

Diploma Thesis

**Development of a
Real-Time General Purpose Online
Monitoring System
for HADES and FAIR
Experiments**

Borislav Milanović

November 5, 2009

This thesis is submitted for the degree of
Diplom-Informatiker
at the
University of Frankfurt

Advisor: Prof. Dr. Uwe Brinkschulte

Declaration

Hereby I declare that this thesis is a result of my own work, except where explicit reference to the work of others is made, and has not been formerly submitted to another university for degree purposes.

Abstract

The High Acceptance Di-Electron Spectrometer (HADES) is a large particle detector and at the same time an extremely complex device. It is located in Darmstadt, Germany at the GSI Helmholtzzentrum and has a wide application range in hadron and heavy ion physics. Currently, the entire GSI research facility undergoes an upgrade, as well as the detector. The electronics require a new bus system, faster data acquisition routines and higher tracking precisions to meet all the requirements for the upcoming experiments. The HADES spectrometer is composed of numerous sub-detectors and sub-systems. Recently, a unified bus has been developed to combine all the different detector parts together. Internal communication now applies a fast optical network with bandwidths of 1–2 *Gbit/s*. Although the system supports error detection and avoids deadlocks, it is of great importance to monitor the devices and analyze their data. For this occasion and to generate new, interesting statistics, a monitoring system needs to be developed.

The main task of this thesis is to create a comprehensive monitoring facility that provides a real-time access to the detector components, allowing precise analysis of the electronics, as well as additional hardware testing. Since all detector data needs to pass the readout nodes and the network hubs, all signals are present in a digitalized state on the FPGAs of these components. The implementation therefore buffers all the monitoring signals on the FPGA chips of front-ends, readout nodes and network hubs in highly customizable storage cells (FIFOs and registers). The buffered signals are read out through the optical network and gathered on a designated board operating as a monitoring server. The board uses a TCP/IP connection to extract the data to the client(s) over Ethernet. On the client side, the user may control the monitoring facility either through a Linux shell, or using an EPICS GUI, in which case the monitored data can be additionally visualized.

The key feature is to develop a system which is versatile and highly scalable. One source code is being used to suit all the sub-detectors. After a proper configuration, all the necessary information is stored on a ROM on every monitoring chip, thus the FPGAs work independently from each other and may contain different monitoring setups. Every signal possesses a timestamp and a synchronization of all signals can be achieved through different timer domains. The client PC is able to reconstruct the entire monitoring setup by reading out the ROM contents and then allows the user to send instructions to the server, grouping the chips together and visualizing them accordingly.

Zusammenfassung

In Darmstadt am GSI Helmholtzzentrum für Schwerionenforschung sind verschiedene Experimente zur Atom- und Kernphysik installiert. Hier befindet sich unter anderem der HADES Teilchendetektor (High Acceptance Di-Electron Spectrometer). Seine Aufgabe ist es, in 'fixed-target' Experimenten nach Elektron-Positron-Paaren zu suchen, die aus dem Zerfall von Vektormesonen (z.B. ρ^0) stammen und auf die Beantwortung der Frage zielen, wie die starke Wechselwirkung (QCD¹) die Massen der Hadrone² generiert. HADES wurde konstruiert um solche äußerst seltenen Ereignisse zu analysieren. Diesbezüglich setzt es eine Kombination aus mehreren unterschiedlichsten Subdetektoren ein.

Gegenwärtig befindet sich an der GSI eine weitere Forschungseinrichtung (FAIR) im Aufbau. Hier sollen zukünftige Experimente mit höherer Präzision und Datenraten installiert werden. In diesem Zusammenhang muss auch HADES an die neuen Anforderungen angepasst werden. Dazu wurde insbesondere das Trigger-System überarbeitet und die Geschwindigkeit der Datenauslese durch ein schnelles optisches Netzwerk verbessert. Die meisten elektronischen Bus-Komponenten besitzen mittlerweile optische Links mit denen eine Bandbreite von 1–2 Gbit/s erzielt werden kann. Erst dadurch eignet sich der Detektor für die neuen anspruchsvolleren Versuche.

Da sich weitere Subdetektoren im Aufbaustadium befinden und der Systemumbau noch nicht abgeschlossen ist, ist es wichtig alle Neuerungen und ihre Eigenschaften im Auge zu behalten und zu analysieren. Der Systembus ist zwar entwickelt worden um Deadlocks zu vermeiden, aber wenn es dennoch zu Fehlfunktionen kommt ist es von äußerster Wichtigkeit schnell die Ursache(n) dafür zu finden. Aus diesen Gründen wurde im Rahmen dieser Arbeit ein allumfassendes Monitoring-System entwickelt. Das System soll eine Möglichkeit bieten die Elektronik zu überwachen, alle Systeme während des Betriebs zu analysieren und Statistiken zu entwerfen.

