Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Diplomarbeit

im Studiengang Physik

vorgelegt von

Jan Knopf

aus Lörrach

2004

Aufbau eines Auslesesystems für die Äusseren Spurkammern des LHCb-Detektors

Die Diplomarbeit wurde von *Jan Knopf* **ausgeführt am**

Physikalischen Institut

unter der Betreuung von

Herrn Prof. *Ulrich Uwer*

Aufbau eines Auslesesystems für die Äussere Spurkammern des LHCb-Detektors:

Das Physikalische Institut ist am Aufbau des Äusseren Spurkammersystems des LHCb-Detektors beteiligt. Im Rahmen dieser Beteiligung werden in Heidelberg Teile der Ausleseelektronik für die Driftkammern entwickelt, und zwar in Form eines TDC-Chips sowie Interfacekarten für die optische Datenübertragung. Ziel dieser Diplomarbeit war es, erstmalig alle Komponenten der gesamten Ausleseelektronik aufzubauen und in Betrieb zu nehmen. Dazu gehörte auch die Entwicklung eines Programmes, das später auf den FPGA-Bausteinen des L1-Zwischenspeichers die Synchronisation der empfangenen Daten sowie deren Zwischenspeicherung sicherstellen soll. Da diese Zwischenspeicherkarte noch nicht zur allgemeinen Verfügung stand, wurden Synchronisations- und Zwischenspeicher-Algorithmen auf einer kommerziellen PCI-Steckkarte mit einem FPGA der gleichen Familie implementiert, was zudem den Vorteil hatte, dass die Daten mittels eines PC ausgelesen werden konnten.

Mit der vollständigen Auslesekette wurden dann zum ersten Mal der OTIS TDC-Chip sowie die optische Datenübertragung unter realistischen Bedingungen erfolgreich getestet. Im weiteren wurde der Aufbau genutzt, um ein Driftkammermodul erstmalig mit der vorgesehenen Elektronik auszulesen. Die aufgezeichneten Driftzeitspektren kosmischer Myonen bzw. einer radioaktiven Ru-Quelle demonstrieren die korrekte Funktion der Auslesekette.

Realisation of a data acquisition system for the outer tracking chamber of the LHCb experiment:

The Physics Institute participates in the construction of the outer tracking system of the LHCb detector. The institute develops parts of the readout electronics such as a TDC chip and the interface boards for the optical data transmission. The topic of this diploma thesis was, for the first time, the assembly and the commissioning of all readout chain components. It was necessary to develop a program which is foreseen to run on the FPGA components of the L1 buffer board to synchronise and to buffer the received data. As the L1 buffer boards were not generally available, a PCI card containing a FPGA of the same family was used to implement the synchronisation and buffer algorithms. This had the additional advantage that the data could be readout by a PC.

The complete readout chain was used to test the OTIS TDC chip and the optical data-transmission for the first time under realistic conditions. Moreover, the setup was used to readout a drift-chamber for the first time with the future electronics. The recorded drift-time spectra of cosmic muon and a radioactive Ru source demonstrate the correct functioning of the readout chain.

Danksagung

Ich möchte an dieser Stelle meinen Dank und meine Hochachtung gegenüber den Personen ausdrücken, die zum Gelingen dieser Arbeit beigetragen haben.

An erster Stelle gilt mein besonderer Dank Herrn Professor Ulrich Uwer für die Möglichkeit, diese Arbeit durchführen zu können, sowie für die gute Betreuung und geleisteten Hilfestellungen.

Bedanken möchte ich mich auch bei den OTIS-Entwicklern insbesondere Harald Deppe und Uwe Stange, die mir den Umgang mit der Laborausstattung und den Chips beibrachten.

Ein ganz besondere Dank gilt Dirk Wiedner, der mich durch die gesamte Diplomarbeit hindurch begleitet und betreut hat und dabei mit stets guter Laune und der richtigen Portion Humor die Macken meines Programms ertragen hat. Danke auch für die zahlreichen Tipps im Umgang mit der Verwaltung. Möge er nie die Datennahme über ein HexaTetris Spiel vergessen ;) .

Weiterhin möchte ich mich bei allen Mitgliedern der HE-Gruppe des Physikalischen Instituts bedanken, insbesondere bei Sebastian Bachmann, Yuri Bagaturia, Herrn Professor Franz Eisele, Tanja Haas, Ralf Muckerheide und Michael Walter.

Mein Dank gilt auch allen Kollegen vom NIKHEF in Amsterdam, an der TU Dresden sowie am EPFL in Lausanne. Insbesondere danke ich Guido Haefeli für die entscheidenden Hinweise im Umgang mit den Programmpaketen sowie Mirco Nedos für die schnelle Umsetzung aller Vorschläge.

Ein Danke geht auch an Birgit Schabinger, Monika Mehler und Stefan Reinauer sowie meinem ehemaligen Physik-LK Lehrer Herrn Göbel.

Zu guter Letzt möchte ich meinen tiefempfunden Dank gegenüber meinen Eltern Inge-Maj Lönnquist-Knopf und Wolfgang Knopf aussprechen, die mich zu jeder Zeit unterstützt und ermutigt haben.

Heidelberg, den 22. Dezember 2004

Inhaltsverzeichnis

1 Einleitung

Das Gebiet der Teilchenphysik untersucht die kleinsten beobachtbaren Strukturen. Dabei will man Antworten auf die Fragen finden, woraus Materie letztlich aufgebaut ist, wie die kleinsten Bausteine miteinader wechselwirken, bis hin zur Frage, warum unser Universum aus Materie und nicht aus Antimaterie aufgebaut ist.

Unser heutiges Wissen über die Antworten auf diese Fragen findet sich im Standardmodell wieder. Als kleinste Teilchen kennt es die sechs Quarks (up, down, charm, strange, top und beauty (bottom)) und die sechs Leptonen (e, μ, τ, ν_e, ν_μ, ν_τ) sowie deren Antiteilchen. Die Wechselwirkungen dieser Teilchen untereinader werden durch sogenannte Austauschbosonen vollzogen. In den letzten 40 Jahren wurden zahllose Experimente durchgeführt, die das Standardmodell mit einer grossen Präzision bestätigt haben.

Leider beinhaltet das Standardmodell mindestens 18 Parameter, deren Werte gemessen werden müssen. Es liefert keine Erklärung dafür, warum ein Parameter gerade den gemessenen Wert annimmt. Aus diesem Grund wird es auch „nur“ ein Modell genannt und nicht in den Stand einer Theorie erhoben. Die heutigen Experimentatoren versuchen deshalb Bereiche des Standardmodell zu finden, in denen die Vorhersagen nicht mit den Messungen übereinstimmen.

Um in diese Bereiche vorzudringen, benötigt man Teichenbeschleuniger. In ihnen werden Teilchen wie zum Beispiel Elektronen oder Protonen auf fast Lichtgeschwindigkeit beschleunigt und zur Kollision gebracht. Da nach der wohl berühmtesten Formel in der Physik, $E = mc^2$, die Energie mit der Masse verknüpft ist, enstehen durch die freiwerdende Energie in den Kollisionen deutlich schwerere Teilchen. Um immer schwerere Teilchen zu erzeugen, muss die Energie der kollidierenden Teilchen erhöht werden.

Es gibt zwei prinzipielle Möglichkeiten, einen solchen Beschleuniger zu bauen. Die erste besteht im Bau einer langen geraden Beschleunigungsstrecke. Diese werden Linearbeschleuniger genannt. Auf diese Weise soll z.B. der „TeV-Energy Superconducting Linear Accelerator“ (TESLA) am Deutschen Elektronen Synchrotron (DESY) bei Hamburg gebaut werden. Die zweite Möglichkeit besteht im Bau eines Ringbeschleunigers. Dabei durchlaufen die Teilchen vor ihrer Kollision mehrere Male dieselbe Beschleunigungsstrecke. Beide Vorgehensweisen haben jeweils ihre Vor- und Nachteile, auf die an dieser Stelle nicht weiter eingegangen werden soll.

Ein Ringbeschleuniger, in dem Protonen mit Protonen bei einer Schwerpunktsenergie von $\sqrt{s} = 14\,\mathrm{TeV}$ zur Kollision gebracht werden sollen, wird zur Zeit am CERN (Convention Européenne de la Recherche Nucléaire) installiert. Dort werden vier[1] grosse Experimente am sogenannten Large Hadron Collider (LHC) stattfinden:

[1] mit TOTEM sind es fünf

- Die Experimente ATLAS und CMS untersuchen dabei die gesamte zugängliche Physik.
- Das ALICE-Experiment will vor allem das Quark-Gluon-Plasma untersuchen, welches durch Kollisionen von Schwerionen erzeugt werden soll.
- Das LHCb-Experiment beschäftigt sich mit der Untersuchung von CP-verletzenden Zerfällen in B-Meson-Systemen.

Die Protonen (bzw. Pb-Ionen) laufen dabei in einem Tunnel mit einem Umfang von 27 km um, der auch schon für die LEP-Experimente genutzt wurde. Die oben genannten Experimente bestehen aus jeweils einem grossen Detektor, der an einen der vier Wechselwirkungspunkte aufgebaut wird.

1.1 LHCb

Das LHCb (Large Hadron Collider beauty) Experiment [1] soll CP-verletzende und andere seltene Zerfälle der B-Mesonen untersuchen. Aufgrund der grossen Schwerpunktsenergie ist es möglich, sämtliche B-Hadronen herzustellen. Die dafür nötigen $b\bar{b}$-Quarkpaare werden dabei hauptsächlich durch Gluon-Gluon-Fusion erzeugt. Die b-Quarks und damit auch die entstehenden B-Mesonen besitzen einen sehr grossen Longitudinalen Impulsanteil und verlassen den Detektor somit unter kleinen Winkeln zum Strahlrohr. Aus diesem Grund wird der Detektor als Vorwärtsspektrometer realisiert.

Der Detektor besteht aus dem Vertex-Locator (VELO), den inneren und äusseren Spurkammern (Inner-Tracker (IT) und Outer-Tracker (OT)), den elektromagnetischen und hadronischen Kalorimetern (ECAL bzw. HCAL), dem Ring-Imaging-Čherenkov-Zähler (RICH) sowie dem Myonsystem (siehe auch Abbildung 1.1).

Zur Vermessung der Teilchenspuren wird ein Magnetfeld benutzt. Dieses ändert die Flugbahn der bei der Kollision entstandenen geladenen Teilchen. Durch das Vermessen ihrer Spuren kann der Impuls dieser Teilchen bestimmt werden. Die erste Komponente zur Spurrekonstruktion ist der Vertex-Locator, der am Wechselwirkungspunkt positioniert ist. Er soll die dortigen Spuren mit einer sehr hohen Auflösung vermessen, um eine genaue Bestimmung von Sekundärvertices zu erreichen. Dabei ist er einer sehr hohen Strahlenbelastung aufgrund der hohen Teilchenflussdichten ausgesetzt. Um die Anforderungen zu erfüllen, wird der Vertex-Locator mit Siliziumstreifendetektoren realisiert. Aus denselben Gründen wird auch der Inner Tracker auf diese Weise gebaut. Er deckt die Region um das Strahlrohr ab. Im Gegensatz dazu wird der Outer Tracker, dessen Ausleseelektronik im Mittelpunkt dieser Arbeit steht, aus Driftkammern bestehen.

Im Experiment kommt es alle 25 ns zu einem Aufeinandertreffen der Protonen. Dies wird Bunch-Crossing (BX) genannt und liefert eine natürliche Zeitreferenz von 40 MHz.[2] Alle Driftzeitmessungen im äusseren Spurkammersystem erfolgen in Relation zu dieser Zeitreferenz. Für jedes Bunch-Crossing nimmt die Elektronik einen Messwert auf. Die

[2] Genauer von 40.0786 MHz ± 11.2 ppm.

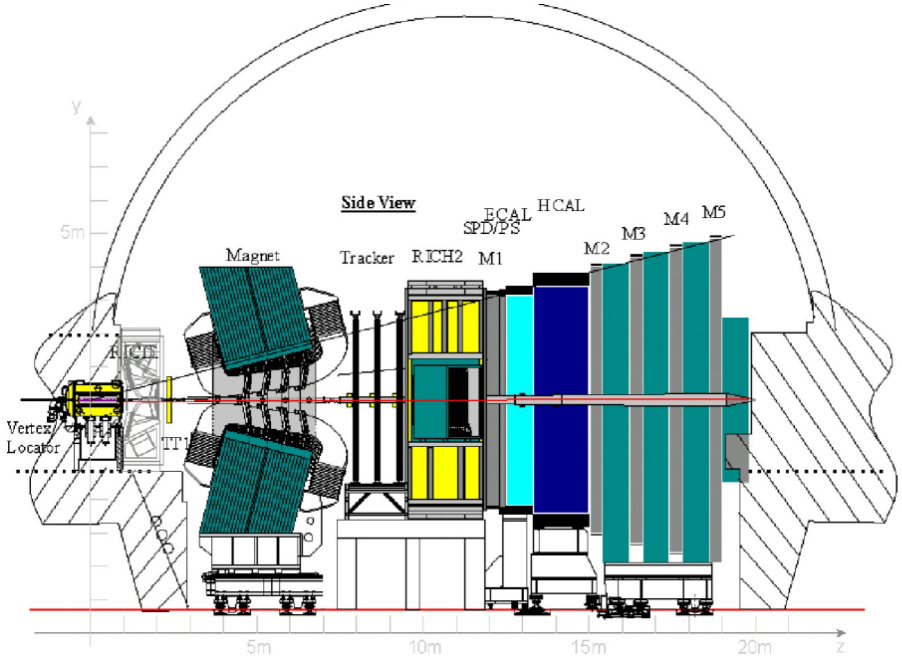

Abbildung 1.1: Der LHCb Detektor in der Seitenansicht

Entscheidung, ob die Messwerte gespeichert werden, wird von einem dreistufigen Triggersystem getroffen. Dieses besteht aus

1. L0-Accept
2. L1-Accept
3. Higher-Level-Trigger (HLT)

Die Eingangsrate für den L0-Trigger beträgt dabei 40 MHz. Die Entscheidung, das Ereignis weiter zu verarbeiten, wird der Elektronik mit einer Rate von maximal 1,11 MHz mitgeteilt. Die Zeit, um eine Entscheidung zu treffen, darf dabei bis zu $4\,\mu s$ betragen. Diese Zeit wird auch Latenz genannt.

Der L1-Trigger prüft das Vorhandensein eines Sekundärvertex. Dazu muss er Informationen aus dem Spurkammersystem auswerten. Bei einer Eingangsrate von 1,11 MHz wird die positive Entscheidungrate bei 40 kHz liegen. Die Latenz darf dabei maximal 1,6 s betragen.

Der Higher-Level-Trigger schliesslich sucht aus diesen Ereignissen die für die physikalische Analyse wichtigen B-Zerfallskanäle heraus und reduziert die zu speichernde Ereignisrate auf etwa 200 Hz.

1.2 Das Äussere Spurkammersystem

Das Äussere Spurkammersystem, auch Outer Tracker (OT) genannt, wird als Driftkammerdetektor realisiert. Es besteht aus drei identischen Stationen, die jeweils sechs auf fünf Meter messen. Jede Station setzt sich aus vier Stereolagen zusammen. Dabei wird die erste Lage senkrecht aufgehängt. Die zweite Lage wird um +5°, die dritte um -5°zur ersten gedreht montiert. Die vierte Lage wird wie die erste senkrecht aufgebaut.

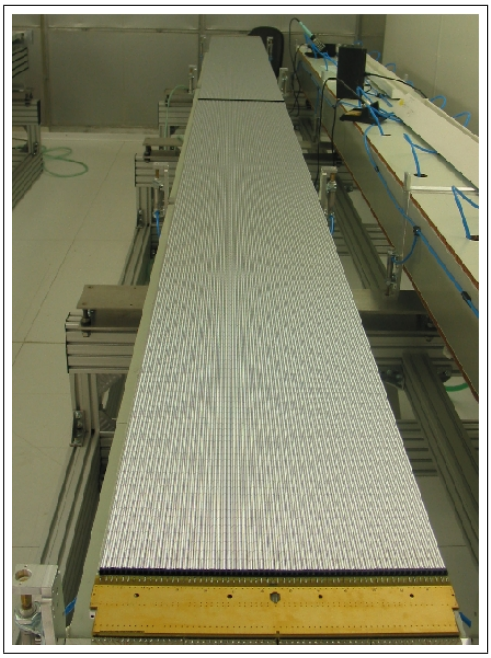

Abbildung 1.2: Ein offenes Driftkammermodul. Die Driftröhrchen werden an beiden Enden des Moduls ausgelesen.

Die Stereolagen setzen sich wiederum aus den Driftkammermodulen zusammen, die die Driftröhrchen (auch Straws genannt) enthalten (siehe Abbildung 1.2). Ein Modul hat eine Länge von fünf Meter bei einer Breite von 34 cm. An den oberen und unteren Seitenflächen sind auf der Innenseite der Module jeweils 64 Driftröhrchen befestigt. Um die Belegungsdichte im Experiment klein zu halten, sind diese in Höhe der Strahlröhre unterbrochen. Somit können an den beiden Enden eines Moduls jeweils 128, insgesamt also 256 Driftröhrchen ausgelesen werden.

Die benutzen Straws haben einen Durchmesser von fünf Millimeter. In deren Mitte wird ein $0,25\,\mu m$ dünner Draht gespannt. Um Signale mit dem Draht messen zu können, wird dieser auf ein Potential von ca. 1600 V gelegt und von einem Zählgas umspült.[3] Ein geladenes Teilchen, das durch ein Driftröhrchen fliegt, ionisiert dieses Gas. Die freigesetzten Elektronen[4] driften zum Anodendraht, wobei sie durch das elektrische Feld beschleunigt werden. Dabei können sie ihrerseits weitere Elektronen[5] aus den Gasmolekülen freisetzten. Durch diese Ladungslawine wird auf dem Draht ein Signal erzeugt, welches elektronisch ausgewertet wird.

Die Zeit zwischen dem Teilchendurchgang und dem auslesbaren Signal am Draht wird Driftzeit genannt. Wenn man die Geschwindigkeit kennt, mit der Elektronen in dem Gas zum Draht driften, kann man aus der gemessenen Zeitdifferenz den Abstand bestimmen, mit dem das Teilchen am Draht vorbeigeflogen ist (siehe Abbildung 1.3 auf der nächsten Seite). Im Outer Tracker soll auf diese Weise eine Ortsauflösung von $200\,\mu m$ erreicht werden. Da die Spur eines Teilchens durch mehrere solcher Driftkammern führt, kann man aus den einzelnen Ortsinformationen diese rekonstruieren. Daraus kann der Impuls des Teilchens bestimmt werden.

[3] Als Gas ist im Experiment $ArCO_2$ vorgesehen.
[4] Primärelektronen
[5] Sekundärelektronen

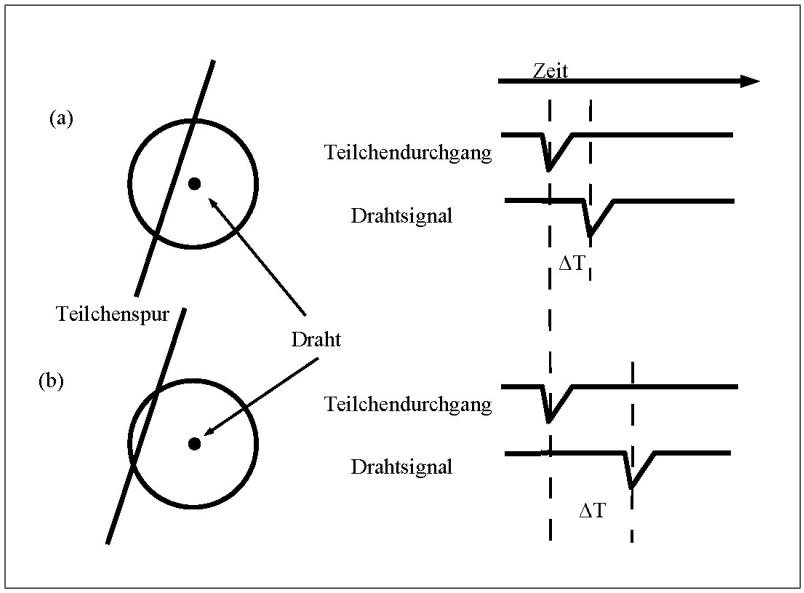

Abbildung 1.3: Die Driftzeit ist der Zeitunterschied zwischen Teilchendurchgang und dem Drahtsignal. In Fall (a) fliegt das Teilchen nahe am Draht durch das Driftröhrchen. Die freigesetzten Elektronen brauchen nur eine kurze Zeit, um zum Draht zu gelangen. Fall (b) zeigt dies für ein Teilchen, das am Rande des Driftröhrchens durchfliegt. Die Grösse des Zeitunterschiedes zwischen beiden Signalen liefert somit ein Mass für den Abstand zwischen Draht und Teilchenspur.

1.3 Outer-Tracker-Elektronik

Um durch die Zeitmessung keine Verschlechterung der Ortsauflösung zu verursachen, soll die Driftzeit mit einer Auflösung von 1 ns gemessen werden. Dazu wird das analoge Signal für jeden Draht zunächst verstärkt und digitalisiert. Anschliessend wird die Zeit zwischen diesem Puls und der Wechselwirkung im Detektor bestimmt. Diese Daten werden dann zu immer grösseren Datenblöcken vereinigt und gespeichert.

Da die Messung der Driftzeiten direkt am Modul erfolgt, muss die Elektronik in diesem Bereich strahlenhart sein. Für jede Wechselwirkung im Detektor müssen die Spuren der entstehenden Teilchen bestimmt werden. Dies bedeutet, dass die Driftzeitmessungen mit einer Rate von 40 MHz erfolgen muss. Die Auslese der gemessenen Daten erfolgt mit einer Rate von 1,11 MHz. Die Outer Tracker Elektronik [2] benutzt für jede dieser Aufgaben Karten mit speziell entwickelten Chips (Abbildung 1.4).

Zunächst wird mit dem „Amplifier Shaper Discriminator with BaseLine Restoration“ (ASDBLR) Chip [4] das Ladungssignal ($\sim 60\,fC$) von den Drähten verstärkt und diskriminiert. Daraus wird ein differentieller digitaler Puls erzeugt und an die folgende Elektronik weitergegeben. Eine Vorverstärkerkarte ist mit zwei ASDBLR-Chips bestückt und kann damit 16 Drähte auslesen.

Zwei Vorverstärkerkarten geben ihre Pulse auf eine OTIS-Karte. [6]. Diese beherbergt

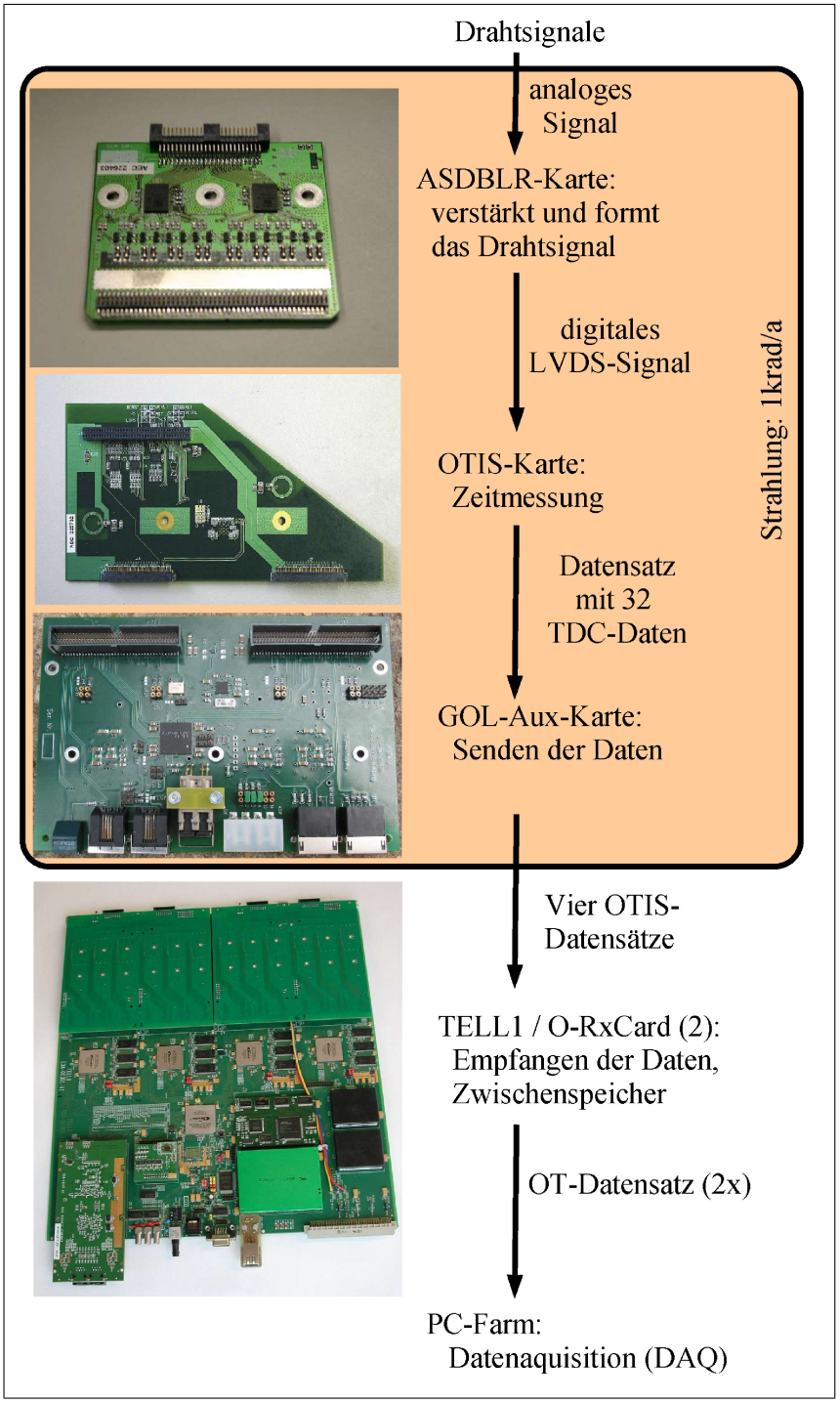

Abbildung 1.4: Aufbau der Ausleseelektronik

den OTIS-Chip [5]. Der OTIS-Chip arbeitet als Time to Digital Converter (TDC) und bestimmt die Zeitdifferenz zwischen dem diskriminierten Ladungspuls des Vorverstärkers und dem Wechselwirkungszeitpunkt. Diese Zeitdifferenzen werden für jeden Draht als TDC-Daten gespeichert. In Kapitel 1.5 werden die Eigenschaften des OTIS-Chips genauer vorgestellt. Vier der OTIS-Karten werden an eine optische Senderkarte (GOL-Aux-Karte [7]) angeschlossen. Diese fasst die Daten zusammen und bildet daraus einen seriellen Datenstrom, der über eine optische Leitung an die weitere Elektronik gesendet wird.

Mit jeweils acht Vorverstärkerkarten, vier TDC-Karten und einer optischen Senderkarte können 128 Drähte ausgelesen werden, was einem Ende eines Moduls entspricht. Um einen guten Kontakt zwischen dem Modul und der Ausleseelektronik zu gewährleisten, sind diese Karten gemeinsam in der sogenannten Frontendbox untergebracht.[6] (Abbildung 1.5)

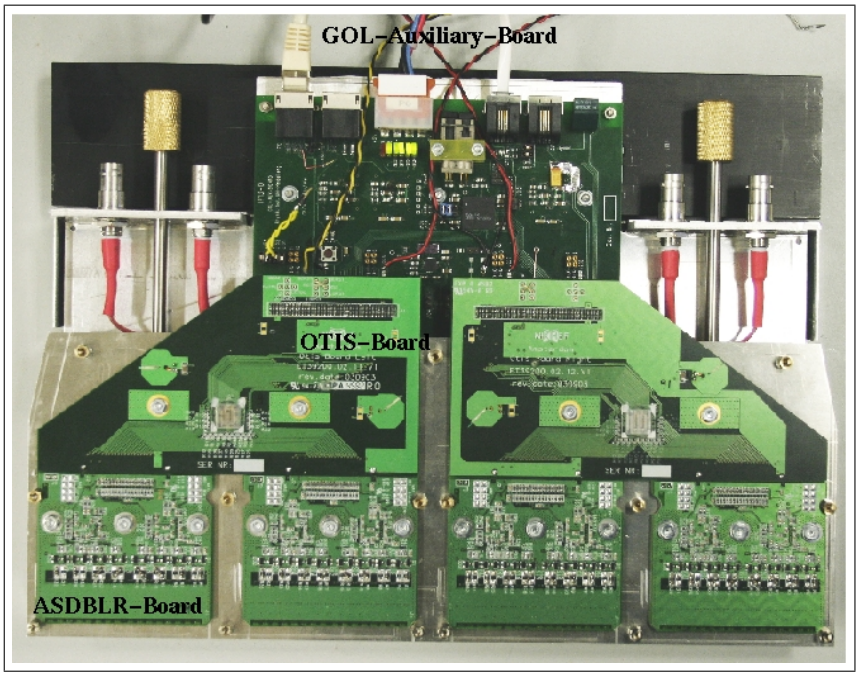

Abbildung 1.5: Eine voll ausgestattete Frontendbox mit Vorverstärkerkarten, TDC-Karten und optischer Senderkarte.

Die weitere Datenverarbeitung findet ausserhalb des Detektor in einem separaten Elektronikraum statt. Die serialisierten Daten von bis zu zwölf optischen Senderkarten werden von einer optischen Empfängerkarte (O-RxCard [10]) entgegengenommen.[7] Diese wandelt die optischen Signale in elektrische zurück. Die TELL1-Karte [12], auf der zwei optische Empfängerkarten sitzen, bearbeitet die Daten von inzwischen neun kompletten

[6] Hinzu kommt noch die Hochspannungsversorgung.
[7] Im Experiment werden 9 Leitungen benutzt.

Driftkammermodulen und speichert sie im Outer-Tracker-Datenformat ab. Die physikalisch interessanten Daten werden von allen TELL1-Karten zur Speicherung an eine PC-Farm geschickt.

1.4 Steuerung der Elektronik

Die Elektronik wird von zwei Systemen gesteuert und kontrolliert. Das erste System wird synchron zum 40 MHz-Takt betrieben und mit „Timing and Trigger Control System“ (TTC-System) bezeichnet. Es ist für die Verteilung

- der Triggersignale,
- der Resetsignale,
- der Testpulse
- und der Taktsignale

zuständig. Diese Signale werden an die Frontendboxen verteilt. Dort sorgen die optisch Senderkarten für die weitere Verteilung der Signale an die angeschlossenen Chips.

Das zweite System ist die „Slow-Control“. Mit diesem können Register in den Chips gesetzt und weitere Einstellungen vorgenommen werden. Da diese Einstellungen während einer Messung nicht verändert werden, braucht das verwendete Protokoll nicht mit 40 MHz betrieben zu werden. Für die Elektronik im äusseren Spurkammersystem kommt eine I^2C-Logik [27] zum Einsatz.

Das TTC-System für den lokalen Aufbau, der in dieser Arbeit realisiert wurde, besteht aus den in Abbildung 1.6 gezeigten Komponenten. Das TTCvx-Modul [15] nimmt einen Referenztakt entgegen und gibt diesen an das TTCvi-Modul [14] weiter. Diese generiert ein Datensignal [18], das alle Trigger-, Reset- und Taktsignale enthält. Dabei besteht die Möglichkeit, den Trigger generieren zu lassen, oder einen externen Trigger in das TTCvi-Modul einzuspeisen. Dieses Datensignal wird an das TTCvx-Modul zurückgegeben und in ein optisches Signal umgewandelt. Beide TTC-Module sind in einem VME-Rahmen untergebracht und können mit einem Programm [19] angesprochen werden. Die optischen Signale werden an den TTCrx-Chip [16] übertragen. Dieser stellt die in den Daten enthaltenen Signale auf einzelnen Pins als TTL-Signal zur Verfügung.

