

Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Diplomarbeit
im Studiengang Physik

vorgelegt von

Marcel Schuh
aus Bad Friedrichshall

September 2007

**Entwicklung und Implementierung
eines Steuersystems
für die Trigger- und Datenausleselogik des
ALICE-Übergangsstrahlungsdetektors
am LHC (CERN)**

Marcel Schuh

Die Diplomarbeit wurde ausgeführt am

Kirchhoff-Institut für Physik

unter der Betreuung von

Herrn Prof. Dr. Volker Lindenstruth

Für Ingrid

Entwicklung und Implementierung eines Steuersystems für die Trigger- und Datenausleselogik des ALICE-Übergangsstrahlungsdetektors am LHC (CERN):

Im Experiment ALICE werden Schwerionenkollisionen untersucht, bei denen enorme Datenmengen produziert werden. Deren Transport und Verarbeitung stellt eine gewaltige Herausforderung dar. Der Übergangsstrahlungsdetektor ist ein innovativer Detektor, dessen Ausleseelektronik weitgehend digital ist. Diese bildet ein komplexes System mit einer Vielzahl von Mess- und Steuerparametern, die konfiguriert und überprüft werden müssen.

Die vorliegende Arbeit beschäftigt sich mit der Überwachung und Steuerung der Trigger- und Datenausleselogik, in der alle im Experiment anfallenden Daten des Übergangsstrahlungsdetektors zusammenkommen und weiter verarbeitet werden. Diese besteht aus 109 Untereinheiten, in denen jeweils ein FPGA mit zwei PowerPCs arbeitet. Sie befindet sich in einem Bereich, der während der Laufzeit des Experiments nicht zugänglich ist. Daher wird ein hochverfügbares Fernwartungssystem entwickelt, das alle Sensoren ausliest, Systemparameter ändert und bei Bedarf die FPGAs neu programmiert. Die Diplomarbeit gibt einen Überblick über die verschiedenen implementierten Schichten des Systems vom Sensor über Eingebettete Systeme und verschiedene Protokolle bis hin zur Überwachung und Visualisierung im Experimentsteuersystem von ALICE.

Development and implementation of a monitoring and control system for the Trigger and Readout Electronics in the ALICE Transition Radiation Detector at the LHC (CERN):

Heavy ion collisions are analyzed in the experiment ALICE which a vast amount of data is produced. Transport and processing of these data is a huge challenge. The Transition Radiation Detector is an innovative detector. Its electronics consists largely of digital electronics. It is an extremely complex system featuring many measure and control parameters which must be configured and checked.

This work deals with the monitoring and controlling of the Trigger and Readout Electronics in which all experimental data of the Transition Radiation Detector is collected and processed. This unit consists of 109 boards each equipped with one FPGA and two PowerPCs in each. There is no access to the installation area of this unit while the experiment is carried out. A highly available system is developed which supports remote access to readout sensors, configure parameters and program the FPGAs. This thesis gives an overview over all layers of the implemented system ranging from sensors over embedded systems and several records to the supervision and visualisation of the experiment control system in ALICE.

Inhaltsverzeichnis

1	Einleitung	15
2	Experiment	19
2.1	Der Large Hadron Collider	19
2.2	ALICE	21
2.2.1	Die Detektoren	22
2.2.2	Die Experimentsteuerung	23
2.3	Der Übergangsstrahlungsdetektor (TRD)	28
2.3.1	Funktionsprinzip und Aufbau	28
2.3.2	Die Front-End- und Readout-Elektronik des TRD	29
2.3.3	Das Detektor Control System im TRD	30
3	Die Global Tracking Unit	33
3.1	Aufbau der GTU-Boards	34
3.1.1	Backplanes, Netzteil und Kühlung	35
3.2	TMU	38
3.2.1	Aufbau der SFP-Modul-Schnittstelle	39
3.3	SMU und TGU	40
4	Das Embedded System in der GTU	43
4.1	Der Embedded-PowerPC-Processor PPC405	43
4.2	Hardware-Architektur	46
4.2.1	Schnittstellen und Bussysteme	47
4.2.2	IP-Cores	48
4.2.3	Externe Timer- und Interrupt-Controller	50
4.3	Software-Framework	51
4.3.1	Bootsequenz	51
4.3.2	Kommandozeileninterpreter	52
4.3.3	Interrupt LED-Steuerung	53
5	Die Schnittstelle DCS-Board	55
5.1	Hardware des DCS-Boards	55
5.2	Kommunikationsschnittstelle zur GTU	55
5.2.1	Hardware der Kommunikationsschnittstelle zur GTU	56
5.2.2	UART-Treiber für das DCS-Baord	58

5.2.3	Software	61
5.3	DIM-Server Software	63
5.3.1	Funktionsweise von DIM	63
5.3.2	Datenquellen und Auslesekanäle	64
5.3.3	Datenpunkte und Kommandokanäle	65
5.3.4	Programmfluss des DIM-Servers	66
5.3.5	Anforderungen und Probleme bei der Implementierung	67
5.3.6	Performance, Probleme und Erweiterungsmöglichkeiten	68
6	Visualisierung und Monitoring	69
6.1	Monitoring mit PVSS und JCOP-Framework	69
6.1.1	Aufbau von PVSS und JCOP	69
6.1.2	Datenbank und DIM-Schnittstelle	71
6.1.3	Panels	73
6.1.4	Stand der Implementierung	74
6.2	Einbindung der GTU in das DCS mit FSM	75
6.2.1	Die allgemeine Kontrollhierarchie der FSM	75
6.2.2	Die Kontrollhierarchie der FSM in der GTU	76
6.2.3	Stand der Implementierung	76
7	Der SD-Karten-Controller	79
7.1	Aufbau einer SD-Karte und ihrer Schnittstelle	80
7.1.1	SPI-Modus	81
7.2	Aufbau des Controllers	84
7.2.1	Implementierung	86
7.2.2	Treiber-Software für den PowerPC	88
7.2.3	Performance und Erweiterungsmöglichkeiten	88
8	Erste Tests am CERN	91
8.1	Messwerte und Kalibrierung	91
8.2	Reaktionszeiten des Systems	94
8.3	Ausblick	96
9	Zusammenfassung	97
A	DIM-PVSS Datenzuordnung	99
B	PVSS Panels	101
C	SD-Karte	103
	Literaturverzeichnis	107

Abbildungsverzeichnis

1.1	Phasendiagramm Kernmaterie	17
2.1	Large Hadron Collider	20
2.2	Detektoren in ALICE	22
2.3	Schematischer Aufbau ECS	24
2.4	Schichtenmodell DCS	25
2.5	PVSS FSM	26
2.6	Schichtenmodell DCS-Software	26
2.7	Baumstruktur DCS TRD	27
2.8	Mittlere Pulshöhen für Elektronen und Pionen	28
2.9	Pulshöhenverlauf über die Driftzeit	28
2.10	Aufbau des Detektors	29
2.11	Schema Datenpfad	30
2.12	Schema DCS TRD	31
2.13	DCS Field Layer GTU	31
2.14	Kommunikationspfad GTU	32
3.1	Schema GTU	33
3.2	Aufbau GTU-Board	34
3.3	Schema I ² C-Backplane	36
3.4	Fotographie GTU Crate	37
3.5	Fotographie TMU	38
3.6	Diagnostik-Schnittstelle SFP-Modul	39
3.7	Fotographie SMU	40
4.1	Aufbau PowerPC-Kern	44
4.2	PowerPC-Einheit	47
4.3	PowerPC im FPGA	47
4.4	Schematischer Aufbau OPB-Schnittstelle	48
4.5	LED-Matrix-Steuerung	49
4.6	Timingdiagramm OPB-IP-Core	50
4.7	Flussdiagramm PowerPC-Software	51
4.8	LED-Matrix-Memory	53
5.1	Schematischer Aufbau UART-Core im DCS-Board	56

5.2	Schematischer Aufbau UART-Kommunikationspfad	57
5.3	Screenshot GTUCOM	62
5.4	Schema DIM	64
5.5	Flussdiagramm DIM-Server	66
6.1	PVSS II Schichten	70
6.2	Screenshot PVSS PARA	72
6.3	PVSS Rack Monitor Panel	74
6.4	FSM Hierarchie GTU	77
6.5	FSM States GTU	77
7.1	Schematischer Aufbau SD-Karte	79
7.2	SPI-Kommando-Zyklus	81
7.3	Diagramm SD-Karte Init-Befehlsfolge	83
7.4	SPI-Lesezyklus	83
7.5	SPI-Schreibzyklus	83
7.6	Schematischer Aufbau SD-Karten-Controller	85
7.7	Flussdiagramm Initialisierung-State-Machine	86
7.8	Flussdiagramm Befehl-State-Machine	87
8.1	Plot Spannungverlauf über 24h	91
8.2	Stromverkabelung	92
8.3	Plot Eingangsspannungen auf Boards	92
8.4	Luftstrom im Rack	93
8.5	Plot Temperaturverteilung im Crate	93
8.6	Plot Temperaturhistogramm	94
8.7	Plot Temperaturverlauf FPGA Einschaltvorgang	95
8.8	Plot Temperaturverlauf FPGA Überhitzungstest	96
B.1	PVSS Panel Baum	102

Tabellenverzeichnis

1.1	Fundamentale Teilchen	16
1.2	Fundamentale Wechselwirkungen	16
2.1	Kenngößen LHC	21
5.1	UART Token	59
5.2	Datenpunkte GTU	66
6.1	PVSS Manager	70
6.2	Framework Pakete	71
7.1	Register SD-Karte	80
7.2	Pinbelegung SD-Karte	80
7.3	SD Token	82
7.4	Steuerregister SD-Karten-Controller	84
8.1	Schwellwerte Alarmhandling	93
8.2	Kommandoantwortzeiten	94
A.1	DIM-Datenpunkte GTU	99
A.2	Komplexe DIM-Datenpunkte	100
B.1	Liste Panels	101
C.1	Befehle SD-Karte	103
C.2	Längen Antworttoken	103
C.3	Bits Antworttoken	104
C.4	Belegung Schnittstellenregister SD-Karte	105
C.5	Codierung Response Selekt	106
C.6	Liste Entities	106

1 Einleitung

Eine Frage, mit der sich die Menschheit beschäftigt, ist der Aufbau der Natur. Von besonderem Interesse ist hierbei der Aufbau und die Struktur der Materie. Deren Bild ist im Laufe der Jahrtausende durch neue Erkenntnisse und Experimente präziser und detailreicher geworden. Galten in der Antike die Atome als die kleinsten Teilchen, so sind heute nach dem Standardmodell Atome aus noch kleineren Teilchen aufgebaut. Die Frage nach der Richtigkeit dieses Modells und nach Strukturen jenseits dieses Modells sind aktuelle Fragestellungen, mit denen sich viele Physiker beschäftigen. Die Hochenergiephysik leistet durch ihre Theorien und Experimente einen wesentlichen Beitrag hierfür.

Das Standardmodell beschreibt den Aufbau der Materie durch Elementarteilchen und deren Wechselwirkungen. Diese Theorie vereinigt drei der vier bisher bekannten fundamentalen Wechselwirkungen. Die elektromagnetische und schwache Wechselwirkung werden zur elektroschwachen vereinigt. Durch die Quantenchromodynamik (QCD) wird die starke Wechselwirkung beschrieben. Die Gravitation bleibt im Standardmodell unberücksichtigt.

Die Materie besteht nach diesem Modell aus zwei Gruppen fermionischer Teilchen¹, den Hadronen, die aus Quarks aufgebaut sind, und den Leptonen. Alle Elementarteilchen sind in der Tabelle 1.1 aufgeführt. Die Eichbosonen, welche in Tabelle 1.2 aufgelistet sind, vermitteln die fundamentalen Wechselwirkungen zwischen den Teilchen. Die Stärke der Wechselwirkung (Kopplung) zwischen Eichboson und Teilchen wird durch ihre Ladung (Teilcheneigenschaft) beschrieben. Jede Wechselwirkung hat eine eigenen Ladungstyp.

Leptonen unterliegen nur der schwachen und elektrischen Wechselwirkung und tragen daher auch nur eine elektrische (dipolare) und eine schwache (unipolare) Ladung. Hadronen unterliegen darüber hinaus der starken Wechselwirkung, deren Ladung als Farbe bezeichnet wird. Es gibt sechs verschiedene Farben: rot, grün, blau und die jeweilige Antifarbe. Die Addition von jeweils einer roten, grünen und blauen Farbladung ist farbneutral in Analogie zur Addition von Elementarfarben in der Optik.

Während die Stärke der elektromagnetischen und schwachen Wechselwirkung mit dem Abstand abnimmt, nimmt sie bei der starken Wechselwirkung mit größer werdendem

¹Teilchen, deren Eigenschaften durch die Fermi-Dirac-Statistik beschrieben werden.

	Quarks	Q/e	Leptonen	Q/e
1. Generation	d (down)	$-1/3$	Elektron-Neutrino ν_e	0
	u (up)	$+2/3$	Elektron e	-1
2. Generation	s (strange)	$-1/3$	Myon-Neutrino ν_μ	0
	c (charme)	$+2/3$	Myon μ	-1
3. Generation	b (bottom)	$-1/3$	Tau-Neutrino ν_τ	0
	t (top)	$+2/3$	Tau τ	-1

Tabelle 1.1: Im Standardmodell ist die uns bekannte Materie aus zwölf fundamentalen Teilchen und ihren Anti-Teilchen aufgebaut. Sie sind in zwei Klassen aufgeteilt, Quarks und Leptonen. Innerhalb der Klassen werden die Teilchen anhand ihrer Masse in drei Generationen eingeteilt.

Wechselwirkung	Austausch- teilchen	Masse	rel. Stärke	Reich- weite [m]
starke W.	Gluonen (g)	0	1	$\approx 10^{-15}$
elektromagn. W.	Photonen (γ)	$<10^{-16} \text{ eV}/c^2$	10^{-2}	∞
schwache W.	W^\pm -, Z^0 -Bosonen	$80/91 \text{ GeV}/c^2$	10^{-14}	$\approx 2 \cdot 10^{-18}$
Gravitation	(Gravitonen)	-	10^{-38}	∞

Tabelle 1.2: Die vier fundamentale Wechselwirkungen. [Hor04][Y+06]

Abstand zweier farbgeladener Teilchen zu. Dies führt dazu, dass nur farbneutrale Objekte beobachtet werden (*Confinement*), welche aus Quark-Triplets (*Baryonen*²) oder Duplets (*Mesonen*³) bestehen.

Die Vielfalt der Hadronen wird durch das Standardmodell sehr gut beschrieben und kann durch mehrere Experimente bestätigt werden. Andere grundlegende Fragen werden durch das Standardmodell jedoch nicht beschrieben, wie zum Beispiel die Natur der Masse. Der Higgs-Mechanismus versucht, die Entstehung der Masse durch die Wechselwirkung der Elementarteilchen mit dem Higgs-Feld zu erklären. Der Nachweis der Austauschbosonen des Higgs-Felds (Higgs-Bosonen) ist ein wichtiges Ziel der heutigen Forschung. Das Auffinden des Higgs-Teilchens würde das Standardmodell weiter gefestigt.

Die QCD sagt einen Phasenübergang voraus, in dem sich die Quarks und Gluonen frei bewegen können. Dieser Zustand wird als Quark-Gluonen-Plasma (QGP) bezeichnet. In Abbildung 1 ist das Phasendiagramm der Kernmaterie dargestellt. Wird die Dichte des Kern ($T = 0K$) auf das Zehnfache erhöht, so beginnen sich die in ihm enthaltenen Nukleonen zu überlappen und existieren nicht mehr als eigenständige Teilchen. In diesem Zustand können sich die Quarks und Gluonen frei im gesamten Kernvolumen bewegen.

²Baryonen bestehen aus drei Quarks, von denen jedes eine andere Farbladung trägt.

³Mesonen bestehen aus Quark-Antiquark-Paaren.

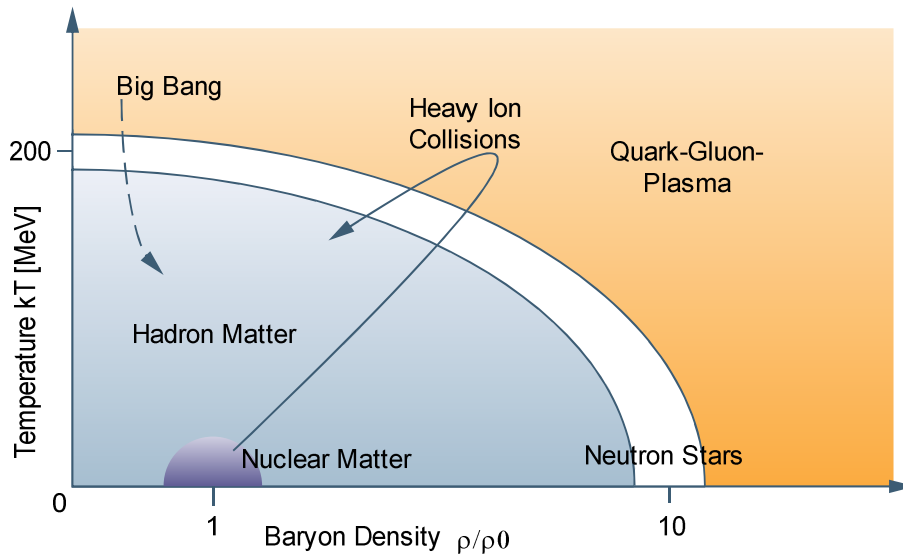


Abbildung 1.1: Phasendiagramm der Kernmaterie. Die normalen Kerne befinden sich bei Dichte ρ_0 und der Temperatur $T = 0$. Die Pfeile geben die Wege an, welche die Kerne bei verschiedenen Schwerionenreaktionen nehmen. Bei relativistischen Schwerionenstößen durchquert die Kernmaterie möglicherweise die Quark-Gluonen-Plasma-Phase. Der Pfeil links oben symbolisiert das Abkühlen des Universums zur Zeit $T \approx 1\mu s$ nach der Entstehung. Neutronensterne haben teilweise eine so hohe Dichte, dass QGP bei niedrigen Temperaturen entstehen kann. Nach [PRSZ04].

Wird die Temperatur erhöht, ohne die Nukleonendichte zu verändern, führt dies bei einem Temperaturäquivalent von 200 MeV dazu, dass Nukleon-Nukleon-Wechselwirkungen stattfinden können. Durch Pionenproduktion wird die Hadronendichte und somit die Stoßhäufigkeit zwischen ihnen so weit erhöht, dass eine feste Zuordnung der Quarks und Gluonen zu einem bestimmten Hadron nicht mehr möglich ist [PRSZ04].

Mit dem Nachweis des QGP wäre eine experimentelle Bestätigung für die momentane Vorstellung der Struktur stark wechselwirkender Materie gefunden. Hiermit könnte der Zustand des Universums zu einem sehr frühen Zeitpunkt seiner Entstehung simuliert werden.

Das Ziel des *A Large Ion Collider Experiment* (ALICE) am *Large Hadron Collider* (LHC) am CERN ist es, das Quark-Gluonen-Plasma nachzuweisen und zu untersuchen. In diesem Experiment werden Bleikerne mit annähernd Lichtgeschwindigkeit zur Kollision gebracht. Die Herausforderung hierbei besteht darin, die sehr vielen Teilchen, welche durch die Kollision entstehenden, detektieren zu können. Hierzu müssen die Detektoren eine sehr hohe Ortsauflösung besitzen, um die Teilchenspuren trennen zu können. Die Menge der anfallenden Rohdaten ist so groß, dass eine Online-Analyse der selben die heute verfügbare Rechenkapazität bei weitem übersteigt und auch das Speichern aller

Rohdaten logistisch nicht möglich ist. Es muss daher ein System verwendet werden, das zur Laufzeit einen Teil der Daten analysiert und daraufhin entscheidet, ob weitere Daten ausgelesen und gespeichert werden. Ein solches System wird *Trigger* genannt. Die Daten des *Transition Radiation Detector* (TRD) fließen in die Trigger-Entscheidung mit ein. Das Auslesen und die erste Analyse der Daten für eine Triggerentscheidung ist die Aufgabe der *Global Tracking Unit* (GTU). Sie ist ein hochkomplexes System, in der diese Aufgabe durch massive Parallelisierung und Einsatz modernster Digitalelektronik bewerkstelligt wird. In den 109 Untereinheiten kommen FPGAs, die zwei PowerPC-Kerne besitzen, für die Analyse der Daten und zum Einsatz.

Die vorliegende Arbeit beschäftigt sich mit der Überwachung und Steuerung der GTU. Das folgende Kapitel führt in das Experiment und die Konstruktion des TRD ein. Das dritte Kapitel erläutert den genauen Aufbau und die Funktion der GTU und gibt einen Überblick über die zu überwachenden Daten. Das Kapitel vier geht auf das Überwachungssystem und die Steuerung innerhalb der Untereinheiten der GTU ein, das durch ein „*System on a Chip*“ realisiert wird. Das fünfte Kapitel stellt die Implementierung der Schnittstelle zwischen den Einheiten der GTU und dem Überwachungssystem des Detektors vor. Mit der Visualisierung der Messdaten und der zentralen Steuerung der GTU beschäftigt sich Kapitel sechs. Im siebten Kapitel wird die Implementierung eines SD-Karten-Controllers für das Embedded System vorgestellt. In Kapitel acht werden die Ergebnisse der ersten Tests am CERN dargestellt. Abschließend fasst das neunte Kapitel die Ergebnisse der Arbeit zusammen.

2 Das Experiment

In der Nähe von Genf, an der Grenze zwischen Schweiz und Frankreich, befindet sich das Europäische Forschungszentrum für Teilchenphysik *CERN*¹. Mit zirka 2000 fest beschäftigten Mitarbeitern und etwa 6500 Gastwissenschaftlern aus über 80 Staaten ist es weltweit das größte Forschungszentrum dieser Art. An diesem sind neben der Universität Heidelberg noch 500 andere Universitäten beteiligt.

Momentan wird dort der *Large Hadron Collider* (LHC) fertig gestellt, der Protonen bzw. Bleiatomkerne fast auf Lichtgeschwindigkeit beschleunigt und mit einer Schwerpunktsenergie von 14 TeV bzw. 1150 TeV zur Kollision bringt. Er übertrifft damit alle bisherigen Beschleuniger wie Tevatron² und RHIC³ in Bezug auf Energie und Luminosität⁴ und wird daher auch als Beschleuniger der nächsten Generation bezeichnet. Der Bau des LHC ist 1994 beschlossen worden und die erste Strahlzeit ist für Frühjahr 2008 vorgesehen.

2.1 Der Large Hadron Collider

Der LHC hat einen Umfang von 27 km und wird im Tunnel des Vorgängerexperiments LEP⁵ aufgebaut, der im Mittel 100 m unter der Erdoberfläche verläuft. Die Teilchen durchlaufen ein System von Vorbeschleunigern, bevor sie in den Ring des LHC injiziert werden. Dort werden sie dann auf maximale Energie beschleunigt und auf Kollisionskurs gebracht.

Der Speicherring besteht aus 2 Röhren mit einem Abstand von 20 cm. Diese befinden sich innerhalb eines Kryostatsystems, das das System auf 1,8 K herunterkühlt. Um die Teilchenstrahlen zu fokussieren und auf der Kreisbahn zu halten, werden supraleitende Dipol- und Quadrupolmagnete verwendet. Die über 1.000 Dipolmagnete sind jeweils 13 m lang und erzeugen ein Magnetfeld von über 8 T.

¹Das Conseil Européen pour la Recherche Nucléaire wurde 1954 gegründet (vgl. [CERa])

²Proton-Proton-Beschleuniger mit 2 TeV am *Fermi National Accelerator Laboratory* (FERMILAB), Chicago (USA) (vgl. [Fer])

³*Relativistic Heavy Ion Collider* am *Brookhaven National Laboratory*, Brookhaven (USA) mit 200 GeV je Nukleon-Paar (vgl. [Bro])

⁴Energie und Luminosität sind zwei gebräuchliche Größen, um einen Beschleuniger zu charakterisieren. Die Energie ist die Schwerpunktsenergie bei der Kollision. Die Luminosität gibt an, wie dicht die Teilchen in einem Beschleuniger gepackt sind und wie häufig es zu Teilchenreaktionen kommt.

⁵Der *Large Electron-Positron Collider* war von 1989 bis 2000 in Betrieb.

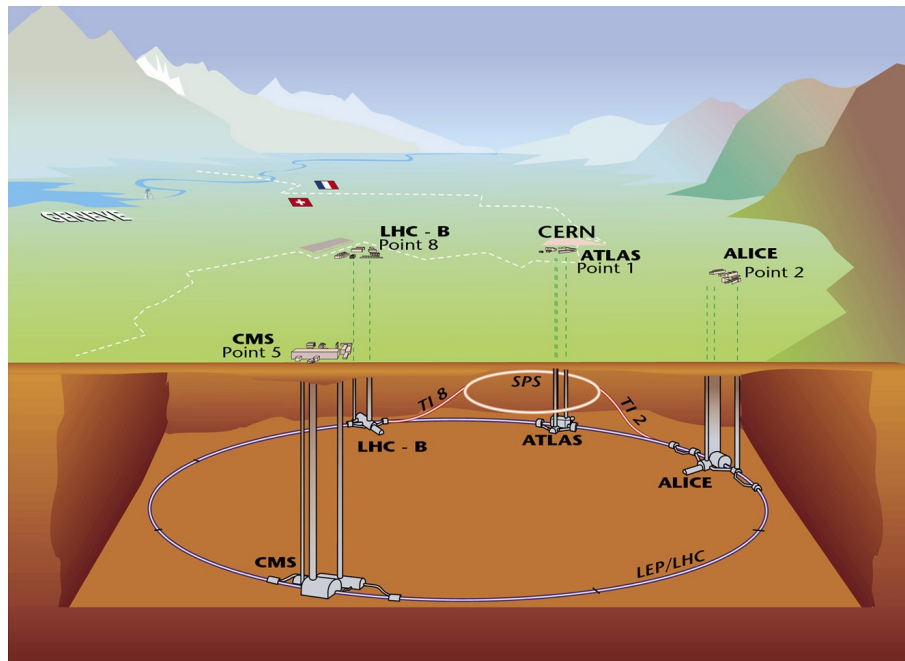


Abbildung 2.1: Der Large Hadron Collider am CERN

Es gibt insgesamt vier *Interaction Points*, an denen sich die Teilchenstrahlen kreuzen. An diesen Punkten werden die fünf Experimente aufgebaut (siehe Abbildung 2.1). Vier davon sind für den Proton-Proton-Betrieb ausgelegt und eines für den Schwerionenbetrieb.

In Bezug auf die Geometrie ist das *A-Toroidal-LHC-AparatuS*-Experiment (ATLAS) das größte. Mit ihm soll der Nachweis des *Higgs-Bosons*⁶ gelingen. Des Weiteren sollen verschiedene Theorien jenseits des Standardmodells damit überprüft werden. In dem sehr starken Magnetfeld von 4 T des *Compact-Muon-Solenoid*-Experimentes (CMS) sollen unter anderem Myonen untersucht werden. Wie ATLAS ist CMS ein Mehrzweckexperiment. Bei CMS befindet sich auch das *TOTEM*-Experiment, welches unter anderem den totalen Wirkungsquerschnitt von Proton-Proton-Kollisionen bestimmen soll. Mit dem Experiment *LHCb* sollen die Eigenschaften von *B-Mesonen*, insbesondere die *CP-Verletzung* bei ihrem Zerfall, untersucht werden. All diese Experimente sind auf den Protonenbetrieb bestimmt. Für den Schwerionenbetrieb ist nur *A Large Ion Collider Experiment* (ALICE) ausgelegt, das im nächsten Abschnitt näher erläutert wird.

⁶Das Higgs-Boson ist ein hypothetisches Elementarteilchen, das im Standardmodell der Elementarteilchenphysik vorhergesagt wird (vgl. [Hig64]). Die momentane Untergrenze für die Masse des H^0 ist 114,4 GeV (vgl. [Y+06]).

Betrieb	Kenngröße	Wert
p-p	Proton-Impuls beim Eintritt	450 GeV/c
	Proton-Impuls bei der Kollision	7 TeV/c
	Umlauffrequenz	11,245 kHz
	Zeit zwischen Kollisionen	24,95 ns
	Dipolfeld bei 450 GeV	0,535 T
	Dipolfeld bei 7 TeV	8,33 T
Pb-Pb	Kernladungszahl der Bleiionen	82
	Massenzahl der Bleiionen	208
	Energie pro Nukleon bei Einleitung	0,18 TeV/u
	Energie pro Nukleon bei Kollision	2,76 TeV/u
	Schwerpunktenergie bei Kollision	1148 TeV
	Anzahl Ionen pro Teilchenpaket	$7,0 \cdot 10^7$
	Anzahl der Teilchenpakete	592
	Breite des Strahls am Wechselwirkungspunkt	15,9 μm
	Luminosität	$1,0 \cdot 10^{27} \text{ cm}^{-2}\text{s}^{-1}$
	Totaler Wirkungsquerschnitt	437 barn
	Zeit zwischen Kollisionen	99,8 ns

Tabelle 2.1: Ausgewählte Kenngrößen des Large Hadron Colliders. Quelle: [Y⁺06]

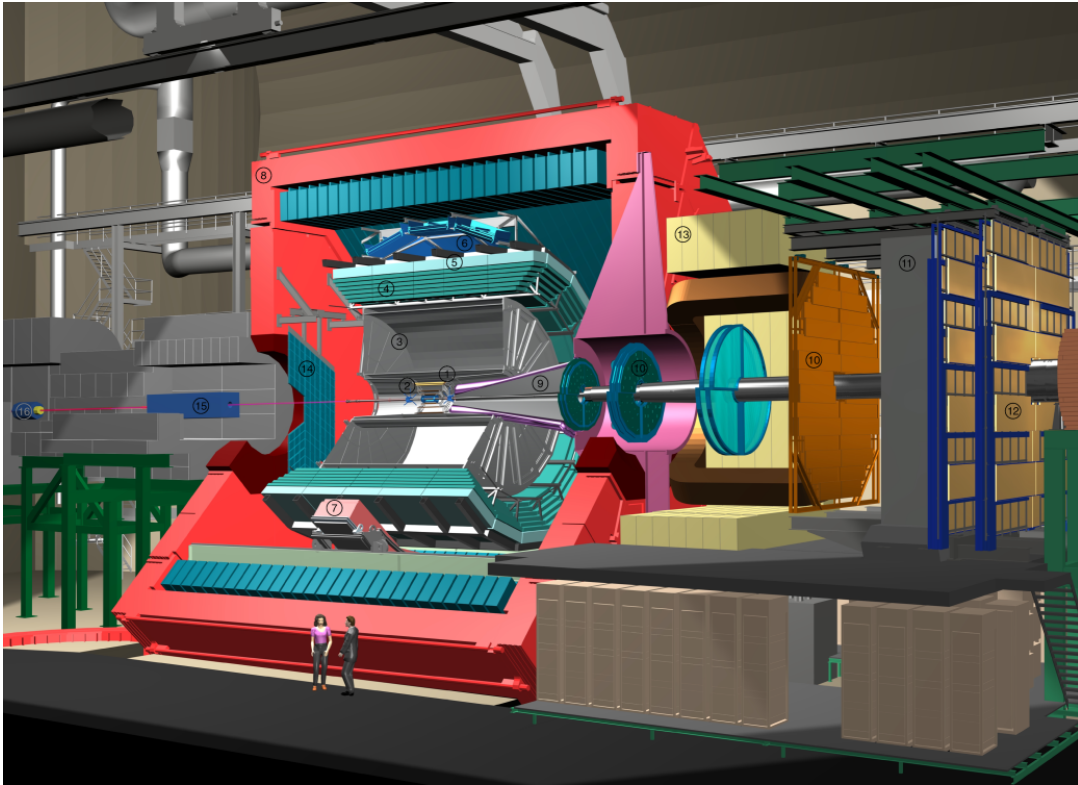
2.2 ALICE

An ALICE sind momentan 900 Wissenschaftler und Ingenieure von über 90 Instituten aus etwa 30 Ländern beschäftigt. Mit diesem Experiment sollen die Eigenschaften von stark wechselwirkender Materie und des *Quark-Gluonen-Plasma* aus Bleikollisionen untersucht werden [A L].

Um das Quark-Gluonen-Plasma nachweisen und untersuchen zu können, muss ein erheblicher Teil der Teilchenspuren erfasst werden. Dies ist eine große Herausforderung, da pro Kollision mehrere tausend geladene Teilchen entstehen. Damit die Spuren der einzelnen Teilchen getrennt werden können, müssen die Detektoren eine ausreichend hohe örtliche Auflösung besitzen.

Der Kollisionspunkt ist von mehreren Detektoren umgeben, die Hadronen, Elektronen und Photonen nachweisen. Innerhalb des 25 m langen und 15 m hohen *L3-Solenoidmagneten* sind die einzelnen Detektoren nach dem Zwiebelschalenprinzip rotationssymmetrisch aufgebaut, wie in Abbildung 2.2 schematisch dargestellt ist. Ein Vorwärtsspektrometer außerhalb des Magneten wird zur Myonendetektion verwendet. Insgesamt haben die Detektoren im Zentralbereich zusammen eine Masse von 10.000 t.

Der L3-Magnet erzeugt ein 0,4 T starkes homogenes Feld, das parallel zur Strahlachse verläuft. Dadurch werden die geladenen Teilchen auf Kreisbahnen gezwungen. Die



- | | |
|--|---|
| 1. ITS (Inner Tracking System) | 9. Absorber |
| 2. FMD (Forward Multiplicity Detector) | 10. Tracking Chambers |
| 3. TPC (Time Projection Chamber) | 11. Muon Filter |
| 4. TRD (Transition Radiation Detector) | 12. Trigger Chambers |
| 5. TOF (Time-of-Flight Detector) | 13. Dipole Magnet |
| 6. HMPID (High-Momentum Particle Identification Detector) | 14. PMD (Photon Multiplicity Detector) |
| 7. PHOS CPV (Photon Spectrometer Charged Particle Veto Detector) | 15. Compensator Magnet |
| 8. L3 Magnet | 16. CASTOR (CentauRO And STRange Object Research) |

Abbildung 2.2: Der Aufbau des ALICE-Experiments. Quelle: ALICE-Collaboration

Krümmung ist dabei impulsabhängig.