Die Herausforderung dabei ist es ein System zu entwickeln, das sowohl auf allen gegenwärtigen Subdetektoren eingesetzt werden kann, als auch zukunfts offen ist. Auf der anderen Seite darf der Ressourcenverbrauch nicht zu hoch ausfallen um in die Auslesesysteme integriert zu werden. Das Monitoring Tool muss auf der anderen Seite jedoch Möglichkeiten bieten, die Hardware in Echtzeit zu analysieren. Es soll wie ein universeller Logic Analyzer überall dort Anwendung finden, wo digitale Komponenten eingesetzt werden.

Die HADES Auslese und Front-Ends verwenden FPGA-Chips. Die rekonfigurierbare Logik der FPGAs bildet das Grundgerüst der Datenauslese. Das ressourcen-schonende Monitoring-System ist auf diesen Chips ebenfalls mitzuintegrieren. Das bietet sich an, da alle Daten ohnehin in den Front-End-Modulen zur Verfügung stehen. Das Monitoring System läuft unter dem Namen DEMON (real-time DEvice MONitoring framework) und wird auf vielen unterschiedlichen

¹Quanten Chromo Dynamik

²Hadronen sind sub-atomare Teilchen die Quarks (spezielle elementare Teilchen) enthalten.

Detektor-Chips der Auslese-Elektronik und in den meisten Fällen direkt auf den Detektor Front-Ends laufen. Die Konfiguration setzt dabei einen orthogonalen Ansatz ein. Wenn man das System auf einem Chip ändert, darf es nicht die Arbeitsweise anderer Systeme oder deren Monitoring Software beeinflussen. Aus diesem Grund werden alle Konfigurations-Informationen lokal auf dem Chip in einem ROM gespeichert. Das ROM ermöglicht die Rekonstruktion aller detektor-spezifischen DEMON Einstellungen des Chips.

Über eine FPGA-Schnittstelle werden die Daten an DEMON weitergegeben. Dieses, als IP-Core eingesetzte Modul, puffert die Signale in Form von Rohdaten in den FIFOs und Registern und versieht sie zusätzlich mit einem Zeitstempel. Um den Anforderungen zu genügen, wurden viele FIFO-Arten vordefiniert und mehrere Timer zur Verfügung gestellt um sich jeder Situation anzupassen. Die Breite, Tiefe, Aufnahmefrequenz, Timer-Informationen, und vieles mehr läßt sich über eine zentrale Konfigurationsdatei einstellen. Die Informationen dieser Datei werden direkt in ein embedded ROM übernommen und das Monitoring dementsprechend auf dem FPGA aufgebaut. Der Benutzer ist anschließend dafür zuständig, dass die Daten aus den Chips herausgelesen werden. Als Möglichkeiten wurden bisher eine Linux Kommandozeile und eine EPICS Oberfläche implementiert, mit Hilfe derer man Anweisungen über eine Ethernet-Verbindung an das optische Netzwerk weitergeben kann. Dort ist dann ein Monitoring-Server in der Lage, die Signale vom DEMON auszulesen und an den Client zu übermitteln. Hier lassen sich viele Signale gruppieren um gleichzeitig ausgelesen zu werden, als auch vom Benutzer spezifizierte Auslese-Intervalle einstellen.

Contents

1	Motivation and Overview	1
1.1	HADES System	3
1.2	Monitoring Facility	6
1.2.1	Overview	6
2	Introduction	7
2.1	The HADES Data Acquisition Scheme	7
2.2	Overview of the Electronics	8
2.2.1	The TRB	8
2.2.2	TRBnet Details and Triggering	11
2.3	Read and Write Access	14
2.4	Room for Monitoring	15
3	Analysis and Design	16
3.1	Basic Specifications	17
3.2	Level 1 Trigger Electronics	17
3.2.1	The Start/Veto Detector	18
3.2.2	The TOF/TOFino System	19
3.2.3	Multiplicity Triggering	20
3.3	Level 2 Trigger	20
3.3.1	The RICH Detector	21
3.3.2	The Pre-Shower Detector	23
3.3.3	Di-Electron Pattern Analysis	24
3.4	Particle Identification	26
3.4.1	The MDC Detector	26
3.5	Analysis Results	28
3.6	Monitoring Design	29
3.6.1	Signal Properties	31
3.6.2	Orthogonal Design	33
3.6.3	Signal Readout	33
3.6.4	Conclusion	34
4	Implementation	36
4.1	Monitoring System Layout	36
4.2	Hardware Monitoring Section	36
4.2.1	DEMON Configuration File	38
4.2.2	The Input Interface	41

