Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Diplomarbeit

im Studiengang Physik

vorgelegt von

Dirk Hutter

aus Heidelberg

2010

Multi-Event Buffering und HDL-Simulationsframework für die Global Tracking Unit des ALICE-Übergangsstrahlungsdetektors am LHC (CERN)

Dirk Hutter

Die Diplomarbeit wurde ausgeführt am

Kirchhoff-Institut für Physik

unter der Betreuung von

Herrn Prof. Dr. Volker Lindenstruth

Multi-Event Buffering und HDL-Simulationsframework für die GTU des ALICE-Übergangsstrahlungsdetektors

Bei der Untersuchung von Schwerionenkollisionen am ALICE-Experiment sorgt ein mehrstufiges, hierarchisches Triggersystem für die Selektion seltener, interessanter Ereignisse. Der Übergangsstrahlungsdetektor (TRD) liefert einen Beitrag zur Level-1-Entscheidung des Triggersystems sowie Rohdaten zur Offline-Analyse. Die Global Tracking Unit (GTU), ein System aus 109 FPGAs, ist neben ihrer Aufgabe als Triggerprozessor des TRD ein wesentlicher Teil seiner Auslesekette. Sie empfängt Daten vom Detektor mit einer Netto-Bandbreite von 2,16 TBit/s, während die maximale Bandbreite zum zentralen Datenaufnahmesystem zwei Größenordnungen kleiner ausgelegt ist.
Im Rahmen dieser Arbeit wird gezeigt, dass sich die so verursachte, auslesebedingte Totzeit durch Multi-Event Buffering in der GTU minimieren lässt. Die Implementierung einer gegen fehlerhafte Eingangssignale robusten multi-event-fähigen Segmentsteuerung der GTU wird vorgestellt. Ein Hardware-Software-Co-Design ermöglicht es, die erforderliche Echtzeitverarbeitung der Eingangssignale in Hardware und eine ressourcensparende, weitreichende Fehleranalyse in Software zu realisieren. Den zweiten Teil der Arbeit bildet ein generisches HDL-Simulationsframework, mit dem sich Simulationen auf allen Stufen des FPGA-Implementationsprozesses erstellen und automatisiert durchführen lassen. Darauf aufbauend wird ein Modell der GTU und aller Datenquellen zur Verfügung gestellt, welches die vollständige Simulation des Trigger- und Auslesepfads der GTU ermöglicht.

Multi-Event Buffering and HDL-Simulation Framework for the GTU of the ALICE Transition Radiation Detector

At the ALICE experiment, a multi-stage, hierarchical trigger system selects rare, interesting events to facilitate the study of heavy-ion collisions. The Transition Radiation Detector (TRD) delivers a contribution to the Level-1 decision of the trigger system as well as event rawdata for subsequent offline analysis. The Global Tracking Unit (GTU), a system comprised of 109 FPGA-based nodes, is, apart from its function as trigger processor for the TRD, a key element of the detector's readout chain. It receives data from the detector at an aggregate net bandwidth of 2.16 TBit/s, while the available maximum bandwidth to the central data acquisition system is designed to be smaller by two orders of magnitude.
This thesis illustrates how dead-time caused by this lower-bandwidth detector readout can be significantly reduced in an efficient manner by employing a multi-event buffer scheme. The implementation of a multi-event capable segment control unit, robust against erroneous input signals, is presented. Meeting input processing real-time requirements and a software-based, complex handling of errors is rendered possible by a hardware-software co-design. A generic HDL simulation framework constitutes the second part of this thesis. It allows to create and automate simulations on all levels of the FPGA design process and forms the basis for a complete model of the GTU and all data sources. This GTU model, which allows a full simulation of trigger- and readout path, is presented thereafter.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Seit ihren Anfängen in den 1920er Jahren hat die experimentelle Teilchenphysik entscheidende Beiträge zum Verständnis des Aufbaus der Materie und der dabei wirkenden Kräfte geleistet. Unsere heutige Vorstellung der Materie ist im *Standardmodell der Elementarteilchenphysik* zusammengefasst. Es kennt zwei Arten fundamentaler Materieteilchen, *Leptonen* und *Quarks*, und beschreibt die kraftausübenden Wechselwirkungen zwischen ihnen mit Hilfe von Quantenfeldtheorien. Die dabei auftretenden Wechselwirkungen sind die *elektromagnetische*, die *schwache* und die *starke Wechselwirkung*.[1] Die Wechselwirkungen selbst werden von weiteren Teilchen vermittelt, den Austausch-Bosonen. Eine Übersicht über die Teilchen des Standardmodells ist in Tabelle 1.1 gegeben. Die Theorien des Standardmodells sind den letzten Jahrzehnten sehr erfolgreich und in der Lage, viele experimentelle Beobachtungen quantitativ zu beschreiben. Ungeachtet dessen sind jedoch viele Fragen noch nicht abschließend geklärt und wichtige Teile der Theorie nicht experimentell bestätigt. So weiß man beispielsweise nicht, warum es genau drei Familien von Leptonen und Quarks gibt. Auch der Higgs-Mechanismus, der eine mögliche Erklärung für die Masse der Fermionen darstellt, ist bisher nicht bestätigt.

Die starke Wechselwirkung, auch Farbwechselwirkung genannt, wird im Standardmodell durch die Quantenchromodynamik (QCD) beschrieben. Sie beschreibt die Wechselwirkung zwischen farbgeladenen Teilchen und wird von den *Gluonen* vermittelt. Neben Quarks tragen auch die Gluonen selbst Farbladung. Die Besonderheit, dass die Austauschteilchen der Wechselwirkung selbst Ladung tragen, führt zu speziellen Effekten, die nur bei der starken Wechselwirkung auftreten. Entfernt man zwei Quarks voneinander, wird die Wechselwirkung zwischen ihnen nicht kleiner, sondern immer größer. Dies führt dazu, dass Quarks bei niedrigen Energien nie alleine, sondern immer nur in farbneutralen, gebundenen Zuständen von zwei oder drei Quarks auftreten. Diesen Effekt nennt man *Confinement*. Obwohl dieser Effekt bekannt und im Einklang mit experimentellen Beobachtungen ist, ist der genaue Mechanismus, der zu Confinement führt, nicht vollständig verstanden und Gegenstand aktueller Forschung. Der entgegengesetzte Effekt wird *asymptotische Freiheit* genannt. Erhöht man die Energie oder verringert man den Abstand zwischen Quarks, wird die Wechselwirkung kleiner, und die Quarks und Gluonen können sich quasi frei bewegen.

Zur Untersuchung stark wechselwirkender Systeme aus vielen Teilchen können Methoden der Thermodynamik angewendet werden. Aufgrund der asymptotischen Freiheit wird erwartet, dass Materie aus Quarks und Gluonen bei genügend hohen Energien oder Dichten einen neuen Phasenzustand, das Quark-Gluon-Plasma (QGP), einnimmt. Die erwarteten

[1] Die vierte fundamentale Wechselwirkung, die *Gravitation*, ist 10^{-38}-mal schwächer als die starke Wechselwirkung und nicht Teil des Standardmodells.

	Familien			Q/e
Leptonen	e^-	μ^-	τ^-	-1
	ν_e	ν_μ	ν_τ	0
Quarks	u	c	t	$^2/_3$
	d	s	b	$-^1/_3$

Wechselwirkung	Boson	Koppelt an
elektromagnetisch	γ	elektrische Ladung
schwach	$W^\pm$, Z^0	schwache Hyperladung
stark	8 Gluonen	Farbladung

Tabelle 1.1: Die Materieteilchen und Austausch-Bosonen des Standardmodells. Alle Materieteilchen (obere Tabelle) sind Fermionen und haben einen halbzahligen Spin von $\frac{1}{2}$, zu jedem Teilchen existiert zusätzlich ein Antiteilchen. Die Bosonen haben einen ganzzahligen Spin.

Phasenzustände der Kernmaterie sind in Abbildung 1.1 dargestellt. In der Natur tritt QGP bei sehr hohen Dichten in Neutronensternen auf. Für die experimentelle Untersuchung werden bei Schwerionenkollisionen hohe Energiedichten erzeugt, welche die Entstehung von QGP ermöglichen.

Anzeichen für die tatsächliche Existenz von QGP geben frühere Hochenergiephysikexperimente [23]. Für genauere Untersuchungen sollen Kollisionen von hoch-relativistischen Bleikernen am Large Hadron Collider (LHC) ein sehr heißes QGP erzeugen. Die Analyse dieses QGP wird entscheidende Beiträge zum weiteren Verständnis der starken Wechselwirkung und des Confinements liefern. Nach dem Standardmodell der Kosmologie existierte ein solcher Materiezustand außerdem in einem frühen Stadium unseres Universums zwischen der Baryogenese und der Entstehung der Hadronen. Die Untersuchung des Phasenübergangs zwischen QGP und hadronischer Phase kann somit auch Aufschlüsse über die Entstehung der heutigen Materie geben.

Die experimentelle Teilchenphysik ist verbunden mit dem Bau immer stärkerer Teilchenbeschleuniger und komplexerer Detektorsysteme. Während Teilchenbeschleuniger in ihren Anfängen noch in ein Labor passten, handelt es sich bei aktuellen Beschleunigern um mehrere Kilometer große Komplexe. Ebenso haben sich die eingesetzten Detektoren entscheidend verändert. Von manuell ausgewerteten optischen Nachweismethoden und wenigen elektronischen Signalen hin zu tonnenschweren Detektoren, die digitale Daten zu mehreren Millionen Messkanälen liefern. Entwicklung und Bau dieser Geräte sind eng verzahnt mit der Entwicklung und Nutzung neuer Technologien. In der Schwerionenphysik stellen vor allem die abertausenden Sekundärteilchen, die bei einer Kollision erzeugt werden, besondere Anforderungen an die Detektorsysteme und die Datenauslese: Die Auflösung der Detektoren muss groß genug sein, um die einzelnen Spuren unterscheiden zu können. Die enormen Mengen an Daten, die erzeugt werden, sind selbst mit modernen Rechenclustern nicht in Echtzeit zu verarbeiten. Deshalb werden Triggersysteme eingesetzt, um nur die interessantesten Ereignisse auszuwählen und die Gesamtausleserate auf ein verarbeitbares

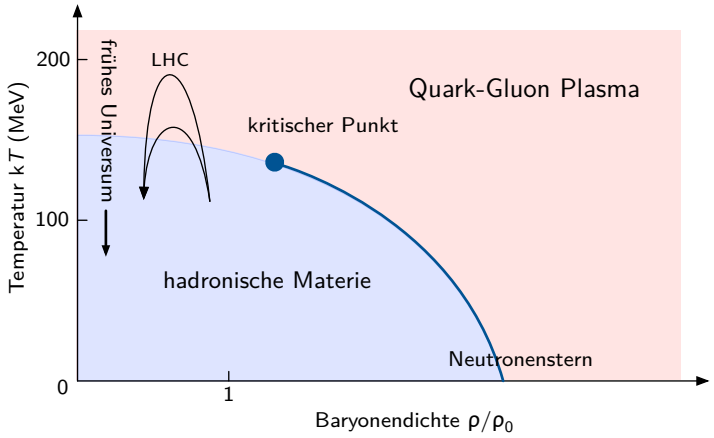


Abbildung 1.1: Schematisches Phasendiagramm der Kernmaterie. Schwerionenkollisionen am LHC sollen ein Quark-Gluon-Plasma bei hohen Temperaturen erzeugen.

Niveau zu senken. Neben der Ausleserate spielt die Totzeit eines Teilchendetektors eine entscheidende Rolle. Die Totzeit ist die Zeit nach einem Ereignis, die ein Detektor nicht bereit ist, ein anderes Ereignis aufzunehmen. Eine hohe Totzeit führt also dazu, dass weniger Ereignisse aufgenommen und wichtige Ereignisse verpasst werden. Eine Minimierung der Totzeit sorgt dafür, dass in der gegebenen Experimentlaufzeit genügend physikalisch interessante Ereignisse aufgenommen werden können.

Der Übergangsstrahlungsdetektor (Transition Radiation Detector, TRD) des ALICE-Experiments[2] am LHC ist ein Beispiel für den Einsatz moderner Technologien bei Detektoren für Hochenergiephysikexperimente. Neben Rohdaten von Ereignissen, aus denen in einer späteren Analyse die Ereignisse rekonstruiert werden können, liefert er Triggerentscheidungen, die bei der Auswahl interessanter Ereignisse helfen. Dazu senden über 65 000 im Detektor verbaute Prozessoren vorberechnete Daten an einen dedizierten Triggerprozessor, die Global Tracking Unit (GTU). Diese rekonstruiert alle hochenergetischen Spuren eines Ereignisses in 2 μs und leitet daraus eine Triggerentscheidung ab.

Die vorliegende Arbeit beschäftigt sich mit der Reduzierung der Totzeit des TRDs durch Multi-Event Buffering (MEB) in der GTU sowie einer entsprechenden Simulation zur Verifikation. In Kapitel 2 wird eine kurze Übersicht über den LHC, das ALICE-Experiment und den TRD gegeben. Kapitel 3 und 4 beschäftigen sich mit dem Aufbau der GTU und der aktuellen Implementierung der Datenauslese. In Kapitel 5 werden Zusammenhänge zwischen Totzeit und Datenauslese diskutiert, und es wird abgeschätzt, wie sich die Totzeit des TRD durch MEB verringern lässt. Kapitel 6 stellt ein Konzept und die konkrete Implementierung einer MEB-Segmentsteuerung der GTU vor. Im zweiten Teil der Arbeit wird eine HDL-Simulationsumgebung vorgestellt, mit der es möglich ist, alle Komponenten der GTU sowie die benötigten Eingangsvektoren zu simulieren. Damit lassen sich neu entwickelte Hardware-Komponenten der GTU überprüfen und bezüglich ihres Verhaltens verifizieren.

[2] ALICE: A Large Ion Collider Experiment

2 Das Experiment

Seit seiner Gründung im Jahr 1953 werden am Europäischen Kernforschungszentrum CERN[1] Beschleunigerexperimente durchgeführt. Heute handelt es sich um das größte Teilchenforschungszentrum der Welt mit über 8000 Wissenschaftlern aus 85 Nationen. Mit dem Large Hadron Collider (LHC) beherbergt das CERN den momentan stärksten Teilchenbeschleuniger der Welt. Die Universität Heidelberg ist mit mehreren Instituten an drei der vier Großexperimente beteiligt. Unter Anderem beteiligen sich das Kirchhoff-Institut für Physik und das Physikalische Institut am Bau des Transition Radiation Detector (TRD) des ALICE-Experimentes. Die folgenden Abschnitte geben eine kurze Übersicht über den LHC, das ALICE-Experiment sowie eine Vorstellung des TRD.

2.1 Der Large Hadron Collider (LHC)

Der LHC ist ein großer Ringbeschleuniger, der voll ausgebaut Proton-Proton-Kollisionen bei einer Schwerpunktsenergie von 14 TeV ermöglicht. Im alternativen Schwerionenmodus sollen Bleikerne mit einer Schwerpunktsenergie von 1148 TeV kollidieren. Sowohl die Energien als auch die erreichbaren Luminositäten überschreiten dabei bisherige Beschleuniger bei Weitem. Der bis dato stärkste Hadronenbeschleuniger, das Tevatron am Fermilab (USA) erreicht eine Schwerpunktsenergie von 2 TeV. Für Schwerionen liefert der Relativistic Heavy Ion Collider (RHIC) Kollisionen von Goldkernen mit einer Schwerpunktsenergie von 15,8 TeV. Die erste Phase der Inbetriebnahme wurde 2010 erfolgreich abgeschlossen. Nach einer kurzen Periode mit Kollisionsenergien von 900 GeV liefert der Beschleuniger nun Proton-Proton-Kollisionen mit einer Schwerpunktsenergie von 7 TeV. Dabei wurde im August 2010 eine integrierte Luminosität von einem inversen Picobarn erreicht. Im November 2010 sollen dann die ersten Schwerionenkollisionen folgen [16].

Der LHC hat einen Umfang von 27 km und befindet sich in einem Tunnel 50 m bis 175 m unter der schweizerisch-französischen Grenze. Ein schematischer Überblick über den Beschleunigerkomplex ist in Abbildung 2.1 gegeben. Die Teilchen werden zunächst in kleineren Anlagen vorbeschleunigt, bevor sie in beide Richtungen in den LHC eingespeist werden. Dort werden die beiden Strahlen dann in mehreren Umläufe bis auf ihre Maximalenergie beschleunigt und an vier Wechselwirkungspunkten zur Kollision gebracht. Hier befinden sich die vier großen Experimente ALICE, ATLAS[2], CMS[3] und LHCb[4].

[1] Das Akronym CERN leitet sich ab von *Conseil Européen pour la Recherche Nucléaire.*
[2] ATLAS: A Toroidal LHC ApparatuS
[3] CMS: Compact Muon Solenoid
[4] LHCb: Large Hadron Collider beauty

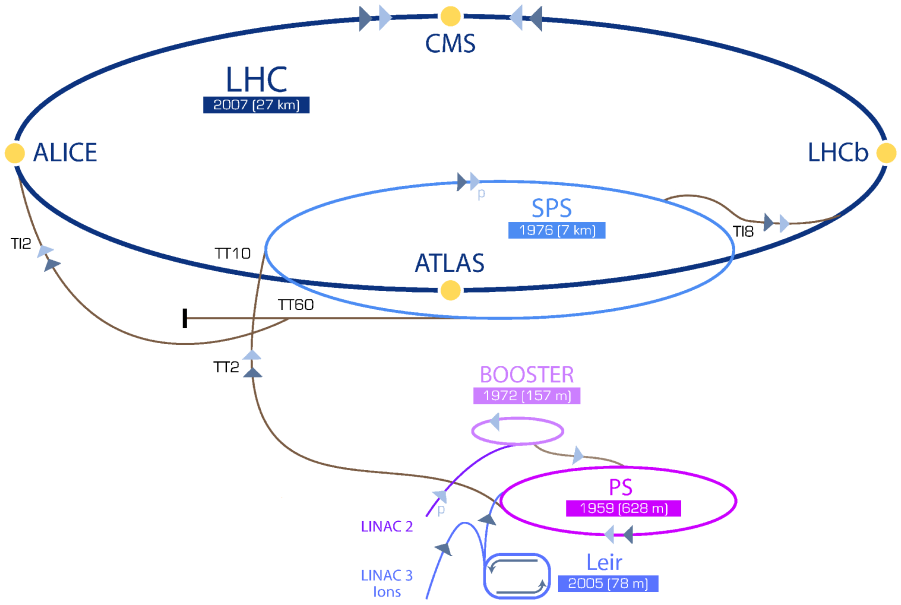


Abbildung 2.1: Überblick über den LHC und die verwendeten Vorbeschleuniger. Die dunkelgrauen Pfeile zeigen den Weg der Schwerionen an, deren Kollisionen von ALICE untersucht werden sollen. Quelle: CERN

Um die Teilchen auf ihrer Kreisbahn zu halten, werden Magnetfelder von über 8,5 Tesla benötigt. Der Ring besteht dafür aus 1232 supraleitenden Dipolmagneten, die mit superfluidem Helium auf 1,8 K gekühlt werden. Für die notwendige Fokussierung des Teilchenstrahls kommen zusätzlich 392 supraleitende Quadrupolmagnete zum Einsatz. Der Gesamtenergieverbauch liegt dabei bei rund 120 MW [15].

Am LHC finden insgesamt sieben Experimente statt. ATLAS und CMS sind Mehrzweck-Experimente, die sich unter anderem mit dem Ursprung der Masse beschäftigen. Sie suchen dafür nach dem bislang unentdeckten Higgs-Boson. Durch die Untersuchung der gleichen Fragestellungen mit unterschiedlichen physikalischen Ansätzen ergänzen sich die Experimente und ermöglichen die gegenseitige Überprüfung von Ergebnissen. LHCb hingegen ist spezialisiert auf die Untersuchung der CP-Verletzung bei b-Hadronen[5]. Hierfür ist der Detektor nicht schalenförmig um den Kollisionspunkt angeordnet, sondern detektiert Teilchen nur in Vorwärtsrichtung. ALICE ist das einzige Experiment, das speziell auf Schwerionenkollisionen optimiert ist. Es soll den Zustand der Materie unmittelbar nach dem Urknall nachstellen und dient der Erforschung des Quark-Gluon-Plasma (QGP). Weiterhin gibt es drei kleinere Experimente, die sich die Wechselwirkungspunkte mit den größeren Experimenten teilen. TOTEM[6] vermisst dabei den totalen Wirkungsquerschnitt der Proton-Proton-Kollisionen, während LHCf[7] Daten für Modelle hoch-intensiver kosmischer Höhenstrahlung sammelt. Während LHCf seine Laufzeit schon fast erreicht hat,

[5]Bei b-Hadronen handelt es sich um gebundene Zustände eines b-Quarks mit einem leichteren Quark.
[6]TOTEM: Total Elastic and Diffractive Cross Section Measurement
[7]LHCf: Large Hadron Collider forward

befindet sich das siebte Experiment, MoEDAL[8], gerade im Aufbau. Es ist für die Suche nach hypothetischen Teilchen wie magnetischen Monopolen konzipiert.

2.2 Das ALICE-Experiment

Die Untersuchung von QGP ist ein Schwerpunkt aktueller Hochenergiephysik. Experimente am RHIC haben erste Erkenntnisse über die Eigenschaften dieses Phasenzustands geliefert [27]. Das ALICE-Experiment soll mit den am LHC erstmals erreichbaren Energien einen entscheidenden Beitrag zum Verständnis und zur Erforschung von QGP leisten. Dabei sollen Kollisionen von Bleikernen bei einer Schwerpunktsenergie von 5,52 TeV pro Nukleonpaar ein QGP erzeugen, dessen Zerfallsprodukte von den Detektoren des Experiments nachgewiesen werden. Die große Zahl entstehender Teilchen stellt dabei besondere Herausforderungen bezüglich Detektierbarkeit und Datenvolumen an das Experiment. Mit einer Länge von 25 m, einer Höhe von 12 m und einem Gewicht von 10 000 t ist das ALICE-Experiment eines der größten Teilchenphysikexperimente der Welt [1].

2.2.1 Die Detektoren

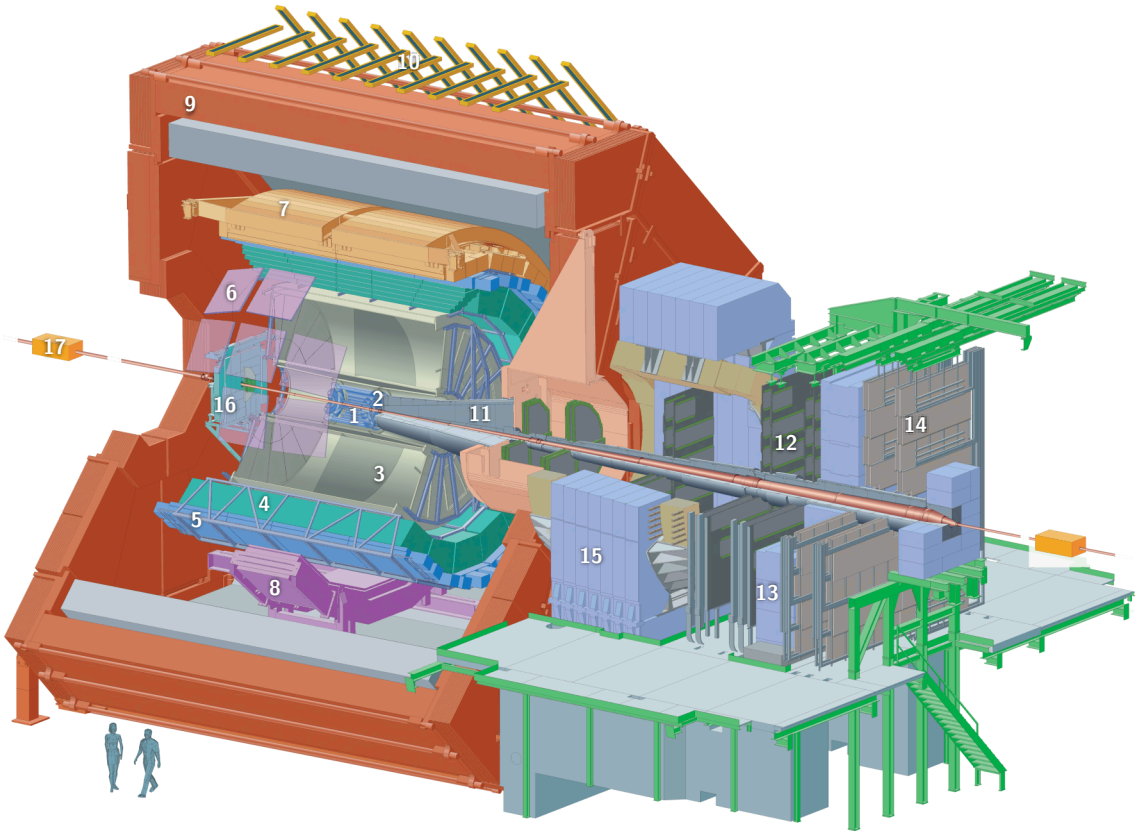
Um ein möglichst vollständiges Bild eines Kollisionsereignisses zu erhalten, ist das ALICE-Experiment aus einer Vielzahl von einzelnen, sich ergänzenden, Detektoren aufgebaut. Ihre Aufgaben lassen sich grob in die Kategorien *Spurrekonstruktion geladener Teilchen*, *Teilchenidentifikation* und *Detektion von Photonen* unterteilen. Zur Erfüllung dieser Aufgaben ist ALICE mit zwei Magneten ausgestattet, die die Teilchen ablenken und somit Rückschlüsse auf Ladung und Impuls zulassen. Abbildung 2.2 zeigt eine Übersicht des Gesamtaufbaus. Dieser lässt sich in einen Zentralbereich links im Bild und einen Vorwärtsbereich auf der rechten Seite unterteilen. Die Detektoren und Absorber des Vorwärtsbereichs bilden zusammen mit dem Dipolmagneten das Myonspektrometer. Es wird zur Untersuchung des $\mu^-\mu^+$-Zerfallskanals von Quarkonia[9] eingesetzt [3]. Der Zentralbereich ist komplett von dem rot dargestellten L3-Solenoid umgeben, der ein homogenes Magnetfeld von 0,45 T parallel zur Strahlachse erzeugt. Die Detektoren des Zentralbereichs sind im Folgenden kurz beschrieben.

Die Detektoren FMD, T0 und V0 bestimmen, zusammen mit den 110 m entfernt vom Kollisionspunkt angebrachten Zero Degree Calorimeters (ZDCs), dass ein Ereignis stattgefunden hat und bilden somit L0-Triggerdetektoren [12, 7].

Die inneren Detektoren des Zentralbereichs umgeben die Strahlachse zylindrisch und dienen der Spurrekonstruktion. Direkt am Strahlrohr befindet sich das Inner Tracking System (ITS). Es besteht aus sechs Lagen unterschiedlicher Siliziumdetektoren und kann Spuren mit einer Auflösung von bis zu 12 μm detektieren. Dadurch können der primäre

[8] MoEDAL: Monopole and Exotics Detector at the LHC

[9] Quarkonium ist ein gebundener Zustand aus einem schweren Quark und dem zugehörigen Antiquark.



1. ITS (Inner Tracking System)
2. FMD (Forward Multiplicity Detector), T0, V0
3. TPC (Time Projection Chamber)
4. TRD (Transition Radiation Detector)
5. TOF (Time of Flight Detektor)
6. HMPID (High Momentum Particle Identification)
7. EMCal (Electromagnetic Calorimeter)
8. PHOS CPV (Photon Spectrometer Charged Particle Veto Detector)
9. L3 Magnet
10. ACORDE (ALICE Cosmic Ray Detector)
11. Absorber
12. Muon Tracking Chambers
13. Muon Filter Wall
14. Muon Trigger Chambers
15. Dipole Magnet
16. PMD (Photon Multiplicity Detector)
17. ZDC (Zero Degree Calorimeter)

Abbildung 2.2: Die Detektoren des ALICE-Experiments. Der Transition Radiation Detector (TRD) ist der mit Nummer 4 bezeichnete Hohlzylinder. Quelle: CERN

Vertex sowie sekundäre Vertices, die von sofort zerfallenden schweren Hadronen stammen, präzise bestimmt werden [4].

Weiter außen folgt die Time Projection Chamber (TPC). Sie bildet ein großes Gasvolumen mit einer radialen Ausdehnung von etwa 0,8 m bis 2,5 m und einer Länge von 5 m. Von ionisierenden Teilchen im Gas erzeugte Elektronen driften in einem homogenen elektrischen Feld in Richtung der beiden Endkappen. Dort werden sie von Vieldraht-Proportional-Zählkammern zeitaufgelöst detektiert. Aus dem Auftreffort und der Driftzeit lässt sich die Teilchentrajektorie rekonstruieren [9].

Der nächste Detektor in radialer Richtung ist der TRD. Er besteht aus sechs Lagen kleiner Driftkammern, die, in Kombination mit einem Radiatormaterial, sowohl Spurrekonstruktion als auch Elektronenidentifikation durch Übergangsstrahlung ermöglichen. Eine Hauptaufgabe des TRD besteht darin, als Triggerdetektor einen Beitrag zum L1-Trigger zu liefern. Eine ausführlichere Beschreibung des TRD folgt in Abschnitt 2.3 [10].

Weiter außen folgen die Detektoren zur Teilchenidentifikation. Dabei misst der Time-of-Flight-Detektor (TOF) die Flugzeit der Teilchen mit einer Zeitauflösung von 100 ps, um daraus ihre Masse zu bestimmen. Für Teilchen mit großen Impulsen misst das High-Momentum-Particle-Identification-System (HMPID) mit Hilfe von Ring-Imaging-Cherenkov-Detektoren die Geschwindigkeiten der Teilchen. Dafür bestimmt es den Öffnungswinkel der beim Durchqueren eines Radiators abgegebenen Cherenkov-Strahlung [8, 2].

Der Zentralbereich wird vervollständigt durch Detektoren zur Photonendetektion: Photon Spectrometer (PHOS), Photon Multiplicity Detector (PMD) und Electromagnetic Calorimeter (EMCal) geben mit ihren kalorimetrischen Messungen Auskunft über die Temperatur der Kollision. EMCal wird zusätzlich dazu eingesetzt, Gruppen von entstehenden Teichen, *Jets*, zu messen, die Information über die Frühphase des Kollisionsereignisses liefern [6, 5, 13].

2.2.2 Das ALICE-Trigger-System

Für Kollisionsexperimente ergibt sich die zu erwartende Ereignisrate aus der Luminosität L des Beschleunigers und dem Wirkungsquerschnitt σ nach

$$\dot{N} = \sigma \cdot L \ . \tag{2.1}$$

Die angestrebten Luminositäten bei ALICE sind $L_{\text{Pb-Pb}} = 10^{27}\,\text{cm}^{-2}\,\text{s}^{-1}$ für Pb-Pb- und $L_{\text{p-p}} = 10^{30}\,\text{cm}^{-2}\,\text{s}^{-1}$ [10] für p-p-Kollisionen. Mit einem totalen hadronischen Wirkungsquerschnitt von $\sigma_{\text{Pb-Pb}} = 8\,\text{b}$ und $\sigma_{\text{p-p}} = 0{,}1\,\text{b}$ liegen die Ereignisraten für *minimum-bias*-Ereignisse bei 8 kHz für Pb-Pb und 100 kHz für p-p [1]. Aufgrund der limitierten Ausleserate der Detektoren und der zu erwartenden Datenmengen, können bei Weitem nicht alle diese Ereignisse aufgenommen werden. Die meisten von ihnen sind aber hinsichtlich aktueller physikalischer Fragestellungen uninteressant.