2.2.1 Die Detektoren

Die Detektoren sind von innen nach außen wie folgt angeordnet:

Das *Inner Tracking System* (ITS) umschließt den Wechselwirkungspunkt. Es besteht aus Siliziumwafern mit einer sehr hohen Ortsauflösung von bis zu $12\ \mu\text{m}$. Diese sind in sechs zylindrischen Lagen im radialen Abstand⁷ von 3 cm bis 50 cm angeordnet. Damit sollen der Wechselwirkungspunkt sowie die Teilchenpositionen bestimmt werden.

Das ITS ist von der *Time Projection Chamber* (TPC) umschlossen, welche die Teilchenspur im Abstand von 0,57 m bis 2,78 m verfolgt. Die Teilchen durchqueren dabei ein Gasvolu-

⁷Alle Abstände sind bezüglich der Rotationsachse Z angegeben.

men und ionisieren dieses. Durch ein angelegtes elektrisches Feld driften die Elektronen zu Elektrodenpads und werden dort detektiert. Die Driftgeschwindigkeit der Elektronen ist in der Kammer konstant. Daher kann durch Messen der Ankunftszeit auf den Ort der Ionisation geschlossen werden.

Die 18 Supermodule des *Transition Radiation Detector* (TRD) umgeben die TPC. Dieser deckt den Bereich von 2,94 m bis 3,69 m ab. Er ist als schneller Trigger-Detektor ausgelegt und besteht aus sechs Lagen dünner Driftkammern, welche die beim Eintritt entstehende Übergangsstrahlung⁸ und das dazugehörige Teilchen nachweisen. Der genaue Aufbau des TRD wird im Abschnitt 2.3 erläutert.

Eine Lage weiter außen befindet sich der *Time-Of-Flight*-Detektor (TOF). In ihm wird mit 160.000 Parallelscheibenzählern die Flugzeit der Teilchen vom Kollisionspunkt bis zum Abstand von 4 m mit einer zeitlichen Auflösung von 150 ps gemessen.

Im *High-Momentum-Particle-Identifikation*-Detektor (HMPID) wird die Masse von hochenergetischen Teilchen mit Hilfe von Cherenkovstrahlung bestimmt. Er deckt 14 m² der äußeren Zylinderfläche ab.

Das *Photon Spectrometer* (PHOS) ist aus Blei-Wolfram-Kristallen aufgebaut und nimmt das Photonenspektrum auf. Darüber lassen sich Rückschlüsse auf die Temperatur im Kollisionsgebiet ziehen.

2.2.2 Die Experimentsteuerung

Der ALICE-Detektor besteht aus einer Vielzahl verschiedener Subsysteme, die zentral gesteuert werden. Hierbei unterscheidet man zwischen der ereignisbezogenen Steuerung (Triggersystem) und der physikalischen Steuerung (Detektor an/aus). In den folgenden beiden Abschnitten wird eine kurze Übersicht über die Struktur dieser Systeme gegeben.

Triggersystem

Alle Detektoren von ALICE zusammen erzeugen eine Datenrate von bis zu 20 TB/s. Es können dabei nicht alle 10.000 Kollisionen pro Sekunde komplett erfasst werden, da die Auslesezeit von vielen Detektoren länger ist. Die TPC als größte Datenquelle erreicht beispielsweise nur eine Ausleserate von 200 Hz, was um den Faktor 50 zu gering ist, um jedes Ereignis auslesen zu können.

⁸Übergangsstrahlung entsteht, wenn ein geladenes, hochrelativistisches Teilchen eine Grenzfläche zwischen zwei Materialien unterschiedlicher Dielektrizitätskonstante durchtritt. Die Intensität der Strahlung ist vom Lorentzfaktor $\gamma = \frac{E}{mc^2}$ abhängig.

Der wichtigste Punkt ist aber, dass nur ein kleiner Teil der Ereignisse physikalisch interessant ist. Ein frühes Filtern von uninteressanten Ereignissen ist daher auch für die spätere Offlineanalyse von sehr großer Relevanz. Des Weiteren sollen die langsamen Detektoren, aufgrund der daraus resultierenden Totzeit, nur bei interessanten Ereignissen auslesen. Daher verwendet man schnelle Trigger-Detektoren, deren Daten schon kurz nach der Kollision online analysiert werden können. Mit dem Analyseergebnis kann entschieden werden, ob weitere Detektoren ausgelesen werden sollen. Der TRD ist ein solcher Detektor, der nach nur $6 \mu\text{s}$ einen Beitrag zur Entscheidung liefert.

Das Triggersystem von ALICE ist vierstufig. Bei jeder höheren Stufe wird die Entscheidungszeit deutlich länger und es werden Daten von weiteren Detektoren berücksichtigt. Die Kontrolleinheit, welche die zentrale Entscheidung fällt und die Trigger generiert, wird *Central Trigger Processor (CTP)* genannt.

Der *Forward Multiplicity Detector (FMD)* entscheidet $1,2 \mu\text{s}$ nachdem eine Kollision stattfand, ob der *Level-0-Trigger (L0)* ausgelöst wird. Der TRD, das Null-Grad-Kalorimeter und das Myonenspektrometer tragen zur *Level-1-Trigger* Entscheidung nach $6 \mu\text{s}$ bei. Zum *Level-2-Trigger* tragen die TPC und weitere Detektoren bei. In der letzten Stufe, dem *High-Level-Trigger (HLT)*, werden die kompletten Daten aus TPC und TRD mit einer Rate von maximal 1 kHz analysiert. Dieser löst maximal 30 mal pro Sekunde aus, worauf alle Ereignisdaten an das *Data-Acquisition-System*⁹ (DAQ) gesendet werden.

Detektorsteuerung

Die Steuerung des Gesamtdetektors und der Subdetektoren wird von mehreren Systemen bewerkstelligt, die wie im Schema 2.2.2 gegliedert sind.

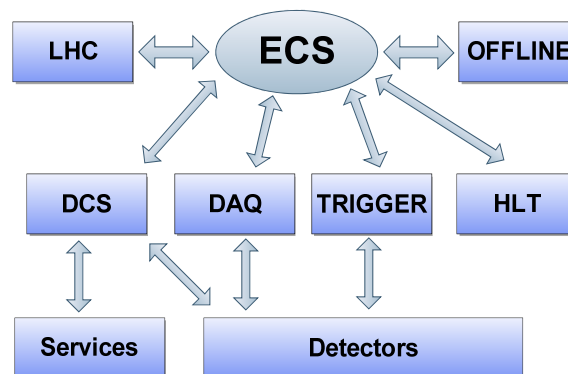


Abbildung 2.3: Hierarchische Struktur der Detektorsteuerung mit dem ECS als zentrale Schicht, über die alle Systeme miteinander verbunden sind und gesteuert werden.

⁹Speichersystem für die spätere Offline-Analyse

Das *Experimental Control System* (ECS) ist die zentrale Schicht, in der die Steuerung und Kontrolle des Experiments stattfindet. Durch diese Schicht sind das *Detektor Control System* (DCS), das *Data-Aquisition-System* (DAQ), das *Trigger-System* (TRG), das *Offline-System*, der *High-Level-Trigger* (HLT) und der LHC miteinander verbunden. Im ECS findet die zentrale Administration und Kontrolle statt.

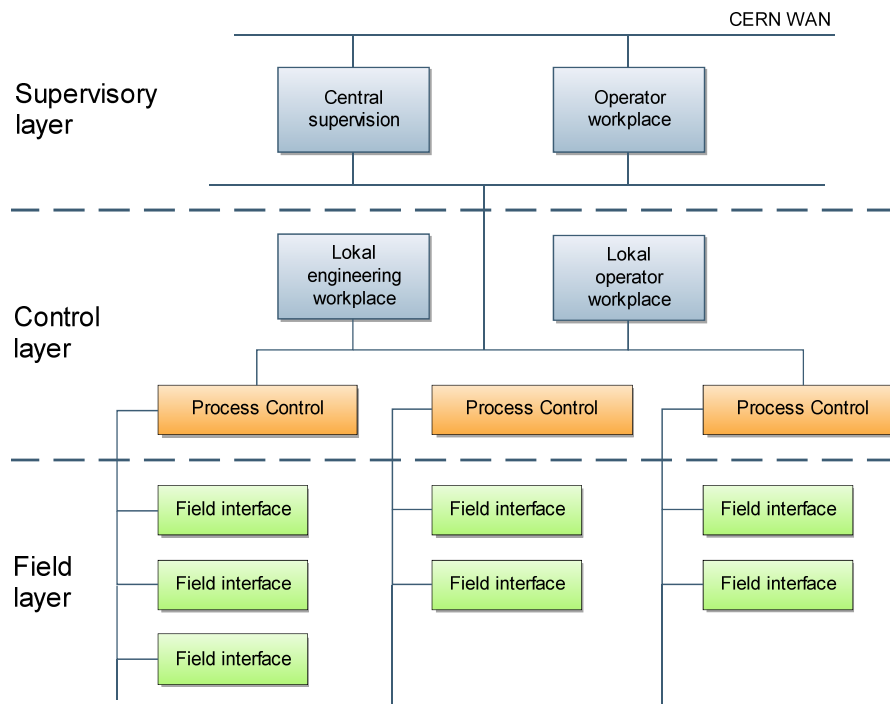


Abbildung 2.4: Übersicht über die Schichten im *Detector Control System*.

Im DCS ist neben dem DAQ die Detektorsteuerung angesiedelt. Es überwacht und steuert die einzelnen Subdetektoren und auch Infrastruktureinheiten wie Spannungsversorgung und Ähnliches. Das „Standard“-DCS-Modell besteht aus einer Vielzahl von Hard- und Softwarekomponenten, die hierarchisch in drei Schichten gegliedert werden können:

- *Supervision Layer*: Hier findet zentral die Überwachung und Steuerung des (Sub-) Detektors statt.
- *Control Layer*: Daten werden hier gesammelt und ausgewertet.
- *Field Layer*: Verbindet das DCS mit der Detektor-Equipment via Sensoren, I/O-Schnittstellen und Feld-Busse.

Diese logische Struktur wird auf die höchste Ebene der Software abgebildet, die durch *Fi-*

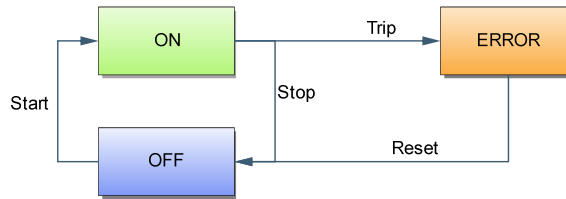


Abbildung 2.5: Übergangsgraph einer einfachen Finite State Machine mit 3 Zuständen

Finite State Machine¹⁰ (FSM) realisiert wird. Hier werden die Knoten auf endliche Automaten abgebildet, die sich in definierten Zuständen befinden und durch Aktionen in andere Zustände übergehen. Ein Zustand basiert entweder direkt auf einem Messpunkt oder aber auf dem Zustand eines untergeordneten Systems. Ein Zustandsübergangsgraph ist in Abbildung 2.2.2 zu sehen.

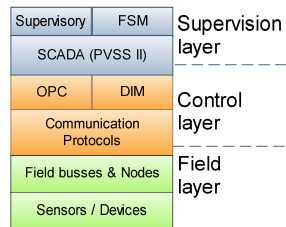


Abbildung 2.6: Überblick über die verwendeten Softwarekomponenten innerhalb des JOCF Frameworks, gegliedert nach den den verschiedenen Schichten des DCS.

FSM ist Teil des JOCF Frameworks¹¹, das auf das SCADA¹² Produkt PVSSII^{TM13} aufbaut. PVSS ist der zentrale Manager für alle Daten und Schnittstellen zu vielen verschiedenen Systemen. Eine Ebene darunter befindet sich die Kommunikationsschicht. Hier kommen verschiedene Protokolle wie zum Beispiel DIM¹⁴ oder OPC¹⁵ zum Einsatz. Hierüber werden alle Systeme gesteuert und Messdaten ausgelesen. In der Hardware laufen Programme, die eines der Protokolle unterstützen. Ein Schema der Softwareschichten ist in Abbildung 2.2.2 dargestellt.

Der logische Aufbau des Detektor Control Systems ist eine Baumstruktur. Jede Subdetektor hat sein eigenes DCS mit den Unterkontrolleinheiten für Niederspannung (LV), Hoch-

¹⁰Finite State Machine ist eine Software, die auf SMI++ basiert und für das DELPHI-Experiment entwickelt wurde.

¹¹Joint Controls Project [CERb]

¹²Supervisory Control And Data Acquisition

¹³Prozess-Visualisierungs- und Steuerungssystem II. Entwickelt von ETM AG, Österreich [ETM]

¹⁴Distributed Information System [Dis]

¹⁵Openness Productivity Collaboration ist eine standardisierte Software-Schnittstelle, die es Anwendungen unterschiedlichster Hersteller ermöglicht, Daten auszutauschen.

spannung (HV), Kühlsystem (Cooling), Ausleseelektronik (FERO¹⁶) und eventuell Gas-system (GAS). In diesem System werden Kommandos von oben nach unten gegeben und der Status bzw. Alarmzustände nach oben propagiert. Dies ist in Abbildung 2.2.2 dargestellt.

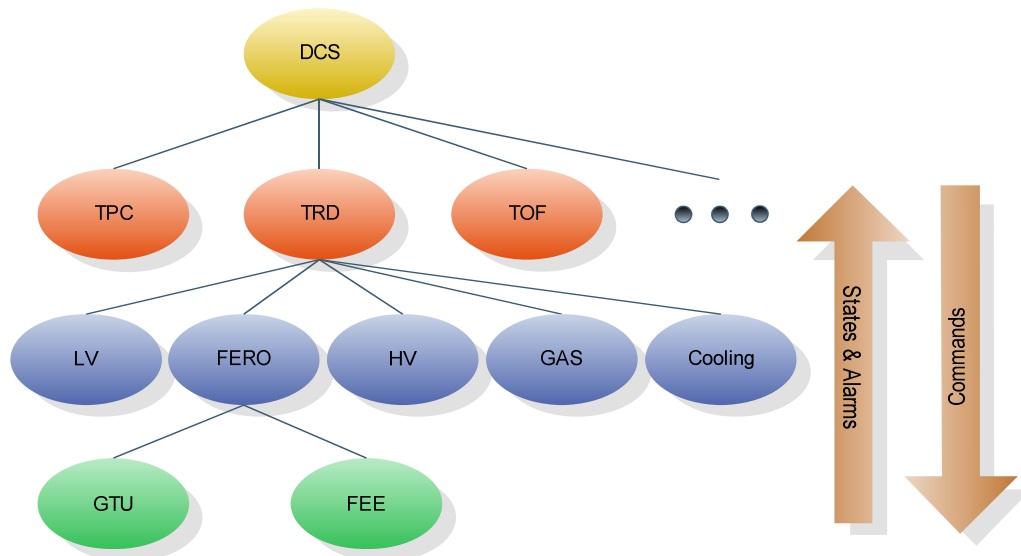


Abbildung 2.7: Vereinfachte Baumstruktur des DCS in ALICE mit Schwerpunkt auf dem TRD.

Ein Teil der Zustandsdaten des Detektors sind nicht nur relevant für die Onlinesteuerung, sondern auch für die spätere Auswertung der Experimentdaten. Um diese Daten hierfür zur Verfügung zu haben, werden diese mit dem *Shuttle*-Programm am Ende einer Messreihe (RUN) an das *Offline*-System geschickt. Der genaue Mechanismus hierfür wird in [OFF06] dargestellt.

Im nächsten Abschnitt wird auf das Funktionsprinzip und die Ausleseelektronik des TRD eingegangen. Des Weiteren wird seine Anbindung an den Trigger, das DAQ-System, den HLT und das DCS vorgestellt.

¹⁶Front-End-Read-Out

2.3 Der Übergangsstrahlungsdetektor (TRD)

Der TRD wurde als schneller Trigger-Detektor konzipiert, der Elektronen bzw. Positronen mit einem Transversalimpuls von mehr als 3 GeV/c erkennt. In diesem Energiebereich ist der Anteil der Pionen sehr viel größer als der der Elektronen, was bei der Konstruktion des TRD berücksichtigt werden muss.

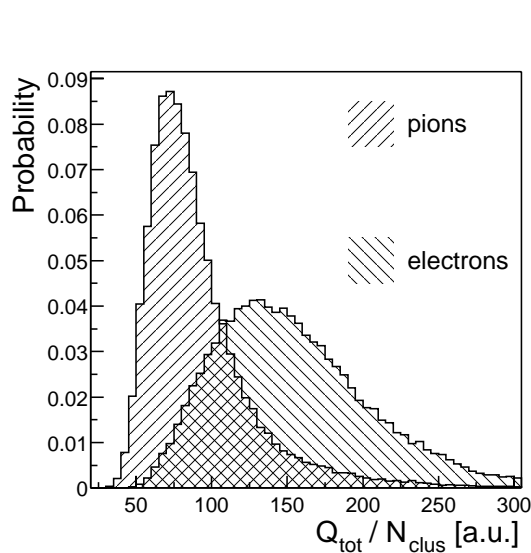


Abbildung 2.8: Pulshöhenverteilung für Elektronen und Pionen mit Transversalimpuls $p_t > 3 \text{ GeV}/c$, integriert über alle Messzeitpunkte einer Kammer. Quelle: [ALI01, S. 102]

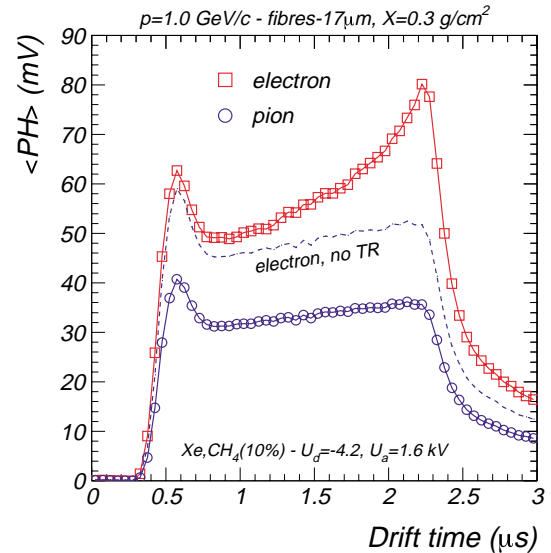


Abbildung 2.9: Pulshöhenverlauf für Elektronen und Pionen über die Driftzeit, gemittelt über viele Einzelmessungen. Quelle: [ALI01]

2.3.1 Funktionsprinzip und Aufbau

Der Detektor besteht aus sechs Lagen, die alle wie folgt aufgebaut sind. An der radialen Innenseite sitzt ein sogenannter *Radiator*, der aus vielen abwechselnden Schichten von *Polypropylenfasern* und *Rohacell-Schaum* aufgebaut ist und die Übergangsstrahlung auslöst. An den Radiator schließt die Driftkammer (Xe, CO₂-Gasvolumen) an. Die hochenergetischen Teilchen, die bei der Kollision entstanden sind, setzen durch Übergangsstrahlung und Ionisation Elektronen frei, welche die Driftkammer durchlaufen. Diese ist in zwei Bereiche unterteilt. In der *Drift-Region* herrscht ein homogenes elektrisches Feld. In ihr driften die Elektronen mit konstanter Geschwindigkeit. In der anschließenden *Amplification-Region* herrscht ein stark inhomogenes Feld, in dem die Elektronen beschleunigt werden und Lawinen weitere Elektronen auslösen. Dies ist nötig, da sonst das Signal zu gering wäre. An den *Kathodenpads*, die am Kammerenden angebracht wird, wird der zeitliche

Verlauf der dort ankommenden Spiegelladungen in Form von Gas-Ionen registriert. Die Elektronen werden zuvor durch Anodendrähten eingefangen. In einer Kammer sind in azimuthaler Richtung 144 solcher *Pads* in einer *Pad Row* angeordnet und je nach Kammerstyp 12 oder 16 *Pad Rows* in longitudinaler Richtung. Der zeitliche Ladungsverlauf in jedem Pad wird in einem Analog-Digital-Wander-Kanal erfasst. Insgesamt hat der Detektor rund 1,2 Millionen Kanäle.

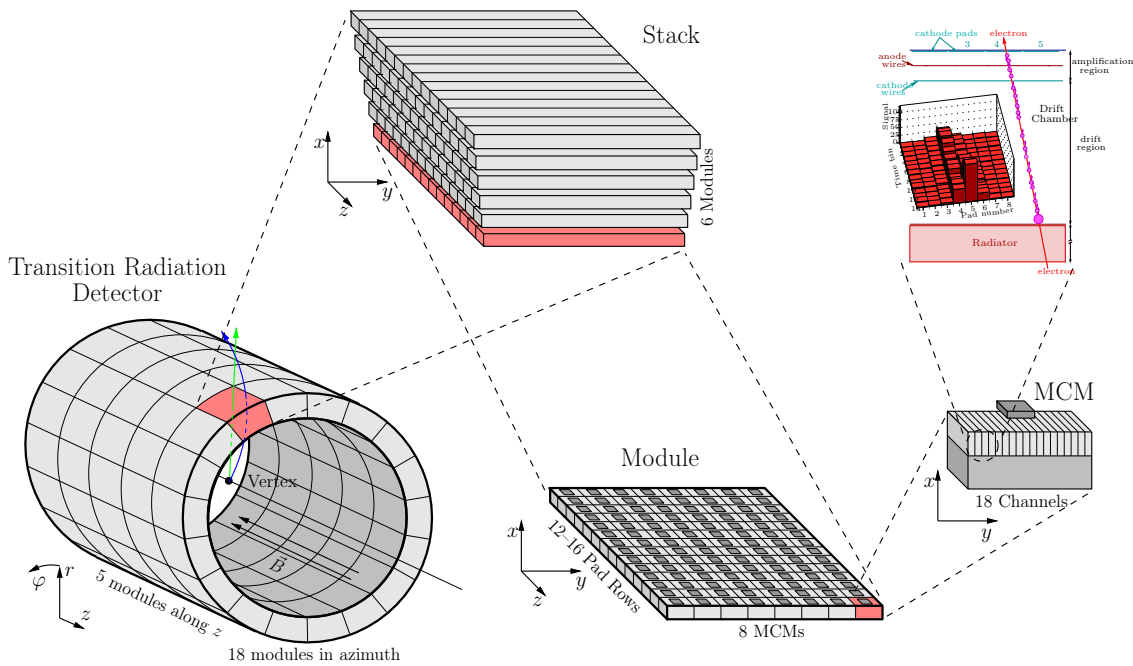


Abbildung 2.10: Aufbau des Übergangsstrahlungsdetektors

Abbildung 2.10 zeigt schematisch den Aufbau des Detektors. Der Radiator, die Driftkammern und die Ausleseelektronik auf der Rückseite bilden ein *Detektormodul*. Sechs Module, übereinander angeordnet, bilden einen *Stack*. Ein *Supermodul* besteht aus fünf Stacks, die in longitudinaler Richtung angeordnet sind. Die insgesamt 18 Supermodule bilden einen Hohlzylinder, der einen Innenradius von 2,9 m und einen Außenradius von 3,7 m hat. Die Supermodule werden im *TRD-Space-Frame* montiert.

2.3.2 Die Front-End- und Readout-Elektronik des TRD

Die digitalisierten Ladungsverläufe auf den *Pads* bilden die Rohdaten eines Events. Ein rauscharmer, ladungsempfindlicher *Pre Amplifier Shaping Amplifier* (PASA) wandelt die Ladung auf den Pads proportional in eine Spannung (6,1 mV/fC) um, die in einem Analog-Digital-Wandler (ADC) digitalisiert werden. Ein digitaler Präprozessor bearbeitet diese Rohdaten und stellt sie in einem Multiportspeicher vier spezialisierten Prozessoren

zur Verfügung. Welche nach kurzen Spursegmenten mit geringem Krümmungsradius suchen. Diese Elektronik befindet sich auf dem *Tracklet Processor Chip* (TRAP). Zusammen mit dem PASA bildet er ein *Multi Chip Module* (MCM). 16 dieser MCMs befinden sich auf einem *Readout Board* (ROB). Je nach Kammerntyp sind sechs oder acht ROBs auf einer Kammerrückseite montiert. Um eine schnelle, latenzarme und redundante Auslese der MCMs zu gewährleisten, sind diese in einer Baumstruktur organisiert. Jedes ROB hat ein zusätzliches MCM als *Board Merger MCM* und jede Kammer hat noch zwei weitere MCMs die so genannten *Half-Chamber-Merger*. Diese tragen jeweils zusätzlich ein *Optical Readout Interface Board* (ORI-Board), welches die Daten einer halben Kammer über eine Glasfaser zur *Global Tracking Unit* (GTU) optisch überträgt. Insgesamt werden 1.080 Fasern verwendet.

Die GTU besteht aus 18 Segmenten, die jeweils ein Supermodul auslesen und bei einer *L2-Accept-Trigger-Message* die Daten weiter an das DAQ-System schicken. Der genaue Aufbau und die weiteren Aufgaben der GTU werden im nächsten Kapitel beschrieben.

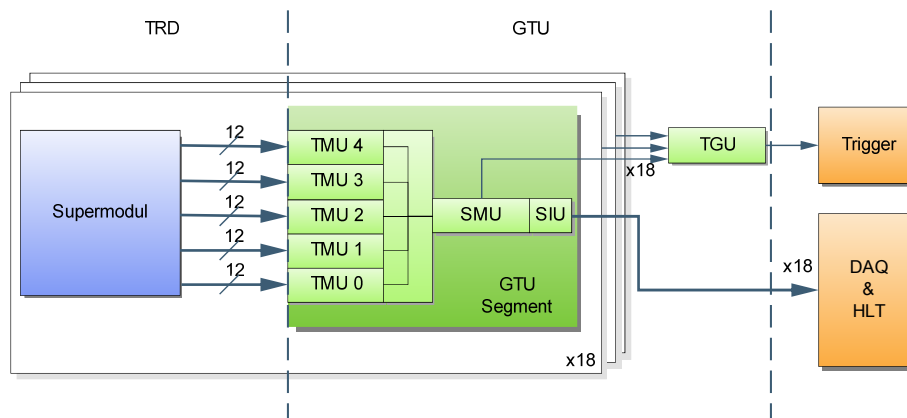


Abbildung 2.11: Schematische Darstellung des Auslesepfades eines Supermoduls durch die GTU in das DAQ-System

2.3.3 Das Detektor Control System im TRD

Der TRD wird nur vom DCS gesteuert und nicht wie andere Detektoren auch vom DAQ-System. Die FERRO-Konfiguration wird in ALICE über den *Detektor Data Link* (DDL) des DAQ-Systems durchgeführt. Eine Übersicht über dieses System ist in [Cho03] dargestellt. Der DDL ist nur in der GTU und nicht im TRD verfügbar. Daher muss die FERRO-Konfiguration im TRD von DCS übernommen werden. Der Daten- und Kontrollfluss durch die verschiedenen Schichten ist in Abbildung 2.12 zu sehen. Ein besonderes Augenmerk gilt hier dem *InterCom Layer*, der den *Control Layer* implementiert, denn in ihm werden die Konfigurationen für die Supermodule abgelegt, die bei der Initialisierung der

Detektorelektronik geladen werden. Der *InterCom Layer* ist transparent für die höheren Schichten.

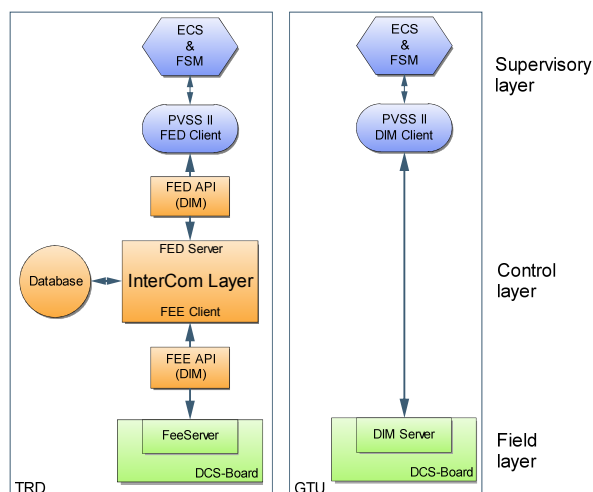


Abbildung 2.12: Aufbau des Detector Control Systems für den TRD und die GTU

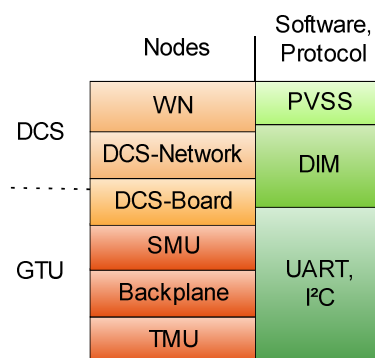


Abbildung 2.13: Aufbau des Field Layer des Detector Control Systems in der GTU

Der Zugriff auf die Hardware erfolgt über DCS-Boards¹⁷, von denen eines auf jeder Kammer sitzt. Auf ihnen läuft ein FEE-Server¹⁸, mit dem über DIM kommuniziert werden kann. Zu den 540 DCS-Boards im Detektor kommen noch 19 weitere für die GTU hinzu. Diese DCS-Boards können die insgesamt 109 Boards der GTU einzeln ansprechen und diverse Sensor- und andere Statusinformationen auslesen und die Boards konfigurieren. Über ein I²C-Bussystem¹⁹ werden Temperatur und Spannungssensoren der Boards ausgelesen. Die Kommunikation mit den Embedded Systems auf den Boards erfolgt über eine erweiterte serielle Schnittstelle, die den parallelen Zugriff auf diese erlaubt. Hierüber werden Statusinformationen der optischen Empfänger und anderen Design-Einheiten im FPGA ausgelesen und Konfigurationskommandos gesendet. Alle Informationen werden über DIM an PVSS übermittelt und Steuerkommandos empfangen. Ein *InterCom Layer* ist für die GTU nicht implementiert, da die Konfigurationen lokal auf den Boards gespeichert werden. In PVSS erfolgt die Überwachung und Visualisierung der über 9000 Messwerte. Dies ist wiederum in das Steuer- und Kontrollsystem des TRD eingebunden. Die genaue Funktionsweise des gesamten Überwachungssystems ist Thema der nun folgenden Kapitel. Im nächsten Kapitel wird die Hardware der GTU vorgestellt.

¹⁷Der Aufbau des *Detector Control System Boards* wird in Kapitel 5 ausführlich dargestellt.

¹⁸*Frontend-Elektronik-Server*, der DIM und eine Schnittstelle zur Datenbank des InterCom Layers implementiert.

¹⁹I²C steht für Inter-Integrated Circuit, gesprochen „I-Quadrat-C“ und ist ein von Philips Semiconductors entwickelter serieller Datenbus.

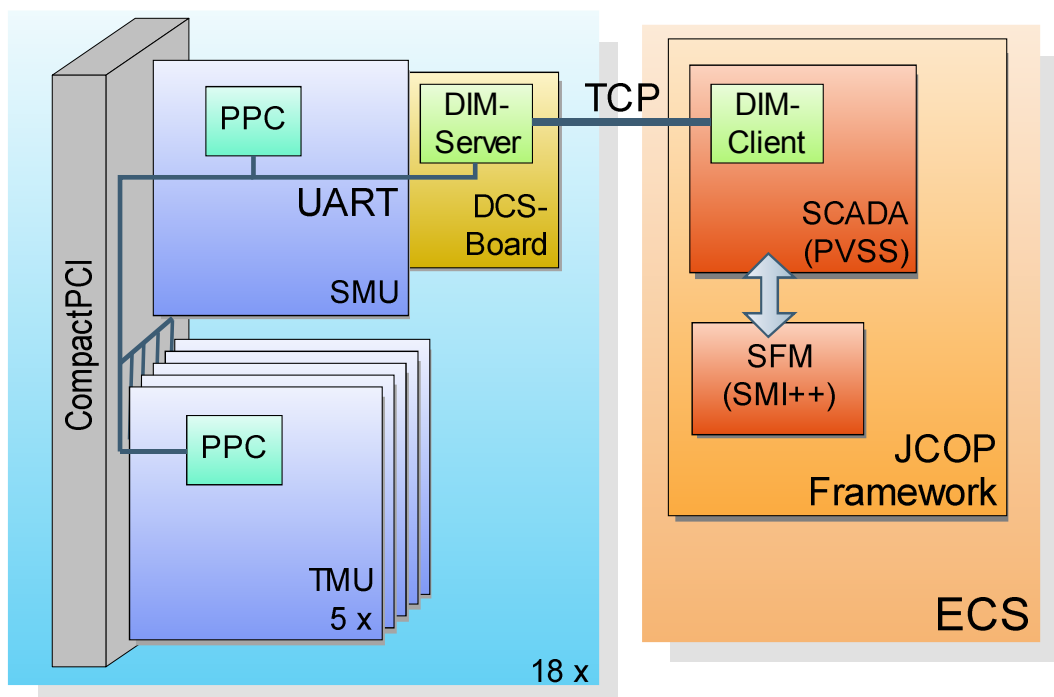


Abbildung 2.14: Kommunikationspfade durch alle Schichten für die Überwachung und Steuerung der GTU.

3 Die Global Tracking Unit

Die *Global Tracking Unit* ist die einzige Schnittstelle zwischen TRD, DAQ und CTP. Über 1.080 optische Fasern werden Daten mit 2 TBit/s Peak vom TRD empfangen, zwischengespeichert, analysiert und je nach Triggerentscheidung über 18 DDLs an das DAQ-System gesendet.