Für die lokale „Slow-Control“ (Abbildung 1.7) wurde ein LabVIEW-Programm von [24] übernommen, welches die Register im OTIS-Chip setzen kann. Dazu wurde ein separater Laptop, auf dem das Programm lief, über die parallele Schnittstelle an eine kommerzielle „ELV-I^2C-PC-Interface-Box“ angeschlossen. Diese generierte die nötigen I^2C-Signale. Das Programm im Rahmen eines Miniforschungsprojektes [22] später erweitert, um auch die Register der optischen Senderkarte beschreiben zu können.

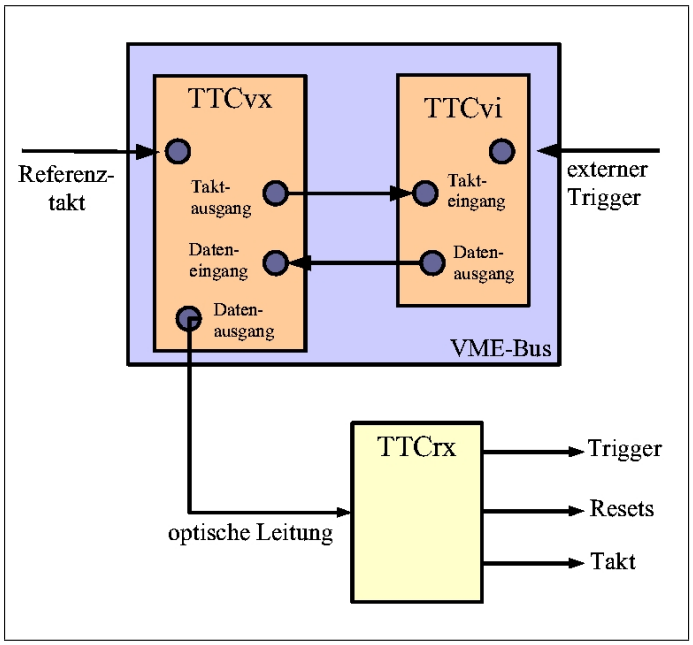

Abbildung 1.6: Schematischer Überblick über das lokale TTC-System.

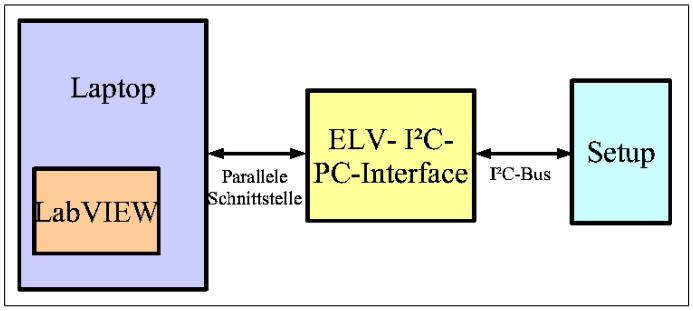

Abbildung 1.7: Schematischer Überblick über die lokale Slow-Control.

1.5 Der OTIS-Chip

Der OTIS-Chip [5] wurde am ASIC-Labor der Universität Heidelberg entwickelt. Seine Aufgabe besteht in der Messung von Driftzeiten in Relation zum Systemtakt. Die Auflösung soll dabei besser als 1 ns sein. Er wird in einem $0,25\mu m$-CMOS-Prozess hergestellt und ist als strahlenhart qualifiziert.

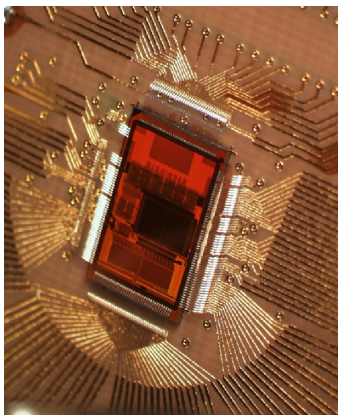

Abbildung 1.8: Der OTIS-TDC

Die erste Version des Chips wurde im Mai 2002 submittiert und als OTIS 1.0 bezeichnet. Eine verbesserte Version wurde im Oktober 2003 submittiert und OTIS 1.1 genannt. Mit beiden Chips konnte im Rahmen dieser Diplomarbeit gearbeitet werden. Im Oktober 2004 kam der Chip in der Version 1.2 aus der Produktion zurück.

Zur Messung der Driftzeiten speichert er in jedem Takt für jeden Kanal ein Datenwort. Dazu durchläuft der Systemtakt eine aus in 64 Teilen (im folgenden mit Bins bezeichnet) bestehende Inverterkette. Sobald ein Ladungspuls vom Vorverstärker einen Treffer anzeigt, wird für diesen geprüft, in welchen der Bins der $0 \rightarrow 1$-Übergang des Systemtaktes liegt. Die Nummer des Bins wird als 6 Bit breite TDC-Zeit kodiert und gespeichert. Gibt es zum Beispiel im achten Bin einen Treffer, so fand in der Zeit zwischen $\left(\frac{8}{64}\ 390\ \text{ps}\right)$ und $\left(\frac{9}{64}\ 390\ \text{ps}\right)$ nach einer steigenden Taktflancke der Übergang statt.[8] Siehe auch Abbildung 1.9.

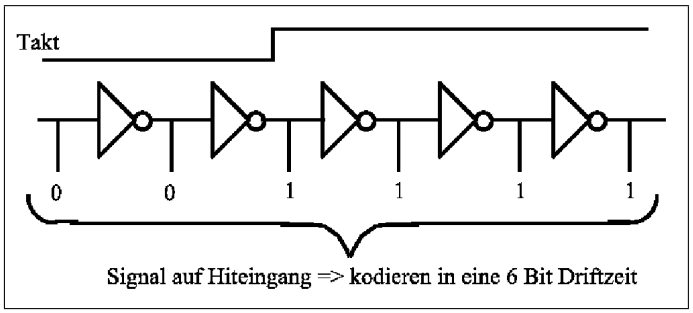

Abbildung 1.9: Erzeugen der Driftzeitinformation im OTIS-Chip (vereinfachte Darstellung). Der Systemtakt wird durch die Inverterkette geschoben. Die steigende Taktflancke führt in einem Bin zu einem 0/1-Übergang. Bei einem Treffersignal wird die Nummer dieses Bins als 6 Bit TDC-Zeit gespeichert.

Bekommt der OTIS nun vom TTC-System einen Trigger geschickt, so schaut er von seiner aktuellen Position aus in der Pipeline die im Register „Latency“ eingestellte Anzahl von Zeilen zurück. In dieser Zeile wird für jeden Kanal nach Treffern gesucht und diese als TDC-Zeit an die weitere Logik übergeben. Je nach Betriebsmodus werden bis zu

[8] 1 Bin entspricht 390 ps (25 ns durch 64 Bins).

zwei weitere Zeilen nach Treffern durchsucht,was notwendig ist, um Driftzeiten grösser als 25 ns zu messen.

Diese TDC-Zeiten werden zu einem Datensatz zusammengefügt und gesendet. Für die OTIS-Versionen 1.0 und 1.1 hat dieser das in Tabelle 1.1 gezeigte Format.[9] Der OTIS-Chip überträgt dabei die Daten nur nach einem Trigger, was eine Synchronisation auf diese Daten bei deren Auswertung erforderlich macht.

Bit Nr.	OTIS 1.0	OTIS 1.1
0..11	OTIS ID [11..0]	
12	„0“	„1“
13	„0“	Read Mode
14	Selftest failed	No. of BX [1]
15	Buffer overflow	No. of BX [0]
16	DLL lock lost	Truncation
17	SEU	Playback
18	No. of BX [1]	SFT, DLL, SEU
19	No. of BX [0]	Buffer overflow
20	Read Mode	Event-ID [3]
21	Playback busy	Event-ID [2]
22	„0“	Event-ID [1]
23	„0“	Event-ID [0]
24..31	BX-ID [7..0]	
32..39	TDC data 0	
⋮	⋮	
280..287	TDC data 31	

Tabelle 1.1: OTIS-Datenformat

Die TDC-Daten setzen sich aus zwei Bit, die anzeigen, in welchem „Bunch-Crossing“ der Treffer, falls mehrere Zeilen nach Treffern durchsucht werden, gemessen wurde, sowie aus den sechs Bit der TDC-Zeiten zusammen. Die Kodierung des „Bunch-Crossing“ ist dabei für das erste „00“, für das zweite „01“ und für das dritte „10“. Falls kein Treffer gefunden wurde, wird dies mit „11“ signalisiert. Damit sind TDC-Zeiten mit Werten von 0 bis 191 möglich, 192^{10} bedeutet kein Treffer. Die für diese Diplomarbeit wichtigen Informationen sind die OTIS-ID sowie die Zeitinformationen.

[9] Für den Otis 1.2 wurde ein weiterer Betriebsmodus eingeführt, der die Daten in einem anderen Format zusammenstellt.

[10] hexadezimal = c0

Der Datensatz eines OTIS, der keine Treffer registriert hat, sieht folgendermassen aus:[11]

```
OTIS ID   Statusbits  Event-ID    BX-ID
0a2       b0          5           e8
Driftzeiten:
c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0
c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0
```

Diese werden nach einem Trigger als 8 Bit-Daten gesendet:

```
... xx xx xx (ff) 0a 2b 05 e8 c0 c0 ... c0 c0 c0 xx xx xx  ...
    xx = unbestimmt (= 00 wenn das Komma genutzt wird)
```

Das „ff“ vor den eigentlichen Datenbytes wird Komma genannt. Dies wurde in der OTIS Version 1.1 als Option eingeführt, um den Start der Daten nach einer Übertragungspause zusätzlich kenntlich zu machen.

1.6 GOL-Aux/O-RxCard

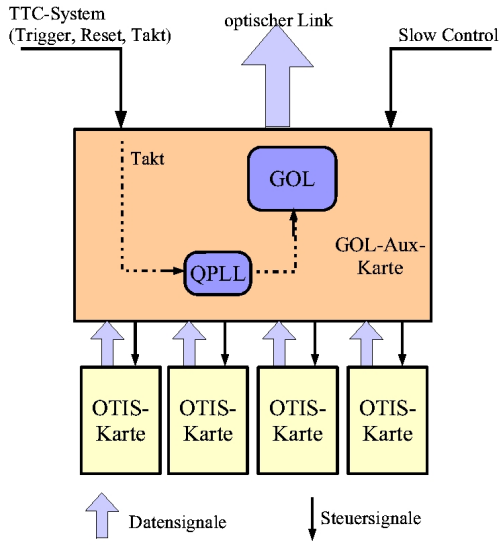

Abbildung 1.10: Die optische Senderkarte im schematisch Überblick

Zum Senden der Daten der OTIS-Chips wurden ebenfalls am Physikalischen Institut der Universität Heidelberg eine optische Senderkarte (GOL-Aux-Karte [7]) sowie eine optische Empfängerkarte (O-RxCard [10]) entwickelt. Abbildung 1.10 gibt einen schematischen Überblick der optischen Senderkarte.[12] Deren wichtigste Komponenten sind der Gigabit Optical Link (GOL) Chip [8], sowie der Quarz crystal based Phase-Locked-Loop (QPLL) Chip [9]. Die QPLL ist ein Chip, der als „Jitter“-Filter für das Taktsignal fungiert. Dies ist notwendig, da die Taktflanke, die vom TTC-System geliefert wird, nicht stabil genug für die Verwendung im GOL-Chip ist. Beide Chips wurden am CERN entwickelt und sind als strahlenhart qualifiziert. Die Aufgabe des GOL-Chips besteht darin, mit 40 MHz 32-Bit-Daten entgegen zunehmen, sie in optische Signale umzuwandeln und mit 1,6 GBit/s zu versenden. Dabei steuert jeder der vier angeschlossenen OTIS-Chips 8 Bit pro Taktzyklus bei.

Die optische Senderkarte bildet auch die Schnittstelle zum TTC-System sowie zur Slow-Control. Die Signale dieser Systeme werden auch an die OTIS- und die Vorverstärkerkarten weitergegeben. Sie regelt weiterhin die Spannungversorgung für diese Karten.

[11] Beispiel für OTIS 1.1, alle Daten sind hexadezimal.

[12] Im September 2004 wurde eine neue Version der Karte entwickelt, in der diverse Verbesserungen integriert wurden.

Die gesendeten optischen Daten werden von der optischen Empfängerkarte (Optical Receiver Card (O-RxCard)) entgegengenommen. Dabei können zwölf optische Links auslesen werden. Aus den empfangenen Daten wird der Takt zurückgewonnen sowie einige Statusbits gesetzt. Weiterhin wandelt sie die optischen Daten in elektrische Signale um. Sie liefert pro Link folgende Signale:

RxClk : Die Taktrate, mit der die Daten übertragen werden.

RxData(15:0) : Die Daten von zwei OTIS-Chips.

RxDV : Zeigt an, ob die Daten gültig sind.

RxErr : Zeigt an, ob während der Übertragung Fehler aufgetreten sind.

Die Daten werden mit 80 MHz an die weitere Elektronik gesendet und dabei auf folgende Weise zusammengestellt:

```
Daten der vier OTIS-Chips eines Links am Ausgang der TDCs:

        Takt 40 MHz : 1         | 2         | 3
             OTIS A : a1        | a2        | a3
             OTIS B : b1        | b2        | b3
             OTIS C : c1        | c2        | c3
             OTIS D : d1        | d2        | d3

werden zu

    ORxCard Ausgang : a1b1 c1d1 | a2b2 c2d2 | a3b3 c3d3
        Takt 80 MHz : 1    2    | 3    4    | 5    6
```

Die Nutzdaten enthalten also im oberen Byte immer abwechselnd die Daten von den OTIS-Chips A und C, im unteren immer die Daten von B und D. Somit lassen sich sehr einfach die Daten der einzelnen OTIS-Chips wieder rekonstruieren.

Für die lokalen Aufbauten standen beide Karten zur Verfügung.

1.7 Die TELL1-Karte

Das „Trigger ELectronics and L1 board“ (TELL1) [12] nimmt die Daten von fast allen Subdetektoren im LHCb-Experiment entgegen. Es wurde im Rahmen einer Doktorarbeit [13] an der Ecole Polytechnique Fédérale de Lausanne (EPFL) unter Beteiligung der Heidelberger LHCb-Gruppe entwickelt.

Für die Datennahme des Outer Tracker werden zwei optische Empfängerkarten auf ein TELL1 gesteckt Die Daten aus den verschiedenen Links werden zunächst in vier PreProcessor-FPGAs[13] (PP-FPGA) synchronisiert. (Abbildung 1.11) Dabei kommen die

[13]FPGA = Field Programable Gate Arrays

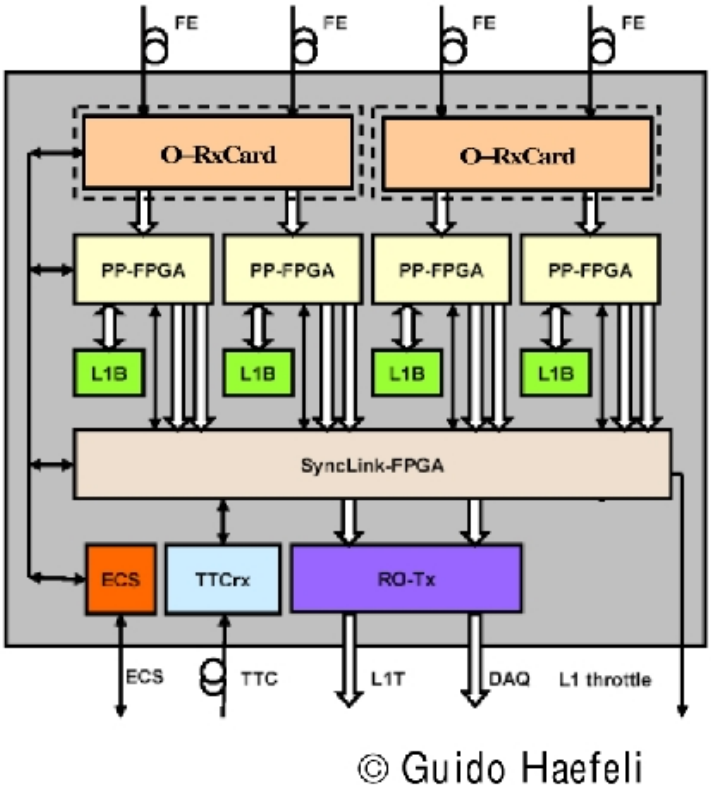

Abbildung 1.11: Datenfluss auf der TELL1-Karte

Stratix^{TM}EP1S20F780-7 FPGA von Altera zum Einsatz. Anschliessend werden Daten, die zur selben Wechselwirkung gehören, zusammengefügt, nullunterdrückt, umformatiert und im L1-Buffer (L1B) gespeichert. Dieser besteht jeweils aus einem DDR-RAM mit 256 Mb. Abbildung 1.12 auf der nächsten Seite zeigt ein Foto einer vollbestückten TELL1-Karte.

Falls die L1-Triggerentscheidung auf den Daten des äusseren Spurkammersystems basiert, werden sie nach der Synchronisation komprimiert und an den SyncLink FPGA[14] geschickt. Dieser leitet sie über die RO-TxCard an das L1/HLT Netzwerk weiter.

Für den Aufbau des lokalen Datenahmesystems wurde eine andere Karte eingesetzt, da die TELL1-Karte zu diesem Zeitpunkt noch in der Entwicklung war. Diese Karte benutzte einen Altera Stratix^{TM}EP1S25F1020-5 FPGA, für den ein Programm entwickelt wurde, welches im folgenden Kapitel vorgestellt wird.

[14]Ebenfalls ein Stratix^{TM}-FPGA in der Version EP1S25F1020-7.

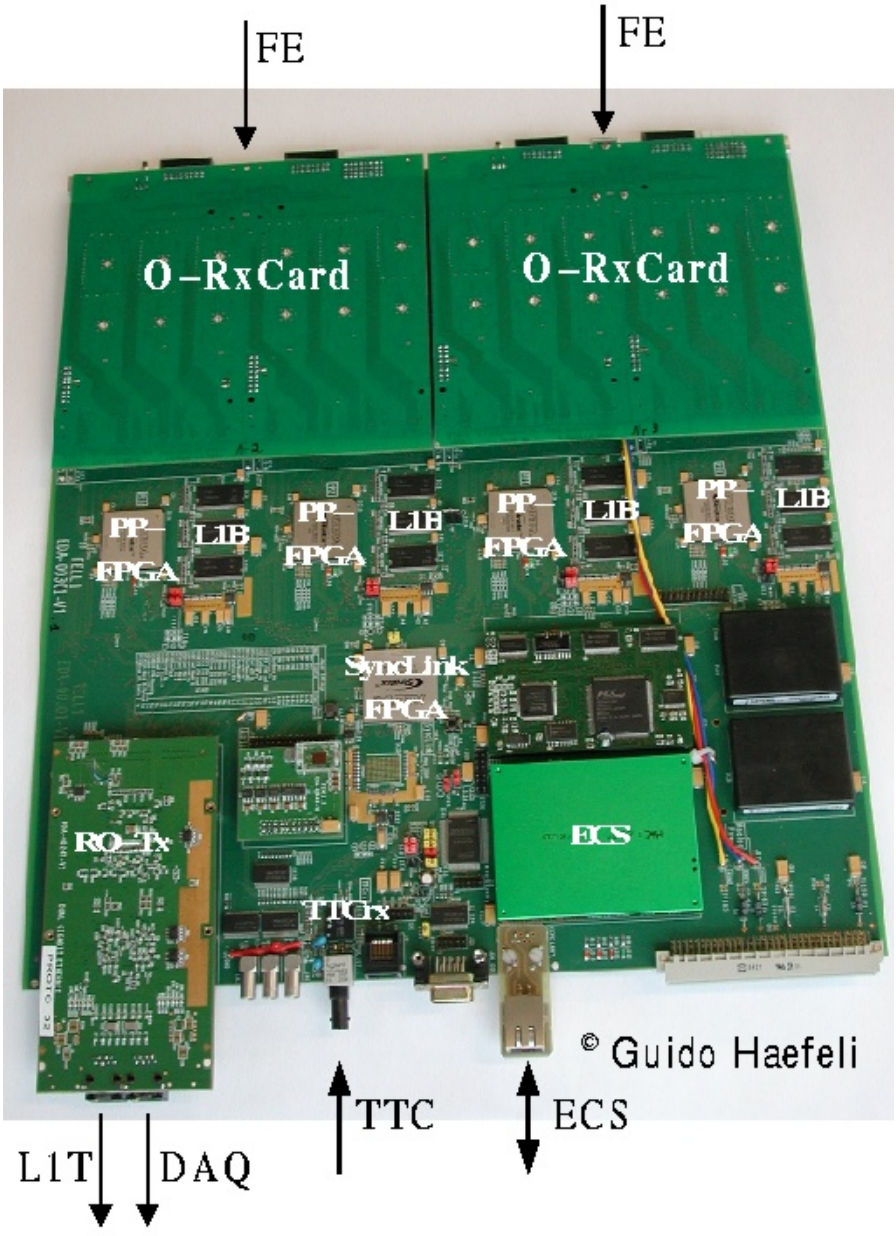

Abbildung 1.12: Die TELL1-Karte (vollständig bestückt).

2 TELL1-Emulation

Abbildung 2.1: Die Stratix-Karte

Da die TELL1-Karte einerseits noch nicht zur Verfügung stand, andererseits für einfache Funktionstests der Auslesekette zu komplex ist, wurde beschlossen, einen einfacheren Ersatzaufbau zu entwerfen. Dazu wurde ein „**PCI-Development-Kit**, Stratix Edition“ von Altera [25] erworben. Dieses besteht aus einer PCI-Steckkarte mit dem Stratix-Chip-EP1S25, also derselben Familie wie der Preprozessor auf der TELL1-Karte. Abbildung 2.1 zeigt ein Foto der Karte.

2.1 Überblick

Eine Hauptaufgabe dieser Diplomarbeit bestand darin, für die Stratix-Karte ein Programm zu entwickeln, welches die benötigten Funktionen des TELL1-Preprozessor-FPGA für die restliche Auslesekette „emuliert“. Dies impliziert folgende Anforderungen an das Programm:

- Deserialisierte Daten von der optischen Empfängerkarte entgegennehmen.
- Im Datenstrom nach bekannten Bitmustern suchen, um eine Synchronisation zu erreichen. Da der OTIS-Chip nur nach einem Trigger seine Daten sendet, müssen diese im Datenstrom wiedergefunden werden.
- Die synchronisierten Daten in einem Fifo zwischenspeichern.
- Die Auslese des Fifos über den PCI-Bus ermöglichen.

Diese Anforderungen übersetzen sich auf natürliche Weise in eigene Blöcke im VHDL[1]-Code, die in den folgenden Unterkapiteln jeweils vorgestellt werden. Die beiden letzten Punkte[2] werden für die lokale Datennahme benötigt und kommen im Programm für die Preprozessoren des TELL1 so nicht vor. Im Verlaufe der Entwicklung wurden zusätzliche

[1] Very High Speed Integrated Circuit Hardware Description Language
[2] Fifo und PCI-Bus

Blöcke eingebaut, um Messungen sowie Aufbauten zu vereinfachen. Sie haben aber keinen prinzipiellen Einfluss auf den Ablauf der Datenaufnahme. Die Abbildung 2.2 auf der nächsten Seite gibt eine Übersicht über den Datenfluss (durch das Programm), Abbildung 2.3 auf Seite 19 zeigt den schematischen Aufbau des Programms. Abbildung 2.4 auf Seite 20 schliesslich zeigt die oberste Programmebene der Implementierung.[3]

Erstellt wurde das Programm mit demselben Programmpaket, das auch bei der Entwicklung der Codes für die FPGAs der TELL1-Karte zum Einsatz kam. Als erstes Programm kommt dabei *FPGA Advantage 6.2*^{TM} zum Einsatz. Mit diesem Werkzeug werden auf einer grafischen Oberfläche die einzelnen Blöcke erstellt, die die Anforderungen in eine Logik umsetzen. Aus diesen Blöcken werden dann von FPGA Advantage die zugehörigen VHDL-Dateien generiert, die eine Beschreibungssprache der gewünschten Hardware darstellt. Anschliessend wurde mit *ModelSim 5.7f*^{TM} das Programm simuliert und nötige Änderungen vorgenommen.

War diese Phase abgeschlossen, so wurde mittels *LeonardoSpectrum 2003b*^{TM} eine Netzliste erstellt. Diese ersten drei Programme bilden zusammen das Programmpaket *HDL Designer 2003.2*^{TM}, welches von Mentor Graphics angeboten wird. Zuletzt wurde diese Netzliste mit *Quartus II 4.0*^{TM}*SP1* von Altera für den Stratix-Chip kompiliert und auf die Karte aufgespielt. Als Name für das Design wurde „Shippo“[4] gewählt.

Um die Versionsverwaltung zu vereinfachen, wurde ab der Version $\beta 5$ ein Register für Versionsnummern eingeführt.[5] Alle Angaben sowie Messungen in dieser Diplomarbeit beziehen sich auf die Versionsnummer 4. Tabelle 2.1 listet alle bis zum 1. September 2004 erstellten Versionen auf.

Versionsnummer	Name
4	Shippo v1.01
3	Shippo v1.00
2	Shippo v1.00 RC 1
1	Shippo $\beta 5$
ohne Nummern	alle vorherigen Versionen.

Tabelle 2.1: Programmversionen

Anmerkung 1: Im gesamten Programm werden insgesamt vier verschiedene Taktbereiche[6] verwendet, auf die im folgenden immer wieder Bezug genommen wird. Im Einzelnen sind dies:

LHCb_80_clk : Die normale Frequenz dieses Taktes beträgt ca. 80 MHz.[7] Dieser Takt sollte jene Frequenz besitzen, aus der alle anderen im Aufbau befindlichen Tak-

[3] Einzelne Blöcke wurden der Übersicht halber weggelassen.
[4] Schnelles Hochenergiephysik Interface und Preprocessing Programm für den Outer-Tracker
[5] Siehe auch Anhang A.4 auf Seite 69
[6] engl. Clock Domain
[7] Genauer die doppelte LHCb-Frequenz

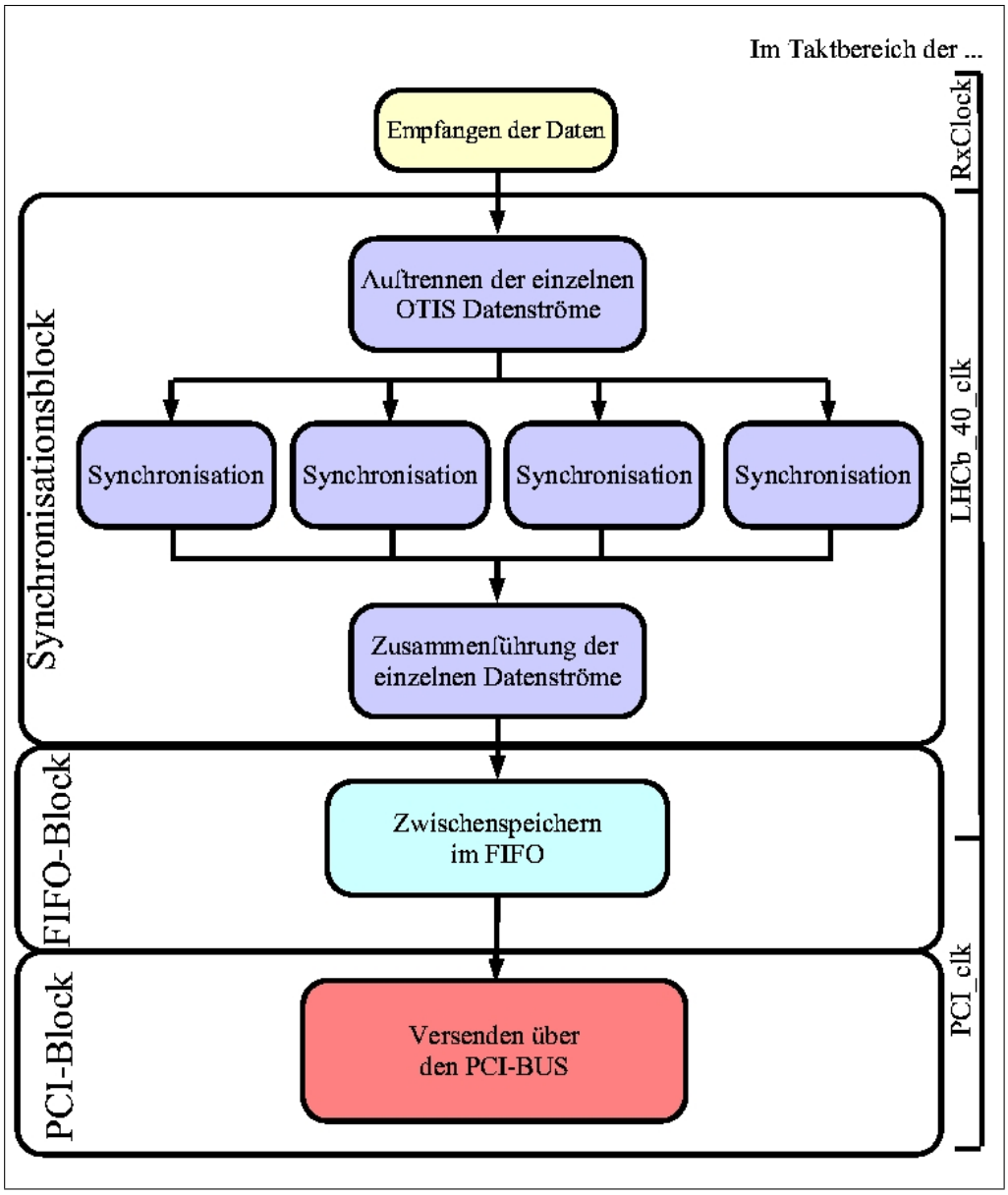

Abbildung 2.2: Datenfluss durch das Shippo-Programm

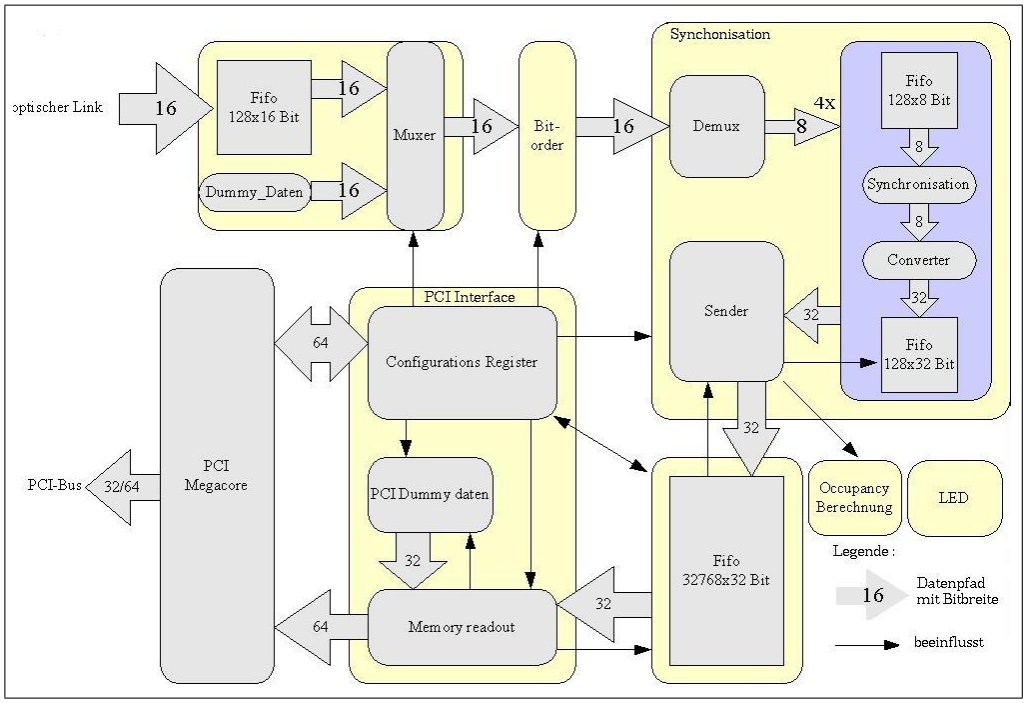

Abbildung 2.3: Schematischer Aufbau des Programms

te abgeleitet werden. Sie muss über den *User Clock Pin* auf der Stratix-Karte eingespeist werden.