4.2.3	The FIFO cell	44
4.2.4	Address Space	47
4.2.5	DEMON Summary	48
4.3	Software Monitoring Section	49
4.3.1	The Server	49
4.3.2	The Client	51
4.4	EPICS Application	53
4.4.1	Brief Introduction	53
4.4.2	GUI Realization	54
4.5	Summary	56
5	Evaluation of the Work	58
5.1	DEMON Tests	58
5.1.1	Performance Measurements	60
5.2	Possibilities	62
5.3	Evaluation	64
5.4	Closing Remarks	67
A	Explanation of '<i>DEMON_config.vhd</i>'	73
B	FIFO Frequency Map	76
C	Generate-Statement for the FIFOs	77
D	Source Files	79

List of Figures

1.1	Theoretic structure of a nucleon	1
1.2	Collision between two gold ions	2
1.3	GSI and FAIR complex	4
1.4	HADES detector 3D view and the collaboration	5
2.1	TRBnet data flow	8
2.2	TRB photo	9
2.3	TRB architecture	11
2.4	Optical link	11
2.5	The star-like architecture of TRBnet	12
2.6	Data transportation and OSI layers	13
2.7	TRBnet full scheme	14
2.8	Monitoring access through RegIO	15
3.1	HADES pipe system	16
3.2	FEE to TRBnet bridge	17
3.3	Start and Veto detector operation	18
3.4	Start/Veto detector photo	18
3.5	TOF detector	19
3.6	TOFino paddles	19
3.7	TOF add-on readout scheme	20
3.8	TOF add-on picture	20
3.9	RICH detector cross-section	21
3.10	RICH data acquisition scheme	22
3.11	RICH ADCM photo	22
3.12	RICH ADCM architecture	22
3.13	Pre-Shower operating method	23
3.14	Pre-Shower readout scheme	24
3.15	Pre-Shower add-on	24
3.16	RICH image algorithm	24
3.17	Level 2 trigger replacement	25
3.18	HADES trigger decision algorithm	25
3.19	HADES tracking with MDC	26
3.20	MDC detector illustration	27
3.21	MDC readout scheme	28
3.22	MDC readout electronics	28
3.23	Complete TRBnet scheme with all add-ons	29

3.24	Timer chaos	30
3.25	Monitoring interface scheme	31
3.26	FIFO specifications	33
3.27	The ROM illustration	34
3.28	Monitoring architecture inside TRBnet	35
4.1	Monitoring principle	37
4.2	Interface and bus width	39
4.3	Configuration cell	40
4.4	Datapacket contents	40
4.5	The input interface.	41
4.6	Control bits	43
4.7	The FIFO controller logic	44
4.8	Ringbuffer behavior analysis	46
4.9	Scheme of the hardware part	48
4.10	Server console output	49
4.11	Client console menu	51
4.12	EPICS system architecture	53
4.13	A simplified EPICS setup	54
4.14	EPICS test GUI	56
4.15	Complete DEMON architecture	57
5.1	DEMON hardware test	59
5.2	TRBnet test setup	60
5.3	TRBnet as a tree	62
5.4	DEMON extensibility	64

List of Tables

2.1	Data acquisition scheme	7
2.2	TRB components	9
2.3	TRBnet channel list	12
2.4	TRBnet communication layers	13
3.1	Monitoring system requirements	17
3.2	Important monitoring signals	30
3.3	Timer types	32
4.1	Summary of all FIFO types	45
4.2	Address range splitting	47
5.1	FIFO test results	58
5.2	Resource consumption test	61
B.1	Frequency map	76
D.1	Source Files	79

1 Motivation and Overview

Heavy ion physics, as a branch of modern particle physics aims to discover the principal interacting forces between microparticles, as well as the corresponding elemental particles which cause them. Thousands of years ago, the ancient Greek philosopher Democritus was occupied with an interesting question: what are actually the basic components of matter and what kind of forces keep them together? Since then, no final answer could be given. It seems that the deeper one can look inside the matter, the more particles and structures can be discovered. During the last century however, revolutionary breakthroughs in the field of physics have been performed, which imposed a completely new way of thinking in order to find the answer to the above question. In that era of Einstein and many others, the fields of particle and nuclear physics were reborn.

The main focus lies thereby on the atomic **nucleus**. The nucleus itself is very hard to examine in a laboratory, but yet it seems to contain the answer to the above question. In order to gain a clearer look inside, large energies and expensive instruments are needed. It is known that the nucleus is composed of smaller particles, **the nucleons**, however it still remains a puzzle how their mass is generated and it is uncertain what the complete properties of their interacting forcefields are. This is currently a highly debated question and the focus of many research studies throughout the world. A picture showing one nucleon (the proton) from three different perspectives can be viewed in figure 1.1.