Um eine passende Ereignisauswahl zu treffen, ist ALICE mit einem 4-stufigen *Trigger-system* ausgestattet. Die ersten drei Stufen werden dabei vom Central Trigger Processor (CTP) evaluiert. Er nimmt Triggerbeiträge, die von bestimmten Detektoren bereitgestellt werden, entgegen und entscheidet daraufhin, ein Ereignis abzulehnen (*reject*) oder zu akzeptieren (*accept*). Die Triggerbeiträge der Detektoren signalisieren interessante Ereignisse und werden von detektoreigenen Triggerprozessoren auf der Grundlage der Detektordaten berechnet. Dabei reicht die Komplexität der Berechnung je nach Detektor von einfachen Koinzidenz-Triggern bis hin zu aufwendigen Spurrekonstruktions-Algorithmen.

Die vierte Stufe wird durch den High-Level Trigger (HLT) realisiert. Hierbei handelt es sich um einen Rechnerverbund, der mit Hilfe einer Echtzeitanalyse der Rohdaten entscheidet, ob ein Ereignis permanent gespeichert wird oder nicht. Abbildung 2.3 gibt einen Überblick über die Triggerstufen und die zu erwartenden Raten. Die Verzögerungen der Triggerstufen sind an die Bedürfnisse der einzelnen Detektoren angepasst. Während manche Detektoren

[10] Um *pile-up* zu vermeiden, läuft ALICE im p-p-Modus mit reduzierter Luminosität, verglichen mit der Spitzenluminosität des LHC von $L_{\text{p-p}} = 10^{34}\,\text{cm}^{-2}\,\text{s}^{-1}$.

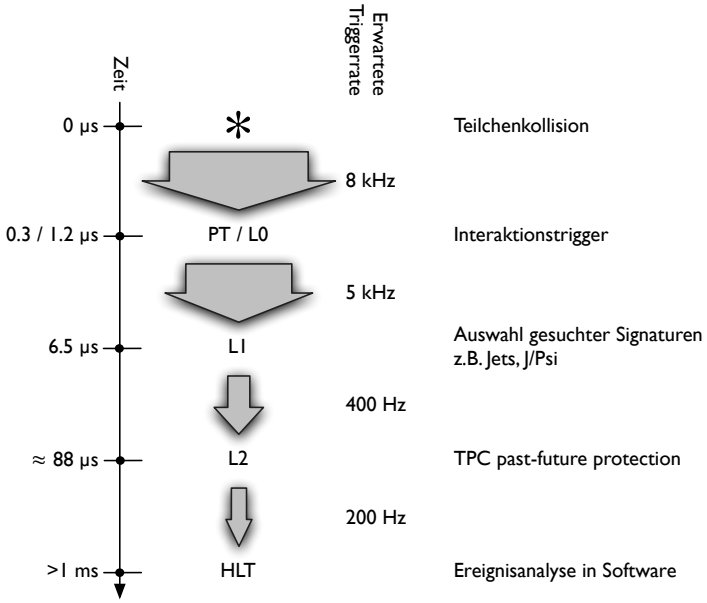


Abbildung 2.3: Die Triggerstufen des ALICE-Experiments. Angegeben sind die erwarteten Triggerraten für Pb-Pb-Kollisionen. Die Hauptauswahl interessanter Ereignisse erfolgt mit dem L1-Trigger.

ein sehr frühes Signal für ihre Elektronik benötigen, können andere Detektoren erst spät ausgelesen werden. Die Zeit wird dann genutzt, um komplexere Triggerentscheidungen zu treffen [11].

Triggersequenzen Sind alle an der Datennahme beteiligten Detektoren bereit, kann der CTP eine Triggersequenz starten. Eine komplette Triggersequenz, die zur Auslese und Speicherung eines Ereignisses führt, ist in Abbildung 2.3 dargestellt und umfasst *L0 – L1 – L2a – HLT-accept*. Jede Triggersequenz beginnt mit einem L0-Trigger, der 1,2 μs nach der Kollision ausgelöst wird. Zweck des L0-Triggers ist es, zu bestimmen, ob eine zentrale Kollision stattgefunden hat. Die Entscheidung hierfür basiert auf Informationen schneller Triggerdetektoren wie FMD, V0, T0 und SDD. Wenn nötig, kann außerdem die Gesamtrate mit Hilfe eines Skalierungsfaktors verringert werden. 6,5 μs nach der Kollision wird ein L1-Trigger ausgelöst. Zu diesem Zeitpunkt liegen genauere Informationen über das Ereignis aus komplexeren Triggerentscheidungen der Detektoren vor. Hierzu gehört beispielsweise Information über das Vorhandensein von Teilchen-Jets oder besonders hochenergetischer Elektronen. Einen wesentlichen Beitrag zum L1-Trigger liefert unter anderem der TRD. Am Ende der TPC-Driftzeit, ungefähr 88 μs nach der Kollision, wird ein L2a- oder L2r-Trigger ausgelöst. Er dient hauptsächlich als *Past-Future Protection* der TPC, welcher Ereignisse, die sich zeitlich mit anderen Ereignissen überschneiden (*pile-up*), verwirft. Ein L2a-Trigger veranlasst alle Detektoren, ihre Rohdaten an das zentrale Datenaufnahmesystem (Data Acquisition System, DAQ) zu schicken. Zusätzlich verfügt der TRD über einen schnellen Wechselwirkungstrigger, den *Pre-Trigger*, der vom Detektor

selbst erzeugt wird. Sein Zweck ist grundsätzlich derselbe wie der des L0-Triggers, jedoch mit einer weitaus geringeren Verzögerung von nur etwa 0,3 μs.

Um die Totzeit des gesamten Experiments zu verringern und gleichzeitig möglichst viele Daten nehmen zu können, dürfen sich die Triggersequenzen überschneiden. Dabei kann bereits nach einem L1-Trigger die nächste Triggersequenz beginnen, sobald alle Detektoren bereit sind. Um schnelle Detektorgruppen wie etwa das Myonspektrometer auch auslesen zu können, während andere Detektoren noch nicht bereit für ein neues Ereignis sind, können die Detektoren vom CTP in sogenannte *Cluster* zusammengefasst werden, die dann unabhängig voneinander Triggersequenzen erhalten.

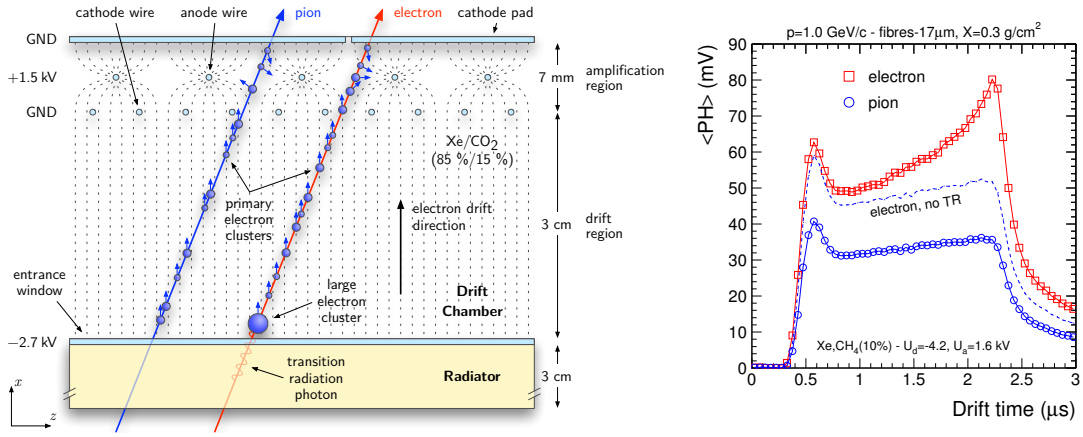
Triggerverteilung Um die Detektoren zu synchronisieren und die Triggerentscheidungen des CTP zu verteilen, wird das RD-12-Timing-Trigger-and-Control-System (TTC) eingesetzt. Die Übermittlung der Daten erfolgt dabei über ein optisches Übertragungssystem, das zwei Datenkanäle zeit-gemultiplext (A- und B-Kanal) an einen speziellen ASIC, den TTCrx-Chip, schickt. Dieser decodiert die Signale und stellt sie gemeinsam mit dem aus dem Datenstrom wiederhergestellten 40.08-MHz-LHC-Takt der Detektorelektronik, Front-End Electronics (FEE) genannt, zur Verfügung.

Auf dem A-Kanal sind dabei die zeitkritischen L0- und L1-Trigger als Pulse verschiedener Breite kodiert. Kanal B überträgt zusätzliche Daten mit verschiedenen Prioritäten. Dazu gehören Triggernachrichten und Synchronisationssignale, die zur Ereignisidentifikation eingesetzt werden.

2.3 Der Transition Radiation Detector (TRD)

Bei der Untersuchung des QGP spielen elektromagnetische Sonden eine entscheidende Rolle, da diese nicht von hadronischen Effekten beeinflusst werden. Sie bieten somit einen direkten Blick ins Innere des Plasmas. Weiterhin ist die Untersuchung des leptonischen Zerfallskanals schwerer Vektormesonen (J/ψ, Υ) von Bedeutung, da ihre Häufigkeit Rückschlüsse auf Plasmazustände zulässt.

Die gewünschten Untersuchungen setzen voraus, dass Elektronen zufriedenstellend von Pionen unterschieden werden können, da diese mit wesentlich höheren Raten produziert werden. Für Teilchen mit Impulsen unter 1 GeV/c ist die Identifikation über ihren relativen Energieverlust dE/dx und der damit verbundenen Bahnänderung in der TPC möglich. Für Teilchen mit höheren Impulsen funktioniert diese Methode aufgrund der zu geringen Änderungen nicht mehr. Für die Untersuchung der Vektormesonen ist es aufgrund ihrer geringen Produktionsrate (z. B. weniger als ein Υ pro Minute) außerdem unverzichtbar, speziell auf diese Ereignisse zu triggern. Beide Aufgaben werden vom TRD erfüllt, der im Folgenden genauer vorgestellt wird.



(a) TRD-Modul **(b)** Pulsform

Abbildung 2.4: Funktionsprinzip eines TRD-Moduls. Jedes Modul besteht aus einem Radiator und einer Driftkammer. Die von Elektronen im Radiator hervorgerufene Übergangsstrahlung erzeugt ein großes Elektronencluster am Eintrittspunkt der Driftkammer. Die gemittelte Pulsform für Elektronen und Pionen ist rechts zu sehen. Die durch die Übergangsstrahlung freigesetzten Sekundärelektronen führen zu einer Überhöhung am Ende des Pulses. Quellen: [18, 10]

Funktionsweise Beim TRD handelt es sich um einen Driftkammerdetektor, der um einen Radiator zur Erzeugung von Übergangsstrahlung erweitert ist. Während die Driftkammer zur Spurrekonstruktion verwendet wird, kann aus der Übergangsstrahlung auf die Teilchenidentität geschlossen werden.

Übergangsstrahlung entsteht beim Durchgang eines geladenen Teilchens durch die Grenzfläche zweier Medien mit verschiedenen Dielektrizitätskonstanten. Die Intensität ist proportional zum Lorentzfaktor $\gamma = \frac{E}{mc^2}$ und stark in Vorwärtsrichtung fokussiert ($\theta \propto \gamma^{-1}$). Aufgrund der Massendifferenz vom $m_\pi \approx 273 m_\mathrm{e}$ ist die entstehende Übergangsstrahlung für Pionen zu vernachlässigen.

Das Funktionsprinzip der Driftkammer ist in Abbildung 2.4a dargestellt. Geladene Teilchen, die die Kammer durchqueren, ionisieren Gasmoleküle. Die dadurch frei gewordenen Elektronen driften in einem homogenen elektrischen Feld in Richtung der Kathodendrähte. Ab diesem Punkt wird das Feld stark inhomogen, was zu Sekundärionisationen und einem Lawineneffekt führt. Die Spiegelladung des so verstärkten Signals wird an den Kathoden-Pads gemessen. Wenn Übergangsstrahlung auftritt, werden zusätzliche Gasmoleküle am Eintrittspunkt der Kammer ionisiert. Der entstehende Elektronencluster führt zu einer Überhöhung am Ende des gemessenen Pulses. Die zu erwartenden Pulsformen für Elektronen und Pionen sind in Abbildung 2.4b gezeigt. Die Überhöhung am Anfang des Pulses ist ein Artefakt der Verstärkung. Der absolute Unterschied zwischen Elektronen und Pionen rührt von den unterschiedlichen Ionisationsverlusten her, wie sie durch die Bethe-Bloch-Formel beschrieben werden.

Aufbau Der gesamte TRD besteht aus 540 Driftkammermodulen. Sie sind in radialen Stapeln (*Stacks*) von je sechs Modulen angeordnet. Je fünf entlang der Strahlrichtung aneinandergereihte Stapel bilden ein Supermodul. Die 18 Supermodule sind als Hohlzylinder mit einem Innendurchmesser von 2,9 m und einem Außendurchmesser von 3,7 m um den Kollisionspunkt angeordnet. Dies ergibt eine volle azimutale Abdeckung und einen Pseudorapiditäts-Bereich[11] von $|\eta| \leq 0{,}9$.

Für die Auslese und Datenverarbeitung von 1,2 Millionen Analogkanälen sind im ganzen Detektor 65 564 Multi Chip Modules (MCMs) verbaut. Sie bestehen jeweils aus zwei spezialisierten ASICs: Dem Preamplifier/Shaper (PASA), der die Signale verstärkt, und dem Tracklet Processor (TRAP), der sie digitalisiert, filtert und mit Hilfe von vier RISC-CPUs verarbeitet. Zusammen mit weiteren Komponenten zur Steuerung und Datenübertragung bilden sie die Front-End-Elektronik (FEE) des Detektors.

Jedes der Driftkammermodule ist über je zwei optische Fasern mit der Global Tracking Unit (GTU) verbunden. Beide Fasern dienen der Datenübertragung in GTU-Richtung. Ein Rückkanal ist nicht vorhanden. Die GTU besteht aus 109 FPGA-basierten Rechenknoten. Neben ihrer Hauptaufgabe als Triggerprozessor ist sie Teil der Auslesekette und leitet Rohdaten an das zentrale ALICE-Datenaufnahmesystem DAQ weiter. Eine genauere Erläuterung der GTU und des GTU-Datenpfades ist in den Kapiteln 3 und 4 zu finden.

Triggerkonzept Um einen Trigger auf hoch-energetische Elektronen zu ermöglichen, ist es nicht nur nötig, sie von Pionen zu unterscheiden, sondern auch ihren transversalen Impuls p_t zu bestimmen. Dies ist aufgrund der gebogenen Bahn, auf der sich geladene Teilchen in einem Magnetfeld bewegen, möglich. Vernachlässigt man den Energieverlust, handelt es sich um eine Kreisbahn, die über $r = \frac{p_\mathrm{t}}{e \cdot B}$ mit dem Impuls verbunden ist. Für Transversalimpulse von $p_\mathrm{t} \approx 3\,\mathrm{GeV/c}$ beträgt der Radius etwa 25 m.

Um den Impuls in Echtzeit zu bestimmen, verfügt der TRD über ein mehrstufiges Rekonstruktionssystem. Einige mit AliRoot simulierte Teilchenspuren mit Transversalimpulsen von 3 GeV/c sind in Abbildung 2.5 dargestellt. Im ersten Schritt berechnen die TRAPs aus den digitalisierten Ladungsclustern bis zu 20 000 kurze Spursegmente, die *Tracklets*. Diese liegen bereits wenige Mikrosekunden nach der Kollision vor und werden über die insgesamt 1080 optischen Fasern an die GTU übertragen. Die Tracklets aus den sechs übereinander liegenden Ebenen werden dort im zweiten Schritt, dem *Track Matching*, einer Teilchenspur zugeordnet. An die Tracklets einer so gefundenen Teilchenspur wird dann ein *GTU-Track* angepasst. Zusammen mit der Annahme, dass die Teilchen aus dem Kollisionspunkt stammen, kann daraus der Impuls rekonstruiert werden. Basierend auf den rekonstruierten Spuren können in der GTU unterschiedliche Algorithmen angewendet werden, um eine Triggerentscheidung abzuleiten. Diese wird nach nur 6,2 μs an den CTP gesendet und dient dort als L1-Beitrag [10].

[11] Die Pseudorapidität ist definiert als $\eta = -\ln[\tan(\frac{\theta}{2})]$ und ein Maß für den Winkel eines Vektors relativ zur Strahlachse.

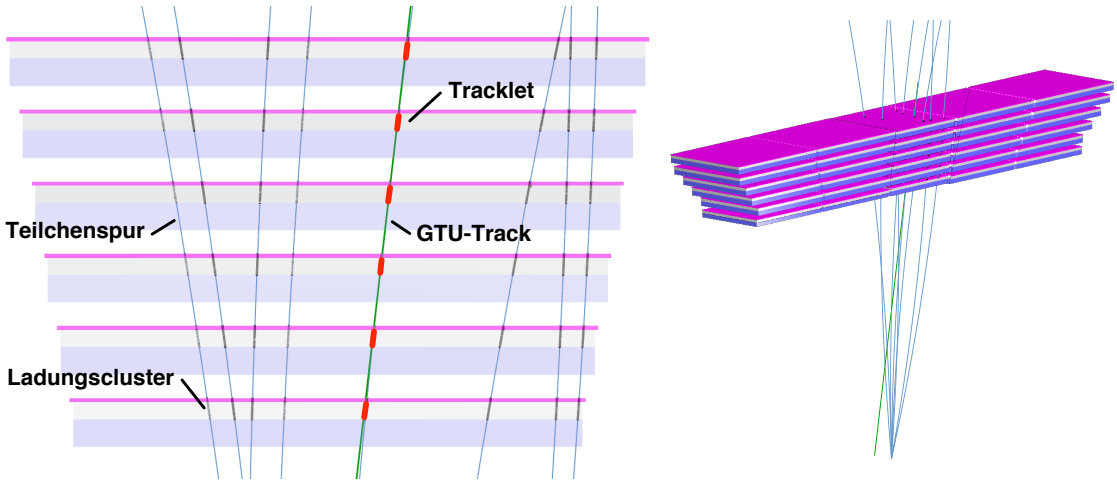


Abbildung 2.5: Online-Rekonstruktion einer Teilchenspur in der GTU des TRD. Dargestellt sind simulierte Teilchen mit $p_t = 3\,\mathrm{GeV/c}$, die den mittleren Stack eines Supermoduls durchlaufen. Für eine Spur sind beispielhaft Tracklets und ein GTU-Track abgebildet.

Datenfluss und Datenauslese Neben der Funktion als Triggerdetektor trägt der TRD mit seinen Rohdaten auch zur späteren Offline-Analyse bei. Dazu müssen die, durch die ADCs des TRAPs digitalisierten, Ladungssignale gepuffert werden, bis ein L2a-Trigger die Rohdatenauslese des Detektors veranlasst.

Eine Übersicht der gesamten Auslesekette und der Pufferstruktur ist in Abbildung 2.6 gegeben. Die Auslesekette besteht aus vier Stufen. Um Übertragungszeit zu minimieren, orientieren sich diese an den Triggerstufen. Erst, wenn ein Ereignis die entsprechende Triggerstufe passiert hat, wird es zur nächsten Pufferstufe weitergeleitet. Ansonsten wird es in der entsprechenden Stufe verworfen. Die Übertragungsbandbreiten zwischen den einzelnen Stufen sind stark unterschiedlich und nehmen zu höheren Triggerstufen hin ab.

Signalisiert ein Pre-Trigger ein Kollisionsereignis, beginnen die ADCs mit der Digitalisierung der Ladungssignale und speichern diese in den Ereignispuffern der TRAP-Chips. Folgt kein L0-Trigger, wird der Vorgang abgebrochen und die Puffer werden geleert. Passiert das Ereignis die L0- und L1-Stufe, werden die Puffer über den Auslesebaum der Front-End-Elektronik (FEE) in fester Reihenfolge an die GTU gesendet. Die Daten werden dabei über die 1080 optischen FEE-Links mit einer Gesamtbandbreite von 270 GB/s ohne Flusskontrolle übertragen und müssen von der GTU in voller Geschwindigkeit aufgenommen werden. Die GTU puffert die Daten im Zeitraum zwischen dem L1- und L2-Trigger. Wird ein L2a-Trigger ausgelöst, fasst die GTU die Daten eines Supermoduls zusammen und leitet das angeforderte Ereignis an die DAQ weiter. Dabei stehen 18 bidirektionale Detector Data Links (DDLs) zur Verfügung, über die mit insgesamt maximal 3,6 GB/s Daten an die erste Stufe der DAQ, die Local Data Concentrators (LDCs), übermittelt werden können. Da diese Übertragungsstrecke über eine Flusskontrolle verfügt, kann die Übertragung vom LDC unterbrochen werden. Die in den LDCs gesammelten Sub-Ereig-

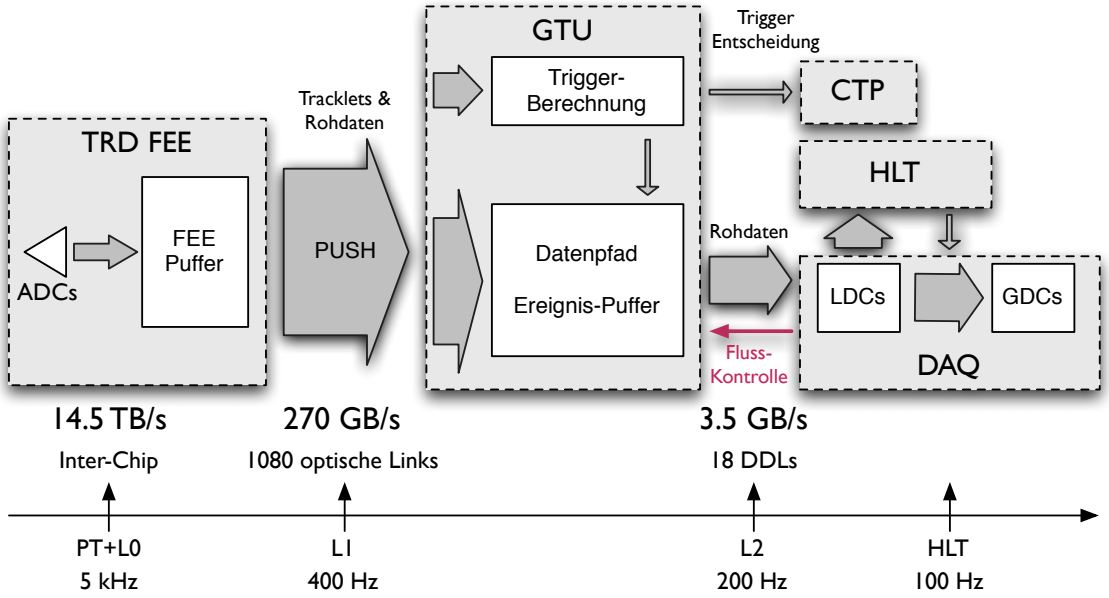


Abbildung 2.6: Auslesekette des TRD von der Front-End-Elektronik bis zur DAQ

nisse werden nach Bestätigung durch den HLT an einen Global Data Concentrator (GDC) weitergeleitet, der sie zum finalen Gesamtereignis zusammenfügt und an das Mass Storage System zur dauerhaften Speicherung für die spätere Offline-Analyse schickt.

3 Die Global Tracking Unit (GTU)

Die GTU bildet als Triggerprozessor und Teil der Auslesekette einen essentiellen Bestandteil des TRD. Ihre Architektur und die zugrunde liegende Hardware sind im nachfolgenden Kapitel beschrieben. Da die Funktion als Triggerprozessor nicht im Fokus dieser Arbeit steht, wird hier auf eine detaillierte Darstellung verzichtet. Das Wort *Trigger* bezieht sich daher in den folgenden Kapiteln, sofern nicht explizit erwähnt, auf die vom CTP an der GTU eintreffenden Trigger-Informationen und nicht auf die von der GTU getroffenen Triggerentscheidungen. Für eine ausführliche Erläuterung der GTU als Triggerprozessor sei auf die Arbeiten von Jan de Cuveland verwiesen [18, 17].

3.1 Architektur

Die hohen Anforderungen an die Latenz der Triggerentscheidung und den Datendurchsatz bestimmen maßgeblich die Architektur der GTU. Die insgesamt 109 FPGA-basierten Rechenknoten arbeiten in einem dreistufigen Aufbau zusammen. Dazu existieren drei unterschiedlich ausgebaute Varianten der gleichen CompactPCI-Karte, denen unterschiedliche Aufgaben zufallen. Die funktionalen Unterschiede ergeben sich sowohl durch die Bestückung und Verwendung von Aufsteckkarten als auch durch unterschiedliche FPGA-Designs.

Die hierarchische Struktur der GTU ist in Abbildung 3.1 dargestellt. Der Segmentierung des TRD entsprechend, lässt sich die GTU in 18 unabhängige Segmente aufteilen, eines für jedes Supermodul des Detektors. Jedes Segment besteht dabei aus fünf *Track Matching Units (TMUs)* und einer *Supermodule Unit (SMU)*. Die Segmente arbeiten weitgehend unabhängig voneinander und verfügen über je einen DDL zur Anbindung an die DAQ und eine Anbindung an das TTC-System. Die TMUs erfüllen zwei Hauptaufgaben. Sie empfangen von der Front-End Elektronik (FEE) die, nach einem L0- und L1-Trigger gesendeten, Tracklets und Rohdaten und speichern sie bis zum L2-Trigger zwischen. Weiterhin führen sie die, für die Triggerentscheidung benötigten, Tracking-Berechnungen aus und leiten die Ergebnisse mit geringer Latenz an die SMU weiter. Die SMU ist für Steuerung, Kontrolle und Datenauslese eines GTU-Segments zuständig und bildet die zweite Stufe bei der Triggerentscheidung. Sie stellt auch die Anbindung an das TTC-System und die DAQ dar. Abhängig von der Triggersequenz steuert sie das Speichern und Verwerfen von Ereignisdaten in den TMUs. Bei einem L2-accept liest sie die TMUs eines Segments aus, formatiert die Daten und sendet sie an die DAQ. Die von den TMUs berechneten lokalen Triggerdaten werden von der SMU verwendet, um eine Segment-Entscheidung zu treffen. Die dritte Ebene stellt die *Trigger Generation Unit (TGU)* dar. Sie fungiert als Konzentrator und

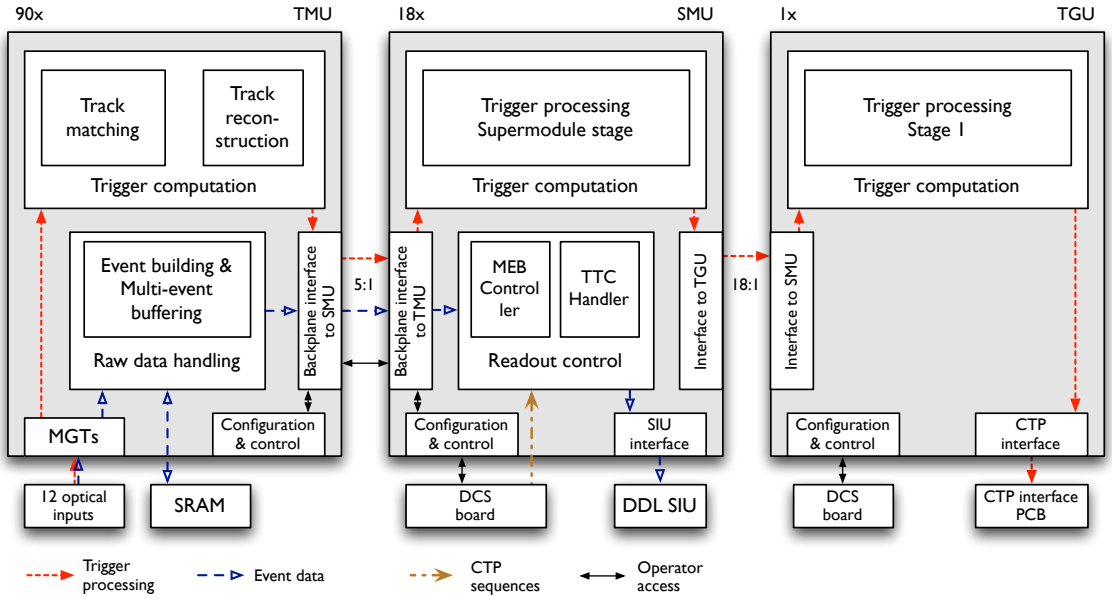


Abbildung 3.1: Dreistufiger Aufbau der GTU-Architektur. Je ein Segment aus fünf TMUs und einer SMU verarbeitet die Daten eines der 18 Supermodule des Detektors. Die Triggerdaten der 18 Segmente werden in der TGU zusammengefasst und als Triggerbeitrag an den CTP gesendet. Quelle: [25]

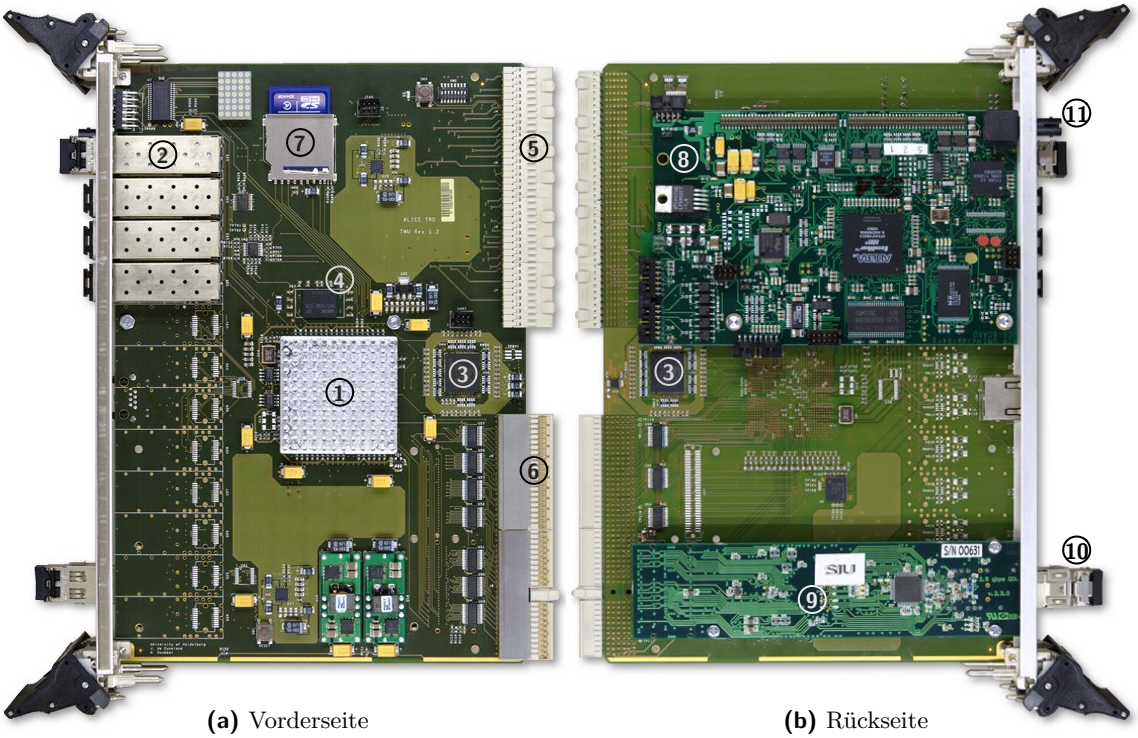
Anbindung an den CTP, und fasst sowohl Triggerentscheidungen als auch *Busy*-Signale aller Segmente zusammen.

Der Informationsaustausch innerhalb eines Segments erfolgt über eine auf geringe Latenz optimierte LVDS-Backplane und eine Standard-CompactPCI-Backplane. Dabei überträgt die LVDS-Backplane alle ereignisspezifischen Daten wie Rohdaten und Triggerberechnungen. Außerdem ermöglicht sie die Kommunikation zwischen der SMU und den TMUs und transportiert die benötigten Steuersignale. Die CompactPCI-Backplane wird zur Energieversorgung verwendet. Für die Datenübertragung zwischen den Segmenten und der TGU stehen schnelle LVDS-Verbindungen auf Basis von Twisted-Pair-Kabeln zur Verfügung.

