestrebt ist ein Ausbau bis maximal 512 Detektormodulen, was 8 Blöcken entspräche.

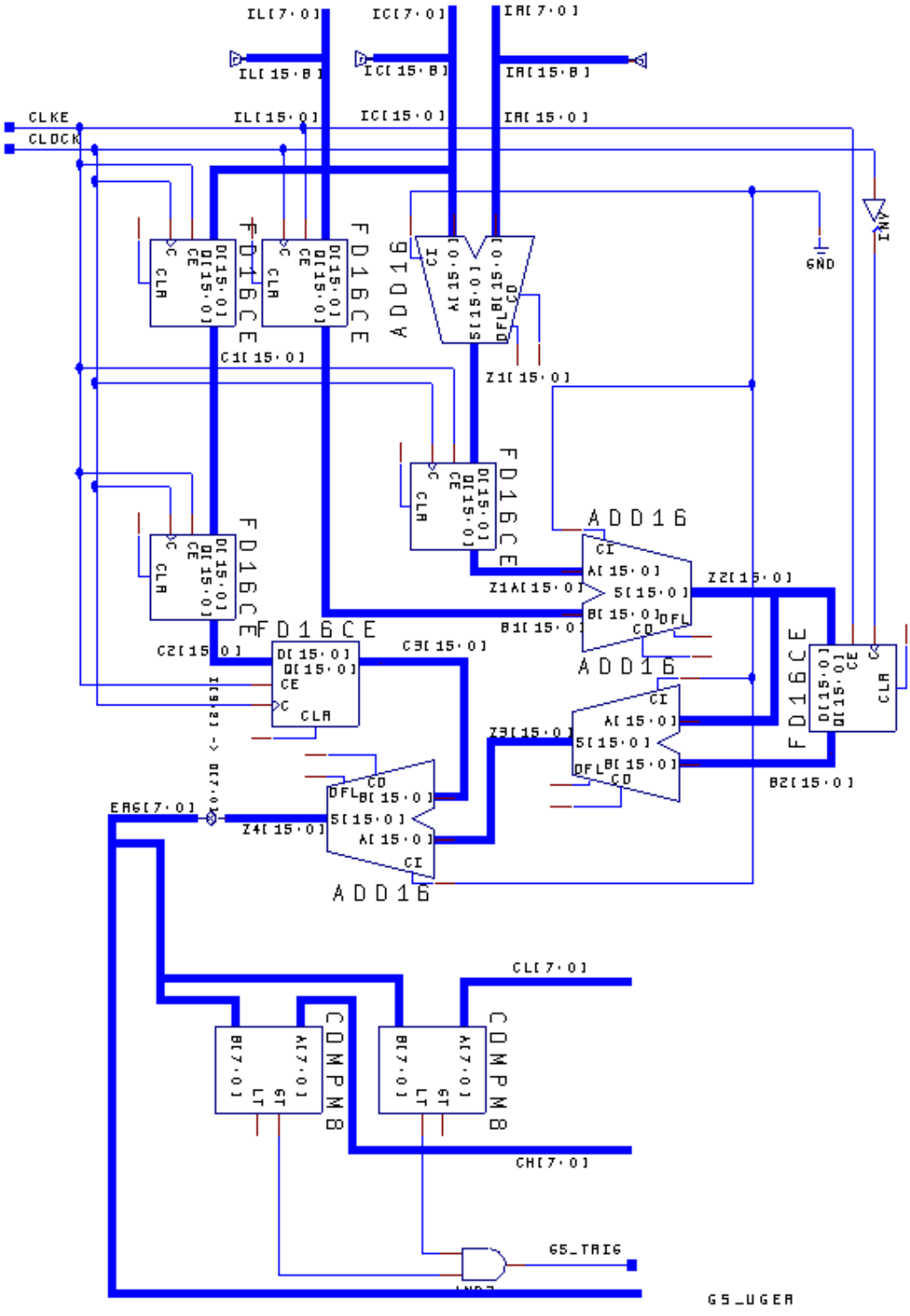
Die Tests mit niedrigen Schwellen und Normen haben zu dem eine weitergehende Verbesserungsmöglichkeit des TSRU-Prototypen in einer revidierten Ausgabe aufgezeigt. So kommt es gerade bei Wahl niedriger Normierungen und Summenschwellen zu Mehrfacherkennung von Schauern. Dieses Problem tritt vor allem in den Überlapp-Regionen der einzelnen FPGAs auf, so daß hier beim Bau einer zweiten TSRU-Einheit eine größere Vetoregion eingeführt werden sollte, was sich durch zusätzliche Datenverbindungen zwischen den benachbarten FPGAs erreichen läßt.

Schließlich bedarf auch die Steuerung der Schwellen für die Energiesummen einer Nachbesserung, da sich die Lösung der direkten Speicherung in den FPGAs als unzureichend erwiesen hat. An dieser Stelle wird die endgültige TSRU-Einheit um zwei Latches ergänzt, welche dann direkt über das VME-Interface geladen werden können.