One huge challenge of modern particle physics lies within the fact that the nucleon constituents can never be isolated or observed freely. It can be apprehended as a form of nature's law. One part of the nucleon are the three so called valence quarks¹ and they seem to be confined

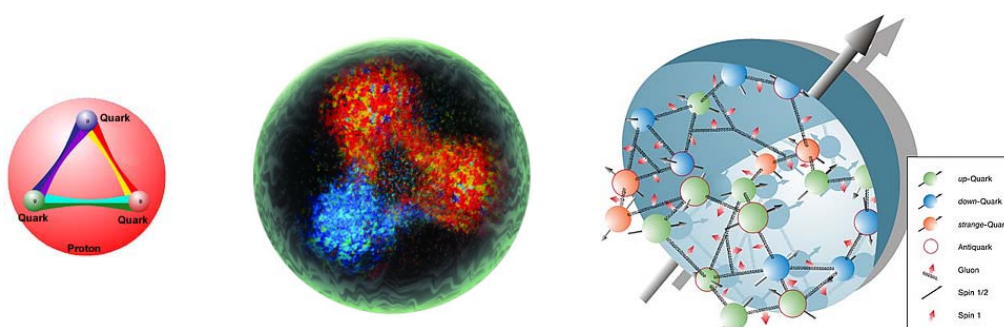


Figure 1.1: The current theory behind the nucleon. Its inner particles are bound with the strong force (QCD) and can not be isolated or separated without further ado. **Sources:** Internet (Google images)

¹Quarks are elemental particles and can neither be destroyed nor isolated. There are six in total (twelve with their corresponding anti-particles) and they are the main components of all matter on earth.

inside it. Since the mass of the three quarks is far below the actual value for the entire nucleon², it is believed that many other, virtual particles evolve around them carrying the missing mass in form of energy. Besides the quarks, there are hence diverse virtual particles being created and destroyed at the same time, interacting in different ways with the forcefields in this equilibrium, keeping the net mass of the nucleon at its value.

In general, many different particle species exist. The atomic shell, for instance, contains electrons which belong to the **lepton** family. Electrons are also responsible for the electric current and they hold the molecules and atoms in the solid and liquid states together. However, they possess an anti-matter opponent – the positron (i.e. the anti-electron), which is also a member of the lepton-family. In fact, every particle of common matter does possess a corresponding anti-particle. Whenever electron and positron collide, they annihilate each other, creating strong gamma radiation. This holds for every particle/anti-particle pair. While leptons are usually rather lightweight, there is the heavier family of **hadrons**. Hadrons can contain one quark and one anti-quark, in which case they are called mesons, or they can build up a nucleon or a similar particle called the baryon/anti-baryon. Baryons contain three valence quarks exactly and anti-baryons three anti-quarks, in fact all hadrons contain quarks or anti-quarks, whereas leptons do not. Leptons are already elemental particles. Hadrons are only elemental in the sense that the quarks they contain (which are the true elemental particles) can not be isolated. Hence hadrons contain other elemental particles but they themselves behave as if they were elemental, too. The branch of physics concerned with the properties of hadrons in bound states is called hadron physics.

In order to study physics at such inordinately low levels as $10^{-15}m$ and below, an accelerator facility with a particle detector is necessary. Since nuclei, as well as nucleons, are rather complex systems composed of many particles accompanied by their interacting forcefields³, accelerating them near lightspeed and 'smashing' them against one another can reveal an insight into their inner structure. Figure 1.2 illustrates such a collision between two heavy ions.

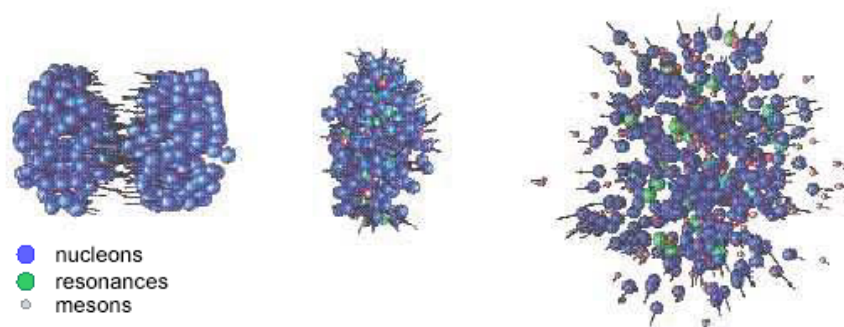


Figure 1.2: A simulation of a central hit between two gold atoms accelerated near lightspeed against each other. At such high energies, new particles, in form of hadronic resonances and mesons are easily created according to Einstein's $E = mc^2$. *Source:* [Lor08]

²Quark mass is around 5 – 10 MeV, whereas the mass of the entire nucleon lies around 1000 MeV.

³Forcefields can be completely described using interacting particles – the bosons.