LHCb_40_clk : Dieser Takt wird aus der LHCb_80_clk abgeleitet und läuft mit der halben Frequenz. Sie wird sowohl im Synchronisationsblock als auch im Fifo verwendet.

Receive clock : (im folgenden RxClk) Mit diesem Takt werden die Daten von der optischen Empfängerkarte entgegengenommen. Sie sollte mit derselben Frequenz wie die LHCb_80_clk laufen, um eine korrekte Funktionsweise zu gewährleisten. Laufen die Takte mit unterschiedlichen Frequenzen, so kann es in Abhängigkeit von Triggerrate und Frequenzunterschied zum Verlust einzelner Datenbytes kommen.[8]

PCI clock : Sie wird dem PCI-Bus entnommen. Es sind Frequenzen von 33 MHz als auch von 66 MHz erlaubt. Der Takt wird im PCI-Block verwendet.

Die ersten drei Taktsignale werden im Experiment aus dem Taktsignal des LHC-Beschleunigers abgeleitet, der über das Timing-and-Trigger-Control-System (TTC) zur Verfügung gestellt wird.

[8] Dies wird nicht kontrolliert und kann somit auch nicht abgefangen werden

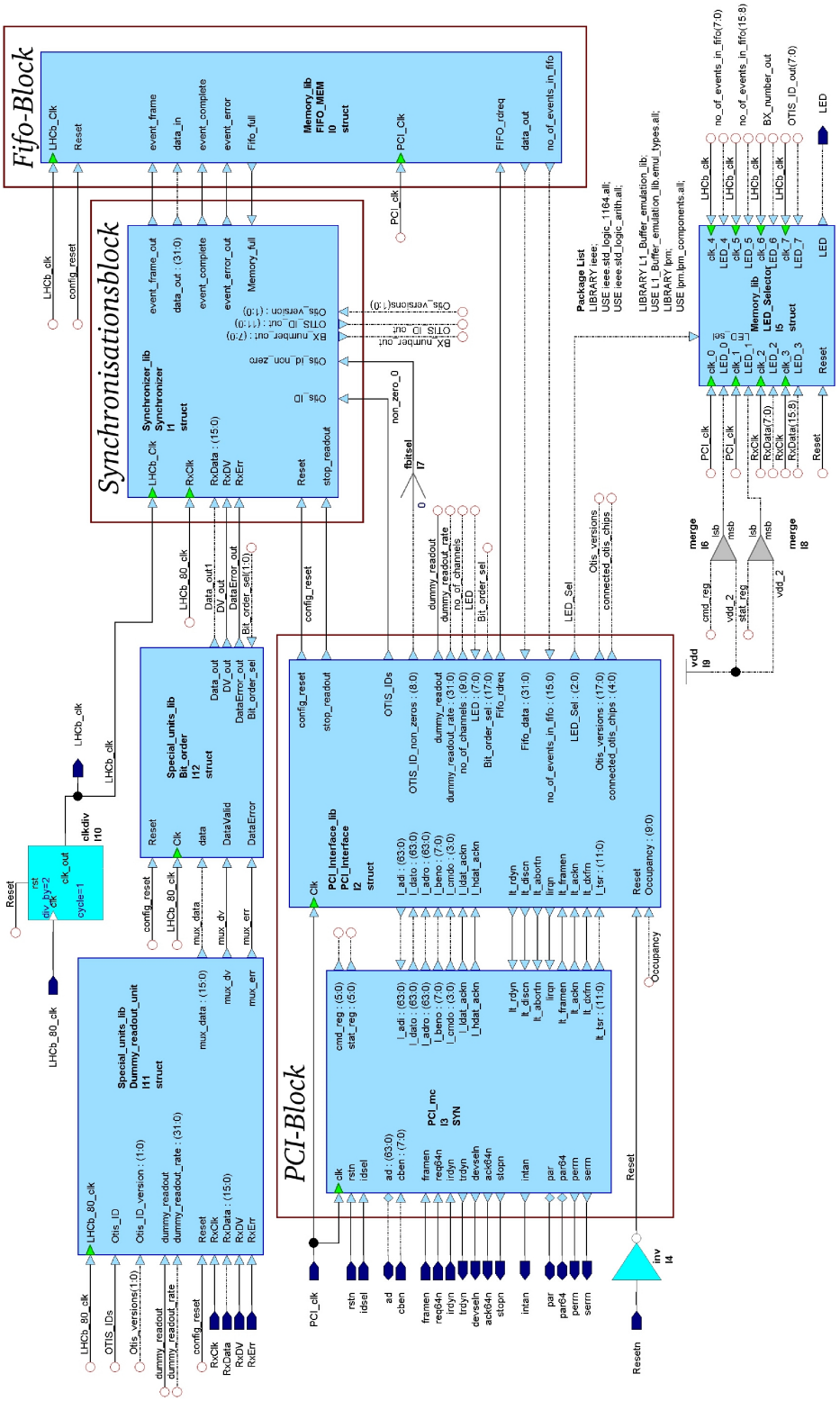

Abbildung 2.4: Oberste Programmebene des gesamten Projekts

2.2 Die Synchronisation

Der erste Hauptblock des Programms ist der Synchronisationsblock. Seine Aufgabe besteht in der Entgegennahme der Daten von der optischen Empfängekarte. Anschliessend soll er aus dem kontinuierlichem Datenstrom die Daten der angeschlossenen OTIS-Chips extrahieren. Dies geschieht in mehreren Stufen.

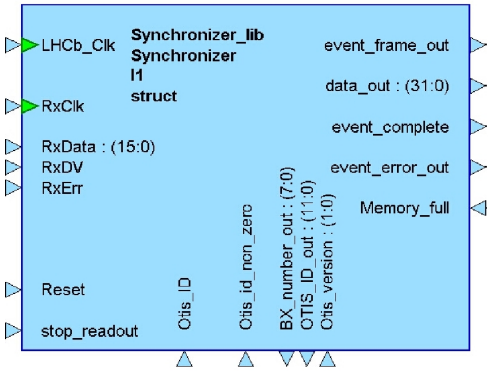

Abbildung 2.5: Der Synchronisationsblock

Zunächst erfolgt ein Umformatieren des Datenformats, um den Synchronisationsablauf zu vereinfachen. Die Daten werden als 16 Bit-Pakete mit einer Frequenz von 80 MHz empfangen. Diese beinhalten die Daten von vier OTIS-Chips in dem Format, das in Kapitel 1.6 vorgestellt wurde. Die Daten der einzelnen OTIS-Chips werden voneinader getrennt und als unabhängige Datenströme behandelt. Diese Datenströme werden nach einem Wechsel des Taktes mit 40 MHz weiterverarbeitet.

Für jeden dieser vier Ströme findet nun die eigentliche Synchronisation statt. Dabei gibt es zwei unterschiedliche Modi. In einem einfacheren Modus werden die ersten Daten genommen, die einen Wert ungleich Null haben. Dieser Modus hat sich als sehr hilfreich bei der Fehlersuche in den später beschriebenen Messungen erwiesen. Im folgenden wird er mit „non_zero_Modus“ bezeichnet.

Der zweite Modus ist der eigentliche Betriebsmodus. Gesucht wird hier nach den IDs der angeschlossenen OTIS-Chips. Es werden also die Daten genommen, die mit den in den entsprechenden Registern eingestellten Werten übereinstimmen.

Unabhängig davon, welcher Modus aktiv ist, werden nach einer erfolgreichen Synchronisation die folgenden 34 Byte als Teil eines Ereignisses angesehen und weitergeleitet. Dabei werden noch einige Informationen wie BX-ID und Fehlerbit aus dem Datenstrom extrahiert. Nach diesen 34 Byte wird wieder geprüft, ob die nächsten anliegenden Daten von einem OTIS-Chip stammen. Ein OTIS-Datensatz besteht damit aus 36 Byte.

Damit die Synchronisation erfolgreich sein kann, müssen zwei weitere Bedingungen erfüllt werden. Zum einen muss das „Data_Valid-Signal“ der optischen Empfängerkarte aktiv sein. Dieses Signal zeigt an, dass die Daten, die von der Auslesekette kommen, gültig sind. Zum anderen muss das nachfolgende Fifo noch genügend Speicherplatz für weitere Daten haben. Ist dies nicht mehr der Fall, so muss erst gewartet werden, bis wieder Platz im Fifo vorhanden ist, bevor weitere Daten entgegengenommen werden können. Dies ist notwendig, um ein Überlaufen des Fifos zu verhindern.[9]

Welcher Modus benutzt wird, kann über ein Register festgelegt werden. Da sich das Format der gesendeten Daten für die verschiedenen OTIS-Versionen unterscheidet, muss

[9] In der TELL1-Karte werden die Daten in einem DDR-RAM gespeichert. Dabei wird mit spezieller Hardware dafür gesorgt, dass bei der Datennahme keine Speicherprobleme auftreten können.

dies natürlich auch bei der Synchronisation berücksichtigt werden. Insgesamt ergeben sich folgende Fälle, die eine Synchronisation herstellen (Alle anderen Kombinationen führen zu *keiner* Synchronisation):

OTIS-Version	Non-zero-Modus an?	empfangene Daten	
		ersten 8 Bit	nächsten 12 Bit
1.0	ja	—	= OTIS-ID
1.0	nein	= x"00"	≠ x"000"
1.1	ja	= x"FF"	= OTIS-ID
1.1	nein	= x"FF"	—

Tabelle 2.2: Bedingungen für eine Synchronisation

Abbildung 2.6 zeigt den Teil des Zustandsautomaten für die Synchronisation.[10]

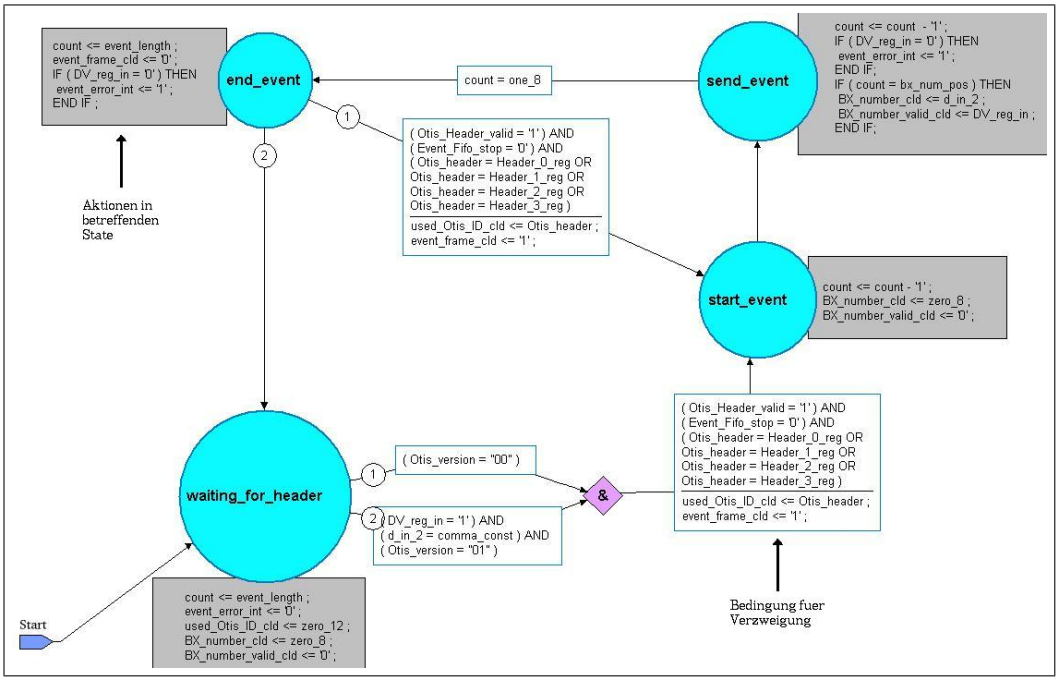

Abbildung 2.6: Zustandsautomat zur Synchronisation (Ausschnitt)

Im Anschluss an die Synchronisation werden die Daten eines OTIS-Chip von 36 x 8 Bit auf 9 x 32 Bit umformatiert. Diese 9 x 32 Bit werden dann in einem Fifo zwischengespeichert. Ein weiterer Zustandsautomat fragt nacheinader jeden der vier Ströme ab, ob Daten gefunden wurden. Ist dies der Fall, so wird dieses Ereignis in das Haupt-Fifo übertragen. Dabei entstehen keine Totzeiten, die den Datenstrom unterbrechen könnten.

[10] Für den normalen Betriebsmodus.

Der Zustandsautomat zeigt auf einem speziellen Signal Fehler an, die bei der Datenübertragung aufgetreten sind. Weiterhin werden auf diesem Signal interne Fehler angezeigt. Die einzelnen Ein- und Ausgangssignale des Synchronisationsblock sind in Tabelle A.1 auf Seite 65 im Anhang A.1 beschrieben.

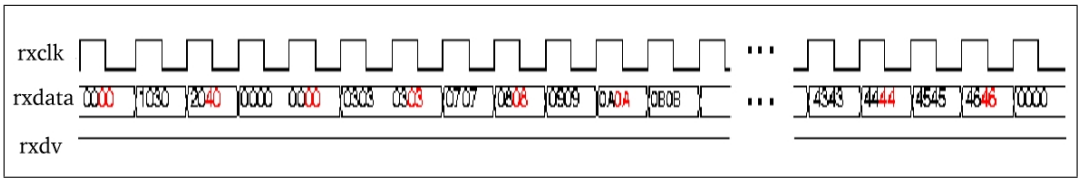

Abbildung 2.7: Die Daten, wie sie von der optischen Empfängerkarte geschickt werden (Simulation). Die Daten des OTIS mit der ID 4 sind farblich hervorgehoben. Bei den Datensignalen ist der hexadezimale Wert angegeben

Die folgenden Abbildungen zeigen eine logische Simulation des Synchronisationsblocks. Abbildung 2.7 zeigt die Daten, wie sie von der optischen Empfängekarte gesendet werden könnten. In dem gewählten Beispiel sind dies Daten von den OTIS-Chips mit den IDs 1 bis 4. Die Daten der Chips 1 und 3 bestehen aus den Werten 07, 09, 0B, usw., die Daten der Chips 2 und 4 aus den Werten 8 bis 46.[11] Die Daten des OTIS-Chips 4 sind dabei farblich hervorgehoben.

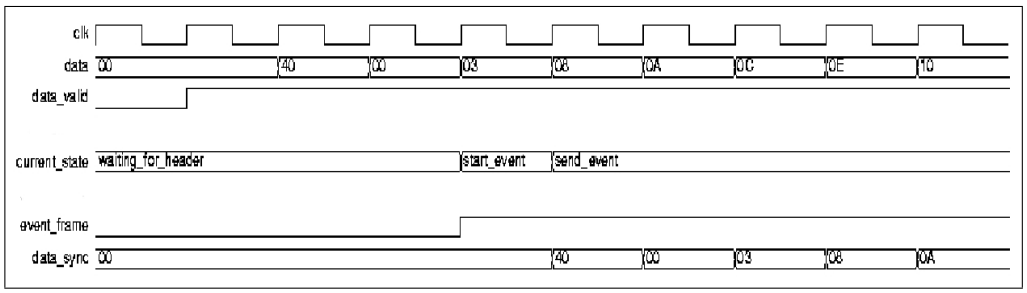

Abbildung 2.8: Start der Synchronisation (Simulation). Die Zustände des Zustandsautomaten sind mit ihrem Namen angegeben.

Nachdem die Daten der einzelnen Chips aufgetrennt wurden, gelangen sie zur Synchronisation zu den entsprechenden Zustandsautomaten. Abbildung 2.8 zeigt den Beginn der Synchronisation. In Takt zwei startet das Ereignis im Datenstrom. Sichtbar wird das durch das Aktivieren von „data_valid“. Dies muss nicht unbedingt zu Beginn eines Ereignisses erfolgen, da das Signal nur anzeigt, dass die Daten auf dem „data“-Bus gültig sind.

Im vierten Takt liegt am Zustandsautomaten die bekannte OTIS ID „004“ an und wechselt daraufhin im nächsten Takt in den Zustand „start_event“. Hier werden auch

[11] Alle Werte sind hexadezimal angegeben.

die Daten auf die Ausgänge durchgeschaltet. Das ein Event erkannt wurde, wird durch das Schalten eines Rahmensignals („event_frame“) signalisiert. Danach wird im nächsten Takt in den Zustand „send_event“ gewechselt.

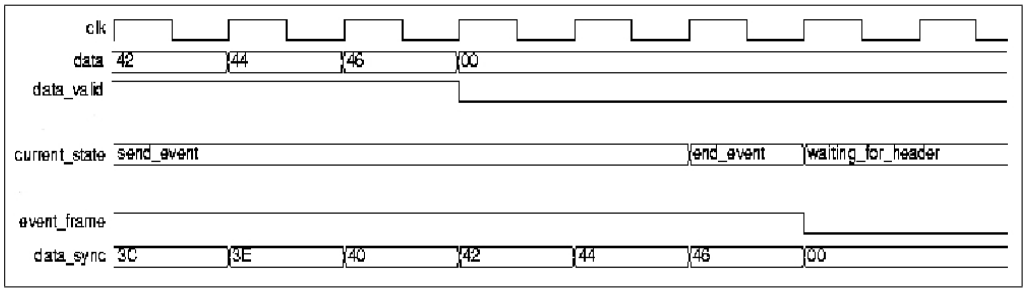

Abbildung 2.9: Ende der Synchronisation (Simulation)

Nachdem ein Datensatz übertragen wurde, zeigt Abbildung 2.9 die Beendigung der Synchronisation. Mit der Ausgabe des vorletzten Datenbytes wird der Zustand „end_event“ angenommen. Da keine weiteren Daten folgen, wird im nächsten Takt das Rahmensignal auf inaktiv geschaltet und im Zustand „waiting_for_header“ auf die nächsten Daten gewartet.

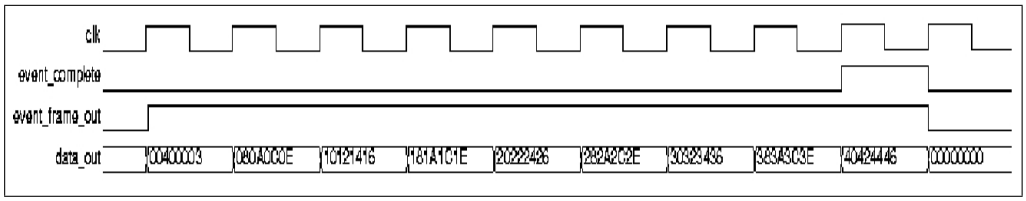

Abbildung 2.10: Ausgabe des Synchronisationsblock (Simulation)

Um den Datensatz im Fifo abzulegen, wird dieser wie schon erwähnt in 9 x 32 Bit umgewandelt. Abbildung 2.10 zeigt das Ereignis, wie es aus dem Synchronisationsblock ausgeben wird. „event_complete“ markiert dabei das letzte Datenwort des Datensatzes.

Diese Simulation wurde nach dem Aufspielen des Programms auf den FPGA-Chip mit echten Daten nachempfunden. Die aufgenommen Signale zeigen dabei exakt dasselbe Verhalten wie in der logischen Simulation.[12] Die Abbildung 2.11 zeigt dabei das „current_state“-Signal aus der Simulation und gibt die Abfolge der Zustände wieder. Die nächste Abbildung (2.12) zeigt die Signale „data_valid“ und „event_frame“. In der letzten Abbildung werden die Daten ins Fifo übertragen. Dabei werden die Signale „event_frame_out“ und „event_complete“ angezeigt.

[12]Dies sollte zwar immer der Fall sein, ist aber keineswegs selbstverständlich.

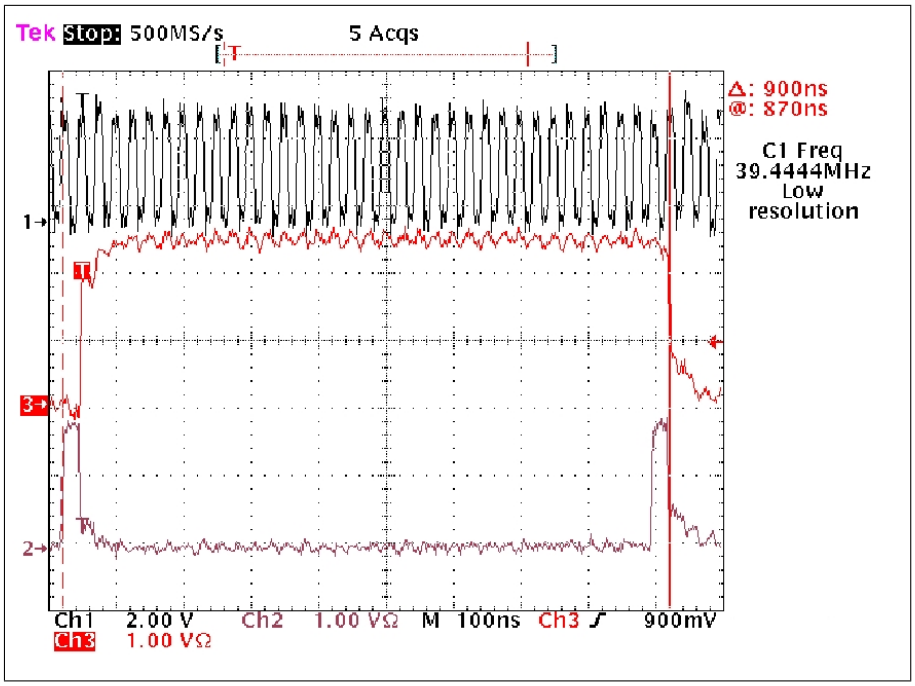

Abbildung 2.11: Die Zustände des Zustandsautomaten für die Synchronisation während eines Ereignisses.
Kanal 1 (oben) zeigt den 40 MHz-LHCb-Takt.
Kanal 3 (mitte) zeigt das Zustandsbit 1.
Kanal 2 (unten) zeigt das Zustandsbit 0.
Die Kodierung der Zustände ist dabei: 00 = idle, 01 = start_event, 10 = send_event und 11 = end_event
Zu Beginn wird vom Zustand „idle“ in den “start_event“-Zustand gewechselt (Kanal 2 auf oberen Pegel). Im nächsten Takt gelangt der Zustandsautomat in den Zustand send_event (Kanal 2 auf unteren, Kanal 3 auf oberen Pegel). Beendet wird das Ereignis durch den Zustand end_event (Kanal 2 auf oberen Pegel) und die Rückkehr in den Wartezustand (beide Kanäle auf unterem Pegel).

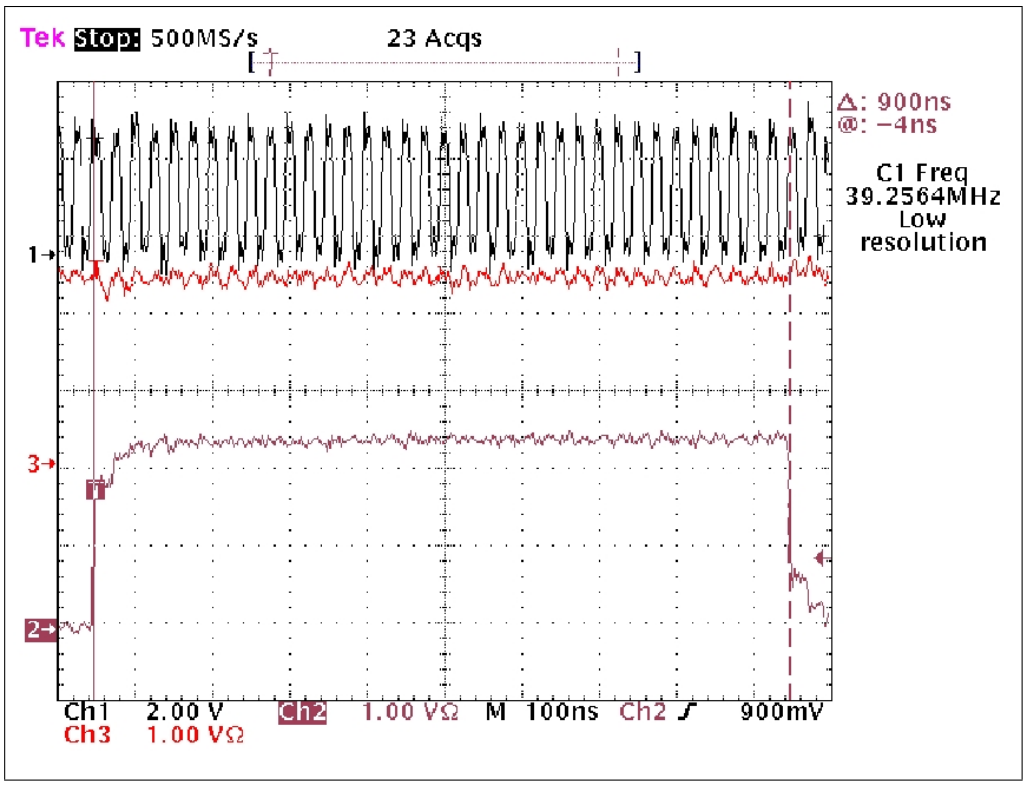

Abbildung 2.12: Die Signale, die am Zustandsautomaten für die Synchronisation während eines Ereignisses anliegen.
Kanal 1 (oben) zeigt den 40 MHz-LHCb-Takt.
Kanal 3 (mitte) zeigt das Data-Valid-Signal (in diesem Fall immer aktiv).
Kanal 2 (unten) zeigt das Rahmensignal, das einen Datensatz umschliesst.
Die wichtige Information dieser Messung ist die Zeit, die das Rahmensignal auf dem oberen Pegel liegt. Die gemessenen 900 ns entsprechen genau der Länge eines Ereignisses.

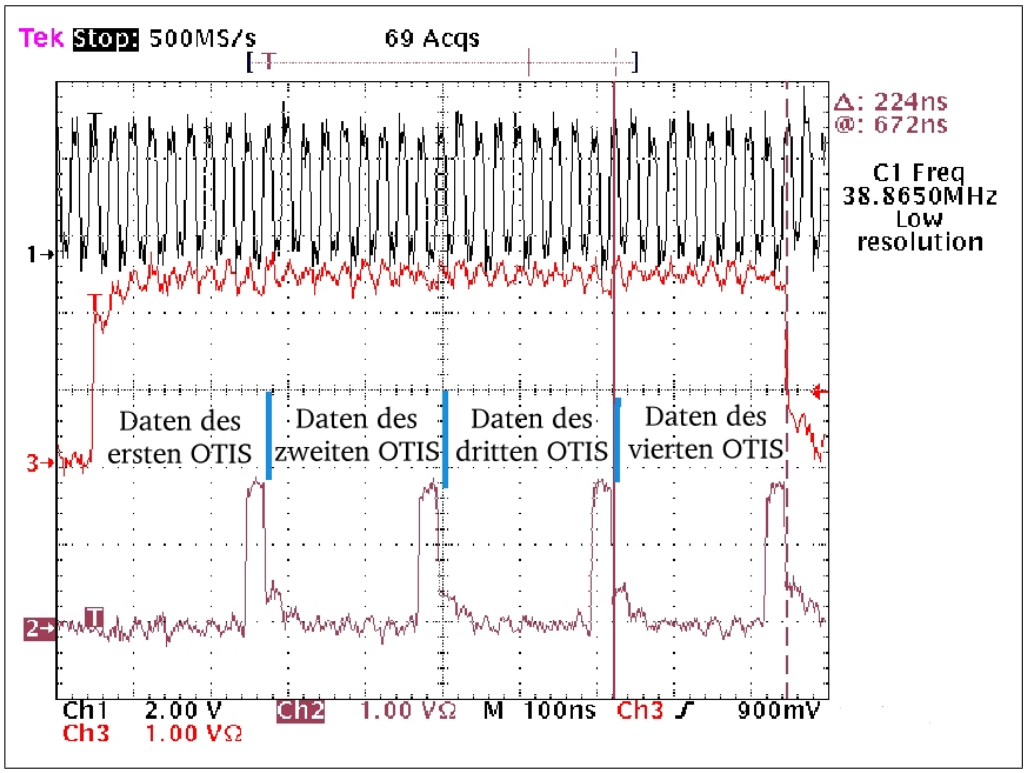

Abbildung 2.13: Die Signale während der Übertragung eines Ereignisses ins Fifo.
Kanal 1 (oben) zeigt den 40 MHz-LHCb-Takt.
Kanal 3 (mitte) zeigt das Rahmensignal für synchronisierte Ereignisse.
Kanal 2 (unten) zeigt das „event_complete“-Signal, dass das letzte Datenwort eines Ereignisses markiert.
In diesem Bild werden die Ereignisse von den vier OTIS-Chips sequentiell ins Fifo geschrieben. Insgesamt dauert dies genau 900 ns, die Übertragung eines OTIS-Datensatz dauert 225 ns. Dies liegt daran, das im Gegensatz zu vorherigen Abbildung die Ereignisse eine Breite von 32 Bit haben.

2.3 Der Fifo-Block

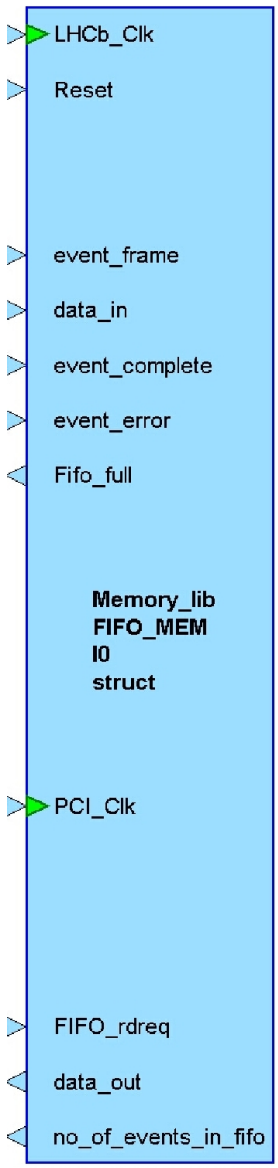

Abbildung 2.14: Der Fifo-Block

Der zweite grosse Block im Programm ist das Fifo. Es dient im Wesentlichen als Zwischenspeicher für die vom Synchronisationsblock gesendeten Daten. Im Gegensatz zu den Preprozessoren des TELL1 wird kein externer Speicher benutzt, sondern die auf FPGA integrierten Speicherblöcke verwendet. Dies führt zu einer grossen Vereinfachung der Abläufe. Es wird weiterhin ein Taktwechsel auf den Takt des PCI-Busses durchgeführt.

Die Daten werden dabei aus dem Synchronisationsblock entgegengenommen und anschliessend in ein 32 768 x 32 Bit grosses Fifo geschrieben. Dabei wird das zwölfte Bit im OTIS-Header mit dem Wert des Fehlersignals aus dem Synchronisationsblock beschrieben. Dieses Bit kann bedenkenlos verwendet werden, da es bisher keine Aufgabe erfüllt.[13] Als Schreibbefehl für das Fifo dient dabei das Rahmensignal.

Sobald mehr als 32 400 Datenworte (= 3 600 Datensätze) im Fifo gespeichert sind, werden die Zustandsautomaten im Synchronisationsblock angewiesen, ausser den aktuellen Daten keine weiteren mehr zu senden. Dadurch wird der Vorinstanz ermöglicht, gerade prozessierte Ereignisse noch abzuschliessen. Danach dürfen aber keine weiteren angefangen werden, solange das Fifo keine gegenteiligen Anweisungen sendet.