3.2 Hardware

Trotz der stark unterschiedlichen Aufgaben basieren die Karten aller drei Stufen auf dem gleichen Platinenlayout. Die Bestückung ist auf die unterschiedlichen Einsatzzwecke angepasst. Als Beispiel ist in Abbildung 3.2 ein Foto einer SMU gezeigt. Im Folgenden sind die gemeinsamen Komponenten aller Karten beschrieben. Danach werden die TMU und SMU nochmals im Detail vorgestellt.

Die Hauptkomponente jeder Karte ist ein *Xilinx Virtex-4 FX100* FPGA [32]. Es bietet neben fast 95 000 frei programmierbaren Logikzellen auch sogenannte *HardIP*-Blöcke, in denen auf bestimmte Aufgaben spezialisierte Komponenten fest in Silizium realisiert sind.



(a) Vorderseite **(b)** Rückseite

1. FPGA
2. SFP-Modul
3. SRAM
4. SDRAM
5. LVDS-Stecker
6. cPCI-Stecker
7. SD-Karte
8. DCS-Board
9. SIU
10. DDL-Interface
11. TTC-Link

Abbildung 3.2: Foto einer SMU. Das FPGA befindet sich unter dem silbernen Kühlkörper. Für die bei der SMU nicht verwendeten SFPs sind nur vier Steckplätze bestückt. Auf der Rückseite sind das DCS-Board und die SIU zu sehen. Foto: [18]

Von diesen werden in der GTU momentan die Multi-Gigabit Serial Transceiver zum Empfang der Detektordaten und die beiden PowerPC-Kerne zur Realisierung eines eingebetteten Prozessorsystems verwendet. Konfiguriert wird das FPGA über ein Konfigurations-EEPROM oder eine JTAG-Schnittstelle zur Backplane.

Zur Datenspeicherung verfügt jede Karte über zwei externe Speicher: einen schnellen SRAM mit $2 \cdot 18\,\text{Mbit}$, der zur Pufferung der Ereignisdaten benötigt wird, und einen 512-Mbit-SDRAM, der als Arbeitsspeicher für die PowerPCs dienen kann. Zusätzlich stehen über eine SD-Karte 4 GB Flash-Speicher zur Verfügung. Als weitere Komponenten enthält jede Karte verschiedene Sensoren, die zur Systemüberwachung verwendet und über mehrere I^2C-Busse angesteuert werden.

3.2.1 Die Track Matching Unit (TMU)

Die TMUs empfangen und verarbeiten Trigger- und Rohdaten aus dem Detektor. Die FEE sendet diese über eine eigens entwickelte optische Übertragungsstrecke [28]. Die

Daten werden dabei 8b/10b-codiert mit 2,5 Gbit/s pro Link übertragen. Zum Empfang sind die TMUs mit je 12 optischen Transceivern bestückt. Dabei werden industrieübliche SFPs-Module[1] verwendet. Die seriell empfangenen Daten werden direkt an die Multi-Gigabit Serial Transceiver (MGT)-Blöcke der FPGAs weitergeleitet, in denen die 8b/10b-Decodierung erfolgt und die Daten parallelisiert werden. Der eingehende Gesamtdatenstrom einer TMU beträgt rund 30 Gbit/s. Zur Pufferung der Daten verwenden die TMUs den schnellen externen SRAM.

3.2.2 Die Supermodule Unit (SMU)

Die Hardwareausstattung der SMU resultiert aus ihrer Aufgabe als Steuereinheit und Anbindung an das TTC- und DAQ-System. Da die SMU keine Ereignisdaten direkt von der FEE empfängt, ist sie nur mit vier SFPs für zukünftige Erweiterungen ausgestattet. Auf der Rückseite trägt die SMU zwei Aufsteckkarten:

Das *Detector-Control-System-Board* (DCS) realisiert die Anbindung an das TTC-System und wird für Kommunikations- und Konfigurationszwecke genutzt. Es verfügt über einen *Altera Excalibur*-Chip, bestehend aus einem vollwertigen ARM-Prozessorsystem und einem FPGA. Damit ist ein System On a Programmable Chip (SOPC) realisiert, auf dem ein Linux-System läuft. Nach außen verfügt das DCS-Board über eine Ethernet-Schnittstelle zur Kommunikation und einen TTCrx-Chip zum Empfang der TTC-Signale [29]. Um die FPGAs und Konfigurations-EEPROMs der GTU programmieren zu können, ist es mit allen JTAG-Schnittstellen eines Segments verbunden. Über eine geteilte UART-Schnittstelle kann über das DCS-Board mit allen Karten eines Segments kommuniziert werden. Außerdem ermöglicht es mittels einer Verbindung zu den I^2C-Sensor-Bussen der GTU-Karten eine unabhängige Überwachung der Betriebsparameter der FPGAs und anderer Komponenten der Karten.

Die Verbindung zur DAQ wird von der *Source Interface Unit (SIU)* hergestellt. Sie ist Teil des DDL und bildet das Interface zwischen der GTU und dem *physical layer* des DDL. Das SIU-SMU-Interface ist durch einen bidirektionalen Bus realisiert, der von der SIU gesteuert wird. Bei einer Datenbus-Breite von 32 Bit und einer maximal erlaubten Taktrate von 50 MHz können Daten mit maximal 200 MB/s an die SIU übertragen werden. Da der DDL über eine Flusskontrolle verfügt, kann die tatsächlich zur Verfügung stehende Bandbreite geringer ausfallen [30].

3.3 Das PowerPC-System

Neben den Kernaufgaben der GTU, die von dedizierten FPGA-Einheiten übernommen werden, ist für den Betrieb im Experiment ein umfangreiches Konfigurations- und Kontrollsystem notwendig. Hilfreich ist es außerdem, aussagekräftige Statistikdaten über Betriebs-

[1] SFP: Small Form-factor Pluggable; Standard für elektrische und optische Transceiver, die in der Netzwerktechnik eingesetzt werden.

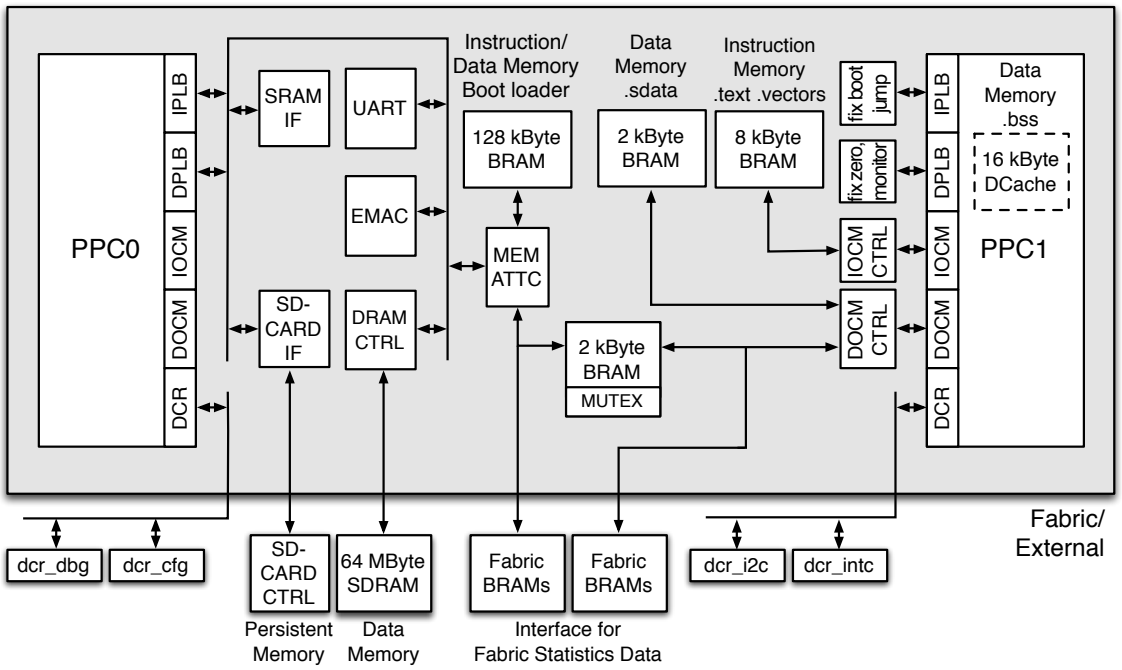


Abbildung 3.3: Schematische Übersicht des Dual-Prozessor-PPC-Systems. Die beiden Bussysteme sind für verschiedene Aufgaben optimiert. Der Datenspeicher von PPC1 ist vollständig im Cache implementiert. PPC0 kann auf den externen SDRAM zugreifen und verfügt über eine UART-Schnittstelle zur Kommunikation. Von beiden Kernen kann zusätzlich auf Speicher im FPGA-Fabrik zugegriffen werden.

parameter zu sammeln und zur Verfügung zu stellen. Um Hardware-Ressourcen zu sparen und umfangreiche arithmetische Operationen zu ermöglichen, kommen hierfür die zwei im FPGA vorhandenen PPC-Kerne zum Einsatz. Sie bilden in Verbindung mit in FPGA-Logik realisierten Systemkomponenten ein eingebettetes Dual-Prozessor-PPC-System.

Ein schematischer Überblick über das PPC-System ist in Abbildung 3.3 dargestellt. Das PPC1-System ist optimiert für die Echtzeiterfassung von Statistikdaten auf der Zeitskala einzelner Ereignisse. Der Systemaufbau und die optimierte Interrupt-Latenz von unter 400 ns ermöglichen Verarbeitungsraten von über 8 kHz. Dies ist ausreichend, um auch bei maximalen Raten von Kollisionsereignissen Statistikdaten zu erfassen. Gleichzeitig ist darauf geachtet, den Ressourcenverbrauch so gering wie möglich zu halten. Den beiden Kernen kommen unterschiedliche Aufgaben zu. *PPC1* wird zur Echtzeiterfassung der Statistikdaten und Systemüberwachung eingesetzt. Er überwacht interne Betriebsparameter, die vom DCS-Board direkt nicht zugänglich sind. Die Statistikdaten werden von spezialisierten Hardwarekomponenten erfasst und in Interruptroutinen vom PPC1 abgerufen, verarbeitet und gespeichert. *PPC0* übernimmt die Aufarbeitung der Daten und die Kommunikation mit dem DCS [25].

Entsprechend der Aufgaben sind die verwendeten Systemkomponenten angepasst. Beispielsweise verfügen die beiden Kerne über völlig unterschiedliche Speicherstrukturen. Der

Datenspeicher von PPC1 ist weitgehend im Cache implementiert, um eine möglichst geringe Interruptlatenz zu erreichen. PPC0 kann über einen PLB-Bus auf den externen 64-MByte-SDRAM zugreifen. Neben den für das eigentliche System verwendeten Komponenten gibt es Speicher und Register, die für das PPC-System und das FPGA-Design zugänglich sind. Sie werden als Konfigurationsregister für in FPGA-Logik implementierte Komponenten und zum Austausch von erfassten Statistikdaten verwendet. Die Kommunikation zwischen den beiden Kernen wird durch einen gemeinsamen BRAM mit Hardware-Mutex-Unterstützung ermöglicht. Zur Kommunikation verfügt PPC0 über eine UART-Schnittstelle, die mit dem DCS-Board verbunden ist. Über einen speziellen Treiber im DCS-Board-Linux ist so die externe Kommunikation mit dem GTU-FPGA möglich [31]. Hierüber ist die Verbindung der GTU zum DCS realisiert. Mit Hilfe spezieller Kommandos können Konfigurationsregister und Speicher geschrieben und ausgelesen werden. In Verbindung mit der externen SD-Karte ist es möglich, auf PPC0 ein eingebettetes Linuxsystem zu betreiben [21].

Mit Hilfe der Konfigurationsmöglichkeiten durch das PPC-System können ohne großen Aufwand flexible, konfigurierbare Hardwarekomponenten im GTU-FPGA realisiert werden. Die Option, Statistikdaten und Messgrößen nach außen zugänglich zu machen, eröffnet umfangreiche Kontroll- und Diagnosefunktionen.

4 Datenauslese und Event Buffering

Die GTU bildet den zentralen Teil der mehrstufigen TRD-Auslesekette. Der Datenfluss zu einer vollständigen Triggersequenz ist in Abbildung 4.1 dargestellt. Die ersten Daten erhält die GTU in Form von Tracklets kurz nach einem L0-Trigger. Diese werden, zusätzlich zu ihrer Verwendung bei der Berechnung des TRD-Triggerbeitrags, für die spätere Auslese in den Ereignispuffern gespeichert. Nach einem L1-Trigger sendet die FEE die ADC-Rohdaten des Ereignisses an die GTU, die ebenfalls in den Ereignispuffern gespeichert werden. Ist diese Übertragung abgeschlossen, befindet sich ein Ereignis vollständig im Speicher der GTU. Dort muss es verbleiben, bis es nach der L2-Entscheidung an die DAQ gesendet oder verworfen wird.

Wie bereits in Kapitel 3 erwähnt, sind die dafür benötigten Komponenten auf TMU und SMU verteilt. Die TMUs übernehmen den Empfang der Daten und puffern diese für den benötigten Zeitraum. Die SMUs steuern die Vorgänge eines Segments entsprechend der Triggersequenzen. Bei der Auslese von Ereignissen fasst die SMU die Daten von fünf TMUs zusammen, versieht sie mit Headern und sendet die Ereignisse an die DAQ. Die Logik, die von TMU und SMU zur Realisierung dieser Aufgaben verwendet wird, ist im Folgenden beschrieben.

4.1 Steuerung der Datennahme durch den CTP

Die Datennahme des Experiments wird vom CTP über das Auslösen einzelner Triggerstufen gesteuert. Die Informationen, welche die Detektoren zu einer bestimmten Triggerstufe benötigen, werden vom TTC-System übertragen. Dabei handelt es sich um die Triggerentscheidung selbst und einige Zusatzinformationen, die beispielsweise zur Ereignisidentifikation benötigt werden.

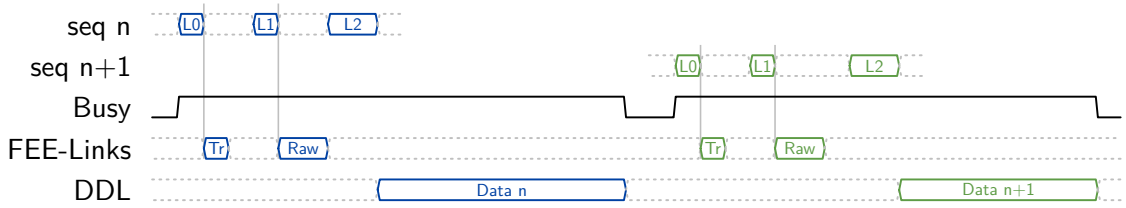


Abbildung 4.1: Datenfluss in der GTU während einer Triggersequenz: Zum L0- und L1-Trigger empfängt die GTU Daten von der FEE. Nach einem L2a-Trigger sendet die GTU ein Ereignis über den DDL an die DAQ.

Ein L0-Trigger wird durch einen eine Periode breiten Puls auf dem A-Kanal repräsentiert. Für einen L1-Trigger (*L1-accept*) wird eine definierte Anzahl an Takten nach einem L0-Trigger ein zwei Takte breiter Puls gesendet. Ein Abbruch der Triggersequenz (*L1-reject*) wird durch das Ausbleiben eines Signals zu diesem Zeitpunkt signalisiert; Ein spezielles L1-reject-Signal ist nicht vorgesehen. Bei einem L1-accept-Trigger wird zusätzlich eine L1-Triggernachricht auf dem B-Kanal gesendet. Sie gibt unter Anderem an, um welchen Trigger-Typ es sich handelt. Für einen L2-Trigger wird kein Triggerpuls, sondern nur eine Triggernachricht gesendet. Bei einem *L2-accept* besteht die Nachricht aus acht Worten und enthält unter Anderem die *Event-ID*, welche ein Ereignis eindeutig identifiziert. Bei einem *L2-reject* besteht die Nachricht aus einem Wort, das ein Subset der Event-ID enthält.

Die Übertragungen über den B-Kanal haben unterschiedliche Prioritäten. Eine L1-Nachricht zum Beispiel unterbricht die Übertragung einer L2-Nachricht. Dadurch ist der genaue Zeitpunkt, zu dem ein L2-Trigger eintrifft, nicht definiert, obwohl der CTP den L2-Trigger zu einem festen Zeitpunkt auslöst. Die spezifizierte Maximalverzögerung für das Eintreffen eines L2-Triggers beträgt 500 μs.

Als Vergleichswert für die Ereignisidentifikation verfügen alle Detektoren über einen *Event-ID-Zähler*. Zur Synchronisation dieser Zähler sendet das TTC-System zu Beginn jedes LHC-Orbits[1] einen *BC-Reset*. Dies geschieht als vorrangige Nachricht auf dem B-Kanal. Dadurch ist sichergestellt, dass alle Zähler synchron laufen. Durch unterschiedliche Verzögerungen der Übertragungsstrecken können die Absolutwerte jedoch um einen festen Betrag abweichen.

4.2 Event-Buffering in der TMU

Die Event-Buffering-Einheiten der TMUs wurden im Rahmen der Diplomarbeit von Felix Rettig entwickelt und sind in [28] detailliert beschrieben. Der Aufbau und die relevanten Kernpunkte sind im Folgenden zusammengefasst.

Eine der Herausforderungen beim Event-Buffering ist es, 12 unabhängige Datenströme mit einer Netto-Datenrate von 23 Gbit/s pro TMU in Echtzeit zu einem dichten Datenstrom zusammenzufassen und zu speichern. Da im FPGA intern nicht genügend Speicher zur Verfügung steht, werden zwei externe SRAMs genutzt. Beide SRAMs werden parallel verwendet und bieten so ein 200-MHz-Dateninterface mit einer Breite von 128 Bit.

Die Datenverarbeitung erfolgt vollständig im FPGA. Der schematische Aufbau des TMU-Designs ist in Abbildung 4.2 dargestellt. Jede TMU empfängt 12 unabhängige Datenströme von der FEE. Nachdem diese SFPs und MGT-Blöcke durchlaufen haben, stehen sie im FPGA als 16-Bit-Datenbusse synchron zu einem globalen 125-MHz-Takt zur Verfügung. Die weiteren Verarbeitungsschritte bis zur Speicherung der Daten erfolgen im *Event Shaper*. Dort werden die Datenströme zunächst mit Hilfe von *Aligning Buffern* in eine gemeinsame 200-MHz-*Clock-Domain* synchronisiert und auf eine Breite von jeweils

[1] Als LHC-Orbit wird die Zeitspanne bezeichnet, die die Teilchenpakete benötigen, um einmal den kompletten Ring zu umrunden. Dies geschieht mit etwa 11 kHz.

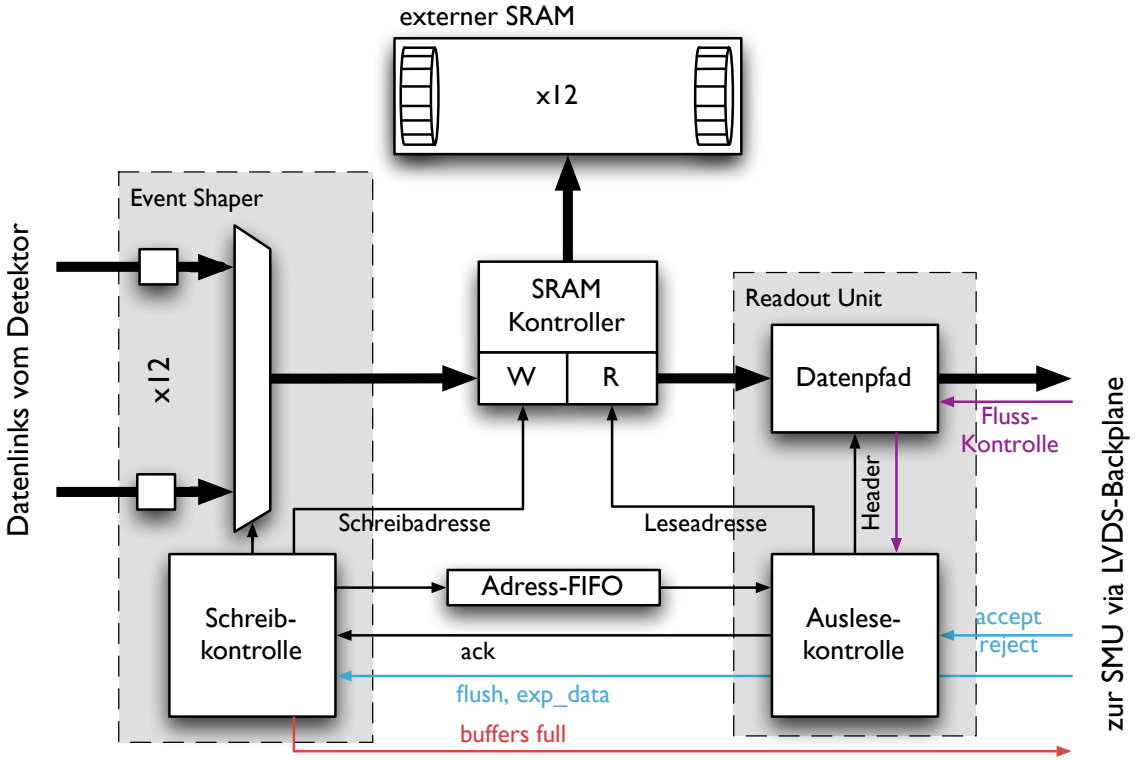


Abbildung 4.2: Datenpfad der TMU. Der Event Shaper nimmt 12 Datenströme der FEE entgegen und speichert diese im externen SRAM. Fordert die SMU ein Ereignis an, wird es von der Readout Unit ausgelesen und über die LVDS-Backplane an die SMU übertragen.

128 Bit umgesetzt, entsprechend der Breite des SRAM-Interfaces. Anschließend wird aus den einzelnen, mit Lücken durchsetzten, Datenströmen ein dichter 128-Bit-Datenstrom zum Schreiben in den SRAM geformt. Um die Daten effizient aus dem Speicher lesen zu können, ist der SRAM in 12 logisch unabhängige Segmente aufgeteilt, eines für jeden Datenstrom. Während des Datenempfangs werden bis zu 94 % aller Taktzyklen zum Schreiben genutzt. Eine entsprechende Priorisierung der Schreib- über Lesezugriffe erfolgt mittels eines speziell entwickelten SRAM-Controllers.

Das Auslesen von Ereignissen aus dem Speicher wird von der *Readout Unit* gesteuert. Wird ein Ereignis von der SMU angefordert, wird zunächst ein *Stack-Header* mit Zusatzinformationen über das Ereignis generiert. Dann werden die 12 Datensätze eines Ereignisses hintereinander aus dem SRAM gelesen und zu einem fortlaufenden Ereignis zusammengesetzt. Aufgrund der Priorisierung wird nur in den zeitlichen Lücken gelesen, in denen keine Ereignisdaten empfangen werden. Ab dieser Stelle im Datenpfad existiert eine bidirektionale Flusskontrolle. Der Datenfluss kann sowohl von der Readout Unit, als auch von der die Daten entgegennehmenden SMU unterbrochen werden. Ist ein Ereignis ausgelesen, signalisiert die Readout Unit dem Event Shaper, dass der Speicherbereich wieder verwendet werden kann.

Anfang und Ende von Ereignissen sind im Datenstrom stets mit *Headern* und *End Markern* gekennzeichnet. Um bei Ausfall einzelner Links oder sonstigem fehlerhaftem Verhalten den Datenstrom in allen Stufen weiterhin verarbeiten zu können, sind dem gesamten Datenpfad *Link Monitore* vorgeschaltet. Sie überwachen kontinuierlich den Datenstrom und können ihn bei Bedarf unterbrechen, End Marker einfügen und Teile als ungültig markieren. Dadurch ist sichergestellt, dass die Auslese auch im Fehlerfall korrekt beendet wird. Lediglich bei der Offline-Analyse der Daten ist auf das Gültigkeits-Flag zu achten.

Um mehrere Ereignisse speichern zu können, sind die 12 Speichersegmente als unabhängige *Ringpuffer* organisiert. Die Daten eines Links werden dazu an aufeinander folgenden Adressen im Speicher abgelegt. Holen sich Schreibadresse und Beginn der ältesten gespeicherten Daten ein, ist der Puffer voll. Informationen über Beginn und Ende jedes Ereignisses in den einzelnen Speicher-Segmenten sowie Zusatzinformationen wie Fehlerbedingungen, die beim Empfang der Ereignisse entstanden sind, werden im *Event Info FIFO* gespeichert, um für die Auslese zur Verfügung zu stehen.

4.3 Segmentsteuerung und Datenauslese in der SMU

Eine Version der Segmentsteuerung und Datenauslese der SMU, die in der Lage ist, einzelne Ereignisse zu verarbeiten, wurde im Rahmen der Diplomarbeit von Stefan Kirsch entwickelt und ist in [24] ausführlich beschrieben. Um die Eingliederung der, in Kapitel 6 beschriebenen, Multi-Event-Buffering-Steuereinheit in das Gesamtdesign besser zu verstehen, wird der Aufbau im Folgenden kurz vorgestellt. Ein schematischer Überblick über die relevanten Teile des SMU-Designs ist in Abbildung 4.3 dargestellt.

4.3.1 Decodierung der TTC-Signale

Die GTU empfängt das TTC-Signal über den, auf dem DCS-Board vorhandenen, TTCrx-Chip. Der Chip ist in der Lage, das TTC-Signal zu decodieren und die enthaltenen Informationen wie Trigger, Triggernachrichten und Synchronisationsinformationen an seinen Ausgangspins bereitzustellen. Aufgrund der begrenzten I/O-Ressourcen des FPGA verwendet die GTU diese Funktion nicht. Stattdessen werden die seriellen Ausgänge des A- und B-Kanals vom DCS-Board-FPGA zeit-gemultiplext zusammen mit dem wiederhergestellten LHC-Takt als LVDS-Signale direkt an die SMU weitergeleitet.

Die vollständige TTC-Decodierungslogik ist im FPGA der SMU implementiert. Diese Aufgabe übernimmt die *TTC Decode Unit*. Die A-Kanal-Signale werden entsprechend ihren Pulsbreiten einem L0- oder L1-Trigger zugeordnet und auf den entsprechenden Ausgängen der TTC Decode Unit signalisiert. Der serielle Datenstrom des B-Kanals wird Hamming-decodiert und die einzelnen Worte den entsprechenden Nachrichten und Kommandosequenzen zugeordnet. Steht eine L1-Triggernachricht an einem Ausgang zur Verfügung, wird dies durch ein `l1m_rdy` signalisiert. Bei den L2-Triggern signalisiert das `l2a` bzw. `l2r` sowohl, um welchen Trigger es sich handelt, als auch das Vorhandensein der Triggernachricht. Das Eintreffen eines BC-Resets wird durch das gleichnamige Signal angezeigt.

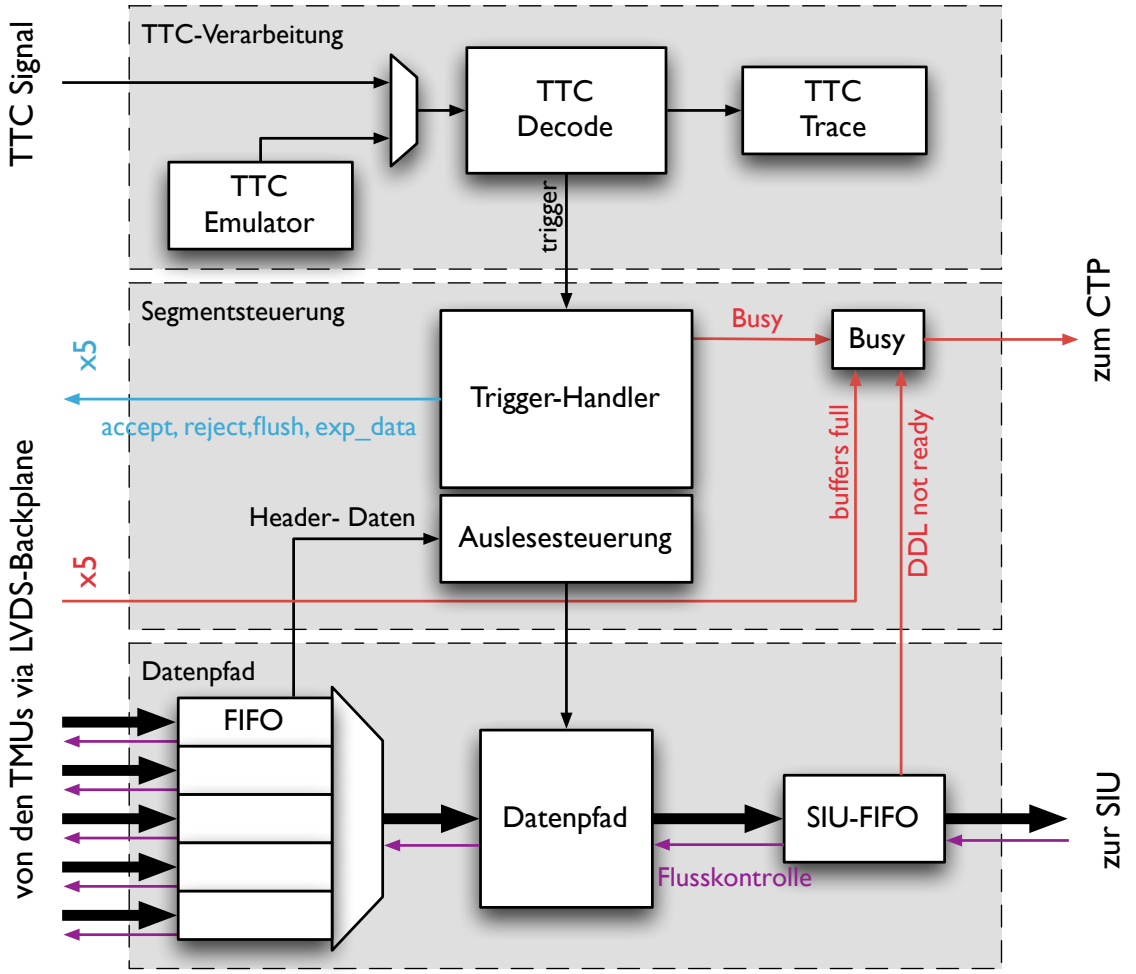


Abbildung 4.3: Vereinfachte Darstellung der Segmentsteuerung und des Datenpfads der SMU. Die erzeugten Kontrollsignale werden über die LVDS-Backplane an die TMUs weitergegeben. Nach einem L2a werden die Ereignisdaten von fünf TMUs durch die SMU zyklisch ausgelesen und an die DAQ gesendet.

Übertragungsfehler, wie zum Beispiel Hammingfehler auf dem B-Kanal oder ein Dreifachpuls auf dem A-Kanal, werden von der TTC Decode Unit erkannt und über entsprechende Fehlerausgänge signalisiert. Es erfolgt an dieser Stelle keine Überprüfung auf zeitliches Verhalten oder Konsistenz der Triggersequenz.

Um bei der Fehlersuche überprüfen zu können, welche Triggersignale an die GTU gesendet wurden, speichert der *TTC Trace* alle eintreffenden Trigger und Nachrichten mit Zeitstempeln in einem internen Speicher des FPGAs. Bei typischen Triggerraten kann mit der aktuellen Speichergröße ein Zeitraum von mehr als 500 μs überdeckt werden. Auf die Daten kann über das PPC-System und das DCS-Board von außen zugegriffen werden, um die Historie der Triggersignale zu analysieren.