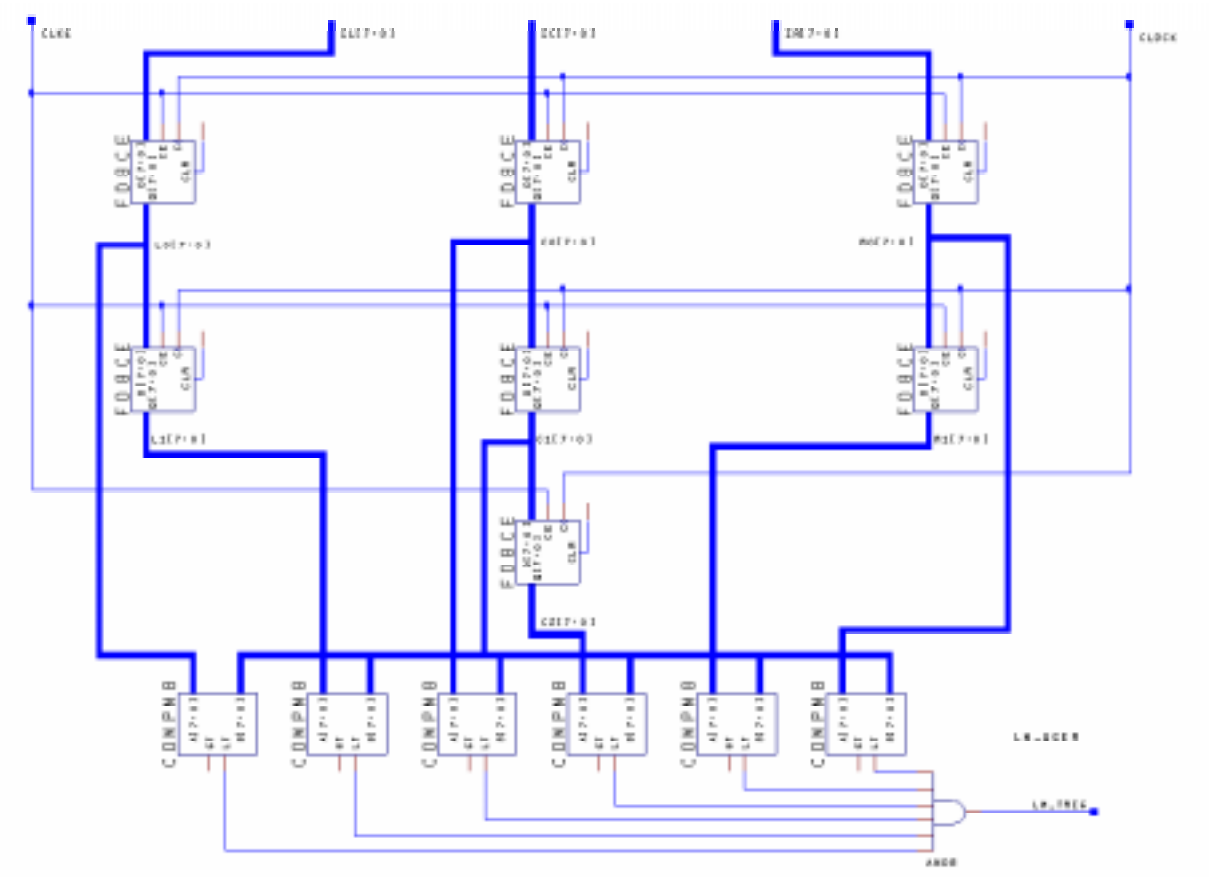
5 Anhang

5.1 FPGA-Makros der Algorithmen für ungerade Spaltennummern

5.1.1 Schaltung zur Bestimmung der Gruppensumme



5.1.2 Makro zur Bestimmung des lokalen Maximums



Wie deutlich zu erkennen ist, ist die Struktur der beiden Algorithmen für ungerade Spaltennummern denen für gerade Spalten nahezu identisch. Lediglich in der Behandlung des 7. Elementes einer Gruppe ist ein Unterschied festzustellen.

5.2 Einige ABEL-Sourcen

5.2.1 ABEL-Programm für den Adreß-Decoder

```
module      TADRDEC

declarations
"Pin-Zuweisungen
      A01,A02,A03,A04,A05     pin 61,60,59,58,57;
      A06,A07,A08,A09,A10     pin 56,48,47,46,45;
      A11,A12,A13,A14,A15     pin 44,43,42,41,40;
      A16,A17,A18,A19,A20     pin 39,38,37,33,32;
      A21,A22,A23             pin 31,30,29;

      REG_ADR01..REG_ADR23        node istype 'reg_d,buffer';

      AS                      pin 15;
      ASN                     node;

      BDSEL                   pin 14;
      TSRU_SEL                node;

      OA0..OA9                node;

      VCON0,VCON1                   pin 67,3;
      VCON2,VCON3,VCON4,VCON5       pin 4,5,6,7;

      BA4,BA5                       pin 8,9;
      DMSEL,SRUSEL,MSEL,SCSEL       pin 10,11,12,13;

"Sets
      ADR               = [A23..A01];
      REG_ADR           = [REG_ADR23..REG_ADR01];
      HI_ADR            = [REG_ADR23..REG_ADR08];
      LO_ADR            = [REG_ADR07..REG_ADR01];
      OUT_ADR           = [OA9..OA0];
      VME_CON           = [SCSEL,MSEL,SRUSEL,DMSEL,
                           VCON5,VCON4,VCON3,VCON2,VCON1,VCON0];

      BOARD_ADRESSE     = ^h8000; "Bit 23:8

      RESET_IFF_ADR     = ^B0000001;
      RESET_OFF_ADR     = ^B0000010;
      RESET_DEMUX_ADR   = ^B0000011;
      RESET_MUX_ADR     = ^B0000100;
      RESET_XC_ADR      = ^B0000101;
      RESET_SYSC_ADR    = ^B0000110;
      RESET_CLK_ADR     = ^B0000111;

      XILINX_ADRESSE          = ^B0001000;
      XILINX_STATUS_ADRESSE   = ^B0001001;
      LOAD_LOWT               = ^B0001010;
      LOAD_HIGHT              = ^B0001011;

      WRITE_IFF1  = ^B0010000;
      WRITE_IFF2  = ^B0010001;
      WRITE_IFF3  = ^B0010010;
      WRITE_IFF4  = ^B0010011;
      WRITE_IFF5  = ^B0010100;
```

```
      WRITE_IFF6  = ^B0010101;
      WRITE_IFF7  = ^B0010110;
      WRITE_IFF8  = ^B0010111;
      WRITE_IFF9  = ^B0011000;
      WRITE_IFF10 = ^B0011001;
      WRITE_IFF11 = ^B0011010;
      WRITE_IFF12 = ^B0011011;
      WRITE_IFF13 = ^B0011100;
      WRITE_IFF14 = ^B0011101;
      WRITE_IFF15 = ^B0011110;
      WRITE_IFF16 = ^B0011111;

      READ_OFF1   = ^B0100000;
      READ_OFF2   = ^B0100001;
      READ_OFF3   = ^B0100010;
      READ_OFF4   = ^B0100011;

      START_SRU   = ^B0101000;
      STOP_SRU    = ^B0101001;

      BOARD_STATUS_ADRESSE    = ^B0110000;

      TESTMODE_ADRESSE        = ^B1000000;

equations
      REG_ADR.CLK = !AS;
      REG_ADR.D   = ADR;

      !BDSEL            = ([A23..A08] == BOARD_ADRESSE) & !AS;

      TSRU_SEL    = (HI_ADR == BOARD_ADRESSE);

      WHEN (TSRU_SEL & (LO_ADR == XILINX_ADRESSE))
            THEN OUT_ADR = ^B0000000001;
      ELSE WHEN (TSRU_SEL & (LO_ADR == LOAD_LOWT))
            THEN OUT_ADR = ^B0010000111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == LOAD_HIGHT))
            THEN OUT_ADR = ^B0010001011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == RESET_IFF_ADR))
            THEN OUT_ADR = ^B0000000100;
      ELSE WHEN (TSRU_SEL & (LO_ADR == RESET_OFF_ADR))
            THEN OUT_ADR = ^B0000001000;
      ELSE WHEN (TSRU_SEL & (LO_ADR == RESET_DEMUX_ADR))
            THEN OUT_ADR = ^B0000001100;
      ELSE WHEN (TSRU_SEL & (LO_ADR == RESET_MUX_ADR))
            THEN OUT_ADR = ^B0000010000;
      ELSE WHEN (TSRU_SEL & (VME_CON == RESET_SYSC_ADR))
            THEN OUT_ADR = ^B0000010100;
      ELSE WHEN (TSRU_SEL & (LO_ADR == RESET_XC_ADR))
            THEN OUT_ADR = ^B0000011000;
      ELSE WHEN (TSRU_SEL & (LO_ADR == RESET_CLK_ADR))
            THEN OUT_ADR = ^B0000011100;
      ELSE WHEN (TSRU_SEL & (LO_ADR == XILINX_STATUS_ADRESSE))
            THEN OUT_ADR = ^B0000000110;
      ELSE WHEN (TSRU_SEL & (LO_ADR == BOARD_STATUS_ADRESSE))
            THEN OUT_ADR = ^B0000001010;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF1))
            THEN OUT_ADR = ^B0001000011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF2))
            THEN OUT_ADR = ^B0001000111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF3))
```

```
            THEN OUT_ADR = ^B0001001011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF4))
            THEN OUT_ADR = ^B0001001111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF5))
            THEN OUT_ADR = ^B0001010011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF6))
            THEN OUT_ADR = ^B0001010111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF7))
            THEN OUT_ADR = ^B0001011011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF8))
            THEN OUT_ADR = ^B0001011111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF9))
            THEN OUT_ADR = ^B0001100011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF10))
            THEN OUT_ADR = ^B0001100111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF11))
            THEN OUT_ADR = ^B0001101011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF12))
            THEN OUT_ADR = ^B0001101111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF13))
            THEN OUT_ADR = ^B0001110011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF14))
            THEN OUT_ADR = ^B0001110111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF15))
            THEN OUT_ADR = ^B0001111011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == WRITE_IFF16))
            THEN OUT_ADR = ^B0001111111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == READ_OFF1))
            THEN OUT_ADR = ^B0100000011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == READ_OFF2))
            THEN OUT_ADR = ^B0100000111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == READ_OFF3))
            THEN OUT_ADR = ^B0100001011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == READ_OFF4))
            THEN OUT_ADR = ^B0100001111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == START_SRU))
            THEN OUT_ADR = ^B1000000111;
      ELSE WHEN (TSRU_SEL & (LO_ADR == STOP_SRU))
            THEN OUT_ADR = ^B1000001011;
      ELSE WHEN (TSRU_SEL & (LO_ADR == TESTMODE_ADRESSE))
            THEN OUT_ADR = ^B0000111110;
      ELSE OUT_ADR = ^B0000000000;

      VME_CON = OUT_ADR;

end tadrdec
```