Die Anzahl der gespeicherten Ereignisse kann vom PCI-Block über Register abgefragt werden. Ausgelesen wird das Fifo auf Anfrage des PCI-Blocks. Dabei sollte sichergestellt werden, dass auch Daten im Fifo gespeichert sind, da sonst die Datenausgänge in einen undefinierten Zustand geraten können.[14] Eine genaue Auflistung aller Ein- und Ausgänge befindet sich im Anhang A.2 auf Seite 66.

[13] Beim OTIS 1.0 ist dieses Bit fest auf 0, beim OTIS 1.1/ 1.2 fest auf 1 gesetzt.

[14] Dies wird durch einen Fehler in der von Altera gelieferten Implementierung der Fifos ausgelöst. Er ist in den neuen Version der Quartus-Software beseitigt worden.

2.4 Die PCI-Schnittstelle

Die dritte Einheit im Programm dient der Kommunikation mit dem PCI-Bus und ist der komplexeste. Deshalb folgt zunächst eine allgemeine, vereinfachte Einführung in die Funktionsweise des PCI-Busses, bevor die eigentliche Implementierung vorgestellt wird.

Den PCI-Bus gibt es in unterschiedlichen Ausführungen. In dem meisten Fällen wird er mit einer Taktrate von 33 MHz und einer Busbreite von 32 Bit in Desktop-PCs eingesetzt. Es gibt davon abweichende Spezifikationen, die z.B. mit 66 MHz arbeiten oder eine Busbreite von 64 Bit besitzen.

Alle Spezifikationen verfügen über eine Arbitrierungslogik. Sie stellt den Verwalter des PCI-Busses dar und entscheidet zum Beispiel, wer als nächstes den Bus benutzen darf. Beim Hochfahren des Computers muss diese Logik zunächst einmal feststellen, wer alles an den Bus angeschlossen ist. Dazu fragt sie nacheinader die einzelnen Schnittstellen ab[15], wieviel Adressraum sie benötigen und wie oft dieser unterteilt ist.

Als einfaches Beispiel soll ein PCI-Bus mit zwei angeschlossenen Karten betrachtet werden. Dabei soll die erste Karte zwei Adressräume belegen. Der erste sei z.B. ein Speicher, der nur ausgelesen werden kann[16] und eine Grösse von 100 kB habe. Der zweite Adressraum soll mit einem beschreibbaren Speicher[17] belegt werden bei einer Grösse von 400 kB. Die zweite Karte soll schliesslich einen Speicher mit 500 kB besitzen.

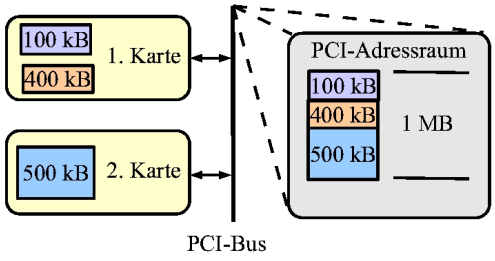

Abbildung 2.15: Einfaches PCI-Bus Beispiel. Die Adressräume der Karten werden auf einem grossen Adressraum abgebildet.

Damit ergibt sich für die Arbitrierungslogik ein Adressraum von 1 MB. Wo die einzelnen Adressräume der Karten in diesem grossen Adressraum liegen, wird den Karten über sogenannte Basisadressen-Register[18] (BAR) mitgeteilt. In dieses Register schreibt die Arbitrierungslogik den Beginn des jeweiligen Adressraums der Karten. In diesem Beispiel wäre dies für den ROM-Speicher der ersten Karte die Adresse 0, für das RAM dieser Karte die Adresse 25600 und für den Speicher der zweiten Karte die Adresse 128000.[19]

Soll die Adresse 400 im Adressraum des RAM-Speichers der ersten Karte angesprochen werden, so muss auf den PCI-Bus die Adresse 25600 + 400 = 26000 anlegt werden (analog ergibt sich die Adresse 128400 für die zweite Karte). Da jede Karte weiss, wo ihr entsprechender Adressraum beginnt und wie gross dieser ist, kann sie automatisch erkennen, ob die Transaktion für sie bestimmt ist.

[15] in Form von z.B. PCI-Slots auf dem Motherboard
[16] Ein sogenanntes read only memory (ROM)
[17] Ein random access memory (RAM)
[18] engl. „Base Adress Register“
[19] Das ROM belegt 100 kB. Zu einer Adresse gehören jeweils 4 Byte Daten. Daraus ergibt sich, dass das ROM die Adresse 0 bis 25599 (= 100 * 1024 Byte / 4 - 1) benötigt. Der nächste Adressraum kann also bei 25600 beginnen. Analog ergibt sich die zweite Adresse zu ((100 + 400) * 1024 / 4).

Eine Transaktion läuft dabei folgendermassen ab. Der Initiator einer Transaktion („Master“) fragt bei der Arbitrierungslogik um Erlaubnis, den PCI-Bus benutzen zu dürfen. Ist diese erteilt worden, legt der Initiator die Adresse des gewünschten Empfängers (Slave) auf die Adressleitungen (sowie weitere Statusinformationen auf spezielle Leitungen).[20] Erkennt nun der Empfänger, dass die anliegende Adresse in seinen Bereich fällt, so teilt er dies dem Initiator mit und die Transaktion kann vollzogen werden. Zum Abschluss gibt der Master seine Erlaubnis zur Benutzung des Busses zurück.

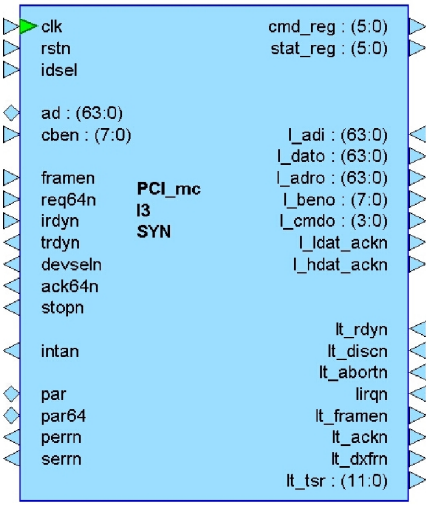

Abbildung 2.16: PCI-Megacore

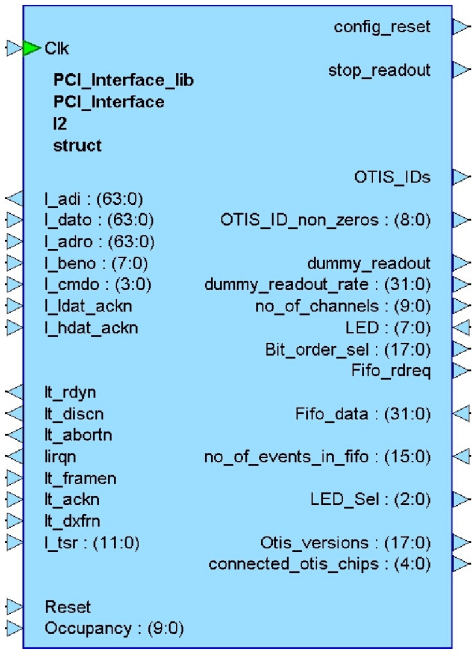

Abbildung 2.17: PCI-Interface

Die Implementierung im Shippo-Programm besteht aus zwei logische Einheiten (siehe Abbildung 2.18 auf der nächsten Seite). Die erste Einheit ist mit dem „*PCI Compiler*[21]*, 64-Bit Target Megacore Function* ^{TM}“ erstellt worden. Dieser ist eine sogenannte „Intellectual Property“ (IP). Damit stellt Altera fertige Einheiten zur Verfügung, die gegen eine Lizenzgebühr benutzt werden können. In diesem Fall wird mit dem Compiler eine Hardwarelogik erzeugt, die PCI-Busbefehle und Daten in einzelne Signale übersetzt und somit die Benutzung desselben erleichtert. Das erzeugt Megacore (Abbildung 2.16) kann dabei sowohl mit einem 64 Bit breiten als auch einem 32 Bit breiten PCI-Bus umgehen und mit 33 MHz und 66 MHz betrieben werden.

Dabei werden vom Programm zwei Basisadressen-Register benutzt. Mit dem ersten Register wird einen Adressraum von 4 kB belegt. Genutzt werden davon die Adressen 0 bis 31 sowie die Adresse 1023. Zugriff auf andere Adressbereiche sind nicht gestattet.

[20] z.B. die Art der gewünschten Transaktion
[21] In der Version 3.0.0

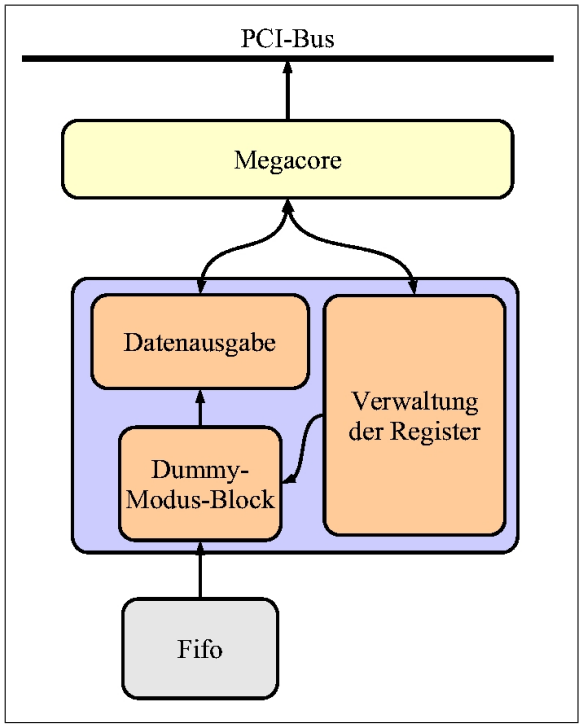

Abbildung 2.18: Schematischer Aufbau der Implementierung des PCI-Blocks

In diesem liegen die Konfigurationsregister, welche im Anhang A.4 auf Seite 69 näher erläutert werden.

Das zweite Basisadressenregister dagegen ist für die Auslese der Ereignisse zuständig. Es belegt einen Adressraum von 256 MB, wovon aber nur die ersten 288 Byte genutzt werden. Dieser grosse Unterschied erklärt sich dadurch, dass ursprünglich auch eine Auslese direkt aus dem DDR-RAM vorgesehen war. Diese Möglichkeit wurde aber später nicht implementiert. Es sind nur Lesezugriffe erlaubt, diese aber im gesamten Adressraum. Die Auslese selbst wird in Kapitel 2.5 beschrieben.

Beide Adressräume können nur als Empfänger einer PCI-Transaktion agieren. Das heisst, sie können nur auf Anfragen von ausserhalb antworten, selber aber keine eigenen stellen. Zugriffe, die nicht bearbeitet werden können, werden abgebrochen. Dies geschieht auch, wenn versucht wird, ein Ereignis aus einem leeren Fifo auszulesen.

Die zweite Einheit ist der Verwaltungsblock[22] (Abbildung 2.17 auf der vorherigen Seite). Dieser muss auf die PCI-Bussignale, die vom Megacore übersetzt wurden, antworten. Er verwaltet den eigentlichen Adressraum, d.h. hier sind die Daten, die zu den Adressen gehören, physisch abgelegt.

Der Verwaltungsblock besteht selbst wieder aus drei logischen Untereinheiten:

- Im ersten ist die Verwaltung der Konfigurationsregister realisiert. Diese steuern

[22] Im Programm mit PCI-Interface bezeichnet.

zentral alle Vorgänge innerhalb des Programms und liegen im Adressraum des ersten Basisadressen-Registers.

- Der zweite Teil ist für die Verwaltung der Daten von Ereignissen verantwortlich. Er sorgt dafür, dass immer ein Ereignis sofort auslesbar ist. Dazu liesst er ein Ereignis aus dem Fifo aus, sobald dieses welche beinhaltet. Im PCI-Dummy-Modus werden die Daten aus den entsprechenden Konfigurationsregistern geholt. Dies wird in Kapitel 2.5 auf Seite 36 noch näher erläutert.

- Die dritte Einheit kommuniziert wieder mit dem Megacore. Sie ist für den Adressraum des zweiten Basisadressen-Registers zuständig und sendet die in der zweiten Untereinheit gespeicherten Ereignisdaten auf Anfrage über den PCI-Bus.

Das Megacore ist dafür ausgelegt, sämtliche Funktionen des PCI-Bus zu unterstützen. Diese werden bei weitem nicht alle im Shippo-Programm benötigt, weshalb auch nicht alle Signale des Megacores ausgewertet werden. Das bedeutet, dass nur die beschrieben Funktionen benutzt werden dürfen. Das muss natürlich beim Erstellen eines Programms, welches die Stratix-Karte ausliest, berücksichtigt werden.

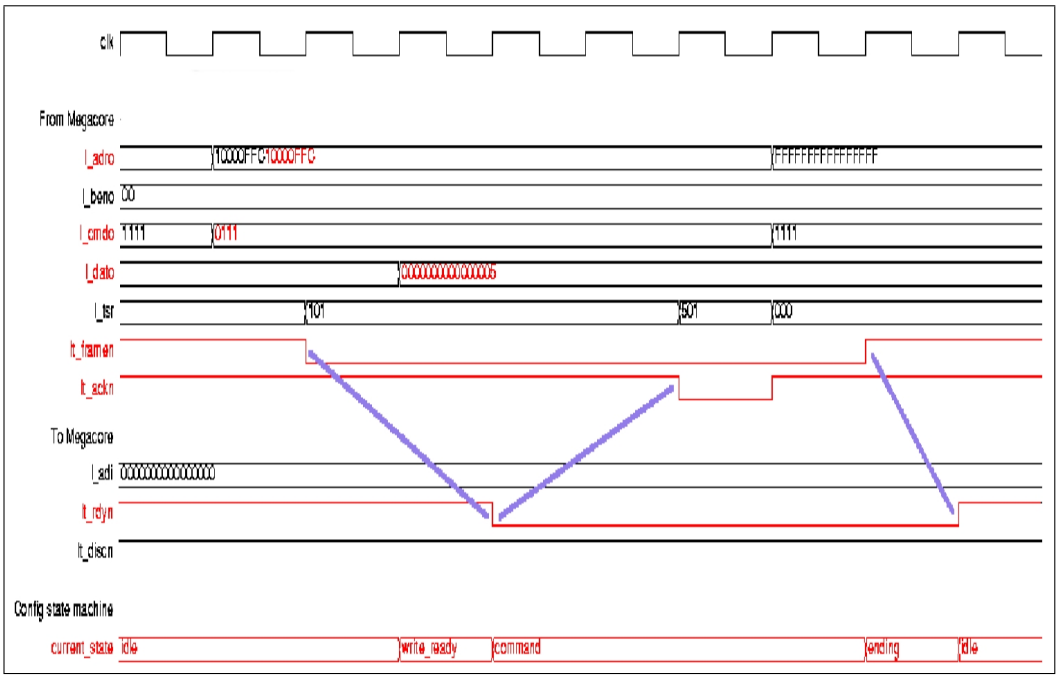

Abbildung 2.19: Schreiben des „set stop readout“-Befehls über den PCI-Bus (Simulation). Die beteiligten Signale sind farblich hervorgehoben. Bussignale werden mit ihren hexadezimalen Werten, die Zustände des Zustandsautomaten mit ihrem Namen angezeigt.

Abbildung 2.19 zeigt die Simulation eines Schreibvorgangs auf das Befehlsregister. In

dieser Simulation beginnt der Adressraum des ersten Basisadressen-Registers bei der Adresse x10000000.

Im zweiten Takt wird nun die Adresse, die vom Initiator der nächsten Transaktion auf dem PCI-Bus angelegt wird, vom Megacore zum Verwaltungsblock durchgeschaltet. Dies geschieht auf dem dafür vorgesehen Bus-Signal „l_adro“.[23] Der Adressabstand der angelegten Adresse von „xFFC“[24] zur Basisadresse x10000000 zeigt an, dass die Adresse 1023 im Adressraum des ersten Basisadressen-Registers angesprochen werden soll. Auf dieser Adresse liegt das Befehlsregister des Shippo-Programms (siehe auch Anhang A.4 auf Seite 69).

Auf einem weiteren Bussignal teilt der Initiator mit, welche Art von Transaktion er vornehmen möchte. Dies wird mit den Befehlssignalen angezeigt.[25] In diesem Fall soll ein Schreibzugriff auf den Speicher ausgeführt werden.

In Takt drei teilt nun das Megacore dem Verwaltungsblock mit, das die anliegende Adresse in den Zuständigkeitsbereich der Konfigurationsregister fällt. Dies wird erreicht, indem das Rahmensignal[26] aktiv geschaltet wird. Solange dieses Signal aktiv ist, findet eine Transaktion im eigenen Zuständigkeitsbereich statt.

Im nächsten Takt erkennt dies der zuständige Zustandsautomat und wechselt vom Wartezustand in den Zustand zur Entgegennahme von Daten (Zustand „write_ready“). Ein Takt später wird anhand des angelegten Adressabstandes festgestellt, dass die zu erwartenden Daten einen Befehl repräsentieren. Deshalb wird weiter in den Befehlszustand gewechselt (Zustand „command“). Die Bereitschaft des Verwaltungsblocks, die Daten zu empfangen, wird der Megacore-Einheit mit dem entsprechenden Signal angezeigt[27].

In Takt sieben zeigt das Megacore schliesslich dem Verwaltungsblock an, dass die Daten auf der Datenleitung[28] gültig sind. Dies wird durch das Aktivieren des Bestätigungssignals mitgeteilt.[29] Anschliessend wird das Ende der Transaktion mit dem Deaktivieren des Rahmensignals angezeigt, worauf der Zustandsautomat über den Zustand „ending“ in den Wartezustand („idle“) wechselt.

Dieselbe Abfolge von Stimuli wurde auch am FPGA-Chip überprüft. Die Abbildung 2.20 zeigt die Abfolge der Zustände und Abbildung 2.21 die zugehörigen Signale.

[23] = local adress out. Da der benutzte PCI-Bus eine Breite von 32 Bit besitzt, das Signal intern aber mit 64 Bit arbeitet, wird die Adresse sowohl in der unteren als auch in der oberen Hälfte angezeigt.
[24] = 1023 dezimal
[25] l_cmdo = local command out
[26] lt_framen = local target frame, activ low(negativ)
[27] lt_rdyn = local target ready, activ low(negativ)
[28] l_dato = local data out
[29] lt_ackn = local target acknowledge, activ low(negativ)

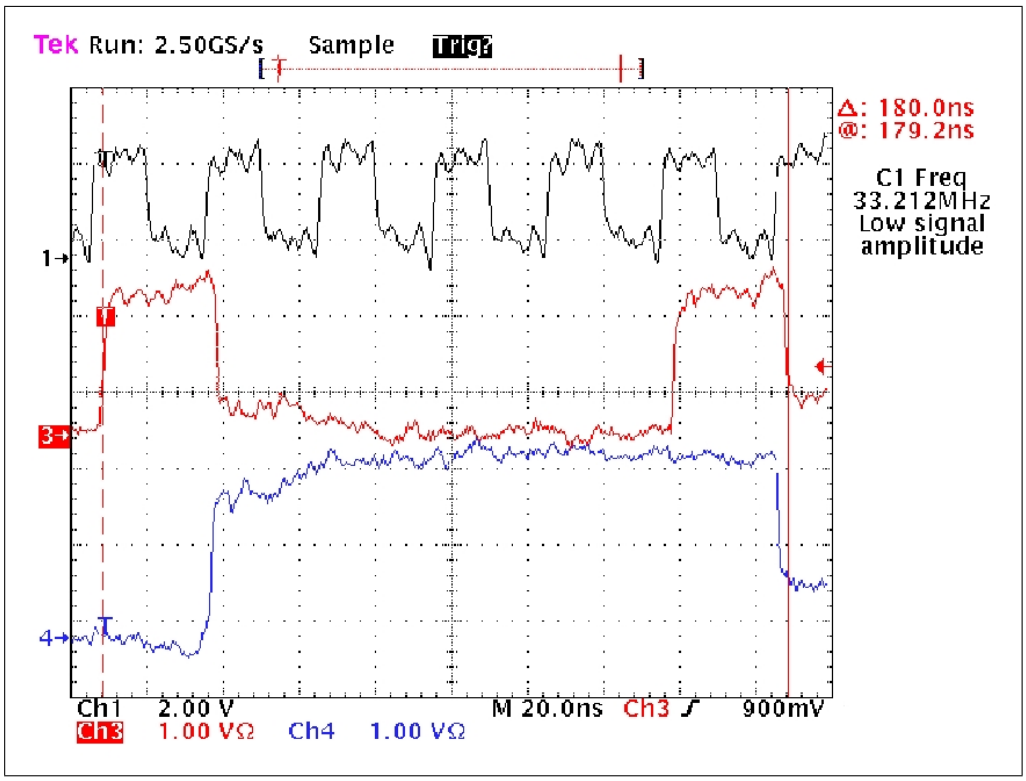

Abbildung 2.20: Zustände während eines Befehls über den PCI-Bus.
Kanal 1 (oben) zeigt den 33 MHz PCI-Bus-Takt.
Kanal 3 (mitte) zeigt das Zustandsbit 0.
Kanal 2 (unten) zeigt das Zustandsbit 1.
Die Kodierung der Zustände ist dabei: 00 = idle, 01 = write_ready, 10 = command und 11 = ending
Vgl. mit Abbildung 2.19 auf Seite 32. Zu Beginn erkennt der Zustandsautomat, dass ein Schreibzugriff vorbereitet wird, und wechselt in den Bereitschaftszustand. Im nächsten Takt wird dann in den Kommandozustand gewechselt. Nach erfolgter Übertragung wechselt der Zustandsautomat über den Zustand „ending“ in der Wartezustand.

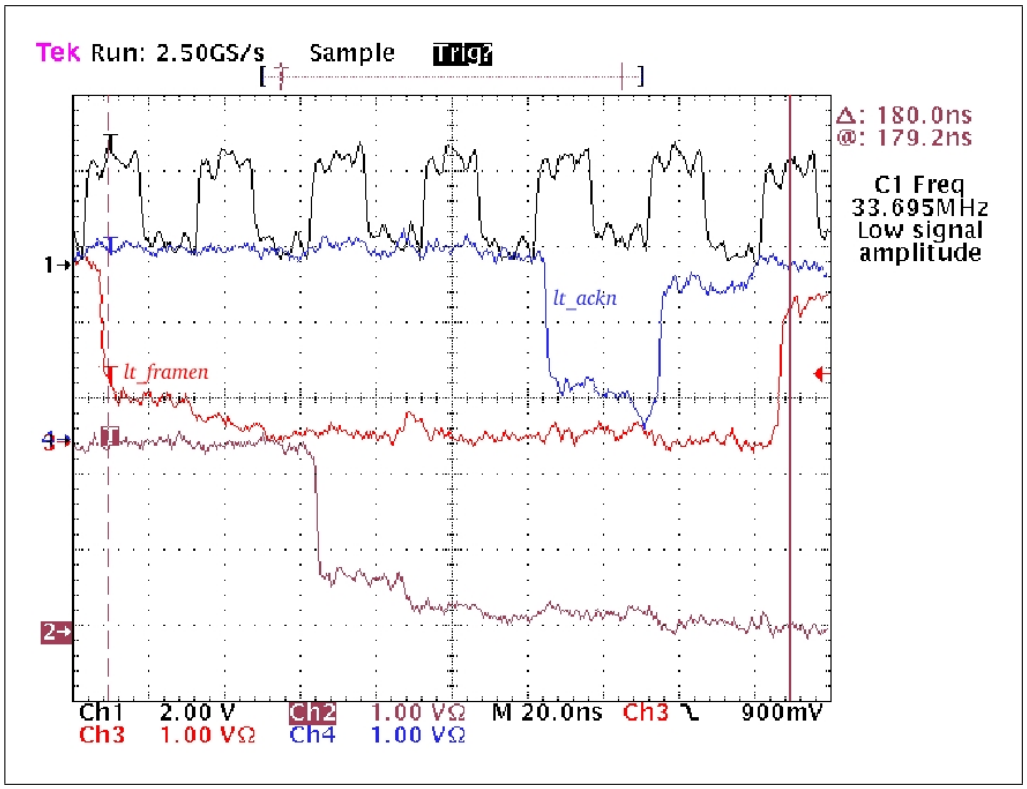

Abbildung 2.21: Signale während eines Befehls über den PCI-Bus.
Kanal 1 (oben) zeigt den 33 MHz PCI-Bustakt.
Kanal 3 (mitte) zeigt das Signal lt_framen. (negativ)
Kanal 4 (mitte) zeigt das Signal lt_ackn. (negativ)
Kanal 2 (unten) zeigt das Signal lt_rdyn. (negativ)
Vgl. mit Abbildung 2.19 auf Seite 32. Zu Beginn signalisiert der Megacore den Wunsch nach einer Transaktion durch das Aktivieren des Rahmensignals lt_framen. Der Verwaltungsblock erklärt seine Bereitschaft zur Annahme der Transaktion durch das Absenken des lt_rdyn-Signals zwei Takte später. Die Transaktion erfolgt schliesslich im fünften Takt, was durch das Bestätigungssignal lt_ackn angezeigt wird. Das Ende des Vorgangs ist am Deaktivieren des Rahmensignals zu erkennen.

2.5 Auslese

Die Auslese der Daten erfolgt über den Adressraum des zweiten Basisadressenregisters. Es wird davon ausgegangen, dass eine sequentielle Auslese erfolgt, d.h. zuerst Adresse 0, danach Adresse 4, 8, ..., 32 ausgelesen wird. Dabei enthält Adresse 0 den Header des Ereignisses, Adresse 4 die TDC-Zeiten der ersten vier Kanäle, Adresse 8 die nächsten vier usw. . Wenn Adresse 32 ausgelesen wurde, wird die Übertragung der Daten als vollständig angesehen und ein neues Ereignis aus dem Fifo geladen.

Insgesamt gibt es drei Auslesemodi. Die ersten zwei dienen dazu, die Funktionalität der Blöcke auf der Stratix-Karte zu testen. Im dritten Modus werden die Daten der optischen Empfängerkarte prozessiert. Tabelle 2.3 gibt eine kurze Zusammenfassung.

Der „PCI-Dummy-Modus“ ist der einfachste Modus. Mit ihm kann die Funktionalität des PCI-Blocks überprüft werden. Dazu reicht es, die Karte in einen PCI-Slot eines PC's einzubauen. Andere Signale müssen nicht angeschlossen werden. Wird nun eine Anfrage nach Daten von Seiten des PCI-Bus gestellt, so werden die Daten aus speziellen Registern geladen und gesendet. Ein nicht Funktionieren dieses Modus deutet auf einen schwerwiegenden Systemfehler auf dem PCI-Bus selbst hin.[30]

Der zweite Debug-Modus ist der „Dummy-Readout-Modus“. In diesem Modus kann das gesamte Programm getestet werden. Dazu muss der 80 MHz-Takt am *User Clock Pin* der Stratix-Karte angeschlossen sein. Die Daten werden dabei auf der Karte generiert. Über ein Register kann die Rate, mit der die Daten erzeugt werden, vorgegeben werden. Die generierten Ereignisse haben dabei folgendes Format:

```
OTIS-ID   Statusbits  Event-ID
002       0           0001
Zahlerdaten:
0a 0c 0e 10 12 14 16 18 1a 1c 1e 20 22 24 26 28
2a 2c 2e 30 32 34 36 38 3a 3c 3e 40 42 44 46 48
```

Dabei wird ein voller optischer Link emuliert, also vier OTIS-Chips. Die OTIS-IDs werden dabei aus den entsprechenden Registern entnommen. Die „Event-ID“ entspricht einem Triggerzähler und gibt die Anzahl der bisher erzeugten Datensätze an. Die Daten werden im Format der optischen Empfängerkarte an die Eingänge des Synchronisationsblocks gelegt.

[30] In der Regel bedeutet dies, dass die Karte nicht richtig im Slot eingebaut ist.

Modus	Beschreibung
PCI-Dummy-Modus	Einfacher Funktionstest der Auslese über den PCI-Bus.
Dummy-Readout-Modus	Funktionstest des gesamten Programms.
Standardmodus	Auslese der Daten von der optischen Empfängerkarte.

Tabelle 2.3: Die Auslesemodi des Programms

2.6 Bekannte Fehler und Probleme

Bei den Messungen sind teilweise Probleme aufgetreten. Die folgende Tabelle gibt Hilfen bei der Behebung dieser Probleme.

Fehlerbeschreibung	Lösung
„Ereignisse“ bestehen nur aus konstanten Werten, z.B. 08080808080808 usw.	Möglicherweise ist der High-Pegel der LHCb_clk. nicht hoch genug. Das verwendete Taktsignal sollte nur auf die Stratix-Karte gegeben und nicht aufgeteilt werden.
„Ereignisse“ bestehen nur aus zufälligen Werten oder beginnen immer „zu früh“	Überprüfen, ob der „Non_zero Modus“ aktiviert ist (ggf. ausschalten). Ist dies nicht der Fall, dann sollte man den Befehl „Config Reset“ schicken, um die Zustandsautomaten in einen definierten Anfangszustand zu versetzen.
Es werden keine Daten angezeigt bzw. gespeichert.	Überprüfen, ob die Übertragung zur Karte funktioniert. Überprüfen, ob die richtigen *OTIS-IDs* gesetzt wurden.

Tabelle 2.4: Probleme und Fehler

3 Messungen

Bei den Messungen wurde zunächst einmal die Funktionsfähigkeit des Programms überprüft. Anschliessend wurden Tests mit immer mehr Bestandteilen der gesamten Auslesekette der Outer-Tracker-Elektronik durchgeführt. Die Messungen entstanden in enger Zusammenarbeit mit *Dirk Wiedner*[11].

Für alle Messungen wurde der Computer „*he-lab1*“ genutzt. Dieser ist wie folgt ausgestattet:

- Pentium 4 mit 1,6 GHz-CPU
- Windows 2000 SP4
- Asus Motherboard mit 33 MHz PCI-Bus
- 786 MB Rambus-Speicher

Die Daten wurden von der *Stratix-Karte* ausgelesen und direkt auf der Festplatte gespeichert. Die Programme für die Auslese sowie für erste Analysen wurden im Rahmen einer Diplomarbeit [23] geschrieben. Die anschiessende Auswertungen erfolgte durch ROOT-Skripte, welche im Anhang B auf Seite 75 zu finden sind.

Fast allen folgenden Setups gemein ist, dass die Stratix-Karte den Referenztakt von 40 MHz aus einem Quarzkristall mit der doppelten Frequenz generiert. Dieser wurde in das TTC-System eingespeist, welches in Kapitel 1.4 vorgestellt wurde. Die Steuersignale werden dabei auf die optische Senderkarte gegeben. Über die Slow-Control wurde das Latency-Registers auf dem OTIS-Chip sowie die Schwellen für den Vorverstärker eingestellt

3.1 Testsetup für die Auslesekette

Der erste Test sollte die prinzipielle Funktionalität der Auslesekette zeigen.

Insgesamt beinhaltet dieser Test folgende Komponenten (siehe Abbildung 3.2):

- TTC-System
- LVDS-Verteilerkarte zur Vervielfältigung von Signalen
- Verzögerungselemente
- OTIS-Testkarte mit dem OTIS-Chip 1.0
- optische Senderkarte
- TLK-Karte (Vorgänger der optischen Empfängerkarte)
- Stratix Karte + PC

Das Taktsignal wurde von der optischen Senderkarte auf die LVDS-Verteiler-Karte gegeben. Diese vervielfältigt das Signal. Davon wurde eines auf den Takteingang der OTIS-Testkarte gegeben. Die anderen wurden zunächst direkt, später über ein Verzögerungselement auf die Hiteingänge 0, 1 und 2 der OTIS-Testkarte gegeben.

Das TTC-System generierte mit einer Rate von 500 Hz Zufallstrigger. Der OTIS-Chip gab seine Daten an die optische Senderkarte weiter. Von dort wurde sie an die TLK-Karte übertragen und über die Stratix-Karte in den PC eingelesen.