4.3.2 Segmentsteuerung

Zentrale Aufgabe der SMU ist es, alle Abläufe innerhalb eines Segments zu steuern. Einige der dazu verwendeten Steuersignale sind in Abbildung 4.3 gezeigt. Die meisten Steuersignale werden in Abhängigkeit von den Triggersignalen des CTP generiert und beeinflussen zum Beispiel des Event-Buffering der TMUs. Die Steuereinheit generiert zudem das *Busy*-Signal, welches dem CTP signalisiert, dass das entsprechende TRD-Modul nicht bereit ist, Ereignisse aufzunehmen. Eine detaillierte Beschreibung der GTU-Segmentsteuerung folgt in Kapitel 6.

4.3.3 Auslesen eines Ereignisses

Das Auslesen von Ereignissen aus den TMU-Ereignispuffern, Zusammenfügen zu einem segmentweiten Datenstrom und Senden an die DAQ wird von der *Readout Control Unit* gesteuert. Sie initiiert die nötigen Abläufe im Datenpfad.

Der SMU-Datenpfad verarbeitet 32-Bit-Datenworte mit einer Taktfrequenz von 60 MHz. Soll ein Ereignis ausgelesen werden, wird dies den TMUs signalisiert, und die TMU Readout Units beginnen, Daten zu senden. Um die Datenströme, die durch vorrangige Schreibzugriffe auf den SRAM der TMUs durchbrochen sein können, zu glätten, durchlaufen sie FIFOs am Dateneingang der SMU. Sind die FIFOs initial voll, wird die Auslese über die Flusskontrolle zunächst angehalten, bis der Common Data Header (CDH) erstellt ist.[2] Dies kann nicht im Voraus geschehen, da hierfür Informationen aus den von den TMUs gesendeten Stack-Headern benötigt werden. Ist der CDH erstellt, werden alle Header in fester Reihenfolge an die DAQ gesendet. Anschließend werden die TMUs nacheinander ausgelesen und die Ereignisdaten zu einem kontinuierlichen Datenstrom zusammengefügt. Der genaue Aufbau des Datenstroms und Inhalt der Header ist in [19] beschrieben. Zum

[2] Der CDH enthält Zusatzinformationen über ein Ereignis, wie beispielsweise eine Ereignis-ID und Daten aus den Triggernachrichten. Er muss von der FEE erzeugt und zu Beginn jedes Ereignisses an die DAQ gesendet werden.

Senden an die DAQ müssen die Daten an die SIU übergeben werden. Vor der SIU befindet sich ein weiteres FIFO, das den Datenstrom weiter glättet und vom 60-MHz-Takt des Datenpfades auf den maximalen Takt des SIU-Interfaces von 50 MHz synchronisiert.

Da der DDL über eine Flusskontrolle verfügt, ist nicht garantiert, dass in jedem Takt Daten gesendet werden können. In der GTU wird Flusskontrolle intern über alle Stufen des SMU-Datenpfades bis an die TMU-Readout Units weitergeleitet.

5 Multi-Event Buffering (MEB)

Bei modernen Experimenten der Teilchenphysik sind interessante Ereignisse sehr selten und der Zeitpunkt ihres Auftretens, aufgrund ihrer statistischen Natur, nicht vorhersagbar. Um bei solch geringen Wirkungsquerschnitten aussagekräftige Statistik während der vorgesehenen Experiment-Laufzeiten akkumulieren zu können, stellen sich besondere Anforderungen an das verwendete Detektorsystem. Die Statistik des ALICE-Experiments ist beispielsweise limitiert durch das seltene Auftreten von Υ. Für eine Luminosität von $L = 10^{27}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ erwartet man etwa 13 600 nicht überlagerte Ereignisse pro ALICE-Jahr, von denen beide Zerfallselektronen eines Υ im Zentralbereich liegen [10]. Ein komplexes Triggersystem ist notwendig, um die physikalisch interessanten Ereignisse effizient zu selektieren. Nicht minder wichtig ist es, das Detektorsystem möglichst immer in einem aufnahmebereiten Zustand zu halten, also die Totzeit der beteiligten Detektor-Subsysteme zu minimieren.

Im folgenden Kapitel wird dargestellt, wie die auslesebedingte Totzeit von Teilchendetektoren von Faktoren wie Ausleserate und Bearbeitungszeit eines Ereignisses abhängt. Dabei wird gezeigt, dass sich durch Multi-Event Buffering (MEB) die Totzeit von Teilchendetektoren deutlich reduzieren lässt. Die abgeleiteten Erkenntnisse werden dann auf den TRD übertragen, um zu einer Abschätzung zu gelangen, wie weit die Totzeit des TRD verringert werden kann.

5.1 Grundlagen

Die Totzeit eines Teilchendetektors bezeichnet die Zeitspanne nach einem Ereignis, innerhalb derer der Detektor nicht in der Lage ist, ein neues Ereignis aufzunehmen. Da die Totzeit alleine keine ausreichende Aussage über die Leistungsfähigkeit eines Detektors im Experiment macht, wird sie oft als relative Totzeit D – bezogen auf die Gesamtzeit der Datennahme – angegeben. Wie leicht nachzuvollziehen ist, vermindert eine große Totzeit die Anzahl der aufgenommenen Ereignisse bei fester Laufzeit. Bei vorgegebener Eingangsrate müssen um den Faktor $\frac{D}{1-D}$ länger Daten aufgenommen werden, um die angestrebte Anzahl Ereignisse zu sammeln. Anders betrachtet, muss die Eingangsrate um den Faktor $\frac{D}{1-D}$ vergrößert werden, um eine angestrebte Ausleserate zu erreichen. Beide Ansätze lassen sich im Experiment meist nur schwierig realisieren.

Bei der Totzeit lassen sich grob zwei Bestandteile unterscheiden: Einer ist die intrinsische Totzeit des Detektors, die durch die physikalischen Eigenschaften des Nachweisprinzips

bestimmt ist. Hierzu gehört beispielsweise die Driftzeit der Elektronen in einem Spurkammerdetektor. Die intrinsische Totzeit lässt sich durch die Optimierung des Detektors, im gegebenen Beispiel etwa die Erhöhung der Driftgeschwindigkeit, verringern. Solchen Maßnahmen sind jedoch meist enge physikalische Grenzen gesetzt. Der zweite Bestandteil der Totzeit wird durch das Auslesen der Ereignisse und die Übertragung an das Datenaufnahmesystem hervorgerufen. Im Folgenden wird diese auslesebedingte Totzeit näher untersucht.

Die auslesebedingte Totzeit hängt von zwei Parametern ab. Sei R_{out} die mittlere Ausleserate, mit der Ereignisse ausgelesen werden und τ die mittlere Bearbeitungszeit, die dazu benötigt wird. Sofern jeweils nur ein Ereignis gespeichert und verarbeitet werden kann, löst jedes Ereignis eine absolute Totzeit der Dauer τ aus. Dann ergibt sich die prozentuale Totzeit D zu

$$D = \frac{\text{abs. Totzeit}}{\text{Gesamtzeit}} = \frac{\tau}{\frac{1}{R_{\text{out}}}} = \tau \cdot R_{\text{out}} \ . \tag{5.1}$$

Die Totzeit ist also direkt proportional zu Ausleserate und Übertragungszeit. Sie erreicht ihr Maximum bei der durch $R_{\text{out}}^{\text{max}} = \frac{1}{\tau}$ gegebenen maximalen Ausleserate. Bezogen auf die mittlere Ereignisrate R zufällig verteilter Ereignisse aus denen ausgewählt wird, ergibt sich

$$D = \frac{\tau}{\frac{1}{R} + \tau} = \frac{x}{1+x} \ , \text{ mit } x = \tau \cdot R \ . \tag{5.2}$$

Dabei entspricht $\frac{1}{R}$ der mittleren Zeit bis zum nächsten Ereignis. Zusammen mit der Bearbeitungszeit τ ergibt sich also die Gesamtzeit $\frac{1}{R} + \tau$. Die Größe x charakterisiert die Betriebsbedingungen des Detektors. Die Totzeit in Abhängigkeit von x ist als Kurve für $n = 1$ in Abbildung 5.1 dargestellt. Die Funktion wächst zunächst schnell mit steigender Eingangsrate und nähert sich dann asymptotisch der maximalen Totzeit an. Am Punkt $x = 1$, an dem die mittlere Eingangsrate genau der maximalen Verarbeitungsrate entspricht ($R = R_{\text{out}}^{max}$), kann im Mittel jedes zweite Ereignis nicht aufgenommen werden. Die relative Totzeit beträgt an diesem Punkt erwartungsgemäß 50 %.

5.2 Reduzierung der Totzeit durch Multi-Event Buffering

Um die auslesebedingte Totzeit zu verringern, können verschiedene Ansätze verfolgt werden. Aus den in Abschnitt 5.1 angegebenen Gleichungen lässt sich ableiten, dass eine geringere Totzeit durch eine Verringerung der Bearbeitungszeit erreicht werden kann. Dieser Ansatz gestaltet sich meist jedoch sehr kostenintensiv. Eine elegantere Lösung ergibt sich durch das Zwischenspeichern mehrerer Ereignisse. Kann mehr als ein Ereignis gespeichert werden, so können bereits neue Ereignisse aufgenommen werden, während ältere noch bearbeitet werden. Bearbeitungszeit und Totzeit werden somit voneinander entkoppelt.

Angenommen, die zur Verfügung stehende Puffertiefe sei $n = \infty$: Solange die mittlere Ereignisrate kleiner ist als die maximale Verarbeitungsrate, entsteht keine Totzeit. Alle Ereignisse können aufgezeichnet werden, da sich die Puffer im Mittel schneller leeren als sie gefüllt werden. Wird die mittlere Ereignisrate größer als die maximale Verarbeitungsrate,

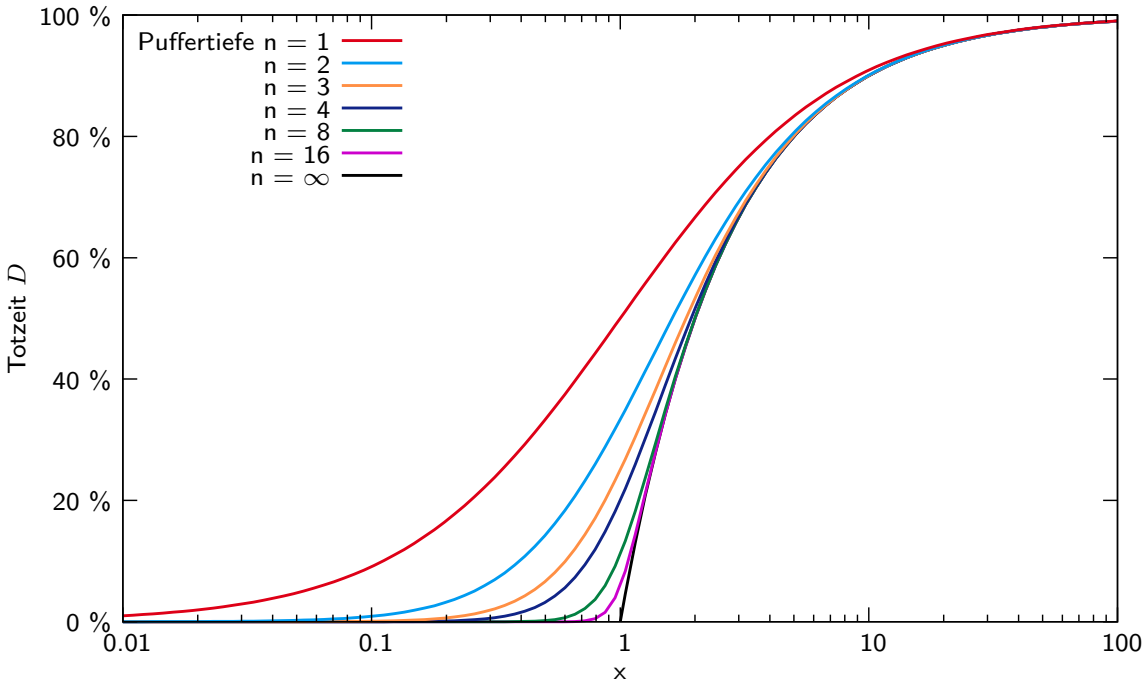


Abbildung 5.1: Auslesebedingte Totzeit eines Detektorsystem, aufgetragen gegen den Parameter $x = R \cdot \tau$ für verschiedene Puffertiefen

wächst die Anzahl der gepufferten Ereignisse unaufhaltsam. Zum Angleichen der Raten müssen überzählige Ereignisse abgelehnt werden. Die Totzeit wächst ab diesem Punkt mit der Eingangsrate. Es ergibt sich (vgl. [22])

$$D_\infty = \begin{cases} 0 & x \leq 1 \ , \\ \frac{x-1}{x} & x > 1 \ . \end{cases} \tag{5.3}$$

Diese Kurve ist in Abbildung 5.1 dargestellt. Man erkennt, dass im Falle unendlicher Puffertiefe bis $x = 1$ keine Totzeit entsteht.

Können zwischen ein und unendlich vielen Ereignisse im Puffer gespeichert werden, so spielen statistische Effekte eine Rolle. Solange mindestens ein Platz im Puffer frei ist, können – wie im Grenzfall unendlich vieler Plätze – immer weiter Ereignisse aufgenommen werden und die Totzeit verschwindet. Treffen kurzfristig mehr Ereignisse als im zeitlichen Mittel ein, kann die maximale Puffertiefe erreicht werden. Gleiches gilt, wenn die Bearbeitung – etwa durch größere Ereignisse – kurzfristig länger dauert.[1] Es können vorübergehend keine Ereignisse mehr aufgenommen werden und Totzeit entsteht. Die Totzeit hängt also von den Eigenschaften der Verteilungen von Ereignisrate und Bearbeitungszeit ab. Entscheidend ist, wie groß die Wahrscheinlichkeit ist, durch eine kurzfristige Überhöhung der Rate oder Bearbeitungszeit, die maximale Puffertiefe zu erreichen. Dieses statistische Problem kann mit Hilfe der Warteschlangentheorie [14] gelöst werden. Wird für die Zeit zwischen

[1] Die mittlere Ereignisrate und Bearbeitungszeit bleiben konstant. Eine Schwankung in diesen Größen würde sich in einer Schwankung in x ausdrücken.

zwei Ereignissen und die Bearbeitungszeit eine Exponentialverteilung angenommen, ergibt sich für die Totzeit

$$D_n = \frac{x^n(1-x)}{1-x^{n+1}} \ . \tag{5.4}$$

Da die Ausleserate der, um die Totzeit korrigierten, Eingangsrate entsprechen muss, gilt außerdem

$$R_{\text{out}} = R(1-D_n) \ . \tag{5.5}$$

Kurven für unterschiedliche Puffertiefen n, $1 \leq n \leq \infty$, sind beispielhaft in Abbildung 5.1 dargestellt. Je größer die Tiefe der Puffer ist, desto höhere Eingangsraten können verarbeitet werden ohne die Totzeit übermäßig anwachsen zu lassen, da statistische Schwankungen besser ausgeglichen werden können. Aus der Abbildung kann abgelesen werden, wie tief die Puffer eines Detektors ausgelegt werden müssen. Soll ein Detektor beispielsweise bei $x = 0{,}4$ betrieben werden, genügt eine Puffertiefe von $n = 8$, damit keine auslesebedingte Totzeit entsteht.

5.3 Auslesedauer und Totzeit des TRD

Die erwartete Ereignisrate von Pb-Pb-Kollisionen, aus denen das Triggersystem eine Auswahl treffen kann, liegt bei etwa 8 kHz. Angestrebte Betriebsbedingungen sind eine Ausleserate von mindestens 200 Hz und eine dabei verursachte Experimenttotzeit, die 10 % nicht überschreitet.

Um abschätzen zu können, wie groß die Totzeit des TRD ist und welche Verbesserung durch MEB erreicht werden kann, muss die Auslesekette der TRD genauer analysiert werden. Von besonderem Interesse sind dabei die Bearbeitungszeiten eines Ereignisses. Die Bearbeitung eines Ereignisses im TRD besteht aus vier Stufen: Die Wartezeiten bis zur L1- und L2-Triggerentscheidung sowie die Übertragungszeiten von der FEE zur GTU und von der GTU zur DAQ. Die Bearbeitungszeiten der Triggerentscheidungen sind vom Experiment vorgegebene Rahmenbedingungen. Sei nachfolgend $\tau_{\text{L1}} = 6{,}5\,\mu$s für die Zeit zwischen L0-Trigger und L1-Entscheidung und $\tau_{\text{L2}} = 80\,\mu$s für die Zeit zwischen L1- und L2-Entscheidung angenommen. Die Übertragungszeiten ergeben sich aus dem Quotienten der Datengröße eines Ereignisses und der zur Verfügung stehenden Bandbreite. Die Datengröße eines Ereignisses hängt stark von der Anzahl der bei der Kollision erzeugten Teilchen und den Einstellungen des Detektors ab. Eine ausführliche Rechnung zur Abschätzung der erwarteten Datengrößen ist in [18] zu finden. Die erwarteten Werte für Datengrößen und Bearbeitungszeiten für ein im Mittel erwartetes und ein zentrales Pb-Pb-Ereignis sind in Tabelle 5.1 gegeben.[2] Die Auslesezeit der FEE ist auf Grund der für die Triggeranwendung benötigten, hohen Eingangsbandbreite der GTU in der selben Größenordnung wie Wartezeiten auf die Triggerentscheidungen. Die Bandbreitenreduzierung um etwa den Faktor 60 auf der Strecke GTU-DAQ spiegelt sich in der entsprechend längeren Übertragungszeit beim Auslesen der GTU wider.

[2] Abweichend von den in [18] gegebenen Parametern ist auf Grund bisheriger Erfahrungen im p-p-Betrieb die Anzahl aktiver Pads pro Teilchen mit 5,6 angenommen.

Typ des Ereignisses		typisch	zentral
Multiplizität	$\frac{\mathrm{d}N_{\mathrm{ch}}}{\mathrm{d}\eta}$	700	2000
Maximale Ereignisgröße pro Halbkammer	$S_{\mathrm{HC}}^{\mathrm{max}}$ / kByte	4,1	8,6
Mittlere Ereignisgröße eines Supermoduls	$S_{\mathrm{SM}}^{\mathrm{max}}$ / kByte	137	362
Maximale Auslesezeit der FEE	$\tau_{\mathrm{ro,GTU}}$ / μs	20,2	41,2
Mittlere Auslesezeit der GTU	$\tau_{\mathrm{ro,GTU}}$ / μs	700	1851
Maximale Anzahl speicherbarer Ereignisse	$n_{\mathrm{buf}}^{\mathrm{max}}$	82	39
Maximale Ausleserate	$R_{\mathrm{out}}^{\mathrm{max}}$ / Hz	1429	540

Tabelle 5.1: Ereignisgrößen und resultierende Bearbeitungszeiten der Auslesekette des TRD

Multiplizität	$\frac{\mathrm{d}N_{\mathrm{ch}}}{\mathrm{d}\eta}$	700	2000
Benötigte Eingangsrate	R / Hz	6285	8845
L1-Totzeit	D_{L1}	3,3 %	3,3 %
FEE-Totzeit	$D_{\mathrm{ro,\ FEE}}$	0,8 %	1,7 %
L2-Totzeit	D_{L2}	2,4 %	1,6 %
GTU-Totzeit	$D_{\mathrm{ro,\ GTU}}$	14,0 %	37,0 %
Gesamt-Totzeit	D_{TRD}	20,4 %	43,5 %

Tabelle 5.2: Totzeit des TRD ohne MEB. Die geringe Bandbreite zwischen GTU und DAQ führt zu einer großen Totzeit bedingt durch die letzte Auslesestufe.

Die resultierenden Totzeiten beim Betrieb der Auslesekette ohne MEB können mit Hilfe von Gleichung 5.1 abgeschätzt werden. Dabei müssen die benötigten Ausgangsraten der einzelnen Stufen um die Reduktionsfaktoren des Triggersystems korrigiert werden. Weiterhin ist zu beachten, dass sich Auslesezeit der FEE und L2-Entscheidungszeit überschneiden. Bei der L2-Entscheidung muss daher nur die verbleibende Zeit beachtet werden. Die Ergebnisse für ein erwartetes L0/L1-Verhältnis von 8 %, ein L1/L2-Verhältnis von 50 % und eine Ausgangsrate von 200 Hz sind in Tabelle 5.2 angegeben. Die dominierende Totzeit wird durch die Übertragung eines Ereignisses an die DAQ hervorgerufen. Die durch die vorherigen Stufen verursachten Totzeiten spielen eine untergeordnete Rolle.

Im Folgenden wird untersucht, wie sich die Totzeit des TRD durch MEB verringern lässt. In der FEE stehen keine Speicher für mehrere Ereignisse zur Verfügung. Die Totzeit der ersten beiden Stufen kann daher nicht durch MEB reduziert werden. Wie bereits ausgeführt, spielen diese jedoch eine untergeordnete Rolle. Für die letzten beiden Stufen ist eine effektive Reduktion der Totzeit durch MEB in der GTU möglich. Da $D_{\mathrm{ro,\ GTU}}$ die Totzeit des TRD dominiert, wird der Effekt von MEB zunächst isoliert für die letzte Auslesestufe analysiert.[3] Die Totzeit für verschiedene Puffertiefen kann mit Hilfe von Gleichung 5.4 abgeschätzt werden. Die resultierenden Werte sind, in Abhängigkeit von der primären Ereignisrate, in Abbildung 5.2 aufgetragen. Für den Idealfall unendlicher Puffertiefe wächst die Totzeit erst, wenn die Eingangsrate der letzten Stufe die maximale Ausleserate über-

[3] Die Reduktion von D_{L2} spielt im Experimentbetrieb eine untergeordneten Rolle, da der TRD im Normalfall zusammen mit der TPC ausgelesen wird. Deren intrinsische Totzeit überdeckt die gesamte Zeit bis zur L2-Entscheidung.

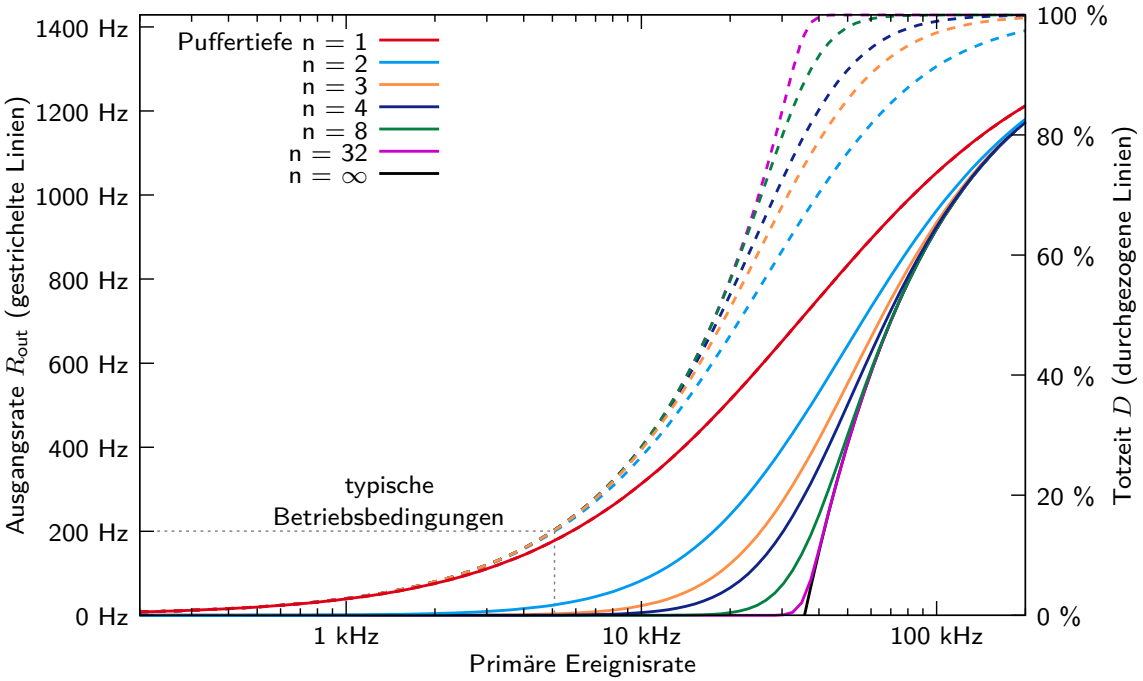


Abbildung 5.2: Totzeit und Ausleserate der letzten Auslesestufe des TRD für verschiedene Puffertiefen. Angenommen sind Ereignisse mit einer mittleren Multiplizität von $\frac{\mathrm{d}N_{\mathrm{ch}}}{\mathrm{d}\eta} = 700$.

schreitet. Korrigiert um die Reduktionsfaktoren der Triggerstufen, entspricht dies einer primären Ereignisrate von rund 36 kHz. Bei einer typischen Ausleserate von 200 Hz und einer Puffertiefe von 3, verschwindet die Totzeit durch die Übertragung an die DAQ beinahe vollständig. Die gleichen Überlegungen gelten für D_{L2}. Für Puffertiefen von $n \geq 3$ verschwindet die Totzeit der letzten beiden Auslesestufen beinahe vollständig. Die Gesamttotzeit des TRD wird dann durch die Totzeit der ersten beiden Auslesestufen bestimmt. Nach Tabelle 5.2 erwartet man etwa 4 %.

Der genaue Verlauf der Gesamttotzeit des TRD kann mit Hilfe einer ereignisbasierten Monte-Carlo-Simulation ermittelt werden. Eine solche Simulation wurde bereits in [18] durchgeführt und für diese Arbeit angepasst. In der Simulation werden folgende Annahmen gemacht:

- Für das Auftreten primärer Ereignisse ist eine Gleichverteilung der Ereignisse angenommen. Die Zeit zwischen zwei Ereignissen ist daher exponentialverteilt.
- Jedem Pre-Trigger folgt ein L0-Trigger, die L0-Wahrscheinlichkeit ist eins.
- Die Wahrscheinlichkeit eines L1-accept ist für alle Ereignisse gleich und keinen Schwankungen unterworfen. Gleiches gilt für die Wahrscheinlichkeit eines L2-accept.
- Die Wartezeiten auf die Triggerentscheidungen sind konstant und für jedes Ereignis gleich. Mögliche Verzögerungen bei der Übertragung der L2-Trigger sind nicht

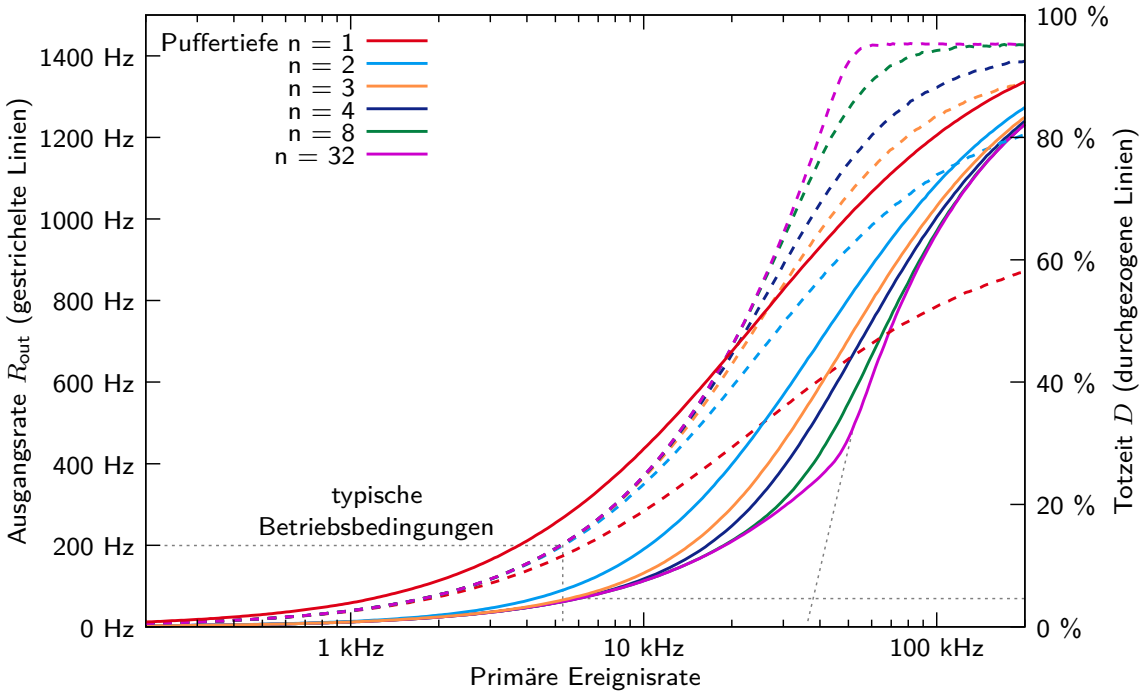


Abbildung 5.3: Gesamttotzeit und Ausleserate des TRD für verschieden Puffertiefen, simuliert für typische Ereignisse mit einer mittleren Multiplizität von $\frac{\mathrm{d}N_{\mathrm{ch}}}{\mathrm{d}\eta} = 700$. Eingezeichnet sind die erwartete Betriebsbedingungen.

berücksichtigt.

- Die Übertragungszeiten sind exponentiell um die mittlere Übertragungszeit verteilt.
- Die Flusskontrolle des DDL bleibt inaktiv, eine Bandbreite von 200 MB/s zur DAQ steht zur Verfügung.

Die Ergebnisse der Simulation sind in Abbildung 5.3 dargestellt. Für eine Ausleserate von 200 Hz liegt die Gesamttotzeit ab einer Puffertiefe von $n = 3$ bei rund 4 %. Die L0-Rate liegt bei etwa 5 kHz. Die Ergebnisse entsprechen den Erwartungen der analytischen Betrachtung. Der Verlauf der Kurven für eine Puffertiefe größer Eins lässt sich in zwei Bereiche zerlegen. Bei niedrigen Ereignisraten dominiert die durch die Puffertiefe nicht beeinflusste Totzeit der ersten beiden Auslesestufen in der FEE. Die Totzeit der beiden Auslesestufen in der GTU verschwindet auf Grund des MEB. Erreicht die Ereignisrate die maximale Ausleserate, kann Totzeit auch durch MEB nicht mehr vollständig eliminiert werden. Bei hohen Eingangsraten steigt die Totzeit deshalb steil an, wobei dieser Effekt bei kleineren Puffertiefen früher einsetzt. Insbesondere für eine Puffertiefe von 32 ist der Kurvenverlauf aus Abbildung 5.2 bei hohen Raten deutlich zu erkennen. Extrapoliert man die Kurve zur x-Achse erhält man analog zu Abbildung 5.2 eine Eingangsrate von etwa 36 kHz.

Die ausgeführten Überlegungen zeigen, dass die Totzeit des TRD durch MEB in der GTU

entscheidend reduziert werden kann. Für eine mittlere Multiplizität von $\frac{\mathrm{d}N_{\mathrm{ch}}}{\mathrm{d}\eta} = 700$ kann die Totzeit ab einer Puffertiefe von $n = 3$ auf ein Minimum reduziert werden. Sie liegt bei einer Ausleserate von 200 Hz unter 4 %. Verglichen mit einer Totzeit von 20 % ohne MEB stellt dies eine erhebliche Reduktion dar. Die angegebenen Werte basieren auf aktuellen Schätzungen für Parameter wie die Ereignisgröße und Annahmen über die zur Verfügung stehenden Bandbreiten. Steht etwa durch die Flusskontrolle des DDL nicht die volle Bandbreite zur Verfügung, fällt die benötigte Puffertiefe entsprechend höher aus. Die benötigte Puffertiefe für abweichende Parameter kann mit Hilfe der gegebenen Gleichungen leicht abgeschätzt werden. Die GTU verfügt auch bei ausschließlich zentralen Ereignissen über Speicherplatz für bis zu 39 Ereignisse. Um von den Annahmen abweichende Parameter bestmöglich ausgleichen zu können, ist es vorteilhaft bei der Implementierung von MEB den gesamten zur Verfügung stehenden Speicher zu nutzen. Bei der Implementierung der Speicherverwaltung in den TMUs ist dieser Aspekt bereits beachtet. Die Implementierung einer MEB-Segmentsteuerung für die SMU wird im nächsten Kapitel ausführlich vorgestellt.