After the collision, the products of the fireball are emitted in all directions in space. Due to the fact that new particles are created, even completely new particle species or whole particle jets⁴ can be formed. In physics, the conservation laws of energy and momentum can never be violated. This applies even for quanta. Therefore, every particle involved in the collision still obeys certain laws of physics, and by determining every product after the collision alongside its physical properties, the structure of the fireball can be analyzed and new theories explaining the substantial properties of energy and matter tested and verified. Mesons sometimes decay directly into electron-positron pairs and since the lifespan of a meson is in some cases significantly smaller than that of the entire fireball, by measuring the di-electron⁵ energy, the meson mass inside the fireball can be determined and hence inaccessible information about the evolution of mass inside dense matter acquired. This is exactly the type of experiment widely applied in hadron physics.

Although Democritus' question still remains unanswered, current state of the field nevertheless provides extremely good estimates and highly accurate calculations and explanations about the governing principles of the universe. By studying the smallest, even such objects like supernovae, neutron-stars and solar radiation can be explained⁶.

1.1 The HADES Detector System

The GSI Helmholtzzentrum [GSI] is a center for heavy ion research located in Darmstadt near Frankfurt, Germany. Currently, the accelerator complex is being used for various experiments regarding nuclear and atomic physics. The complete center consists of a linear accelerator, a synchrotron, an ion storage ring, several detectors and a facility for tumor therapy with high-energetic carbon ions [GSIB]. Currently, the entire research establishment is being upgraded. First experiments are planned for the year 2015 and the new accelerator facility, which runs under the name FAIR (Facility for Antiproton and Ion Research) [GSIC], will provide an opportunity for more sophisticated experiments involving higher energies, anti-matter and ion beams of much higher quality. In total, the expenditure amounts to 1.2 billion euro. An illustration of the current facility and the upgrade can be seen in figure 1.3.

One of the several particle detectors on the complex is the HADES (High Acceptance Di-Electron Spectrometer) detector system [GSID, HAD09], which is tailor made for the detection of rare events producing electron-positron pairs. Di-electrons are excellent probes for the analysis of nuclear matter. They themselves do not interact strongly with nuclear matter (since they do not contain any quarks) and are due to their low mass ideally suited for collision experiments at lower energies of several GeV per nucleon. In such collisions, new vector-mesons can be created (for example the ρ -meson), which can immediately decay into di-electrons. By measuring the di-electron momenta, the properties of the meson inside the fireball can be understood, as well as additional fireball properties. By doing so, HADES experiments can reveal novel aspects

⁴In this case the impact energy is converted into mass and numerous new particles conserving the laws of physics re-emerge.

⁵Di-electrons denote the electron-positron pairs, since electron and positron are the same particle, only with the opposite charge (positron = anti-electron).

⁶Such field of study is referred to as 'nuclear astrophysics'

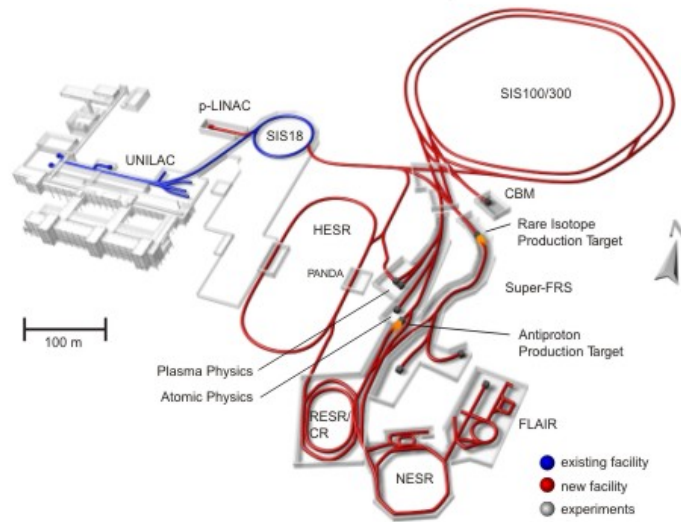


Figure 1.3: The entire GSI research facility. The upcoming FAIR complex is depicted in red. *Source:* [GSI]

to nuclear matter and energy/mass spectra which are normally confined within. In contrast to traditional scattering experiments, di-electrons allow particle analysis from the inside of nuclear matter.

Measuring di-electrons originating from (and indicating) special events like vector-meson decays has always been a great challenge. First of all, the branching ratio of such a decay is extremely low making it especially hard to emerge from a collision⁷, and second, the processes in a collider are always accompanied by background radiation which can mask the good di-electrons indicating real ρ decays with bad pairs originating from ordinary background radiation. Furthermore, di-electrons are extremely lightweight, hence all detector parts must be designed as permeable as possible, not to disturb their path before measurement⁸. Background radiation can also induce conversion processes in the detector parts causing more unwanted effects. Therefore, all inner HADES components were designed extremely lightweight and thin in order to minimize the undesired effects [Lan08].