Die GTU befindet sich außerhalb des L3-Magneten unterhalb des Myonen-Systems und ist in drei 19“-Racks (C16 bis C18) untergebracht. Pro Rack sind drei Crates, eine Spannungsversorgung und eine Turbine für die Kühlung montiert. In jedem Crate sind zwei Segmente installiert, die jeweils die Daten eines Supermoduls verarbeiten. Jedes Segment besteht aus einer *Super Module Unit* (SMU) und fünf *Track Matching Units* (TMU). In jeder TMUs kommen die Daten von einem Supermodulstack an. In diesen Daten wird in jeder TMU parallel nach Teilchen mit hohem Transversalimpuls gesucht, indem deren Spur aus den *Tracklets*¹ rekonstruiert wird. Liegt der Transversalimpuls für eine bestimmte Kombination von Spuren über einem Schwellwert, wird eine positive Triggerentscheidung an die jeweilige SMU propagiert und über die *Trigger Unit* (TGU), die im mittleren Rack im mittleren Crate untergebracht ist, gesammelt an den CTP gesendet. Für die gesamte Triggerentscheidung in der GTU stehen weniger als 2 µs zur Verfügung. Nach einem *L2-Accept* werden die Daten von den TMUs an die SMUs übertragen und dann über die DDLs an das DAQ-System geschickt.

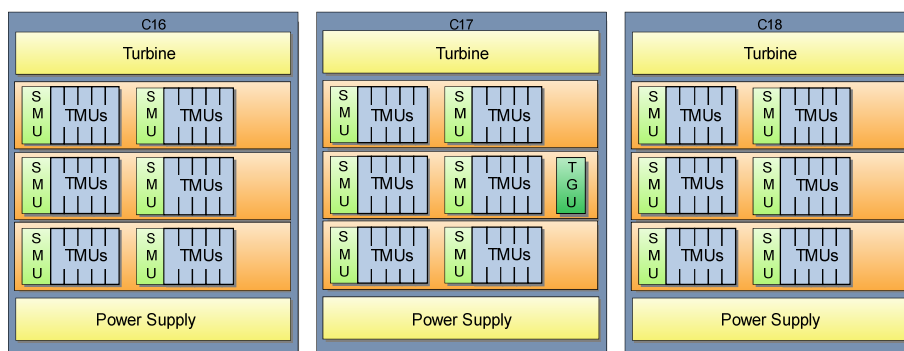


Abbildung 3.1: Schematischer Aufbau der Racks, in welchen die *Global Tracking Unit* installiert ist. In jedem Rack ist unten das Netzteil, darüber drei Crates und oben die Turbine montiert.

¹Tracklet: Parametrisiertes Spursegment einer Lage

In den folgenden Abschnitten wird auf den Aufbau und die Funktion der einzelnen Einheiten eingegangen sowie ein Überblick über die Betriebsparameter und die Sensorik gegeben.

3.1 Aufbau der GTU-Boards

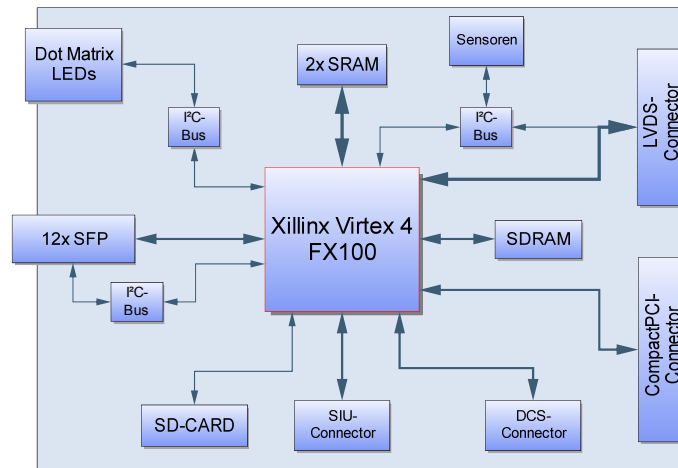


Abbildung 3.2: Schematischer Aufbau eines GTU-Boards.

Alle Module bauen auf einem 14-lagigen CompactPCI-Board mit sechs Höheneinheiten auf, das von Jan de Cuveland im Rahmen seiner Doktorarbeit entwickelt [Cuv]. Die zentrale Einheit ist ein Xilinx Virtex™-4 FX 100. Dieses leistungsstarke FPGA² besitzt 94.896 Logikzellen, 768 frei konfigurierbare I/O-Pins, 20 *Multi-Gigabit Trans-Receiver*-Blöcke (MGT), 160 Digitale-Signal-Prozessor-Einheiten sowie mehrere *Digital Clock Manager* (DCM). Die FX-Serie besitzt darüber hinaus eingebettete PowerPC-Prozessoren, die mit bis zu 450 MHz getaktet werden können. In einem FX 100 sind 2 Prozessoren enthalten. An dieses FPGA sind mehrere Peripheriekomponenten angeschlossen. In einem PROM³ wird die Konfiguration des FPGAs gespeichert. Zwei 512 kbitx36-DDR⁴-II-SRAM⁵-Module werden parallel über einen 128-Bit-Datenbus (zuzüglich Parity-Bits) angesprochen. In diesen werden die Event-Rohdaten gepuffert. Ein 64 MB DDR-II-DRAM⁶ kann als Hauptspeicher für die PowerPC-Kerne verwendet werden. Für nichtflüchtigen Daten-

²Ein *Field Programmable Gate Array* besteht aus einer Vielzahl frei konfigurierbarer Logikblöcke, die über ein komplexes Netz miteinander verbunden werden können. Zur Beschreibung dieser Logik kann die Hardwarebeschreibungssprache VHDL verwendet werden.

³*Programmable Read Only Memory*

⁴*Double-Data-Rate*: Daten werden sowohl zur steigenden als auch zur fallenden Takt-Flanke übertragen. Bei 72 Leitungen werden 144 Bits pro Takt übertragen.

⁵Static Random Access Memory: Daten werden in SRAM-Zellen anstelle von Kondensatoren gespeichert.

⁶Dynamic Random Access Memory: Daten werden als Ladezustand eines Kondensators gespeichert.

speicher steht ein SD⁷-Kartenschacht zur Verfügung. Verschiedene Spannungs- und Temperatursensoren werden über I²C -Busse ausgelesen. Über I²C werden auch die beiden 7x5-Dot-Matrix-LEDs und die übrigen LEDs angesprochen, die auf dem Board vorhanden sind. Zum Empfangen der optischen Daten vom Supermodul können zwölf *Small-Form-Factor-Pluggable-Module* (SFP) aufgesteckt werden, die über impedanzangepasste differentielle Leitungen an die MGT-Blöcke des FPGA angeschlossen sind. Der Systemtakt von 200 MHz wird von einem Quarzoszillator erzeugt. Des Weiteren sind zwei sehr stabile 250 MHz Oszillatoren mit geringem Jitter für die MGTs vorhanden. An der Rückseite des Boards befinden sich Steckverbindungen für die CompactPCI- und die LVDS⁸-Backplane. Auf der Unterseite befinden sich Steckverbinder für das DCS-Board und die DDL-SIU⁹. Zum Testen und zur Fehlersuche sind auch Pins für eine serielle Schnittstelle und für JTAG¹⁰ vorhanden.

3.1.1 Backplanes, Netzteil und Kühlung

In der gesamten GTU kommen drei verschiedene Backplanes zum Einsatz. In jedem Crate der GTU sind je CompactPCI- und LVDS-Backplanes montiert. Diese verbinden jeweils fünf TMUs und eine SMU. In dem Crate mit der TGU ist noch zusätzlich eine CompactPCI- und die *Trigger-Backplane* eingebaut. Auf den Aufbau und die Funktion der einzelnen Platinen wird in den folgenden Abschnitten eingegangen.

LVDS- und Trigger-Backplane

LVDS- und Trigger-Backplane werden von J. de Cuveland [Cuv] entworfen. Die Hauptaufgabe der LVDS-Backplane ist der Datentransfer zwischen den TMUs und der SMU. Jede SMU ist mit allen TMUs über eine Stern- und die TMUs untereinander nochmals über eine Doppelring-Topologie verbunden. Die Leitungen sind LVDS-Leitungen deren Richtung durch das FPGA konfiguriert wird.

Neben den Datenleitungen befindet sich auch noch ein I²C -Bussystem auf der Platine, das alle Einheiten eines Segments miteinander verbindet. Über dieses System können die ADCs der einzelnen Einheiten direkt abgefragt werden. Die Topologie ist in Abbildung 3.1.1 dargestellt. Das DCS-Board kann so über I²C die Sensoren auf allen Boards auslesen. Das I²C -Bussystem ist unabhängig vom Zustand des FPGA, so dass auch dann die Sensoren ausgelesen werden können, wenn das FPGA nicht programmiert ist. Diese Sensorinformationen sind wichtig, um den physikalischen Status der GTU überwachen zu

⁷Secure Disk, Aufbau und Funktion wird in Kapitel 7 dargestellt

⁸Low Voltage Differential Signaling: Schnittstellen-Standard für Hochgeschwindigkeits-Datenübertragung nach ANSI/TIA/EIA-644-1995

⁹Detector Data Link Source Interface Unit: siehe Abschnitt 3.3

¹⁰Joint Test Action Group bezeichnet den IEEE-Standard 1149.1, der ein Verfahren zum Testen und Fehlersuche von elektronischer Hardware beschreibt.

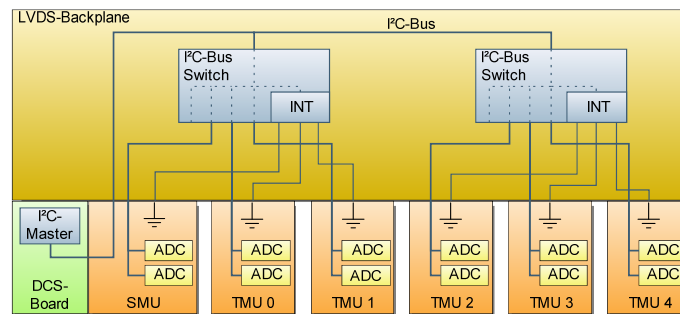


Abbildung 3.3: I²C -Bustopologie auf der LVDS-Backplane. Die I²C -Bussegmente der einzelnen Platinen werden über zwei I²C -Bus-Multiplexer mit dem I²C-Bus verbunden, der über die SMU-Platine direkt an den DCS-Board-Steckverbinder angeschlossen ist.

können. Von besonderem Interesse sind hier Betriebstemperatur und Versorgungsspannungen. Die maximale Arbeitstemperatur des FPGA ist 85 °C. Diese Temperatur wird bei Ausfall der Kühlung (siehe Abschnitt 8.2) binnen kurzer Zeit (ca. 120 s) überschritten und das FPGA kann dadurch zerstört werden. Dies muss das Überwachungssystem erkennen und durch Abschalten der betroffenen Segmente darauf reagieren. Das FPGA und die SFP-Module arbeiten nur bei korrekter Eingangsspannung stabil. Ein Spannungsabfall bei hohem Stromverbrauch muss zum Beispiel erkannt werden, da die Sensleitungen des Netzteils nicht auf den Boards, sondern an der Zuleitung zu den Backplanes angebracht sind und so der Spannungsabfall über die Backplanes vom Netzteil nicht erfasst werden kann. Der genaue Auslesemechanismus wird in Abschnitt 5.3.2 beschrieben.

Über die Backplane werden an alle Boards JTAG-Signale verteilt, über die jedes FPGA und PROM direkt vom DCS-Board programmiert werden kann. Dies ist wichtig, um im späteren Experimentbetrieb das Design aktualisieren zu können. Der genaue Mechanismus ist in [Kir07] dargestellt. Jede LVDS-Backplane besitzt einen Trigger-Uplink in Form eines RJ45-Streckers zur Trigger-Backplane.

Die Trigger-Backplane besitzt 18 RJ45-Strecker für die Trigger-Uplinks der LVDS-Backplanes. Über einen LVDS-Steckverbinder ist sie mit der TGU verbunden, die die Trigger-Informationen auswertet. Der I²C-Bus vom DCS-Board ist im Steckverbinder direkt mit dem I²C-Bus der TGU verbunden, da nur ein Board auf der Trigger-Backplane steckt. Das Auslesen der ADCs vereinfacht sich dadurch, da es keine Bus-Multiplexer gibt, die geschaltet werden müssen.

CompactPCI-Backplane

Die CompactPCI-Backplane ist eine Standardplatine mit drei Höheneinheiten und acht Steckplätzen. Über die Spannungsleitungen des PCI-Busses werden die Boards mit den Primärspannungen von 5 V und 3,3 V versorgt. Des Weiteren kann über den PCI-Bus

mit dem im FPGA instantiierten PCI-Controller auf die Hardware zugegriffen werden, was vor allem für die Fehlersuche verwendet wird, aber im späteren Betrieb nicht mehr vorgesehen ist. Hierfür wird ein zusätzlicher PC mit passender Bauform in das Crate eingeschoben. Die Steuerleitungen des PCI-Busses werden auch für die Kommunikation der Boards untereinander eingesetzt. Hierauf wird in Abschnitt 5.2.1 näher eingegangen.

Netzteil und Kühlung

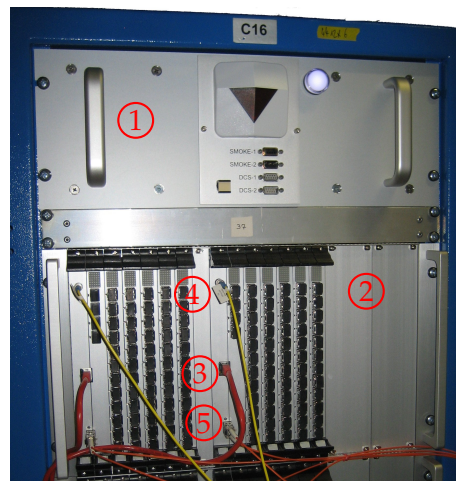


Abbildung 3.4: Oberes Drittel des Racks C16 mit der Turbine (1) und einem GTU-Crate (2), das mit dem DCS-Netz (3), dem CTP (4) und der DAQ (5) verbunden ist. Die optischen Fasern der Supermodule sind nicht gesteckt.

Für ein komplettes Crate wird ein Netzteil benötigt, das mindestens 25 A auf dem 5 V-Kanal und 55 A auf dem 3,3 V-Kanal liefert. Am CERN versorgt ein wassergekühltes Netzteil¹¹ mit zwölf Kanälen ein Rack. Von den zwölf Kanälen werden drei als Doppelkanäle für 3,3 V und drei für 5 V verwendet werden. Jedes Crate besitzt so einen eigenen 3,3-V- und 5-V-Kanal, die als logische Einheit (Gruppe) geschaltet werden. Das Netzteil ist so konfiguriert, dass immer nur Gruppen, nicht einzelne Kanäle, an- oder abgeschaltet werden. Die GTU-Boards könnten durch Ausgleichsströme zerstört werden, die entstehen, wenn nur ein Kanal aktiviert ist. Das Netzteil hat ein Netzwerkinterface und kann darüber per SNMP¹² oder über das darauf aufgesetzte OPC gesteuert werden. Dies wird über eine Graphische Oberfläche in PVSS durchgeführt.

Für die Rackkühlung kommt eine Turbine mit einem Wärmetauscher mit einer maximale Kühlleistung von 3,5 kW zum Einsatz. Sie befindet sich oberhalb der Crates und saugt

¹¹Wiener, Modell PL 512 (vgl. [W-I])

¹²*Simple Network Management Protocol* ist ein einfaches Netzwerkprotokoll das auf UDP aufsetzt und entwickelt wurde, um Netzwerkelemente von einer zentralen Station aus überwachen und steuern zu können.

die Luft an, kühlt und bläst sie über Umlenbleche an der Rackwand nach unten. Die kalte Luft strömt dann von unten durch die Crates und sorgt damit für einen konstanten Luftstrom. Ein Rack erzeugt im laufenden Betrieb knapp 1 kW Abwärme.

3.2 TMU

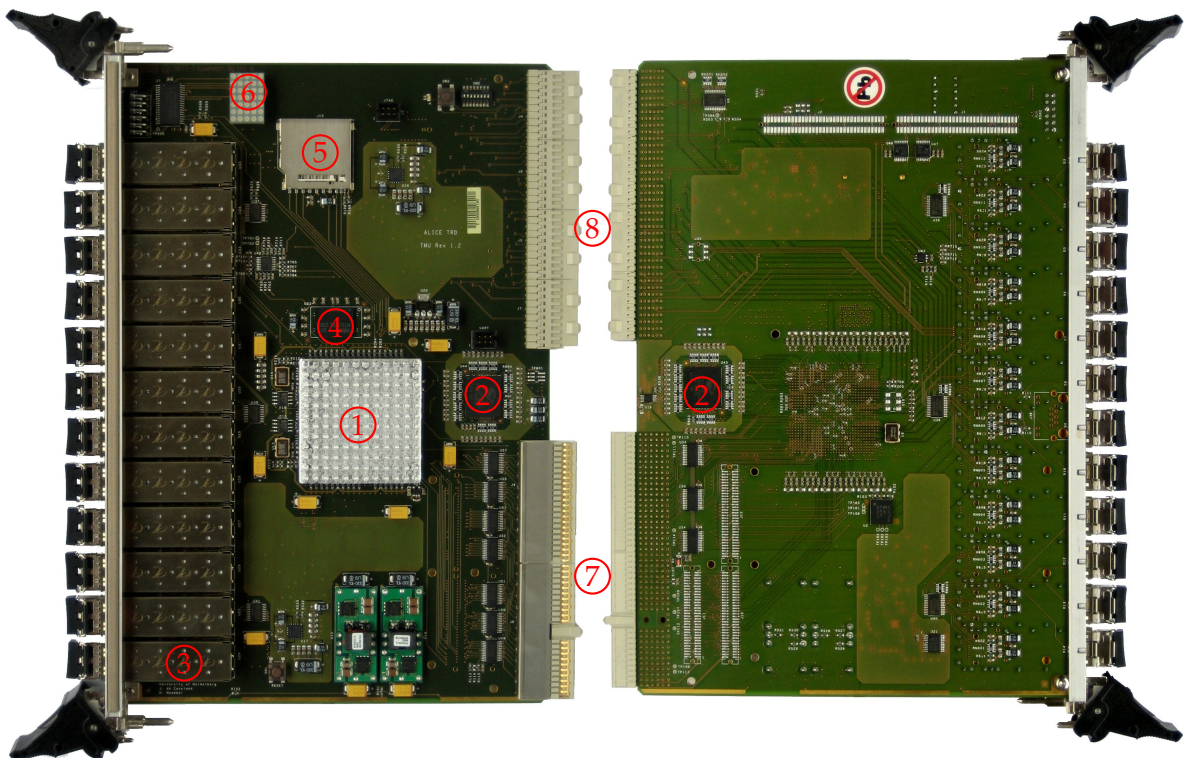


Abbildung 3.5: Die Ober- und Unterseite der *Track Matching Unit*. Die Fotografie zeigt eine voll bestückte TMU mit 12 SFP-Modulen, wie sie am CERN eingebaut ist. Die gekennzeichneten Bauteile sind: (1) FPGA, (2) SRAM, (3) SFP-Module, (4) DRAM, (5) SD-Karten-Schacht, (6) DOT-Matrix-LED, (7) CompactPCI-Stecker, (8) LVDS-Backplane-Stecker.

Die TMU ist an ihrer Frontseite mit zwölf SFP-Modulen bestückt und ist 4 TE breit. Jedem SFP-Modul ist eine LED zugeordnet, die durch ihr Leuchtverhalten (hell, dunkel, blinken usw.) den Linkstatus anzeigt. Über die Dot-Matrix-LED kann ein beliebiger Text oder eine Grafik angezeigt werden. So können weitere Statusinformationen ausgegeben werden. Das FPGA-Design der TMU enthält neben der PowerPC-Einheit zwei weitere große Einheiten. Die eine enthält die Event-Daten-Empfangs- und Speicher-Logik, die F. Rettig in seiner Diplomarbeit [Ret07] entwickelt hat. Aus den Trackletdaten rekonstruiert

die *Track Matching Unit* Teilchenspuren und bestimmt daraus den Transversalimpuls der Teilchen. Es wird auf Teilchen getriggert, deren Transversalimpuls über einem Schwellwert liegt. Diese Einheit benötigt den meisten Platz im FPGA. Dieses Trigger-Design hat J. de Cuveland in seiner Diplomarbeit [Cuv03] für den Schwerionenbetrieb entworfen. Für den Proton-Proton-Betrieb entwickelt von F. Rettig ein Trigger-Design [Ret].

3.2.1 Aufbau der SFP-Modul-Schnittstelle

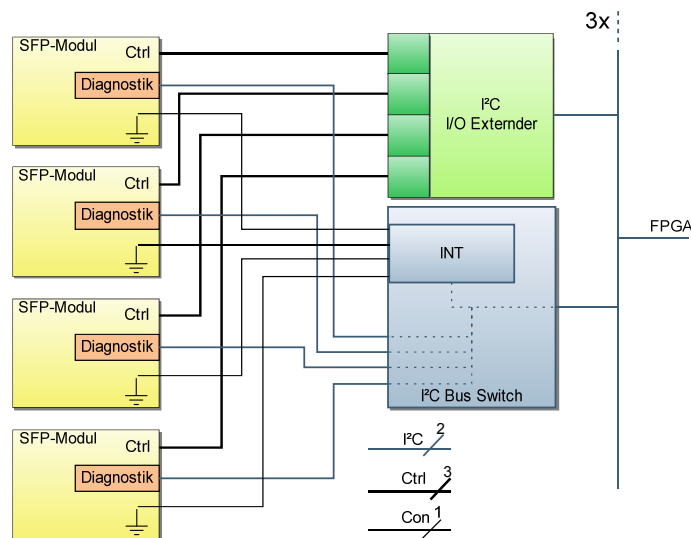


Abbildung 3.6: I²C -Diagnostik-Schnittstelle der SFP-Module mit daran angeschlossenen I²C-Bussystem. Über den I²C-I/O-Extender und den I²C-Bus-Switch ist es möglich, die insgesamt 7 Signale pro SFP und LED über einen Bus abzufragen und zu setzen. Dieses System ist dreimal in jeder TMU und einmal in jeder SMU/TGU vorhanden.

Um die Daten vom *Supermodul* verarbeiten zu können, müssen die optischen Signale in elektrische Signale gewandelt werden. Dies geschieht im SFP-Modul¹³. Der verwendete Typ¹⁴ wandelt optische Signale mit einer Wellenlänge von 850 nm bis zu einer Bitfrequenz von 4 GHz in elektrische. Nach Spezifikation liegt die Fehlerrate unter 10^{-12} bei einer Empfangsleistung von über $105 \mu\text{W}$. Die gemessenen Fehlerraten liegen deutlich darunter. Bei Tests im Labor ist die Verbindung bei einer Sendeleistung von $20 \mu\text{W}$ über mehrere Stunden fehlerfrei. Die SFP-Module besitzen noch eine Diagnostik-Schnittstelle¹⁵, welche über ein I²C-Bussystem mit dem FPGA verbunden ist. Ein im FPGA implementierter Bus-Master liest zyklisch die einzelnen Diagnostikwerte der SFP-Module aus und schreibt die empfangen Daten in einen *Block-RAM*. Ein kompletter Auslesezyklus von allen SFPs dauert etwa 140 ms. Der *Block-RAM* kann über die PowerPC-Einheit

¹³Abmessung und Schnittstelle sind in dem Multi-Source Agreement [SFF00a] festgelegt.

¹⁴Finisar FTLF8524P2BNV

¹⁵Die Spezifikation hierfür ist in [SFF00b] festgelegt.

ausgelesen werden, worauf in Abschnitt 4.2.2 eingegangen wird. Somit stehen alle Diagnostikdaten der SFP-Module zur weiteren Auswertung zur Verfügung.

3.3 SMU und TGU

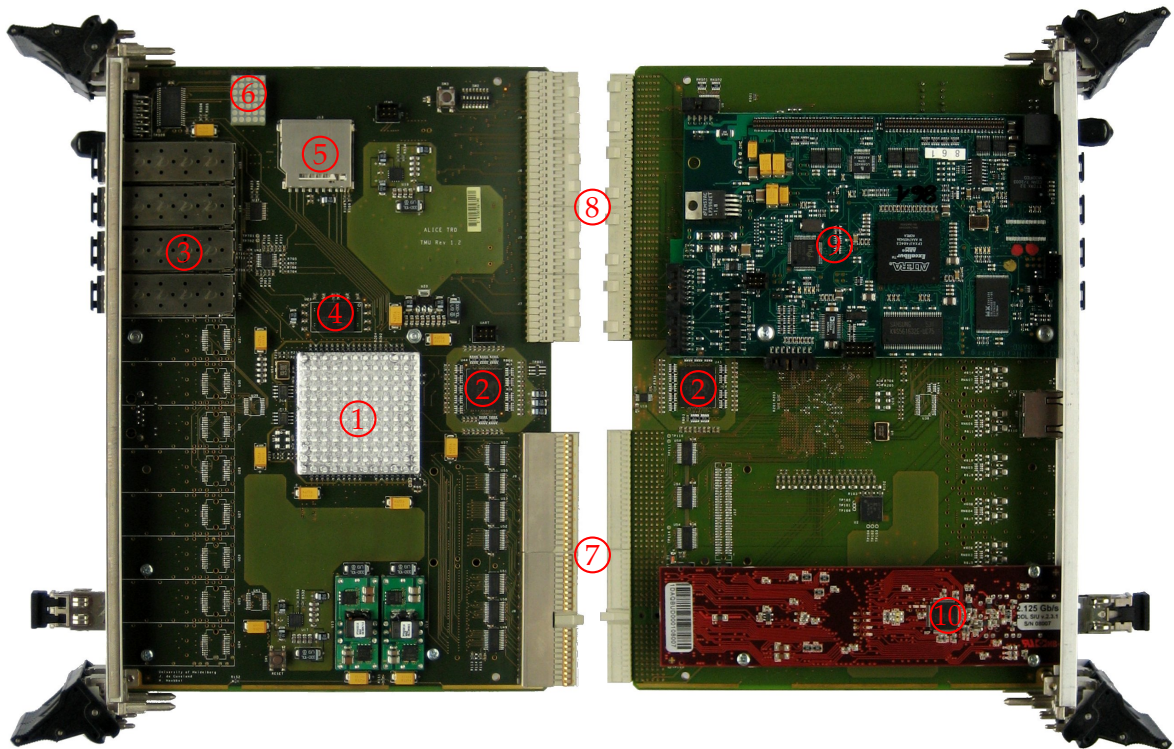


Abbildung 3.7: Die Ober- und Unterseite der *Super Module Unit*. Die Fotografie zeigt eine voll bestückte SMU mit aufgestecktem *DCS-Board* (oben) und *DDL-SIU* (unten) auf der Unterseite, wie sie am CERN eingebaut ist.

Die gekennzeichneten Bauteile sind: (1) FPGA , (2) SRAM, (3) SFP-Module, (4) DRAM, (5) SD-Karten-Schacht, (6) DOT-Matrix-LED, (7) CompactPCI-Stecker, (8) LVDS-Backplane-Stecker, (9) DCS-Board, (10) DDL-SIU.

Die SMU und die TGU unterscheiden sich von der TMU durch eine etwas andere Bestückung. Sie haben nur vier SFP-Cages, die aber momentan nicht genutzt werden. Sie sollen in Zukunft für optisches Ethernet genutzt werden. Dies wird zur Zeit von T. Gerlach entwickelt [Ger]. Auf beiden Boards ist ein DCS-Board aufgesteckt, dessen Aufbau und Funktion in Kapitel 5 erläutert wird. Während die TGU ein Interface zum CTP besitzt, hat die SMU eine DDL-SIU als weitere Aufsteckplatine. Durch diese Erweiterungen sind beide Boards doppelt so breit wie eine TMU.

Das FPGA-Design der SMU kann wie bei der TMU in zwei Einheiten neben der Power-

PC-Einheit aufgeteilt werden. Der eine Design-Block ist für das *Trigger-Handling*, der andere für das *Event-Handling* zuständig. Beide Design-Blöcke werden von Stefan Kirsch [Kir07] entwickelt. Der Status dieser beiden Design-Blöcke, sowie diverse Fehlerzählerstände sind Daten, die sowohl zur Experimentlaufzeit als auch später bei der Offlineanalyse zur Verfügung stehen sollen. Ein Teil dieser Daten werden mit jedem *Event* im *Event Header* sowie mit dem *End of Run Event* an das DAQ geschickt. Andere Teile werden über die in den weiteren Kapiteln vorgestellte Auslekette zur weiteren Verarbeitung zur Verfügung gestellt.

Die Aufgabe der TGU besteht darin, die Triggerbeiträge der 18 SMUs zu einer Triggerentscheidung zu bündeln und zusammen mit dem *Busy-Signal* an den CTP zu senden. Wichtige Systeminformationen sind hier, neben den physikalischen, statistische Daten über den Trigger. Hierzu gehören unter anderem Zähler wie oft ein Trigger ausgelöst wurde und wie viele positive Triggerentscheidungen von den einzelnen SMUs kommen.

DDL-SIU

Die *Detector Data Link Source Interface Unit* ist die Schnittstelle der GTU zum DAQ und HLT. Sie sendet die Daten auch über ein optisches SFP-Modul, das ähnlich aufgebaut ist wie das Modul in der TMU. Leider stehen keine Statusinformationen über Betriebsparameter in der Schnittstelle zur SMU zur Verfügung. Der Linkstatus wird über eine LED angezeigt. Nach einem *L2-Accept* liest die SMU die *Eventdaten* aus den TMUs aus und schickt sie über den DDL zur weiteren Verarbeitung an das DAQ-System und den HLT.

Die Aufgabe ist nun die in diesem Kapitel vorgestellte Hardware zu überwachen und zu steuern. Hierfür wird in der untersten Schicht ein *System on a Chip* mit einem der PowerPC-Kernen als CPU verwendet, das Thema des nächsten Kapitels ist.

4 Das Embedded System in der GTU

Nachdem die ersten Kapitel in das Experiment und den Aufbau der GTU eingeführt haben, wird in diesem Kapitel die PowerPC-Einheit in den FPGAs der GTU dargestellt, die einen der beiden PowerPC-Blöcke verwendet. Es wird gezeigt, wie der PowerPC zum Steuern und Überwachen der Sensoren und Betriebszustände andere Design-Blöcke innerhalb des FPGAs und der Peripheriekomponenten auf dem Board verwendet wird. Die implementierte Schnittstelle bietet die Möglichkeit, sehr einfach Betriebsparameter von Design-Blöcken per Kommandozeile zur Laufzeit auszulesen und zu verändern. Es können Sequenzen von Hardware in Software ausgelagert werden, die in Hardware sehr viel schwieriger zu implementieren sind als in Software und nur mit hohem Aufwand dieselbe Flexibilität erreichen.

Die PowerPC-Einheit ist als eigenständige Entity¹ in das Gesamtdesign eingebunden. Die Entity wird mit dem graphischen Werkzeug *Xilinx Platform Studio*² erstellt. Zunächst wird der Aufbau der PowerPC-Blöcke dargestellt und danach auf die daran angebundene Peripherie und die Software für den PowerPC eingegangen.

4.1 Der Embedded-PowerPC-Processor PPC405

Der PPC405³ ist eine 32-Bit-Implementierung⁴ der PowerPC[®] *Embedded-Environment*-Architektur, die von der PowerPC-Architektur von IBM abstammt. In dieser Architektur wird sowohl die *Big-Endian*⁵ als auch die *Little-Endian*-Adressierung unterstützt. Programme, die allgemein für den PowerPC kompiliert werden, sind voll lauffähig auf dem PPC405. Die PowerPC-Architektur gehört zur Klasse der RISC⁶-Prozessoren.

Im Folgenden wird die Architektur vorgestellt und auf die in diesem Projekt gewählte Implementierung eingegangen, soweit es für den weiteren Verlauf der Arbeit nötig ist. Der Aufbau des Prozessors ist in Abbildung 4.1 dargestellt.

¹Entity: Design-Einheit mit einer definierten Schnittstelle (Port-Deklaration).

²*Xilinx Platform Studio*: GUI, in der ein Embedded-System mit Prozessor und Peripherie-Komponenten zusammengestellt werden kann.

³Embedded PowerPC 405D5 Prozessorkern

⁴Die PowerPC-Architektur ist eine 64-Bit-Architektur mit einer 32-Bit-Untereinheit.

⁵*big-endian*: Das *Most-Significant-Byte* eines Operanden ist an der niedrigsten Speicheradresse im Gegensatz zu *little-endian* bei dem das *Least-Significant-Byte* eines Operanden an der niedrigsten Speicheradresse ist.