6 Die MEB-Segmentsteuerung

Zur Realisierung einer multi-event-buffering-fähigen Datenauslese des TRD mit Puffern in der GTU werden eine MEB-fähige TMU und SMU benötigt. Seitens der SMU liegt besonderes Augenmerk auf der Segmentsteuerung. Im Folgenden wird zunächst ausgeführt, welche Anforderungen eine MEB-fähige Segmentsteuerung erfüllen muss. Danach wird ein Konzept vorgestellt, mit dem sich eine solche Segmentsteuerung realisieren lässt. Im weiteren Teil des Kapitels folgt eine Übersicht über die Implementierung sowie eine Zusammenfassung.

6.1 Anforderungen an eine MEB-Segmentsteuerung

Hauptaufgabe der Segmentsteuerung ist es, die eintreffenden Triggersignale des CTP zu verarbeiten und entsprechende Steuersignale zu erzeugen. Dazu gehören vor allem Steuersignale, die in Verbindung mit der Datenauslese und den TMU-Ereignispuffern benötigt werden. Eine weitere Aufgabe ist die Erzeugung des segmentweiten TRD-Busy-Signals, das in der TGU konzentriert und an den CTP weitergeleitet wird. Einige Teileinheiten von TMU und SMU benötigen zusätzlich Signale zu definierten Zeitpunkten inmitten einer Triggersequenz, die ebenfalls von der Segmentsteuerung erzeugt werden. Zur Unterstützung von MEB muss die Segmentsteuerung dabei insbesondere in der Lage sein, verschachtelte Triggersequenzen zu verarbeiten. Das bedeutet, dass bereits neue L0- und L1-Trigger eintreffen dürfen, während die zugehörigen L2-Entscheidungen noch ausstehen oder vorangegangene Ereignisse noch nicht vollständig ausgelesen wurden. Dabei ist darauf zu achten, dass sich die Triggersequenzen nicht gegenseitig beeinflussen. Die Verarbeitung der L0-L1-Sequenzen muss zwingend in Echtzeit ablaufen, da Aktionen von FEE und GTU nur anhand der eintreffenden Triggersignale synchronisiert werden. Die Anzahl parallel verarbeitbarer Triggersequenzen entspricht der maximalen Anzahl unterstützter Ereignispuffer. Für die Segmentsteuerung sind des Weiteren Eigenschaften wie Fehlererkennung und Konfigurationsmöglichkeiten von besonderem Interesse.

Fehlererkennung Beim Betrieb eines so komplexen Systems wie ALICE kommt es unausweichlich zu Fehlern. Die Fehlerquellen sind dabei sehr unterschiedlicher Natur. Fehler können beispielsweise durch fehlerhaftes Verhalten einzelner Komponenten, aber auch durch Übertragungsfehler von Steuersignalen oder fehlerhafte Konfigurationen verursacht werden. Von entscheidender Bedeutung ist es, möglichst viele Fehler erkennen und die Fehlerquelle auch im Nachhinein identifizieren zu können. Nur so ist es möglich, die Fehler zuverlässig zu beheben.

Fehlerquellen für die Segmentsteuerung sind vor allem ausbleibende oder unerwartete Triggersignale. Vorrangig muss sichergestellt sein, dass die an die DAQ gesendeten Ereignisse konsistent sind. Fehler dürfen nicht dazu führen, dass Daten verschiedener Ereignisse gemischt werden. Des weiteren dürfen Fehler nicht zu unkontrollierbaren Zuständen der Segmentsteuerung führen. Es ist sonst nicht möglich, den Fehler sinnvoll zu analysieren. Die Segmentsteuerung muss daher neben der eigentlichen Steuerung der Datenauslese mögliche Fehlerbedingungen erkennen, sicher behandeln und signalisieren können.

Konfigurierbarkeit Viele Betriebsparameter des Experimentes sind nicht fest, sondern können in gewissen Grenzen verändert werden. Das gilt etwa für den Zeitpunkt der L1-Entscheidung, aber auch für etliche andere Parameter. Des weiteren werden zusätzliche Segmente der GTU in Testaufbauten eingesetzt.[1] Betriebsparameter und Struktur der Testaufbauten unterschieden sich stellenweise deutlich von denen im Experiment. Es ist daher unerlässlich, die Segmentsteuerung auf unterschiedliche Gegebenheiten anpassen zu können. Da die Logik in FPGAs realisiert ist, bestünde die Möglichkeit, für jede Situation angepasste HDL-Designs zu erstellen und die FPGAs entsprechend zu konfigurieren. Dieser Ansatz hat jedoch mehrere Nachteile. Die Pflege mehrerer Designs ist aufwendig und fehleranfällig. Jedes einzelne Design muss bei Veränderungen zeitaufwendig vollständig neu getestet werden. Während der Experimentlaufzeit von geplanten 15 Jahren müssen Experten zur Verfügung stehen, die die Designs anpassen können. Die *Toolchain* zur Erzeugung der FPGA-Konfigurationen muss über die gesamte Laufzeit gepflegt werden. Um das zu vermeiden, sollte das HDL-Design der Segmentsteuerung umfangreiche Konfigurationsmöglichkeiten bieten. Dabei müssen auch Spezialfälle, die im eigentlichen Experimentbetrieb nicht vorkommen, bedacht werden, um den Betrieb in anderen Umgebungen oder den Testbetrieb im Experiment zu ermöglichen.

6.2 Funktionsprinzip

Die Herausforderung bei der Realisierung der MEB-Segmentsteuerung liegt in den vielfältigen Kombinationsmöglichkeiten der Abfolge einzelner Trigger in verschachtelten Triggersequenzen und der dabei möglichen Fehlerbedingungen. Für eine kontinuierliche Verarbeitung von Triggersequenzen ist es unerlässlich, dass die Steuerung zyklisch arbeitet. Dabei müssen aber zugleich alle Kombinationsmöglichkeiten abgedeckt sein. Mit einer einzigen Kontrollmaschine ist dies kaum zu realisieren.

Ein besser geeigneter Ansatz ist die Aufspaltung der Bearbeitung. Eine Möglichkeit besteht darin, jede einzelne Triggersequenz mit einer *Single-Event*-Kontrollmaschine zu verarbeiten. Für jede parallel gestartete Sequenz wird eine separate Instanz der Single-Event-Kontrollmaschine verwendet. Ein Koordinator verwaltet diese Instanzen und verteilt die eintreffenden Trigger auf die entsprechenden Kontrollmaschinen. Auch dieser Ansatz hat jedoch einige Nachteile. Der Koordinator ist aufwendig zu implementieren. Vor der Verteilung müssen fehlerhafte Trigger erkannt werden, um sie nicht falsch zuzuordnen. Jede

[1] Momentan existieren Aufbauten von einzelnen GTU-Segmenten am CERN und am IKP in Münster.

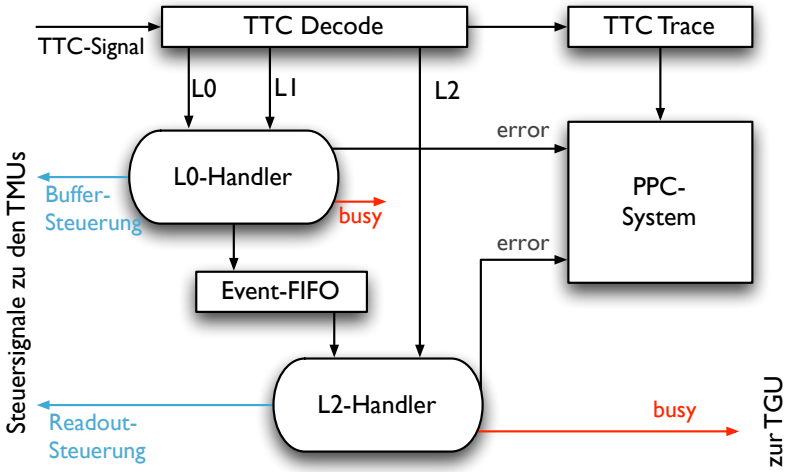


Abbildung 6.1: Strukturierung der MEB-Segmentsteuerung. Die decodierten Triggersignale werden von zwei getrennten Trigger-Handlern verarbeitet. Für den Fehlerfall hat das PPC-System Zugriff auf die Triggerdaten im TTC-Trace.

benötigte Instanz der Kontrollmaschine muss parallel zu den anderen in Hardware vorhanden sein. Dadurch steigt der Ressourcenverbrauch mit der Anzahl verwaltbarer Puffer stark an.

Eine genaue Analyse der Eigenschaften des Triggersystems eröffnet eine weitere Möglichkeit der Aufteilung. Dabei sei auf zwei Merkmale des Triggersystems hingewiesen:

- Alle Trigger treffen im Regelfall in chronologischer Reihenfolge bei der GTU ein, verschiedene Triggersequenzen dürfen sich nicht überholen.
- Zwischen L0- und L1-Trigger sind keine weiteren L0-Trigger (und somit auch keine L1-Trigger) erlaubt. L0 und L1-Trigger bilden eine feste, nicht unterbrechbare Teilsequenz.

Das bedeutet, dass lediglich Triggersequenzen wie $L0_0 - L1_0 - L0_1 - L2_0 - L1_1 - L2_1$ erlaubt sind, während $L0_0 - L0_1 - L1_0 - L2_0 - L1_1 - L2_1$ eine ungültige Sequenz wäre. Das Eintreffen der L2-Trigger ist zeitlich nicht exakt festgelegt und kann auch innerhalb einer der nachfolgenden L0-L1-Sequenzen liegen. Diese Eigenschaften ermöglichen es, die Verarbeitung der L2-Trigger von der der L0-L1-Sequenzen zu trennen. Im Folgenden ist erläutert, wie sich mit dieser Aufteilung eine effiziente, ressourcensparende und gegen Triggerfehler tolerante MEB-Segmentsteuerung verwirklichen lässt.

6.2.1 Verarbeitung der Triggersequenzen

Ein schematischer Überblick über die Gesamtstruktur ist in Abbildung 6.1 dargestellt. Die Verarbeitung der Trigger ist auf zwei Steuereinheiten aufgeteilt, den *L0-Handler* und den *L2-Handler*. Der L0-Handler verarbeitet alle L0- und L1-Trigger. Der L2-Handler

verarbeitet alle eintreffenden L2-Trigger. Jeder Handler arbeitet sequenziell die für ihn bestimmten Trigger ab. Die Auftrennung der Verarbeitung von L0/L1- und L2-Triggern entspricht einer Trennung von Schreib- und Lesesteuerung des TMU-Event-Bufferings.

In den Aufgabenbereich des L0-Handlers fällt die Kontrolle über das Aufzeichnen der FEE-Daten und Speichern in den TMU-Ereignispuffern nach einem L1-accept. Er generiert das Busy-Signal während die FEE keine Trigger entgegennehmen kann und steuert die Event Shaper der TMUs. Der L0-Handler übernimmt außerdem die Interpretation eines zum L1-Zeitpunkt ausbleibenden L1-Triggersignals als L1-reject.

Für die Abarbeitung der L2-Trigger benötigt der L2-Handler neben den Triggern Zusatzinformationen über die in den TMUs gepufferten Ereignisse, zum Beispiel den Wert des Event-ID-Zählers beim Eintreffen eines L0-Triggers, der zur Ereignisidentifikation genutzt wird. Da die Informationen in der L0/L1-Phase generiert werden, müssen sie, genau wie die Ereignisse selbst, zwischengespeichert werden. Dafür werden sie vom L0-Handler im *Event-FIFO* gespeichert, aus dem sie vom L2-Handler ausgelesen werden können.

Der L2-Handler arbeitet parallel zum L0-Handler und unabhängig von dessen Zustand. Bei der Verarbeitung der L2-Entscheidungen initiiert der L2-Handler die Übertragung von Ereignisdaten aus den TMU-Ereignispuffern zur DAQ nach einem L2-accept bzw. das Verwerfen eines Ereignisses nach einem L2-reject. Die erzeugten Steuersignale werden von den Readout Units der TMUs und der Auslesekontrolle der SMU verarbeitet.

Software-Trigger Zusätzlich zu physikalisch motivierten Trigger-Sequenzen kann der CTP sogenannte Software-Trigger auslösen. Gegenwärtig sind die bei jedem Start und Ende der Datennahme an alle Detektoren gesendeten Start-of-Data (SoD)- und End-of-Data (EoD)-Trigger-Sequenzen relevant. Dabei handelt es sich jeweils um eine volle Triggersequenz. Diese werden zur Synchronisation der LDCs genutzt und bieten den Detektoren die Möglichkeit, anstelle von Ereignisdaten arbiträre Daten wie Konfiguration und Statistikinformationen zu übertragen. Die FEE des TRD macht momentan keinen Gebrauch von dieser Möglichkeit und ignoriert Software-Trigger; es werden also keine Daten an die GTU gesendet. Um möglichst flexibel auf eine Änderung dieses Verhaltens reagieren zu können, wird das Ereignis dennoch wie jedes andere verarbeitet. Treffen keine Daten ein, generieren die Link-Monitore Endmarker, und ein leeres Ereignis wird gepuffert. Die Information, dass es sich um einen Software-Trigger handelt, wird mit im Event-FIFO gespeichert. Es besteht daher die Möglichkeit, beim Auslesen des Ereignisses Informationen wie GTU-Konfiguration vom L2-Handler in den Datenstrom einzufügen zu lassen.

Die TMU-seitige Implementierung der Speicher in Ringpuffern erlaubt eine dynamische Anzahl von Puffern und eine effektive Speichernutzung bei weit gestreuten Ereignisgrößen. Die Implementierung verlangt jedoch, dass die Ereignisse nur sequenziell ausgelesen und verworfen werden können. Das bedeutet, dass auch Ereignisse, für die bereits ein L2r eingetroffen ist, erst dann verworfen werden können, wenn alle älteren Ereignisse abgearbeitet sind. Die Puffer werden also länger als benötigt belegt, wodurch effektiv weniger Puffer zur Verfügung stehen. Aufgrund der stark unterschiedlichen Ereignisgrößen im p-p- und Pb-Pb-Betrieb und einem erwarteten L2r-Anteil von etwa 50 % überwiegt der Vorteil

der dynamischen Puffergröße gegenüber einer Implementierung mit statischer Puffergröße aber beliebigem Speicherzugriff. Die vorgestellte Segmentsteuerung ist darauf ausgelegt, analog zur Implementierung der TMUs eine dynamische Anzahl an Puffern sequenziell zu verwalten. Um eine Implementierung mit beliebigem Speicherzugriff zu unterstützen besteht die Möglichkeit den L2-Handler aufzuteilen, damit L2a-Trigger und L2r-Trigger unabhängig verarbeitet werden können. Im Folgenden wird dies jedoch außer Acht gelassen und lediglich eine sequenziell arbeitende Pufferverwaltung vorgestellt.

Durch die Aufspaltung in zwei Kontrollmaschinen können die eintreffenden Trigger kontinuierlich und zyklisch verarbeitet werden. Die Anzahl der parallel verarbeitbaren Triggersequenzen hängt nur noch von wenigen Elementen ab und beeinflusst den Ressourcenverbrauch der Gesamteinheit nur minimal.

6.2.2 Umgang mit Triggerfehlern

Die Vielzahl der möglichen Fehlerbedingungen macht die Fehleranalyse und Wiederherstellung des korrekten Betriebszustandes nach einem Fehler komplex. Dies sei im folgenden einfachen Beispiel verdeutlicht:

Trifft ein L1-Trigger ein, ohne dass ihm ein L0-Trigger vorausgegangen ist, so liegt ein Fehler vor. Es ist jedoch nicht klar, ob der L1-Trigger falsch ausgelöst wurde oder der zugehörige L0-Trigger gefehlt hat. Dies kann erst durch das Eintreffen eines L2-Triggers festgestellt werden. Also muss auch eine mit einem L1-Trigger beginnende Sequenz bearbeitet werden bis feststeht, welcher Fehler vorliegt.

Für andere Fehlerfälle gilt Ähnliches. Daher muss die Segmentsteuerung neben korrekten Sequenzen auch inkorrekte Sequenzen verarbeiten, um Fehler identifizieren zu können. Eine entsprechende Implementierung in Hardware ist aufwendig und ressourcenintensiv. Im selten auftretenden Fehlerfall können allerdings die harten Anforderungen an Totzeit und Verarbeitungsdauer außer Acht gelassen werden. Für die Abwicklung von Fehlerfällen kann daher ein Hardware/Software-Co-Design gewählt werden. Die Verarbeitung fehlerfreier Triggersequenzen erfolgt in Echtzeit von den Trigger-Handlern. Beide Trigger-Handler verifizieren die eintreffenden Trigger, bevor sie sie verarbeiten. Dadurch ist sichergestellt, dass keine falschen Steuersignale ausgelöst werden. Die Verifikation des L0-Handlers sorgt dafür, dass nur Ereignisse mit korrekten L0- und L1-Triggern gepuffert werden. Das Event-FIFO enthält daher auch nur Informationen zu korrekten Ereignissen. Im Fehlerfall stoppt der entsprechende Trigger-Handler die Verarbeitung von Triggern und setzt das Busy-Signal des Detektors. Dadurch wird verhindert, dass der CTP weitere Triggersequenzen startet. Tritt der Fehler im L0-Handler auf, arbeitet der L2-Handler zunächst weiter, bis alle Ereignisse im Event-FIFO abgearbeitet sind und verbleibt dann automatisch im Grundzustand. Dies kann abhängig vom Füllstand der Ereignispuffer einige Zeit in Anspruch nehmen.

Tritt der Fehler im L2-Handler auf, erhält der L0-Handler aufgrund des Busy-Signals keine weiteren L0-Trigger. Er verarbeitet somit gegebenenfalls noch das aktuelle Ereignis und verbleibt dann ebenfalls im Grundzustand. Ist das System zur Ruhe gekommen, können mit

Hilfe des PPC-Systems in Software realisierte Analyse- und Wiederherstellungsroutinen ausgeführt werden, um den gewünschten Systemzustand wiederherzustellen.

Für die Fehleranalyse stehen dem PPC-System die im TTC-Trace gespeicherten Informationen über die empfangenen Trigger und die Fehlerinformation der Trigger-Handler zur Verfügung. Anhand dieser Informationen kann in Software entschieden werden, welcher Fehler vorliegt. Ist die Fehleranalyse abgeschlossen, wird ein CDH mit den entsprechenden Fehler-Flags generiert und an die DAQ gesendet. Danach können die Trigger-Handler in den Grundzustand zurückkehren und ihre normale Arbeit wieder aufnehmen.

Diese Aufteilung der Fehlerbehandlung in eine Fehlererkennung in Hardware und eine Fehleranalyse in Software ermöglicht es, mögliche Fehlerfälle ohne den Ressourcenverbrauch und Aufwand, den eine Implementierung in Hardware mit sich bringt, behandeln zu können.

6.3 Implementierung

Eine Gesamtübersicht der MEB-Segmentsteuerung ist in Abbildung 6.2 gegeben. Zu sehen sind die Zustandsmaschinen der beiden Trigger-Handler und zusätzliche Logik, die für den Betrieb benötigt wird. Die decodierten TTC-Signale werden vom TTC-Receiver bereitgestellt. Die Aufgabe und Implementierung der einzelnen Komponenten sind im Folgenden kurz vorgestellt. Eine genaue Beschreibung der Funktionsweise der Trigger-Handler und ihrer Implementierung folgt in den nächste Abschnitten.

Event-ID-Zähler Der Event-ID-Zähler bildet die absolute Zeitbasis für die Ereignisidentifikation. Er ist mit zwei Zählern implementiert: einem 12-Bit-Zähler, dem so genannten Bunch Counter (BC) und einem 24-Bit-Zähler, dem Orbit Counter (OC). Der BC wird mit dem LHC-Takt inkrementiert und zu Beginn jedes Orbits auf Null zurückgesetzt. Damit weist er jeder der 3564 möglichen Bunchpositionen einen Wert zu. Der OC wird mit jedem LHC-Orbit, also Reset des BC inkrementiert. In Kombination ergibt sich eine eindeutige Ereignisidentifikation für einen Zeitraum von etwa 24 Minuten.

Der Wert des Event-ID-Zählers wird für zwei Vergleiche mit der, in der L2-Nachricht enthaltenen, Event-ID des CTP verwendet (vgl. Abschnitt 6.6.1). Zur Synchronisation des lokalen Event-ID-Zählers mit dem Zähler des CTP sendet das TTC-System zu Beginn jedes LHC-Orbits einen BC-Reset. Setzt man den BC zu diesem Zeitpunkt auf Null zurück, kommt es aufgrund der Laufzeit- und Decodierungsverzögerung des Signals zu einer festen Differenz zwischen lokalem Zähler und Zähler des CTP. Da 3464 keiner Zweierpotenz entspricht, ist der Vergleich auf eine Differenz aufwendig. Der BC ist daher als umlaufender Zähler von 0 bis 3563 mit einem konfigurierbaren Reset-Wert implementiert. Hierdurch kann der Absolutwert des BC angeglichen werden. Die Vergleiche reduzieren sich dann auf einfache Prüfung auf Gleichheit.

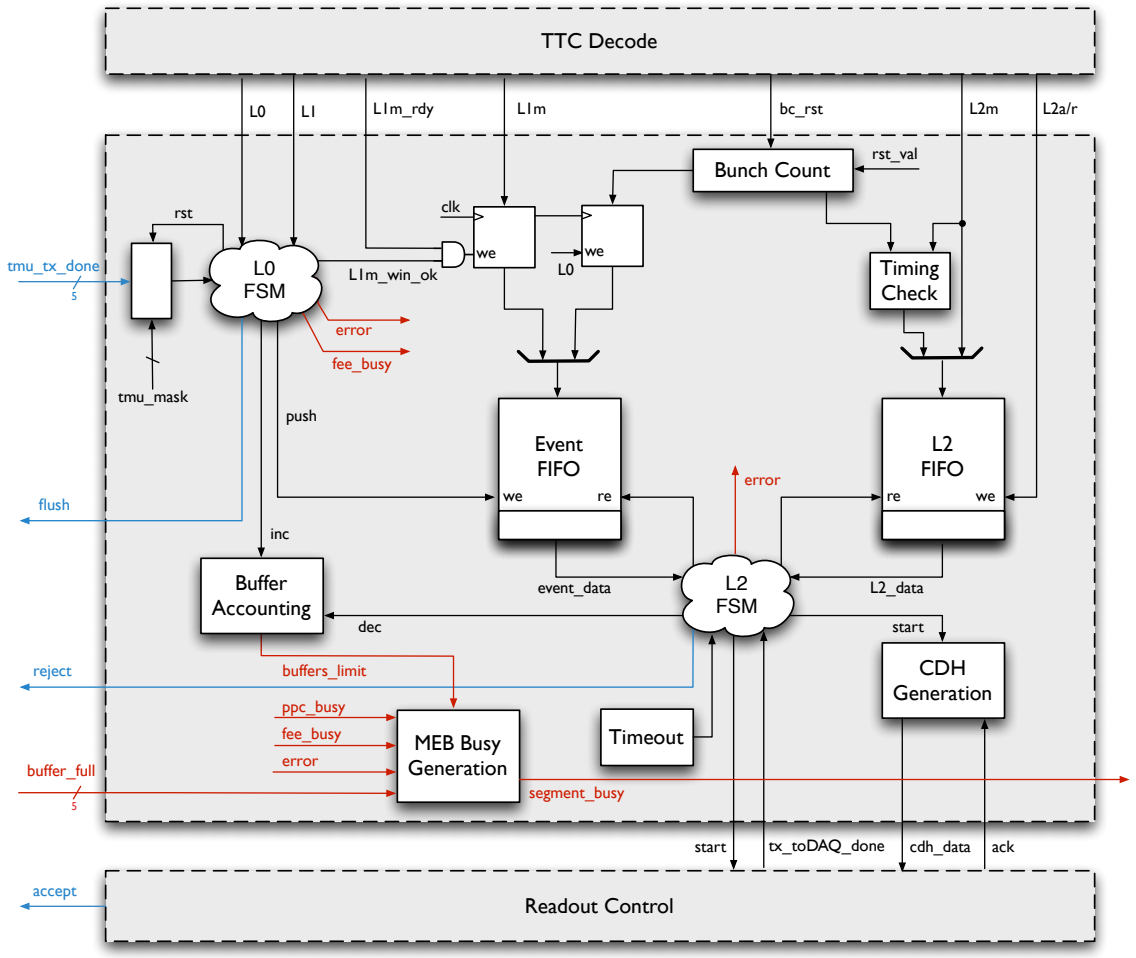


Abbildung 6.2: Gesamtübersicht der MEB-Segmentsteuerung. Neben den Zustandsmaschinen der Trigger-Handler ist zusätzliche Logik zu sehen, die Daten bis zu ihrer Verwendung zwischenspeichert oder benötigte Hilfssignale erzeugt.

Event-FIFO und L2-FIFO Die vom L2-Handler benötigten Informationen über ein gepuffertes Ereignis werden im Event-FIFO gespeichert. Die Struktur eines Eintrags im Event-FIFO ist in Abbildung 6.3 dargestellt. Die unteren 36 Bit enthalten die sogenannte *local Event-ID*. Sie entspricht dem zum Zeitpunkt des L0-Triggers erfassten Wert des Event-ID-Zählers. Die oberen Bits enthalten aus der L1-Nachricht kopierte Informationen, die in den CDH eingefügt werden müssen.

Da der L2-Trigger zu einem Ereignis im allgemeinen Fall bereits eintrifft, während noch ein älteres Ereignis vom L2-Handler verarbeitet wird, müssen auch die L2-Trigger zwischengespeichert werden. Dies geschieht im *L2*-FIFO. Die Struktur eines Eintrags im L2-FIFO ist ebenfalls in Abbildung 6.3 dargestellt. Neben den Informationen, die direkt aus der L2 -Nachricht kopiert werden, wird zusätzlich der Typ des Triggers und das Ergebnis der in Abschnitt 6.6.1 erwähnten Zeitüberprüfung gespeichert. Im Falle eines L2r-Triggers sind

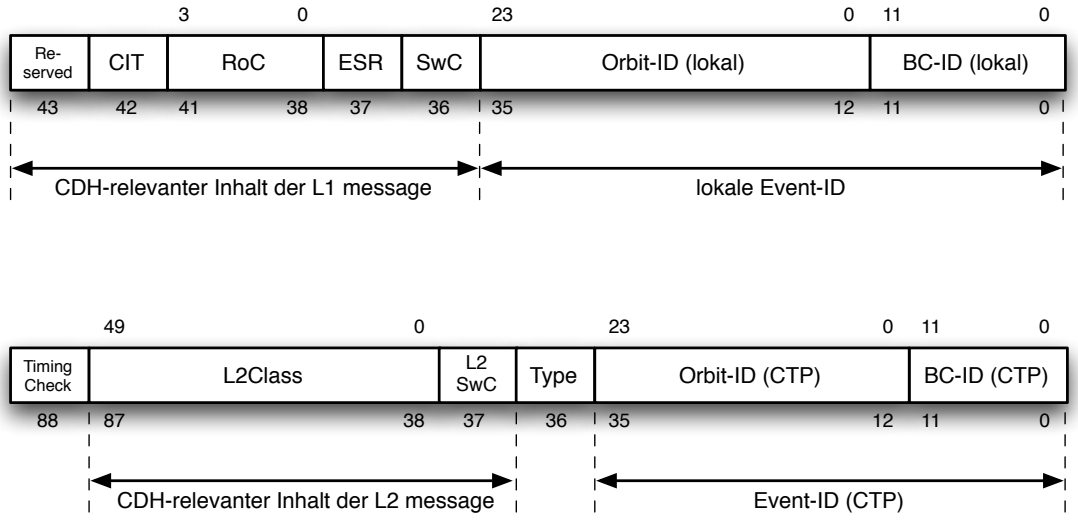


Abbildung 6.3: Struktur eines Eintrags in den FIFOs der Segmentsteuerung. Oben dargestellt das Event-FIFO, unten das L2-FIFO. Die untersten 36 Bit enthalten jeweils die Event-ID des CTP. In den oberen Bits sind Informationen gespeichert, die unter anderem benötigt werden, um den CDH zu erstellen.

alle Felder außer Typ und BC-ID des CTP leer.

Der Speicher der FIFOs ist auf der Basis von BRAMs implementiert. Im Virtex-4 stehen dedizierte 18-kBit-BRAM-Instanzen mit einer maximalen Datenbusbreite von 36 Bit zur Verfügung. Zur Implementierung der FIFOs werden daher zwei bzw. drei BRAMs benötigt. Es sind die ohne zusätzlichen Ressourcenverbrauch erreichbaren Datenbreiten von 72 Bit bzw. 108 Bit gewählt, um potenzielle Erweiterungen einfach durchführen zu können. In der Tiefe stehen damit 512 Einträge zur Verfügung.

Pufferverwaltung Die Pufferverwaltung kontrolliert, wie viele Ereignisse maximal gepuffert werden. Dafür ist ein Zähler implementiert, der vom L0-Handler inkrementiert wird, wenn ein Ereignis gepuffert wurde. Der L2-Handler dekrementiert den Zähler, wenn er ein Ereignis vollständig verarbeitet hat. Überschreitet der Zähler eine konfigurierbare Schwelle wird das Busy-Signal gesetzt, um zu verhindern, dass weitere Triggersequenzen gestartet werden. So kann die maximale Puffertiefe an die Eigenschaften anderer Einheiten angepasst werden, ohne dass all diese Komponenten ein eigenes Busy-Signal erzeugen müssen. Ein Beispiel hierfür ist die benötigte Zwischenspeicherung von Kontrollinformationen der Triggerprozessor-Einheiten, die beim Auslesen von Ereignissen zu den Daten-Headern hinzugefügt werden können. Außerdem ermöglicht der Zähler eine ständige Überprüfung der Pufferauslastung. Steht bereits vor Erreichen der eingestellten Schwelle in einer der TMUs nicht mehr genügend Speicherplatz für ein weiteres Ereignis zur Verfügung, wird dies von der TMU signalisiert und ebenfalls das Busy-Signal gesetzt.

Datenheader Jedes Ereignis enthält mehrere Datenheader, welche Zusatzinformationen über das Ereignis enthalten. Von der SMU müssen zwei Header generiert werden. Der *SMU-Header*, der auf SMU-Ebene anfallende Informationen wie die TMU-Maske enthält, und der *CDH*, der von der DAQ festgelegte Informationen enthält und zu Beginn jedes Ereignisses gesendet werden muss. Das genaue Format der Header ist in [19] und [20] dokumentiert. Die für die Header benötigten Daten laufen in einer separaten Einheit zusammen. Sind alle Header-Worte vollständig, können sie von der Auslesesteuerung in den Datenstrom eingefügt werden.