Behind a big name, there is always a great collaboration. Over one hundred researchers throughout Europe are involved in the HADES project. A map of all the contributing countries is shown in figure 1.4. Experiments have been conducted since 2002 and its original purpose was to clarify the *DLS-puzzle* from the DLS experiment at BEVALAC in Berkley, USA. The puzzle has been solved [Pac08a] and the discrepancy between theoretical calculations and experimental results confirmed. Yet, the deviance can not be explained by any theoretic model and additional

⁷Only every 10 000-th ρ -meson actually decays directly into a di-electron. The process rather prefers a conversion into heavier particles like the pions or pure gamma radiation.

⁸While electrons move through matter at ultra-relativistic speed, they get slowed down and emit bremsstrahlung. Bremsstrahlung creates secondary electrons which interfere with particle measurements. In the Pre-Show sub-detector however, this effect is actually favored to detect electrons (see section 3.3.2).

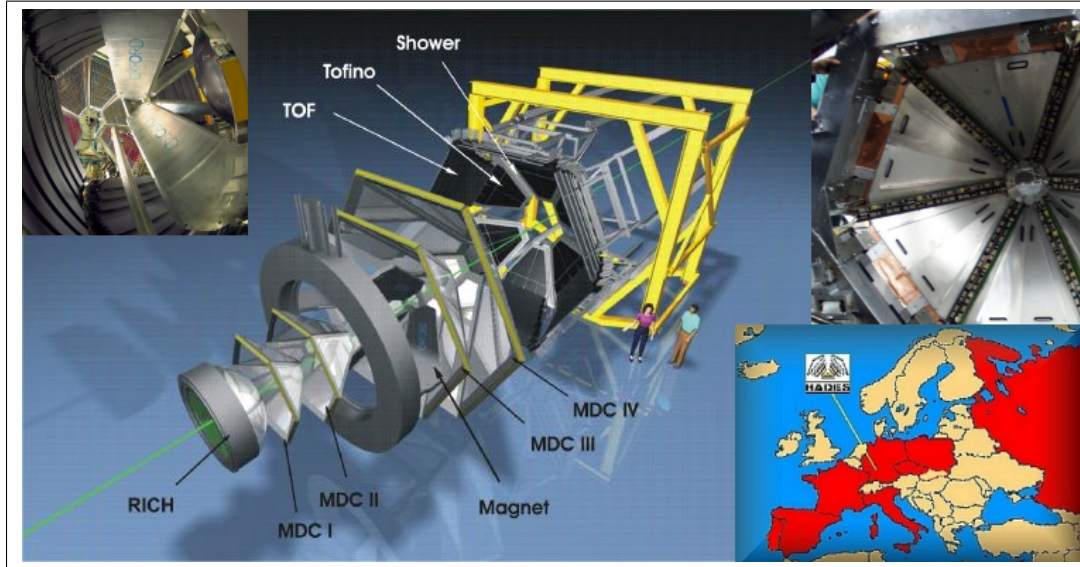


Figure 1.4: The HADES detector in exploded view, showing the numerous sub-detectors in layers. The collaboration involves 9 countries and 17 institutions. *Source:* [GSId]

research in this field is still required.

The HADES detector system is designed specifically for fixed-target⁹ experiments. Interesting events can be observed only in one hemisphere in the lab frame, which is ideal for detecting di-electrons with large opening angle, usually originating from vector mesons with large mass. The detector geometry can be viewed in figure 1.4. While the azimuthal angles are almost completely covered due to the six trapezoidal sectors spreading in every direction in space, the polar angles cover 18 - 85 deg. Each of the six sectors is built alike and the detector is perfectly six-fold symmetric. Due to the large opening angles, the geometric acceptance is very high allowing detection of approx. 40 % of all di-electrons. All the different layers from figure 1.4 consist of different sub-detectors (TOF, TOFinO, Pre-Shower, RICH and MDCs). As already mentioned, many different particles are created during the experimental runs, therefore distinct detectors have been designed to analyze each significant particle group. The RICH- [Zei99] and the Pre-Shower-detector [Bal04] are used for di-electron detection, while serving at the same time as triggers for rare events. MDCs [Mun04] together with the toroidal supra-conducting magnet can identify charged particle tracks alongside their momenta. The TOF [Ago02] and TOFinO allow particle time of flight measurements and contribute to additional particle identification. In order to collect sound statistics, not all events can be measured and recorded. Due to the large data loads, a triggering system [Tra00] has been developed to allow an online discrimination of events and storing only the wanted, rare ones for further analysis. As mentioned above, the generation of hadron mass is currently one of the most interesting problems in particle physics. After the FAIR upgrade, HADES should be able to collect better statistics on larger systems (e.g.