5.2.2 VME-Kontroller ABEL-Source

Von besonderem Interesse dürfte in diesem Quelltext die bidirektionale Nutzung verschiedener Pins sein.

```
module tvmecon

declarations

      BDSEL                         pin 70;
      TSRUSEL                       node;
      DS0,DS1,DTACK,LWORD,WRITE     pin 79,78,73,72,71;
      BERR                          pin 69;
```

```
OE,WE0,WE1,CYC                  pin 57,56,55,54;
DT152,DT104,DT56,DT28           pin 10,89,87,66;

BD0,BD1,BD2,BD3,BD4,BD5         pin 98,3,4,5,6,7 istype'reg_d,buffer';
BD6,BD7                         pin 8,9 istype'reg_d,buffer';
LID0..LID7                      node;
ID0..ID7                        node istype'reg_g,buffer';
OD0..OD7                        node;

VME_CON0,VME_CON1,VME_CON2      pin 92,93,97;
VME_CON3,VME_CON4,VME_CON5      pin 96,95,94;

READDATA,WRITEDATA              node;

MRS1,MRS2,MRO                   pin 53,48,47;
SCRST,MRST,DMRST,XCRST,CLKRST pin 46,45,44,43,40;

XCINIT                          pin 41;
TESTMODE                        pin 42;
TMREG                           node istype 'reg_d,buffer';
TSRUBSY,XCCE                    pin 91,90;

SCD0,SCD1,SCD2,SCD3             pin 68,67,59,58;
RW_TIM0,RW_TIM1                 node;

HDC3,HDC2,HDC1,HDC0             pin 36,35,34,33;
HDC                             node;
DONE3,DONE2,DONE1,DONE0         pin 32,31,30,29;
DONE                            node;
CCLK                            pin 28;
DIN,RDYBSY,INIT                 pin 23,22,21;
PROG3,PROG2,PROG1,PROG0         pin 20,19,18,17;
PROG_MODE                       node;
XPRG,XDIN,XCCLK                 node istype 'reg_d,buffer';

DATA        = [BD7..BD0];
INDATA      = [ID7..ID0];
KNOT        = [LID7..LID0];
OUTDATA     = [OD7..OD0];

VME_CON     = [VME_CON5..VME_CON0];

PROG        = [PROG3..PROG0];
BRD_SRG     = [0,0,0,0,XCCE,TMREG,DONE,!INIT];

RESETS      = [!MRS1,!MRS2,!MRO,!SCRST,!MRST,!DMRST,XCRST,!CLKRST];

XILINX            = ^B000001;
XILINX_STATUS     = ^B000110;
BOARD_STATUS      = ^B001010;
TM_SWITCH         = ^B111110;

RESET_IFF   = ^B000100;
RESET_OFF   = ^B001000;
RESET_DEMUX = ^B001100;
RESET_MUX   = ^B010000;
RESET_SC    = ^B010100;
RESET_XC    = ^B011000;
RESET_CLK   = ^B011100;

SET_PROG    = ^B00000001;
```

```
      UNSET_PROG  = ^B00000010;
      SET_DIN     = ^B00000100;
      UNSET_DIN   = ^B00001000;
      SET_CCLK    = ^B00010000;
      UNSET_CCLK  = ^B00100000;
      TM_HIGH     = ^B00000001;
      TM_LOW      = ^B00000010;

PLSI PROPERTY 'STRATEGY DELAY';

equations
      "Allgemeine Steuersignale
      TSRUSEL = !BDSEL;
      DTACK = !DT152 & TSRUSEL;
      OE    = !WRITE;
      BERR  = 0;
      WE0   = !DT56;
      "****************************************************************
      "Bidirektionaler Datenbus-Treiber
      WHEN (VME_CON == BOARD_STATUS) THEN
            OUTDATA = BRD_SRG;
      ELSE
            OUTDATA = ^B00000000;

      READDATA    = (VME_CON == BOARD_STATUS);
      DATA.D            = OUTDATA;
      DATA.CLK    = READDATA & !DT56;
      DATA.OE     = READDATA;

      KNOT        = DATA.pin;

      WRITEDATA   = (VME_CON == XILINX) # (VME_CON == TM_SWITCH);

      INDATA.D    = KNOT;
      INDATA.LH   = DT56 & WRITEDATA;
      "****************************************************************
      "Xilinx-Konfiguration
      PROG_MODE   = (VME_CON == XILINX);

      XPRG.D            = 1;
      XPRG.CLK    = (INDATA == SET_PROG) & PROG_MODE;
      XPRG.RE     = (INDATA == UNSET_PROG) & PROG_MODE;
      PROG        = !XPRG;

      XDIN.D            = 1;
      XDIN.CLK    = (INDATA == SET_DIN) & PROG_MODE;
      XDIN.RE     = (INDATA == UNSET_DIN) & PROG_MODE;
      DIN         = XDIN;

      XCCLK.D     = 1;
      XCCLK.CLK   = (INDATA == SET_CCLK) & PROG_MODE;
      XCCLK.RE    = (INDATA == UNSET_CCLK) & PROG_MODE;
      CCLK        = XCCLK;

      DONE        = DONE0 # DONE1 # DONE2 # DONE3;

      HDC         = HDC0 # HDC1 # HDC2 # HDC3;
      "****************************************************************
      "Testmode
      TMREG.D     = 1;
      TMREG.CLK   = (VME_CON == TM_SWITCH) & (INDATA == TM_HIGH);
```

```
      TMREG.RE    = (VME_CON == TM_SWITCH) & (INDATA == TM_LOW);
      TESTMODE    = TMREG;
      "*****************************************************************
      "Schreib-/Lese-Timing
      RW_TIM0     = DT28;
      RW_TIM1     = DT152;
      SCD0        = RW_TIM0;
      SCD1        = RW_TIM1;
      "*****************************************************************
      "Reset-Funktionen
      WHEN (VME_CON == RESET_IFF)
            THEN RESETS = ^B11000000;
      ELSE WHEN (VME_CON == RESET_OFF)
            THEN RESETS = ^B00100000;
      ELSE WHEN (VME_CON == RESET_DEMUX)
            THEN RESETS = ^B00010000;
      ELSE WHEN (VME_CON == RESET_MUX)
            THEN RESETS = ^B00001000;
      ELSE WHEN (VME_CON == RESET_SC)
            THEN RESETS = ^B00000100;
      ELSE WHEN (VME_CON == RESET_XC)
            THEN RESETS = ^B00000010;
      ELSE WHEN (VME_CON == RESET_CLK)
            THEN RESETS = ^B00000001;
      ELSE RESETS = ^B00000000;

end tvmecon
```