Die Erwartung ist nun, dass die Kanäle immer dieselbe „Driftzeit“ messen. Abbildung 3.1 zeigt für Kanal 2, dass das auch der Fall ist.

Alle 1000 Messpunkte sind in das Zeitbin 36 des OTIS gefallen. Das bedeutet, das die steigende Taktflanke etwa um $\frac{36}{64}25\ ns \approx 14\ ns$ später am Hiteingang als am Takteingang anlag. Das alle Einträge in einem Bin liegen, zeigt auch, dass die Signale in diesem Setup sehr stabil sind, d.h. wenig „Jitter“ produzieren.

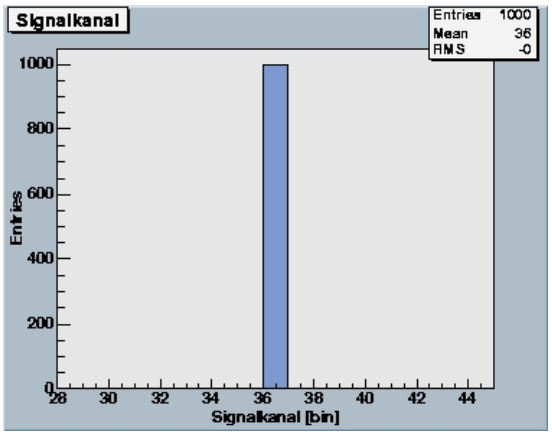

Abbildung 3.1: Taktsignal auf Hiteingang 2. Der Jitter des Signals ist im Vergleich zur Bingrösse von 390 ps so gering, dass alle Ereignisse in einem Bin landen.

Um nun zu bestätigen, das die Kanäle 0, 1 und 2 auch wirklich das richtige Signal sehen, wurde zusätzlich ein Verzögerungselement für die Hitsignale eingefügt. Für eine eingestellte Verzögerung von z.B. 8 ns lagen alle Einträge im Zeitbin 58. Dies sind 22 Bins mehr als ohne das Verzögerungselement, was einer Differenz von ca. 8,5 ns entspricht. Diese Differenz beinhaltet auch die Laufzeit der 10 cm Lemokabel, mit denen das Verzögerungselement an den Aufbau angeschlossen wurde.

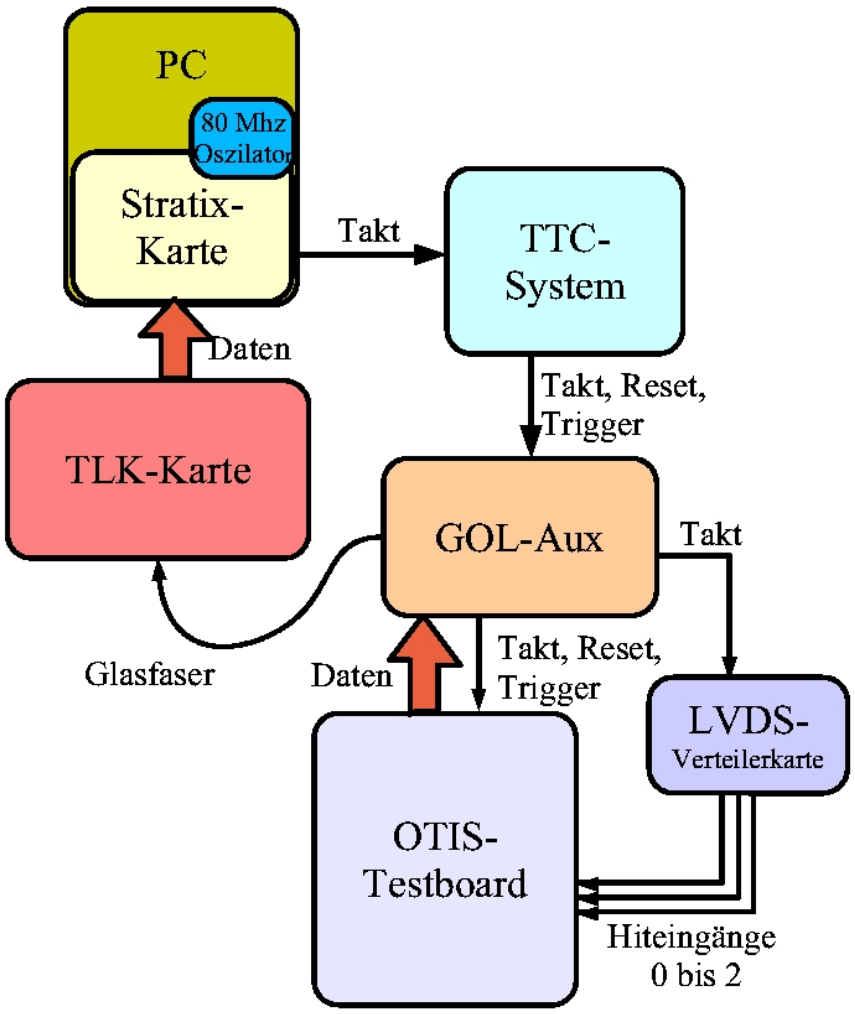

Abbildung 3.2: Aufbau für den ersten Test der Auslesekette (ohne Verzögerungselemente)

3.2 Einbau der Frontendbox

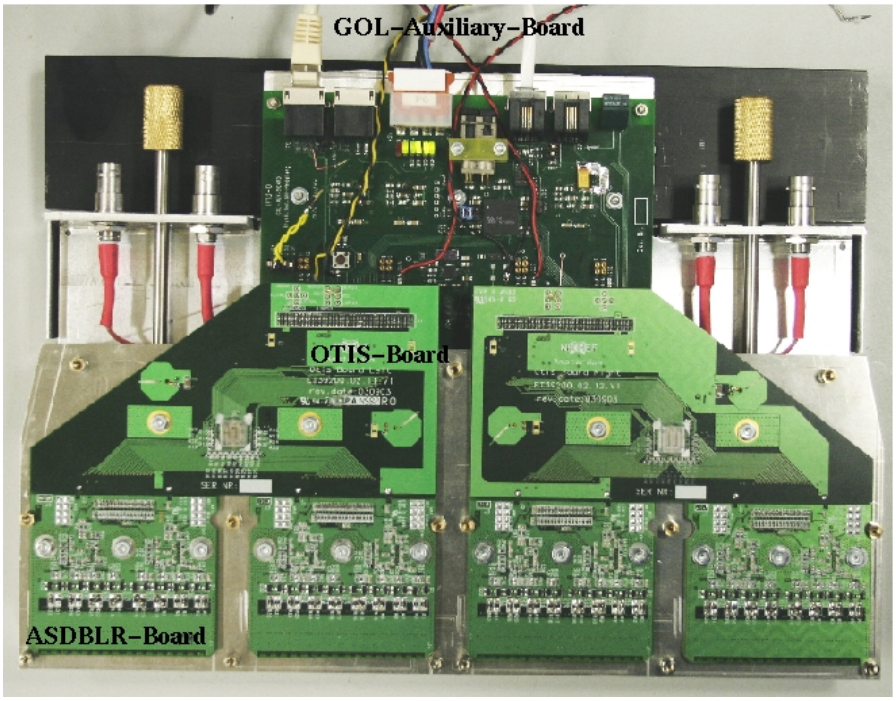

Abbildung 3.3: Die Frontendbox mit allen Karten. Das Foto wurde aus [11] entnommen.

Nach der vorherigen, erfolgreichen Messung wurde nun die Frontendbox (Abbildung 3.3) mit folgenden Komponenten in den Aufbau eingefügt.

- ASDBLR-Karten in der Version 1.0
- OTIS-Karten in der Version 1.0
- GOL-Aux-Karte in der Version IF13-0

Um ein Kammersignal zu simulieren, wurde der verzögerte Takt von der LVDS-Verteilerkarte auf die Signaleingänge der Vorverstärkerkarte gegeben. Das negative Signal wurde dabei über einen 10pF-Kondensator eingekoppelt. Über die Slow-Control wurde die Schwelle für den Vorverstärker auf 100 mV gesetzt, was einer Ladung von unter 2 fC entspricht. Das negative LVDS-Signal lieferte eine Amplitude von 300 mV. Damit ergibt sich für das Signal eine Ladung von $\sim 3000\ fC$. Abbildung 3.4 auf der nächsten Seite gibt einen Überblick über den Aufbau. Der Vorverstärker wurde dann von einem der vier OTIS-Chips ausgelesen. In der Synchronisation wurde nur nach dieser OTIS-ID gesucht, um die Daten der anderen drei Chips auszublenden.

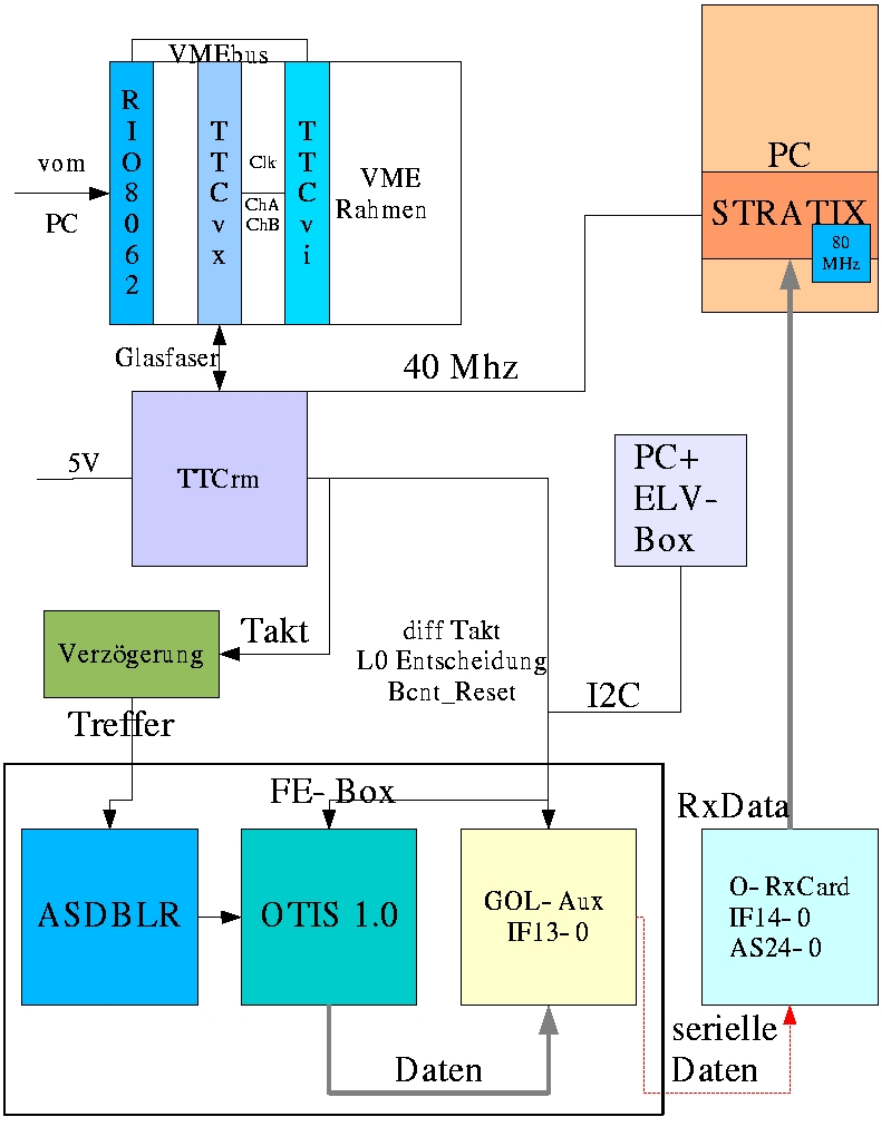

Abbildung 3.4: Aufbau für die Messung der Linearität des OTIS-Chips mit der Frontendbox. Abbildung aus [11] entnommen.

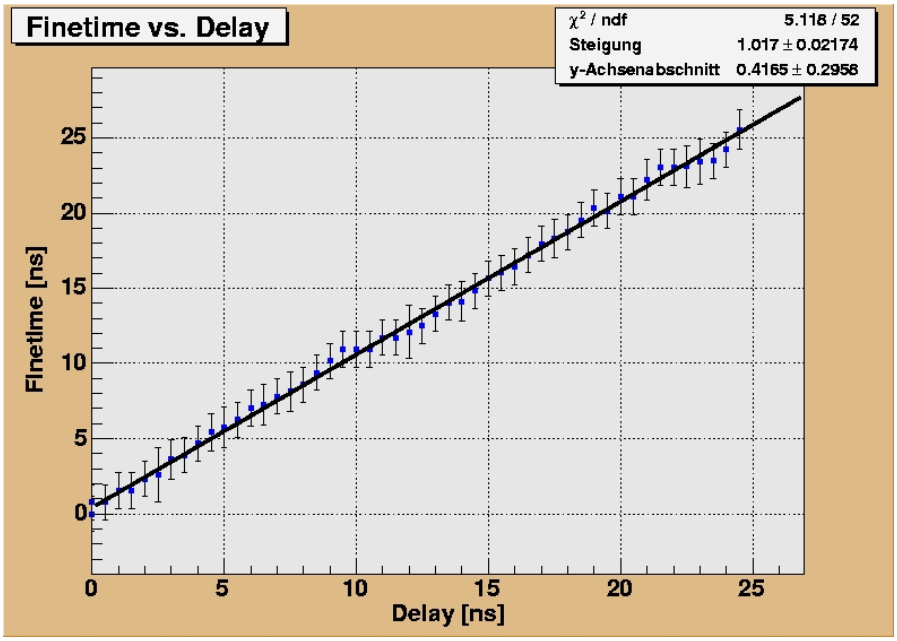

Abbildung 3.5: Messung der Linearität des Otis 1.0 mit Frontendbox

Dieses Signal wurde nun von 0 ns bis 25 ns in 0,5 ns Schritten verzögert. Für jeden Schritt wurden jeweils 1000 Ereignisse aufgenommen. Die so gewonnene TDC-Zeit wurde in Abhängigkeit der eingestellten Verzögerung aufgetragen. Die Messung (Abbildung 3.5) zeigt, dass die TDC-Zeiten der Verzögerung linear folgen. Mit einer ermittelten Steigung von 1,017±0,022 liegt das Ergebnis innerhalb des Erwarteten.[1] Der y-Achsenabschnitt gibt dabei nur den Startpunkt der Messung innerhalb der TDC-Zeitbins an. Dieser ist unabhängig von der Verzögerung und wird durch den Aufbau festgelegt. Die leichten Schwankungen der Messwerte um die Gerade werden durch das eingekoppelte Signal verursacht. Dieses wird aus dem negativen LVDS-Signal des TTC-Taktes gewonnen. Dabei wird ein zusätzlicher Jitter von bis zu 500 ps eingeführt. Dadurch kann es dazu kommen, dass das Signal des Vorverstärkers nicht immer in dasselbe Bin des TDC-Chip fällt. Es ist bekannt, dass der OTIS in der Version 1.0 in diesem Fall Probleme mit der TDC-Messung hat. Dies gilt insbesondere für die mittleren Bins sowie für die Bins am Ende der Inverterkette. Dennoch zeigt die Messung, dass die Datennahme mit Vorverstärker und TDC-Chip über den optischen Link funktioniert.

OTIS 1.1 Die Messung wurde später auch mit dem OTIS 1.1 durchgeführt. Dabei musste der Aufbau allerdings ein wenig verändert werden, da noch keine Karten mit dem OTIS 1.1 produziert waren. Eine offene Frontendbox, auf der eine Vorverstärkerkarte geschraubt war, wurde dabei an ein Driftkammermodul angeschlossen. Die Ausgangs-

[1] Im Idealfall wird eine Steigung von 1 erwartet.

pins dieser Vorverstärkerkarte wurden dann mit Fädeldraht auf einen grösseren Stecker geführt. Im Einzelnen waren dies die Signale für

- Schwelle
- +3 V
- -3 V
- Kanal 16
- Kanal 15
- Masse (mehrfach)

Dieser Stecker wurde dann über ein Flachbandkabel mit einer OTIS-Testkarte verbunden. Zusätzlich wurde auch die Frontendbox über ein breites Kupferkabel an den Massepegel der OTIS-Testkarte angeschlossen.

Dabei wurde nicht mehr das Taktsignal auf die Hiteingänge des OTIS gelegt, sondern mit einem Pulser ein Signal erzeugt. Dieses Signal wurde dann aufgeteilt und

1. ... als Trigger in das TTC-System eingespeist.
2. ... direkt auf Kanal 5 des OTIS-Testboard gelegt.
3. ... über ein Verzögerungselement auf Kanal 6 gelegt.

Da der Pulser asynchron zum Takt aus der Stratix-Karte lief, benötigt man eine Zeitreferenz. In dieser Messung war dies der Kanal fünf, im folgenden Referenzkanal genannt. Die gemessene Verzögerung wird nun aus der Differenz von gemessenen TDC-Zeiten des sechsten Kanals (im folgenden Signalkanal genannt) und des Referenzkanals gebildet.

Am Beispiel der Messung für eine Verzögerung von 5,0 ns soll der Ablauf verdeutlicht werden. Zuerst werden 1 000 Ereignisse aufgenommen. Die Ergebnisse sind in den Abbildungen 3.6 und 3.7 gezeigt.

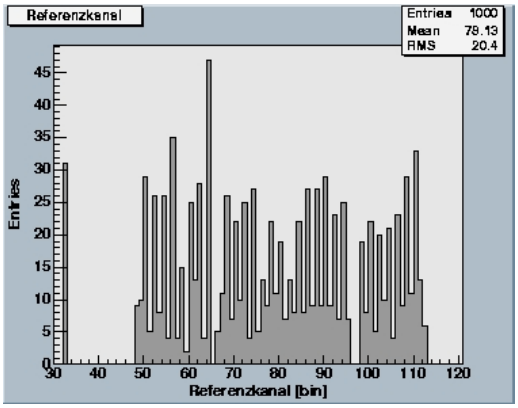

Abbildung 3.6: Referenzkanal

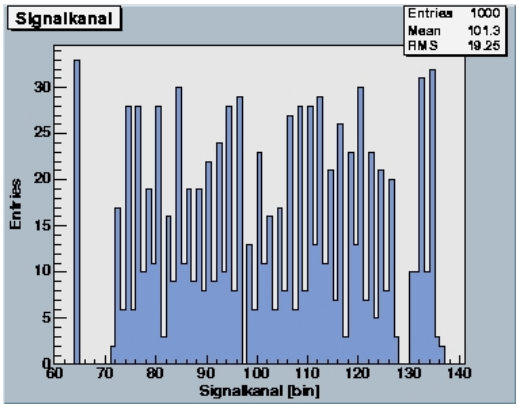

Abbildung 3.7: Signalkanal

Anschliessend wird für jedes Ereignis die Differenz der beiden Kanäle gebildet. Das Ergebnis für dieses Beispiel beträgt (22,71 ± 0,83 Bins). Allerdings flossen in diesen Peak nur 860 von 1000 Ereignissen ein (Abbildung 3.9).

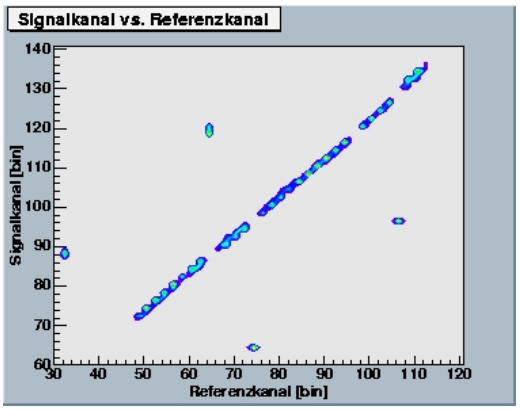

Abbildung 3.8: Signalkanal gegen Referenzkanal

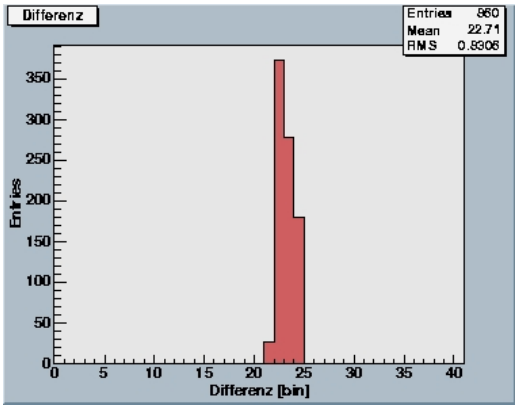

Abbildung 3.9: Differenz zwischen Referenz und Signal

Die fehlenden Ereignisse bildeten zwei Nebenpeaks 40 Bins vor bzw. nach diesem Hauptpeak. Für 140 Events wurde also eine „falsche“ Zeit bestimmt, wie man auch der Abbildung 3.8 entnehmen kann. Hier wurde der Referenzkanal gegen den Signalkanal aufgetragen. Dabei treten die Lücken immer dann auf, wenn ein Kanal den Wert 96 oder 128 annimmt. Dies konnte später auf ein Problem mit dem QPLL-Chip auf der optischen Senderkarte zurückgeführt werden, welches in Kapitel 3.3 näher erläutert wird.

Abbildung 3.10 zeigt das Ergebnis der gesamten Messung. Als Fit an die Steigung der Geraden erhält man 1,005 ± 0,005 und somit das erwartete Verhalten.

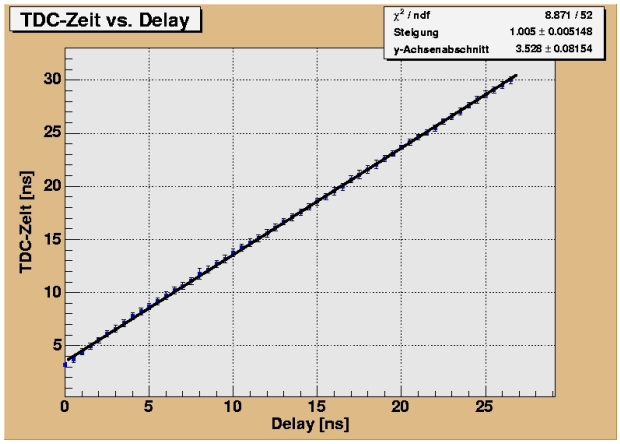

Abbildung 3.10: Messung der Linearität des OTIS 1.1

3.3 Probleme mit dem QPLL-Chip

Im vorherigen Kapitel wurde auf die Probleme des QPLL-Chips verwiesen. Diese sollen an dieser Stelle näher erläutert werden. Der QPLL-Chip sitzt auf der optischen Senderkarte und soll dort für eine Stabilisierung der Taktflanke sorgen. Die optische Senderkarte wurde aber schon vor den endgültigen Spezifikationen der QPLL entworfen und konnte somit nicht alle Beschaltungsvorschriften berücksichtigen. Daneben handelt es sich bei der verwendeten QPLL um einen Prototypen, der nicht ausgereift ist.

Dies hatte zur Folge, dass der Taktbereich, in dem sich die QPLL auf den vom TTC-System gelieferten Takt einstellen kann, nur etwa 5 kHz bei Idealbedingungen betrug. Dies führte bei Temperaturen ab 26°C zum zeitweisen Ausfall der Übertragung zwischen optischer Sender- und Empfängerkarte.

Da auch der OTIS-Chip an dieser Taktversorgung angeschlossen ist und Messungen der TDC-Zeiten immer in Referenz zum Takt auf dem OTIS durchgeführt werden, wirken sich alle Probleme mit dem Taktsignal auf die Messungen aus. Dies zeigt sich durch ein „Zerfliessen“ der Daten von eigentlich scharf definierten Grenzen, in denen die TDC-Zeiten gemessen werden sollten.

Die nachfolgenden Abbildungen verdeutlichen diesen Effekt. Abbildung 3.11 auf der nächsten Seite zeigt eine Messung mit einer fehlerfrei arbeitenden QPLL. Die Messung ist dabei so gestaltet, dass die Signale gleichverteilt innerhalb von 25 ns kommen. Abbildung 3.12 auf der nächsten Seite zeigt die Situation, wenn die QPLL keinen stabilen Takt liefert. Die Verteilung der Einträge wird dabei gaussförmig verschmiert.[2] Dies liegt daran, dass die QPLL, wenn sie es nicht mehr schafft, den Takt nachzuregeln, eine neue Synchronisation auf den TTC-Takt versucht. Dabei ändert sich die Phasenlage des Taktes, so dass aus Sicht der OTIS-Chips die Signale mal früher, mal später relativ zum eigentlichen Zeitpunkt ankommen. Weiterhin sieht man in dieser Abbildung, dass einige Bins überhaupt nicht mehr getroffen werden, andere hingegen überproportional oft.

Um diese Fehler bei Messungen zu minimieren, muss die QPLL gekühlt werden. Nachdem mehrere Ansätze ausprobiert wurden, bestand die Lösung schliesslich darin, auf die QPLL ein kleines Kühlelement zu kleben. An dieses wurde ein Peltierelement angebracht, auf dessen heisse Seite ein weiteres Kühlelement geklebt wurde. Der Aufbau wurde zusätzlich in einen beständigen Luftstrom gestellt.

Für die Zukunft gibt es zwei Entwicklungen, die dieses Problem beseitigen sollen. Zum einen gibt es eine neue Version des QPLL-Chips, die sich über einen grösseren Bereich auf das TTC-Taktsignal synchronisieren kann. Zum anderen werden in der neuen Version der optischen Senderkarte die Beschaltungsvorschriften der QPLL besser erfüllt.

[2] Gezeigt ist nicht die Verschmierung des „perfekten“ Datensatzes. Dieser würde für diese Messung etwa zwischen Bin 60 und 120 liegen.

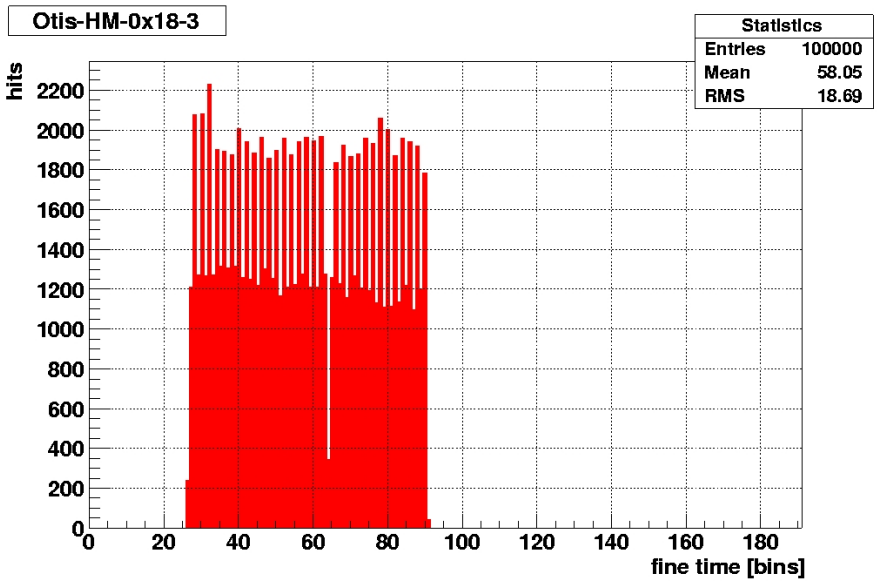

Abbildung 3.11: „perfekter“ Datensatz, stabiler Takt von der QPLL

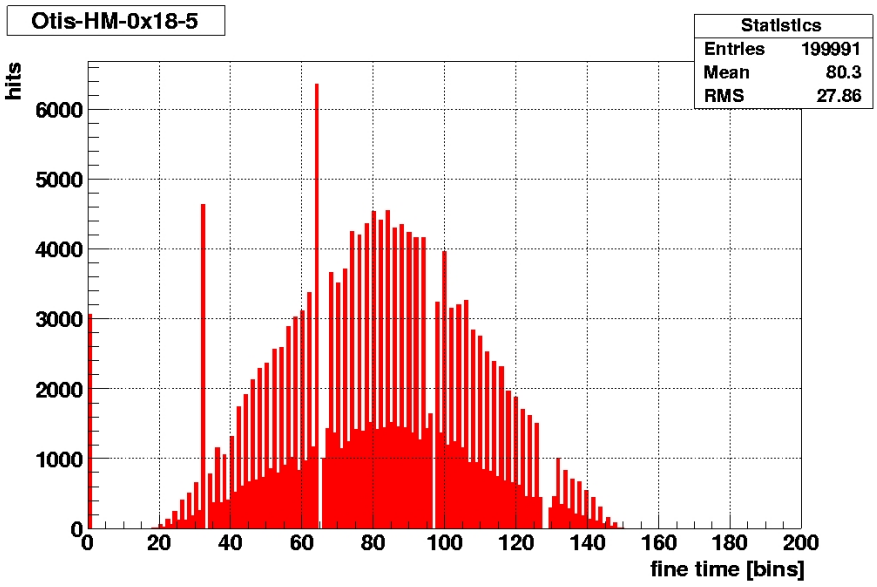

Abbildung 3.12: Zerfliessen des Datensatzes bei Fehlern der QPLL

3.4 DNL des OTIS-Chips

Mit der differentiellen Nichtlinearität (kurz DNL) bestimmt man, wie sehr sich die einzelnen Zeitbins des OTIS in ihrer Grösse unterscheiden. Dazu wurde mit einem Pulsgenerator ein 10,7-MHz Signal erzeugt. Dieses Signal wurde in ein differentielles umgewandelt und auf die Hiteingange 3 und 4 der OTIS-Testkarte gegeben. Zum Einsatz kam dabei der OTIS 1.1. (Für den OTIS 1.0 ist die DNL schon früher vermessen worden.)

Zunächst wurde das TTC-System auf „Random trigger“ mit 100 kHz gestellt. Dieser Trigger wurde allerdings nicht direkt auf die OTIS-Testkarte geführt, sondern, nachdem es in ein NIM-Signal umgewandelt wurde, auf den Starteingang eines Gate-Generators gelegt. Dasselbe Signal wurde um 16 ns verzögert auf den Stoppeingang gegeben. Daraus resultierte ein 10,2 ns langes Signal. Dieses wurde in ein LVDS-Signal umgewandelt und an den Triggereingang der OTIS-Testkarte angeschlossen. Dabei wurde darauf geachtet, das es genau mittig über der fallenden Taktflanke liegt. Dieser Aufwand wurde betrieben, da zum damaligen Zeitpunkt das Triggersignal in Verdacht stand, Probleme in der Messung der mittleren TDC-Bins zu verursachen. Dies konnte später jedoch widerlegt werden.

Mit diesem Aufbau erwartet man, dass alle Zeitbins des OTIS statistisch gesehen gleich oft getroffen werden, da das Hitsignal asynchron zum Triggersignal und Taktsignal läuft. Abbildung 3.13 zeigt das Ergebnis der Messung für Kanal 3.

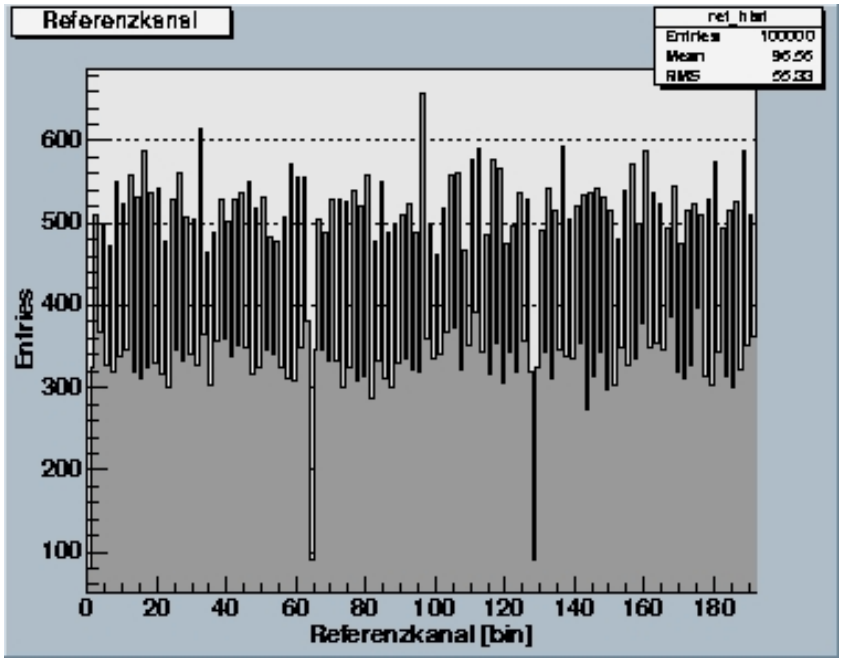

Abbildung 3.13: Verteilung der Hits in den OTIS-Bins für den Kanal 3. Die 18 771 Ereignisse mit Driftzeit 192 wurden der Übersicht halber ausgeblendet.