⁹In fixed-target experiments, and in contrast to collider experiments, the ion beam is accelerated whereas the target atoms remain still. This sort of experiment produces particles only in forward direction and requires higher energies than collider experiments.

$Au \rightarrow Au$ with 10 GeV) and acquire more information to understand mechanisms responsible for hadron interactions, behavior and structure.

1.2 Monitoring Facility

As already mentioned, the HADES detector system is currently being redesigned due to the FAIR upgrade [Fro09]. As will be explained in the subsequent chapter, the sub-detectors now acquire data over FPGA-chips and then forward it over a fast optical network for trigger-analysis. If an interesting event has occurred, the data from all sub-detectors is stored for further evaluation. Thus the data acquisition electronics need to be highly flexible and have to rest upon one unified architecture for all sub-detectors. To this effect, a general-purpose readout board, the TRB [Fro08], has been manufactured. Moreover a centralized bus system, the TRBnet [Mic08], has been developed. Although the detector is composed of many different parts, in this way, the data acquisition relies on the same architecture.

Apart from that, there is a huge space limitation inside the MDC detector. The front-end components once mounted can not be accessed anymore. Moreover, the chips and the electronics are exposed to tremendous radiation during the experiments. The radiation however does not seem to have a threatening impact on the data. In worst case, a bit is flipped in an SRAM cell¹⁰ and the data has to be discarded or the FPGA¹¹ reprogrammed. The bus system is conceived to avoid deadlocks, however if something does go wrong, it is of great importance to analyze the reason(s). For this occasion and to generate new, interesting statistics, as well as to keep an eye on all devices, a monitoring system needs to be developed.

1.2.1 Overview of the Work

As part of this thesis, a general-purpose, scalable monitoring system has been developed to support HADES and future FAIR experiments. The system draws on the FPGA-driven bus architecture and can therefore be utilized on any FPGA chip. It has been designed strongly resource-friendly and is considered as lightweight. It allows critical signal monitoring (like temperature, voltages, busy times and similar) as well as statistical conception and detector analysis. It can be used for example as a logic analyzer providing real-time hardware signals, or for various testing purposes.

In the next chapter the bus architecture and the triggering system will be introduced in more detail, as well as the hardware they reside on. Afterwards, in chapter 3, the requirements and the design of the monitoring system will be elaborated after an in-depth detector analysis. After that, in chapter 4, an implementation based on VHDL and C will be explained, leaning on the EPICS API. Finally, chapter 5 presents a summary and an evaluation of the entire work, accompanied by future plans and an outlook.

¹⁰Reconfigurable FPGA chips usually use Static RAM cells (SRAMs) to store their current configuration and program.

¹¹The FPGA will be explained in the following chapter.

2 Introduction

The HADES system combines numerous different components with modern and most demanding technology. This chapter introduces the readout operation and explains some basic data acquisition principles, as well as the internal optical network.

2.1 The HADES Data Acquisition Scheme

The HADES detector system is responsible for detailed hadron and lepton analysis. Hardly approachable states of nuclear matter need to be analyzed on the basis of vector meson decays in medium. Vector meson events, again, should be evaluated on the basis of their di-electron decays. Additionally, all particles resulting from such events need to be identified. However, not all events are interesting. Only central collisions are likely to produce vector mesons, hence the data load can be significantly reduced by examining only the central hits¹. If a hit was central enough, the data needs to be checked once more whether an actual di-electron occurrence could be detected. Therefore, a two-level triggering system has been developed to decide which of the millions events per second are interesting and which not.

The great challenge of HADES experiments lies in the incredibly low decay rate of vector mesons into di-electrons (the di-electron branching ratio). It is of size $10^{-4} - 10^{-5}$, meaning that every 10 000 – 100 000 central collisions, one desired vector meson decays directly into a di-electron. Therefore the online trigger analysis needs to be performed with high rates, i.e. every of the 1 000 000 events per second in which the beam reacts with the target needs to be examined carefully.

According to [Pal08], the data acquisition is performed within five major steps, as summarized in table 2.1. The sub-detectors are applying certain technologies allowing them to detect specific particles flying through. These technologies are referred to as **front-end electronics**

Step	Data Acquisition Task
1	Front-end electronics
2	Readout electronics
3	Trigger distribution and data flow
4	Central-Trigger System
5	Event building and mass storage

Table 2.1: The HADES data acquisition procedure can be arranged into these steps.

¹Besides the 'good' hits, some *minimum bias* events are also recorded to support more precise analysis.