6.4 Verwendete Steuersignale

Die von der Segmentsteuerung generierten Steuersignale müssen so gestaltet sein, dass alle für den Betrieb notwendigen Vorgänge eindeutig signalisiert werden können. Dabei ist es aufgrund der beschränkten Anzahl von Signalen, die in SMU-TMU-Richtung zur Verfügung stehen, erstrebenswert, möglichst wenige Signale zu verwenden. Des weiteren ist bei der Festlegung der Signale zu beachten, dass die Signale in mehrere Clock-Domains synchronisiert werden müssen. Folgende Vorgänge müssen von der SMU an die TMUs signalisiert werden können:

- Beginn einer Triggersequenz und gültiger Daten auf den FEE-Links
- Endgültiges Puffern eines Ereignisses nach dem Abschluss einer L0-L1-Sequenz
- Verwerfen eines in Bearbeitung befindlichen Ereignisses nach einem L1-reject oder in gewissen Fehlerfällen (*Event Flush*)
- Auslesen des ältesten gepufferten Ereignisses nach einem L2-accept
- Verwerfen des ältesten gepufferten Ereignisses nach einem L2-reject
- Eintreffen eines L1-accept als Zeitbasis für Statistikmessungen

Vorgänge, welche die TMUs signalisieren müssen, sind:

- Das Ende der Übertragung eines Ereignisses von der FEE zur TMU
- Die Nichtbereitschaft, ein neues Ereignis entgegen zu nehmen, etwa weil die Puffer voll sind

Insgesamt werden dazu sieben Steuersignale benötigt, deren Funktion im Folgenden erläutert wird. Um das Zusammenspiel zu veranschaulichen, sind in Abbildung 6.4 drei typische Abläufe des Anfangs einer Triggersequenz dargestellt. Eine steigende Flanke auf `expect_data` signalisiert der TMU den Beginn einer Triggersequenz und das Eintreffen gültiger Daten auf den FEE-Links. Im Falle eines L1-accept (Abb. 6.4 (a)) wird ein Puls auf `l1_received` ausgelöst. Haben die TMUs das Ereignis vollständig von den FEE empfangen, müssen sie dies durch einen Puls auf `tmus_rx_done` signalisieren.[2] Haben alle

[2] Dabei ist in den TMUs durch die Link-Monitore sichergestellt, dass Ereignisse immer als vollständig angesehen werden können.

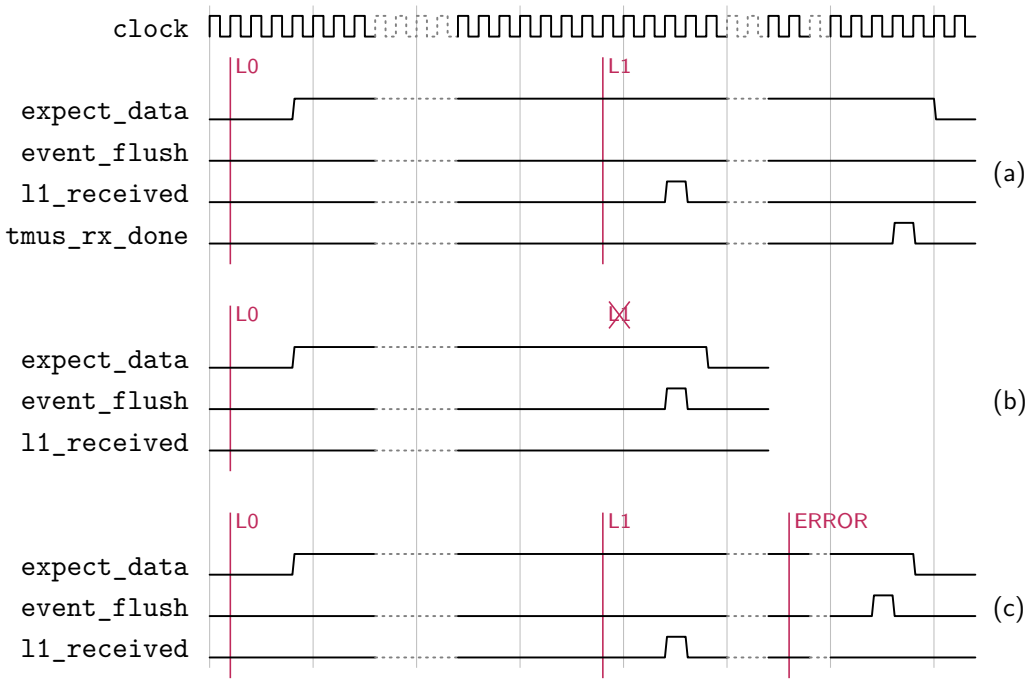


Abbildung 6.4: Kontrollsignale des TMU-SMU-Interface am Beispiele typischer Triggersequenzen. Abbildung (a) und (b) zeigen eine Triggersequenz mit positiver bzw. negativer L1-Entscheidung. Abbildung (c) zeigt das Beispiel einer fehlerhaften Triggersequenz. Tritt ein Fehler auf, bleibt `expect_data` so lange gesetzt, bis die Fehleranalyse abgeschlossen ist und das Ereignis, wenn nötig, verworfen wurde.

aktiven TMUs einen Puls gesendet und ist kein Fehler aufgetreten, signalisiert die SMU durch eine fallende Flanke auf `expect_data` den Event-Shapern der TMUs, das Ereignis endgültig zu puffern, das heißt die Adressinformationen über das Ereignis in das Adress-FIFO zu schreiben und in den Grundzustand zurückzukehren. Das Setzen von `tmu_busy` signalisiert den SMUs, dass keine neuen Ereignisse entgegengenommen werden können.

Ein Puls auf `event_flush` signalisiert dem Event Shaper das Verwerfen des in Bearbeitung befindlichen, noch nicht endgültig gepufferten Ereignisses. Dies ist in den Abbildungen 6.4 (b) und (c) beispielhaft illustriert. Um die Steuerlogik der TMUs nicht unnötig komplex zu gestalten, wird auf die Möglichkeit, ein bereits abgeschlossenes Ereignis mittels `event_flush` zu löschen, verzichtet. Daher ist `event_flush` nur gültig und auszuwerten, wenn `expect_data` gesetzt ist.

Eine steigende Flanke auf `event_accept` signalisiert der Readout Unit der TMUs das Auslesen des ältesten gepufferten Ereignisses, eine steigende Flanke auf `event_reject` das Verwerfen. Hat eine TMU das angeforderte Ereignis komplett an die SMU übertragen, signalisiert sie dies der Readout Control Unit der SMU über `tmu_tx_done`.

Da die Signale in mehrere *Clock Domains* synchronisiert werden, ist ihre Pulsbreite und zeitliche Beziehung untereinander beim Eintreffen an den TMUs nicht eindeutig definiert.

Daher wird `event_flush` einen Takt vor der fallenden Flanke von `expect_data` ausgelöst, um mindestens gleichzeitig an den TMUs einzutreffen. Dies führt jedoch zu dem Fall, dass `expect_data` gesetzt ist, obwohl keine gültigen Daten vorliegen. Die TMU darf deshalb nach einem `event_flush` erst wieder Daten entgegen nehmen, wenn eine neue Sequenz durch eine steigende Flanke auf `expect_data` gestartet wird. Um Fehlinterpretationen durch verbreiterte Pulse zu verhindern, dürfen bei `event_flush`, `event_accept` und `event_reject` nur die steigenden Flanken ausgewertet werden.

6.5 Der L0-Handler

Der L0-Handler übernimmt die Verarbeitung aller L0- und L1-Trigger. Dabei müssen die eintreffenden L1-Triggersignale als L1-Trigger interpretiert werden, sofern sie zum definierten L1-Zeitpunkt eintreffen. Die L1-Triggernachricht muss ausgewertet werden, um festzustellen, um welche Art von Triggersequenz es sich handelt. Parallel müssen eventuelle Fehler erkannt und behandelt werden. Abhängig vom Status der aktuellen Triggersequenz muss der TMU signalisiert werden, Daten zu empfangen, zu speichern oder zu verwerfen.

6.5.1 Sequenzverifikation

Für die Verifikation der Triggerereignisse kann von der Tatsache Gebrauch gemacht werden, dass eine Triggersequenz innerhalb der L0-L1-Phase ein festes zeitliches Verhalten hat und keine weiteren L0- oder L1-Trigger erlaubt sind. Zu jedem Zeitpunkt ist daher immer nur ein bestimmtes oder überhaupt kein Triggerereignis erlaubt.

Trifft ein L0-Trigger ein und ist der Detektor aufnahmebereit, muss a priori davon ausgegangen werden, dass dieser korrekt ist und die Bearbeitung einer Sequenz beginnt. Wird während der Abarbeitung ein anderes als das erwartete Triggerereignis registriert, liegt ein Fehler vor. Mit dieser Methode können zuverlässig alle Fehler, die auf falsch gesendete Trigger und Triggernachrichten zurückgehen, erkannt werden.[3] Wird ein Fehler erkannt, setzt der L0-Handler das Fehlersignal und stoppt die Triggerverarbeitung. Er bleibt in einem Wartezustand, bis das PPC-System ein Signal zum Weiterarbeiten gibt.

Als Zeitbasis für die Interpretation des L1-Triggersignals und für die Überprüfung der L1-Nachricht wird der sogenannte *Sequenz-Zähler* verwendet. Er wird bei Beginn jedes Zyklus zurückgesetzt und ansonsten mit dem LHC-Takt inkrementiert. Die konfigurierbare *Fenster-Logik* wertet den Zählerwert aus und löst zu bestimmten Zeitpunkten Kontrollsignale für ein gültiges L1- und L1-Nachricht-Fenster aus. Verzögerungen, die durch Registerstufen verursacht werden, sind in der Fenster-Logik berücksichtigt. Bei der Konfiguration kann daher direkt die Anzahl von Bunch Crossings nach dem L0-Trigger angegeben werden. Obwohl das Fenster für einen L1- Trigger definitionsgemäß eine Breite von Eins hat, kann es zu Testzwecken größer konfiguriert werden.

[3] Ein fehlender L1-Trigger kann nicht von einem zum L1-Zeitpunkt verworfenen Ereignis unterschieden werden. Diese Fehlerbedingung wird daher vom L2-Handler überprüft.

Trigger, die keinen Fehler ausgelöst haben, werden als valide angesehen. Auf Basis dieser *validen Trigger* werden die entsprechenden Steuersignale ausgelöst. Damit ist sichergestellt, dass nur bis dahin korrekte Ereignisse gepuffert und Informationen hierüber im Event-FIFO gespeichert werden. Zusätzlich werden die validen Trigger als Zeitbasis an andere Einheiten weitergegeben. Im Falle eines validen L1-reject wird ein explizites L1-reject-Signal erzeugt. Die ausschließliche Verwendung validierter Trigger in allen Einheiten sorgt dafür, dass Fehlerbedingungen nicht zu unvorhersehbaren Systemzuständen führen können.

Um festzustellen, ob ein L0-Trigger eintrifft, während das Busy-Signal gesetzt ist, überprüft der L0-Handler das von der Segmentsteuerung erzeugte segmentweite Busy-Signal. Mit der Überprüfung ist sichergestellt, dass in diesem Segment kein solcher Fehler vorliegt. Da die Segmente unabhängig arbeiten, könnte jedoch die Situation auftreten, dass einige Segmente ihr Busy-Signal noch gesetzt haben, während ein L0-Trigger eintrifft und andere nicht. Das globale Busy-Signal wird von der TGU nicht zurück an die SMUs verteilt. Die Segmentsteuerungen würden diesen L0-Trigger dann unterschiedlich interpretieren. Während die beschäftigten Segmente einen Fehler erkennen würden, arbeiten die anderen Segmente die Triggersequenz normal ab. Dies stellt jedoch kein Problem dar, da in den arbeitenden Segmenten keine wirklichen Fehler vorliegen und die anderen Segmente den Fehler korrekt erkennen und entsprechend behandeln.

6.5.2 Erzeugung der Steuersignale

Die Verarbeitung und Interpretation der Trigger-Sequenzen basiert auf einer Zustandsmaschine, die den erwarteten Ablauf einer Triggersequenz nachbildet. Die Zustände sind in Abbildung 6.5 dargestellt. Die Verarbeitung lässt sich in zwei Phasen aufteilen: Die in jeder Sequenz vorhandene Phase bis zur L1-Entscheidung und die Verarbeitung des L1-accept oder L1-reject.

Nach einem Reset befindet sich die Zustandsmaschine im *IDLE*-Zustand. Dies ist der einzige Zustand, in dem `l0_busy` nicht gesetzt ist. Trifft ein L1-Trigger ein, erfolgt der Wechsel in den Zustand *SEEN_L0*. Der Sequenz-Zähler wird gestartet, der Wert des Event-ID-Zählers gespeichert und `expect_data` gesetzt. Signalisiert der Sequenz-Zähler den Beginn des L1-Fensters, erfolgt der Übergang in den Zustand *WINDOW_L1*. Wird innerhalb des gültigen Fensters kein L1-Trigger empfangen, wird ein `event_flush`-Puls ausgelöst und über den *NO_L1*-Zustand zurück in den *IDLE*-Zustand gewechselt. Wird ein L1-Trigger empfangen, beginnt die zweite Phase mit dem Wechsel in den *SEEN_L1*-Zustand. Ist das Zeitfenster für die L1-Nachricht gültig, wird in den *WINDOW_L1M*-Zustand gewechselt und der Empfang der Nachricht abgewartet. Trifft keine oder eine fehlerhafte L1-Nachricht ein, wird ein Fehler ausgelöst. Ansonsten wird in den *DATA_RX*-Zustand gewechselt und gewartet, bis alle TMUs den vollständigen Empfang eines Ereignisses von der FEE signalisiert haben. Ist dies geschehen, wird `expect_data` deaktiviert und die Informationen über das Ereignis im Event-FIFO gespeichert. Nach dem Wechsel in den *IDLE*-Zustand ist der L0-Handler bereit für eine neue Triggersequenz. Wird in einem Zustand ein Fehler erkannt, erfolgt ein sofortiger Wechsel in den Zustand *ERROR_WAIT* und `busy_error`

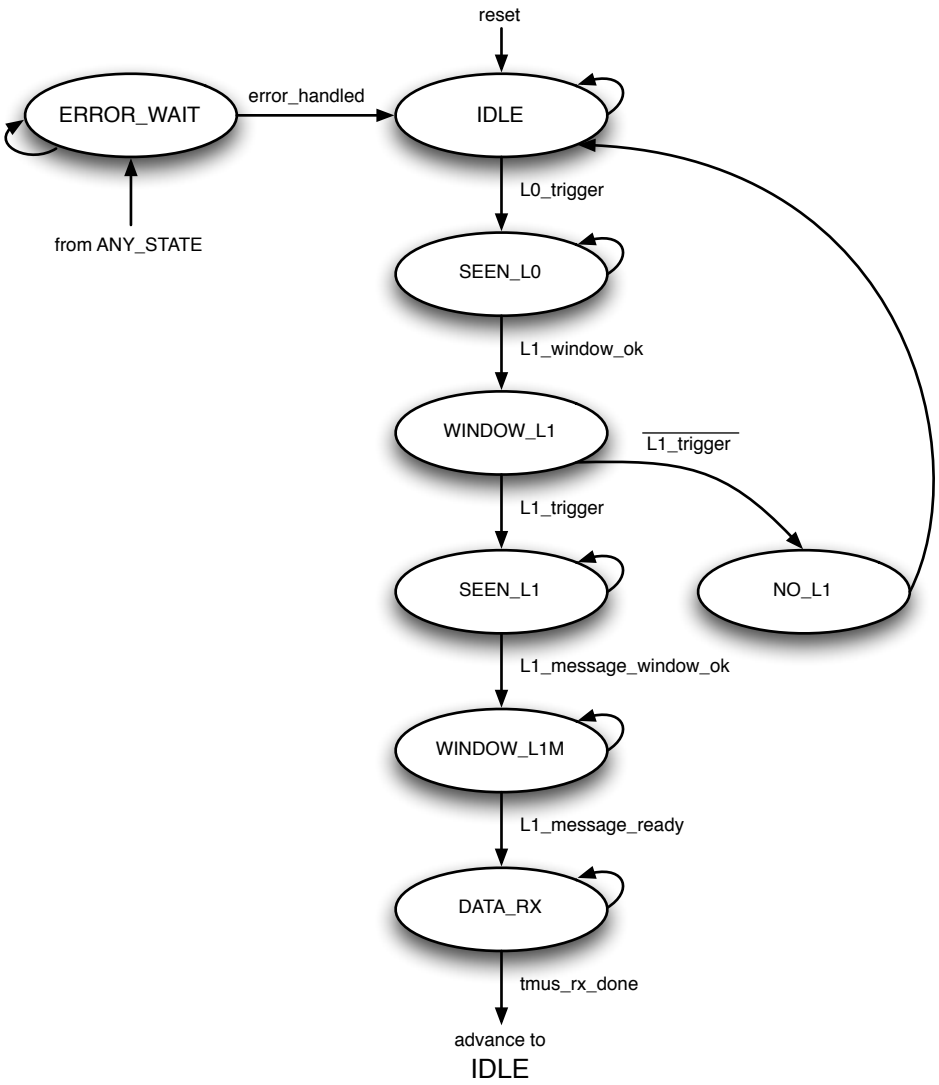


Abbildung 6.5: Die Zustandsmaschine des L0-Handlers. Die Zustände bilden den erwarteten Ablauf einer L0-L1-Triggersequenz nach. Wird in einem der Zustände ein Fehler erkannt, erfolgt ein sofortiger Wechsel in den Fehlerzustand.

wird gesetzt. Um zu verhindern, dass die TMUs ein fehlerhaftes Ereignis endgültig puffern, bleibt `expect_data` gesetzt. Ist die Fehleranalyse abgeschlossen erfolgt abhängig vom Ergebnis der Analyse ein Event Flush. Danach kehrt der L0-Handler in den Grundzustand zurück.

6.6 Der L2-Handler

Der L2-Handler verarbeitet die eintreffenden L2-Trigger. Zu seinen Aufgaben gehören damit die Auslese und das Verwerfen von gepufferten Ereignissen nach einer L2 -Entscheidung sowie die Überprüfung der im Zusammenhang mit L2-Triggern möglichen Fehler.

6.6.1 Sequenzverifikation

Bei der Fehlererkennung durch den L2-Handler wird die Tatsache ausgenutzt, dass allen gepufferten Ereignissen korrekte L0- und L1-Trigger zugrunde liegen. Dies wird von der Fehlererkennung des L0-Handlers sichergestellt.

Eine Fehlerbedingung, die bei der Abarbeitung von L2-Triggern erkannt werden muss, ist ein zu spät eintreffender L2-Trigger. Die Bedingung für einen validen L2-Trigger ist ein Eintreffen in einem Zeitraum zwischen 88 μs und 500 μs nach der Kollision. Im Gegensatz zum L0-Handler ist es beim L2-Handler nicht vernünftig, die Zeitfenster mittels eines Zählers zu bestimmen. Denn hier wäre für jede gestartete, aber noch nicht abgearbeitete Trigger-Sequenz ein eigener Zähler notwendig. Ein aufwendiges System zur Verwaltung der Zähler würde benötigt.

Stattdessen ist die Kontrolllogik auf der Basis von Zeitstempeln implementiert. Trifft ein L2-Trigger ein, wird die in der L2-Nachricht enthaltene Event-ID (vgl. 4.1) mit dem aktuellen Wert des lokalen Event-ID-Zählers verglichen. Aus der Differenz geht der zeitliche Abstand des L1-Triggers und des Eintreffens des L2-Triggers hervor. Das Ergebnis der Prüfung wird mit der Triggernachricht im L2-FIFO gespeichert.

Zwei weitere, mögliche Fehlerfälle sind ein unerwarteter und ein fehlender L2-Trigger. Einem unerwarteten L2-Trigger kann ein falsch gesendeter L2-Trigger, aber auch ein fehlender L0-Trigger zugrunde liegen. In beiden Fällen liegt kein gepuffertes Ereignis für diesen L2-Trigger vor. Bei einem fehlenden L2-Trigger ist ein Ereignis zu viel gepuffert. Alle Fehlerbedingungen können durch einen Vergleich der in der L2-Nachricht enthaltenen Event-ID des CTP und der für das gepufferte Ereignis gespeicherten Local-Event-ID erkannt werden. Der L2-Handler bearbeitet immer das älteste vorliegende Tupel aus Ereignisinformation und L2-Trigger. Passen die Werte der Event-IDs zueinander, liegt kein Fehler vor. Ist die Differenz größer als erwartet, handelt es sich bereits um die Ereignisinformation des nächsten Ereignisses. Es liegt also ein unerwarteter L2-Trigger vor. Ist die Differenz zu klein, handelt es sich um den L2-Trigger des nächsten Ereignisses, es hat also ein L2-Trigger gefehlt.

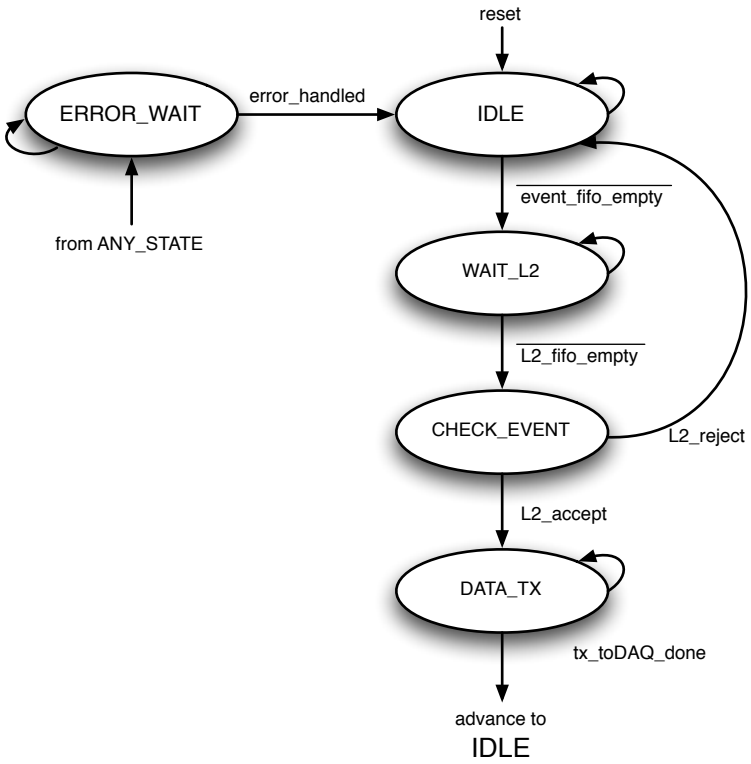


Abbildung 6.6: Die Zustandsmaschine des L2-Handlers. Liegen in beiden FIFOs Informationen vor, wird das Ereignis überprüft und anschließend ausgelesen bzw. verworfen. Wird ein Fehler erkannt oder löst einer der Timeout-Zähler aus, erfolgt ein sofortiger Wechsel in den Fehlerzustand.

Im Spezialfall, dass ein Fehler in der letzten Triggersequenz auftritt, funktioniert dieses Verfahren nicht mehr, da keine nächsten Ereignisse oder Trigger eintreffen. Es existiert also kein Tupel, das verarbeitet werden kann. Aus diesem Grund wird ein zusätzliches Timeout-Signal benötigt. Dieses kann in beiden Fällen vom selben Timeout-Zähler erzeugt werden. Der Zähler wird dabei gestartet, wenn entweder eine Ereignisinformation oder ein L2-Trigger zur Verarbeitung vorliegt, und zurückgesetzt, wenn ein Ereignis komplett verarbeitet ist. Weil dieser Timeout nicht sehr genau bestimmt sein muss, wird der Zähler um Ressourcen zu sparen nicht mit jedem Takt, sondern lediglich mit jedem Orbit inkrementiert. Erreicht der Zähler einen konfigurierbaren Wert, wechselt der L2-Handler in den Fehlerzustand. Um zu verhindern, dass ein Timeout auftritt, wenn durch einen Fehler im L0-Handler die Bearbeitung gestoppt ist, darf der Timeout-Zähler in diesen Fall nicht inkrementiert werden.

6.6.2 Erzeugung der Steuersignale

Der L2-Handler bearbeitet zyklisch Tupel aus gepufferten Ereignissen und L2-Triggern. Die fünf Zustände der Zustandsmaschine des L2-Handlers sind in Abbildung 6.6 dargestellt. Die Zustandsmaschine verbleibt so lange im *IDLE*-Zustand, bis Informationen über ein gepuffertes Ereignis im Event-FIFO vorliegen. Danach wartet sie im *WAIT_L2*-Zustand bis ein L2-Trigger im L2-FIFO vorliegt. Wird in einem der beiden Zustände ein Timeout vom Timeout-Zähler signalisiert, erfolgt der Wechsel in den *ERROR_WAIT*-Zustand. Ist ein Tupel komplett, wird im *CHECK_EVENT*-Zustand die in Abschnitt 6.6.1 dargestellte Verifikation durchgeführt und, wenn kein Fehler vorliegt, fortgefahren. Handelt es sich um einen L2r-Trigger, wird ein `event_reject`-Puls ausgelöst, der den TMUs signalisiert, das älteste Ereignis zu verwerfen. Danach kehrt der L2-Handler in den *IDLE*-Zustand zurück. Im Falle eines L2a-Triggers wird der Readout Control Unit signalisiert, mit der Datenübertragung zur DAQ zu beginnen. Diese löst dann einen `event_accept`-Puls aus und steuert die weitere Übertragung. Signalisiert ein Puls auf `tx_toDAQ_done` eine abgeschlossene Datenübertragung, wechselt der L2-Handler von *DATA_TX* wieder in den *IDLE*-Zustand und ist bereit, einen weiteren Eintrag im L2-FIFO zu bearbeiten.

6.7 Status und Ausblick

Mit Hilfe des als Hardware/Software-Co-Designs realisierten Konzepts lässt sich in effizienter Weise eine MEB-fähige Segmentsteuerung implementieren. Durch die gewählte Aufteilung ist die Anzahl der verwaltbaren Puffer nur durch die Größe der verwendeten FIFOs beschränkt. Gegenwärtig ist eine Maximalzahl von 512 gewählt. Dies überschreitet den zur Verfügung stehenden Speicher in den TMUs für durchschnittlich große Ereignisse bei Weitem. Zusammen mit einer dynamischen Speicherverwaltung in den TMUs kann so stets eine größtmögliche Anzahl von Puffern bereitgestellt werden. Dies macht das System auch bei sich ändernden Bedingungen flexibel und anpassungsfähig.

Die komplette Segmentsteuerung verbraucht in der vorliegenden Implementation 6 Block-RAMs, 1795 4-Input-LUTS, 59 5-Input-LUTs, 814 Carry-Blöcke sowie 1907 Register. Der größte Teil der Register entfallen dabei auf die Synchronisierung von Signalen zwischen verschiedenen Clock Domains.

Das bestehende HDL-Design ist in Simulationen auf korrekte Funktionalität und umfassende Fehlererkennung getestet. Dabei erfolgte die Ablaufsteuerung über den in Kapitel 7 vorgestellten CTP-Emulator unter Berücksichtigung des von der Segmentsteuerung generierten Busy-Signals. Weiterhin konnte gezeigt werden, dass das Design Triggersequenzen kontinuierlich korrekt verarbeitet und die benötigten Steuersignale auslöst. Mit Hilfe fehlerhafter Triggersequenzen wurde überprüft, dass typische Triggerfehler erfolgreich erkannt werden können.

In Tests mit realer GTU-Hardware wurde das Design zusammen mit einer auf einen Puffer beschränkten Variante der TMU in einem Testaufbau am CERN erfolgreich getestet.

Anstelle der FEE wurden die, in den TMUs vorhandenen Testmustergeneratoren verwendet, um einen überprüfbaren Datenstrom zu erzeugen. Die TTC-Signale wurden von einer Local Trigger Unit (LTU) im Emulatormodus generiert. Um Fehler aufzudecken, welche nur in bestimmten Konstellationen von Parametern auftreten können, wurden die Tests mit unterschiedlichen Kombinationen von Ereignisgrößen und Triggerraten durchgeführt. Ausgelesene Daten und erzeugte Header wurden bitweise überprüft und waren in allen Konstellationen korrekt.

Nach einer Anpassung des TMU-Designs für MEB können weitere Tests mit mehreren Puffern in Simulation und echter Hardware folgen. Ist sichergestellt, dass SMU und TMU im MEB-Betrieb wie vorgesehen zusammenarbeiten, kann mit der detaillierten Implementierung der PPC-Software für die automatische Fehleranalyse und Wiederherstellung begonnen werden.

Die notwendige Priorisierung der Schreibzugriffe auf den SRAM der TMUs (vgl. 4.2) führt zu Lücken im Auslesedatenstrom. Diese werden bisher von FIFOs im SMU-Datenpfad geglättet. Sollte sich herausstellen, dass diese FIFOs nicht mehr ausreichen und zu große Lücken im Datenstrom entstehen, kann der auf der SMU ungenutzte SRAM als weiterer Zwischenpuffer eingesetzt werden. Dieser Puffer kann ohne Änderungen am restlichen Design transparent in den Datenpfad eingebunden werden. Um eine Aussage zu treffen, wann sich ein solcher Einsatz lohnt, müssen die realen Übertragungszeiten inklusive der Lücken im Betrieb überwacht werden.

7 Die GTU-Simulation

Für Weiterentwicklungen im GTU-Projekt ist es notwendig, das HDL-Design möglichst umfassend simulieren zu können. Insbesondere für Aufgaben, bei denen mehrere FPGAs interagieren müssen, wie z. B. bei Multi-Event Buffering, reichen isolierte Simulationen einzelner Komponenten nicht aus. Viele Effekte, die zu Fehlverhalten führen können, kommen erst in Gesamtsystem zum Tragen. Weiterhin genügt es nicht, lediglich den meist als Verhaltensbeschreibung realisierten HDL-Code zu simulieren. Das Verhalten dieser Beschreibung weicht unter Umständen deutlich von dem des synthetisierten FPGA-Designs ab. Aus diesen Gründen wird eine möglichst realitätsnahe GTU-Simulation benötigt, mit der sich alle FPGA-Designs der GTU gemeinsam simulieren lassen. Es muss dabei möglich sein, sowohl Verhaltenssimulationen, als auch *Gate-Level*-Simulationen von implementierten Designs durchzuführen. Der Aufbau einer entsprechenden Simulation und die dafür verwendeten Modelle werden in den folgenden Abschnitten vorgestellt.

7.1 Erstellung von Gate-Level-Simulationen

Ein FPGA-Design durchläuft einen mehrstufigen Implementationsprozess. Für die Xilinx-Toolchain ist der vollständige Implementationsprozess in der linken Spalte von Abbildung 7.1 dargestellt [33]. Ein Design kann mittels seiner HDL-Beschreibung und nach jeder einzelnen Stufe des Implementationsprozesses simuliert werden. Die Simulation des initialen Designs wird als Verhaltenssimulation bezeichnet. Hat das Design einen oder mehrere Schritte des Implementationsprozesses durchlaufen, wird im Folgenden vereinfachend von einer Gate-Level-Simulation gesprochen. Je weiter die Implementierung fortgeschritten ist, desto genauer bildet die Simulation das Verhalten der realen Hardware nach. Eine Verhaltenssimulation beispielsweise beinhaltet keinerlei Information über die Verzögerungen durch Logikelemente oder Signallaufzeiten im Chip. In einer Gate-Level-Simulation mit Timing-Backannotation hingegen stehen sämtliche Zeitinformationen zur Verfügung. Entsprechend steigt die Komplexität der Simulation und die Rechenzeit für die Ausführung der Simulation stark an.

Während Verhaltensbeschreibungen in VHDL direkt simuliert werden können, werden für Gate-Level-Simulationen Instanzen von, durch die Technologie vorgegebenen, Grundkomponenten verwendet. Je nach Stufe der Implementation sind diese in der UNISIM- oder SIMPRIM-Bibliothek enthalten. Um ein implementiertes Design simulieren zu können, muss aus dem Ergebnis eines Implementationsschrittes eine simulierbare HDL-Netzliste erzeugt werden. Hierfür stellt Xilinx des Programm *NetGen* zur Verfügung. Neben der Netzliste gibt dieses Programm auch die Zeitinformation als zusätzliche SDF-Datei aus.