⁶RISC: Reduced Instruction Set Controller

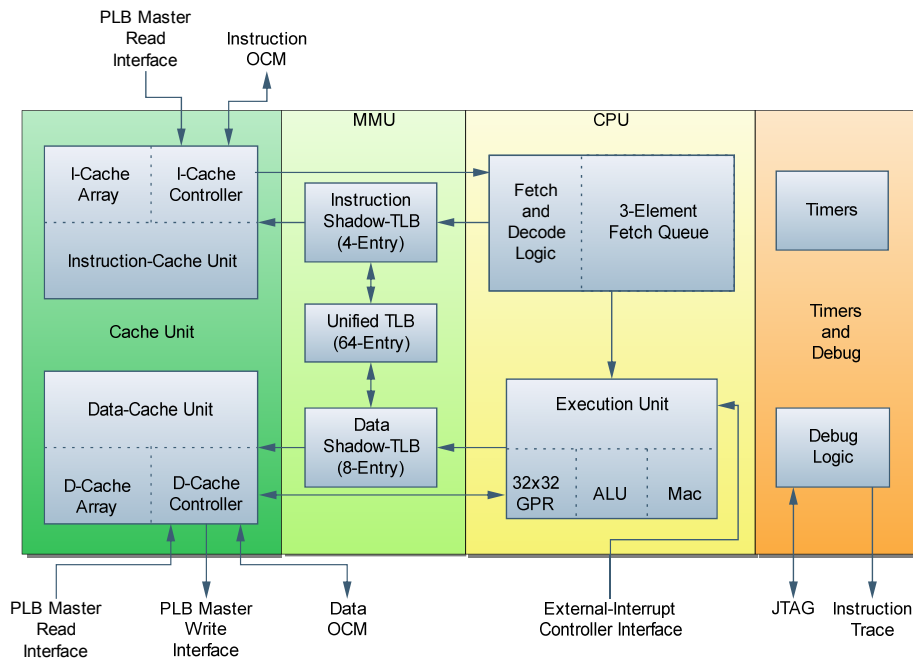


Abbildung 4.1: Schematischer Aufbau des PowerPC-Kerns mit CPU, MMU, Cache, Timer und Debugging-Schnittstelle. Nach [Inc07].

Central Processing Unit (CPU) Die 32-Bit-*Fixed-Point*-Ausführungseinheit ist vollständig kompatibel mit der PowerPC UISA⁷ und besitzt 32 32-Bit-*General-Purpose-Register*. Die Embedded-Erweiterung unterstützt darüber hinaus Funktionen wie *Multiply-Accumulate*-Instruktionen für rechenintensive Anwendungen. Um eine Beschleunigung der Ausführung zu erreichen, wird eine statische *Branch Prediction* und eine fünfstufige *Pipeline*⁸ verwendet. Für die Integerarithmetik kommen schnelle Hardware-Multiplizierer (4 Zyklen) und Dividierer (35 Zyklen) zum Einsatz. Eine FPU⁹ kann extern angebunden werden.

Memory Management Unit (MMU) Die integrierte MMU übernimmt das Speicher-Management und die Übersetzung vom logischen in den physikalischen Adressraum. Sie unterstützt verschiedene *Page*-Größen und eine Vielzahl von Speicherschutz-eigenschaften und Zugriffsoptionen. Ein voll assoziativer *Translation-Look-Aside-Buffer* (TLB) mit 64 Einträgen wird für die Adressübersetzung und den Speicherschutz verwendet. Um Probleme beim gleichzeitigen Daten- bzw. Instruktionszugriff zu vermeiden, werden

⁷User Instruction Set Architecture: Spezifiziert die Basisinstruktionen, Registerbelegung, Datentypen, sowie das Speicher-, Programm- und Exception-Modell. Weitere Spezifikationen sind die *Virtual Environment Architecture* (VEA) und *Operating Environment Architecture* (OEA), die aber nur teilweise unterstützt werden.

⁸Pipeline-Stufen: fetch, decode, execute, write-back, load writeback

⁹Floating Point Unit: Optimierte Einheit für Fließkommaarithmetik

Schattenkopien verwendet, die transparent für höhere Schichten sind. Zwei verschiedene Übersetzungsmodi stehen zur Verfügung, wobei in dieser Arbeit nur der *Real Mode* verwendet wird, in dem die logische Adresse direkt für den Zugriff auf den physikalischen Adressraum verwendet wird. Im *Virtual Mode* wird eine Adressübersetzung von der logischen in die physikalische durchgeführt. Hierüber ist eine bessere Kontrolle und ein besserer Speicherschutz möglich, was grundlegend für den Einsatz von Linux und Multitasking-Software ist.

Cache und Speicher Zwei getrennte 16 kByte zweifach assoziative Caches für Instruktionen und Daten kommen zum Einsatz. Durch integrierte Controller können diese durch *On Chip Memory*¹⁰ (OCM) erweitert werden, deren Zugriffszeiten genauso groß wie bei einem *Cache-Hit* ist. Der Controller übernimmt zusätzlich den Zugriff auf externen Speicher über eine PLB¹¹-Schnittstelle. Hier vermischen sich *Harvard*- und *Von Neumann*-Architektur, da der PowerPC sowohl mit getrenntem als auch gemeinsamem Speicher für Daten und Instruktionen arbeiten kann. In der gewählten Konfiguration des PowerPCs in diesem Projekt wird eine *Harvard*-Architektur verwendet. Die komplette Software liegt im OCM, das als Cache-Erweiterung angebunden ist.

Debugging Erweiterte Debugging-Möglichkeiten werden durch mehrere Komparatoren für Daten, Adressen und Instruktionen und ein JTAG-Interface realisiert.

Time Für die Zeitberechnung wird eine 64-Bit-Zeitbasis verwendet. Somit wird sichergestellt, dass der Zähler in absehbarer Zeit nicht überläuft¹². Es stehen zwei Timer-Register, zwei Kontroll-Register sowie drei verschiedene, programmierbare Timer-Interrupts zur Verfügung. Der *Watchdog*-Timer löst in definierbaren Zeitintervallen einen Interrupt aus. Da der *Watchdog*-Interrupt ein kritischer ist, kann er andere Interrupts unterbrechen (siehe unten) und einen Reset auslösen. Die beiden anderen Interrupts sind ein *Fixed-Interval-Timer* (FIT) und ein *Programmable-Interval-Timer* (PIT). Für den PIT steht ein 32-Bit-Register zum Festsetzen des Zeitintervalls¹³ bereit. Timer mit erweitertem Funktionsumfang können auch als externe Komponenten angebunden werden und über einen externen Interrupt-Controller mit dem PowerPC verbunden werden.

Operating Mode Der PowerPC unterstützt zwei Modi, in denen Software ausgeführt werden kann. Im *Privileged Mode* darf die Software auf alle Prozessorregister zugreifen und alle Instruktionen ausführen. In diesem Modus arbeitet das Betriebssystem und

¹⁰Block-RAM im FPGA, der aus 18 kBit Makrozellen besteht.

¹¹Processor Local Bus, siehe Abschnitt 4.2.1

¹²Bei einer Frequenz von 300 MHz sind es kanpp 2000 Jahre.

¹³Maximale einstellbare Zeit bei 300 MHz sind somit ca. 14 s.

manche Treiber. Der *User Mode* erlaubt den Zugriff auf einen eingeschränkten Teil der Register und Instruktionen. In diesem Modus ausgeführt wird Benutzersoftware.

Exceptions und Interrupts Der Exceptionmechanismus ist in der Embedded Variante erweitert. Es ist eine Zwei-Stufen-Exceptionstruktur definiert, die sich in kritische und nicht-kritische Exceptions unterteilt. Hierfür werden zusätzliche Register und Instruktionen eingeführt. Insgesamt gibt es 19 verschiedene Exceptions. Eine kritische Exception kann eine nicht-kritische unterbrechen, da verschiedene Register für die Zustandsspeicherung verwendet werden. Kritische Exceptions sind folgende Interrupts: *Critical Input, Machine Check, Watchdog Timer, Debug*.

Interface Der PowerPC-Block ist über verschiedene Schnittstellen mit der Peripherie verbunden, die hier aufgelistet sind. Auf einzelne Schnittstellen wird im nächsten Abschnitt noch detaillierter eingegangen.

- Die *Processor-Local-Bus*-Schnittstelle unterstützt einen 32-Bit-Adress- und drei 64-Bit-Datenbusse um die *Cache*-Einheit anzubinden [Xil04].
- Über die *Device-Control-Register*-Busschnittstelle (DCR) können direkt Register im Chip angebunden werden, um Geräte zu steuern. Auf diese Register kann mit speziellen Instruktionen von der Software zugegriffen werden.
- Für Taktverteilung und Leistungsmanagement steht die *Clock- und Power-Management*-Schnittstelle zur Verfügung.
- Der JTAG-Port steht für externes Debugging bereit.
- Eine Schnittstelle für einen *On-Chip-Interrupt-Controller* (OCR) ermöglicht, Interrupts von Quellen inner- oder außerhalb des FPGAs an den PowerPC-Block weiterzugeben. Es gibt jeweils einen Eingang für kritische und nicht-kritische Interrupts.
- Die *On-Chip-Memory*-Schnittstelle (OCM) erlaubt, zusätzlichen Speicher für Daten- und Instruktions-Cache zu verwenden, um die Performance zu steigern.

4.2 Hardware-Architektur

Im Folgenden Abschnitt wird auf die Komponenten eingegangen, die für das Auslesen der systemrelevanten Daten und die Steuerung verwendet werden. Die PowerPC-Einheit und deren Einbindung in das FPGA-Gesamtdesign sind in Abbildung 4.3 und 4.2 dargestellt.

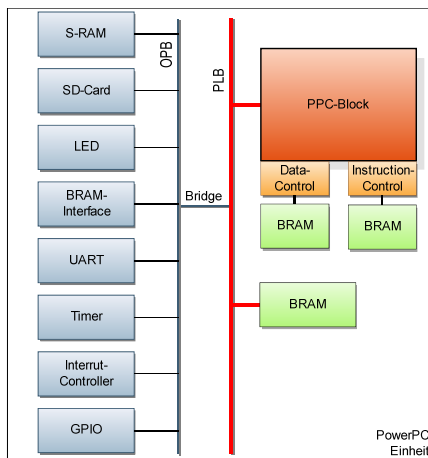


Abbildung 4.2: Überblick über die instantiierten IP-Cores in der PowerPC-Einheit

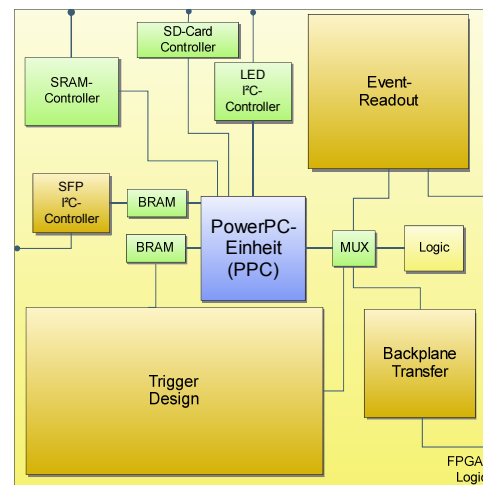


Abbildung 4.3: Schematische Darstellung des Designs im FPGA.

4.2.1 Schnittstellen und Bussysteme

Der PowerPC-Block wird über die oben genannten Schnittstellen über verschiedene Bussysteme mit den Peripheriekomponenten verbunden. Die verwendeten Bussysteme werden nun kurz vorgestellt.

Processor Local Bus (PLB) Der PLB besteht aus einem Controller, einem Watchdog-Timer und separaten Adress-, Schreib- und Leseleitungs-Einheiten, die in nur drei Taktzyklen arbitriert werden können. Der Bus hat 32-Bit-Adress- und 64-Bit-Datenleitungen und unterstützt bis zu 16 Master- und Slave-Einheiten. In der verwendeten Implementierung wird der PLB mit einem 100-MHz-Takt betrieben und ist direkt mit dem PowerPC-Block verbunden. Der Bus ist im Allgemeinen für schnelle, prozessornahere Komponenten wie RAM-Controller und dergleichen vorgesehen. [Xil04]

On-Chip Peripheral Bus (OPB) Der OPB ist über eine Bridge mit dem PLB verbunden. Über diesen Bus werden langsamere Komponenten wie UART oder GPIO¹⁴ angesprochen. Der Bus hat eine 32-Bit-Adress- und zwei 32-Bit-Datenleitungen und kann mit bis zu 100 MHz getaktet werden. In der Implementierung von Xilinx werden nur 16 Slave-Komponenten unterstützt. Eine Komponente hat 16 Takte Zeit, um auf eine Anfrage zu antworten, danach wird die Anfrage mit einem Timeout abgebrochen, falls die Komponente nicht durch Setzen des *Toutsup*-Signals anzeigt, dass sie für die Anfrage länger als 16 Takte benötigt. Ist dies der Fall, wartet der Bus, bis das Signal zurückgenommen wird.

¹⁴General Purpose In Out

Wenn die Komponente das Signal nicht zurück nimmt, steht der Bus bis zu einem Reset. Die Logik des Busses besteht im Wesentlichen darin, die Signale der einzelnen Einheiten miteinander über *OR*-Logik miteinander zu verknüpfen. Je mehr Komponenten am OBP angeschlossen sind, desto mehr sinkt die Performance. [Xil05a]

An diese Busse könne verschiedene Design-Einheiten angebunden werden, die im folgenden Abschnitt erläutert werden.

4.2.2 IP-Cores

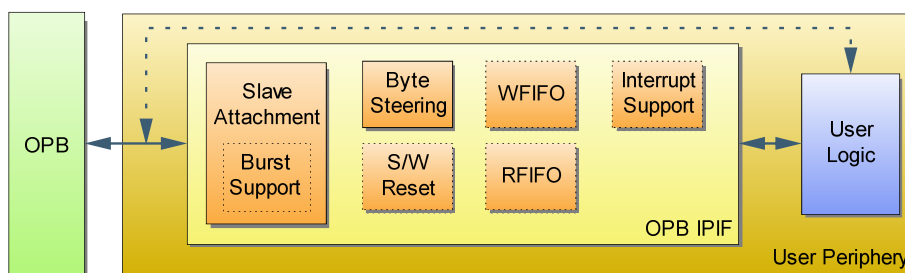


Abbildung 4.4: Aufbau der OPB-Schnittstelle. Über einen Assistenten kann die Schnittstelle mit verschiedenen Komponenten konfiguriert werden.

IP-Cores¹⁵ sind im Allgemeinen Design-Blöcke, die eine bestimmte Funktionalität implementieren. Sie haben je nach Anwendung eine Busschnittstelle als Interface. Das Einbinden dieser Blöcke in Designs ist in XPS sehr einfach. XPS stellt ein Werkzeug bereit, mit dem Schnittstellen für den PLB oder OBP erzeugt werden können. Die Struktur einer dieser Schnittstellen ist in Abbildung 4.2.2 dargestellt.

Im Folgenden werden die für das Projekt implementierten IP-Cores vorgestellt. Alle Cores besitzen eine OPB-Schnittstelle¹⁶, die mit XPS erzeugt werden.

LED-Steuerregister

Die auf den GTU-Boards vorhandenen *Dot-Matrix-LED*-Bausteine sind über das in Abbildung 4.2.2 dargestellte System an die PowerPC-Einheit angebunden. Einem I²C-Controller im FPGA, der 5 × 7 Eingangssignale für die LEDs besitzt, übernimmt die Ansteuerung des Steuerbausteins. Diese Signale sollen per Software gesetzt werden, um die Anzeige der LEDs verändern zu können.

¹⁵Intellectual Property Core: wiederverwendbare Beschreibung eines Logikblocks

¹⁶Die Top-Entity beinhaltet die Deklarationen für die eigentlichen Schnittstellen-Einheit (siehe [Xil05b]) und die User-Einheit, in der sich die Logik befindet, sowie die Port-Deklaration für alle externen Signale.

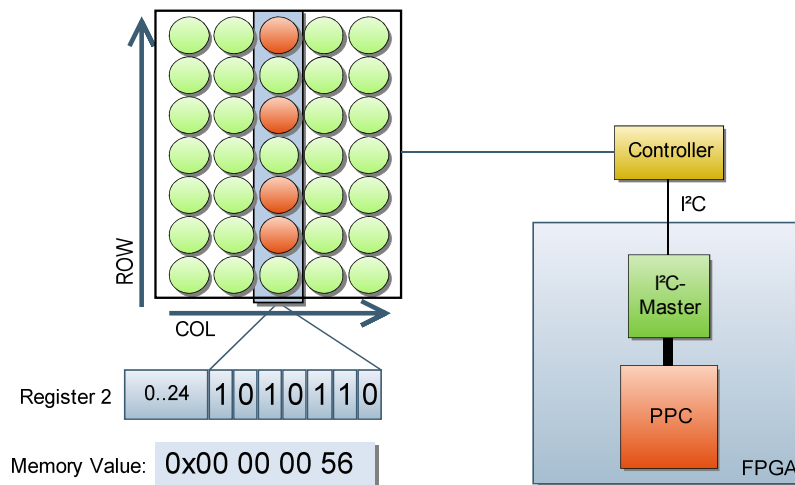


Abbildung 4.5: Zuordnung von LED-Punkten auf Registerwerte und Anbindung des LED-Bausteins an die PowerPC-Einheit. Dieser ist an einen Controller angeschlossen, der über I²C gesteuert wird.

Um dies zu realisieren, wird ein IP-Core generiert, der fünf 7-Bit-Register besitzt, die mit den Eingängen des I²C-Controllers verbunden werden. Da der OPB eine 32-Bit-Datenleitung hat, werden die Register auf fünf aufeinander folgende 32-Bit-Adressen abgebildet. Durch Schreiben auf diese Register lassen sich die LEDs an- und abschalten. Eine LED leuchtet, wenn das ihr zugeordnete Bit auf '1' gesetzt ist. Die Animationssoftware wird in Abschnitt 4.3.3 beschrieben.

GPIO-Schnittstellen

Einfache Schnittstellen ohne aufwendige Transaktionslogik sollen den Zugriff auf Register und Block-RAM-Bausteine außerhalb der PowerPC-Einheit ermöglichen. Diese Einheiten befinden sich größtenteils in anderen Clock-Domains¹⁷ als die PowerPC-Einheit, was bei der Implementierung und Anbindung der Einheiten berücksichtigt werden muss.

Zwei Schnittstellen werden implementiert, wobei die eine nur lesend, die andere sowohl lesend als auch schreibend auf die Logik zugreift. Das Grundprinzip der beiden Einheiten ist gleich. Für den OPB wird eine Schnittstelle generiert und in der User-Logik die Kommunikation mit dem OPB implementiert. Des Weiteren werden Adress- und Datenleitungen als externe Ports definiert und mit der Design-Einheit außerhalb der PowerPC-Einheit verbunden. Bei einem Lesezugriff des OPBs wird die Leseadresse weitergegeben und einen Takt später das *Read-Ack*-Signal zusammen mit den Lesedaten, die gegebenenfalls auf 32 Bit erweitert werden, eine Takt lang gesetzt. Die Schreibzugriffe werden sofort

¹⁷Clock-Domain: Gebiet in einem FPGA, in dem alle Einheiten dasselbe Taktsignal (Clock) haben.

durch Setzen des *Write-Ack*-Signals bestätigt. Bei der Schnittstelle mit Schreibzugriff wird einen Takt lang das *Write-Enable*-Signal zusammen mit der Schreibadresse und den Daten gesetzt und an die externe Logik weiter gegeben, während bei der anderen keine weitere Aktion durchgeführt wird. Das Timingdiagramm ist in Abbildung 4.6 dargestellt.

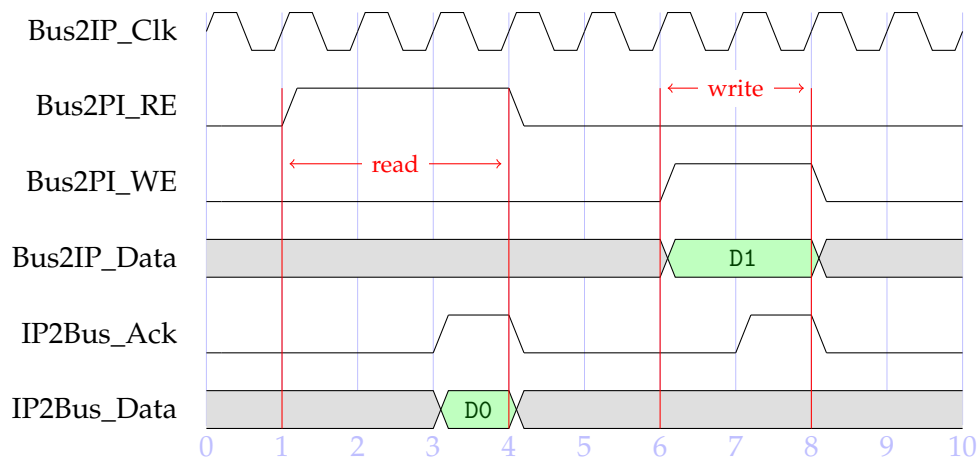


Abbildung 4.6: Timingdiagramm eines Lese- und Schreibzugriffs auf einen OPB-IP-Core

Diese einfachen Schnittstellen erlauben ohne großen Aufwand Einheiten an die PowerPC-Einheit anzubinden und diese darüber zu überwachen und zu steuern. Es muss nur dafür Sorge getragen werden, dass ein *Clock-Domain-Crossing* implementiert wird, wenn sich die Einheit in einer anderen Clock-Domain befindet, da es sonst zu Timing-Problemen in der Synthese des Designs kommen kann. Die Schnittstelle muss nicht mit einem Design verbunden sein, da Zugriffe auf die Schnittstelle auf jeden Fall abgearbeitet werden und bei einem Lesezugriff „Null“ zurück gegeben wird. Der Nachteil hierbei besteht darin, dass nicht direkt überprüfbar ist, ob ein externer Logik-Block angeschlossen ist.

Mit der Leseschnittstelle werden vorzugsweise *Block-RAM*-Bausteine angeschlossen, die zwei Ports¹⁸ zum Lesen und Schreiben besitzen. Der eine Port wird für das Lesen und Schreiben des Logik-Blocks verwendet, der zweite Port für den Zugriff über die PowerPC-Einheit.

4.2.3 Externe Timer- und Interrupt-Controller

Um den externen Interrupteingang des PowerPC-Blocks zu testen, werden ein externer Interrupt-Controller und ein Timer, die als IP-Cores zur Verfügung stehen, instantiiert und mit dem OPB verbunden. Sie zeigen dasselbe Verhalten wie der interne Interrupt-Controller und der interne Timer des PowerPCs. Die Steuerregister der beiden Einheiten

¹⁸Die Clocks der beiden Ports können unabhängig voneinander sein.

werden über den OPB gesetzt. An den Interrupt-Controller können mehrere Komponenten angeschlossen werden, wie zum Beispiel die Interruptleitung des UART-Cores. Für weiterführende Informationen sei auf die Dokumentation der IP-Cores verwiesen. [Xil06a] [Xil06b]

4.3 Software-Framework

In diesem Abschnitt wird auf die Software eingegangen, die speziell für die GTU-Boards zugeschnitten ist. Diese stellt das „Betriebssystem¹⁹“ für die PowerPC-Einheit dar. Sie muss alle Hardwarekomponenten initialisieren und dann in einen Zustand übergehen, der eine Benutzerinteraktion erlaubt. Eine Fehlfunktion der Software würde einen Hardwareneustart erfordern, der im laufenden Betrieb nicht toleriert werden kann. Die Software implementiert einen Kommandozeileninterpreter, der die serielle Schnittstelle für die Ein- und Ausgabe verwendet.

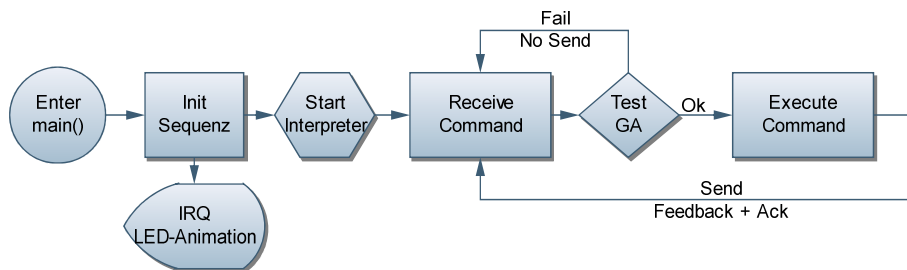


Abbildung 4.7: Flussdiagramm der PowerPC-Software. Nach der Initialisierung wird der Kommandozeileninterpreter gestartet, der auf ein ankommendes Kommando wartet und dieses verarbeitet.

Im Folgenden wird zunächst auf den Aufbau des Kommandozeileninterpreters im Allgemeinen eingegangen und dann exemplarisch Aufbau und Implementierung von Kommandos dargestellt. Anschließend wird die LED-Animation mit Hilfe von Interrupts realisiert. Die gesamte Software ist in ANSI-C geschrieben und wird mit einem von Xilinx angepassten `gcc`²⁰ kompiliert.

4.3.1 Bootsequenz

Das Ergebnis der Hardwaresynthese wird zusammen mit der kompilierten Software zu einem Bit-File²¹ kombiniert. Beim Laden des Bit-Files in den FPGA wird der übersetzte Programmcode in die OCMs geschrieben. Eine in Assembler geschriebene Bootroutine

¹⁹Die Software ist kein vollständiges Betriebssystem, da zum Beispiel kein Scheduler implementiert ist.

²⁰GNU Compiler Collection

²¹Ein Bit-File enthält die Konfiguration der Logik eines FPGAs

initialisiert den PowerPC nach dem Reset und springt am Ende dieser Routine an den Anfang des Hauptprogrammes, das im OCM liegt. Im Hauptprogramm wird zunächst eine Startroutine ausgeführt, die verschiedene Design-Einheiten initialisiert und die Interrupt LED-Animation startet. Danach wird der Kommandozeileninterpreter gestartet, der auf eine Eingabe wartet.

4.3.2 Kommandozeileninterpreter

Ein Kommando ist ein Textstring, der über die serielle Schnittstelle empfangen wird und folgende Struktur aufweisen muss:

```
[GA] Kommandoname [Parameter] 'CR'
```

GA ist die geographische Adresse des Boards, die gleich der PCI-Slot-Adresse ist, in der das Board steckt. Diese ist nur optional. Wenn sie angegeben wird, muss sie mit der Adresse des Boards übereinstimmen, da sonst das Kommando nicht ausgeführt wird. Dies ist nötig, um über eine serielle Schnittstelle mit mehreren Boards kommunizieren zu können. Hierauf wird in Abschnitt 5.2 näher eingegangen. Dem Kommandonamen, der kein Leerzeichen enthalten darf, folgen leerzeichensepariert optionale Parameter für das Kommando. Am Ende eines jeden Kommandos steht ein *Carriage Return*-Zeichen. Ein Kommando ist maximal 80 Zeichen lang. Die Ausgabe des Kommandos wird mit einem *Acknowledge*-Zeichen beendet.

Der Kommandozeilenparser beginnt nach dem Empfang des 'CR' die Eingabe zu parsen. Zunächst wird die GA überprüft. Stimmt diese nicht mit der GA überein, wird die weitere Ausführung abgebrochen und in den Eingabemodus zurückgekehrt, ohne ein *Acknowledge*-Zeichen auszugeben. Andernfalls wird der String in seine Substrings aufgeteilt und diese in einem Array abgelegt. Kann der Kommandonamen in der Liste mit den implementierten Kommandos nicht gefunden werden, gibt der Kommandozeilenparser eine Fehlermeldung aus und beendet das Ausführen des Kommandos. Kann der Kommandoname in der Liste gefunden werden, ruft der Kommandozeilenparser die dazu gehörige Routine mit den übergebenen Parametern auf.

Die Liste ist in Form eines zweidimensionalen Arrays implementiert, die zur Kompilierzeit automatisch generiert wird. Das Array enthält unter anderem Felder für Namen und einen Funktionspointer, der bei einer Erfolgreichen Suche aufgerufen wird. Die Kommando-Routinen bestehen aus einem *Strukt*, der den Kommandonamen und den Hilfetext für das Kommando enthält. Die Übergabe der Parameter beim Aufruf erfolgt über die Parameter `int argc` und `char** argv`.

Mit diesem Framework lässt sich die Software sehr einfach und bequem um zusätzliche Kommandos erweitern. Über die Software kann direkt auf die Hardware zugegriffen werden, die eine Schnittstelle mit der PowerPC-Einheit besitzt.

4.3.3 Interrupt LED-Steuerung

In diesem Abschnitt wird auf die Verwendung von Interrupts am Beispiel einer LED-Animation eingegangen. Für das Auslösen der Interrupts werden die im Abschnitt 4.2.3 vorgestellten IP-Cores verwendet. Die Initialisierung und Konfiguration des Interruptcontrollers und des Timers wird in der Software über Makrofunktionen²² durchgeführt.

Zunächst wird das Exception-Handling für den PowerPC aktiviert, indem die *Exception Vector Table* mit Initialroutinen für jede Prioritäts-Klasse angelegt und die Basisadresse des *Exception-Handling-Programmes* ins *Exception Vector Prefix Register* geschrieben wird. Danach kann ein Interrupt-Handler registriert werden, dem eine Priorität und eine Routine übergeben wird. Die Routine wird beim Interrupt ausgeführt.

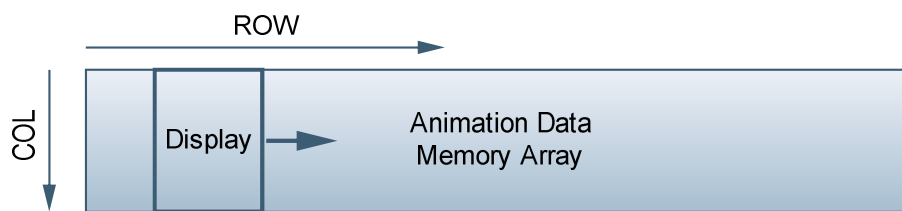


Abbildung 4.8: Darstellung der Anzeigefensterfunktion für die LED-Matrix.

Die Interruptroutine schreibt Daten in die LED-Register, um so eine Animation zu erzeugen. Die „Bildwiederholrate“ ist gleich der Interruptauslöserate, die über den Timer eingestellt wird. In der gewählten Implementierung wird ein *Speicher-Array* über eine Fensterfunktion auf die LED-Register abgebildet, die bei jedem Schritt um eine Zeile weiter springt. So dauert es sieben Aufrufe, bis ein komplett neues Bild auf den LEDs angezeigt wird.

Über ein Kommando kann der angezeigte Text und die Anzeigegeschwindigkeit verändert werden. Darüber hinaus gibt es die Möglichkeit, die aktuelle Temperatur ausgeben zu lassen, die aus dem Temperatursensor ausgelesen wird. Diese Animation läuft in jedem Board. Um die Anzeige der einzelnen Boards zu synchronisieren, können auch Interrupts verwendet werden. Das Design der PowerPC-Einheit sieht einen externen Interrupteingang hierfür vor.

Das vorgestellte Embedded System überwacht und steuert die Boards und die Design-Einheiten im FPGA. Im nächsten Kapitel wird auf die Schnittstelle zwischen der GTU und dem DCS eingegangen, um damit den Zugriff auf die Boards über dieses zu ermöglichen.

²²Durch Makrofunktionen werden die Zugriffe auf die Register der Controller gekapselt. Näheres bei [Jac05]

5 Die Schnittstelle DCS-Board

Das Ziel bei der Entwicklung des *Detector Control System Boards* ist, eine Schnittstelle zwischen der Hardware und dem *Detektor Control System* zu implementieren, die stabil unter extremen Bedingungen arbeitet und einen einfachen Zugriff auf die Hardware via Netzwerk gestattet. Da das Board auch innerhalb des L3-Magneten eingesetzt wird, muss es resistent gegen das dort herrschende Magnetfeld und die Strahlung sein.

Im Folgenden wird zunächst die Hardware des DCS-Boards vorgestellt und anschließend die Anpassungen für die GTU diskutiert.

5.1 Hardware des DCS-Boards

Das *Detector Control System Board* ist eine Platine, auf der ein *Embedded-Linux* System läuft. Das zentrale Element ist der *ALTERA Excalibur™-Chip*, der eine in das *FPGA-Fabric* eingebettete *ARM-CPU* besitzt. Bestückt mit einem 8 MB Flashspeicher und 32 MB SDRAM ist es möglich, ein *Embedded Linux* zu installieren, auf das über Ethernet zugegriffen werden kann. Im FPGA ist unter anderem ein I²C-Busmaster instantiiert. Die in der GTU eingesetzten Boards sind mit einem *TTCrx*-Chip bestückt, der die Nachrichten des CTP decodiert, die über den optischen Link (TTC) empfangen werden.

In den folgenden Abschnitten werden die Anpassungen und Erweiterungen der Hard- und Software diskutiert, die für die Kommunikation mit den GTU-Boards nötig sind.

5.2 Kommunikationsschnittstelle zur GTU

Es gilt folgende Problemstellung bestmöglich mit wenigen Ressourcen und Programmieraufwand zu lösen:

Eine zuverlässige, Ressourcen sparende Kommunikation des DCS-Boards mit den GTU-Boards soll bereit gestellt werden, wobei nur zwei direkte Leitungen vom DCS-Board zur SMU genutzt werden können.

Für eine Kommunikation über zwei Leitungen kommt nur ein serielles Protokoll in Frage, das kein Taktsignal zum Abtasten benötigt. Das Protokoll RS232 der UART¹-Schnitt-

¹Universal Asynchronous Receiver/Transmitter

stelle stellt ein solches bereit. Hierfür gibt es eine sehr breite Unterstützung sowohl auf Seiten der PowerPC-Einheit als auch von Linux. Des Weiteren stehen instantiierbare IP-Cores für die verwendeten Plattformen zur Verfügung. Im FPGA des DCS-Boards kann ein zusätzlicher UART-Core instantiiert werden, der mit den Leitungen zur SMU verbunden und an den Avalon-Bus² des ARM-Cores angeschlossen ist. Auf Seiten der GTU-Boards werden die Signale auf alle Boards verteilt und mit der lokalen UART-Schnittstelle verknüpft.

Alle Änderungen und Anpassungen des bisherigen Designs sind in den nun folgenden Abschnitten dargestellt. Zuerst wird auf die Hardware eingegangen, dann auf die Software. Dabei wird vor allem der neu geschriebene UART-Treiber für das DCS-Board vorgestellt.

5.2.1 Hardware der Kommunikationsschnittstelle zur GTU

Sowohl auf dem DCS-Board als auch auf den GTU-Boards müssen Erweiterungen des bisherigen Designs vorgenommen werden.