5.2.3 Beispielsource für einen Eingabedemultiplexer

```
module DEMUX1

declarations

      IN0..IN7                pin 3,4,5,6,7,8,9,17;
      DGATE                   pin 66;
      BD0..BD7                pin 18,19,20,21,22,23,28,29;

      ID0..ID7                node istype 'reg_d,buffer';

      TIM_IN                        pin 98;
      W_TIM                   node;

      DA0..DA8                pin 30,31,32,33,34,35,36,40,41;
      MWA                     pin 42;
      DB0..DB8                pin 43,44,45,46,47,48,53,54,55;
      MWB                     pin 56;
      DC0..DC8                pin 57,58,59,67,68,69,70,71,72;
      MWC                     pin 73;
      DD0..DD8                pin 78,79,80,81,82,83,84,85,86;
      MWD                     pin 90;

      OD0..OD39               node;

      BA0..BA3,DMSEL          pin 91,92,87,89,10;

      DAVAL                   pin 93;
      DMRST,RST,LATRST        pin 94,95,96;
      TMODE                   pin 97;

      SWEID1..SWEID4          node istype 'reg_d,buffer';
```

```
      TDATA = [BD7..BD0];
      IDATA = [ID7..ID0];

      OUTDATA = [OD0..OD39];
      OUTSTR      = [!MWA,DA8,DA7,DA6,DA5,DA4,DA3,DA2,DA1,DA0,
                     !MWB,DB8,DB7,DB6,DB5,DB4,DB3,DB2,DB1,DB0,
                     !MWC,DC8,DC7,DC6,DC5,DC4,DC3,DC2,DC1,DC0,
                     !MWD,DD8,DD7,DD6,DD5,DD4,DD3,DD2,DD1,DD0];

      DM_ADR      = [BA3,BA2];
      FF_ADR      = [BA1,BA0];

      "Subadressen der Demultiplexereinheit
      "DEMUX1
      DEMUX_ADR   = ^B00;
      "DEMUX2
      "DEMUX_ADR  = ^B01;
      "DEMUX3
      "DEMUX_ADR  = ^B10;
      "DEMUX4
      "DEMUX_ADR  = ^B11;

      FIFO1_ADR   = ^B00;
      FIFO2_ADR   = ^B01;
      FIFO3_ADR   = ^B10;
      FIFO4_ADR   = ^B11;

equations
      "RESET des Demultiplexers
      LATRST      = DMRST; " # RST
      "Timing
      W_TIM = TIM_IN & TMODE & DMSEL;
      "*****************************************************************
      "Steuerung des Datenstromes
      "Testdaten
      IDATA.CLK   = (DM_ADR == DEMUX_ADR) & W_TIM;
      IDATA.D     = TDATA;
      IDATA.RE    = !TMODE;
      "*****************************************************************
      "EID-Marker
      SWEID1.D    = 1;
      SWEID1.CLK  = MWA.PIN;
      SWEID1.RE   = !TMODE;
      SWEID2.D    = 1;
      SWEID2.CLK  = MWB.PIN;
      SWEID2.RE   = !TMODE;
      SWEID3.D    = 1;
      SWEID3.CLK  = MWC.PIN;
      SWEID3.RE   = !TMODE;
      SWEID4.D    = 1;
      SWEID4.CLK  = MWD.PIN;
      SWEID4.RE   = !TMODE;
      "*****************************************************************
      "Datenausgabe
      WHEN ((DM_ADR == DEMUX_ADR) & (FF_ADR == FIFO1_ADR) & TMODE)
            THEN OUTDATA = [W_TIM,!SWEID1,ID7,ID6,ID5,ID4,ID3,ID2,ID1,ID0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0];
      ELSE WHEN ((DM_ADR == DEMUX_ADR) & (FF_ADR == FIFO2_ADR) & TMODE)
```

```
            THEN OUTDATA = [0,0,0,0,0,0,0,0,0,0,
                           W_TIM,!SWEID2,ID7,ID6,ID5,ID4,ID3,ID2,ID1,ID0,
                           0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0];
      ELSE WHEN ((DM_ADR == DEMUX_ADR) & (FF_ADR == FIFO3_ADR) & TMODE)
            THEN OUTDATA = [0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,
                           W_TIM,!SWEID3,ID7,ID6,ID5,ID4,ID3,ID2,ID1,ID0,
                           0,0,0,0,0,0,0,0,0,0];
      ELSE WHEN ((DM_ADR == DEMUX_ADR) & (FF_ADR == FIFO4_ADR) & TMODE)
            THEN OUTDATA = [0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,
                           W_TIM,!SWEID4,ID7,ID6,ID5,ID4,ID3,ID2,ID1,ID0];
      ELSE OUTDATA = [0,0,0,0,0,0,0,0,0,0,
                     0,0,0,0,0,0,0,0,0,0,
                     0,0,0,0,0,0,0,0,0,0,
                     0,0,0,0,0,0,0,0,0,0];
      OUTSTR       = OUTDATA;

end demux1
```