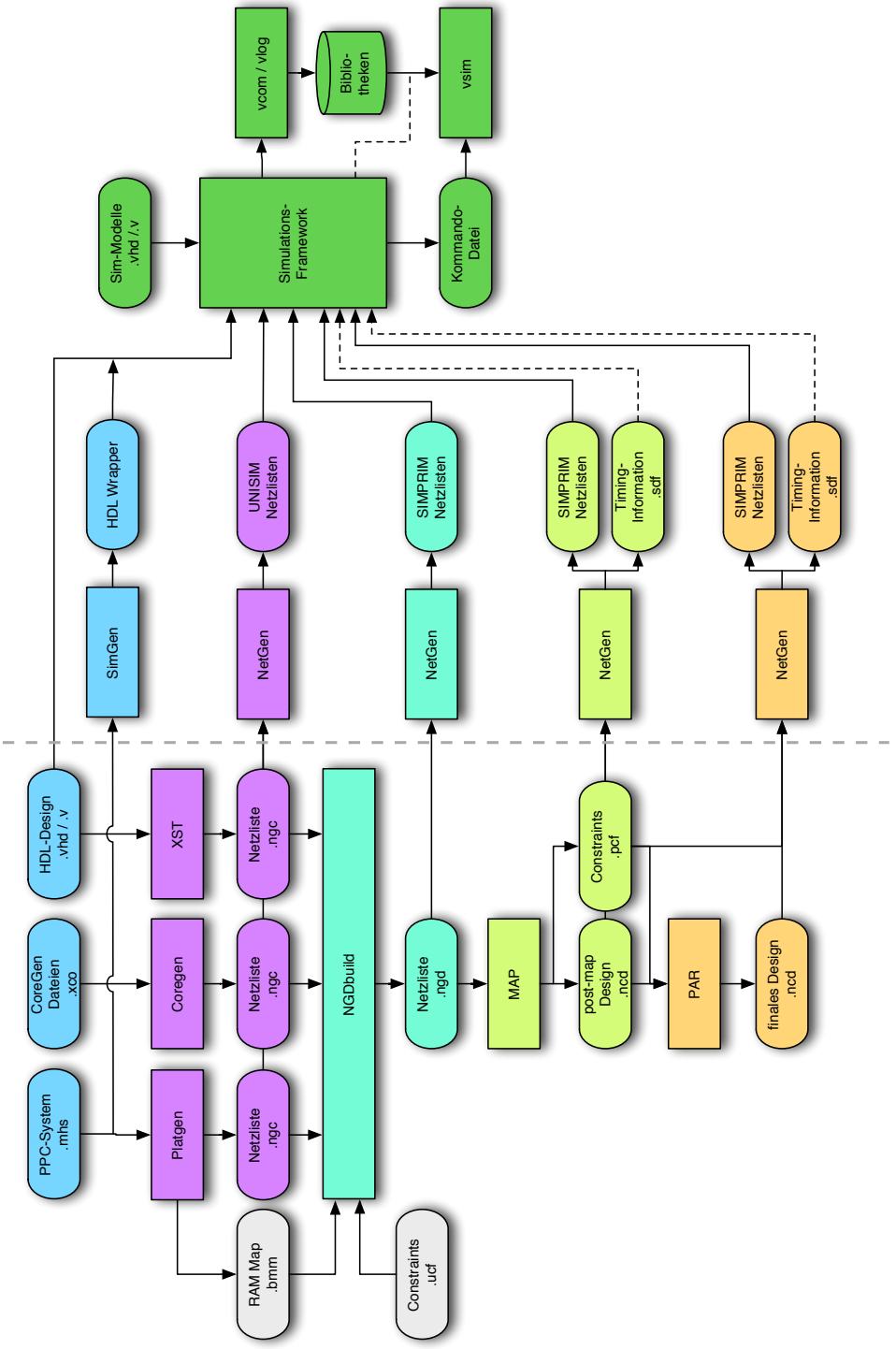


Abbildung 7.1: Links abgebildet ist der vollständige FPGA-Implementationsprozess mittels Xilinx-Software. In jeder Stufe können Netzlisten zur Simulation erzeugt werden. Mit Hilfe des Simulationsframeworks kann eine beliebige Kombination dieser Netzlisten simuliert werden.

Sollen nur Teile eines Designs auf Gate-Level-Ebene simuliert werden, können separate Netzlisten einzelner Komponenten erzeugt werden [34].

Für das GTU-Projekt wird der gesamte Implementationsprozess durch ein Makefile gesteuert. An den entsprechenden Stellen finden sich nun auch die Ziele für die Erzeugung der für die Simulation benötigten Netzlisten. Da deren Erstellung für die gesamte GTU rund 9 Stunden dauert, werden diese Modelle normalerweise nur in den *Nightly Builds* erzeugt.

7.2 Das Simulationsframework

Zur Durchführung einer HDL-Simulation werden die vorhandenen HDL-Quellen in simulatoreigene Simulationsbibliotheken übersetzt. Dies geschieht mittels eines vom Simulator zur Verfügung gestellten Compilers. Ist dieser Schritt abgeschlossen, kann die Simulation beginnen. Werden Quellen verändert, müssen die Simulationsbibliotheken entsprechend aktualisiert werden. Bei großen Projekten ist das vollständige Recompilieren vieler Quellen recht zeit-intensiv und selbst, wenn dies in Form eines Skripts erfolgt, ist das manuelle Pflegen der Quellenliste mühsam und fehleranfällig. Für eine gezielte Aktualisierung einer Bibliothek müssen, neben den veränderten Quellen selbst, auch die in der Hierarchie übergeordneten Komponenten neu kompiliert werden. Das im Rahmen dieser Arbeit entwickelte Simulationsframework unterstützt den Entwickler beim Erstellen der Quellenlisten und automatisiert das Recompilieren betroffener Quellen.

Dazu wird jede Simulation in einer *Simulations-Projektdatei* mit den folgenden Angaben definiert:

- Liste der Quellen für das Design und die zugehörigen Test-Modelle
- Bibliotheken, in welche die Quellen zu kompiliert sind
- Zu verwendende Modelle der Komponenten (Verhaltens- oder Gate-Level-Modell)
- Simulatorkommandos und -skripte für den Start und den Ablauf der Simulation

Mit einer solchen Projektdatei ist eine Simulation vollständig beschrieben und kann automatisiert wieder erstellt und durchgeführt werden. Die teils mehrere Megabyte großen, sich ständig ändernden Bibliotheken werden gegebenenfalls neu erzeugt und müssen nicht gespeichert werden. Dies bietet vor allem bei der Verwendung einer Versionskontrolle wie Subversion Vorteile.

Um die Herstellung einer Projektdatei und die darin enthaltene Liste aller Quellen für den Entwickler zu vereinfachen, bietet das Simulationsframework die Möglichkeit, eine Vorlage mit allen vorhandenen Quellen zu erstellen, aus denen der Entwickler nur die benötigten Quellen auswählen kann. Diese Vorlage basiert ausschließlich auf den auch für den Implementierungsprozess verwendeten Quellen und Netzlisten. Dadurch wird ausgeschlossen, dass andere Quellen simuliert werden, die im implementierten Design nicht enthalten sind.

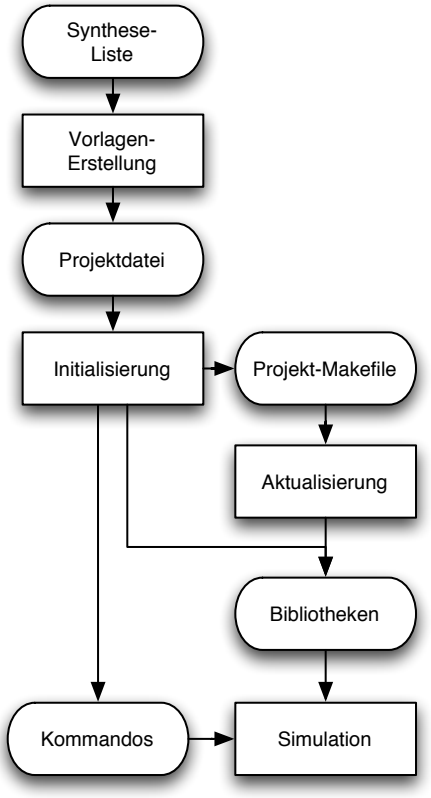


Abbildung 7.2: Ablauf einer Simulation mit Hilfe des Simulationsframeworks.

Da unabhängige FPGA-Designs eventuell gleichnamige Komponenten enthalten, dürfen sie nicht in eine gemeinsame Simulationsbiliothek zusammengefasst werden. Alle FPGA-Designs werden sonst mit der zuletzt kompilierten Komponente eines Namens simuliert. In der Projektdatei wird dementsprechend jeder Quelle eine passende Ziel-Bibliothek zugeordnet.

Durch die Vorgabe eines Simulationsskripts kann eine Simulation automatisiert gestartet und durchgeführt werden. So können beispielsweise zu Beginn der Simulation die relevanten Signale in die Ansicht der Simulatoroberfläche geladen und eine vorgegebene Zeitspanne simuliert werden. Der Entwickler kann ohne zusätzliche Kommandos direkt das Ergebnis der Simulation analysieren. Werden die Ausgangsvektoren der Simulationsmodelle in Dateien geschrieben, kann weiterhin nach Anschluss der Simulation eine vom Simulationsframework ausgelöste Überprüfung erfolgen. Somit ist es möglich, ausgewählte Simulationen mit vorgegebenen Tests voll automatisiert durchzuführen. Dies kann beispielsweise im Anschluss an den Nightly-Build-Prozess geschehen um alle erzeugten Designs automatisch zu überprüfen.

Die Arbeitsweise des Simulationsframeworks ist in Abbildung 7.2 dargestellt. Die Abläufe werden mittels eines Makefiles gesteuert, es werden daher immer nur die benötigten Schritte ausgeführt. Im Gegensatz zu einer starren Ablaufsteuerung mit Skripten kann

so häufig viel Zeit eingespart werden. Erster Schritt für eine neue Simulation ist die Erstellung der Projektdatei. Ist keine Projektdatei vorhanden, kann eine Vorlage auf Basis der zur Synthese verwendeten Quellen erstellt werden. Soll eine reine Verhaltenssimulation durchgeführt werden, kann die Vorlage direkt ohne Änderungen für die Simulation genutzt werden. Für andere Simulationskonfigurationen kann der Entwickler die Projektdatei entsprechend anpassen. Beim ersten Start einer Simulation werden die benötigten Bibliotheken im Initialisierungsschritt erstellt. Des weiteren erstellt das Simulationsframework eine Kommandodatei, welche dem Simulator beim Start übergeben wird und unter anderem Angaben über die auszuführenden Simulationsskripte enthält. Um für die Aktualisierung einer Bibliothek nicht alle Quellen neu kompilieren zu müssen, wird zusätzlich ein projektspezifisches Makefile erstellt, in dem die hierarchischen Abhängigkeiten der Bibliotheken abgebildet sind. Die Erstellung dieses Makefiles erfolgt mittels des von ModelSim bereitgestellten Kommandos *vmake* [26]. Ändern sich Quellen, können die Bibliotheken mittels dieses Makefiles gezielt aktualisiert werden.[1]

7.3 Gesamtaufbau der GTU-Simulation

Für eine Simulation werden neben den GTU-FPGA-Designs weiterhin Simulationsmodelle der sonstigen GTU-Komponenten und der externen Datenquellen benötigt. Bei den Komponenten sind vor allem externe Speicher und die Tochterkarten von SMU und TGU von Interesse. Betrachtet man die GTU, so gibt es zwei Datenquellen, die als Eingangssignale dienen: die FEE und das TTC-System. Auf der Ausgangsseite werden Daten an die DAQ und an den CTP gesendet. Für eine automatisierte Simulation und Verifikation werden zusätzlich Modelle benötigt, die die Ausgangssignale zur automatisierten Überprüfung aufzeichnen.

Die im Experiment eingesetzte GTU besteht aus insgesamt 109 FPGA-Karten. Es ist bei weitem nicht möglich, 109 FPGA-Designs gleichzeitig zu simulieren. Die dafür benötigte Rechenleistung und der Speicherverbrauch überfordert auch moderne Computersysteme. Für die Simulation muss daher ein auf ein Testszenario angepasstes Subset der GTU ausgewählt werden können. So genügt es beispielsweise in den meisten Fällen, eine Kombination aus einem GTU-Segment, bestehend aus fünf TMUs und einer SMU, und einer TGU zu simulieren. Um die Simulation möglichst modular zu gestalten, ist die Segmentierung der Simulationsmodelle an die Segmentierung der realen Hardware angelehnt. Für jede der drei GTU-Karten existiert ein eigenes Simulationsmodell, in dem das entsprechende FPGA-Design und die Modelle für externen Komponenten der Karte, wie Speicher und Tochterkarten, gekapselt sind. Des Weiteren existiert ein Modell einer TTC-Partition, mit dem sich die benötigten TTC-Signale erzeugen lassen. Der Datenstrom der FEE wird von unterschiedlichen Generatoren erzeugt, die im Modell der TMU hinzugefügt sind. Eine Gesamtübersicht der GTU-Simulation ist in Abbildung 7.3 dargestellt.

[1] Falls die grafische Simulatoroberfläche verwendet wird, muss diese für die Aktualisierung nicht mehr zeitaufwendig beendet und neu gestartet werden.

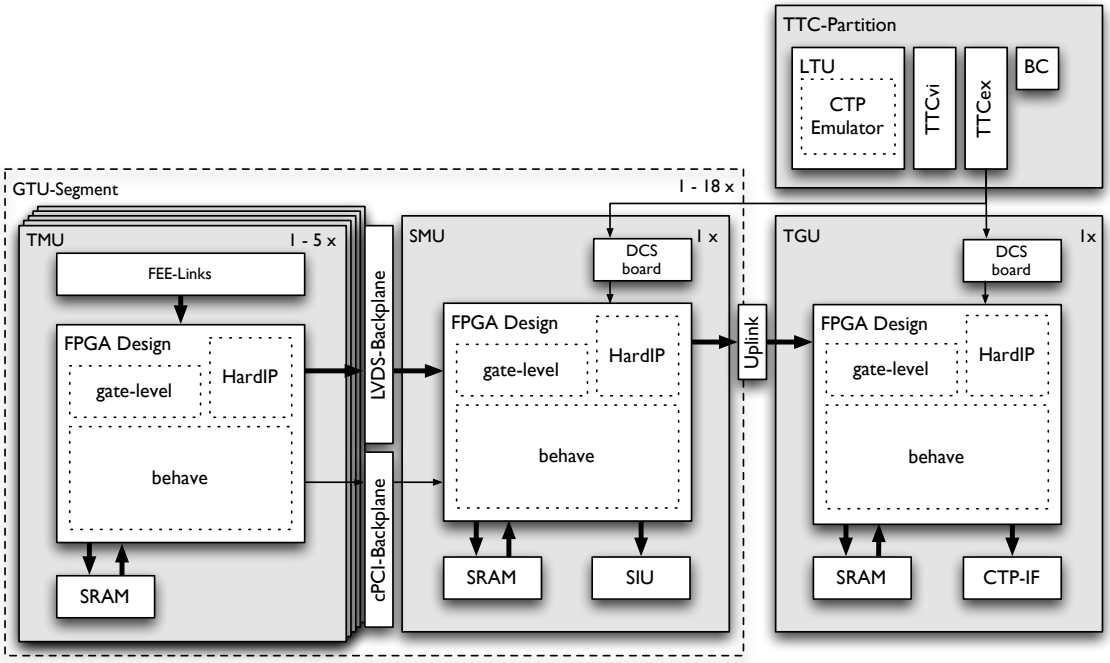


Abbildung 7.3: Übersicht der GTU-Simulation. Alle Simulationsmodelle sind in einer globalen Testbench zusammengefasst. Welcher Satz an Karten simuliert werden soll, kann frei konfiguriert werden. Das FPGA-Design kann als Verhaltensbeschreibung, als Gate-Level-Beschreibung oder gemischt simuliert werden.

Die Modelle der einzelnen Karten und der TTC-Partition sind in einer *globalen Testbench* zusammengefasst. Die Auswahl, welche Kombination von Karten simuliert werden soll, erfolgt über Masken in der Testbench. Hier kann ausgewählt werden, welche TMUs eines Segments und welche der 18 GTU-Segmente in der Simulation vorhanden sein sollen. Zusätzlich lässt sich die Simulation mit und ohne TGU durchführen, wenn beispielsweise lediglich das Auslesen von Ereignissen simuliert werden soll. Ist keine TGU vorhanden, wird das Busy-Signal der SMU direkt zur TTC-Partition weitergeleitet. Die FPGA-Designs unterstützen zur Anpassung an verschiedene Detektorkonstellationen den Betrieb mit verschiedenen Kombinationen von Karten. Im SMU-Design lässt sich beispielsweise konfigurieren, welche TMUs eines Segments vorhanden sind und angesprochen werden. Die gewählte Segmentierung der Simulationsmodelle macht es möglich, von diesen Funktionen für die Simulation Gebrauch zu machen. Die Designs können ohne Änderungen so konfiguriert werden, dass sie nur den in der Testbench ausgewählten Satz von Karten erwarten. Es ist nicht nötig, vereinfachte *Dummy-Modelle* einzelner Karten zu instantiieren, um eine Simulation zu ermöglichen.

Neben der Einbindung der Simulationsmodelle sind in der Testbench die Backplane-Verbindungen zwischen TMU und SMU sowie der TGU-Uplink zwischen SMU und TGU modelliert. Die LVDS-Backplane im realen System wird source-synchron zur SMU betrieben. Die Laufzeitverzögerung der Signale auf der Backplane und im FPGA führt zu

einer Phasenverschiebung zwischen Takt und Signal. Im realen System wird diese mit Hilfe der im FPGA vorhandenen *IDELAY*-Zellen kalibriert um die Laufzeitverzögerung auszugleichen. In der reinen Verhaltenssimulation werden Verzögerungen durch Logik und Signaltransport normalerweise nicht modelliert. Logik und Signaltransport geschehen in Nullzeit. Während die Xilinx-Verhaltensmodelle der IDELAY-Zellen keine Verzögerung einführen, weisen die Verhaltensmodelle der Digital Clock Managers (DCMs) jedoch eine Verzögerung in Form einer Phasenverschiebung im abgeleiteten Takt auf. Es ist daher auch in der Simulation nötig, die Verbindungen zu kalibrieren. Hierfür können die Laufzeitverzögerungen der Backplane-Verbindungen und des TGU-Uplinks in der Testbench individuell spezifiziert werden. Der aufwendige Kalibrierungsmechanismus wie im realen Betrieb ist dadurch nicht erforderlich.

7.4 Simulationsmodelle der GTU-Karten

Die FPGA-Designs und Modelle für alle weiteren Komponenten einer GTU-Karte sind in drei Kartenmodellen gekapselt. Um Simulationszeit zu sparen, sind die Simulationsmodelle so einfach wie möglich gehalten. Beispielsweise wird auf die Modellierung der auf einer GTU-Karte vorhandenen Sensoren verzichtet. Momentan beinhaltet jedes Kartenmodell neben den FPGA-Designs drei Oszillatoren sowie ein Modell für die externen SRAMs, welches vom Hersteller stammt. Die Laufzeitverzögerung zwischen FPGA und SRAM ist aus den gleichen Gründen wie bei der LVDS-Backplane frei konfigurierbar. Mit dieser Konfiguration lässt sich das vollständige Verarbeiten eines Ereignisses und die Funktion der, die Triggerberechnungen ausführenden, Einheiten simulieren. Die verbleibenden I/O-Pins des FPGA sind entsprechend ihrer Verwendung im realen System fest auf neutrale Werte gesetzt. Somit lassen sich die Modelle auf einfache Weise um weitere Komponenten erweitern. Dies gilt beispielsweise für ein Modell des externen SDRAM, das bereits vorhanden ist, aber gegenwärtig nicht verwendet wird.

Zusätzlich zu diesen Komponenten enthalten die Modelle für SMU und TMU weitere Modelle für die Tochterkarten. Bei der SMU handelt es sich dabei um ein DCS-Board und eine SIU. Die TGU ist mit einem DCS-Board und einem CTP-Interface bestückt. Die Simulationsmodelle der Tochterkarten werden in den folgenden Abschnitten genauer erläutert.

7.4.1 Das DCS-Board-Modell

Wie in Kapitel 3 beschrieben, erfüllt das DCS-Board mehrere Funktionen. Für eine hinreichende GTU-Simulation ist es nicht notwendig, alle Funktionen in einem Simulationsmodell abzubilden. Dadurch kann die Komplexität des Modells erheblich verringert werden. Da in HDL-Simulationen nur Logik simuliert wird, ist es nicht notwendig, die FPGA-Konfigurationsfunktion via JTAG zu modellieren. Die Kommunikationsfunktion des DCS-Boards wird im realen System dazu verwendet, Konfigurationsregister im GTU-Design zu setzen und Statusregister auszulesen. Diese Funktionen sind auch in der Simulation nötig,

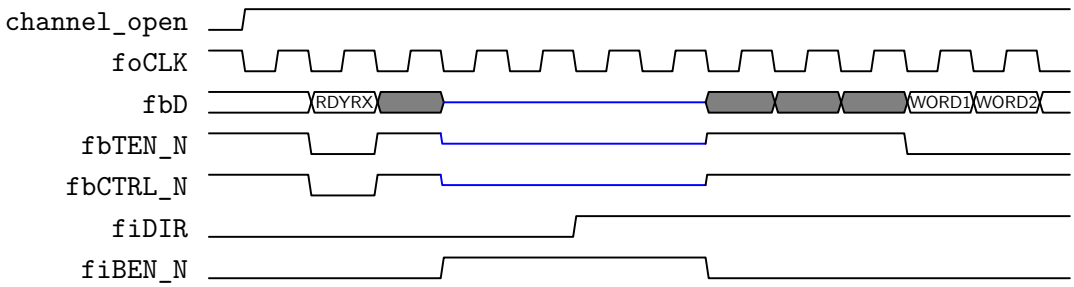


Abbildung 7.4: Signale des SIU-Interface beim Wechsel des DDL in den Ereignisdatenmodus. Der Wechsel wird durch die Aktivierung des Eingangs `channel_open` ausgelöst. Die dargestellten Signale werden automatisch vom SIU-Modell generiert.

hier kann jedoch direkt auf die Register zugegriffen werden. Für die Simulation ist das Interface des DCS-Boards zum TTC-System von Interesse. Da die Decodierungsfunktion des TTCrx-Chips bei der GTU keine Verwendung findet, besteht kein Bedarf für ein vollständiges TTCrx-Modell. Die Signale, die als Input für das DCS-Board-FPGA dienen, können direkt von der Triggersystem-Simulation bezogen werden. Das vom DCS-Board-FPGA ausgeführte Zeit-Multiplexing von A- und B-Kanal und die Erzeugung differentieller Signale wird vom Simulationsmodell gekapselt. Außerdem bietet das Modell die Möglichkeit, auch ohne TTC-Eingangssignale den 40-MHz-LHC-Takt bereitzustellen.

7.4.2 Das SIU-Modell

Die SIU ist Teil des DDL und bildet das Interface zwischen der FEE, hier der GTU, und dem DDL *physical layer*. Für eine vollständige Simulation der GTU ist ein SIU-Modell unverzichtbar. Im Experiment besteht die Hauptaufgabe des DDL darin, Ereignisdaten von den Detektoren zur DAQ zu übertragen. Weiterhin bietet er auch andere Modi, die es erlauben, mit dem Detektor zu kommunizieren und arbiträre Daten in beide Richtungen zu übertragen. Der einzige von der GTU verwendete DDL-Modus ist der zur Übertragung von Ereignisdaten. Um ein Ereignis in der GTU zu simulieren, muss der DDL vom Grundzustand in den Ereignisdaten-Modus geschaltet werden. Ansonsten verhindert die GTU, analog zum realen System, den Start einer Triggersequenz über das Setzen ihres Busy-Signals. Zur Überprüfung des Datenpfades ist es zudem erforderlich, die über den DDL gesendeten Daten aufzuzeichnen. Beide Funktionalitäten sind im SIU-Modell implementiert.

Die Kommunikation zwischen SIU und GTU erfolgt über einen 32-bit breiten bidirektionalen Datenbus. Um einen DDL-Modus zu wählen, schickt die DAQ Kommandoworte zur SIU. Diese beeinflussen entweder die SIU direkt oder werden, den momentanen Busstatus beachtend, an die GTU weitergeleitet [30].

Um die Verwendung des SIU-Modells so einfach wie möglich zu halten, sind sowohl das Busprotokoll als auch die Generierung der Kommandoworte im Modell gekapselt. Das Modell hat neben dem SIU-Interface zum FPGA lediglich zwei Signale. Über den Eingang

`channel_open` lässt sich der DDL-Modus zwischen Grundzustand und Ereignisdatenmodus umschalten. Die erzeugten Signale sind in Abbildung 7.4 beispielhaft für den Wechsel in den Ereignisdatenmodus dargestellt. Bei einer steigenden Flanke auf `channel_open` wird vom SIU-Modell das Kommandowort *Ready to Receive (RDYRX)* an die GTU gesendet. Danach signalisieren `fiDIR` und `fiBEN_N` den Wechsel der Busrichtung und das Modell ist bereit, Ereignisdaten entgegen zunehmen. Durch eine fallende Flanke auf `channel_open` wird der Ereignisdatenmodus beendet. Wenn keine Datenübertragung aktiv ist, wird die Busrichtung vom SIU-Modell erneut gedreht und das Kommandowort *End of Block Transfer (EOBTR)* an die GTU gesendet. Wird gerade ein Ereignis übertragen, wartet das Modell, bis die GTU das *Front-End Status Word (FESTW)* mit gesetztem *End of Data (EOD)* Flag sendet und beendet den Modus erst anschließend. Mit Hilfe des Eingangs `ddl_busy` lässt sich ein Ansprechen der DDL-Flusskontrolle simulieren. Um auch hier die Handhabung zu vereinfachen, ist der Eingang asynchron ausgelegt. Wird er gesetzt, signalisiert das SIU-Modell der GTU die Nichtbereitschaft, Daten entgegen zunehmen. Dies geschieht mittels des Signals `fiLF_N` des SIU-Interfaces synchron zum SIU-Takt.

Um auf einfache Weise verschiedene Simulationsdurchgänge miteinander vergleichen zu können und eine automatisierte Verifikation zu ermöglichen, werden die, von der GTU gesendeten, Ereignisdaten und Statusworte in einer Textdatei gespeichert. Das Schreiben der Datei ist mit Hilfe des VHDL-TextIO-Paketes realisiert. Das Format entspricht dem des Datenstroms in der DAQ. Damit können Simulationsdurchläufe auch mit Testdurchläufen in realer Hardware verglichen werden. Ein weiterer Vorteil ist, dass sich die bereits vorhandene Software zum Überprüfen von Testmustern mit kleinen Anpassungen auch in der Simulation einsetzen lässt.

7.4.3 Das CTP-Interface-Modell

Die CTP-Interface-Karte dient als Adapterkarte für die LVDS-Verbindungen zum CTP und ist mit LVDS-Treibern bestückt. Sie leitet den L1-Triggerbeitrag und das TRD-Busy-Signal vom TGU-FPGA an den CTP weiter. Für eine einfache Simulation wird kein eigenes Modell dieser Karte benötigt. Um automatisierte Simulationen durchführen zu können, ist dennoch ein Simulationsmodell vorhanden. Zur Überprüfung der, in der Simulation ausgelösten, zum CTP übertragenen Signale schreibt das Simulationsmodell der CTP-Interface-Karte alle Signaländerungen mit Angabe des Simulationszeitpunktes und der Anzahl der bisher vergangenen LHC-Takte in eine Textdatei. Diese kann nach Abschluss der Simulation überprüft und mit anderen Simulationen verglichen werden.

7.5 Simulation der FEE-Daten und MGTs

Die wichtigste Eingangsdatenquelle der GTU ist die FEE. Für die Simulation werden Generatoren benötigt, die einen Datenstrom bereitstellen wie ihn die FEE sendet. Wann welche Daten gesendet werden, muss, wie in der Realität, von den Triggersignalen des CTP abhängen. Je nach Testszenario kann der Datenstrom unterschiedlich aufgebaut sein. Um

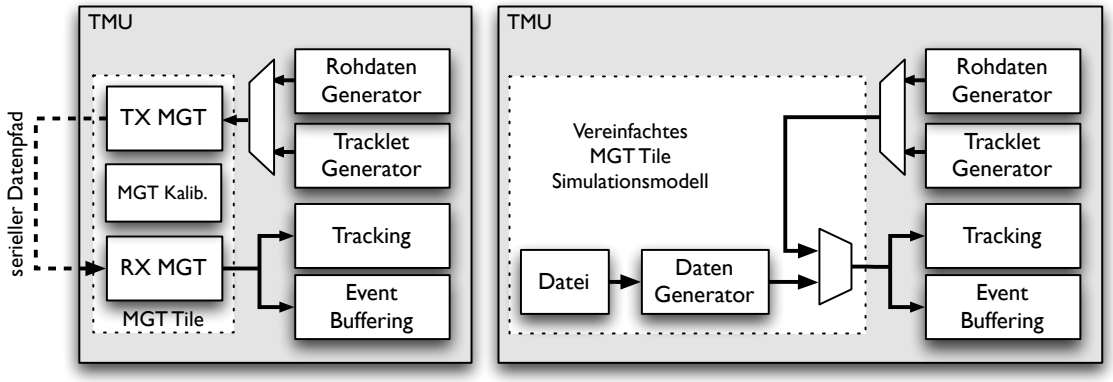


Abbildung 7.5: Simulationsmodell der MGTs. Links zu sehen ist der Aufbau der realen Hardware. Für die Simulation können die MGT-Blöcke mit einem vereinfachten Modell ersetzt werden, das den Datenstrom der FEE liefert.

die korrekte Funktion der Datenübertragung zu testen sind pseudo-zufällige Testmuster am besten geeignet. Diese können, nachdem sie die GTU durchlaufen haben, automatisiert überprüft werden. Soll die Funktion der triggerrelevanten Einheiten der GTU überprüft werden, müssen hingegen realistische Tracklets an die GTU gesendet werden. Diese können etwa aus AliRoot-Simulationen oder aus realen, am Experiment aufgenommen Daten gewonnen werden.

Die seriell gesendeten Daten der FEE werden in den TMUs mit Hilfe der im FPGA vorhandenen MGTs parallelisiert und stehen dann im Design zur Verfügung. Die Simulation der realen Empfangseinheiten inklusive der MGTs ist mit Hilfe von *SmartModels*[2] möglich, jedoch ergeben sich dadurch einige Nachteile. Die Modelle benötigen eine Simulationsauflösung von Picosekunden anstatt der, für anderen Zwecke ausreichenden, Auflösung von Nanosekunden. Dies erhöht den Simulationsaufwand und die dadurch benötigte Zeit deutlich. Für Ansteuerung und Kalibrierung der MGTs enthält das TMU-Design aufwendige Zusatzlogik. Die realen MGTs benötigen sehr lange Kalibrierungssequenzen von mehreren Millionen Takten, die von der Ansteuerlogik realisiert werden. Dies führt zu extrem langen Vorlaufzeiten, bevor mit der Simulation der Datenübertragung begonnen werden kann. Da die Funktion der MGTs und der Ansteuerlogik bereits ausführlich getestet und in Hardware verifiziert ist, und an diesen Stellen des Designs in Zukunft keine Änderungen mehr vorgenommen werden sollen, müssen die realen MGT-Blöcke nicht zwingend detailgenau simuliert werden.

Die GTU-Simulation enthält daher, zusätzlich zu den SmartModels der MGTs, vereinfachte Simulationsmodelle, welche die MGTs und die Ansteuerungslogik ersetzen. Der Aufbau der Modelle ist in 7.5 dargestellt. Die Modellen bieten die Möglichkeit, zwischen verschiedenen Datenquellen für die Simulation zu wählen. Dabei stehen mehrere Varianten zu Verfügung. In den TMU-Designs existiert ein Testmustergenerator für Tests in echter Hardware. Dieser kann auch in der Simulation genutzt werden. Er sendet einen festen

[2] *SmartModels*: siehe 7.7.1.