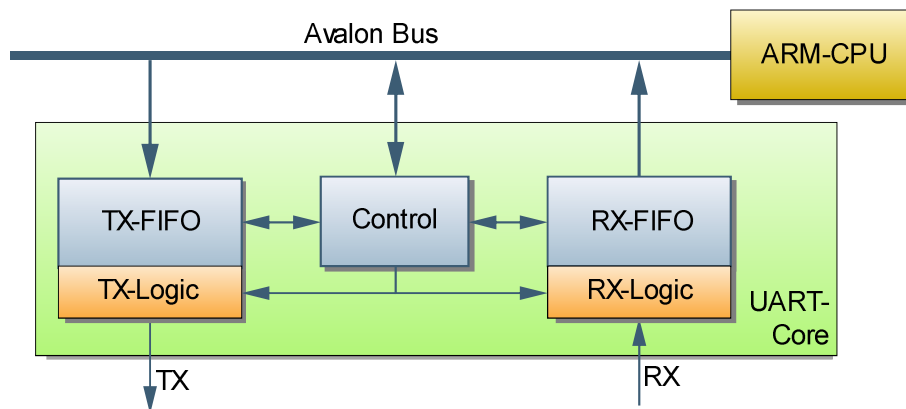


Abbildung 5.1: UART-Cores im FPGA des DCS-Board mit Anbindung an den Avalon-Bus

DCS-Board Im FPGA des DCS-Boards wird ein UART16550-IP-Core³ mit einer Wishbone-Bus-Schnittstelle von Open Cores⁴ verwendet. Die Wishbone-Bus-Schnittstelle ist kompatibel mit der Schnittstelle des Avalon-Buses und kann direkt mit diesem verbunden werden. Der IP-Core unterstützt Interrupts und die Übertragungsrates ist frei über

²Peer-To-Peer-Bus-System von Altera

³Mit dem Industriestandard „NS16550A“ (National Semiconductors’ 16550A device) konforme UART-Schnittstelle.

⁴<http://www.opencores.org>

Register einstellbar⁵. Die genaue Implementierung ist in der Arbeit von S. Kirsch [Kir07] dargestellt.

GTU-Boards Das Leitungssystem für die Kommunikation ist in Abbildung 5.2.1 dargestellt. Die beiden Leitungen vom DCS-Board sind direkt an das FPGA der SMU angeschlossen. Das RX-Signal⁶ wird über die CompactPCI-Backplane an die TMUs verteilt und jeweils mit dem lokalen RX-Signal über ein AND-Gatter verknüpft und an den RX-Eingang des PowerPC-Blocks gelegt. Die TX-Signale der TMUs teilen sich eine Leitung auf der CompactPCI-Backplane. In der SMU wird das Signal von der Backplane mit dem TX-Signal des PowerPC-Blocks mit einem AND-Gatter verknüpft und an den Port für das RX-Signal des DCS-Boards angeschlossen. Auf allen Boards sind *Pull-Up*-Widerstände an den Pins, um die Boards auch alleine, ohne Backplane, betreiben zu können, da die Signale *Low-Active* sind. Dies ist auch der Grund für die Verknüpfung mit AND-Gattern an Stelle von OR-Gattern. Mit dieser Architektur ist es möglich, allen

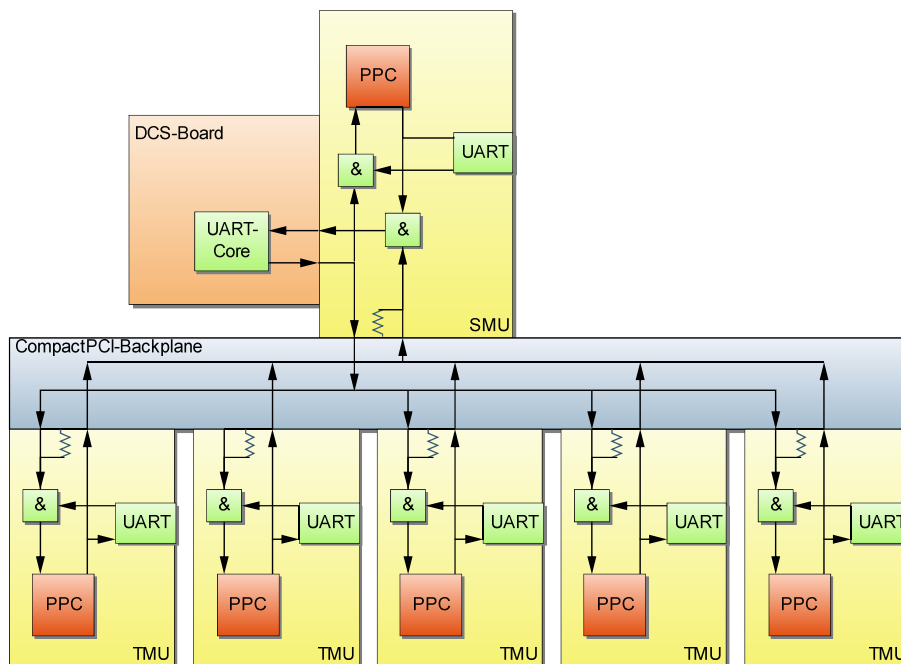


Abbildung 5.2: Leitungssystem der „active-low“ Kommunikation über UART zwischen dem DCS-Board und den GTU-Boards. An den Aus- und Eingangspins der FPGAs sind mit *Pull-Up*-Widerstände verbunden.

sechs Boards gleichzeitig Kommandos zu schicken. Ein weiterer Vorteil besteht darin, dass kein großen *Protokoll-Overhead* anfällt und zur Fehlersuche auch direkt die serielle

⁵Die Dokumentation und der Source-Code bei [Jac02] zu finden.

⁶Aus Sichtweise der Schnittstellen in den FPGAs. Das TX-Signal des DCS-Boards wird mit dem RX-Signal im FPGA verbunden und umgekehrt

Schnittstelle der Boards zur Kommunikation mit einem PC verbinden und über RS232 kommunizieren kann. Diese Architektur ist sehr schlank und Ressourcen sparend.

Ein Nachteil dieser Architektur besteht darin, dass die Kommunikation mit nur einem Board ausschließlich über ein Protokoll in Software (siehe Abschnitt 5.2.2) möglich ist. Wenn mehrere Boards gleichzeitig senden, überlagern sich die Signale auf der Leitung und sind nicht mehr zuverlässig decodierbar im DCS-Board. Es besteht darüber hinaus die Gefahr, dass die Kommunikation gestört wird, wenn ein Board in einen undefinierten Zustand gelangt und unkontrolliert sendet. Im schlimmsten Fall ist kein Empfangen von Antworten auf dem DCS-Board mehr möglich. Dies könnte durch getrennte Leitungen für die TX-Signale der einzelnen Boards und einem *Multiplexer* in der SMU behoben werden. Dadurch würde aber ein weiterer Overhead entstehen, da der Multiplexer geschaltet werden muss. Ein Decoder muss in der SMU laufen, der das Kommando decodiert und den Multiplexer passend schaltet. Dies muss in Echtzeit geschehen, da sonst die Antwort des Boards verloren geht. Da es bisher keine Probleme mit der einfachen Implementierung gibt, wird von der Erweiterung bisher abgesehen. Sie bringt zwar die Sicherheit, dass falls eine TMU ausfällt die Kommunikation nicht blockiert wird, aber auch neue Ausfallmöglichkeiten, da nun eine aktive Leitungsarbitrierung nötig ist. Darüber hinaus werden mehr Ressourcen benötigt. Hier fallen vor allem die Backplane-Leitungen ins Gewicht, da von diesen nur wenige zur Verfügung stehen und dann nicht mehr für andere Zwecke verwendet werden können.

Des Weiteren ist es problematisch, wenn in der SMU kein FPGA-Design geladen ist. Dann ist keine Kommunikation möglich, da die Leitungen durch das FPGA an die *Backplane* gelegt werden. Dieses Problem kann nicht behoben werden, da keine direkten Leitungen von DCS-Board zu der *Backplane* für diesen Zweck vorhanden sind. Im laufenden Betrieb wird unter normalen Bedingungen nie der Fall eintreten, dass kein Design in der SMU geladen ist.

Nachdem nun die Hardwareschicht der Kommunikation beschrieben ist, wird die darauf aufbauende Treiber bzw. Softwareschicht vorgestellt.

5.2.2 UART-Treiber für das DCS-Board

Der Standard-Linux-UART-Treiber stellt ein Vielfaches an Funktionalität bereit, die hier nicht benötigt wird. Daher wird ein schlanker, funktionaler Treiber implementiert, der für die Aufgabenstellung optimiert ist und minimale Ressourcen benötigt.

Bevor auf die Implementierung eingegangen wird, wird die Funktionsweise eines Treibers in Unix/Linux vorgestellt und die Anforderungen in diesem Fall aufgelistet.

Treiberkonzept bei Linux

Bei Unix wird die Philosophie verfolgt „Everything is a file“. Für ein Programm stellt es daher keinen Unterschied dar, ob es auf eine Datei im Dateisystem oder auf eine Hardwarekomponente in Form einer Gerätedatei zugreift. Der Treiber muss daher eine Schnittstelle bereit stellen, die Funktionen wie Öffnen, Schließen, Lesen und Schreiben implementiert. Ein Treiber ist ein *Kernel-Modul*, das im *Kernel-Space* des Hauptspeichers läuft und nicht im *User-Space* wie normale Programme. Für eine tiefere Einführung in die LinuxTreiberProgramierung sei auf [Ale01] verwiesen.

Gegebenheiten und Anforderungen an den Treiber

Die Kommunikation mit den GTU-Boards folgt einem festen Protokoll, das wie folgt festgelegt ist:

1. Das DCS-Board sendet ein Kommando.
2. Ein Kommando, das eine Antwort erwartet, wird mit der „Adresse“⁷ des GTU-Boards begonnen und endet mit einem *Carriage Return*.
3. Ein Kommando ohne Adresse ist ein *Broadcast* und wird von jedem Board ausgeführt. Es gibt hier keine Antwort.
4. Ein GTU-Board sendet nie, ohne zuvor eine Kommando erhalten zu haben, das für das Board bestimmt ist.
5. Die Antwort der GTU-Bords endet mit einem *Acknowledge*⁸

Beginnzeichen	Kommando/Antwort	Endezeichen
GA	Kommando (max 255 Zeichen)	'CR'
keins	<i>Broad-Cast</i> -Kommando (max 255 Zeichen)	'CR'
keins	Antwort (max 32 kB)	'ACK'

Tabelle 5.1: Codierung der Token. 'CR' = *Carriage Return*, 'ACK'=*Acknowledge*

Der Treiber muss eine stabile Kommunikation mit den GTU-Boards ermöglichen und dabei möglichst wenig Prozessorlast erzeugen.

⁷PCI-Bus-Adresse: SMU(2), TMU(3-7)

⁸ASCII-Zeichen mit der Codenummer 6

Implementierung

Ein *Charakter Device* wird implementiert, der folgende Funktionen bereit stellt:

Init registriert ein CHAR-Device und legt die Datei 0 im Verzeichnis `/dev/uart_ioarea/` an. Diese Funktion wird aufgerufen, wenn das Kernel-Modul geladen wird.

Open öffnet das Device. Ein 32 kByte großer Speicher wird als Empfangspuffer reserviert und der *Interrupt-Handler* registriert. Die Hardware wird initialisiert und die Übertragungsrates für das UART-Device eingestellt. Um zu gewährleisten, dass maximal ein Programm auf den Treiber zugreifen kann (*Single Access Device*), wird das Abfragen und Inkrementieren des Zugriffszählers in einer *Critical Section*⁹ durchgeführt.

Write führt ein Reset des Sende- und Empfangs-FIFO durch, aktiviert den Interrupt des UART-Cores, kopiert das Kommando vom *User-* in den *Kernel-Space* und schreibt es dann in das Sende-FIFO.

Read fragt mit einer Rate von 50 Hz ab, ob Zeichen im Empfangspuffer stehen oder ein Acknowledge-Zeichen empfangen worden ist. Ist dies der Fall, wird das Acknowledge-Zeichen durch das 0-Zeichen ersetzt und der Empfangspuffer in den *User-Space* kopiert. Wenn kein Acknowledge innerhalb einer definierten Zeit nach dem letzten eingegangenen Zeichen empfangen wird, bricht die Funktion mit einem *Time-Out-Fehler* ab und der Inhalt des Empfangspuffers wird in den *User-Space* kopiert. Hierüber kann sicher gestellt werden, dass der Treiber nicht für immer in der Routine bleibt, wenn keine Antwort auf ein Kommando kommt. Vor dem Verlassen der Funktion wird der Interrupt des UART-Cores deaktiviert. Dadurch wird die Empfangsdatenverarbeitung der UART-Schnittstelle deaktiviert und erzeugt keine Prozessorlast, falls doch Zeichen über die Empfangsleitung im FIFO ankommen sollten. In diesem Fall läuft das FIFO nach 16 Zeichen über und ein Fehlerbit wird gesetzt. Da die FIFOs und damit auch die Fehlerbits zu Beginn der Write Funktion zurückgesetzt werden, stellt das Empfangen von Zeichen außerhalb des erlaubten Fensters kein Problem dar.

Close meldet den *Interrupt-Handler* ab und gibt den belegten Speicher frei.

Exit entfernt das Kernel-Modul und löscht den Eintrag in `/dev/uart_ioarea/`.

Die meiste Logik steckt in der Interrupt-Routine, die bei einem Interrupt aufgerufen wird. Der Inhalt des Empfangs-FIFO wird in den Empfangspuffer geschrieben und dabei auf Empfangsfehler und auf das Acknowledge-Zeichen geprüft. Die Routine bricht ab, wenn ein Acknowledge-Zeichen empfangen wird, der Puffer voll oder das Empfangs-FIFO leer ist. Das Schreiben inkrementiert einen Pointer, der auf die Schreibposition im Empfangspuffer zeigt.

⁹In einer *Critical Section* darf sich maximal ein Programm aufhalten. Sie wird atomar bearbeitet, was durch den Kernel sicher gestellt wird.

Diese Implementierung ist stark an die Problemstellung angepasst. Bei einer allgemeineren Implementierung würde der Empfangspuffer als Ringpuffer realisiert und kontinuierlich ausgelesen werden. Diese Methode wird nicht verwendet, um den Treiber einfach zu halten und *Context-Switches* zu vermeiden, die einen sehr großen Overhead darstellen. Ein Ringpuffer könnte aber als Erweiterung implementiert werden. Dadurch könnte das Erkennen des Nachrichtenendes in das *User-Programm* verlagert werden. Wenn das Auslesen des Ring-Puffers nicht schnell genug erfolgt, läuft dieser über und Daten gehen verloren. Die Einschränkung der gewählten Implementierung ist die maximale Antwortgröße von 32 kByte. Des Weiteren steht die Antwort erst beim Empfang des Acknowledge-Zeichens dem *User-Programm* zur Verfügung.

Stabilität und Performace

Der Treiber läuft seit März 2007 stabil ohne Probleme und befindet sich im produktiven Einsatz. Ein über Tage andauernder Stresstest hat keine Fehler ergeben. In diesem Test werden verschiedene Szenarien wie zu lange Antwortstrings, falsche Kommandos, Verhalten unter Volllast getestet. Die Prozessorlast beim allen Zugriffsszenarien auf das *Gerät* liegt bei unter 1%.

5.2.3 Software

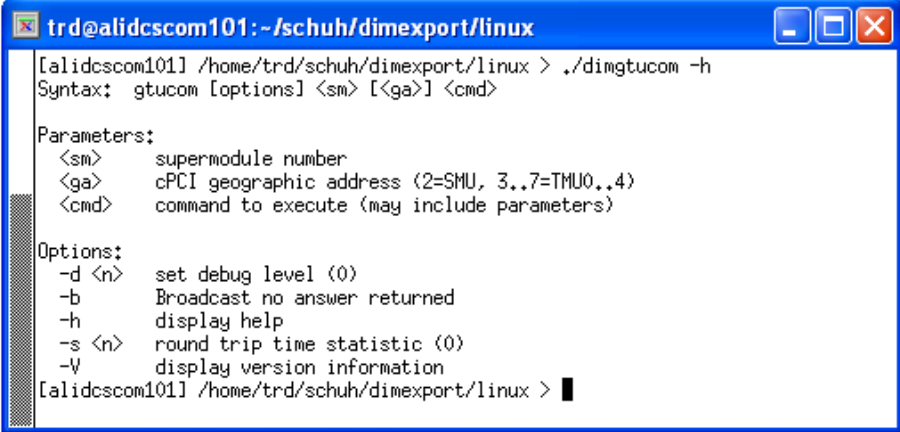
Die Kommandozeilenprogramme *gtucom* und *dimgtucom* werden für die Kommunikation mit den GTU-Boards verwendet. Die Konzepte dieser beiden Programme werden nun kurz vorgestellt.

GTUCOM Die lokale Variante (*gtucom*) läuft auf dem DCS-Board und greift direkt auf das *Device* zu. Es erhält als Parameter das auszuführende Kommando und gibt die Antwort auf der *stdout* aus. Dieses Programm kommt nur im Testbetrieb zum Einsatz, da im Normalbetrieb der im Abschnitt 5.3 vorgestellte *DIM-Server* das Device belegt. Die mittlere Zugriffszeit beträgt 0,09 s bei einem Kommando, das nur einen Wert ausgibt.

DIMGTUCOM Das Programm verwendet einen *RPC-Service*¹⁰ des *DIM-Servers*. Dieses Programm muss nicht auf dem DCS-Board, sondern kann auf jedem Rechner im Netzwerk ausgeführt werden, vorausgesetzt der *DIM-Server* auf dem ZielDCS-Board ist gestartet. Die Bedienung ist die selbe wie *gtucom*, nur dass als weitere Parameter die Supermodulnummer vor dem Kommando verwendet werden muss. Damit können zentral, von einem Rechner aus, Kommandos an alle GTU-Boards gesendet werden. Bei diesem Programm liegt die Zugriffszeit im Mittel bei 0,43 s, wenn es auf dem DCS-Board im

¹⁰Funktion wird in Abschnitt 5.3.1 vorgestellt

selben GTU-Segment gestartet wird. Eine ausführlichere Statistik ist in Kapitel 8 zu finden.



```
trd@alidcscom101:~/schuh/dimexport/linux
[alidcscom101] /home/trd/schuh/dimexport/linux > ./dimgtucom -h
Syntax: gtucum [options] <sm> [<ga>] <cmd>

Parameters:
<sm>    supermodule number
<ga>    cPCI geographic address (2=SMU, 3..7=TMU0..4)
<cmd>   command to execute (may include parameters)

Options:
-d <n>  set debug level (0)
-b      Broadcast no answer returned
-h      display help
-s <n>  round trip time statistic (0)
-V      display version information
[alidcscom101] /home/trd/schuh/dimexport/linux > █
```

Abbildung 5.3: Screenshot einer Konsole, in der die Hilfe von dimgtucom aufgerufen wird.

5.3 DIM-Server Software

Das *Distributed Information Management System* (DIM) dient zum Verteilen von Informationen über ein Netzwerk, unabhängig von Betriebssystem und Netzwerkarchitektur. Es wird am CERN ([Dis]) entwickelt und Versionen für alle gängigen Betriebssysteme (Unix, BSD, Windows) sind verfügbar. Der Quellcode liegt in den Programmiersprachen C/C++, Fortran oder Java vor. Diese Software wird im DCS-Netz am CERN verwendet, um Daten zwischen Hardware und Steuersystem zu transferieren. Dieses System wird auch in der GTU eingesetzt. Es werden Sensor- und Statusdaten der Boards für PVSS veröffentlicht und Steuerkommandos entgegen genommen. Diese Aufgabe wird von Serverclients auf den DCS-Boards der GTU übernommen.

Im folgenden wird auf die prinzipielle Funktionsweise von DIM eingegangen und anschließend auf die Implementierung für die DCS-Boards der GTU.

5.3.1 Funktionsweise von DIM

DIM basiert wie die meisten anderen Kommunikationssysteme auf dem Server-Client-Paradigma. Das Grundkonzept in DIM ist das *Service*-Konzept. Server stellen den Clients *Services* bereit. Ein *Service* ist ein Datensatz mit einem Typ, der eine Größe und einen eindeutigen Namen besitzt. *Services* werden normalerweise von einem Client nur einmal abgefragt (beim Start) und sie werden automatisch vom Server in festen Zeitintervallen oder bei Veränderung aktualisiert. Um Transparenz und schnelle Wiederherstellung nach einem Systemausfall oder den Umzug eines *Services* auf einen anderen Server zu ermöglichen, wird ein *Name-Server* (*DIM-DNS-NODE*) eingeführt. Dieser hält eine Tabelle der aktuell im System verfügbaren Server und ihren *Services* bereit. So benötigt der Client nur noch die Adresse des *Name-Server* und nicht mehr alle Serveradressen.

Die Server veröffentlichen (*publish*) ihre *Services* durch die Registrierung beim *Name-Server*. Clients melden sich bei einem *Service* an (*subscribe*), indem sie den *Name-Server* fragen, welcher Server den *Service* anbietet und dann den Server direkt kontaktieren.

Wenn ein Prozess abstürzt (Server oder *Name-Server*), werden alle Prozesse, die mit ihm verbunden gewesen sind, benachrichtigt und wieder mit ihm verbunden, sobald er wieder verfügbar ist. Diese Funktionalität ermöglicht es, im laufenden Betrieb einen Server auf eine andere Maschine umzuziehen, ohne das Gesamtsystem neu starten zu müssen.

DIM definiert drei Arten von *Services*. Der Kommando-*Service* ermöglicht das Senden von Daten vom Client zum Server ohne Rückgabewert. Um einen Rückgabewert zu erhalten, gibt es die Möglichkeit einen RPC¹¹-*Service* zu verwenden. Hier werden Daten an

¹¹Remote Procedure Call

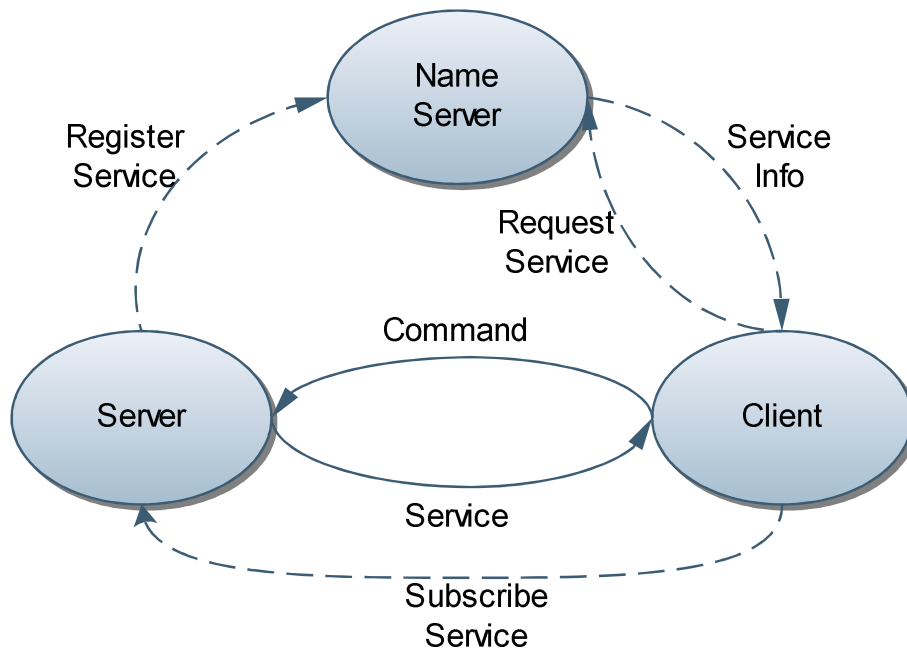


Abbildung 5.4: Zusammenspiel der einzelnen Komponenten von DIM. Server registrieren *Services* beim *Name Server*. Client fragt einen *Service* von einem Server ab und schicken Kommandos an diesen.

den Server übertragen und eine *Callback*-Funktion aufgerufen, deren Rückgabewert veröffentlicht wird. Der Daten-Service veröffentlicht einen Datenpunkt vom Typ Short, Int, Long, Float, Double, Char oder einer komplexen Struktur, die aus diesen aufgebaut ist. Dieser kann von allen Clients unabhängig abgefragt werden. Jeder Datenpunkt besitzt ein *Quality Flag* und einen *Time Stamp*, die gesetzt werden können.

5.3.2 Datenquellen und Auslesekanäle

Bevor das Strukturieren der Systeminformationen und Anlegen der Datenpunkte erläutert wird, muss auf die Datenquellen dieser Informationen eingegangen werden. Dies ist wesentlich für eine sinnvolle Strukturierung.

Eine der Informationsquellen ist die am Anfang des Kapitels vorgestellte UART-Kommunikationsschnittstelle. Über diese kann das DCS-Board den Status der SMU und der fünf TMUs abfragen, die zu diesem Segment gehören. Ein spezielles Kommando `dcs_dump` liefert alle systemrelevanten Daten in einem String mit festem Format.

Betriebsparameter wie Spannungen und Temperaturen der Boards können über das in Abschnitt 3.1.1 eingeführte I²C-Bussystem ausgelesen werden. Das DCS-Board verfügt

über einen I²C-Busmaster, von dem aus die Abfragen gestartet werden. Der Zugriff auf diesen wird über einen *Char-Device*-Treiber durchgeführt, in dem eine Befehlssequenz geschrieben und die Antwort zurückgelesen wird.

Diese Informationen werden nun über eine Vielzahl von Datenpunkten veröffentlicht, deren Struktur nun vorgestellt wird.

5.3.3 Datenpunkte und Kommandokanäle

Die Hauptaufgabe beim Anlegen der Datenpunkte und deren Benennung ist die Überlegung, welche Daten von Interesse sind, um dann ein System von Datenpunkten zu finden, das folgenden Anforderungen genügt:

- Übersichtlichkeit
- Zusammenfassen von zusammenhängenden Daten
- Berücksichtigung der verschiedenen Datenquellen bei der Strukturierung der Daten
- Aktualisierungsrate der einzelnen Daten
- Einhalten der Nomenklaturregeln des DCS [P⁺06]

Insgesamt werden etwa 100 Datenpunkte pro Server veröffentlicht, in denen über 500 Messwerte zusammengefasst sind. Von der gesamten GTU werden so fast 10.000 Messwerte in über 1800 Datenpunkten bereitgestellt.

Die Daten können in vier Klassen eingeteilt werden. In der ersten Klasse sind boardspezifische Daten enthalten, die sich während eines *RUNs*¹² nicht verändern. Hierzu gehören *Board-ID* und Versionsinformationen über das FPGA-Design. Die Versionsinformationen sind vor allem für die spätere Offline-Analyse interessant, um nachvollziehen zu können, welcher Trigger-Algorithmus beispielsweise in den TMUs verwendet wird. Diese Daten werden nur einmal beim Start über das *UART-Device* ausgelesen und jeweils in einem Datenpunkt veröffentlicht.

Die zweite Klasse besteht aus Sensordaten der Boards, die über das I²C ausgelesen werden. Hierunter fallen zwei Temperaturen und zehn Spannungen pro Board. Diese Daten werden in zwei Datenpunkten pro Board bereitgestellt.

Alle SFP-bezogenen Daten gehören in die dritte Klasse. Von den SFPs werden folgende Daten veröffentlicht: Betriebstemperatur und -spannung sowie optische Empfangsleistung, Verbindungsstatus und Fehlerzählerstände. Die Temperaturen und Spannungen werden von allen 12 SFPs pro Board in zwei Datenpunkten abgelegt. Ein weiterer Datenpunkt pro SFP fasst die Empfangsleistung und die übrigen Statusinformationen zusammen.

¹²Als RUN wird die Experimentierphase bezeichnet, in der Messdaten genommen werden

In der vierten Klassen befinden sich alle übrigen Daten, die teilweise boardspezifisch sind wie zum Beispiel der Triggerstatus und die Triggerstatistik oder die Verfügbarkeit der TMUs.

In jedem DIM-Server steht ein RPC-Service zur Verfügung. Mit diesem Kommandokanal werden Befehle an die PowerPC-Einheit geschickt. Die Antwort der PowerPC-Einheit wird als Rückgabewert zurück geschickt.

Die Nomenklatur folgt der ALICE-DCS-Integration-Guidline [P⁺06]. Als Präfix ergibt sich somit für alle Datenpunkte „trd_fero_gtusm_“. Danach folgt zweistellig die Supermodulnummer und „s“ bzw „tx“¹³ für die Boards. SFP-Daten haben ein „sfp“ im Namen, Kommandokanäle ein „com“. Für die Übersichtlichkeit wird die *Underscore*-Notation anstelle der *InterCap*-Notation verwendet. In der Tabelle 5.2 sind exemplarisch Datenpunkte und dessen Datenformat aufgeführt. Eine vollständige Übersicht befindet sich in der Tabelle A.1 im Anhang A.

Name	Typ	Beschreibung
trd_fero_gtusm_10_t0_tmon	Service	Temperaturen der TMU0 im SM 10
trd_fero_gtusm_03_s_svn	Service	Versionsinformation der SMU im SM 3
trd_fero_gtusm_18_com	RPC	Kommandokanal der TGU

Tabelle 5.2: Beispieldatenpunkte der GTU für Temperatur und Versionsinformationen und einem Kommandokanal mit Rückgabewert.

5.3.4 Programmfluss des DIM-Servers

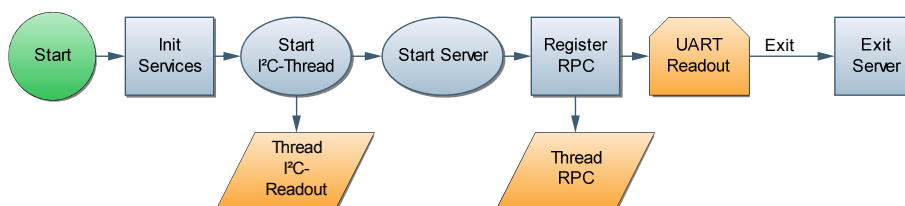


Abbildung 5.5: Flussdiagramm des DIM-Server Programms

Für die Strukturierung des Programmablaufs gibt es folgende Bedingungen, die berücksichtigt werden müssen:

- **langsamer Hardware Zugriff:** Die Zugriffszeiten auf das I²C- und UART-Device sind im Bereich von 10 ms.

¹³x: 0..4

- **kein Device-Sharing:** Es kann jeweils nur ein Prozess auf das jeweilige Device zugreifen.
- **asynchrone Anfragen:** Es werden asynchron Kommandos empfangen, die verarbeitet werden müssen.

Um all diesen Anforderungen zu genügen, wird ein Programmfluss gewählt, der schematisch in Abbildung 5.5 dargestellt ist. Das Programm lässt sich in zwei Phasen gliedern: Initialisierungs- und Arbeitsphase. In der Initialisierungsphase wird überprüft, welche Hardware zur Verfügung steht, dementsprechend die DIM-Services registriert und der DIM-Server gestartet. Die Arbeitsphase lässt sich ihrerseits in drei Teile aufteilen, zwei für das periodische Auslesen über I²C bzw. UART und ein weiterer Teil, der asynchron Kommandos an die PowerPC-Einheit über das UART absetzt.

5.3.5 Anforderungen und Probleme bei der Implementierung

In den vorherigen Abschnitten sind die Strukturierung der Datenpunkte und Kommandokanäle, der Programmfluss und das allgemeine Funktionsprinzip von DIM dargestellt.

Bei der Implementierung erweist sich die geringe Rechenleistung des *ARM-Prozessors* in Kombination mit dem Zugriff auf das I²C-Device als Hauptproblem. Beim Zugriff auf dieses Device liegt die Prozessorlast bei 100%, was die Stabilität des Gesamtsystems gefährdet, da hierdurch auch andere Dienste blockiert werden.

Die hier vorgestellte Lösung berücksichtigt alle Anforderungen, die in den vorherigen Abschnitten dargestellt sind und versucht, eine möglichst hohe Ausleserate bei geringer Prozessorlast zu realisieren. Zunächst wird auf die Strukturierung der Auslesevorgänge eingegangen und dann auf die dabei auftretenden Probleme und deren Lösungsansätze.

Aufgrund der großen Antwortzeiten sind die Zugriffe auf das UART- und I²C -Device langsam. Da die Zugriffe auf diese beiden Geräte komplett unabhängig voneinander sind, können diese parallelisiert werden, was durch die Verwendung von *Threads*¹⁴ realisiert wird. Das Auslesen der Sensoren über I²C wird in einem Thread ausgelagert. Auf das UART-Device greifen sowohl die Ausleseroutine als auch die asynchronen RPC-Kommandos zu. Um einen wechselseitigen Ausschluss zu gewährleisten, kommt ein *MUTEX*¹⁵ zum Einsatz.

Für die Implementierung der DIM-Funktionen wird der C++-Quellcode verwendet. Die Protokollierung der Fehler und sonstiger Zustände des Programmes werden von einem *syslog*-Dienst übernommen, der von einer Betriebssystembibliothek bereitgestellt wird.

¹⁴Leichtgewichtiger Prozess

¹⁵*Mutual Exclusion* bezeichnet ein Verfahren, mit denen verhindert wird, dass nebenläufige Prozesse bzw. Threads gleichzeitig auf Daten zugreifen und so unter Umständen inkonsistente Zustände herbeiführen.