5.2.4 Source für Ausgabemultiplexer mit vier FIFOGruppen

```
module TSRUMUX

declarations
      FRB0..FRB5        pin 2,3,4,5,6,7;
      FRB6..FRB11       pin 8,9,10,11,12,13;
      FRB12..FRB17      pin 14,15,16,17,18,19;
      DW0..DW17         node istype 'reg_g,buffer';

      FRD0..FRD3        pin 29,28,27,26;
      RD0..RD3          node;
      READ              node;

      BD0..BD7          pin 37,38,39,40,41,45,46,47;
      VCON0,VCON1,MSEL  pin 36,35,34;
      OE                pin 33;

      MTIM              pin 31;

      TMODE,RECBSY,XCRDY      pin 84,67,30;
      TM                node;

      LT                pin 75;

      FG0,FG1,FG2       node istype 'reg_d,buffer';

      CTR0,CTR1         node;


      LO_IN       = [FRB7..FRB0];
      ME_IN       = [FRB15..FRB8];
      HI_IN       = [FRB17,FRB16];

      LT_LO       = [DW7..DW0];
      LT_ME       = [DW15..DW8];
      LT_HI       = [DW17,DW16];

      VODATA      = [BD7..BD0];
```

```
      VCON  = [VCON1,VCON0];

      FGC   = [FG2,FG1,FG0];

equations

      TM    = TMODE & MSEL;

      FGC.D = FGC.Q + 1;
      FGC.RE      = !TMODE;
      FGC.CE  = !(FGC == ^B100);
      FGC.CLK = ((LT_LO == ^Hfe) & (LT_ME == ^H55) & (LT_HI == ^B10));

      LT_LO.D     = LO_IN;
      LT_ME.D     = ME_IN;
      LT_HI.D     = HI_IN;
      LT_LO.LH    = READ;
      LT_ME.LH    = READ;
      LT_HI.LH    = READ;

      CTR0        = DW16;
      CTR1        = DW17;
      RD0   = TM & MTIM & (VCON == ^B00) & (FGC == ^B000);
      RD1   = TM & MTIM & (VCON == ^B00) & (FGC == ^B001);
      RD2   = TM & MTIM & (VCON == ^B00) & (FGC == ^B010);
      RD3   = TM & MTIM & (VCON == ^B00) & (FGC == ^B011);

      READ  = RD0 # RD1 # RD2 # RD3;
      FRD0  = !RD0;
      FRD1  = !RD1;
      FRD2  = !RD2;
      FRD3  = !RD3;

      VODATA.OE   = TM & MTIM & !(VCON == ^B00);

      WHEN (TM & (VCON == ^B01))
            THEN VODATA = [0,0,0,0,CTR1,CTR0,FG1,FG0];
      ELSE WHEN (TM & (VCON == ^B10))
            THEN VODATA = LT_ME;
      ELSE WHEN (TM & (VCON == ^B11))
            THEN VODATA = LT_LO;
      ELSE
            VODATA = [0,0,0,0,0,0,0,0];

end TSRUMUX
```

5.2.5 Quelltext für Systemcontroller

```
module      SYSCON

declarations
"Pin-Zuweisung
      REC_BSY               pin 15;
      TSRU_BUSY             pin 62;
      XCRDY0..XCRDY3        pin 66,65,64,63;
      XCRDY                 node;
      OFF0..OFF3            pin 5,4,3,67;
      OF_FULL               node;
      DMF0..DMF3            pin 9,8,7,6;
      IF_FULL               node;
```

```
      BA0..BA3                pin 10,11,12,13;
      SCSEL                   pin 14;

      RW_TIM0,RW_TIM1         pin 46,45;
      SCD2,SCD3               pin 44,43;

      DM_TIM,M_TIM            pin 61,48;

      XCINIT,TMODE            pin 42,41;

      DAVAL,XCBUSY,XCCE       pin 60,59,57;
      XC_ENABLE               node istype 'reg_d,buffer';
      SC0..SC7                node istype 'reg_d,buffer';

      DT28,DT104              pin 33,32;

      XCCLK                   pin 16;
      SF0..SF3                pin 28,26,24,22;
      SFREG                   node istype 'reg_d,buffer';

      ADR   = [BA3..BA0];
      SC    = [SC7..SC0];

      START_XC    = ^B0001;
      STOP_XC     = ^B0010;

PLSI PROPERTY 'CLK XCCLK CLK0';
PLSI PROPERTY 'PULLUP ON';

equations
      "*****************************************************************
      "Board/Pipe-Busy-Flag
      OF_FULL     = !OFF0 # !OFF1 # !OFF2 # !OFF3;
      IF_FULL     = !DMF0 # !DMF1 # !DMF2 # !DMF3;

      SFREG.D           = 1;
      SFREG.CLK   = SF0 # SF1 # SF2 # SF3;

      TSRU_BUSY   =  SFREG;
      "*****************************************************************
      "Starte Xilinxe & Step-Counter
      XC_ENABLE.D = 1;
      XC_ENABLE.CE      = !XCBUSY;
      XC_ENABLE.CLK     = TMODE & SCSEL & (ADR == START_XC);
      XC_ENABLE.RE      = (TMODE & SCSEL & (ADR == STOP_XC))
                           # (SC == ^H46);

      SC.D        = SC.Q + 1;
      SC.CLK      = XCCLK;
      SC.CE       = XC_ENABLE;
      SC.RE       = !XC_ENABLE;

      XCCE        = XC_ENABLE;
      "*****************************************************************
      XCRDY       = XCRDY0 # XCRDY1 # XCRDY2 # XCRDY3;
      "*****************************************************************
      "Timing
      DM_TIM      = DT28 & DT104;
      M_TIM       = DT28 & DT104;
```

```
end syscon
```