Da die OTIS-Bins 0, 64 und 128 im OTIS-Chip aus demselben Inverter erzeugt werden und nur durch die BX-Information unterschieden werden, kann man diese Bins zusammenfassen. Dasselbe gilt natürlich auch für die Bins 1, 65 und 129, für 2, 66 und 130 usw. . In Abbildung 3.14 sieht man die Ausgangssituation für die DNL-Bildung.

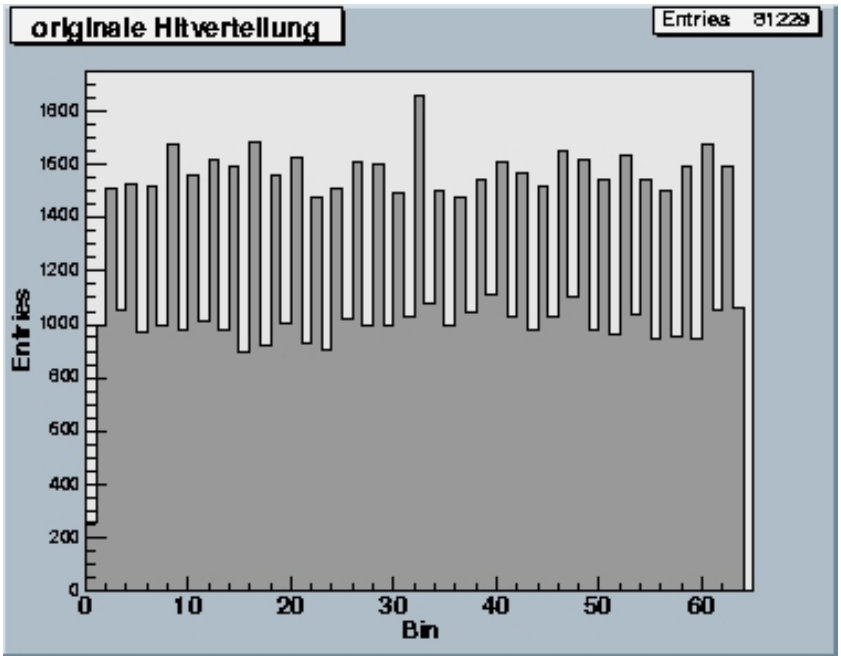

Abbildung 3.14: Die Verteilung der Hits, nachdem dieselben Bins zusammengefasst wurden

Nun wird für jedes Zeitbin die DNL bestimmt. Die Formel dafür lautet:

$$\mathrm{DNL(i)} = \frac{\text{Einträge in Bin (i)} - \text{Einträge in Bin (i-1)}}{(\sum \text{aller Einträge})/\text{Anzahl der Bins}}$$

Um die DNL für den gesamten Chip zu bestimmen, wird die Differenz aus dem maximalen und dem minimalen Wert der DNL für die einzelnen Bins gebildet. Ein Wert von 0 LSB [3] würde bedeuten, dass alle Zeitbins gleich lang sind und stellt damit den Idealwert dar. Ein Wert grösser als 1 LSB bedeutet hingegen, dass Driftzeiten unter Umständen falschen Bins zugeordnet werden. Das Ergebnis dieser Messung ist (1,288 ± 0,071) LSB, wie in Abbildung 3.15 auf der nächsten Seite zu sehen ist.

Betrachtet man die Abbildung 3.14, so weicht das Ergebnis sehr stark von der idealen Gleichverteilung ab. Zwei Merkmale fallen dabei besonders auf:

- Bin 0 enthält deutlich weniger Einträge als alle anderen.

[3] LSB = least significant bit

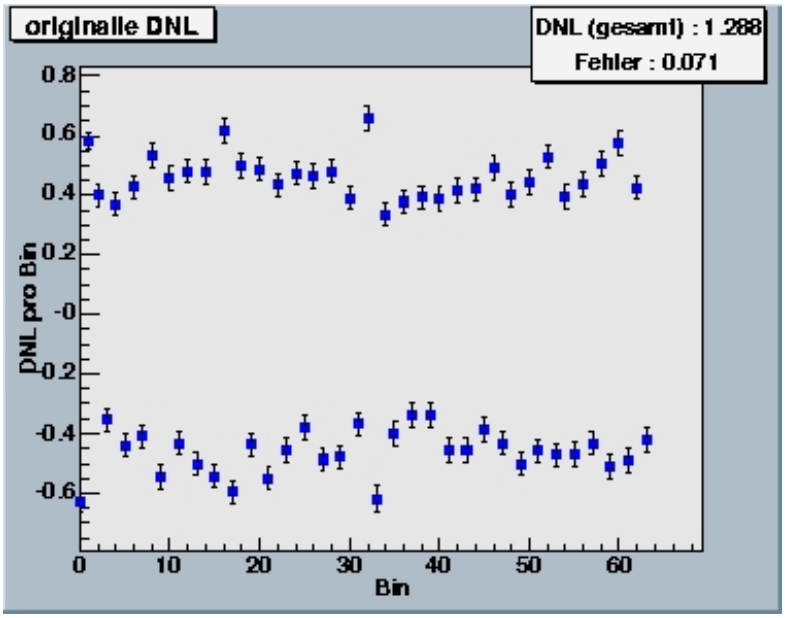

Abbildung 3.15: Ergebnis der DNL-Messung für Kanal 3

- Bins mit einer gerader Nummer enthalten mehr Einträge als Bins mit einer ungerader Nummer.

Beide Merkmale waren schon vor der Messung bekannt und auch so erwartet worden. Sie sind beide im Design des Chips begründet. Dazu muss man sich vergegenwärtigen, wie die Bins erzeugt werden.

Der OTIS-Chip bedient sich dabei einer Delay-Lock-Loop (DLL) Schaltung. Dabei werden 64 Inverter hintereinader geschaltet. Die Laufzeit durch diese Inverterkette wird nun mittels einer Kontrollspannung so angepasst, dass diese genau einem Taktzyklus entspricht.

Bei der Herstellung des Chips können aber nicht alle Inverter gleich gebaut werden. Vielmehr bestehen die Inverter abwechselnd aus PMOS und NMOS Transistoren. Diese belegen aber unterschiedlich viel Platz auf dem Chip, was letztlich zu unterschiedlichen Laufzeiten führt. Somit erhält man abwechselt einen „langen“ Zeitbin, gefolgt von einem „kurzen“. Abbildung 3.16 auf der nächsten Seite verdeutlicht diesen Effekt.

Bei Bin 0 kommt ein weiterer Effekt zum Tragen. Damit die Laufzeit gleichmässig über alle Inverter verteilt wird, sollte für alle Inverter dieselbe Kapazität an ihren Ausgängen anliegen. Leider ist dies bei Bin 0 nicht der Fall, da dieser sich am Anfang bzw. Ende der Kette befindet. In der OTIS-Version 1.2 ist versucht worden, dies durch ein manuelles Anpassen der Impedanz auszugleichen, was auch zumindest zum Teil gelungen ist.[4]

[4]Der OTIS 1.2 stand leider erst im Oktober 2004 zur Verfügung, so dass er nicht mehr für Messungen in dieser Diplomarbeit verwendet werden konnte.

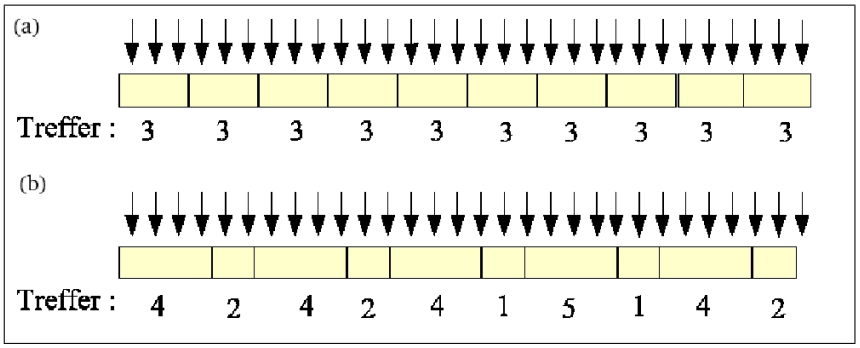

Abbildung 3.16: Trefferverteilung bei unterschiedlichen Binlängen. In (a) ist der Idealfall gezeigt. Alle Bins enthalten gleichviele Treffer. (b) dagegen zeigt den Fall, wenn die Bins unterschiedlich gross sind.

3.4.1 Korrekturen der DNL durch Gewichte

Mit diesem Wissen kann man nun die Einträge gewichten. Die bisherigen Messungen zeigen für die kurzen Bins eine Laufzeit von etwa 290 ps und für die langen Bins eine Laufzeit von 490 ps. Beide Werte beinhalten einen geschätzten Fehler von 5 bis 10%. Bin Null entspricht einer Laufzeit von 90 ps. Damit erhält man folgende Gewichtungsfaktoren: Für Bin Null 4,333, für gerade Bins 0,796 und für ungerade 1,345. Abbildung 3.17 zeigt die so erhaltene Hitverteilung, Abbildung 3.18 die resultierende DNL. Das Ergebnis ist mit (0,452 ± 0,078) LSB deutlich besser. Das bedeutet, dass unter Berücksichtigung der Binlängen die Auflösung von 390 ps erreicht wird.

Dies wird im LHCb-Experiment automatisch durchgeführt. Da die unterschiedlichen Längen der Bins für die OTIS-Chips genau bekannt sind, kann dieses auf der TELL1-Karte korrigiert werden. Dazu werden die unterschiedlichen Längen in einer Nachschlagetabellen gespeichert.

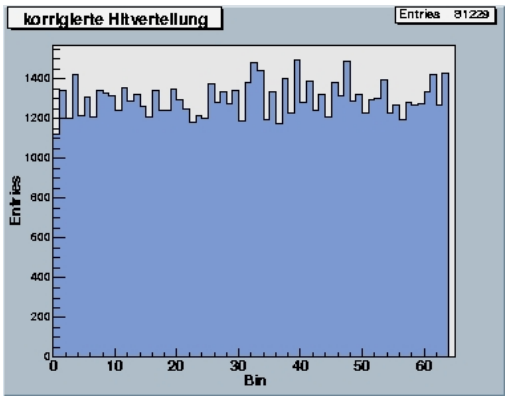

Abbildung 3.17: Gewichtete Hitverteilung für Kanal 3

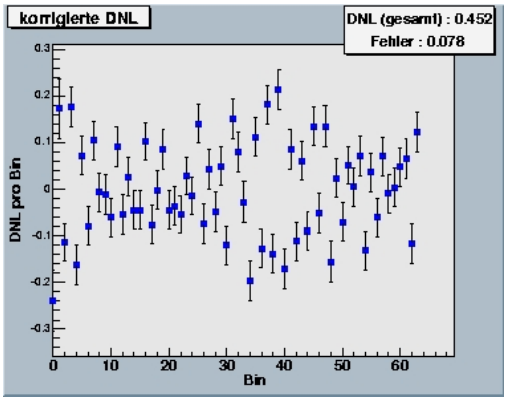

Abbildung 3.18: Ergebnis der DNL-Messung mit gewichteten Einträgen

3.4.2 Korrektur durch Zusammenfassen zweier Bins

Eine zweite mögliche Betrachtungsweise liegt in der Zusammenlegung zweier benachbarter Bins. Damit haben die zusammengelegten Bins alle eine Länge von 780 ps. Eine Ausnahme bildet weiterhin das Bin 0, das auch für diese Analyse mit einem Faktor gewichtet werden muss.

Das Ergebnis wird in den Abbildungen 3.19 und 3.20 gezeigt. Die DNL beträgt nun (0,596 ± 0,103) LSB, wobei sich diese Angabe jetzt im Gegensatz zu den vorherigen auf Bins der Breite 780 ps bezieht. Wie man erkennen kann, trägt Bin Null trotz der Gewichtung noch zur DNL bei. Ohne dieses Bin lautet das Ergebnis (0,347 ± 0,093) LSB.

Alle diese Messungen zeigen, dass der OTIS-Chip die geforderte Auflösung von $< 1ns$ liefern kann.

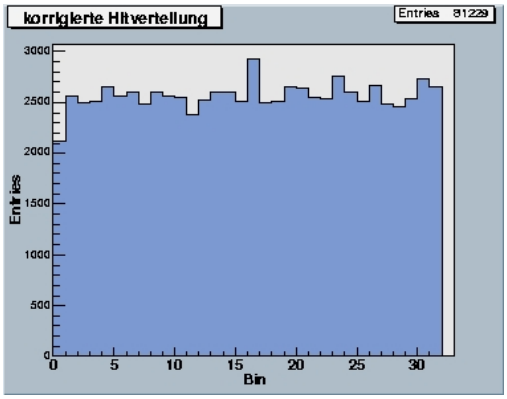

Abbildung 3.19: Zusammengefasste Hitverteilung für Kanal 3. In der Messung fallen insbesondere im zweiten BX überproportional viele Ereignisse in das mittlere Bin. Dies ist aller Wahrscheinlichkeit eine statistische Schwankung in dieser Messung, da dieses Verhalten in keiner anderen Messung für den OTIS 1.1 bestätigt wurde.

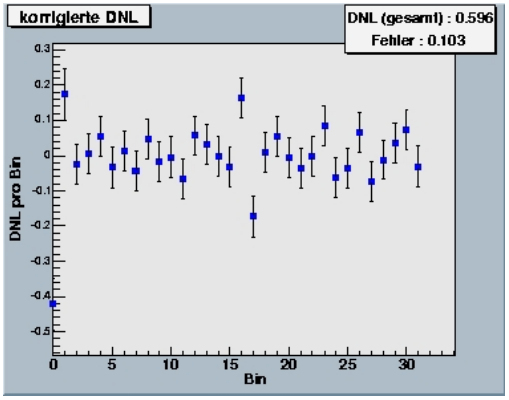

Abbildung 3.20: Ergebnis der DNL-Messung mit zusammengefassten Einträgen

3.5 Driftzeitmessungen

Nachdem die Kontrollmessungen erfolgt waren, wurde ein Driftzeitspektrum angefertigt. Das Gelingen dieser Messung zeigt, dass alle Komponenten funktional zusammenarbeiten. Für die Messung wurde ein 1m langes Driftkammermodul [21] genutzt. Der Aufbau war dabei ähnlich wie in Kapitel 3.2 für den OTIS 1.1 beschrieben. Die Frontendbox wurde aber nicht mehr über einen Pulser mit Signalen gespeist, sondern an das 1m Modul angeschlossen. Eine Rutheniumquelle wurde über den Szintillatoren und dem Modul plaziert.

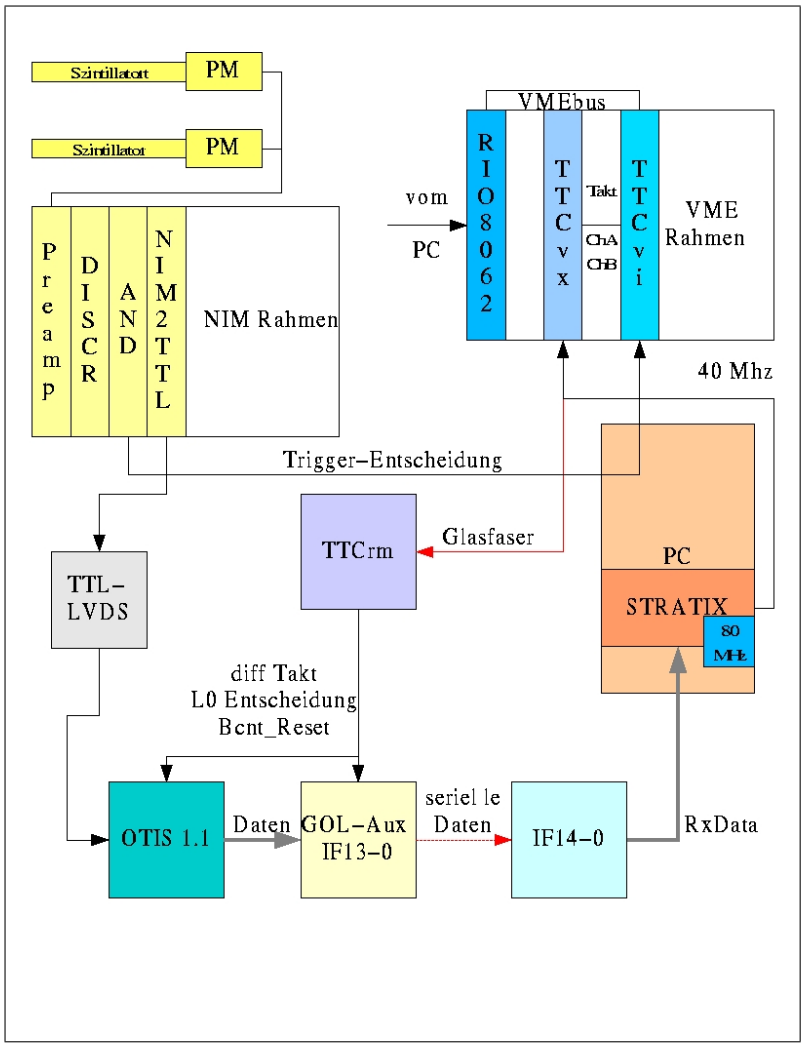

Abbildung 3.21: Triggererzeugung für die Driftzeitspektren. Abbildung aus [11] entnommen.

Der Trigger wurde mit zwei Szintillatoren erzeugt (siehe Abbildung 3.21). Dazu wur-

den die Signale von den Szintillatoren zunächst verstärkt und anschliessend auf einen Diskriminator gegeben. Dieser lieferte dann ein 10 ns breites Gate. Sprachen beide Szintilatoren an, wurde daraus ein TTC-Trigger generiert. Ausserdem dienten die Signale aus dem Diskriminator auch als Zeitreferenz und wurden auf die OTIS 1.1-Testkarte gegeben.

Tabelle 3.1 zeigt alle Signale, die auf die Testkarte gegeben wurden.

OTIS Kanal	Belegung
0	—
1	Szintilator 1 (Referenz)
2	Szintilator 2 (Referenz)
3	—
4	nicht angeschlossener ASDBLR-Pin
5	—
6	ASDBLR-Kanal 15 (Driftkammersignal)
7	ASDBLR-Kanal 16 (Driftkammersignal)
8 - 31	—

Tabelle 3.1: Anschlüsse an die OTIS-Testkarte

Ab hier kam dann die Auslesekette zum Einsatz, die auch in den vorherigen Messungen eingesetzt wurde. Im einzelnen ist dies:

OTIS-Testkarte ⇒ optische Senderkarte ⇒ optische Empfängerkarte

An die optischen Empfängerkarte wurde die Stratix-Karte angeschlossen.

Latency	15	16	17	18	19
Kanal 6	10	48	71	82	43
Kanal 7	14	41	59	76	42

Tabelle 3.2: Anzahl der Treffer auf den Kanäle 6 und 7 bei verschiedener Latency

Das Modul wurde mit einer Spannung von 1 600 V betrieben, der Pegel der Schwellen des Vorverstärkers wurde mit 750 mV festgelegt, was einer Ladung von ∼5 fC entspricht. Um den richtigen Wert für das Latency-Register des OTIS-Chips zu bestimmen, wurde für verschiedene Einstellungen die Anzahl der Treffer auf den Driftkammerkanälen bestimmt. Tabelle 3.2 zeigt das Ergebnis. Leider konnten die Referenzsignale nicht schnell genug auf die Testkarte gegeben werden, so dass diese bei einer Latency von 18 nicht mehr in das 75 ns breite Auslesefenster gepasst haben. Deshalb wurden alle Messung mit der Latency 17 durchgeführt (siehe auch Abbildung 3.22).

Mit diesem Aufbau wurden dann 200 000 Ereignisse aufgezeichnet. Aus diesen wurden dann alle Ereignisse ausgewählt, die folgende Bedingungen erfüllen:

1. Die Referenzkanäle zeigen einen Treffer.

2. Nur einer der beiden Signalkanäle zeigt einen Treffer.

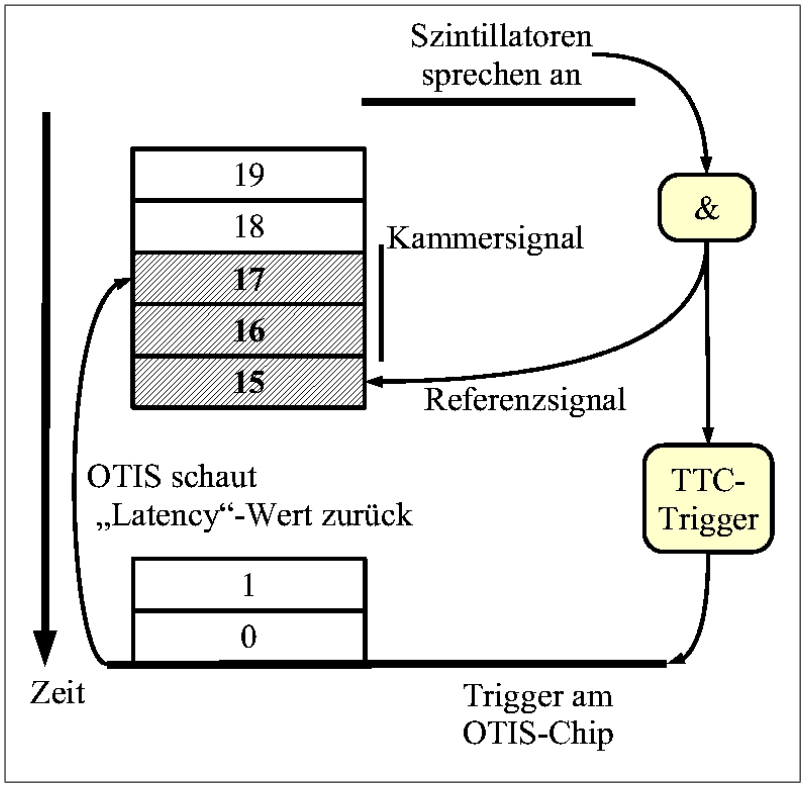

Abbildung 3.22: Abfolge der Signal bei der TDC-Messung. Die numerierten Blöcke stellen die Einträge eines BX in der Pipeline des OTIS dar. Bei einer eingestellter Latency von 17 werden die Daten der BX 15 bis 17 übertragen.

3. Das Ereignis ist vollständig übertragen worden.[5]

Diese Auswahl wurde mit einem ROOT-Skript durchgeführt.[6] Punkt drei stellt sicher, dass die Verbindung stabil war. Die benutzte Version des QPLL-Chips hat einige Probleme, die zu einer Unterbrechung der Übertragung führen können. In den Daten haben dadurch ab einem bestimmten Kanal alle TDC-Zeiten den gleichen Wert. Punkt zwei eliminiert die Ereignisse, bei denen sowohl Kanal 6 als auch Kanal 7 eine Driftzeit gemessen haben. Das Ansprechen beider Kanäle ist ein Indiz für ein Übersprechen auf die Fädeldrähte aus anderen Quellen. Nach Durchführung dieser Auswahl standen noch 31 029 Ereignisse für die Analyse zur Verfügung. Abbildung 3.23 und 3.24 auf der nächsten Seite zeigen die so gewonnenen Histogramme.

Anschliessend wurden wieder die Differenz und die Kontur aus beiden Signalen gebildet. Dies wurde mit einem weiteren ROOT-Skript durchgeführt.[7] Das Ergebnis ist

[5] Die Übertragung kann durch den QPLL-Chip unter bestimmten Umständen gestört werden.
[6] siehe Anhang B.1 auf Seite 75
[7] siehe Anhang B.2 auf Seite 77

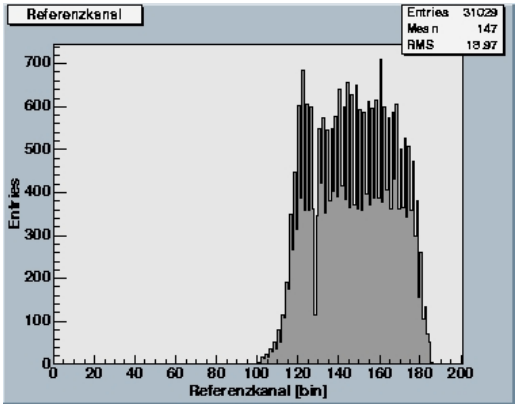

Abbildung 3.23: Selektierte Daten für den Referenzkanal der Driftzeitmessung (Kanal 2). Die Messung wurde mit einer Ru-Quelle durchgeführt.

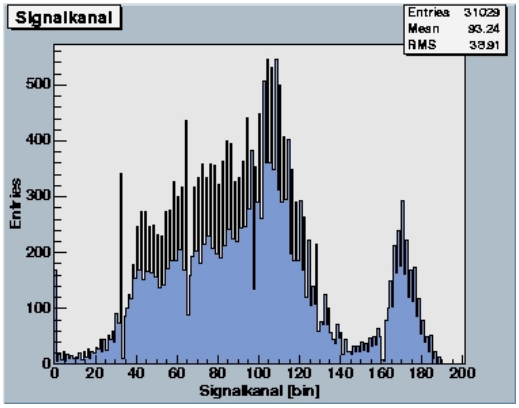

Abbildung 3.24: Selektierte Daten für die Signalkanäle der Driftzeitmessung (Kanal 6 und 7 kombiniert). Die Messung wurde mit einer Ru-Quelle durchgeführt.

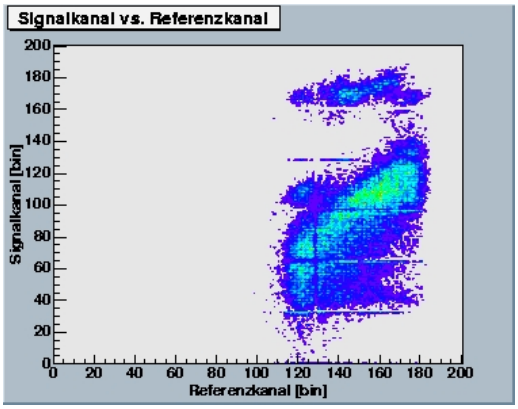

Abbildung 3.25: Signalkanal vs. Referenzkanal für die Driftzeitdaten.

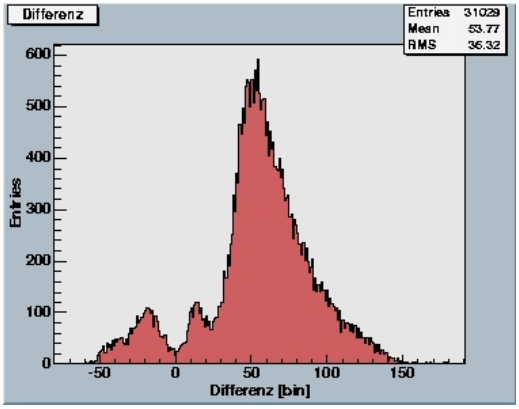

Abbildung 3.26: Die Differenz von Signal und Referenz der Driftzeitmessung.

in den Abbildungen 3.25 und 3.26 auf der vorherigen Seite gezeigt. In diesen Bildern wurden noch keine Schnitte durchgeführt.

Man kann dabei zwei Arten von „Untergrund“ erkennen. Zum einen gibt es im Signalkanal einen Peak um den Bin 170 herum. Wie man dem Konturplot entnehmen kann, sind die zugehörigen Referenzzeiten gleichmässig verteilt. Eine Vermutung ist, dass dieser Peak durch ein taktsynchrones Übersprechen zustande kommt.

Einen zweiten „Untergrund“ bildet das leichte Verschmieren des Referenzsignals. Dies ist der schon früher erwähnte Effekt der unzuverlässig funktionierenden QPLL. Um diese Effekte nun zu umgehen, wurden zwei weitere Schnitte eingeführt:

- Der Wert der Referenz musste grösser als der des Signals sein.
- Der Wert der Referenz muss grösser als 125 sein.

Nach diesen Schnitten blieben noch 23 075 von 31 029 Ereignissen übrig. Die Abbildungen 3.27 und 3.28 zeigen den Referenz- bzw. den Signalkanal nach allen Schnitten. Die Abbildungen 3.29 und 3.30 auf der nächsten Seite zeigen schliesslich die Kontur und die Differenz. Der verbliebene Untergrund in den ersten 20 bis 25 Bins stammt zum grössten Teil aus dem oben erwähnten Peak um das Bin 170. Der steile Anstieg zu Beginn und das langsame Abfallen über 55 ns hinweg ist typisch für ein TDC-Spektrum mit dem Zählgas $ArCO_2$. Das gemessene Spektrum zeigt eine gute Übereinstimmung mit früheren Garfield Simulationen [21]. Die Messung kann daher als ein erfolgreicher Test der Elektronikkomponenten im Zusammenspiel mit einem Modul gewertet werden.

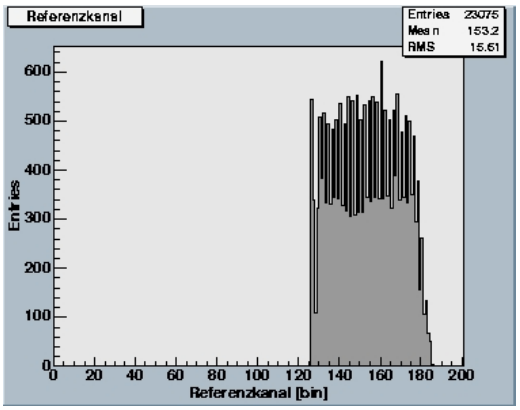

Abbildung 3.27: Einträge im Referenzkanal nachdem alle im Text erwähnten Schnitte durchgeführt wurden.

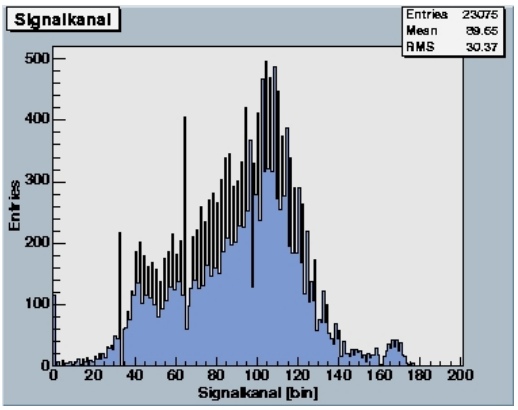

Abbildung 3.28: Einträge im Signalkanal nachdem alle im Text erwähnten Schnitte durchgeführt wurden.

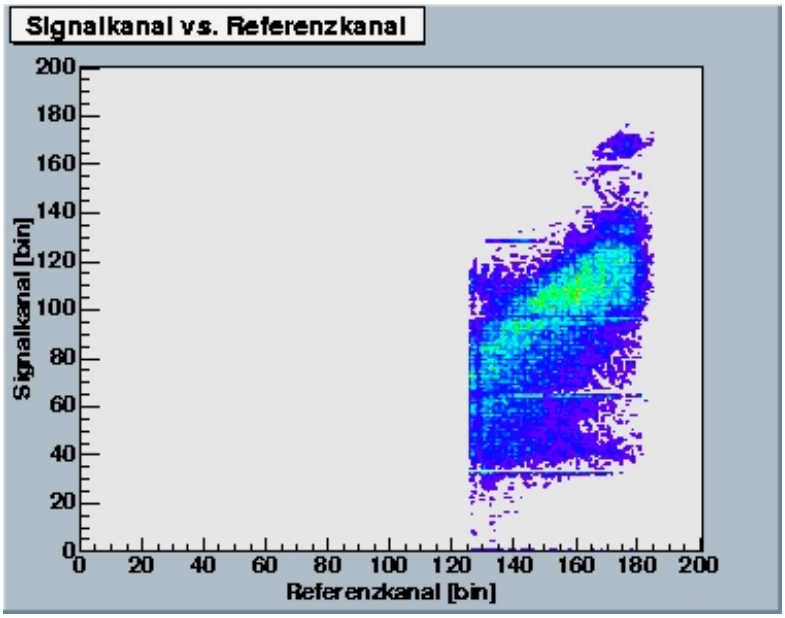

Abbildung 3.29: Referenz gegen Signal nachdem alle im Text erwähnten Schnitte durchgeführt wurden. Die Messung wurde mit einer Ru-Quelle durchgeführt.