5.3.6 Performance, Probleme und Erweiterungsmöglichkeiten

In der jetzigen Implementierung erzeugt der DIM-Server eine Prozessorauslastung von fast 100%, wenn der Zugriffe auf das I²C-Device nicht durch Wartezyklen verlangsamt wird. Dieses Phänomen ist etwas verwunderlich, da die eigentlichen I²C-Anfragen in Hardware durchgeführt werden und der Prozessor nur für den Zugriff auf Register des Gerätes benötigt wird. Dies legt die Vermutung nahe, dass der Treiber für das I²C-Device das eigentliche Problem darstellt. Diese Vermutung wird durch die Tatsache, dass die Zeit, die in Systemroutinen verbracht wird, sinkt, wenn zwischen Schreib- und Lesezugriffen Wartezyklen in Form von *sleeps* eingefügt werden. Der Treiber verwendet vermutlich ein blockierendes *Busy-Wait*, um auf die Antwort des I²C-Busses zu warten, anstelle eines zyklischen Abfragen mit nichtblockierenden *Sleeps*. Durch Einfügen von Wartezyklen beim Auslesen über I²C wird die Prozessorlast auf 30% reduziert. Die Auslesezeit erhöht sich dadurch von 4 s auf 6 s. Dies ist nur eine temporäre Lösung. Die bessere und gewünschte Lösung ist die Überarbeitung des I²C-Treibers, was aber ein eigenständiges Projekt darstellt.

Weitere Funktionen wie das Neustarten des DIM-Servers via eines Kommandokanals oder einer Hardwareerkennung während des Betriebes und der daraus folgenden Anpassung der Daten-Services sind denkbar. Eine andere Erweiterung ist das Programmieren der FPGAs, das via DIM gestartet wird. Hierfür müsste die Programmiersoftware in die DIM-Server-Software integriert oder über einen Systemaufruf ausgeführt werden. Hierfür gilt es noch zu klären, ob die *Bit-Files* ohne Probleme übertragen werden können oder über ein *Share*-Verzeichnis zur Verfügung gestellt werden.

Mit der in diesem Kapitel vorgestellten Kommunikationsschnittstelle ist es nun möglich, zentral über das Netzwerk die gesamte GTU zu überwachen und zu steuern. Wie das bewerkstelligt werden kann, wird nun im folgenden Kapitel dargestellt.

6 Visualisierung und Monitoring

Das Kapitel beschäftigt sich mit Überwachung und Steuerung der GTU im späteren Experimentalbetrieb. Das hierfür entwickelte System, das eine große Anzahl von Messdaten (über 9.000) verwalten und geeignet darstellen kann, untergliedert sich in zwei Aufgabenkomplexe. Der erste befasst sich mit der Überwachung und Steuerung der GTU, der zweite mit der Einbindung der GTU in das Gesamtsteuersystem des Detektors.

6.1 Monitoring mit PVSS und JCOP-Framework

Wie schon im Abschnitt 2.2.2 kurz dargestellt, wird die gesamte Experimentsteuerung mit PVSS und dem darauf aufgesetzten JCOP-Framework realisiert. Dieser Abschnitt gibt einen Einblick in die Software, da dieses Verständnis wesentlich für die Strukturierung des PVSS-Systems ist. Danach wird auf die Entwicklung eines PVSS-Systems für die GTU nach den Richtlinien des Frameworks (siehe [Tea01]) eingegangen.

6.1.1 Aufbau von PVSS und JCOP

Wie schon in Abschnitt 2.2.2 vorgestellt, ist PVSS ein SCADA-System, das verwendet wird, um eine Verbindung zu Hardware- und Softwareeinheiten herzustellen, Messdaten auszulesen und für die Systemsteuerung zu analysieren. Die Software ist aus folgenden Komponenten aufgebaut:

- Laufzeitdatenbank
- Archivierung von Daten
- Alarmerzeugung und Management
- Graphischer Editor (GEDI)
- Skriptsprache
- Graphisches Parametrisierungstool (PARA)
- Treiber

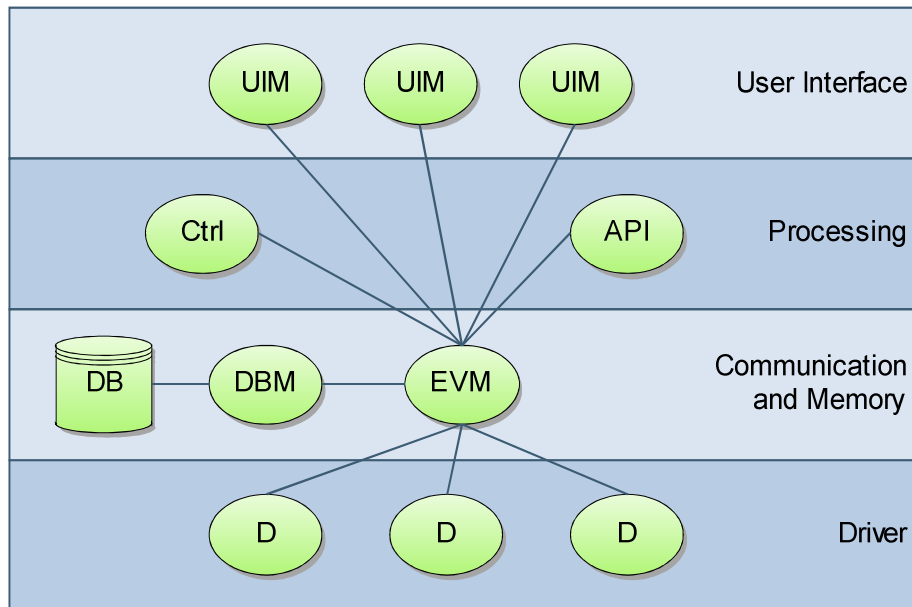


Abbildung 6.1: Die verschiedenen Schichten von PVSS. Die unterste Schicht bilden die Treiber. Darauf baut die Datenverwaltung und -management auf. Kontrollskripte und Benutzerschnittstellen greifen auf die Daten zu.

PVSS hat ein stark modulares, verteiltes Design. Die einzelnen Module der Software werden *Manager* genannt und sind unabhängige Prozesse, die verschiedene Aufgaben übernehmen und mit einem speziellen Protokoll über TCP/IP miteinander kommunizieren. Die Manager verbinden sich über den *Event Manager* mit den Daten. Der *Event Manager* ist das zentrale Element des Systems. Dieser sendet nur bei Veränderungen die Daten an die verbundenen Manager. Die Treiber sind so konfiguriert, dass sie nur Veränderungen propagieren. In einem gut konfigurierten System ist in einem stationären Zustand kein Datenverkehr vorhanden. Wie in Abbildung 6.1.1 dargestellt, können die Manager in verschiedene Schichten eingeteilt werden. Die Aufgabe der einzelnen Manager in einem Gesamtsystem ist in Tabelle 6.1 aufgeführt. Ein PVSS-System besteht aus einer Daten-

Manager	Funktion
<i>Event</i> (EVM)	verantwortlich für die gesamte Kommunikation
<i>DataBase</i> (DBM)	Schnittstelle zur Datenbank
<i>User Interface</i> (UIM)	Benutzerschnittstelle
<i>Ctrl</i> (Ctrl)	Hintergrundprozesse
<i>API</i> ¹ (API)	Zugriff von anderen Programmen auf PVSS
<i>Drivers</i> (D)	Schnittstelle zu den zu kontrollierenden Einheiten
<i>Distributed</i> (Dist)	Kommunikation zwischen mehrere Systemen

Tabelle 6.1: Übersicht über die Funktion der wichtigsten Manager in PVSS

bank, einem oder mehreren EVMs und einer beliebigen Anzahl von Treibern und Benutzerschnittstellen. Das System kann als verteiltes System auf mehreren Rechnern laufen und wird dann als *distributed* bezeichnet. Die Benutzerschnittstelle ist aus *Panels*² und *Scripts* aufgebaut. Diese können mit GEDI entwickelt werden. Für eine tiefer gehende Einführung sei auf [Sas04] verwiesen.

Das Ziel des *Joint Controls Project Frameworks* (JCOPFw) ist die Bereitstellung einer breiten Unterstützung für die Entwicklung des DCS in den einzelnen Subdetektoren, um den Entwicklungsaufwand der einzelnen Gruppen so zu minimieren. Bei der Implementierung werden PVSS und SMI++³ verwendet. Das System ist modular aufgebaut. Das so genannte Basispakete (fwCore) verwaltet alle übrigen Pakete, die hierüber nachinstalliert oder entfernt werden können. In Tabelle 6.2 sind Pakete aufgelistet, die später Verwendung finden. Darüber hinaus gibt es auch die Möglichkeit selbst Pakete zu erstellen. Das

Paket	Funktion
fwTrend	Darstellen und Verwalten von Graphen und Histogrammen
fwAlert	Definition von Alarmzuständen und Verwaltung von diesen
fwFSM	Finite State Machine Erweiterung
fwDIM	Schnittstelle für DIM

Tabelle 6.2: Auswahl von Frameworkpaketen, die in diesem System verwendet werden. Eine Liste aller verfügbaren Pakete ist unter [CERb] zu finden

im weiteren Verlauf des Kapitels vorgestellte PVSS-System wird in Form eines solchen in das bereits am CERN laufende System integriert.

6.1.2 Datenbank und DIM-Schnittstelle

In diesem Unterabschnitt wird darauf eingegangen, welche Datenpunkttypen für die GTU in der PVSS-Datenbank angelegt werden und wie diese struktuiert sind. Die Hauptaufgabe hierbei ist es, eine möglichst gut gegliederte Struktur zu erstellen. Hierbei müssen folgende Punkte beachtet werden:

- Abbildung von DIM-Datenpunkten auf Datenpunkte in der Datenbank
- Logisch zusammengehörende Daten in passenden Strukturen in der Datenbank ablegen
- Vermeidung von Redundanz innerhalb der Datenpunkttypen
- Integrationsrichtlinien [P⁺06]

²Fenstersystem

³Finite State Machine (FSM) Toolkit, das bereits für das DELPHI Experiment entwickelt wurde.

Aufbau der Datenpunkttypen Die Daten werden in hierarchischen Strukturen, so genannten *Datenpunkttypen* in der Datenbank abgelegt. Wenn dies mit einer klassische Datenbank verglichen werden soll, entspricht der Datenpunkttyp der Tabellendefinition. Die Elemente der Tabelle sind in diesem Fall die Datenpunkte. Es gibt einen ausgezeichneten Datenpunkt (*Master*). Wenn in diesem Eigenschaften geändert werden, werden die Änderungen auf alle Datenpunkte des selben Typs übernommen. Ein Datenpunkttyp ist aus weiteren Elementen aufgebaut. Jedem Element ist ein Datentyp weitere Attribute wie Gültigkeitsbereich oder Archivierung zugeordnet. Eine spezielles Attribut ist das so genannte *Alert-Handling*. Hier können Wertebereiche definiert werden, denen Fehlerzustände zugeordnet werden. In Abhängigkeit des Zustandes können Aktionen ausgelöst werden. Es gibt eine Klassen von Fehlerzuständen, denen jeweils eine Priorität zugeordnet ist.

Angelegte Datenpunkttypen Für die GTU werden mehrer Datenpunkttypen angelegt, die unterschiedliche Funktionalitäten implementieren.

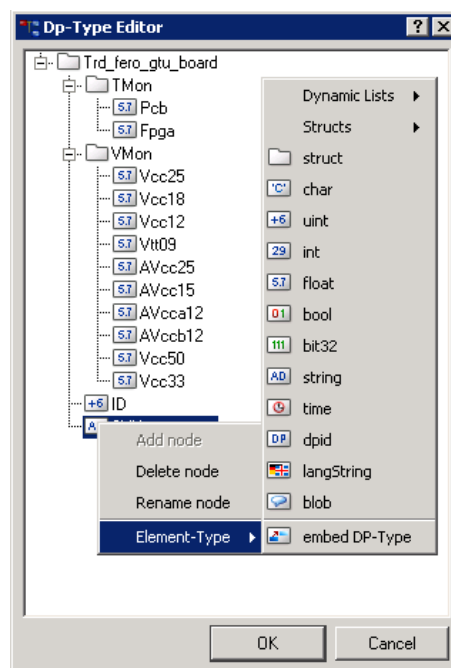


Abbildung 6.2: Screenshot aus PVSS PARA, der die Struktur des Datenpunkttyps TRD_fero_gtu_board zeigt.

Im Datenpunkttyp TRD_fero_gtu_board werden alle boardspezifischen Daten wie Temperaturen, Spannungen oder IDs abgelegt. Von diesem Typ werden 109 Datenpunkte angelegt - einen für jedes Board. Für die Temperaturen und Spannungen sind Schwellwerte für das Alarm-Management definiert, um Fehler in den Betriebsparametern zu erkennen. Die DIM-Services *_tmon und *_vmon werden direkt auf die Datenpunktelemente

*.TMon und *.VMon abgebildet. Statusbits zeigen an, ob das Datenpunktelement mit den DIM-Daten synchronisiert ist. Eine komplette Auflistung aller Datenpunktelemente und die Zuordnung zu den DIM-Services ist im Anhang A zu finden.

Der Datenpunkttyp TRD_fero_gtu_sfp fasst alle SFP, MGT und Eventempfangsstatus bezogenen Daten eines Stacks zusammen. Aus Gründen der Übersichtlichkeit wird davon abgewichen, innerhalb des Datenpunkttyps Redundanz zu vermeiden. Es gibt daher 12, in Bezug auf die Struktur, identische Datenpunktelemente, die den SFP- und MGT-Status der einzelnen Module enthalten.

Alle für die Kommunikation mit den GTU-Segmenten relevanten Daten werden in dem Datenpunkttyp TRD_fero_gtu_com zusammengefasst. Hiervon werden 19 Datenpunkte instanziiert - 18 für die GTU-Segmente und ein weiterer für die TGU.

Der Datenpunkttyp TRD_fero_gtu_config, von dem es nur eine Instanz gibt, fasst allgemeine Konfigurationen, Schwellwerte und Grenzwerte zusammen.

Auf der Grundlage dieser Daten bauen die im nächsten Abschnitt vorgestellten Panels auf. Zuvor wird kurz auf das Empfangen und Senden von Daten über DIM eingegangen.

DIM-Schnittstelle Für das DIM-Protokoll gibt es ein Frameworkpaket, das einen Treiber und eine Benutzerschnittstelle bereitstellt. So können über ein Panel ein oder mehrere Manager konfiguriert und gestartet werden. Ein Manager übernimmt die Synchronisation zwischen einem DIM-Datenpunkt und der Datenbank. Die Zuordnungen von DIM-Daten und Datenpunkten in der Datenbank werden in einem Datenpunkt gespeichert, der beim Starten des Managers ausgelesen wird.

6.1.3 Panels

Im Folgenden wird zunächst auf den allgemeinen Aufbau eines Panels eingegangen und anschließend die für die GTU entwickelten Panels vorgestellt.

Aufbau und Struktur von Panels In einem Panel können Daten aus der Datenbank visualisiert und manipuliert werden. Dies wird über verschiedene graphische Objekte wie beispielsweise Buttons oder Textfelder realisiert. Das Auslesen und Schreiben der Daten in der Datenbank übernehmen Skriptfunktionen. Die Übergabewerte für die Funktionen können fest in den Code geschrieben sein oder per Parameter beim Aufruf des Panels mit übergeben werden. Mit dieser Option lassen sich so genannte generische Panels erzeugen, die in anderen Panels als Objekte eingebunden werden. Ein Panel kann so aus vielen anderen Panels bestehen.

Panels der GTU Im allgemeinen Operator-Panel des TRD im späteren Betrieb wird die GTU nur durch ein Symbol dargestellt, das den den FSM-Status der GTU widerspiegelt. Wenn Unregelmäßigkeiten auftreten und für weitere Informationen kann das Übersichtspanel der GTU aufgerufen werden. In diesem Panel sollen daher alle für den Betrieb relevanten Daten in einer verständlichen und übersichtlichen Form dargestellt sein. Für die Darstellung des physikalischen Status der GTU sind die drei Racks sche-

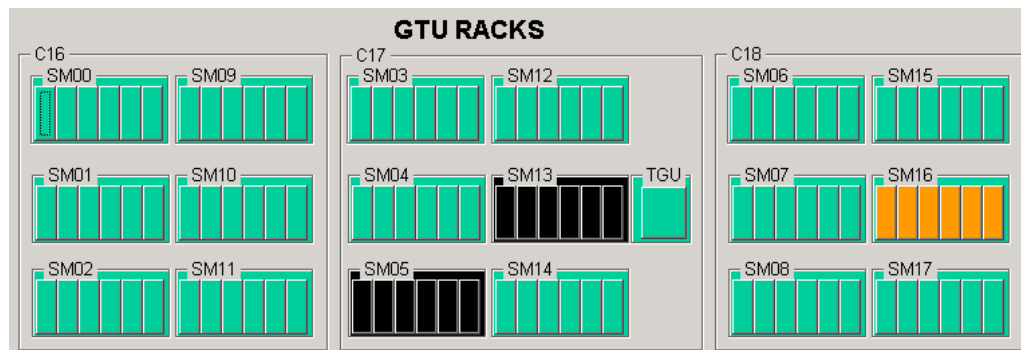


Abbildung 6.3: Screenshot des Rack Monitor Panels, das aus GTU-Segment-Panels aufgebaut ist. Jedes Board ist durch ein Rechteck dargestellt, dessen Farbe den Status des Boards anzeigt, der sich aus Temperatur-, Spannung- und Online-Status ergibt. Die Farbcodierung ist wie folgt: grün: alles Ok, orange: Fehler (alle Boards aus SM 16 - Fehler beim Auslesen der Sensoren), schwarz: offline (SM 05,13 - noch nicht eingebaut)

matisch nachgebaut (siehe Abbildung 6.1.3). Hier wird ein Überblick gegeben, welche Segmente online sind und wo Probleme auftreten. Eine direkte Anzeigen aller Betriebsparameter ist aus Platzgründen und der Übersichtlichkeit halber nicht erstrebenswert. Durch klicken auf ein Board, werden alle Betriebsparameter dieses Boards in einem neuen Panel angezeigt.

Die Möglichkeit, die optischen Links der einzelnen Segmente bzw. der TMUs gezielt an- und abschalten zu können, wird in einem weiteren Panel realisiert.

Mit Hilfe von Histogrammen können Temperatur- und Spannungsverteilung der Boards dargestellt werden. Dies ist hilfreich, um Boards zu finden, deren Betriebsparameter sich noch innerhalb der erlaubten Grenzen befinden, jedoch von denen der anderen Boards abweichen. Die Kommunikation mit den einzelnen Boards ist durch ein Panel realisiert, das im Übersichtspanel integriert ist. Alle weiteren Panels und deren Funktion sind in der Tabelle B.1 im Anhang aufgelistet, ebenso die Abhängigkeiten der einzelnen Panels.

6.1.4 Stand der Implementierung

Alle physikalischen Daten der GTU-Boards, die über die oben genannten Panels dargestellt werden können, sind im PVSS-System verfügbar. Das Alarmmanagement für die Temperaturen und Spannungen ist implementiert und getestet. Hierauf wird in Kapitel

8 nochmals näher eingegangen. Ein Panel, das einen Betriebsparameter aller Boards in einem Histogramm darstellt ist fertig. Damit steht ein PVSS-System für die GTU zur Verfügung, das den physikalischen Status überwacht, was in der momentanen Phase des Projektes sehr wichtig ist. Bei einem Ausfall der Kühlung zum Beispiel muss das System sofort herunter gefahren werden (siehe Abschnitt 8.2), was nur geschehen kann, wenn eine funktionierende Überwachung vorhanden ist.

Die Datenpunkte und deren Darstellung, die den Status des Triggers und Auslesepfades wiedergeben fehlen noch. Welche Daten hierfür sinnvoll sind, ist momentan noch nicht spezifiziert. Darüber hinaus fehlt noch die Möglichkeit, feste Konfigurationen für verschiedene Betriebsmodi der GTU in einem Panel zusammenzustellen. Hierfür stehen die Parametersätze bisher nicht fest, da die Entwicklung in diesem Bereich noch nicht abgeschlossen ist. Die Archivierung relevanter Systemdaten ist nicht implementiert, da die Standardarchive von PVSS nicht für so viele Datenpunkte ausgelegt sind. Daher muss ein neuer Archivmanager konfiguriert werden, was nicht trivial ist. Ein weiterer Punkt, der bisher außer Acht gelassen wird, ist die Implementierung der Benutzerrechte, die im späteren Betrieb notwendig sind. In Abhängigkeit der Rechte des Benutzers stehen diesem bestimmte Funktionen zur Verfügung oder werden ausgeblendet. Dies ist in der momentanen Projektphase noch uninteressant, da sich nur eine Person einloggen kann.

6.2 Einbindung der GTU in das DCS mit FSM

In diesem Abschnitt wird zuerst noch einmal auf die Struktur des DCS und die Implementierung mit FSM eingegangen und danach auf die Einbindung der GTU in dieses System.

6.2.1 Die allgemeine Kontrollhierarchie der FSM

Die Kontrolle ist im DCS, wie in Kapitel 2 schon dargestellt, hierarchisch strukturiert und über eine Baumstruktur implementiert. Befehle gehen von der Wurzel in Richtung der Blätter (nach unten) und Messwerte nehmen den umgekehrten Weg. Es gibt drei Typen von Einheiten in dieser Struktur. Allen Einheiten kann ein definierter FSM-Zustand zugeordnet werden.

Die *Device Unit* (DU) implementiert eine direkte Schnittstelle zur Hardware. Sie ist in der Lage, ankommende Befehle direkt zu einem Kommando für die Hardware zu übersetzen. Der Zustand der DU leitet sich direkt aus Werten von der Hardware ab und folgt einem Zustandsdiagramm, beinhaltet aber kein Kontrollprogramm, um komplexe Entscheidungen zu treffen.

Die *Logical Unit* (LU) ist ein Objekt, das mehrere DUs zusammenfasst. Sie kann entweder an- oder abgeschaltet sein, aber nie alleine arbeiten. Der Zustand ergibt sich aus dem

Zustand ihrer Kinder. Eine LU wird verwendet, um in niederen Schichten des Baumes mehre DU zu gruppieren oder auch andere LU zusammenzufassen.

Die *Control Unit* (CU) kann eine oder mehre LUs oder DUs kontrollieren. Sie kann in die Hierarchie eingebunden sein oder unabhängig operieren. Eine CU besitzt ein Kontrollprogramm, das in der Lage ist in Abhängigkeit von seinem Zustand komplexe Entscheidungen zu treffen. Die CU kann auch andere CUs integrieren⁴.

Es gibt prinzipiell zwei Möglichkeiten die FSM-Hierarchie zu implementieren:

- Eine systemorientierte Hierarchie, die auf den Einzelsystemen wie Hochspannung, Kühlung usw. basiert.
- Eine detektororientierte Hierarchie, die sich in Supermodule, Module, und Sektoren untergliedert.

Bei einer detektororientierten Implementierung kann man einzelne Module separat betreiben, was im Fall der GTU auch angewendet wird. Die einzelnen Supermoduleinheiten arbeiten unabhängig voneinander und sollen daher auch unabhängig voneinander steuerbar sein. Eine systemorientierte Hierarchie ist nicht sinnvoll, da es keine Unabhängigen Untersysteme in der GTU gibt. Während der Startphase des Experiments werden auch noch nicht alle Supermodule installiert sein, was ein weiterer Aspekt ist, sich für diese Struktur zu entscheiden.

6.2.2 Die Kontrollhierarchie der FSM in der GTU

Der Graph 6.2.2 zeigt die hierarchische Strukturierung der FSM in der GTU. Die Struktur basiert darauf, dass das Auslesen der Supermodule unabhängig voneinander ist und innerhalb eines Supermoduls stackweise orientiert ist. Eine DU repräsentiert ein Board. Es gibt drei verschiedene CUs. Eine für die GTU allgemein, eine für den Trigger, eine für das Auslesen jedes Supermoduls.

6.2.3 Stand der Implementierung

Die Integration der GTU in den FSM-Baum ist nicht abgeschlossen, da die oben vorgestellte FSM-Hierarchie noch implementiert werden muss. Des Weiteren fehlen noch die Zustände der FSM für die einzelnen CUs. Hierfür gibt es ein allgemeines Schema⁵, das aber nicht direkt auf die GTU anwendbar ist. Nur ein Teil der Zustände ist für die GTU sinnvoll. Eine Auswahl an FSM-Zuständen für die GTU ist in Abbildung 6.2.3 dargestellt.

⁴Eine ausführlichere Darstellung und Definition ist in [Cla01] zu finden.

⁵Siehe [Mar07]

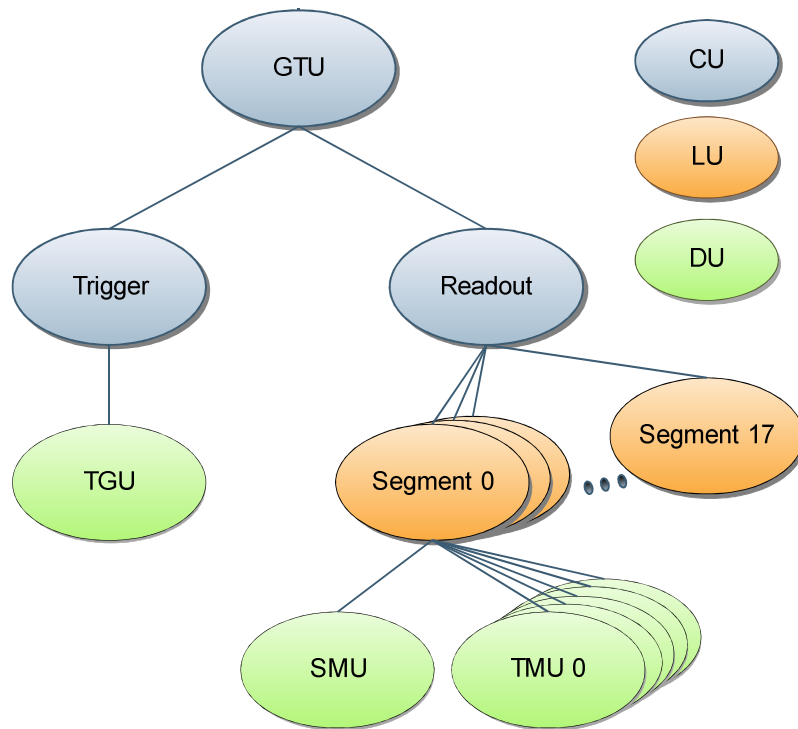


Abbildung 6.4: Aufbau der FSM-Hierarchie innerhalb der GTU. Sie teilt sich in die 2 Untereinheiten Trigger und Readout auf. Der Unterbaum Readout ist 18 Untereinheiten unterteilt für die 18 Supermodule.

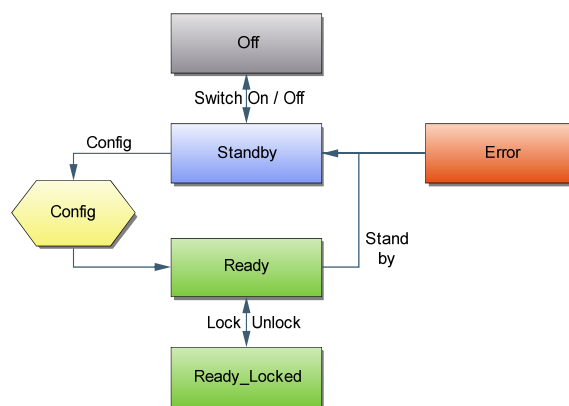


Abbildung 6.5: Übergangsgraph der FSM-States in der CU der GTU. Die Zustände OFF, Standby, Config, Ready, Ready_Locked werden nacheinander durchlaufen. Der Config-Zustand ist ein Übergangszustand, der ohne Aktion verlassen wird. Der Fehlerzustand wird im Normalbetrieb nicht erreicht.

7 Der SD-Karten-Controller

Die GTU-Boards verfügen über einen Kartenschacht für SD-Karten¹ [SD]. Die SD-Karte kann als nichtflüchtiger Speicher verwendet werden, um Konfigurationsdaten für die Boards darauf abzuspeichern. Es können zum Beispiel Simulationsdaten für den TRD darauf abgelegt werden. Diese Daten können dann in den SRAM geladen und für Tests der Ausleseketten und der nachfolgenden Triggerstufen und Analysetools verwendet werden. Eine andere Möglichkeit für den Einsatz einer SD-Karte ist die Verwendung als *Root File System* für eine Linux-Installation. T. Gerlach bearbeitet dieses Thema in seiner Diplomarbeit [Ger].

In diesem Kapitel geht es um die Implementierung eines SD-Karten-Controllers für den OPB². Einleitend wird auf den Aufbau einer SD-Karte, deren Schnittstelle und Kommunikationsprotokolle eingegangen. Anschließend wird die logische Struktur des Controllers diskutiert und schließlich die Implementierung in VHDL vorgestellt. Als Referenz für die Implementierung wurde das *SanDisk SD Card Produkt Manual*³ [San04] verwendet. Für ausführlichere Informationen sei darauf verwiesen.

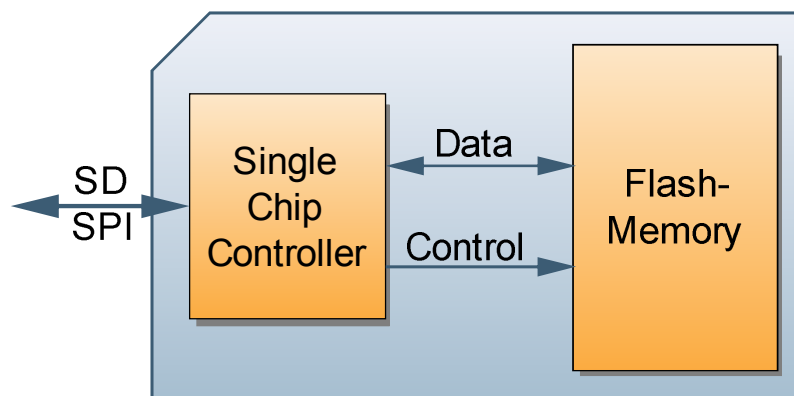


Abbildung 7.1: Schematischer Aufbau einer SD-Karte. Sie besteht aus einem Controller und Flash-Speicher

¹Secure Digital Memory Card

²On-Board Peripheral Bus

³Mit der *SD Card Physical Layer System Specification* Version 1.10 konform

7.1 Aufbau einer SD-Karte und ihrer Schnittstelle

Eine SD-Karte ist ein *Flash*⁴-Speicher mit integriertem Controller, der die Karte steuert und den Zugriff auf den Speicher durchführt. Die gesamte Kommunikation wird von diesem Controller abgewickelt. Der Controller ist eine *Finite State Machine* mit einer komplexen Struktur, auf die im Folgenden nur soweit eingegangen wird, wie sie für die konkrete Implementierung einer Kommunikationsschnittstelle benötigt wird. Der Zustand des Controllers und kartenspezifische Daten sind in einer Reihe von Konfigurationsregistern abgelegt, die in Tabelle 7.1 aufgelistet sind.

Name	Breite [Bit]	Beschreibung
CID	128	Individuelle Nummer zur Identifikation
RCA	16	Lokale Systemadresse der Karte
CSD	128	Informationen über den Status der Karte
SCR	64	Informationen über kartenspezifische Funktionen
OCR	32	Arbeitsspannungsbereich der Karte

Tabelle 7.1: Konfigurationsregister einer SD-Karte

Die Schnittstelle der SD-Karte besteht aus neun Pins, deren Belegung teilweise vom verwendeten Kommunikations-Protokoll abhängt. Die Belegung ist in Tabelle 7.2 dargestellt.

Pin	Name	Typ	Beschreibung
1	CS	In	Chip Select (active low)
2	DataIn	In	Host-to-Card: Kommandos und Daten
3	VSS1		Masse
4	VDD		Spannungsversorgung
5	CLK	In	Takteingang
6	VSS2		Masse
7	DataOut	Out	Card-to-Host: Daten und Status
8	RSV4		Reserviert
9	RSV5		Reserviert

Tabelle 7.2: Interface-Signale der SD-Karte im SPI-Protokoll

Der Zugriff auf die Daten ist blockorientiert. Die Standardblockgröße ist 512 Byte bei Speicherkarten mit bis zu 2 GByte Speicher⁵. Es gibt zwei Kommunikationsprotokolle - das *SD-CARD*-Protokoll und das *SPI*⁶-kompatible Protokoll. Das *SD-CARD*-Protokoll

⁴Flash-Speicher (EEPROM) sind digitale Speicherchips, die nicht byteweise gelöscht werden können, sondern nur in größeren Blöcken

⁵Bei Speicherkarten mit mehr als 2 GByte wird eine größere Blockgröße verwendet und auch eine andere Adressierung.

⁶Serial Peripheral Interface

ist das eigentliche Kommunikationsprotokoll, das alle Funktionalitäten unterstützt, aber auch eine gewisse Komplexität mit sich bringt. Das im nächsten Abschnitt genauer vorgestellte SPI-kompatible Protokoll ist einfacher struktuiert als das *SD-CARD*-Protokoll und für eine erste *Proof-of-Concept*-Implementierung besser geeignet. Im Folgenden werden die jeweiligen Protokolle nur noch mit SPI- oder SD-Modus bezeichnet. Die Gründe für die Verwendung des SPI-Modus sind folgende:

Statt bidirektionaler Leitungen wie im SD-Modus werden unidirektionale Leitungen verwendet. So muss keine Arbitrierung für diese Leitungen implementiert werden. Die SD-Karte antwortet immer auf ein Kommando, was im SD-Modus nur bei gültigem Kommando⁷ der Fall ist. Die Verwendung der CRC-Prüfsumme ist nur optional und muss daher zunächst nicht implementiert werden. Eine spätere Erweiterung für den SD-Modus wird in der Implementierung nach Möglichkeit schon berücksichtigt, worauf noch detaillierter eingegangen wird.

7.1.1 SPI-Modus

Das *Serial-Peripheral-Interface*-Protokoll ist seriell und byteorientiert. Alle Kommandos und Datenblöcke sind aus Bytes aufgebaut und *byte aligned* zum CS-Signal.