5.3 Auszüge aus den C-Programmen

5.3.1 C-Header-Datei „TAPS.H“

Diese Datei enthält alle wichtigen Adressdefintionen sowie alle Datenstrukturen, die im Testbetrieb benutzt wurden.

```
#define BOARD_ADRESSE
      0xff800000

/* Reset Adressen Taps Board */
#define RESET_IN_FIFOS        0x02
#define RESET_OUT_FIFOS       0x04
#define RESET_DEMUX           0x06
#define RESET_MUX             0x08
#define RESET_XILINX          0x0a
#define RESET_SYSCON          0x0c
#define RESET_CLOCK           0x0e

/* Xilinx config commands  */
#define XILINX                0x10
#define XILINX_STATUS         0x12
#define LOAD_LOWT             0x14
#define LOAD_HIGHT            0x16

#define PROG                  0x0001
#define NO_PROG               0x0002
#define DIN_H                 0x0004
#define DIN_L                 0x0008
#define CCLK_H                0x0010
#define CCLK_L                0x0020

unsigned short STD_LOWT       =
0x0003;
unsigned short STD_HIGHT      =
0x00ff;

/* Board-Statusregister */
#define BOARD_STATUS
      0x60

/* VME Zugriffadressen Test Modus */
#define W_IFF1                0x20
#define W_IFF2                0x22
#define W_IFF3                0x24
#define W_IFF4                0x26
#define W_IFF5                0x28
#define W_IFF6                0x2a
#define W_IFF7                0x2c
#define W_IFF8                0x2e
#define W_IFF9                0x30
#define W_IFF10               0x32
#define W_IFF11               0x34
#define W_IFF12               0x36
#define W_IFF13               0x38
#define W_IFF14               0x3a
#define W_IFF15               0x3c
#define W_IFF16               0x3e

const unsigned long IFF[] =
{0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,
0x2e,
0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0
x3e};

#define R_OFF                 0x40
#define G_HOB                 0x42
#define G_MOB                 0x44
#define G_LOB                 0x46

const unsigned long OFF[] =
{0x40,0x42,0x44,0x46};

#define XILINX_START          0x50
#define XILINX_STOP           0x52

/* Testmodus */
#define TESTMODUS             0x80

#define TM_HIGH               0x0001
#define TM_LOW                0x0002

/* Eingabeformat */
typedef struct
{
      unsigned long adr;
      unsigned short val;
} indatatype;

/* Ausgabeformat */
typedef struct
{
      int hdb;
      int mdb;
      int ldb;
} outdatatype;

/* Ergebnistyp */
typedef struct
{
      int es;
      int x;
      int y;
      int ct;
} dc_datatype;
```

```
#define EVENT_END       0x55

/* TAPS-Datenformat */
typedef struct
{
      int EID;
      int ZaD;
} event_headertype;
```

```
typedef struct
{
      int BNUM;
      int DNUM;
      int EiK;
} event_datatype;

/* Datenmatrix */
#define MAX_X     16
#define MAX_Y     63
```

5.3.2 Allgemeine Routinen zur Boardsteuerung

```
/* allgemeine Routinen */

void twrite(unsigned long offset,unsigned short data)
/* schreibt Datenwort in die Offsetadresse */
{
      *((unsigned short *) (BOARD_ADRESSE + offset)) = data;
}

unsigned short tread(unsigned long offset)
/* liest Datenwort von Offsetadresse */
{
      return *((unsigned short *) (BOARD_ADRESSE + offset));
}

void warte()
{
      tsleep(0x80000006);
}
```

5.3.3 C-Routine zur FPGA-Konfiguration

```
/* Xilinx-Laderoutinen */

void prog(int l) /* Steuerung der PROG-Pins */
{
      if(l)
            twrite(XILINX,PROG);
      else
            twrite(XILINX,NO_PROG);
      warte(); /* Fuer korrekte Ausfuehrung unbedingt notwendig */
}

void din(int l) /* Setzt oder Loescht DIN-Pin */
{
      if(l)
            twrite(XILINX,DIN_H);
      else
            twrite(XILINX,DIN_L);
}

void cclk(int l) /* Generiert Configuration-CLocK */
{
      if(l)
```

```
            twrite(XILINX,CCLK_H);
      else
            twrite(XILINX,CCLK_L);
}

static void Str2Bit(char *Str) /* Schiebt Konfigurationsstring in Xilinx */
{
      int Pos;
      for (Pos = 0; Str[Pos] == '1' || Str[Pos] == '0';Pos++)
      {
            if (Str[Pos] == '0')
                  din(0);
            else
                  din(1);
            cclk(1);
            cclk(0);
      }
}

int init() /* Auswerten des INIT-Signales */
{
      unsigned short val;
      val = tread(BOARD_STATUS);
      return val & 0x0001 ? 1 : 0;
      warte();
}

int done()
{
      unsigned short val;
      val = tread(BOARD_STATUS);
      return val & 0x0002 ? 1 : 0;
      warte();
}

int xilinx_load()
{
      int i, ExitCode, schrott,a;
      char confile[80];
      FILE *xifile;
      char Line[255];

      printf("Name der Konfigurationsdatei: ");
      scanf("%s",confile);

      printf("Oeffne Datei %s\n",confile);
      if ((xifile = fopen(confile,"r")) == NULL)
      {
            printf("Fehler beim Oeffnen der Datei !!!\a\n");
            exit();
      }
      else printf("Datei: %x\n",(unsigned long) xifile);
      if (xifile != NULL)
            printf("Oeffnen erfolgreich...\n");
      else
      {
            printf("Schwachsinn...\n");
            exit();
      }
      sleep(5);
      i = 0;
```

```
	printf("Starte Konfigurationszyklus...\n");
	prog(1);
	printf("PROG-Pin gesetzt\n");
	prog(0);
	printf("PROG-Pin zurueckgesetzt\n");
	printf("Warte auf das Ende der Intialisierung");
	while(init())
	{
		i++;
		if (!(i % 10))
			printf(".");
		sleep(1);
	}
	printf("\nInitialisierung erfolgreich !\n");
	printf("Schreibe Konfigurationsdaten\n");
	if (!(init()) && !(done()))
	{
		a = 0;
		while (schrott = fgets(Line,255,xifile) != 0 && (!(init())))
		{
			switch(Line[0])
			{
				case '0':
				case '1':
					Str2Bit(Line);
					a++;
					break;
				default:
					printf("\t\a%s",Line);
					break;
			}
		}
		printf("Zeile %d\n",a);
		printf("Schrott: %d\n",schrott);
		sleep(2);
		if (!(init()))
		{
			Str2Bit("1111111111111111\n");
			printf("Ende des Bit-Transfers\n");
			sleep(1);
			if (done())
			{
				printf("Konfiguration erfolgreich !!!\n\a");
				ExitCode = 0;
			}
			else
				ExitCode = 1;
		}
	}
	fclose(xifile);
	return ExitCode;
	twrite(LOAD_LOWT,LOT);
	warte();
	twrite(0,0);
	warte();
	twrite(LOAD_HIGHT,HIT);
	warte();
	twrite(0,0);
}
```