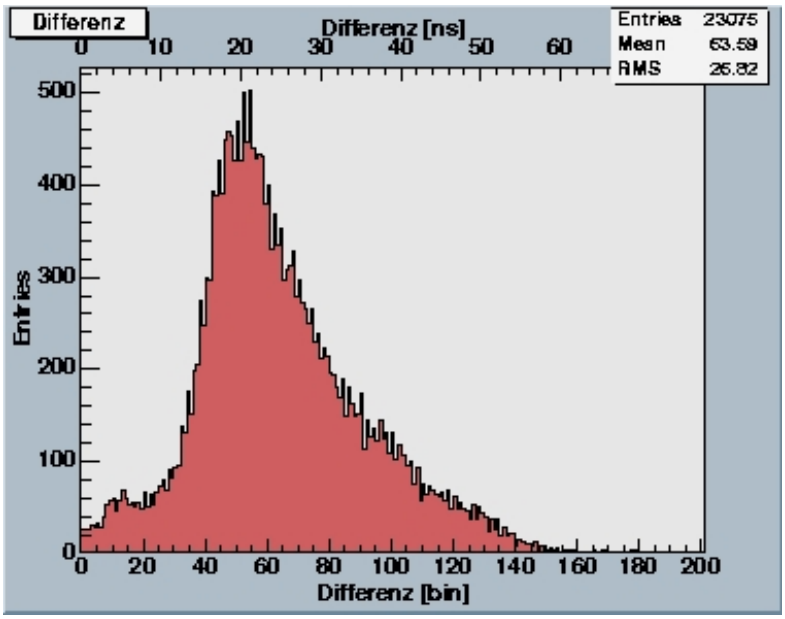

Abbildung 3.30: Driftzeitspektrum nach Durchführung aller Schnitte. In der oberen Achse wurden die TDC-Bins in eine Zeit umgerechnet. Die Messung wurde mit einer Ru-Quelle durchgeführt.

3.6 TDC-Messung mit kosmischen Myonen

Nach der erfolgreichen Durchführung mit der Ru-Quelle wurde die Messung mit kosmischen Myonen wiederholt. Aufgrund der Tatsache, dass nur zwei Straws an den TDC-Chip angeschlossen werden konnten, liefen diese Messungen bei einer geringen Rate. Um dies zu kompensieren, wurde jeweils über einen deutlich längeren Zeitraum ($\sim 10\,\mathrm{h}$) gemessen. Dabei wurden zunächst 20 000 Ereignisse, in einer späteren Messung nocheinmal 30 000 Ereignisse aufgenommen. Diese beiden Datensätze wurden dann vereinigt.

Nachdem die Ereignisse herausgefiltert waren, bei denen Kanal 6 oder 7 angesprochen haben, standen noch 6 550 Ereignisse für die Analyse zur Verfügung. Die Verteilung zeigt Abbildung 3.31.

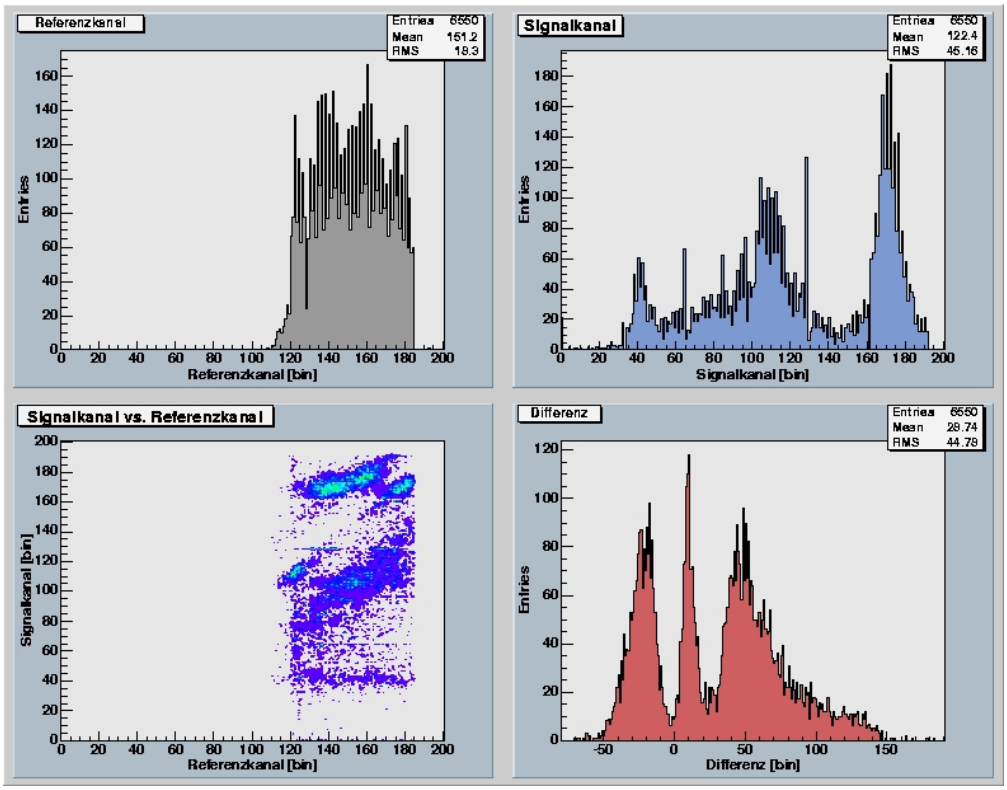

Abbildung 3.31: Verteilung der kosmischen Myonen in den Rohdaten.

Wie man erkennen kann, treten die Untergrundeffekte hier wesentlich stärker zu Tage. Die wahrscheinlichste Erklärung für das Zustandekommen des Untergrundes ist ein taktsynchrones Übersprechen auf die Fädeldrähte, die vom Vorverstärker zum Stecker für den TDC-Chip gehen. Da der Referenztakt für das TTC-System aus der Stratix-Karte stammt, musste eine fast 10 m lange Leitung mit diesem Signal durch das Labor geführt werden. Dabei konnte dieses Signal nicht vollkommen abgeschirmt werden, so dass im gesamten Labor dieses Taktsignal zu messen war. Dieses taktsynchrone Signal ist der

bevorzugte Messwert bei einer Zufallskoinzidenz der beiden Szintillatoren. In der Differenzbildung bilden diese taktsynchronen Signale Peaks bei Werten von kleiner 30 Bins. Mit denselben Schnitten wie im vorherigen Kapitel erhält man die in Abbildung 3.32 gezeigten Histogramme.

Um das Signal der kosmischen Myonen gegenüber dem taktsynchronen Übersprechen deutlicher hervorzuheben, wurde der erste Schnitt modifiziert. Nun wurde verlangt, das die Referenz um 30 Bins grösser als das Signal ist. Damit erhält man das in Abbildung 3.33 auf der nächsten Seite gezeigt Spektrum. In dieses gehen noch 3 113 Ereignisse ein. Dabei zeigt sich das für Driftzeitspektren typische schnelle Ansteigen des Signals. Die Abklingzeit von etwa 55 ns wird durch das verwendete Zählgas in den Driftröhrchen bestimmt.

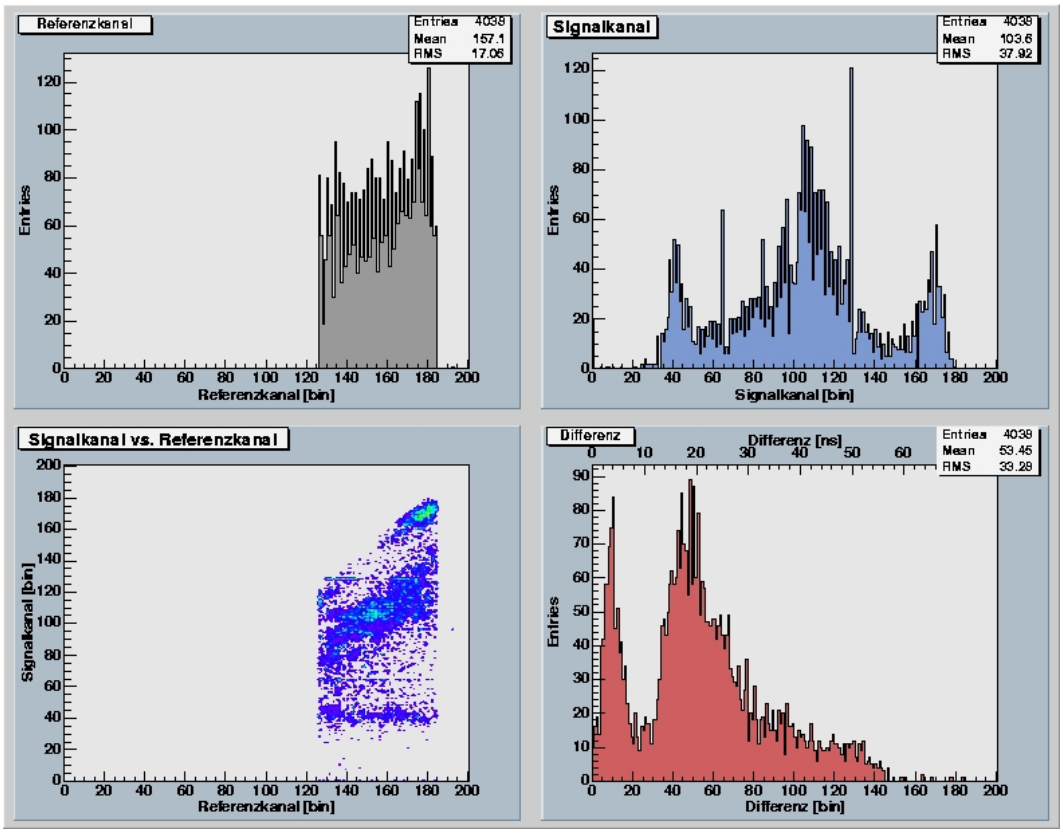

Abbildung 3.32: Verteilung der kosmischen Myonen nachdem die im vorherigen Kapitel vorgestellten Schnitte durchgeführt wurden.

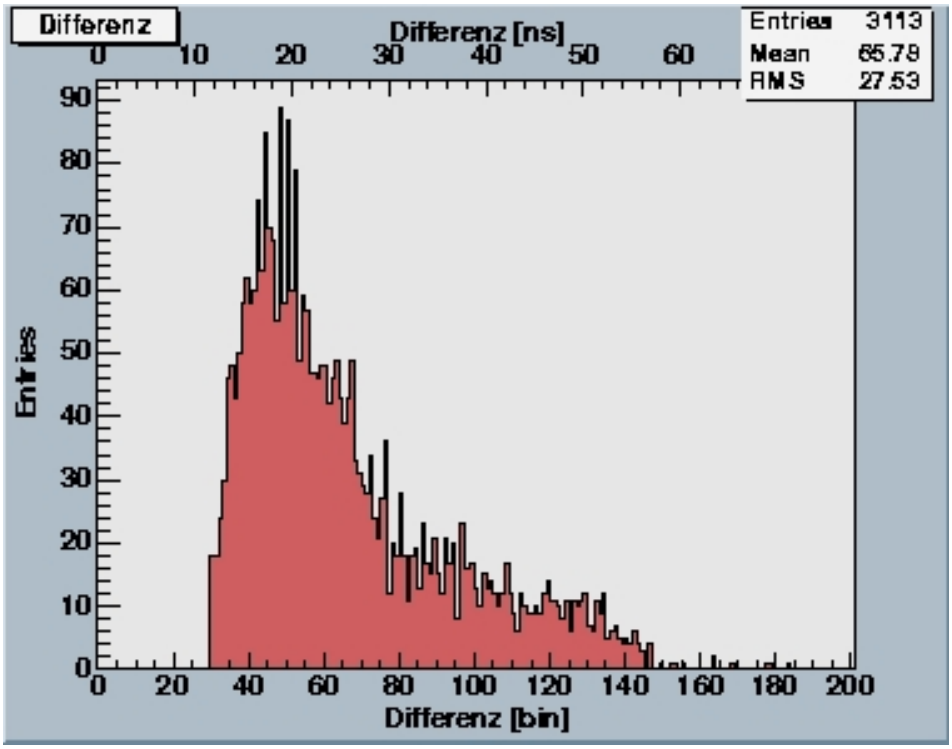

Abbildung 3.33: Driftzeitspektrum mit den im Text vorgestellten Schnitten. In der oberen Achse wurden die TDC-Bins in eine Zeit umgerechnet.

4 Zusammenfassung und Ausblick

4.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde erstmalig eine Auslesekette mit allen Elektronikkomponenten, die für die Auslese der Äusseren Spurkammersystems vorgesehen sind, aufgebaut. Zur Auslesekette gehörte auch die Entwicklung eines FPGA-Programmes, dass die Funktion des L1-Zwischenspeichers emuliert, welches eine Synchronisation auf den Datenstrom durchführt und die Daten zwischenspeichert. Das Programm wurde auf einer Stratix-PCI-Karte implementiert, die eine Auslese der Daten mittels eines PCs erlaubt.

Mit der vollständigen Auslesekette wurden zum ersten Mal die einzelnen Elektronikkomponenten unter realistischen Bedingungen getestet. Hierzu gehörte der OTIS TDC-Chip, der für die Zeitmessung im Äusseren Spurkammersystem verantwortlich ist und somit eine Schlüsselrolle für die Auslese im Detektor einnimmt. Mehrere Otis-Chips der Version 1.0 wurden gleichzeitig über mehrere Wochen hinweg stabil betrieben. Weiter konnte gezeigt werden, dass der OTIS-Chip in der Version 1.1 alle an ihn gestellten Anforderungen erfüllt.

Als Teil der Auslesekette wurde auch die optische Datenübertragung einem realistischen Langzeittest unterzogen.Dabei gefundene Probleme in der Stabilität des Taktsignals konnten auf der optischen Senderkarte lokalisiert werden. Eine neue Version dieser Karte wurde daraufhin entwickelt. Auch die optische Empfängerkarte wurde in diesem Aufbau erfolgreich getestet.

Im weiteren wurde der Aufbau genutzt, um ein Driftkammermodul erstmalig mit der vorgesehenen Elektronik auszulesen. Die aufgezeichneten Driftzeitspektren kosmischer Myonen bzw. einer radioaktiven Ru-Quelle demonstrierten die korrekte Funktion der Auslesekette.

4.2 Integration des TELL1-Design

Das in Kapitel 2 beschriebene Programm ist so geschrieben worden, dass die originalen TELL1-Codes für die PP-FPGAs leicht eingebunden werden können. Der Code für diese FPGAs wurde zusammen mit der TELL1-Karte von Guido Haefeli und Alex Gong [12] in Lausanne entwickelt. Bei meinen Ausführungen beziehe ich mich auf das Programm, welches im April 2004 im Internet zur Verfügung gestellt wurde.

Da der PP-FPGA auch Slow-Control Signale sowie Daten von anderen Subdetektoren verarbeiten können muss, enthält er natürlicherweise eine Vielzahl von logischen

Funktionen. Für eine Integration in das Shippo-Programm ist vor allem der L1-Buffer interessant. Diesen würde man auf folgende Weise integrieren:

Die Synchronisation wird aus dem Shippo-Programm entnommen. Dessen prozessierte Daten werden in das vom TELL1 benötigte Datenformat umgewandelt. Die Daten müssen dabei synchron zum 40 MHz-Takt kommen. Danach kann der gesamte Datenpfad durch den L1-Buffer bis zur Ausgabe an den SyncLink-FPGA übernommen werden. Dieser Teil würde den Fifo-Block des Shippo-Programms ersetzen. Statt der Ausgabe zum SyncLink-FPGA kommt dann der PCI-Block des Shippo-Programms zum Einsatz. Die Hauptarbeit bei einer solchen Änderung läge in der richtigen Steuerung der Kontrollsignale des L1-Buffers.

4.3 Lehren aus dem Shippo-Programm

Umgekehrt wurden aus dieser Arbeit auch Lehren für die Entwicklung einer Synchronisation der Outer-Tracker-Daten im TELL1 gezogen. So hat sich im Verlauf der Messungen deutlich gezeigt, dass nicht nur ein Merkmal der Daten für die Synchronisation verwendet werden sollte. Die OTIS-ID, die im Shippo-Programm zur Synchronisation verwendet wurde, wäre im Experiment nicht praktikabel. Viel besser sind folgende Signale geeignet[1]:

- das Komma zu Beginn eines neuen Datenstroms
- das Data-Valid-Signal
- der Eventzähler
- die BX-ID

Dabei würde man in einem ersten Schritt alle ankommenden Daten von verschiedenen OTIS-Chips mit Hilfe des Kommas und des Data-Valid-Signals auf einen gemeinsamen „Startpunkt“ synchronisieren. Anschliessend kann man anhand des Eventzählers und der BX-ID Ereignisse, die zur selben Wechselwirkung gehören, zusammenfügen und sie dem L1-Buffer übergeben. Ein erster Versuch einer solchen Implementierung wurde im Rahmen einer Diplomarbeit [23] unternommen.

4.4 Weitere Tests der Ausleseelektronik

Zum weiteren Testen der Elektronikkomponenten soll im Januar 2005 ein Test am DESY in Hamburg durchgeführt werden. Dieser wird die Eigenschaften der Ausleseelektronik in einem Aufbau untersuchen, der zur Spurrekonstruktion geeignet ist. Dabei sollen vier 2,5 m-Module in einem Elektronenstrahl getestet werden. Eventuell werden noch zwei kleinere Module um 90° gedreht zu diesen aufgebaut, um auch in dieser Richtung eine Positionsbestimmung durchführen zu können.

[1] seit dem OTIS 1.1 sind diese Signale implementiert

Aus Sicht der L1-Buffer-Elektronik ist vor allem interessant, die gleichzeitige Auslese mehrerer Module testen zu können. Für diesen Test werden zunächst einmal sechs Frontendboxen benötigt. Die Frontendbox werden mit den OTIS-Chips der Version 1.1 und der zugehörigen Karte ausgestattet sein. Diese werden zur Zeit am NIKHEF in Amsterdam produziert. Eine Frontendbox soll weiterhin in Heidelberg mit den neuen OTIS 1.2 bestückt werden.

Das TTC-System kann komplett aus den Aufbauten für die Messungen in dieser Diplomarbeit übernommen werden. Auch die Slow-Control-Steuerung wurde im Rahmen eines Miniforschungsprojekts [22] an die Verwendung mehrerer Frontendboxen angepasst. Die Signale aus beiden System werden von einer Verteilerbox zentral an alle Frontendboxen verteilt.

Die Daten aller verwendeten Frontendboxen werden dann über optische Fasern auf eine optische Empfängerkarte geführt. Von dort werden dann die Signale an eine Stratix-PCI-Karte geschickt. Dies bietet die Möglichkeit, eine erste Implementierung der Synchronisation mehrerer Links zu testen, wie sie im vorherigen Kapitel beschrieben wurde.

Die vielfältigen Erfahrungen und Probleme, wie sie beim Aufbau einer Auslesekette mit einem OTIS-Chip im Labor aufgetreten sind, belegen deutlich, dass solche Untersuchungen zum Testen neu entwickelter Hardware unbedingt erforderlich sind.

A Shippo-Programm

A.1 I/O-Ports des Synchronisationsblocks

Name	Breite	I/O	Kurzbeschreibung
LHCb_clk	1 Bit	Input	Systemtakt
Memory_full	1 Bit	Input	Anzeige, daß das Fifo voll ist.
Reset	1 Bit	Input	Resetsignal
OTIS_ID	4x 12 Bit	Input	OTIS-IDs, auf die synchronisiert wird
OTIS_ID_non_zero	1 Bit	Input	Schaltet „Non_zero Modus“ aus/an
OTIS_Version	2 Bit	Input	verwendete OTIS-Version
RxClock	1 Bit	Input	Takt, mit der die Daten empfangen werden
RxData	16 Bit	Input	Daten von der Auslesekette
RxDV	1 Bit	Input	Gibt an, ob die Daten gültig sind
RxErr	1 Bit	Input	Gibt an, ob ein Fehler auftrat
stop_readout	1 Bit	Input	Synchronisation anhalten
BX_number_out	8 Bit	Output	BX-ID des Events, welches ins Fifo übertragen wird
Data_out	32 Bit	Output	Daten für das Fifo
event_complete	1 Bit	Output	Markiert das Ende eines Events
event_error_out	1 Bit	Output	Fehleranzeige
event_frame_out	1 Bit	Output	Rahmen für Eventdaten
OTIS_ID_out	12 Bit	Output	OTIS-ID des Events, welches ins Fifo übertragen wird.

Tabelle A.1: I/O-Ports des Synchronisationsblocks

A.2 I/O-Ports des Fifoblocks

Name	Breite	I/O	Kurzbeschreibung
Data_in	32 Bit	Input	Daten vom Synchronisationsblock
event_complete	1 Bit	Input	Markiert das Ende eines Events
event_error	1 Bit	Input	Fehleranzeige
event_frame	1 Bit	Input	Rahmen für Eventdaten
Fifo_rdreq	1 Bit	Input	PCI-Bus fordert ein Event an
LHCb_clk	1 Bit	Input	Systemtakt
PCI_clk	1 Bit	Input	Takt des PCI-Busses
Reset	1 Bit	Input	Resetsignal
Data_out	32 Bit	Output	Daten für PCI-Block
Fifo_full	1 Bit	Output	Fifo voll
no_of_events_in_fifo	16 Bit	Output	Anzahl gespeicherter Events

Tabelle A.2: I/O-Ports des Fifoblocks

A.3 I/O-Ports der PCI-Blöcke

Bemerkung : Signale, deren Namen auf „n“ enden, sind Active Low				
	Bit-	Input/Output		
Name	breite	Interface	Megacore	Kurzbeschreibung
Fifo_data	32	Input	—	Daten aus dem Fifo
LED	8	Input	—	s. Register
Occupancy	10	Input	—	s. Register
Reset	1	Input	—	Resetsignal
no_of_events_in_fifo	16	Input	—	s. Register
Bit_order_select	2	Output	—	s. Register
config_reset	1	Output	—	Register geändert oder Reset
connected_otis_chips	1	Output	—	s. Register
dead_time	16	Output	—	s. Register
dummy_readout	1	Output	—	s. Register
dummy_readout_rate	1	Output	—	s. Register
Fifo_rdreq	1	Output	—	Daten vom Fifo anfordern
Latency	8	Output	—	s. Register
LED_sel	1	Output	—	s. Register
no_of_channels	10	Output	—	s. Register
Occupancy	10	Output	—	s. Register
OTIS_IDs	4x 12	Output	—	s. Register
OTIS_IDs_non_zeros	6	Output	—	s. Register
OTIS_versions	2	Output	—	s. Register
stop_readout	1	Output	—	s. Register
Szin_enable	3	Output	—	s. Register
Trigger_length	4	Output	—	s. Register
Clk	1	Input	Input	PCI-Bustakt
cmd_reg	6	—	Output	Command-Register
stat_reg	6	—	Output	Statusregister
l_adro	64	Input	Output	angelegte Adresse auf PCI-Bus
l_beno	8	Input	Output	Byte enable vom PCI-Bus
l_cmdo	4	Input	Output	Befehle vom PCI-Bus
l_dato	64	Input	Output	Daten vom PCI-Bus
l_hdat_ackn	1	Input	Output	Acknowledge fürs High DWord
l_ldat_ackn	1	Input	Output	Acknowledge fürs Low DWord
lt_ackn	1	Input	Output	Acknowledge vom Megacore
lt_dxfrn	1	Input	Output	Daten erfolgreich übertragen
lt_framen	1	Input	Output	Rahmen für Transaktion
lt_tsr	12	Input	Output	Status Register des Megacore
l_adi	64	Output	Input	Daten zum PCI-Bus
lirqn	1	Output	Input	Interrupt-Befehl
... weiter auf der nächsten Seite				

Bemerkung : Signale, deren Namen auf „n“ enden, sind Active Low				
	Bit-	Input/Output		
Name	breite	Interface	Megacore	Kurzbeschreibung
lt_abortn	1	Output	Input	Abort-Befehl
lt_discn	1	Output	Input	Disconnect-Befehl
lt_rdyn	1	Output	Input	Ready-Signal vom Interface
*** Ab hier folgen die PCI-Bus Signale ***				
cben	8	—	Input	Befehls- und Byte-Enable-Bus
framen	1	—	Input	Rahmen für Transaktionen
idsel	1	—	Input	Initialization Device Select
irdyn	1	—	Input	Initiator ready
req64n	1	—	Input	64-Bit Transfer-Request
rstn	1	—	Input	Reset
ad	64	—	In/Out	Daten- und Adreßbus
par	1	—	In/Out	Parity
par64	1	—	In/Out	Parity 64-Bit-Transfer
ack64n	1	—	Output	Acknowledge 64-Bit-Transfer
devseln	1	—	Output	Device select
intan	1	—	Output	Interrupt A
perrn	1	—	Output	Parity error
serrn	1	—	Output	System error
stopn	1	—	Output	Stop
trdyn	1	—	Output	Target ready

Tabelle A.3: I/O-Ports der PCI-Blöcke

A.4 Register

Die Register bilden die zentrale Steuereinheit. Mit ihnen wird das gesamte Verhalten festgelegt. Im folgenden werden sämtliche Register beschrieben. Sofern nicht anders gekennzeichnet, sind die Register sowohl beschreibbar als auch lesbar.

- Register, die die Synchronisation beeinflussen (siehe auch Kapitel 2.2 auf Seite 21).

  OTIS_ID_x_y : Abgelegt werden hier die IDs der OTIS-Chips, die an dem Link angeschlossen sind. x steht für die Nummer des Links, y für die OTIS-Chips null bis drei dieses Links. Nur x = 0 wird verwendet, x = 1 bis 5 sind Platzhalter für eine Erweiterung auf sechs Links. Alle x-y-Kombinationen sind jeweils 12 Bit breit.

Register	Anfangswert
OTIS_ID_0_0	= 0x01
⋮	⋮
OTIS_ID_8_3	= 0x24

  OTIS_id_non_zero : Bit null legt fest, ob zur Synchronisation eine OTIS_Id im Bitstrom gefunden werden muß. Ist der Wert „1“, so reicht ein $\neq$ Null im Bitstrom. Beim Wert „0“ wird eine der in OTIS_ID_0_y angegebenen IDs verlangt. Bit eins bis fünf sind wieder Platzhalter für eine Erweiterung auf sechs Links. Siehe auch Tabelle 2.2 auf Seite 22 . *Anfangswert* ist „0“.

  OV_x : (= OTIS Version). Da sich die verschiedenen OTIS-Versionen unterschiedlich verhalten, muß dies natürlich berücksichtigt werden. Folgende Tabelle gibt die möglichen Werte für dieses Register an:

Wert	Beschreibung
„00“	OTIS Version 1.0
„01“	OTIS Version 1.1
„10“	reserved
„11“	reserved

  Wiederrum ist nur das Register für den Link 0 angeschossen. Siehe auch Kapitel 1.5 auf Seite 10. *Anfangswert* ist „00“.

- Register für die Auslese (siehe auch Kapitel 2.5 auf Seite 36).

  PCI_Dummy_Header : Header für die Ausgabe im PCI-Dummy-Mode. Siehe auch *Statusbits*. *Anfangswert* ist „0xDEADFACE“.

  PCI_Dummy_Data_(0-7) : Daten für die Ausgabe im PCI-Dummy-Mode. Siehe auch *Statusbits*.

Register	Anfangswert
PCI_Dummy_Data_0	= 0xDEAD0000
⋮	⋮
PCI_Dummy_Data_7	= 0xDEAD0007

Dummy_Readout_Event_rate : Hier kann man einstellen, mit welcher Rate die Daten im Dummy-Readout-Modus produziert werden. *Anfangswert* ist „0“ (entspricht einer Triggerrate von 1,11 MHz = maximal Wert). Die Formel für den einzutragenden Wert lautet:

$$\text{Wert} = \left\lfloor \frac{10.000.000}{\text{Rate in Hz}} \right\rfloor - 9 > 0$$

- Statusanzeigen

  Design_Version : Nur *lesbar*. Versionsnummer des Designs. (s.a. Tabelle 2.1 auf Seite 17.)

  events_in_memory : Nur *lesbar*. Gibt die Anzahl der auslesbaren Ereignisse im Fifo an. Damit die Auslese über den PCI-Bus reibungslos verläuft, ist der angegebene Wert eins niedriger als der tatsächliche.

  LED : Nur *lesbar*. Spiegel der LED Anzeige. (s.a. Kapitel A.5 auf Seite 74.)

  Statusbits : Nur *lesbar*. Können mittels Befehle geändert werden. *Anfangswert* ist „0x41“. Die einzelnen Bits haben folgende Bedeutung:

Bit	Name	Beschreibung
0	PCI_dummy_mode	Schaltet den PCI-Dummy-Mode an/aus.
1	stop_readout	Startet/Stoppt die Datennahme.
2	dummy_readout	Schaltet den Dummy-Readout-Modus an/aus.
3	reserved	
4-6	LED_sel(0:2)	Wählt die LED Ausgabe aus. (s.a. Tabelle A.4).
7	reserved	

  Befehlsregister : Nur *beschreibbar*. An dieses Register können Befehle geschickt werden. Tabelle A.5 auf Seite 73 listet alle Befehle auf. Ein Befehl kann gesendet werden, in dem man den „Wert“ des Befehls an die Adresse 1023 des BAR_0 schreibt.

- Sonstige (siehe auch Kapitel A.5 auf Seite 74).

  Szin_trigger dead time : Gibt die Anzahl der LHCb_clk Zyklen an, die nach einem Trigger vergehen müssen, bis ein neuer akzeptiert wird. *Anfangswert* ist „0x0400“.

  Szin_trigger Latency : Gibt die Verzögerung des ausgehenden Triggers zum eingehenden in LHCb_clk-Zyklen an. *Anfangswert* ist „0“.

Szin_trigger length : Gibt die Länge des ausgehenden Triggersignals in LHCb_clk-Zyklen an. *Anfangswert* ist „8“.

Szin_enable : Entscheidet, ob die Szin_trigger_(1-3) Signale zur Triggerbildung benutzt werden oder nicht. „1“ bedeutet ja, „0“ nein. *Anfangswert* ist „7“.

Occupancy : Nur *lesbar*. Gibt die „Occupancy“ der letzten 4096 „Events“ an, die im Fifo gespeichert wurden.

connected_otis_chips : Anzahl angeschlossener OTIS Chips. Wird zur Berechnung der Occupancy benötigt. *Anfangswert* ist „4“.

no_of_channels : Anzahl der ausgelesenen Kanäle. Wird zur Berechnung der Occupancy benötigt. *Anfangswert* ist „0x80“.

BO_x : (= Bit order) Manipulation des Eingangsdatenstroms. Mit diesem Register können die einzelnen Bytes auf *big-endian* oder *little-endian* eingestellt werden. x steht wieder für die Nummer des Links. Nur x = 0 ist verwendet, x = 1 bis 5 sind Platzhalter für eine Erweiterung auf sechs Links. *Anfangswert* ist „00“. Die nachfolgende Tabelle zeigt die Einstellungen:

BO_0	RxData [0:15]
„00“	= RxData [0:15]
„01“	= RxData [7:0] + RxData [8:15]
„10“	= RxData [0:7] + RxData [15:8]
„11“	= RxData [15:0]

Die folgende Tabelle enthält eine Auflistung der Anzeige auf den LEDs, in Abhängigkeit vom Register LED_sel.