Die SPI-Nachrichten werden aus Kommando-, Antwort- und Datenblock-Token aufgebaut. Der Host (*Master*) kontrolliert die Kommunikation mit der Karte (*Slave*). Jeder Transaktionszyklus wird vom Host gestartet, indem dieser das CS-Signal auf *low* zieht und nach acht Takten⁸ ein Kommando-Token sendet. Je nach Kommando antwortet die Karte in einem vom Kommando abhängigen Zeitintervall mit einem Antwort- oder Daten-Token. Nach Empfang des Antwort-Tokens wird das CS-Signal wieder auf *high* gezogen und der Transaktionszyklus ist beendet.

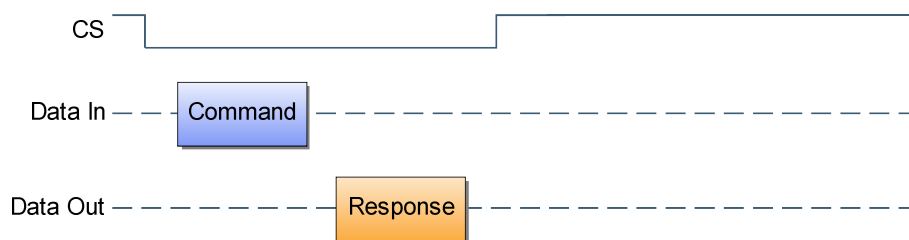


Abbildung 7.2: SPI-Kommando-Zyklus: Host setzt CS-Signal und sendet ein Token. Karte antwortet mit einem Token. Host nimmt CS-Signal zurück.

⁷Ein gültiges Kommando heißt, dass der Controller dieses erwartet und dass die CRC-Prüfsumme korrekt ist

⁸laut Spezifikation nach $8n, n \in \mathbb{N}$ Takten.

Kommando-Token Es gibt insgesamt 60 Kommandos, die nach ihrer Funktion in 11 verschiedene Klassen eingeteilt werden. Die Liste aller Kommandos ist in [San04] zu finden. Jedes Kommando ist 6 Byte lang und ist wie in Tabelle 7.3 aufgebaut. Das *Most Significant Bit* wird dabei immer als erstes gesendet. Jedem Befehl ist ein Antwortformat zugeordnet, das als Antwort erwartet wird.

Byte	0	1 - 4	5
Token	01CMD	Parameter	CRC1

Tabelle 7.3: Codierung der Kommando-Token. CMD ist das Kommando, codiert in 6 Bit. Der Parameter ist ein 32-Bit-Wort.

Antwort-Token Es gibt verschiedene Antwortformate, die wie folgt unterteilt werden:

- Reine Antwort-Token (R1-R6), die direkt als Antwort auf ein Kommando gesendet werden.
- Daten Antwort-Token, das nach einer Schreiboperation gesendet wird.

Die Längen der wichtigsten Antwort-Token und die Bedeutung der Bits sind in den Tabellen C.2 und C.3 aufgelistet.

Daten-Token Am Anfang und am Ende jedes Datenblocks befindet sich ein Token. Das Start-Token ist 1 Byte lang und unterscheidet sich für verschiedenen Operationen⁹. Das End-Token enthält eine 16 Byte lange CRC32-Prüfsumme, die im SPI-Modus standardmäßig nicht überprüft wird. Wenn bei einem Lesebefehl ein Fehler auftritt, wird anstelle eines Daten-Tokens ein Fehler-Token gesendet, in dem der Fehler codiert ist.

Initialisierungssequenz Nach Anlegen der Betriebsspannung benötigt die Karte 74 Takte bei einer Taktrate von 400 kHz, um sich selbst zu initialisieren. Die SD-Karte wacht standardmäßig im SD-Modus auf. Für den Wechsel zum SPI-Protokoll wird das Kommando *CMD0* des SD-Modus mit aktivem¹⁰ CS-Signal ausgeführt. Die Karte wechselt dann in den SPI-Modus und schickt ein *R1*-Antwort-Token. Danach muss mit der Befehlskombination <CMD55>, <ACMD41> der Status der Karte abgefragt werden, bis das *Idle-Status-Bit* der *R1*-Antwort '0' ist. Anschließend kann die Taktfrequenz auf 25 MHz erhöht werden und die Karte ist betriebsbereit. Diese Sequenz kann bis zu einer Sekunde lang dauern.

⁹Single Block Read/Write: 1111110b

¹⁰Signalpegel „low“.

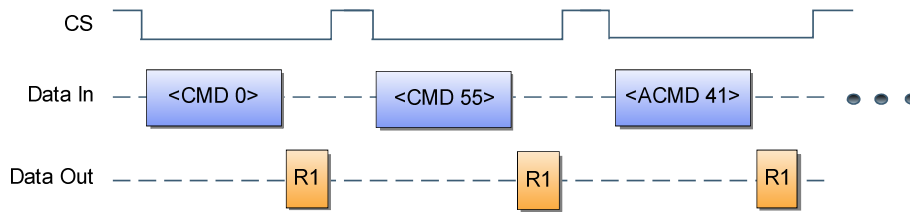


Abbildung 7.3: Initialisierungsbefehlsfolge für den SPI-Modus

Lese- und Schreibzyklen Es gibt zwei verschiedene Lesezugriffe: *Single Block Read* und *Multi Block Read*. Im Folgenden wird nur auf den *Single Block Read* eingegangen. Ein Lesezyklus wird durch den Befehl <CMD17> eingeleitet. Im Parameter wird die Leseadresse angegeben. Nach dem Antwort-Token folgt ein Datenblock-Token. Das Ablaufdiagramm ist in Abbildung 7.1.1 dargestellt. Der Zugriff auf die Statusregister ist in eigene Befehle gekapselt, folgt aber dem Schema eines Lesezyklus mit einer Blockgröße von 16 Byte.

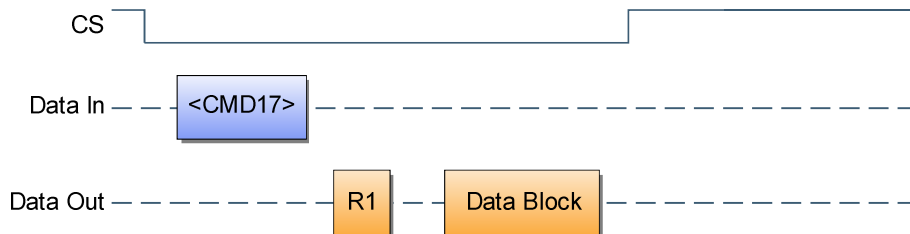


Abbildung 7.4: SPI-Lesezyklus: Host sendet <CMD17> und die Karte antwortet mit R1 und einem Datenblock.

Ein *Single Block Write* wird mit dem Befehl <CMD24> gestartet und läuft wie in Abbildung 7.1.1 dargestellt ab.

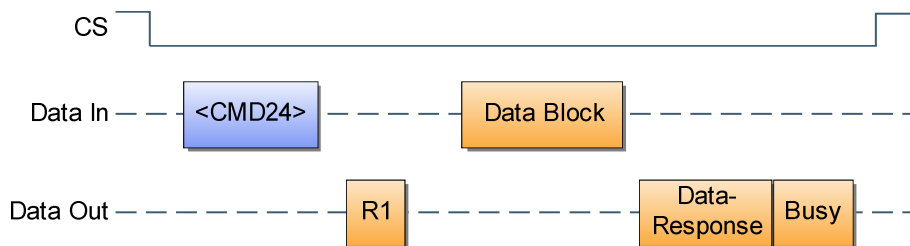


Abbildung 7.5: SPI-Schreibzyklus: Senden von <CMD24>, nach Empfang der Antwort wird der Datenblock gesendet. SD-Karte sendet ein *Busy*, bis die Daten verarbeitet sind.

Die Umsetzung des in diesem Abschnitt vorgestellten Protokolls wird in den nun folgenden Abschnitten dargestellt. Zunächst wird die Schnittstelle zum PowerPC und der Datenpfad vorgestellt und danach auf die Struktur der *Controller-Finite-State-Machine* ein-

gegangen.

7.2 Aufbau des Controllers

Zunächst ist die Frage zu klären, warum ein Controller in Hardware entwickelt und nicht einfach eine Kombination aus einem *IP-Core* für SPI und Controller in Software verwendet wird. Zum einen ist SPI nicht kompatibel mit dem SD-Modus, weshalb bei einer Umstellung des Protokoll ein komplett neuer Controller geschrieben werden müsste. Zum anderen muss der Controller neu geschrieben werden, wenn ein Betriebssystem in der PowerPC-Einheit verwendet wird (zum Beispiel Linux). Unter Linux besteht dann das Problem, dass das Einhalten des Timings für die Kommunikation durch den *Scheduler* nicht sicher gestellt werden kann. Bei der Verwendung eines Controllers in Hardware muss nur ein einfacher *Block-Device*-Treiber geschrieben werden, wenn die SD-Karte unter Linux verwendet wird. Darüber hinaus würde sich die Logik des *Bootloaders* erheblich ändern, da dieser den Controller implementieren müsste, um das Betriebssystem von der SD-Karte laden zu können.

OPB-Schnittstelle Die Schnittstelle zum OPB besteht aus sieben Registern und zwei FIFOs, auf die von der PowerPC-Einheit teils schreibend, teils lesend über den OPB zugegriffen werden kann. Die Aufteilung ist in Tabelle 7.4 dargestellt. Über diese Register und FIFOs wird auch das *Clock Domain Crossing* zwischen PowerPC-Einheit und SD-Karte durchgeführt. Dies ist nötig, da der OPC mit 100 MHz und die SD-Karte mit 25 MHz oder 400 kHz getaktet ist. Der komplette Controller wird mit demselben Taktsignal betrieben, mit dem auch die SD-Karte getaktet wird. Dadurch werden *Sampling*-Probleme vermieden, die entstehen könnten, wenn Karte und Controller in unterschiedlichen *Clock-Domains* liefen.

Adresse (hex)	Zugriff	Inhalt
0x0	R/W	Steuerparameter
0x4	R/W	Befehlsparameter
0x8	R	Controllerstatus
0xc	R	R1, R2, CRC Antwort
0x10	R	R3(OCR)
0x14	R	Fehler- und Zykluszähler
0x18	R/W	Schreibdaten
0x1c	R	Lese-FIFO
0x1c	W	Schreib-FIFO

Tabelle 7.4: Steuerregister des SD-Karten-Controllers. Die Adresse ist der Offset zur Basisadresse. Die genaue Belegung der Bits ist in Tabelle C.4 zu finden.

Datenpfad Die Lese- und Schreibdaten werden jeweils in einem FIFO-Speicher gepuffert, dessen Tiefe ausreicht¹¹, um einen kompletten Datenblock zu puffern. Das Lesen bzw. Schreiben auf die SD-Karte erfolgt so unabhängig von Zugriffen der PowerPC-Einheit auf den Controller. Der Füllstatus der beiden FIFO-Speicher ist über das Controllerstatus-Register abfragbar.

Für die Serialisierung kommen zwei *Parallel-to-Serial*-Konverter zum Einsatz, der eine für die Daten (32 Bit) und der andere für die Kommandos (48 Bit). Es hätte theoretisch auch ein Konverter genügt, der beides serialisiert. Mit dieser Implementierung wird berücksichtigt, dass in SD-Modus Kommando- und Datenleitungen getrennt sind und somit bei der Umstellung einfach der Multiplexer wegfällt, der jetzt die beiden Konverter in Abhängigkeit des Status des Befehlszykluses auf den Ausgang zur SD-Karte legt. In der Empfangseinheit läuft kontinuierlich ein *Serial-to-Parallel*-Konverter, der 32 Bit breit ist. Sein Ausgang ist mit dem Lese-FIFO und den Statusregistern verbunden. Die *Write-Enable-Signale* von diesen Einheiten werden von der Befehlszyklus-State-Machine passend gesetzt.

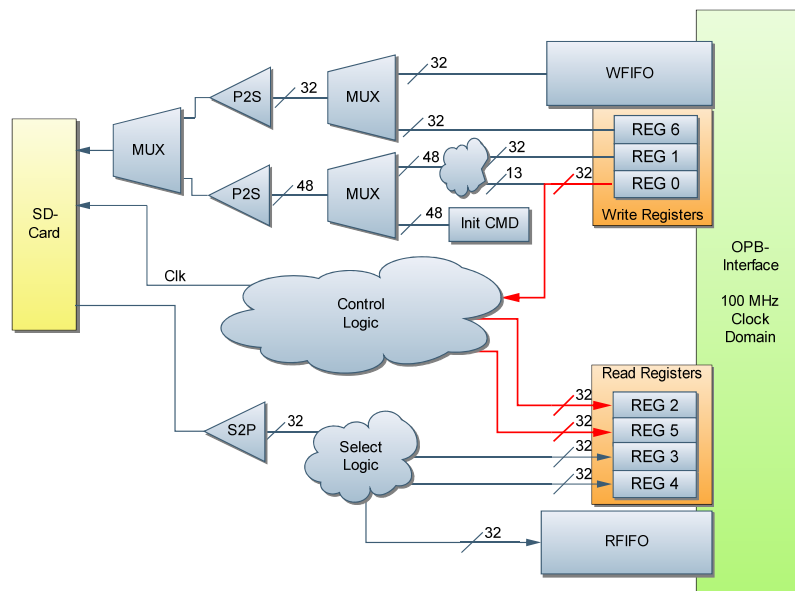


Abbildung 7.6: Aufbau der Datenpfade des *SD-Karten-Controllers*. Das *Clock-Domain-Crossing* wird über die FIFOs und mehrstufige Register durchgeführt. Zur Karte gesendete Daten werden zuvor serialisiert und empfangene Daten parallelisiert.

Timing Das Timing zwischen den einzelnen Blöcken in einem Befehlszyklus ist nicht fest und kann von Karte zu Karte variieren. Die Zugriffszeiten sind im CSD-Register der

¹¹Eine Minimaltiefe von 128 32-Bit-Worten (≈ 512 Byte) wird benötigt, weshalb ein FIFO mit der Tiefe von 256 32-Bit-Worten verwendet werden muss

Karte codiert. Der Controller arbeitet mit Standardwerten, die aber per Register verändert werden können. Für die genauere Beschreibung des Timing zwischen den einzelnen Blöcken sei auf [San04] verwiesen.

Die Hauptaufgabe des Controllers ist die Kommunikation mit der SD-Karte gemäß dem SPI-Modus. Hierfür werden zwei geschachtelte *Finit-State-Machines* verwendet, die eine, um einen Befehlszyklus abzuarbeiten, die andere für die Initialisierungssequenz.

Initialisierung Die *State Machine* zur Initialisierung der SD-Karte hat als Eingangssignale das Reset-Signal, ein Statusbit, das anzeigt ob eine Karte im Slot ist und das *Idle-State-Bit* der R1-Antwort. Die Ausgabesignale sind ihr Status und Steuersignale für die *Befehlszyklus-State-Machine*, *Eingangsmultiplexer* und *Takt-Controller*. Der Übergangsgraph zwischen den Zuständen ist in Abbildung 7.2 dargestellt.

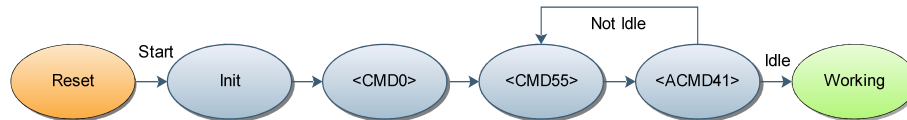


Abbildung 7.7: Flussdiagramm der *State Machine* zur Initialisierung. Nach einem Reset wird im Zustand „Init“ mit der Initialisierung begonnen. Hier sind aus Gründen der Übersichtlichkeit nur die wichtigsten Zustände und Übergänge dargestellt.

Befehlszyklus-State-Machine Diese modelliert einen Befehlszyklus, der aus Kommando, Antwort und einem optionalen Datenblock besteht. Die Eingangssignale umfassen Befehl, Befehlsparameter, Antworttyp und Statussignale der anderen Einheiten. Die Ausgabe besteht zum einen aus dem Status, der Antwort-Token und gegebenenfalls Lese-Daten, zum anderen aus Steuersignalen für alle anderen Komponenten. Diese *State Machine* hat, wie in Abbildung 7.2 dargestellt, viele Zustände, die größtenteils sequentiell durchlaufen werden. Der Übersicht halber wurden nur die wichtigsten Zustände dargestellt und die Übergänge von allen Zuständen in den Fehlerzustand weggelassen. Insgesamt hat die *State Machine* 18 Zustände.

7.2.1 Implementierung

Die Implementierung des Controllers erfolgt in VHDL und ist in mehrere Design-Blöcke (*Entities*) unterteilt, die in Tabelle C.6 im Anhang C aufgelistet sind.

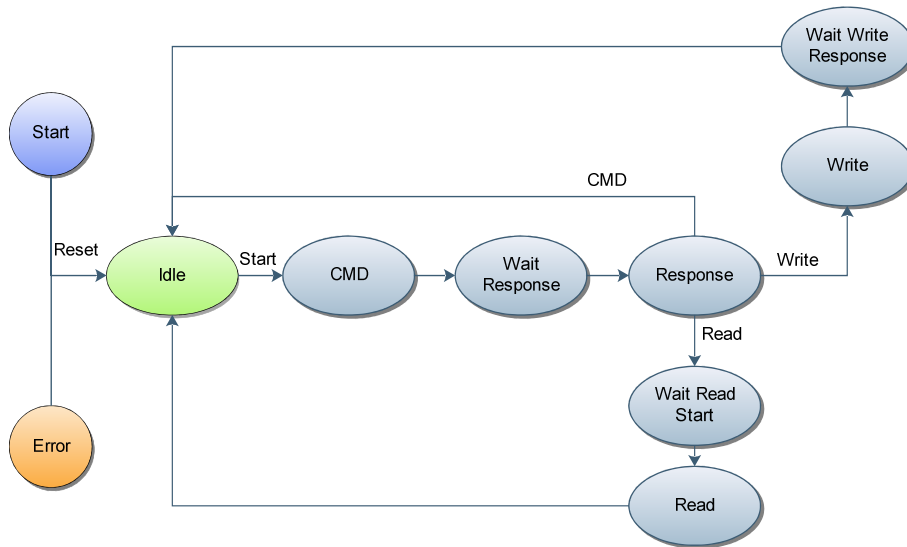


Abbildung 7.8: Flussdiagramm der *Befehlszyklus-State-Machine*. Ein Befehlszyklus startet und endet im „Idle“-Zustand. Nur die wichtigsten Zustände sind hier dargestellt.

Verifikation Um das Verhalten des Controllers zu überprüfen und zu testen, wird eine ModelSim-Testumgebung¹² verwendet, mit der sich die Initialisierungssequenz sowie Lese- und Schreibzyklen simulieren lassen. Ein vollständiges Simulationsmodell einer SD-Karte im SPI-Modus steht leider nicht zur Verfügung.

Es gibt zwei verschiedene Strukturierungen der Implementierung, deren Vor- und Nachteil im folgenden diskutiert werden.

IP-Core Die erste Implementierungsvariante ist ein IP-Core mit einer OBP-Schnittstelle, der vollständig lauffähig ist, ohne weitere externe Komponenten zu benötigen. Der Core kann in anderen Projekten über die XPS-GUI instantiiert werden. Dies birgt den Vorteil, dass man den Controller ohne Anpassung des Quellcodes verwenden kann. Der Nachteil besteht darin, dass der Controller in die PowerPC-Einheit gekapselt wird, was ein späteres *Constraining* der Logik-Zellen erschwert. Dies ist in diesem Projekt nötig, da für das *Clock*-Signal ein dediziertes *Clock*-Netz benötigt wird, von dem es nur eine begrenzte Anzahl im FPGA vorhanden ist.

Externer Controller In der zweiten Variante wird ein *Core* für die Schnittstelle zum OPB in der PowerPC-Einheit instantiiert. Dieser wickelt nur die Kommunikation mit dem OPB ab und routet alle Signale ins *FPGA-Fabric*, in dem der eigentliche Controller instantiiert wird. Bei dieser Variante ist das *Constraining* der Einheit deutlich einfacher, da sie

¹²ModelSim ist ein Simulationprogramm für Schaltungen, die zum Beispiel in VHDL beschrieben werden

nicht in der PowerPC-Einheit gekapselt ist und ihre Logik über das gesamte FPGA verteilt wird. Die SD-Karte kann hier auch ohne die PowerPC-Einheit benutzt werden, da die Anbindung an die PowerPC-Einheit mit einer Alternativlogik „verodern“ werden könnte. Hiermit ist zum Beispiel ein Bootloader in Hardware denkbar, der nach einem Reset, den DRAM mit Daten von der SD-Karte initialisiert.

Der Zugriff auf den Controller über die PowerPC-Einheit wird in dem nun folgenden Abschnitt beschrieben.

7.2.2 Treiber-Software für den PowerPC

Der Zugriff auf die SD-Karte erfolgt über die Register, die direkt in den Adressraum des PowerPCs abgebildet werden. Mit einer Reihe von C-Routinen werden die wichtigsten Funktionen wie Reset, Read, Write, Erase und Ausführen eines Kommandos implementiert. Sie sind in dem Kommando `sdcard` gekapselt. Die Routine schreibt Parameter in die Register und fragt dann das Statusregister ab, bis der Kommandozyklus beendet ist und gibt das Ergebnis aus. Über Eingabeparameter können Quelle bzw. Ziel der Daten gesetzt werden.

Für den Einsatz von Linux würden diese Routinen in einem Kernel-Modul implementiert werden und den Treiber für die SD-Karte darstellen.

7.2.3 Performance und Erweiterungsmöglichkeiten

Performance

Da der DRAM noch nicht an die PowerPC-Einheit angebunden ist, können bisher keine aussagekräftigen Tests bezüglich der I/O-Geschwindigkeit beim Transfer von Daten zwischen Speicher und Karte durchgeführt werden. Ein weiteres Problem stellt die Zeitmessung im PowerPC dar, die bisher noch nicht implementiert ist. Die durchgeführten Messungen haben daher immer noch den UART-Overhead, da die Zeit für ein Befehlszyklus über die Ausführungszeit des `gtucom`-Kommandos gemessen wird. Hier hat sich gezeigt, dass die Ausführungszeit eines Kommandos kleiner ist als die Minimalzeit eines `gtucom`-Kommandos. Einen sichtbaren Effekt hatte nur das Ausführen der Testroutine, die `n` Datenblöcke löscht, schreibt und liest. Die hier gemessenen Werte streuen sehr, was auf unterschiedliche Antwortzeiten der SD-Karte schließen lässt. Dies ist aber angesichts der systematischen Fehlern bei der Zeitmessung sehr spekulativ. Die momentan implementierte Software ist nicht auf Schnelligkeit, sondern auf Stabilität, Ausfallsicherheit und möglichst wenig Fehlerquellen durch *Race-Conditions*¹³ optimiert. So könnte bei einem Schreibzyklus zum Beispiel das FIFO während des Schreibvorgangs befüllt werden

¹³Als *Race-Conditions* werden Konstellationen bezeichnet, in denen das Ergebnis einer Operation vom zeitlichen Verhalten bestimmter Einzeloperationen abhängt

und nicht nur vor dem Schreibzyklus, wie es momentan der Fall ist. Dies sollte ohne Probleme möglich sein, da $32 \cdot 4 = 128$ OPB-Taktzyklen Zeit ist, um ein 32-Bit-Wort zu schreiben, was ein Schreibzugriff ist. Die mit ChipScope¹⁴ gemessene Zeit zwischen zwei Schreibzyklen auf den OPB liegt bei 30 Takten.

Erweiterungsmöglichkeiten

Der hier vorgestellte SD-Karten-Controller ist eine erste Implementierung, die an vielen Stellen noch erweitert und optimiert werden kann. Die wichtigsten Erweiterungen, die noch implementiert werden sollten, sind die Unterstützung von Multi-Block-Read- und Write-Operationen und eines CRC-Prüfers für die Daten.

Eine weitere Verbesserung ist ein interruptgesteuerter Zugriff auf den Controller von PowerPC aus. Als Interruptsignal kann das *Cycle-Busy*-Signal verwendet werden, das eine Flanke am Ende des Befehlszykluses erzeugt. Dies ist vor allem im Hinblick auf eine Portierung von Linux auf die GTU-Boards interessant, da dann der Treiber die I/O mit Interrupts abwickeln kann und nicht das Statusregister abfragen muss. Die Implementierung des SD-Modus ermöglicht den Zugriff auf die SD-Karte zu beschleunigen, da hier vier Daten- und eine Steuerleitung bidirektional genutzt werden, was mindestens eine Beschleunigung um den Faktor vier bedeuten würde. Wenn darüber hinaus zusätzlich der *Fast-Modul* (50 MHz) genutzt würde, der nur im SD-Modus zur Verfügung steht, könnte rein rechnerisch die I/O-Geschwindigkeit um den Faktor acht erhöht werden.

¹⁴Programm, mit dem der zeitliche Verlauf von Signalen im FPGA über JTAG ausgewertet werden kann

8 Erste Tests am CERN

Seit Ende Juli sind acht von neun Crates am CERN installiert und im Testbetrieb. Ein zweiwöchiger Dauertest des Überwachungssystems verläuft ohne Systemabsturz. Bei diesem Test waren allerdings noch nicht alle Komponenten des DIM-Servers auf dem DCS-Board aktiviert. In den nun folgenden Abschnitten wird auf erste Tests und Messergebnisse der einzelnen Komponenten und Schichten des Gesamtsystems eingegangen. Diese Messergebnisse werden genutzt, um die Schwellwerte für das Alarmmanagement zu setzen. Anschließend wird der aktuelle Status des Projekts dargestellt und ein Ausblick auf künftige Aufgaben gegeben.

8.1 Messwerte und Kalibrierung

Seit der Installation am CERN ist es möglich, Aussagen über die eingesetzten Sensoren zu machen. Es zeigt sich, dass die Streuung der Messwerte bei den Spannungs- und Temperatursensoren gering ist.

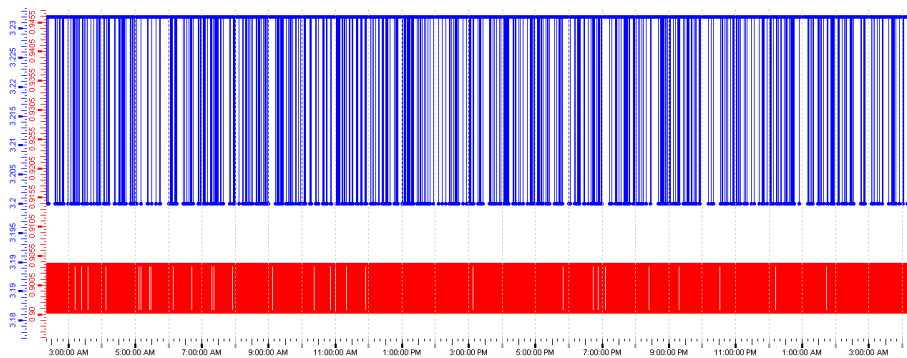


Abbildung 8.1: Screenshot: Spannungsverlauf der 3,3 V (blau) und 0,9 V (rot) Spannung auf einem Board über 24 h. Im Plot ist zu sehen, dass die Spannung zwischen zwei Werten hin und her springt, die der Genauigkeit des Sensors entspricht. Die 3,3 V schwankt um 0,32 V zwischen 3,20 V und 3,323 V. Die 0,9 V schwankt um 0,009 V zwischen 0,895 V und 0,904 V

Spannungen Die Eingangsspannungen (5,0 V und 3,3 V) auf den einzelnen Boards variieren erheblich, was auf den unterschiedlichen Spannungsabfall auf der CompactPCI-

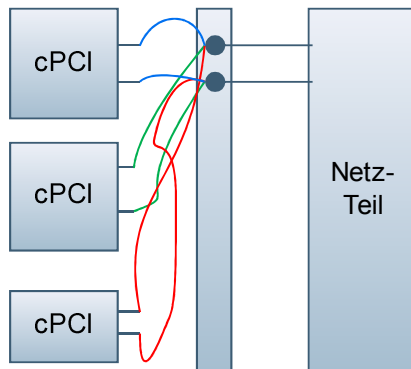


Abbildung 8.2: Verkabelung eines Crates aus der Vogelperspektive. Die Stromkabel vom Netzteil werden an der Crate-Rückseite mit den Kabeln der CompactPCI-Backplane verbunden. An dieser Stelle sind auch die Sensleitungen des Netzteils angebracht.

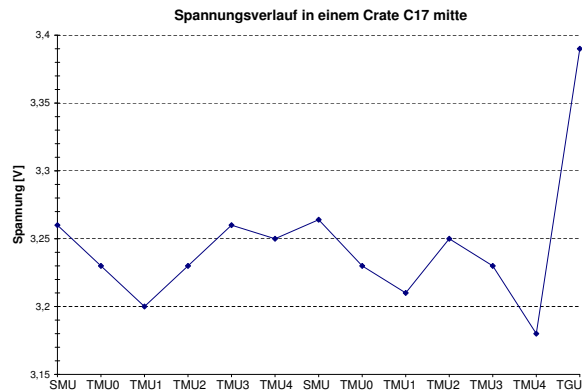


Abbildung 8.3: Verlauf der Eingangsspannung an den Boards im mittleren Crate des Racks C17. Die Spannung schwankt deutlich über die Boards, ist aber zeitlich stabil.

Backplane in den Zuleitungen für die Boards zurückzuführen ist. Der zeitliche Verlauf der Spannung auf den einzelnen Boards ist stabil und schwankt nur um den Messfehler des Sensors. In Abbildung 8.1 ist die Schwankung der 3,3-V-Eingangsspannung auf den einzelnen Boards für das Crate mit der TGU (Rack C17 Mitte) dargestellt. Die gemessenen Eingangsspannungen in der TGU sind deutlich höher als bei den anderen Boards. Dies kann dadurch erklärt werden, dass die TGU eine eigene CompactPCI-Backplane für die Spannungsversorgung hat. Die Spannung für dieses Crate kann nicht weiter erhöht werden, da die TGU sonst Schaden nehmen würde, wenn die Spannung über 3,4 V steigt, da mehrere Komponenten direkt an die 3,3-V-Eingangsspannung ohne Spannungswandler angeschlossen sind und nicht mit höheren Spannungen betrieben werden sollten. Die durch die Spannungswandler erzeugten Sekundärspannungen sind dagegen genau und zeitlich stabil auf allen Boards.

Temperaturen Die Temperaturverteilung innerhalb eines Racks ist in Abbildung 8.4 dargestellt. Sie steigt von unten nach oben, wie auch der Luftstrom der Kühlung. Die TMUs sind deutlich wärmer als die SMUs, was auf die MGTs zurückzuführen ist, die mit einer Frequenz von 5 GHz laufen und dabei sehr viel Wärme freisetzen. In der TMU kommen zwölf, in der SMU nur vier zum Einsatz. Die allgemeine Temperaturverteilung aller Boards ist in Histogramm 8.1 zu sehen. Es sind drei Häufungspunkte zu erkennen, die auf die drei Einbauhöhen der Crates, in denen die FPGAs jeweils in einer Ebene liegen, zurückzuführen sind. Da die SMUs kühler als die TMUs ist, sind die Häufungs-

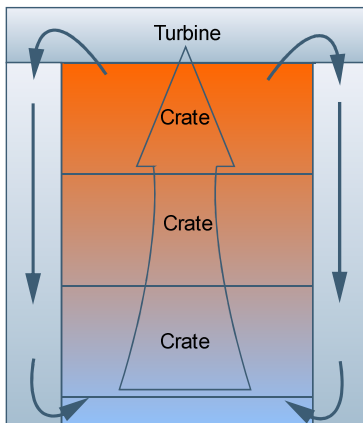


Abbildung 8.4: Der Luftstrom in einem Rack. Die Luft wird von der Turbine eingesaugt, gekühlt über die Seitenwände nach unten und wieder zurück ins Crate geblasen.

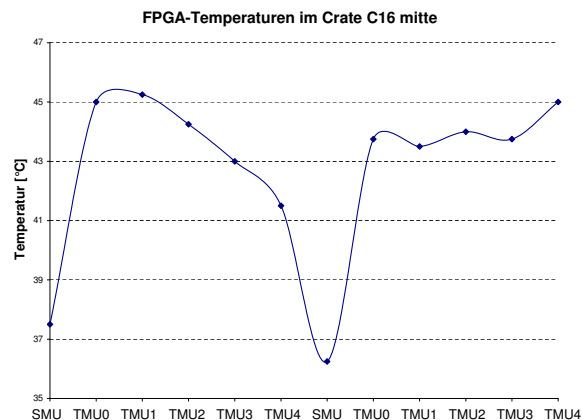


Abbildung 8.5: Temperaturverteilung im Crate C16 Mitte. Die FGAs in der Mitte werden besser gekühlt als außen. Die SMUs sind deutlich kühler als die TMUs.

punkte zu kleineren Temperaturen hin ausgeschmiedet. Des Weiteren kommt hinzu, dass im mittleren Rack zur Zeit nur zwei Crates installiert sind. Hier gibt es daher nur zwei Temperaturmesszonen.

Einstellen der Alarmschwellwerte in PVSS Nachdem durch die ersten Tests ausreichend viele Daten zur Verfügung stehen, um eine Statistik zu erstellen, kann nun begonnen werden, die Alarmschwellwerte sinnvoll zu setzen. Die gesetzten Schwellwerte von den wichtigsten Werten sind in Tabelle 8.1 zusammengefasst. Sie werden etwas niedriger als der niedrigste gemessene Wert und etwas höher als der höchst gemessene Wert festgelegt. Darüber hinaus muss natürlich beachtet werden, dass sich die Grenzwerte noch innerhalb der für die Bauteile vorgegebene Spezifikation befinden. Eine untere Grenze für die FPGA-Temperatur ist sinnvoll, da nicht programmierte FGAs deutlich kälter sind und somit einen Alarm auslösen. Dies ist wichtig, da in einem über längere Zeit unprogrammiertes FPGA die MGTs zerstört werden.