5.3.4 Beispiel einer Testroutine

```
/* Tests mit realen TAPS-Daten */

int normiere_energie(int val,int n)
{
      int temp;
      temp = ((val * 255) /  n);
      if(temp > 255)
        temp = 255;
      return temp;
}

void schreibe_matrix()
{
      int i,j;
      for(i = 0;i <= MAX_X;i++)
        for(j = 0;j <= MAX_Y;j++)
        {
          twrite(IFF[i],(0xff & datenmatrix[i][j]));
        }
}


void lese_ergebnisse(int mw)
{
      int i,j;
      outdatatype val;
      dc_datatype dc;
      int ende = 0;
      while(!ende)
      {
            val = TSRU_get_word();
            ende = ((val.mdb == 0x55) & (val.ldb == 0xfe)
                  & (val.hdb == 0x0f)) |
                   ((val.mdb == 0xff) & (val.ldb == 0xff));
            if(!ende)
            {
              dc = decode_outdata(val);
              if(!(dc.es == 0) &
                 !((dc.es == 254) & (dc.ct == 3)))
              {
                fprintf(lf,"Schauer bei X: %d  Y: %d mit E = %d
                gefunden\n",dc.x,dc.y,dc.es);
                lego[dc.x][dc.y]++;
                histo[dc.es]++;
                depen[dc.x][dc.y] = depen[dc.x][dc.y] + dc.es;
                if(mw)
                  outmatrix[dc.x][dc.y] = dc.es;
              }
            }
      }
}

int bestimme_norm(int zyklen)
{
      event_headertype bnh;
      event_datatype bnd;
```

```
      int abbruch = 0;
      int cyc = 0;
      int max_eid,max_bn,max_dn,max_e = 0;
      int omax1,omax2,omax3 = 0;
      int i;
      int norm;
      printf("Bestimme Norm");
      while(!feof(df) && !abbruch)
      {
            fscanf(df,"%d%d",&bnh.EID,&bnh.ZaD);
            printf(".");
            for(i = 1;i <= bnh.ZaD;i++)
            {
              fscanf(df,"%d%d%d",&bnd.BNUM,&bnd.DNUM,&bnd.EiK);
              if (max_e < bnd.EiK)
              {
                omax3   = omax2;
                omax2   = omax1;
                omax1   = max_e;
                max_e   = bnd.EiK;
                max_bn  = bnd.BNUM;
                max_dn  = bnd.DNUM;
                max_eid = bnh.EID;
              }
            }
            cyc++;
            if(cyc == zyklen)
              abbruch = 1;
      }
      if( ((omax1 + omax2 + omax3) / 3) < (max_e - 1000) )
            norm = ( omax1 + omax2 + omax3 + max_e ) / 4;
      else
            norm = max_e;
      printf("\nNorm bestimmt\n");
      printf("Norm ist %d\n",norm);
      fprintf(lf,"Verwendete Norm: %d\n",norm);
      fprintf(lf,"Norm berechnet aus Event %d mit folgenden Daten:\n",
            max_eid);
      fprintf(lf,"B# %d\tD# %d\tEnergieinhalt [100KeV] %d\n\n",
            max_bn,max_dn,max_e);
      return norm;
}

void test_maxnorm()
{
      int zyklen = 0;
      int norm;
      int abbruch = 0;
      int cyc = 0;
      int i,j;
      FILE *legofile,*histofile;
      event_headertype eh;
      event_datatype ed;
      int ne,dx,dy;
      initialisiere_ergebnisspeicher();
      if ( (df = fopen("helium_events.dat","r")) == NULL)
        {
          printf("cannot open file !\n");
          exit(1);
        }
      if ((lf = fopen("testrun.maxnorm.log","w")) == NULL)
```

```
{
  printf("cannot open log-file !\n");
  exit(1);
}
printf("\n");
printf("Anzahl der Events [0 =EOF] (Standard %3d): ",zyklen);
scanf("%d",&zyklen);
norm = bestimme_norm(zyklen);
fclose(df);
if ((df = fopen("helium_events.dat","r")) == NULL)
{
  printf("cannot open file !\n");
  exit(1);
}
while(!feof(df) && !abbruch)
{
      fscanf(df,"%d%d",&eh.EID,&eh.ZaD);
      fprintf(lf,"Event %d mit %d aktiven Modulen\n",eh.EID,eh.ZaD);
      printf("Bearbeite Event %d mit %d aktiven Modulen\n",eh.EID,
            eh.ZaD);
      loesche_matrix();
      for(i = 1;i <= eh.ZaD;i++)
      {
            fscanf(df,"%d%d%d",&ed.BNUM,&ed.DNUM,&ed.EiK);
            ne = normiere_energie(ed.EiK,norm);
            dx = (ed.DNUM - 1) % 8;
            dy = (ed.DNUM / 8) + BOFFY[ed.BNUM - 1] + 1;
            datenmatrix[dx][dy] = ne;
      }
      cyc++;
      if (cyc == zyklen)
            abbruch = 1;
      printf("Matrix gefuellt\n");
      if(testmode())
      {
            sw_testmode(0);
            warte();
            sw_testmode(1);
      }
      else
            sw_testmode(1);
      warte();
      masterreset();
      warte();
      warte();
      warte();
      reset_iff();
      warte();
      reset_demux();
      warte();
      schreibe_matrix();
      printf("Download der Matrix erfolgreich\n");
      reset_syscon();
      warte();
      reset_off();
      warte();
      reset_xilinx();
      warte();
      toggle_xce(1);
      while(xcce());
      toggle_xce(0);
```

```
            lese_ergebnisse(0);
      }
      if((legofile = fopen("lego.dat","w")) == NULL)
      {
            printf("cannot open lego.dat !\n");
            exit(1);
      }
      if((histofile = fopen("histo.dat","w")) == NULL)
      {
            printf("cannot open histo.dat !\n");
            exit(1);
      }
      for(j = 0;j <= MAX_Y;j++)
      {
        for(i = 0;i <= MAX_X;i++)
        {
            fprintf(legofile,"%d ",lego[i][j]);
        }
        fprintf(legofile,"\n");
      }
      for(i = 0;i <= 255;i++)
            fprintf(histofile,"%d\n",histo[i]);
      printf(lf,"%d Events wurden verarbeitet\n",cyc);
      fclose(df);
      fclose(lf);
      fclose(legofile);
      fclose(histofile);
}
```