Satz realistischer Tracklets und ein bekanntes Testmuster für die Rohdaten. Die Daten werden vom Simulationsmodell an die Dateneingänge des TMU-Designs zurückgeschleift. Ein weitere Testmustergenerator ist im Simulationsmodell enthalten. Mit ihnen kann ein Pseudo-Zufalls-Muster für Rohdaten und Tracklets oder ein frei konfigurierbarer Satz an Tracklets gesendet werden. Welcher Generator zum Einsatz kommt, kann zur Simulationszeit gewählt werden. Die vorhandenen Generatoren ermöglichen alle bisher benötigten Testszenarien. Besteht der Bedarf für andere Testmuster, können neue Generatoren dem Simulationsmodell in einfacher Weise hinzugefügt werden.

7.6 Die Trigger-System-Simulation

Im Experiment erhält die GTU alle zur Datennahme benötigten Steuersignale vom Trigger-System. Daher ist es unerlässlich, diese Signale auch in der Simulation bereitzustellen. Dabei möchte man auf möglichst einfache Weise die benötigten Signale für beliebige, auch fehlerhafte, Triggersequenzen erzeugen können. Dazu enthält die GTU-Simulation das Modell einer vollständigen TTC-Partition, die mit Hilfe eines CTP-Emulators gesteuert werden kann. Das enthaltene LTU-Modell mit CTP-Emulator, sowie die Modelle für TTCvi und TTCex, imitieren Funktion und Verhalten der im Experiment eingesetzten Hardware und wurden von Stefan Kirsch entwickelt. Ausgehend von einer Sequenzbeschreibungsdatei ermöglichen sie das Senden originalgetreuer, wenn gewünscht auch fehlerhafter, Triggersequenzen.

7.6.1 Der CTP-Emulator

Um die Reaktion der GTU auf verschiedene Kombinationen von Triggersequenzen simulieren zu können, muss es möglich sein, eine Abfolge vordefinierter Trigger abzuspielen. Dabei entsteht jedoch das Problem, dass der Start einer Triggersequenz vom Busy-Signal der GTU beeinflusst wird. Die auftretenden Busy-Phasen a priori zu berechnen und in der zeitlichen Abfolge der Trigger zu beachten, ist insbesondere bei aktiviertem MEB äußerst aufwendig und macht die Erstellung solcher Sequenzen sehr unflexibel. Um dieses Problem zu umgehen, enthält das Modell der LTU einen CTP-Emulator, der vordefinierte Triggersequenzen unter Berücksichtigung eines eingehenden Busy-Signals abspielen kann. Das Zeitverhalten innerhalb einer Sequenz und welche Trigger eine Sequenz enthält ist in einer Triggersequenz-Beschreibung festgelegt. Der Startzeitpunkt einer Sequenz wird allerdings nur relativ zum Fallen des Busy-Signals der vorherigen Triggersequenz angegeben. Da auch im MEB-Betrieb jedem L0-Trigger eine Busy-Phase bis zum L1-Zeitpunkt folgt, ist sichergestellt, dass die Sequenzen sich nicht in verbotener Weise überschneiden. Beim Abarbeiten der Triggersequenzen beachtet der CTP-Emulator den aktuellen Wert des Busy-Signals und löst die Sequenzen entsprechend verzögert aus. Die zweite Sequenz wird vom CTP-Emulator erst dann ausgelöst, wenn nach der fallenden Flanke des GTU-Busy Signals die Zeit t_{start} vergangen ist. Um auch fehlerhafte Triggersequenzen auslösen bzw.

das Zeitverhalten vollständig festzulegen zu können, kann in der Definition der Triggersequenzen vorgegeben werden, das Busy-Signal zu ignorieren. Dann wird die Triggersequenz um t_{start} verzögert nach dem Start der vorherigen Sequenz ausgelöst.

Die Definition der Triggersequenzen erfolgt in einer XML-Datei. Am Anfang der Definition können Standardwerte für alle Parameter angegeben werden. Diese werden angewendet, wenn der entsprechende Parameter in einer Triggersequenz nicht anderweitig definiert ist. Anschließend folgt eine Liste von Triggersequenzen. Für jede Triggersequenz kann definiert werden, welche Stufen die Sequenz enthält, wie oft sie wiederholt werden soll und, ob der Busy-Status für den Start beachtet werden soll. Innerhalb einer Sequenz kann neben dem Zeitverhalten der einzelnen Trigger auch der Inhalt der Triggernachrichten definiert werden. Die Event-ID einer Triggersequenz, die aufgrund des nicht voll bestimmten Zeitverhaltens nicht bekannt ist, muss hingegen nicht definiert werden. Wird die Event-ID auf Null gesetzt, fügt der CTP-Emulator die, zur Triggersequenz gehörende, Event-ID automatisch in die L2-Nachricht ein. Die Definitionen einer XML-Datei werden in ein, vom CTP-Emulator lesbares, Format übersetzt und von der Testbench in den Speicher des CTP-Emulators geladen.

Mit den verfügbaren Wahlmöglichkeiten kann jede beliebige Abfolge von Triggern erzeugt werden. Die Triggersequenzen lassen sich so exakt für die zu simulierende Situation anpassen. Für Fälle, in denen ein vorher bekannter Zeitablauf über mehrere Triggersequenzen simuliert werden soll, kann das Busy-Signal ignoriert und alle Zeiten absolut angegeben werden. Sollen die Triggersequenzen möglichst realitätsnah ablaufen, können die die Startzeitpunkte aufeinander folgender Triggersequenzen zufällig exponentialverteilt werden. Die Verteilung der L0-Triggers entspricht dann wie bei realen Kollisionsereignissen einer Zufallsverteilung.

7.6.2 Das TTC-Partition-Modell

Der CTP-Emulator ist Teil des LTU-Simulationsmodells, das zusammen mit weiteren Modellen eine vollständige TTC-Partition nachbildet. Dadurch ist es möglich, die Triggersequenzen wie im realen System als TTC-Signal an die GTU zu senden. Der Aufbau des Modells ist in Abbildung 7.3 dargestellt. Neben der LTU enthält das Modell eine TTCvi und eine TTCex-Einheit, welche die Codierung des TTC-Signals übernehmen. Spezielle Eigenschaften des TTC-Systems, wie die durch L1-Nachrichten verzögerte Übertragung von L2-Nachrichten, sind somit auch in der Simulation vorhanden. Weiterer Teil des Modells ist ein Event-ID-Zähler aus dessen Wert die, in der L2-Nachricht übertragene, Event-ID erzeugt wird. Bei jedem neuen Orbit wird, analog zum realen TTC-System, ein BC-Reset gesendet.

Im TTC-Partition-Modells lässt sich angeben, welche Sequenzdatei geladen werden soll, und wie oft die vollständige Sequenzliste zu wiederholen ist. Des Weiteren verfügt das Modell über Ausgänge für A- und B-Kanal des TTC-Signals sowie den zugehörigen LHC-Takt und einen Eingang für das Busy-Signal.

7.7 System-Konfiguration in der Simulation

Im realen Betrieb durchläuft die GTU bei jedem Neustart eine aufwendige Start-up- und Konfigurationsphase. Dabei werden im FPGA unter anderem das Backplane-Interface und die Verbindung zum SRAM kalibriert. Außerdem werden die benötigten Parameter durch einen PowerPC in die Konfigurationsregister geschrieben. Eine detailgetreue Simulation dieser Konfigurationsprozesse ist dementsprechend mit der Simulation des PPC-Systems verbunden und macht die Simulation extrem komplex. Alleine die Zeit, die benötigt würde, um alle nötigen Konfigurationen durchzuführen, läge im Bereich von mehreren Stunden.

In der Simulation sind interne Register und Signale des FPGAs direkt zugänglich. ModelSim bietet dafür unter Anderem das *force*-Kommando, mit dem sich jedem Signal beliebige Werte zuweisen lassen. Dieses Kommando kann genutzt werden, um die nötigen Konfigurationen vorzunehmen und die Startsequenz durch Manipulation einzelner Signale zu verkürzen. Dabei müssen mehrere Kommandos jeweils in einer bestimmten Reihenfolge ausgeführt werden. Ein oft gewählter Ansatz ist, die Kommandos in Form einer Liste auszuführen. Dieser Ansatz ist für die GTU-Simulation nicht ausreichend. In vielen Fällen wären Modifikationen an der Kommandoliste erforderlich, zum Beispiel wenn sich Simulationseinstellungen wie die TMU-Maske ändern oder eine andere Systemkonfiguration benötigt wird. Da viele Kommandos zeitlich voneinander abhängig sind oder andere Kommandos bedingen, sind manuelle Änderungen an der Liste aufwendig und fehleranfällig.

Der für die GTU-Simulation gewählte Ansatz nutzt die, von ModelSim bereitgestellte, Unterstützung für TCL-Skripte. Die benötigten Einzelkommandos werden von einem Konfigurationsskript ausgeführt. Voneinander abhängige Kommandos können blockweise aktiviert oder deaktiviert werden. Welche Kommandos benötigt werden und welche Werte zu setzten sind, wird dabei aus den Simulationseinstellungen in der Testbench abgeleitet. So wird vermieden, dass Konfiguration und gewählte Simulationseinstellungen von einander abweichen. Der Vorgang soll am nachfolgenden Beispiel verdeutlicht werden: Die in der Testbench gesetzte TMU-Maske wählt aus, welche TMUs simuliert werden. Entsprechend wird die SMU vom Konfigurationsskript mit der passenden TMU-Maske konfiguriert. Des Weiteren werden die Konfigurationskommandos, welche die TMUs betreffen, nur für in der Simulation vorhandene TMUs ausgeführt.

Die Verwendung des force-Kommandos ermöglicht es, alle nötigen Konfigurationen der FPGA-Designs vorzunehmen, ohne das PPC-System simulieren zu müssen. Durch die Verwendung eines TCL-Skripts zur Konfiguration kann die Auswahl verschiedener Simulationseinstellungen weitgehend automatisch erfolgen. Änderungen sind nur noch an einer Stelle notwendig und Fehlerquellen werden minimiert.

7.7.1 Simulation des PPC-Systems

Das PPC-System der GTU besteht aus zwei als HardIP im FPGA vorhandenen PPC-Kernen und zusätzlichen in Logik implementierten Komponenten wie das Bussystem und Speicherkontroller. Für die Simulation dieser HardIP-Blöcke existieren keine Modelle in

den Standard-Simulationsbibliotheken. Stattdessen bietet Xilinx separate Simulationsmodelle an. Die so genannten *SmartModels* werden mittels des *SWIFT-Interfaces* des Simulators in die Simulation eingebunden.

Für die Simulation des PPC-Systems werden sowohl die SmartModels für die PPC-Kerne, als auch Modelle für die zusätzlich instantiierten Komponenten benötigt. Die Möglichkeit, Modelle der verwendeten Komponenten zu erzeugen, ist in den Implementationsprozess integriert. Auch der auszuführende Programmcode kann zum Simulationsmodell hinzugefügt werden. Mit den SmartModels ist die Simulation des High-Level-PowerPC-Kernes der GTU möglich. Der zweite PPC-Kern, dessen Funktionen in der Simulation von größerem Interesse sind, kann aufgrund von Einschränkungen der vorhandenen SmartModels gegenwärtig nicht simuliert werden. Aus Performanzgründen wird der PLB-Bus des zweiten PPC-Kerns in der GTU nicht verwendet. Abweichend von der realen Hardware erfordert das SmartModel der PPC-Kerne jedoch aktive Komponenten am PLB-Bus, um zu arbeiten. Das PPC-System der GTU ist daher in der gewählten Ausbaustufe mit SmartModels nicht vollständig simulierbar. Beginnend ab Version 10.1 führt Xilinx mit den so genannten *SecureIP*-Modellen eine weitere Möglichkeit zur Simulation von HardIP-Komponenten ein. Dabei handelt es sich um verschlüsselte Verilog-Modelle, die vom Simulator direkt verarbeitet werden können. Erst nach einer aufwendigen Portierung des GTU-Projekts auf eine entsprechende Software-Version kann untersucht werden, ob die neuen SecureIP-Modelle eine vollständige Simulation des PPC-Systems erlauben.

7.8 Fazit und Ausblick

Mit der vorgestellten GTU-Simulation steht ein Werkzeug zur Verfügung, welches die Entwicklung und Verifizierung neuer Komponenten für das GTU-Design in effizienter Weise ermöglicht. Dabei kann durch die Anpassbarkeit der Simulation meist auf die aufwendige Erstellung separater Testbenches verzichtet werden. Die Reduzierung der simulierten GTU-Komponenten auf verschiedene Sätze, wie zum Beispiel eines Segments, ermöglichen Simulationen mit akzeptablen Simulationslaufzeiten auf aktuellen Rechnersystemen. Die vorgesehenen Einstellmöglichkeiten erlauben eine einfache Anpassung an das zu untersuchende Problem. Die Automatisierung der Systemkonfiguration vermeidet dabei Fehler durch inkonsistente Einstellungen.

Die Simulation von Verhaltensbeschreibungen unkritischer Designkomponenten zusammen mit voll implementierten Gate-Level-Komponenten eröffnet die Möglichkeit, auch in der Verhaltensbeschreibung nicht reproduzierbare Fehler mit vergleichsweise niedrigem Simulationsaufwand zu untersuchen. Die Leistungsfähigkeit der Simulation hat sich bereits bei aktuellen Entwicklungen gezeigt. Dabei konnte unter Anderem mit Hilfe einer gemischten Verhaltens- und Gate-Level-Simulation ein kritischer Implementierungfehler der Synthese-Software in den Tracking-Einheiten gefunden und dann in einfacher Weise umgangen werden.

Beliebige Triggersequenzen erzeugen zu können, eröffnet vielfältige Testmöglichkeiten. Vor allem im Bezug auf Multi-Event Buffering kann so die korrekte Funktion der Designs für

alle denkbaren Situationen überprüft werden. Die gezielte Erzeugung fehlerhafter Triggersequenzen lässt dabei weiter reichende Tests als in der realen Hardware zu. Treten im realen System Probleme auf, können die dort aufgezeichneten Triggersequenzen taktgenau nachgestellt werden, um in der Simulation die Ursache des Problems zu finden.

Die Simulationsmodelle der TTC-Partition und des enthaltenen CTP-Emulators sind voll synthetisierbar. Es ist also denkbar, die Komponenten in das SMU-Design zu integrieren. Dadurch könnten in Testaufbauten außerhalb des Experiments auch ohne externe Trigger-Infrastruktur weitaus komplexere Tests als bisher ausgeführt werden. Zudem ließen sich in realer Hardware exakt die gleichen Triggersequenzen wie in der Simulation testen und umgekehrt.

Die vollständige Automatisierung der Simulation erlaubt es, vordefinierte Tests in einem an das Nightly-Build-System angeschlossenen Simulationsprozess regelmäßig durchzuführen. Erste Tests sind implementiert und belegen, dass das Verfahren korrekt funktioniert. Weitere Tests können in Zukunft leicht durch Variieren der Eingangsdaten und Triggersequenzen erstellt und hinzugefügt werden.

8 Zusammenfassung

Im Rahmen des ALICE-Experiments am CERN sollen Schwerionenkollisionen untersucht werden, um Aufschluss über die Eigenschaften von Quark-Gluon-Plasmen zu bekommen. Die von den Detektoren des ALICE-Experiments erzeugten Rohdatenmengen können nicht in Echtzeit verarbeitet und gespeichert werden. Deshalb dient ein hierarchisches Triggersystem der Auswahl viel versprechender Ereignisse. Der zentrale Triggerprozessor (CTP) formt aus den Triggerbeiträgen der Detektoren eine globale Triggerentscheidung und steuert so die Aufnahme von Ereignissen. Der Triggerprozessor des Transition Radiation Detector (TRD), die Global Tracking Unit (GTU), liefert den TRD-Beitrag zur Level-1-Entscheidung des Triggersystems. Neben ihrer Aufgabe als Triggerprozessor ist die GTU ein essentieller Teil der Auslesekette des TRD. Ereignisdaten werden vom Detektor empfangen und entsprechend den Triggerentscheidungen des CTP zwischengespeichert und anschließend an das zentrale Datenaufnahmesystem gesendet. Jedes Segment der GTU verfügt dafür über eine zentrale Segmentsteuerung, welche die Steuersignale des CTP interpretiert und alle internen Abläufe während der Aufnahme von Ereignissen steuert.

In dieser Arbeit wird untersucht, wie sich die auslesebedingte Totzeit des TRD mit der Methode des Multi-Event Buffering (MEB) verringern lässt. Es wird gezeigt, dass für typische Betriebsbedingungen eine Reduktion der Totzeit von 20 % auf 4 % erreicht werden kann. Ein Konzept und eine konkrete Implementierung einer GTU-Segmentsteuerung werden vorgestellt. Die Verwaltung mehrerer, im Puffer gespeicherter Ereignisse stellt hohe Anforderungen an eine Segmentsteuerung. Die Verarbeitung der Triggerentscheidungen und Auslösen der Steuersignale muss ohne große Verzögerung erfolgen, um den Echtzeitanforderungen des Experiments gerecht zu werden. Im einem komplexen System wie ALICE kommt es unvermeidlich zu Fehlern. Die Segmentsteuerung muss daher robust gegenüber fehlerhaften Eingangssignalen gestaltet sein. Diese dürfen nicht dazu führen, dass die Segmentsteuerung inkorrekt arbeitet und falsche Ereignisse gesendet werden. Diese Problemstellung lässt sich durch die Aufteilung der Segmentsteuerung auf Hardware- und Software-Komponenten in effizienter Weise lösen. Die schnelle Triggerverarbeitung und Fehlererkennung erfolgt in spezialisierten Hardwarekomponenten. Die detaillierte Fehleranalyse kann mit Hilfe des eingebetteten Prozessorsystems in Software erfolgen. Die Fehleranalyse in Software ist flexibler als in Hardware und ermöglicht eine ressourcensparende Implementierung. Die Bearbeitung in Hardware ist in zwei unabhängig arbeitende Komponenten aufgeteilt um Triggerentscheidungen zyklisch verarbeiten zu können. Die Puffertiefe ist konzeptionell nicht begrenzt, sondern hängt lediglich vom der Segmentsteuerung zur Verfügung stehenden Speicherplatz ab. In der vorliegenden Implementierung werden bei vorgesehenem minimalem Speicherbedarf bis zu 512 Plätze im Puffer unterstützt. Die Implementierung der Fehleranalyse in Software steht noch aus.

Eine für die Entwicklung vom MEB unverzichtbare HDL-Simulation der GTU ist mit Hilfe des im Rahmen dieser Arbeit entwickelten Simulationsframeworks und der erstellten GTU-Simulation möglich. Die GTU-Simulation enthält Modelle für alle Komponenten der GTU und bietet die Möglichkeit, alle, für die GTU relevanten Eingangsdaten zu erzeugen. Die Simulation ist modular aufgebaut und kann leicht auf das jeweilige Testszenario angepasst werden. Dabei ist frei wählbar, welche Komponenten der GTU simuliert werden, um den Simulationsumfang auf das notwendige Maß zu reduzieren. Ebenso können beliebige Kombinationen von Eingangsdaten gewählt werden. Verschiedene Generatoren für den Datenstrom des Detektors stehen zur Verfügung. Mit Hilfe des CTP-Emulators können beliebige Triggersequenzen erzeugt werden. Mit diesen Möglichkeiten lassen sich alle Komponenten des GTU-Designs ausführlich testen und verifizieren.

Alle Schritte, um ein FPGA-Design zu simulieren, sind im entwickelten Simulationsframework automatisiert. Die Simulations-Konfigurationen mit allen benötigten Informationen zum Ausführen der Simulation, sind in Projektdateien definiert. Erstellte Simulationen und darin definierte Tests können voll automatisiert durchgeführt werden. Dies eröffnet die Möglichkeit, bei Designänderungen alle Funktionen ohne zusätzlichen Aufwand vordefinierten Testprozeduren zu unterziehen. Das Simulationsframework unterstützt die Erstellung von Simulationen auf allen Detailebenen des FPGA-Implementationsprozesses. Es besteht dabei die Möglichkeit, Gate-Level-Beschreibungen des gesamten FPGA-Designs oder einzelner Komponenten gemischt mit Verhaltensbeschreibungen zu simulieren. Die Gate-Level-Simulation kann zusätzlich wahlweise mit Timing-Backannotation durchgeführt werden. Das Simulationsframework ist weitgehend generisch gestaltet und kann so auch für andere HDL-Entwicklungen eingesetzt werden. Die weitreichenden Konfigurationsmöglichkeiten der GTU-Simulation und des Simulationsframeworks erlauben es, den Zeitbedarf der Simulation so weit zu verringern, dass auch Simulationen mehrerer Ereignisse möglich sind. Damit steht ein effektives Werkzeug für weitere Entwicklung im GTU-Projekt zur Verfügung.

Danksagung Ich danke besonders herzlich meinem betreuenden Professor, Herrn Prof. Dr. Volker Lindenstruth, für die Möglichkeit an diesem Projekt mitzuarbeiten und mehrere Aufenthalte am CERN zu realisieren. Ebenso gilt besonderer Dank meinen betreuenden Doktoranden Felix Rettig und Stefan Kirsch, die mich stets in allen Belangen unterstützt und gefördert haben. Ihre Anregungen haben mich ermutigt, neue Wege zu gehen und mein Wissen zu erweitern. Ich danke Dr. Jan de Cuveland für unzählige aufschlussreiche Diskussionen und hilfreiche Ratschläge während meiner Zeit in der Arbeitsgruppe. Mein Dank gilt auch den anderen Mitgliedern der Arbeitsgruppe, insbesondere Florian Painke und Dr. Venelin Angelov, die immer ein offenes Ohr für Fragen und Probleme hatten. Von ihnen allen habe ich viel gelernt. Nicht vergessen möchte ich, allen hier nicht namentlich Genannten zu danken, die mich während meiner Arbeit und bei der Korrektur tatkräftig unterstützt haben. Abschließend danke ich meinen Eltern, die mir ein so unbeschwertes Physikstudium ermöglicht haben.

Akronymverzeichnis

ACORDE	ALICE Cosmic Ray Detector
ADC	Analog to Digital Converter
ALICE	A Large Ion Collider Experiment
ASIC	Application Specific Integrated Circuit
ATLAS	A Toroidal LHC ApparatuS
BC	Bunch Counter
BRAM	Block-RAM
CDH	Common Data Header
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
CPU	Central Processing Unit
CTP	Central Trigger Processor
DAQ	Data Acquisition
DCM	Digital Clock Manager
DCS	Detector Control System
DDL	Detector Data Link
EEPROM	Electrically Erasable Programmable Read-Only Memory
8b/10b	8 Bit/10Bit-Code
EMCal	Electromagnetic Calorimeter
EoD	End-of-Data
FEE	Front-End Electronics
FIFO	First In – First Out
FMD	Forward Multiplicity Detector
FPGA	Field Programmable Gate Array
GDC	Global Data Concentrator

GTU	Global Tracking Unit
HDL	Hardware Description Language
HLT	High-Level Trigger
HMPID	High Momentum Particle Identification
I/O	Input/Output
ITS	Inner Tracking System
I²C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
LDC	Local Data Concentrator
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
LHCf	Large Hadron Collider forward
L1	Level-1
LTU	Local Trigger Unit
L2	Level-2
L2a	L2-accept
L2r	L2-reject
LVDS	Low Voltage Differential Signal
L0	Level-0
MCM	Multi Chip Module
MEB	Multi-Event Buffering
MGT	Multi-Gigabit Serial Transceiver
MoEDAL	Monopole and Exotics Detector at the LHC
OC	Orbit Counter
PASA	Preamplifier/Shaper
PHOS	Photon Spectrometer
PLB	Prozessor Logic Bus
PMD	Photon Multiplicity Detector
PPC	PowerPC
QCD	Quantenchromodynamik

QGP	Quark-Gluon-Plasma
RHIC	Relativistic Heavy Ion Collider
RISC	Reduced Instruction Set Computer
SDF	Standard Delay Format
SD-Karte	Secure Digital Speicherkarte
SDRAM	Synchronous Dynamic Random Access Memory
SFP	Small Form-factor Pluggable
SIU	Source Interface Unit
SMU	Supermodule Unit
SoD	Start-of-Data
SOPC	System On a Programmable Chip
SRAM	Static Random Access Memory
TCL	Tool Command Language
TGU	Trigger Generation Unit
TMU	Track Matching Unit
TOF	Time of Flight
TOTEM	Total Elastic and Diffractive Cross Section Measurement
TPC	Time Projection Chamber
TRAP	Tracklet Processor
TRD	Transition Radiation Detector
TTC	Timing Trigger and Control
TTCex	TTC Encoder Transmitter Board
TTCrx	TTC Receiver ASIC
TTCvi	TTC VMEbus Interface Board
UART	Universal Asynchronous Receiver/Transmitter
VHDL	VHSIC (Very-High-Speed Integrated Circuit) Hardware Description Language
XML	Extensible Markup Language
ZDC	Zero Degree Calorimeter

Literaturverzeichnis

[1] ALICE Collaboration. *Technical Proposal for A Large Ion Collider Experiment at the CERN LHC*. CERN/LHCC 95-71. CERN/LHCC, Geneva, December 1995. ISBN 92-9083-077-8.

[2] ALICE Collaboration. *Technical Design Report of the High Momentum Particle Identification Detector*. CERN/LHCC 98-19. CERN/LHCC, Geneva, August 1998. ISBN 92-9083-134-0.

[3] ALICE Collaboration. *Technical Design Report of the Dimuon Forward Spectrometer*. CERN/LHCC 99–22. CERN/LHCC, Geneva, 1999. ISBN 92-9083-148-0.

[4] ALICE Collaboration. *Technical Design Report of the Inner Tracking System (ITS)*. CERN/LHCC 99-12. CERN/LHCC, Geneva, June 1999. ISBN 92-9083-144-8.

[5] ALICE Collaboration. *Technical Design Report of the Photon Multiplicity Detector (PMD)*. CERN/LHCC 99–32. CERN/LHCC, Geneva, 1999. ISBN 92-9083-153-7.

[6] ALICE Collaboration. *Technical Design Report of the Photon Spectrometer (PHOS)*. CERN/LHCC 99-4. CERN/LHCC, Geneva, March 1999. ISBN 92-9083-138-3.

[7] ALICE Collaboration. *Technical Design Report of the Zero Degree Calorimeter (ZDC)*. CERN/LHCC 99–5. CERN/LHCC, Geneva, 1999. ISBN 92-9083-139-1.

[8] ALICE Collaboration. *Technical Design Report of the Time of Flight System (TOF)*. CERN/LHCC 2000-12. CERN/LHCC, Geneva, February 2000.

[9] ALICE Collaboration. *Technical Design Report of the Time Projection Chamber*. CERN/LHCC 2000-001. CERN/LHCC, Geneva, January 2000. ISBN 92-9083-155-3.

[10] ALICE Collaboration. *Technical Design Report of the Transition Radiation Detector*. CERN/LHCC 2001-021. CERN/LHCC, Geneva, 2001. ISBN 92-9083-184-7.

[11] ALICE Collaboration. *Technical Design Report of the Trigger, Data Acquisition, High-level Trigger and Control System*. CERN/LHCC 2003-062. CERN/LHCC, Geneva, 2004. ISBN 92-9083-217-7.

[12] ALICE Collaboration. *Technical Design Report on Forward Detectors: FMD, T0 and V0*. CERN/LHCC 2004-025. CERN/LHCC, Geneva, September 2004. ISBN 92-9083-229-0.

[13] ALICE Collaboration. *Electromagnetic Calorimeter Technical Design Report*. CERN-LHCC 2008-014. CERN/LHCC, Geneva, 2008. ISBN 978-92-9083-320-8.

[14] Allen, A. O. *Probability, statistics, and queueing theory.* Computer science and scientific computing. Acad. Press, Boston [u.a.], second edition, 1990. ISBN 0-12-051051-0 ; 978-0-12-051051-1.

[15] Brüning, O. S., Collier, P., Lebrun, P., Myers, S., Ostojic, R., Poole, J., and Proudlock, P. *LHC Design Report.* CERN, Geneva, 2004.

[16] CERN. Web Site.
URL `http://www.cern.ch`

[17] de Cuveland, J. *Entwicklung der globalen Spurrekonstruktionseinheit für den ALICE-Übergangsstrahlungsdetektor am LHC (CERN).* Diplomarbeit, Universität Heidelberg, Kirchhoff-Institut für Physik, Heidelberg, 2003.

[18] de Cuveland, J. *A Track Reconstructing Low-latency Trigger Processor for High-energy Physics.* Dissertation, Universität Heidelberg, Kirchhoff-Institut für Physik, Heidelberg, 2009.

[19] de Cuveland, J., Rettig, F., and Kirsch, S. ALICE TRD DAQ Data Format. Internal Note, June 2009.

[20] Divià, R., Jovanovic, P., and Vande Vyvre, P. *Data Format over the ALICE DDL.* Internal Note ALICE-INT-2002-010, European Organization for Nuclear Research, March 2007.

[21] Gerlach, T. *Development of an Embedded Linux System for the Global Tracking Unit of the ALICE TRD at the LHC (CERN).* Diplomarbeit, Universität Heidelberg, Kirchhoff-Institut für Physik, Heidelberg, September 2008.

[22] Heath, G. Dead time due to trigger processing in a data acquisition system with multiple event buffering. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 278(2):431 – 435, 1989. ISSN 0168-9002. doi:DOI:10.1016/0168-9002(89)90861-9.

[23] Heinz, U. and Jacob, M. Evidence for a New State of Matter: An Assessment of the Results from the CERN Lead Beam Programme.
URL `http://arxiv.org/abs/nucl-th/0002042v1`

[24] Kirsch, S. *Development of the Supermodule Unit for the ALICE Transition Radiation Detector at the LHC (CERN).* Diplomarbeit, Universität Heidelberg, Kirchhoff-Institut für Physik, Heidelberg, 2007.

[25] Kirsch, S., Rettig, F., Hutter, D., de Cuveland, J., Angelov, V., and Lindenstruth, V. An FPGA-based High-speed, Low-latency Processing System for High-energy Physics. *International Conference on Field Programmable Logic and Applications (FPL) 2010*, September 2010.

[26] Mentor Graphics Corporation. *ModelSim SE User's Manual*, software version 6.5d edition.

[27] Relativistic Heavy Ion Collider (RHIC) Experimental Collaborations. *Hunting the Quark Gluon Plasma – Results from the First 3 Years at RHIC*. BNL-73847-2005. Brookhaven National Laboratory, Upton, NY, April 2005. Formal Report.

[28] Rettig, F. *Entwicklung der optischen Auslesekette für den ALICE-Übergangsstrahlungsdetektor am LHC (CERN)*. Diplomarbeit, Universität Heidelberg, Kirchhoff-Institut für Physik, Heidelberg, 2007.

[29] Richter, M., Alme, J., Alt, T., Bablok, S., Campagnolo, R., Frankenfeld, U., Gutierrez, C., Keidel, R., Kofler, C., Krawutschke, T., et al. The Control System for the Front-end Electronics of the ALICE Time Projection Chamber. *Nuclear Science, IEEE Transactions on*, 53(3):980–985, 2006. ISSN 0018-9499. doi:10.1109/TNS.2006.874726.

[30] Rubin, G. and Soós, C. *ALICE Detector Data Link – Hardware Guide for the Front-end Designers*. Internal note, European Organization for Nuclear Research, September 2007.

[31] Schuh, M. *Entwicklung und Implementierung eines Steuersystems für die Trigger- und Datenausleselogik des ALICE-Übergangsstrahlungsdetektors am LHC (CERN)*. Diplomarbeit, Universität Heidelberg, Kirchhoff-Institut für Physik, Heidelberg, 2007.

[32] Xilinx, Inc. *Virtex-4 FPGA User Guide, UG070 (v2.4)*, April 2008.

[33] Xilinx, Inc. *Xilinx Command Line Tools User Guide, UG628 (v 11.4)*, 2009.

[34] Xilinx, Inc. *Xilinx Synthesis and Simulation Design Guide, UG626 (v 11.4)*, 2009.