LED sel	LED-Anzeige	
	Bit[0:5]	Bit[6:7]
000	stat_reg des PCI-Megacore	vdd
001	cmd_reg des PCI-Megacore	vdd
010	RxData [8:15]	
011	RxData [0:7]	
100	no_of_events_in_fifo [0:7]	
101	no_of_events_in_fifo [8:15]	
110	BX_number_out[0:7] vom Synchronisations-Block	
111	OTIS_ID_out[0:7] vom Synchronisations-Block	

Tabelle A.4: LED Ausgabe in Abhängigkeit von LED_sel

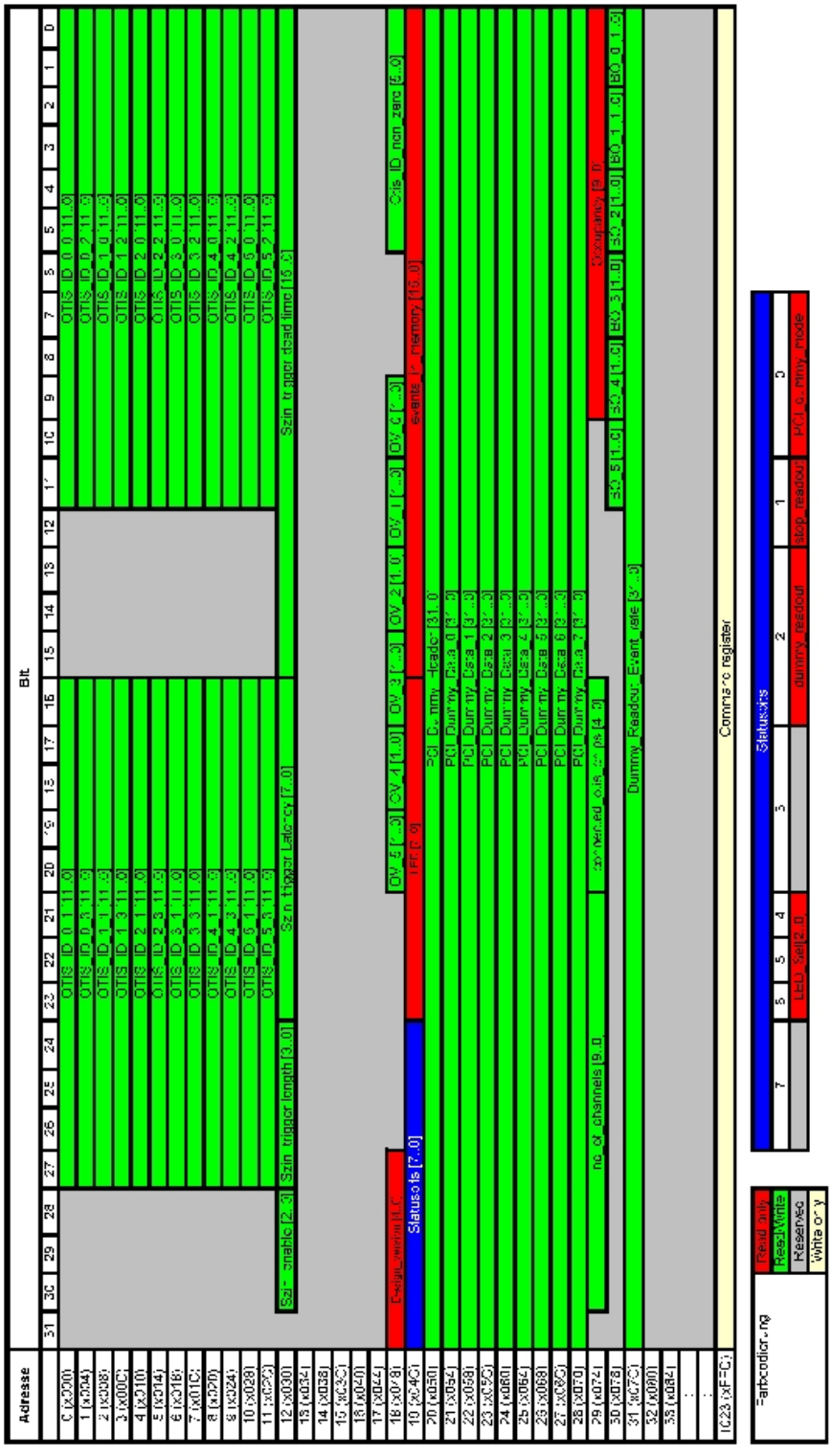

Abbildung A.1: Konfigurations Register

Die nächste Tabelle enthält alle Befehle, die an das Befehlsregister gesendet werden können. Ist kein Wert in einer Spalte angegeben, so ändert der Befehl nichts an dem bisherigen Wert. Die Resetspalte gibt an, ob nach dem Befehl ein Reset durchgeführt wird. Dies löscht das Fifo und bricht gleichzeitig laufende Synchronisationsevents ab. Außerdem werden alle Zustandsautomaten im Synchronisationsblock gezwungen, mit den gesetzten Werten neu zu starten.

Wert	stop readout	D.R.M.	PCI D.M.	LED sel	Reset	Beschreibung
00	-	-	-	—	–	no command
01	1	0	0	100	ja	User Reset
02	-	-	0	—	ja	clear pci dummy mode
03	1	0	1	—	ja	set pci dummy mode
04	0	-	-	—	–	clear stop readout
05	1	-	-	—	–	set stop readout
06	-	-	-	—	ja	Reset Fifo
07	1	0	0	—	ja	set init values
08	-	-	-	000	–	LED_ sel = 0
09	-	-	-	001	–	LED_ sel = 1
0A	-	-	-	010	–	LED_ sel = 2
0B	-	-	-	011	–	LED_ sel = 3
0C	-	-	-	100	–	LED_ sel = 4
0D	-	-	-	101	–	LED_ sel = 5
0E	-	-	-	110	–	LED_ sel = 6
0F	-	-	-	111	–	LED_ sel = 7
10	-	0	-	—	ja	clear dummy readout
11	1	1	0	—	ja	set dummy readout
other	-	-	-	—	–	not used

D.R.M. = Dummy-Readout-Mode
PCI D.M. = PCI-Dummy-Mode

Tabelle A.5: Befehlsübersicht

A.5 Sonstige Blöcke

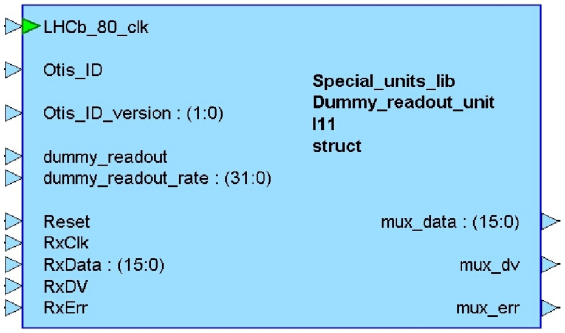

Abbildung A.2: Dummy-Readout-Block

Der Dummy-Readout-Block erfüllt zwei Funktionen. Zum einen nimmt er die Daten von der O-RxCard entgegen und führt einen Taktwechsel von der RxClk auf die LHCb_80_clk durch. Zum anderen generiert der Block auch die Daten für den Dummy-Readout-Modus.

Der Bit-Order-Block ermöglicht eine Manipulation am Bitstrom. Die genaue Arbeitsweise ist in den entsprechenden Registern in Kapitel A.4 auf Seite 69 beschrieben.

Der Szintilatortriggerblock generiert aus drei Eingängen einen Trigger. Dabei kann eingestellt werden, welche Eingänge zum Trigger beitragen, wie lang und nach welcher Zeit der generierte Trigger ausgegeben wird, sowie die „Totzeit“ nach einem Trigger. Einstellbar ist dies über die entsprechenden Register.

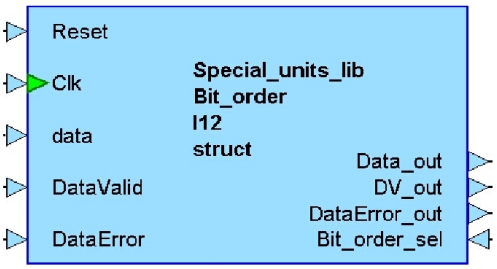

Abbildung A.3: Bit-Order-Block

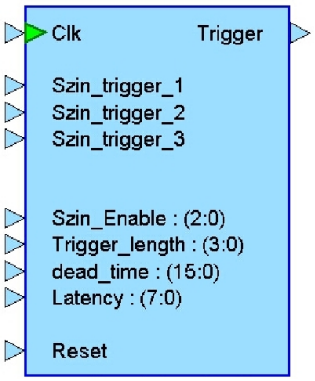

Abbildung A.4: Szintilatortriggerblock

B ROOT-Skripte

B.1 Datenfilter

```
{
  // Standardeinstellungen
  #include "Riostream.h"

  gROOT->Reset();

  // Steuerungsparameter
  UInt_t otis_id = 4;    // OTIS-ID = 004

  // Programmparameter
  UInt_t otis_adr;       // ausgelesene OTIS-Adresse
  Int_t channel[33];     // Hilfsfeld
  bool take_event;       // soll Event genommen werden?
  int event_error;       // Anzahl der Fehler im Event
  ifstream in;           // Input-File
  ofstream out;          // Ouput-File
  Int_t Dummy;           // Dummy-Container
  // File öffnen
  out.open("<filename>_pure.txt");
  in.open("<filename>.txt");

  while (true) {
    in >> std::hex >> otis_adr;      // erster Wert = Header
    if (!in.good()) break;           // kein Input -> File zu Ende
    take_event = false;
    event_error = 0;
    //OTIS ID extrahieren und vergleichen
    if (((otis_adr & 0xFFF00000) >>20)  == otis_id) {
      for (int f = 0; f < 32; f++){  // 32 Driftzeiten lesen
        in >> std::hex >> channel[f];
        if (!in.good()) break;       // kein Input -> File zu Ende
        // Kanal 6 oder 7 = Signal ?
        if (((f == 7) || (f == 6)) && (channel[f] != 192))
```

```
34                take_event = true;
35            // Event kaputt ?
36            if ((f > 15) && (channel[f] != 192))
37                event_error++;
38          }
39          // Alles ok? -> Speicher Event
40          if (take_event && (event_error < 3)) {
41            for (int g = 0;g < 32; g++) out << channel[g] << " ";
42            out << endl;
43          }
44        }
45        // Event mit falschem Header
46        else {
47          for (int f = 0; f < 32; f++){  // 32 Driftzeiten lesen
48            in >> std::hex >> Dummy;
49            if (!in.good()) break;       // kein Input -> File zu Ende
50          }
51        }
52      }
53
54      // File schließen
55      in.close();
56      out.close();
57    }
```

B.2 Auswertung

```
1  {
2    // Standardeinstellungen
3  #include "Riostream.h"
4
5    gROOT->Reset();
6    gStyle->SetOptStat(1110);
7    gStyle->SetPalette(1);
8    gStyle->SetCanvasColor(33);
9    gStyle->SetFrameFillColor(18);
10
11   // Steuerungsparameter
12   Int_t ref_channel = 2;            // Referenzkanal
13   Int_t time_channel_0 = 6;         // Zeitkanal 1
14   Int_t time_channel_1 = 7;         // Zeitkanal 2
15
16   // Programmparameter
17   Int_t value;                      // Hilfsvariable
18   Int_t referenz;                   // Referenzzeit
19   Int_t signal;                     // Signalzeit
20   Int_t TDC_signal;                 // Differenz ref,sig
21   Int_t ref_high = 200, sig_high = 200; // obere Bingrenze
22   Int_t ref_low  = 0, sig_low = 0;  // untere Bingrenze
23   Int_t refbins = 1 + ref_high - ref_low; // Anzahl Referenz Bins
24   Int_t sigbins = 1 + sig_high - sig_low; // Anzahl Signal Bins
25   Int_t diff_low = 0, diff_high = 200;  // Bingrenzen Differenz
26   Int_t diff_bins = 1 + diff_high - diff_low; // Anzahl Differenzbins
27   bool input;                       // Schnittvariable
28   bool file = true;                 // File noch nicht zu Ende
29   ifstream in;                      // input ascii file
30   Int_t cancolor = 17;              // root setting
31   Int_t i,j,tmp;
32   Int_t baseline = 0;
33
34   Int_t diff[diff_high - diff_low + 2]; // Hilfsfeld
35
36   // 1d Histogramme
37   TH1S hist_ref("ref_hist","Referenzkanal",refbins,ref_low,
38                 ref_high + 1);
39   TH1S hist_time("time_hist","Signalkanal",sigbins,sig_low,
40                  sig_high + 1);
41   TH1S hist_TDC("tdc_hist","Differenz",diff_bins,diff_low,
42                 diff_high + 1);
```

```
43    // 2d Histogramm
44    TH2S hist_2d("hist_2d","Signalkanal vs. Referenzkanal",refbins,
45                 ref_low,ref_high+1,ybins,sig_low,sig_high+1);
46
47    // Aussehen der Histogramme
48    hist_ref.SetFillColor(15);
49    hist_time.SetFillColor(38);
50    hist_TDC.SetFillColor(46);
51    hist_2d.SetFillColor(46);
52    hist_2d.SetStats(kFALSE); // keine Statistik anzeigen
53
54    // File öffnen
55    in.open("<filename>.txt");
56
57    for(i=0; i < (diff_high - diff_low + 2); i++) diff[i] = 0;
58
59    while (file) {
60      input = true;
61
62      for (int f = 0; f < 32; f++){  // 32 Driftzeiten lesen
63         in >> value;
64         if (!in.good()) {
65           file = false;
66           break;       // kein Input -> File zu Ende
67         }
68         if (f == diff_channel) referenz = value;
69         if (f == time_channel_0) signal = value;
70         if (f == time_channel_1) tmp = value;
71      }
72
73      if ((tmp != 192) && (signal != 192)) input = false;
74
75      if (signal == 192) signal = tmp;
76
77      // Schnitte
78      //    if (referenz < 126) input = false;
79      //    if (referenz < signal) input = false;
80      //    if ((referenz < 120) || (referenz > 190)) input = false;
81      //    if ((referenz < 117) || (referenz > 186)) input = false;
82      //    if ((referenz < signal+30)) input = false;
83      //    if (signal != 128) input = false;
84
85      if (input) {
86        hist_2d.Fill(referenz,signal,1); // Histogramme füllen
```

```
87          hist_ref.Fill(referenz,1);
88          hist_time.Fill(signal,1);
89          tmp = referenz - signal;
90          if ((tmp >= diff_low) && (tmp <= diff_high))
91              diff[tmp - diff_low]++;
92        }
93      }
94
95      for (i = 0; i < (diff_high - diff_low + 2); i++) {
96          tmp = diff[i];
97
98          if (tmp < baseline) tmp = 0;
99          else tmp -= baseline;
100
101         for (j = 0; j < tmp; j++) hist_TDC.Fill(i+diff_low,1);
102     }
103
104     // File schliessen
105     in.close();
106
107     // Fenster öffnen, 2x2 Feld anlegen
108     TCanvas *TDC = new TCanvas("TDC","Ref vs. Time",1280,1024);
109     TDC.Divide(2,2);
110     TDC.SetFillColor(cancolor);
111
112     // Links oben kommt der Referenzkanal
113     TDC.cd(1);
114     hist_ref->GetXaxis()->SetTitle("Referenzkanal [bin]");
115     hist_ref->GetYaxis()->SetTitle("Entries");
116     hist_ref->GetXaxis()->CenterTitle();
117     hist_ref->GetYaxis()->CenterTitle();
118     hist_ref->GetYaxis()->SetTitleOffset(1.15);
119     hist_ref.Draw();
120
121     // Rechts oben kommt der Signalkanal
122     TDC.cd(2);
123     hist_time->GetXaxis()->SetTitle("Signalkanal [bin]");
124     hist_time->GetYaxis()->SetTitle("Entries");
125     hist_time->GetXaxis()->CenterTitle();
126     hist_time->GetYaxis()->CenterTitle();
127     hist_time->GetYaxis()->SetTitleOffset(1.15);
128     hist_time.Draw();
129
130     //Links unten beide Kanale gegeneinader als Kontur
```

```
131     TDC.cd(3);
132     hist_2d->GetXaxis()->SetTitle("Referenzkanal [bin]");
133     hist_2d->GetYaxis()->SetTitle("Signalkanal [bin]");
134     hist_2d->GetXaxis()->CenterTitle();
135     hist_2d->GetYaxis()->CenterTitle();
136     hist_2d->GetYaxis()->SetTitleOffset(1.15);
137     hist_2d.Draw("cont");
138
139     // Rechts unten die Differenz von beiden
140     TDC.cd(4);
141
142     hist_TDC->GetXaxis()->SetTitle("Differenz [bin]");
143     hist_TDC->GetYaxis()->SetTitle("Entries");
144     hist_TDC->GetXaxis()->CenterTitle();
145     hist_TDC->GetYaxis()->CenterTitle();
146     hist_TDC->GetYaxis()->SetTitleOffset(1.15);
147     hist_TDC.Draw();
148
149     TDC->Update();
150
151     Float_t axis_4_low = (Float_t) diff_low;
152     Float_t axis_4_high = (Float_t) diff_high;
153
154     axis_4_low *= .39;
155     axis_4_high *= .39;
156
157     TGaxis *axis_4 = new TGaxis(TDC_4->GetUxmin(),TDC_4->GetUymax(),
158                                 TDC_4->GetUxmax(), TDC_4->GetUymax(),
159                                 axis_4_low,axis_4_high,510,"-");
160
161      axis_4->SetTitle("Differenz [ns]");
162      axis_4->CenterTitle();
163      axis_4->SetTitleOffset(-1.15);
164      axis_4->Draw("same");
165
166      TPaveStats *st_4 = (TPaveStats*)
167          hist_TDC->GetListOfFunctions()->FindObject("stats");
168      st_4->Draw("same");
169   }
```

C Change Logs

```
Shippo v1.01:
  28.05.04:
        * Clock divider unit eingefuegt.
  01.06.04:
        * Trigger unit eingefuegt
        * Design Version ist nun 4.
Shippo v1.00:
  11.05.04:
        * Fix eines Fehlers, der bei vollem Fifo zum Verlust der
          Synchronisation fuehren kann.
        * Vereinfachung der State-Machine, die die Synchronisation
          durchfuehrt.
        * Design Version ist nun 3.
        * Die Defaultwerte fuer folgende Register haben sich geaendert:
                - OTIS_id_non_zero sind nun = 0.
                - LED_sel ist nun = "100"
        * folgende Befehle haben sich geaendert:
                Datenwort       Befehl
                3               pci_dummy_mode = 1.
                                -> stop_readout = 1
                                -> dummy_readout = 0    (!)
                7               Clear Registers.
                                -> pci_dummy_mode = 0
                                -> stop_readout = 1     (!)
                                -> dummy_readout = 0    (!)
                11              dummy_readout = 1
                                -> pci_dummy_mode = 0
                                -> stop_readout = 1     (!)
        * kleinerer Bugfix im Dummy-Readout-Mode.
  12.05.04:
        * Event Buffer vergroessert.
  13.05.04:
        * Vereinfachung des Datenpfades.
        * Austausch der Unit Event_collector durch Event_sender.
        * Vereinheitlichung der Fifos.
  17.05.04:
```

```
        * Bit_order unit eingefuegt.
  18.05.04:
        * Unit event_builder durch event_converter ersetzt.
        * Erweiterung der Bit_order unit.
  24.05.04:
        * erneutes Redesign der Occupancyberechnung.
        * Vereinfachung des Memory Fifos
                -> Jetzt ~ 3600 Event speicherbar
Shippo v1.00 RC 1:
  22.04.04:
        * Fix eines Bugs der unter Umstaenden falsche Werte in das
          event_in_memory register geladen hat.
                -> kleine Reduzierung des Fifos auf 3562 Events.
        * neue Befehle eingefuehrt:
                Datenwort       Befehl
                6               Config reset : Loescht Fifos,
                                               update aller register
                7               Clear pci_dummy_mode & stop_readout
                10              dummy_readout = 0
                11              dummy_readout = 1
                                -> pci_dummy_mode = 0
                                -> stop_readout = 0
        * Befehle 2,3,7,10,11 fuehren nun auch einen config_reset aus.
        * Dummy-Readout-Modus hinzugefuegt.
          Legt auf Board generierte Daten auf RxDataeingang.
          Bestehen aus OTISID gefolgt von einem Counter.
        * default Einstellungen des Statusbytes nicht mehr ueber
          Dipswitch. ( Statusbyte = 2 beim aufwachen/User Reset)
        * diverse kleinere Bugfixe im Synchronizer Teil.
        * Fix eines schweren Fehlers bei der Uebertragung ins Fifo,
          der zum Verlust des letzten Datenworts fuehren kann.
  23.04.04:

        * Einstellbare Eventrate im Dummy-Readout-Modus.
          -> Formel:
                          160.000.000 x 36
                        ------------------
                        Events pro sekunde

        * Occupancy wird berechnet (in Promille).
        * Register connected_otis_chips hinzugefuegt.
        * Register no_of_channels hinzugefuegt.
  27.04.04:
        * Dummy_readout Daten ergaenzt:
```

```
                Jetzt : OTIS_ID & "0000"
                        16 Bit "event counter"
                        Counterdaten
        * kleinere Bugfixe.
  28.04.04:
        * komplettes Redesign der Occupancyberechnung.
        * Fix eines kleinen Fehlers im Befehlsteil.
Shippo Beta 5 :
  8/13/14/15.04.04:
        * Register Hitdetection_enable entfernt.
        * Register Trigger_enable entfernt.
        * Register pci_not_present entfernt.
        * OTIS_id_x_non_zero von 4 Bit auf 1 Bit zusammengefasst.
        * Register Designversion eingefuehrt. Dieses Register enthaelt
          die Versionsnummer des Designs.
                -> Diese Version ist Nummer 1.
        * Register OV_x [1..0] eingefuehrt. Dieses Register enthaelt
          die Version der verwendeten OTIS-Chips auf Link x.
                Codierung:      0 = OTIS 1.0
                                1 = OTIS 1.1
                                2 = reserved
                                3 = reserved
        * Register LED [7..0] eingefuehrt. In diesem Register steht
          der Wert, der auf den User LEDs ausgegeben wird.
        * Auf Addresse 31 (0x7C) laeuft ein 32 Bit Counter.
        * Das Statusbyte ist nun readonly.
        * Fixed Bar 1 readout.
        * Auf den User LEDs koennen nun Werte in abhaengigkeit von
          LED_sel angezeigt werden.
          Zur Zeit:
                LED_sel     LED[7..0]
                -------     ---------
                000             vdd, vdd, cmd_reg[5..0] des PCI-Buses.
                001             vdd, vdd, stat_reg[5..0] des PCI_Buses
                010             RxData[15..8]
                011             RxData[7..0]
                100             events_in_memory [7..0]
                101             events_in_memory [15..8]
                110             BX Nummer des aktuellen Events
                111             OTIS_id [7..0] des aktuellen Events
        * stop_readout ist automatisch 1 wenn pci_dummy_mode 1 ist.
        * Addresse 1023 (0x"FFC") ist nun ein write_only Register,
          ueber das Befehle an das Board gegeben werden koennen.
          Zur Zeit:
```

```
                Datenwort       Befehl
                1               User Reset : Die Boardeigenen Default-
                                werte werden in die Register geschrie-
                                ben, das Memory wird geloescht.
                2               pci_dummy_mode = 0.
                3               pci_dummy_mode = 1.
                                     ( -> stop_readout = 1)
                4               stop_readout = 0.
                5               stop_readout =1.
                6               reserved
                7               reserved
                8               LED_sel = 000
                9               LED_sel = 001
                A               LED_sel = 010
                B               LED_sel = 011
                C               LED_sel = 100
                D               LED_sel = 101
                E               LED_sel = 110
                F               LED_sel = 111
                Rest            reserved
      * In der Synchronisation wird in Abhaengigkeit des OV_x
        Registers das Komma des OTIS-Chips verwendet.
      * Die Userclock erwartet nun 80 MHz. Eine davon abgeleitete
        40 MHz-Clock wird ausgegeben.
      * diverse Bugfixe.
16.04.04:
      * Bugfixing
19.04.04:
      * Bugfixing
```

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ASDBLR	Amplifier Shaper Discriminator with BaseLine Restoration (Vorverstärker)
BAR	Base Adress Register
BX	Bunch-Crossing
CERN	Convention Européenne de la Recherche Nucléaire
CPU	Central Processing Unit
DAQ	Data Acquisition
DDR-RAM	Double Data Rate Random Access Memory
DESY	Deutsches Elektronen SYnchrotron
DLL	Delay Lock Loop
DNL	Differentielle Nicht-Linearitat
ECAL	Elektromagnetisches Kalorimeter
FE-Box	Frontendbox
Fifo	First In, First Out (Speicher)
FPGA	Field Programable Gate Array
GOL-Aux-Karte	optische Senderkarte
HCAL	Hadronisches Kalorimeter
HLT	Higher Level Trigger (endgultige Entscheidung)
I^2C-Bus	Inter-Integrated Circuit Bus
IT	Inner Tracker
L0	Level 0 (Vorentscheidung)
L1	Level 1 (erste Auswahlentscheidung)
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty Experiment
O-RxCard	Optical Receiver Card
OT	Outer Tracker
OTIS	Outer-Tracker Time Information System
PCI	Peripheral Component Interconnect
PLL	Phase-Locked-Loop
PP-FPGA	PreProzessor-FPGA
QPLL	Quarz crystal based Phase-Locked-Loop
RAM	Random Access Memory
RICH	Ring-Imaging-ČHerenkov-Zahler
ROM	Read Only Memory
SHIPPO	Schnelles Hochenergiephysik Interface und Preprocessing Programm fur den Outer-Tracker

TDC	Time to Digital Converter
TELL1	Trigger ELectronics and L1 board
TESLA	TeV-Energy Superconducting Linear Accelerator
TTC	Timing and Trigger Control
TTCrx	TTC receiver chip
TTCvi	TTC-VME-Bus Interface
TTCvx	VME-sized multiplexer, encoder and fiber-optics transmitter module
VELO	Vertex Locator
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VME	Versa Module Eurocard

Literaturverzeichnis

[1] *„The LHCb Experiment“*, K.A.Gerorge, on behalf of the LHCb Collaboration, Czechoslovak Journal of Physics, Vol.53 (2003),Suppl.A

[2] Jorgen Christiansen,
“Requirements to the L1 front-end electronics”
LHCb Technical Note LHCb 2003-078.

[3] A.N.M. Zwart,
“OTIS Board Version 1.0”,
August 2003, NIKHEF Project no.:39200, Email: a.zwart@nikef.nl

[4] N. Dressnandt et al.
“Implementation of the ASDBLR Straw Tube Readout ASIC in DMILL Technology”
IEEE (2000) Trans. on Nucl. Sci. V48 n4 p1239 R. Bevensee et al.,
“An Amplifier-Shaper-Discriminator with Baseline Restoration for the ATLAS Transition Radiation Tracker”
IEEE (1996) Trans. on Nuc. Sci. V43 p1725
http://www.hep.upenn.edu/atlas/asdblr

[5] Harald Deppe, Uwe Stange , Ulrich Trunk, Ulrich Uwer
Physikalisches Institut, Universität Heidelberg
* *“The OTIS Reference Manual”*,
Version 1.1β , 02.02.2004.

[6] T.Sluijk, U.Uwer
“Specification of the OTIS-to-ASDBLR interface”
NIKHEF, Physikalisches Institut der Uni Heidelberg, 1. September 2003.

[7] Ad Berkien, Tom Sluijk, U.Uwer, D.Wiedner, Albert Zwart.
“Specifications IF13-1 Prototype of the Auxiliary Board for the Outer Tracker”,
Version 2.0, LHCb-2004-073 September 16, 2004
http://cdsweb.cern.ch/search.py?recid=793180&ln=en

[8] P. Moreira, T. Toifl, A. Kluge, G. Cervelli, A. Marchioro, and J. Christiansen
“GOL Reference Manual, Gigabit Optical Link Transmitter manual”,
CERN - EP/MIC, Geneva Switzerland May 2002 Version 1.4.

[9] Paulo Moreira, QPLL Manual,
"Quartz Crystal Based Phase-Locked-Loop for Jitter Filtering Application in LHC"
CERN-EP/MIC, Geneva Switzerland 2004-01-26 Version 1.0
http://proj-qpll.web.cern.ch/proj-qpll/images/qpllManual.pdf

[10] Haefeli, G; Uwer, U; Vollhardt, A; Wiedner, D
"Prototype IF14-1 for an Optical 12 input Receiver Card for the LHCb TELL1 Board"
LHCb 2004-072, electronics, public; Geneva : CERN 6 Sep 2004
http://cdsweb.cern.ch/search.py?recid=792529&ln=en

[11] Dirk Wiedner,
„Aufbau der Ausleseelektronik für das äußere Spurkammersystem des LHCb-Detektors",
Physikalisches Institut der Universität Heidelberg 2004

[12] Guido Haefeli, Aurelio Bay, Federica Legger, Laurent Locatelli, Jorgen Christiansen, Dirk Wiedner.
"Specification for a common read out board for LHCb",
Version 3.0, LHCb 2003-007 IPHE 2003-02 September 2, 2003

[13] Guido Haefeli,
"Contribution to the development of the acquisition electronics for the LHCb experiment",
École Polytechnique Fédérale de Lausanne 2004

[14] *"TTC-VMEbus INTERFACE TTCvi-MkII, Module Identification: EP 680-1128-050-C"*,
RD12 Project, Ph. Farthouat, P.Gälnö CERN EP-ATE, Rev1.6 May 2000

[15] *"TTCvx, Technical description and users manual. A VME-sized multiplexer, encoder and fiber-optics transmitter module for the Timing, Trigger and Control System of the LHC detectors."*
Per Gällnö CERN/EP/ATE/dq, per.gallno@cern.ch, May 21, 1999 Draft

[16] Jorgen Christiansen et al.,
"TTCrx Reference Manual, A Timing, Trigger and Control Receiver ASIC for LHC Detectors"

[17] P.Moreira,
"TTCrq Reference Manual, CERN-EP/MIC",
October 2003, Version 1.0

[18] Beat Jost,
"TFC Broadcast Format",
LHCb Note 2001-017

[19] R.Schwemmer,
"User Interface for the TTC-VMEbus Interface TTCvi"

[20] Maria Manuela Gama da Silva Cunha Spieker:
"Simulation und Analyse der Daten aus einem Teststand zum Nachweis von kosmischen Myonen mit dem LHCb Outer Tracker"
/ vorgelegt von Maria Manuela Gama da Silva Cunha Spieker 2002.
Heidelberg, Univ., Dipl., 2002

[21] Haas, Tanja:
"Bau und Inbetriebnahme eines Teststandes zur Untersuchung von Straw-Kammern mittels kosmischer Myonen"
/ vorgelegt von Tanja Haas. - 2003. - II, 90 S. : graph. Darst.
Heidelberg, Univ., Dipl., 2003

[22] Gerd Modzel
"Aufbau und Test eines optischen Tastkopfes",
Physikalisches Institut Uni Heidelberg 2004

[23] Mirco Nedos,
"Entwicklung und Implementierung eines mit FPGAs realisierten Systems zur Auslese des Äußeren Spurkammersystems des LHCb-Detektors",
TU Dresden 2004

[24] Uwe Stange,
"Entwicklung und Test eines strahlenharten TDCs für das äußre Spurkammersystem von LHCb",
Uni Heidelberg 2004

[25] Altera Corporation.
"Stratix PCI Developement Board Data Sheet",
Version 2.0, September 2003.

[26] Altera Corporation.
„PCI Compiler User Guide,
Version 3.0.0, February 2004.

[27] Philips Semiconducters,
"The I^2C-bus specification",
version 2.1, January 2000

Erklärung:

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den ....

Unterschrift