5.3.5 Programm zum Merging verschiedener RBT-Dateien für XILINX-daisy-chains

```
/* MERGE_RBT by S.Plueschke        Version 1.0             */
/*                                                         */
/* Erzeugt aus bis zu 10 RBT-Files einen                   */
/* RBT-File fuer die Programmierung einer                  */
/* Xilinx-Daisy-Chain                                      */

#include <stdio.h>
#include <string.h>

   char iname[9][80],oname[80];
   FILE *ifile[9],*ofile,*tfile;
   unsigned long sum;

int MRBT_Anzahl()
{
   int Anzahl;
   printf(" Anzahl der zu kombinierenden Dateien [2..10]: ");
   scanf("%d",&Anzahl);
   return Anzahl;
}

int MRBT_open_inputs(int anzahl)
{
   int i;
   for(i = 0;i <= anzahl; i++)
   {
      while(ifile[i] == NULL)
      {
```

```
            printf("Name der %d. Eingabedatei: ",i+1);
        scanf("%s",iname [i]);
        ifile[i] = fopen(iname [i],"r");
      }
   }
}

int MRBT_open_output()
{
   while((ofile == NULL) || (tfile == NULL))
   {
      printf("Name der Ausgabedatei: ");
      scanf("%s",oname);
      ofile = fopen(oname,"w");
      tfile = fopen("merge.tmp","w");
   }
   fclose(ofile);
}

void schreibe_zeile(char *zeile,int dat)
{
   int i;
   for(i = 0;zeile[i] == '0' || zeile[i] == '1' ;i++)
   {
      putc(zeile[i],tfile);
      sum++;
   }
   putc('\n',tfile);
}


int MRBT_countbits(int anzahl)
{
   int i,schrott;
   char line[500];
   char oline[500];
   for(i = 0;i <= anzahl;i++)
   {
      while((schrott = fgets(line,500,ifile[i]) != 0))
      {
            switch(line[0])
        {
            case '0':   if (strlen(line) > 40)
                        {
                      schreibe_zeile(line,i);
                    }
                    else
                    {
                        schreibe_zeile("01111111\n",i);
                    }
                    break;
          case '1': break;
          default : printf("%s",line);
                        break;
        }
      }
   }
   fclose(tfile);
}

int MRBT_close_all(int anzahl)
```

```
{
   int i;
   for(i = 0;i <= anzahl;i++)
   {
      fclose(ifile[i]);
   }
}

unsigned long pow(int b,int exponent)
{
   int i;
   unsigned long val;
   if(exponent)
   {
      val = b;
      for(i = 2;i <= exponent;i++)
      {
            val = val * b;
      }
   }
   else val = 1;
   return val;
}

void MRBT_out()
{
   char ostr[40],zeile[500];
   int i,schrott;
   sum = sum + 40;
   printf("Summe der Datenbits: %d\n",sum);
   ofile = fopen(oname,"w");
   fprintf(ofile,"daisy-chain RBT-File %s\n",oname);
   fprintf(ofile,"MERGE_RBT by S.Plueschke\n");
   strcpy(ostr,"111111110010");
   for(i = 0;i < 24 ;i++)
   {
      if(sum & pow(2,(23-i)))
         strcat(ostr,"1");
      else
         strcat(ostr,"0");
   }
   strcat(ostr,"1111\n");
   printf("%s",ostr);
   fprintf(ofile,ostr);
   tfile = fopen("merge.tmp","r");
   while((schrott = fgets(zeile,500,tfile) != 0))
   {
      fprintf(ofile,zeile);
   }
   fclose(ofile);
   fclose(tfile);
}

main()
{
   int DAT_ANZAHL,i;
   DAT_ANZAHL = 0;
   printf("************************************************************\n");
   printf("* MERGE_RBT V1.0                by            S.Plüschke *\n");
   printf("*                                          II.Phys.Inst. *\n");
   printf("*                                    Tel.: 0641/99-33275 *\n");
```

```
   printf("*********************************************************\n");
   printf("\n");
   while((DAT_ANZAHL < 2) || (DAT_ANZAHL > 10))
   {
     DAT_ANZAHL = MRBT_Anzahl();
   }
   DAT_ANZAHL = DAT_ANZAHL - 1;
   MRBT_open_inputs(DAT_ANZAHL);
   MRBT_open_output();
   sum = 0;
   MRBT_countbits(DAT_ANZAHL);
   MRBT_close_all(DAT_ANZAHL);
   sum++;
   MRBT_out();
   printf("Merging erfolgreich !!!\n");
   remove("merge.tmp");
}
```

Danksagung

Ich möchte nun die Gelegenheit nutzen, um mich bei all denen zu bedanken, die zum Gelingen dieser Arbeit ihr Scherflein beigetragen haben.
An erster Stelle sind hier Herr Prof. Dr. Wolfgang Kühn und Herr Prof. Dr. Volker Metag zu nennen, die mich in ihr Institut aufgenommen und mich mit dieser interessanten, herausfordernden Aufgabe betraut haben. Ich habe in dieser Zeit viel gelernt – Danke !!!
Danken möchte ich auch den Mitgliedern der Arbeitsgruppe HADES für das angenehme Arbeitsklima und die freundliche Unterstützung. Hier seien besonders Markus, Jörg, Erik, Hans und Arndt erwähnt, die mich bei keiner technischen Frage im Stich gelassen haben. Ein ganz besonderes Dankeschön gilt unserem „FH-Studenten“ Torsten, der mir in jeder Situation mit Rat und Tat zur Seite gestanden hat.
Auch den Mitgliedern der TAPS-Gruppe sei ein herzliches Dankeschön gewidmet.
Außerdem möchte ich all denen danken, ohne die im Institut eigentlich gar nichts laufen würde – Anita, Kerstin, Marianne, Werner, Jürgen, Rainer und Bernd und natürlich unsere Werkstätten.
Mein besonderer Dank gilt Stefan Schade, mit dem ich oft und ausgiebig über Gott und die Welt diskutieren konnte.
Desweiteren möchte ich an dieser Stelle meinen Eltern danken, ohne deren Unterstützung und Förderung das alles gar nicht möglich gewesen wäre.
Und schließlich möchte ich noch jemandem danken, ohne den ich die letzten sehr anstrengenden Monate nicht so gut überstanden hätte. Dieser Dank geht an „meine kleine Maus“, die mich immer wieder aufgebaut hat, wenn es mal nicht so lief, die immer zugehört, auch wenn ich ihr auf die Nerven gegangen bin – ich sage einfach: DANKE