Messgröße	Alarm Min	Min	Avg	Max	Alarm Max
FPGA Temperatur [°C]	25,0	31,5	43,8	57,25	65,0
PCB Temperatur [°C]	5,0	20	26,5	35	40,0
VCC 5,0 [V]	4,8	4,87	4,94	5,04	5,1
VCC 3,3 [V]	3,16	3,17	3,23	3,39	3,4

Tabelle 8.1: Gemessene Temperatur- und Spannungsbereiche und die daraus abgeleitete Alarmschwellwerte.

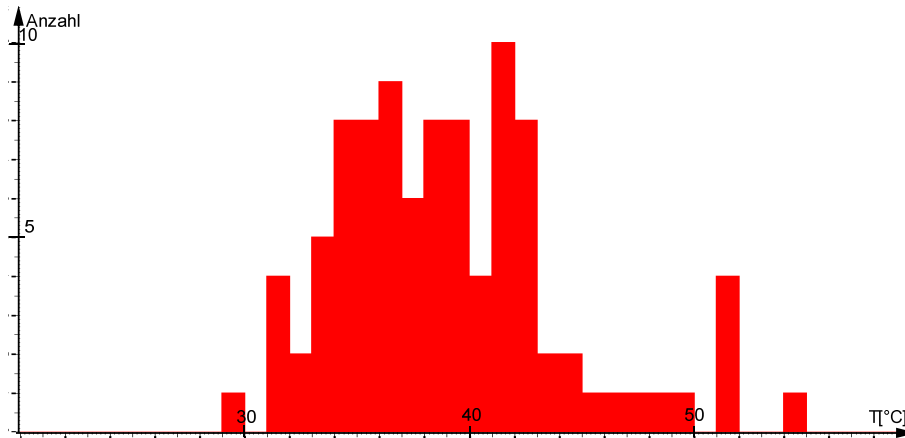


Abbildung 8.6: Screenshot: Temperaturverteilung von 95 FPGAs. Durch die unterschiedlichen Temperaturregionen und Unterschiede zwischen TMU und SMU ist die Verteilung sehr breit. Die drei Häufungspunkte können den Einbauebenen der FPGAs in den Racks zugeordnet werden.

8.2 Reaktionszeiten des Systems

Das Überwachungssystem registriert zeitnah Veränderungen, wobei hier die Auslesezyklen der Messwerte berücksichtigt werden müssen. Die Temperatur- und Spannungswerte werden etwa alle 10 s aktualisiert. In Abbildung 8.2 ist der Temperaturverlauf von zwei FPGAs dargestellt.

Antwortzeiten auf Kommandos Durch jede weitere Schicht erhöht sich die Antwortzeit auf ein Kommando. Eine Übersicht hierüber ist in Tabelle 8.2 zu finden. Die Antwortzeiten erhöhen sich über das Netzwerk um etwa 0,66 s.

Host	Programm	Parameter	Zeit \bar{t} [s]	σ [s]
alidcsdcb0611	gtucom	"2 dimping -d 1"	0,08	0,01
alidcsdcb0611	dimgtucomrpc	11 "2 dimping -d 1"	0,43	0,04
alitrdown005	dimgtucom	11 "2 dimping -d 1"	0,74	0,03
alitrdown005	dimgtucom	11 -b"2 1 75*0"	0,46	0,03

Tabelle 8.2: Messung der Antwortzeiten im DCS-Netz am CERN bei Ausführung desselben PPC-Kommandos in verschiedenen Schichten. alidcsdcb0611 ist das DCS-Board und alitrdown005 ein Rechner im DCS-Netzwerk.

Überhitzungstest Die Turbinen werden zur Zeit noch nicht überwacht. Dies ist eine große Gefahr für die GTU, da die FPGAs zerstört werden würden, wenn sie ohne Küh-

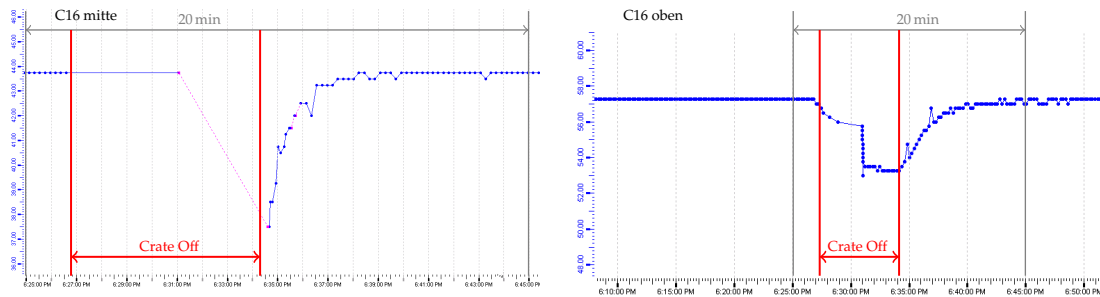


Abbildung 8.7: Die beiden Graphen zeigen den zeitlichen Temperaturverlauf von zwei FPGAs aus übereinander eingebauten Crates, von denen das untere (hier links abgebildet) aus- und wieder angeschaltet wird. Die Skalierung der Graphen ist unterschiedlich. Es ist deutlich zu erkennen, dass die Temperatur im oberen FPGA (hier rechts abgebildet) sinkt um insgesamt 4 K, sobald das untere Crate ausgeschaltet wird, was an der Nichtaktualisierung des Messwertes (Linie ohne Punkte) oder seiner Ungültigkeit (Pink) erkannt werden kann. Die Temperaturen stabilisieren sich auf dem Ausgangsniveau, nachdem das Crate wieder angeschaltet ist.

lung betrieben werden. Das muss das Überwachungssystem erkennen und die Crates abschalten. Für einen Überhitzungstest wird die Turbine eines Racks deaktiviert. Die Segmente werden durch einen *Softwareinterlock* nach Überschreiten einer FPGA-Temperatur von 70 °C abgeschaltet. Die Abbildung 8.2 zeigt den Temperaturverlauf von jeweils einem FPGA pro Segment, während des Tests. Deutlich zu sehen ist, dass die Temperatur im Rack nach oben hin zunimmt und die Temperatur eines FPGAs jeweils stabil ist, bis die Turbine deaktiviert wird. Daraufhin beginnt die Temperatur der FPGAs zu steigen und beim Überschreiten des Schwellwerts von 70 °C wird das Crate abgeschaltet. Das erste Crate wird nach 60 s, das letzte nach 13 s abgeschaltet. Danach sind keine weiteren Temperaturmesspunkte verfügbar, da das DCS-Board, das die Temperaturen ausliest, mit abgeschaltet wird. Wenn die Crates wieder eingeschaltet werden, steigen die Temperaturen wieder an. Alle Segmente haben sich bei Erreichen der Grenztemperatur abgeschaltet. Damit wird gezeigt, dass das System bei einem Komplettausfall der Kühlung schnell reagiert und keine Überhitzung der FPGAs droht. Das einzige Problem hierbei ist, dass noch gültige Messwerte angezeigt werden, obwohl die Boards schon abgeschaltet sind. Dies ist ein allgemeines Problem, da das System auf Veränderung basiert und die Werte nicht regelmäßig über DIM abfragt. Wenn die DIM-Server nicht normal beendet werden, sondern das DCS-Board einfach ausgeschaltet wird, kann dies von PVSS nicht sofort erkannt werden und der letzte Wert behält seine Gültigkeit. Abhilfe für das Problem wird dadurch erreicht, dass ein Datenpunkt periodisch abgefragt wird und somit sehr schnell erkannt werden kann, dass ein DIM-Server nicht mehr verfügbar ist. Dieses Problem tritt nicht auf, wenn der DIM-Server normal beendet wird, da er dies dem DNS-Server mitteilt. Dieser gibt die Information an alle Clients und damit auch an PVSS weiter.

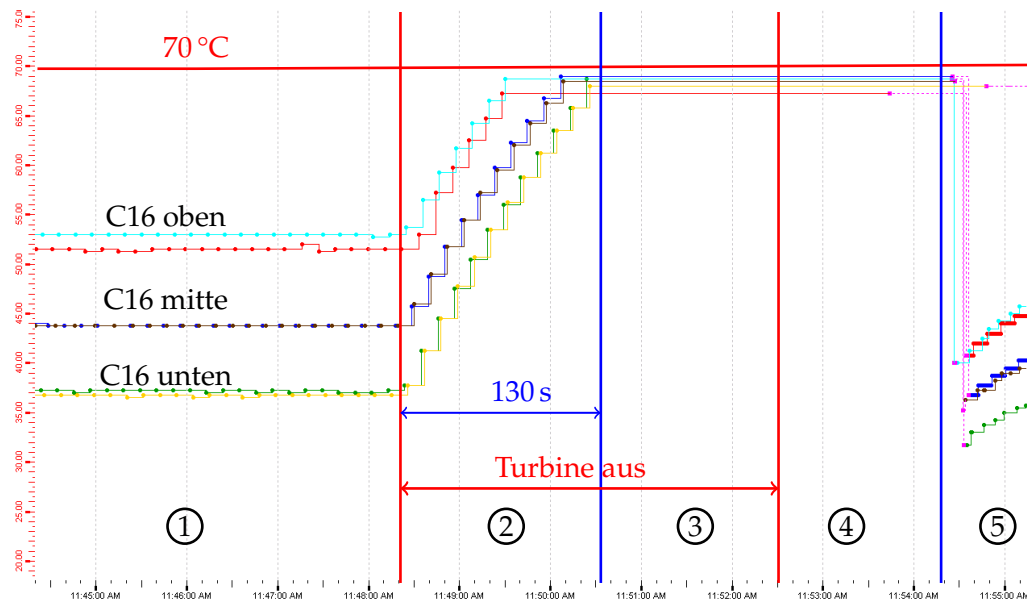


Abbildung 8.8: Der Graph zeigt den Temperaturverlauf von 6 FPGAs (jeweils TMU3) in einem Rack bei einem Turbinenausfall. Die Messwerte werden durch Punkte dargestellt. Pink bedeutet, dass der Messwert ungültig ist. Der Graph kann in folgende Abschnitte unterteilt werden: Normalbetrieb (1), Turbine fällt aus und Temperatur steigt (2), alle Crates sind abgeschaltet - keine neue Messwerte (3), Turbine ist wieder aktiv (4), Crates wieder an, neue Messwerte - Temperatur steigt wieder auf Betriebsniveau (5).

8.3 Ausblick

Das System hat in den ersten Tests gezeigt, dass die Daten ohne Fehler über die gesamte Ausleseketten transportiert und korrekt in PVSS dargestellt werden. Board die eine Temperatur oder Spannung außerhalb des definierten Wertebereichs aufweisen, werden sofort durch das Alarmmanagement erfasst. Die Stabilität des Systems muss noch verbessert werden. Während der Testphase sind Probleme mit dem DIM-Server aufgetreten, der sich bei bestimmten Bedingungen nicht mehr reagiert. Die Ursache hierfür ist der Zugriff auf den UART-Treiber. Die *Read*-Funktion wird bei einem Timeout im Treiber nicht beendet. Dieses Problem muss schnellstmöglich behoben werden, da hierdurch die Stabilität des Systems gefährdet ist.

Es stellt sich auch heraus, dass der Funktionsumfang noch erweitert werden muss, um eine effizientere Arbeit während der noch laufenden Entwicklungsphase und dem späteren Experimentalbetrieb zu ermöglichen. Insbesondere sind weitere Funktionen und Panels in PVSS wünschenswert. Als nächstes sollte eine Suchfunktion implementiert werden, die Boards mit bestimmten Messwerten ausgibt, da es momentan schwierig ist, Boards zu finden, die bestimmte Eigenschaften haben. Die komplette Integration in die FSM-Hierarchie des Detektors muss ebenfalls noch durchgeführt werden.

9 Zusammenfassung

Eine Herausforderung bei Kollisionsexperimenten mit Schwerionen ist der Umgang mit sehr großen Datenmengen, die von den Detektoren erzeugt werden. Ein mehrstufiges Triggersystem selektiert daher zur Experimentlaufzeit die physikalisch interessanten Daten. Die Entscheidungsbasis hierfür sind die Ergebnisse der Triggerdetektoren, in denen die Rohdaten sehr schnell verfügbar sind und online analysiert werden. Der Übergangsstrahlendetektor in ALICE ist ein solcher Triggerdetektor, der für die Impulsbestimmung und Teilchenidentifikation zuständig ist. Das Auslesen des Detektors und die Onlineanalyse der Daten findet in der *Global Tracking Unit* (GTU) in weniger als $2\ \mu\text{s}$ statt. Die Ergebnisse fließen in die *L1-Trigger*-Entscheidung ein. Die GTU ist ein komplexes System, bestehend aus 109 Untereinheiten. In jeder dieser Einheiten arbeitet ein FPGA mit zwei PowerPC-Prozessoren.

Die vorliegende Arbeit beschäftigt sich mit der Überwachung und Steuerung der *Global Tracking Unit*. Das hierfür entwickelte Überwachungs- und Fernwartungssystem muss hochverfügbar sein, da der Ausfall der GTU, den Ausfall des gesamten Detektors zur Folge hätte. Das System ist in drei Schichten untergliedert. Die in der jeweiligen Schicht enthaltenen Subsysteme sind unabhängig voneinander. Erst in der höchsten Schicht laufen die Daten zentral zusammen. Daraus erschließt sich ein breites Aufgabenspektrum, das sich vom FPGA-Design über Linuxtreiberprogrammierung bis hin zur Entwicklung graphischer Oberflächen erstreckt.

Die unterste Schicht bilden die FPGAs. In ihnen ist ein „*System on a Chip*“ implementiert, das einen der beiden PowerPC-Kerne als Prozessor verwendet. Mit dem speziell für dieses Embedded System entwickelten Soft- und Hardwaresystem werden die anderen Design-Einheiten im FPGA und die Peripheriekomponenten überwacht und gesteuert. Das System muss hochverfügbar sein, da ein Neustart im Fehlerfall während der Experimentlaufzeit mit unwiederbringlichen Datenverlusten einhergehen würde. Die Kommunikation mit der nächst höheren Schicht erfolgt über eine spezielle serielle Schnittstelle, über die mehrere Einheiten verbunden sind.

In der nächst höheren Schicht, dem DCS-Board, laufen die Informationen von einem Segment, das aus sechs Einheiten besteht, über verschiedene Auslesekanäle zusammen. Steuerbefehle von höheren Schichten werden an die Einheiten weitergegeben. Das DCS-Board besitzt ein Embedded Linux System. Auf jedem der insgesamt 19 DCS-Boards der GTU läuft unabhängig ein Server, der die Schnittstelle zur nächst höheren Schicht darstellt. Er veröffentlicht über ein spezielles Protokoll (DIM) die Daten des GTU-Segments

im Netzwerk und empfängt Kommandos von höheren Schichten. Die Herausforderung hierbei liegt in der Implementierung des Servers, die stabil und Ressourcen sparend sein muss.

Die oberste Schicht stellt PVSS dar. In ihr laufen die Daten von der gesamten GTU zusammen, die sich aus über 9.000 Einzelmesswerten zusammensetzen. Diese werden in PVSS verarbeitet, überwacht und visualisiert. Über verschiedene Eingabefenster kann die komplette GTU ferngesteuert werden.

Die ersten Tests, die Mitte August am CERN stattgefunden haben, sind positiv verlaufen. In einem Dauertest ist das System über eine Zeitspanne von zwei Wochen gelaufen und die Daten werden korrekt durch alle Schichten propagiert. Bei den Tests sind zum ersten Mal Probleme mit dem Treiber für die serielle Schnittstelle aufgetreten, welche die Stabilität des Servers im DCS-Board sehr beeinträchtigen. Die Ursache hierfür ist bereits gefunden, kann jedoch innerhalb der für diese Arbeit noch zur Verfügung stehenden Zeit nicht mehr behoben werden.

Darüber hinaus wird in dieser Arbeit ein SD-Karten-Controller in VHDL implementiert, der eine wesentliche Voraussetzung für eine Portierung von Linux auf die PowerPCs der FPGAs ist. Dies ist eine wichtige Vorarbeit für die Weiterentwicklung des Gesamtsystems GTU. Die SD-Karte kann als Speicher nicht nur für ein Linux, sondern auch für die verschiedenen Betriebskonfigurationen der GTU verwendet werden.

Danksagung Ich danke besonders herzlich meinem betreuenden Professor, Herrn Prof. Dr. Volker Lindenstruth, für die interessante Aufgabenstellung und die Möglichkeit der Aufenthalte am CERN. Bei der Unterstützung durch Jan de Cuveland, dem „Vater der GTU“, der mir immer mit Rat und Tat zur Seite stand und von dem ich viel gelernt habe. Herrn Dr. Venelin Angelov möchte ich mich für seine Hilfsbereitschaft danken. Mein Dank gilt auch den anderen Mitgliedern der Arbeitsgruppe, besonders Felix Rettig und Stefan Kirsch für die gute Zusammenarbeit. Es sei allen gedankt, die mich im Laufe der Arbeit durch Tipps und Anregungen unterstützt haben.

Mein herzlichster Dank gilt Daniela Ultsch, die mich immer motiviert und wo sie nur konnte unterstützt hat.

Anhang A

DIM-PVSS Datenzuordnung

Tabelle A.1: Übersicht der DIM-Datenpunkte eines GTU-Segments. Das Präfix für alle Datenpunkte ist "trd_fero_gtusm_". Danach folgt zweistellig das Supermodulesegment (00-18). Bei "_sfp_omon_" folgt zweistellig die SFP-Nummer. Die TGU wird als Supermodul 18 geführt. Dort sind die SFP-bezogenen Datenpunkte nicht vorhanden. Die Typen sind wie folgt: Data Service (S), Commando Service (C), Remote Procedure Call (R). Im Format steht zuerst der Datentyp (Float (F), Char (C), Long (L)) und danach die Anzahl seiner Elemente.

Name	Quelle	Anzahl	Typ	Format
_vmon	Backplane	6	S	F:10
_tmon	Backplane	6	S	F:2
_id	PowerPC	6	S	S
_svn	PowerPC	6	S	C
_tmumask	PowerPC	1	S	L
_cmd	PowerPC	1	R	C,C
_cmd_anim	PowerPC	1	C	C
_sfp_tmon	PowerPC	5	S	F:12
_sfp_vmon	PowerPC	5	S	F:12
_sfp_omon_*	PowerPC	60	S	F:1,L:4

Tabelle A.2: Aufbau der komplexen DIM-Datenpunkte

Typ	Index	Zuordnung
vmon	F[0]	VCC_25
	F[1]	VCC_18
	F[2]	VCC_12
	F[3]	VTT_09
	F[4]	AVCC_25
	F[5]	AVCC_15
	F[6]	AVCCA_12
	F[7]	AVCCB_12
	F[8]	VCC_50
	F[9]	VCC_33
tmon	F[0]	PCB
	F[1]	FPGA
sfp_tmon	F[0]-F[11]	Temperatur SFP 0-11
sfp_vmon	F[0]-F[11]	Vcc SFP 0-11
sfp_omon*	F	RX-Power
	L[0]	MGT-Status
	L[1]	MGT-Status
	L[2]	MGT-Status
	L[3]	MGT-Status

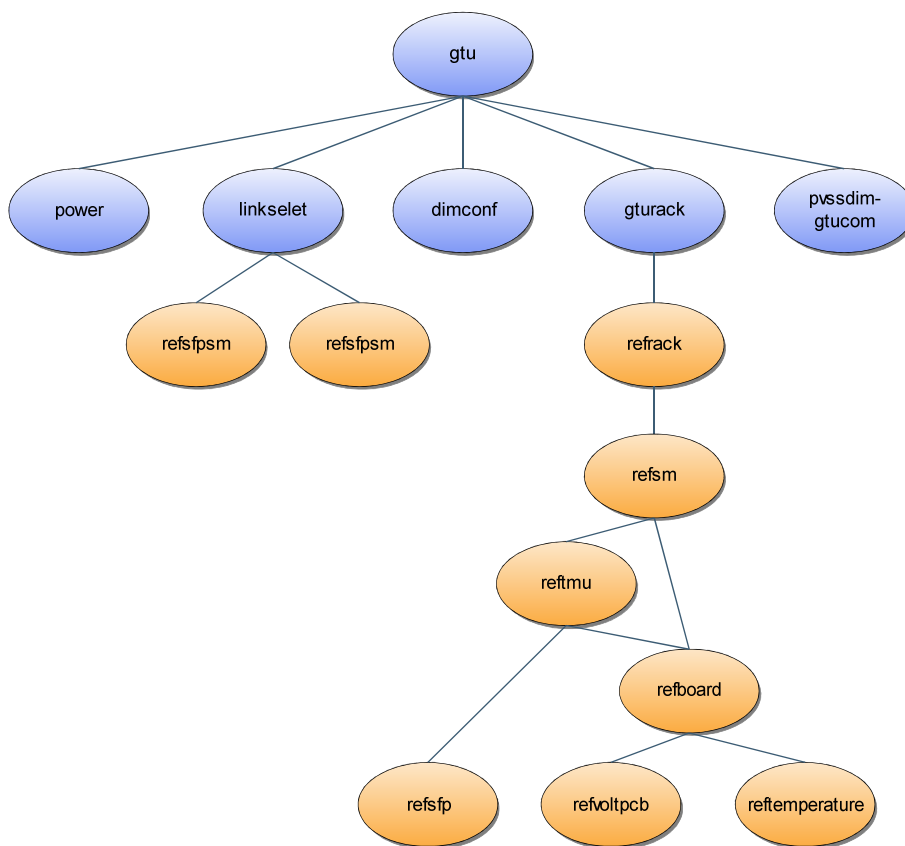
Anhang B

PVSS Panels

Tabelle B.1: Erstellte Panels und ihre Funktion

Name	Funktion
gtu	Übersicht, die in das Framework eingebunden wird
gturack	Überichtspanel über den physikalischen Status aller Borads
linkselekt	Auswahlpanel, um die Links eines Segments zu konfigurieren
power	Ein- und Ausschalten von GTU-Segmenten
dimconf	Konfigurationspanel für DIM-Manager
pvssdimgtucom	Panel, um mit einem Segment zu kommunizieren
reftmu	Anzeige von refboard und refsfp
rack	Anzeige des Status eines Racks
refsm	Anzeige von Status eines Segmentes
refboard	Anzeige von Temperatur, Spannung, SVN und ID eines Boards
reflink	Konfigurationspanel für die Linkmaske eines Segements
refsfp	Anzeige des SFP- und MGT-Status eines Stacks
refspsm	Anzeige des SFP- und MGT-Status eines Segments
reftemperature	Anzeige der Temperaturen eines Boards
refvoltpcb	Anzeige der Spannungen eines Boards

Abbildung B.1: PVSS Panel Hierarchie



Anhang C

SD-Karte

Tabelle C.1: Wichtige Befehle für die Kommunikation im SPI-Modus. Ein Kommando ist die sechsstellige binäre Codierung des Index (z.B. CMD17 wird mit „010001b“ codiert).

Index	Paramter	Antwort	Beschreibung
CMD0	keine	R1	Reset Karte
CMD9	keine	R1	Karte sendet CSD-Register
CMD10	keine	R1	Karte sendet CID-Register
CMD13	keine	R2	Karte sendet Status
CMD17	Adresse	R1	Liest einzelnen Datenblock
CMD24	Adresse	R1	Schreibt einzelnen Datenblock
CMD32	Adresse	R1	Setzt Startadresse zum Löschen
CMD33	Adresse	R1	Setzt Endadresse zum Löschen
CMD38	keine	R1b	Löscht markierten Bereich
CMD55	keine	R1	Nächstes Kommando ist ein ACMD
CMD58	keine	R3	Karte sendet OCR-Register
ACMD41	keine	R1	Startsynchronisierung

Tabelle C.2: Längen der wichtigsten Antwort-Token und ihre Substruktur

Antwortformat	Länge [Byte]	Aufteilung
R1	1	-
R1b	1	R1 + busy
R2	2	R1 + Byte 2
R3	6	R1 + OCR-Register
Data	1	-

Tabelle C.3: Bedeutung der Bits der wichtigsten Antwort-Token

Antwort	Bit	Bedeutung
R1	7	0
	6	Parameterfehler
	5	Adressfehler
	4	Löschsequenzfehler
	3	CRC Fehler
	2	Illegales Kommando
	1	Löschreset
	0	Idle Status
R2 (Byte 2)	7	Außerhalb des Adressraumes
	6	Löschparameterfehler
	5	Schreibschutzfehler
	4	ECC-Fehler
	3	Interner Fehler
	2	Fehler
	1	Fehler bei Lock/Unlock , Schreibschutz
	0	Karte gelockt
Data	7-5	x
	4	0
	3-1	Status: „010“: Ok; „110“ / „101“: Fehler
	0	1

Tabelle C.4: Belegung der Schnittstellenregister des SD-Karten-Controllers

Register	Bits	Beschreibung
0	25:31	CRC
	24	–
	18:23	CMD
	16:17	–
	15	Reset (aktiv low)
	14	Trigger (Debug)
	13	Reset FIFO (aktiv high)
	12	–
	8:11	Response Selekt (Tab. C.5)
	0:7	User Block Length
1	0:31	Kommandoparameter
2	31	Fehler
	30	Lesefehler
	29	Schreibfehler
	28	Idle
	26	Karte Schreibgeschützt
	25	Karte in Schacht
	24-25	Status Initialisierung: „00“: Reset, „11“: Ready
	23	Write-FIFO leer
	22	Read-FIFO voll
	21	Write-FIFO voll
	20	Read-FIFO leer
	8:19	Fehlerzähler
	0:7	Versionsnummer
3	24:31	R1
	16:23	R2 (Byte 2)/ Lesefehler
	0:15	CRC
4	0:31	R3
5	16:31	Befehlszähler
	8:15	Lesefehler
	0:8	Schreibfehler
6	0:31	Schreibdaten, statt FIFO

Tabelle C.5: Codierung der Response Selekt Signale. Mit diesem Signalen wird der Befehlzyklus-State-Machine mitgeteilt, welche Antwort zu erwarten ist. Bei *Write* wird nach Empfang der R1-Antwort begonnen, Daten zu senden.

Bitbelegung	Antwort	Beschreibung
0000	R1b	R1 Antwort folgendem Busy
0001	R1	R1 Antwort
0010	R2	R2 Antwort
0011	R3	R2 Antwort
0100	Read	R1 + Daten Block (Länge: 512 Byte)
0101	Read	R1 + Daten Block (Länge: 12 Byte)
0110	Read	R1 + Daten Block (Länge: User Block Length)
1000	Write	R1 + Daten Block (Länge: 512 Byte)
1001	Write	R1 + Daten Block (Länge: 12 Byte)
1010	Write	R1 + Daten Block (Länge: User Block Length)

Tabelle C.6: Liste der Entities des SD-Karten-Controllers

Name	Beschreibung
sdman_spi	Top Entity
control_spi	Controller
lock_control	Clockmanagement
sd_fifo	Sende und Empfangsfifo
gen_ser2par	Serial zu Parallel Wandler
gen_par2ser	Parallel zu Serial Wandler

Literaturverzeichnis

- [A L] A LARGE ION COLLIDER EXPERIMENT (ALICE) AT CERN LHC. *Homepage*. URL: <http://aliceinfo.cern.ch>
- [Ale01] ALESSANDRO RUBINI, Jonathan Corbet (Hrsg.): *Linux Device Drivers*. Second. O'Reilly, 2001. – 586 S. – ISBN 978-0-596-00008-0
- [ALI01] ALICE COLLABORATION: ALICE Technical Design Report of the Transition Radiation Detector / CERN. Genf : CERN/LHCC, 2001 (CERN/LHCC 2001-021). – Forschungsbericht. – 246 S. – ISBN 92-9083-184-7
- [Bro] BROOKHAVEN NATIONAL LABORATORY. *Homepage*. URL: <http://www.bnl.gov/RHIC/>
- [CERa] CERN. *Homepage*. URL: <http://www.cern.ch>
- [CERb] CERN IT/CO. *Homepage*. URL: <http://cern.ch/itcobe/Projects/Framework>
- [Cho03] CHOCHULA, P. *Proposal for ALICE Front-end and Readout Electronics Monitoring and Configuration*. URL: <http://alicedcs.web.cern.ch/AliceDCS/Documents/FEEConfig.pdf>. 2003
- [Cla01] CLARA GASPAR. *JCOP Framework Hierarchical Controls Configuration & Operation*. URL: <http://lhcb-online.web.cern.ch/lhcb-online/ecs/fw/FSMConfig.pdf>. 2001
- [Cuv] DE CUVELAND, Jan: *Online Track Reconstruction of the ALICE Transition Radiation Detector at LHC (CERN)*. Heidelberg, Universität Heidelberg, Kirchhoff-Institut für Physik, Diss.. – Veröff. gepl.
- [Cuv03] DE CUVELAND, Jan: *Entwicklung der globalen Spurrekonstruktionseinheit für den ALICE-Übergangsstrahlungsdetektor am LHC (CERN)*. Heidelberg, Universität Heidelberg, Kirchhoff-Institut für Physik, Diplomarbeit, 2003. – 117 S
- [Dis] DISTRIBUTED INFORMATION MANAGEMENT SYSTEM. *Homepage*. URL: <http://www.cern.ch/dim>
- [ETM] ETM AG. *Homepage*. URL: <http://www.etm.at>
- [Fer] FERMILAB. *Homepage*. URL: <http://www-bdnew.fnal.gov/tevatron/>

- [Ger] GERLACH, Thomas: Heidelberg, Universität Heidelberg, Kirchhoff-Institut für Physik, Diplomarbeit. – Veröff. gepl.
- [Hig64] HIGGS, P. W.: Broken Symmetries and the Masses of Gauge Bosons. In: *Phys. Rev. Lett.* 13 (1964)
- [Hor04] HORST KUCHLING (Hrsg.): *Taschenbuch der Physik*. 18. neu bearbeitete Aufl. Fachbuchverlag Leipzig im Carl Hansen Verlag, 2004. – 711 S. – ISBN 3-446-22883-7
- [Inc07] INCORPORATION XILINX: PowerPC Processor Reference Guide. (2007)
- [Jac02] JACOB GORBAN. *UART IP Core Specification*. URL: <http://www.opencores.org/projects.cgi/web/uart16550/overview>. 2002
- [Jac05] JACOB GORBAN: Using and Creating Interrupt-Based Systems. (2005)
- [Kir07] KIRSCH, Stefan: *Development of the Supermodule Unit for the ALICE Transition Radiation Detector at the LHC (CERN)*. Heidelberg, Universität Heidelberg, Kirchhoff-Institut für Physik, Diplomarbeit, 2007
- [Mar07] MARCO BOCCIOLI, Giacinto De Cataldo. *Alice DCS FSM integration guidelines*. URL: http://alicedcs.web.cern.ch/AliceDCS/IntegrationDCS/examples/Alice_DCS_FSM_integration_guidelines_0.4.doc. 2007
- [OFF06] OFFLINE GRUP. *Shuttle Program for automatic data retrieval from DCS*. 2006
- [P+06] P. CHOCHULA, A. Augustinus [u. a.]. *Alice DCS integration guidelines*. URL: <http://alicedcs.web.cern.ch/AliceDCS/Documents/Rules/Integrationguidelines1.3.2.doc>. 2006
- [PRSZ04] POVH, Bogdan ; RITH, Kalus ; SCHOLZ, Christoph ; ZENTSCHKE, Frank: *Teilchen und Kerne : eine Einführung in die physikalischen Konzepte*. 6. Berlin ; Heidelberg ; [u.a.] : Springer, 2004. – 416 S. – ISBN 3-540-21065-2
- [Ret] RETTIG, Felix: Heidelberg, Universität Heidelberg, Kirchhoff-Institut für Physik, Diss.. – Veröff. gepl.
- [Ret07] RETTIG, Felix: *Entwicklung der optischen Auslese für den ALICE TRD Detektor am LHC (CERN)*. Heidelberg, Universität Heidelberg, Kirchhoff-Institut für Physik, Diplomarbeit, 2007. – 141 S
- [San04] SANDISK: SanDisk SD Card Produkt Manual. (2004)
- [Sas04] SASCHA MARC SCHMELING. *Joint PVSS and JCOP Framework Course*. 2004
- [SD] SD CARD ASSOCIATION. *Homepage*. URL: <http://www.sdcard.org>

-
- [SFF00a] SFF COMMITTEE. *INF-8074i Small Form-factor Pluggable (SFP) Transceiver MultiSource Agreement (MSA)*. URL: <http://www.sffcommittee.com/ie/Specifications.html>. 2000
- [SFF00b] SFF COMMITTEE. *SFF-8472 Specification for Diagnostic Monitoring Interface for Optical Xcvs*. 2000
- [Tea01] TEAM, JCOP F. *JOINT CONTROLS PROJECT (JCOP) FRAMEWORK SUB-PROJECT GUIDELINES AND CONVENTIONS*. URL: <http://alicedcs.web.cern.ch/AliceDCS/Documents/Rules/JCOPguidelines.pdf>. 2001
- [W-I] W-IE-NE-R. *Homepage*. URL: <http://www.wiener-d.com>
- [Xil04] XILINX LOGICORE: Processor Local Bus (PLB) v3.4. (2004)
- [Xil05a] XILINX LOGICORE: On-Chip Peripheral Bus V2.0 with OPB Arbiter (v1.10c). (2005)
- [Xil05b] XILINX LOGICORE: OPB IPIF (v3.01c). (2005)
- [Xil06a] XILINX LOGICORE: OPB Interrupt Controller (v1.00c). (2006)
- [Xil06b] XILINX LOGICORE: OPB Timer/Counter (v1.00b). (2006)
- [Y⁺06] YAO, W.-M. [u. a.]: Review of Particle Physics. In: *Journal of Physics G* 33 (2006), S. 1+

Erklärung zur selbständigen Verfassung

Ich versichere, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, im September 2007

Marcel Schuh