(FEE for short) and strongly vary from system to system. With FEE, the detectors acquire event data and send it to the next readout node. Afterwards, the **readout electronics** prepare the data for transport through the network system. In step 3, data is forwarded through the bus network² and made available for trigger analysis. A **central-triggering system** (CTS) is then responsible for quick online decisions, whether the data contains the desired di-electrons, or not. If a trigger does occur, i.e. the data contains certain characteristics and exhibits unusual patterns indicating possible di-electron presence, then the entire event requires an in-depth analysis plus reconstruction and has to be stored. Therefore, in the last step, the **event-builder** assembles coherent data packets coming from all different sub-systems and records them on the external tape. A data flow illustration is depicted in figure 2.1.

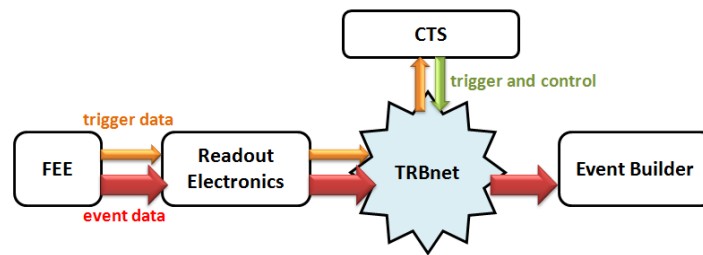


Figure 2.1: The basic event data flow. While waiting for trigger decisions, the data is buffered in different pipes.

2.2 Overview of the Electronics

An in-depth analysis on the operating method of all sub-detectors currently in use, together with their FEE, readout systems and properties will be presented in the subsequent chapter. However, this section will briefly outline that subject-matter and focus more on the optical network.

One major step during the HADES upgrade was to unify all data acquisition systems in one common bus. Every sub-detector needs to support this bus system, regardless of the technology it employs. Since the readout method strongly depends on the sub-detector it resides on, every detector must use a specialized board for data digitization and data distribution over the bus system³. This board has the purpose to prepare the data in the bus-specific format and rests upon a universal architecture presented in the remaining sections.

2.2.1 The TRB

Data acquisition at HADES is accomplished using general purpose **Trigger and Readout Boards** (TRBs). These boards force different detectors to use one common bus system named **TRBnet**. Since many different teams are developing the sub-detectors, it is of great importance to connect all of them with a unified data bus. Every detector applies rather complicated technology that

²The bus network is called TRBnet, see section 2.2.2

³In some cases, the conversion is even performed directly on the front-ends.

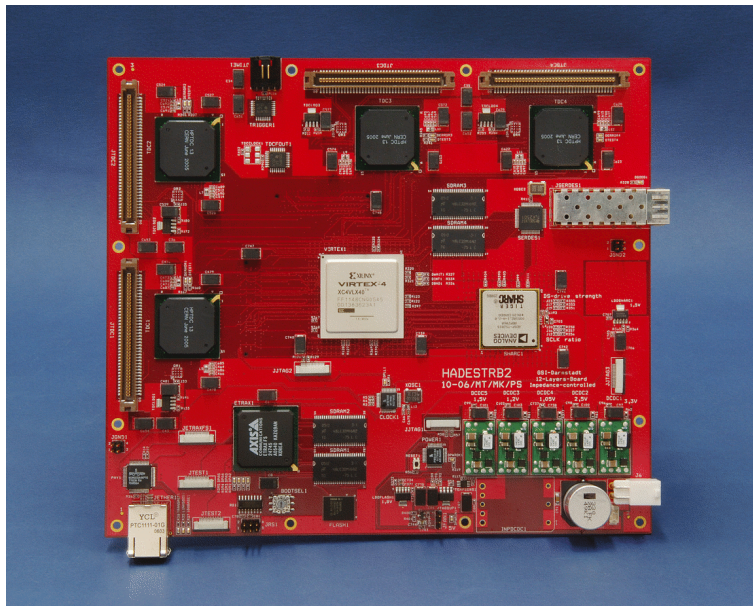


Figure 2.2: The general purpose trigger and readout board model 'v2b'. The TRB is the key element of data acquisition at HADES. *Source:* [Fro08]

is hard to comprehend especially for other teams, and therefore using a common bus allows everyone, particularly future developers, to acquire data without possessing much knowledge about the actual technology behind it.

Every network node inside the TRBnet consists of either one TRB containing eventually additional add-on boards directly attached to it, or an optical FEE driver board. The TRB opens up various possibilities. It has a highly generic design applying reconfigurable logic. The current design is marked under the version 'v2c' and (as shown in figure 2.2) contains diverse components summarized in table 2.2.

Component	Description
FPGA	Xilinx Virtex 4 family - LX40 (XC4VLX-10FF1148).
CPU	ETRAX FS with Linux 2.6 kernel and 128 MB RAM.
Optical Link Transceiver	Internal data flow inside the TRBnet with 2 Gbit/s.
Ethernet Connector	100 Mbit external network connectivity.
DSP	500 MHz TigerSharc digital signal processor.
HPTDC	Four multi-hit time-to-digital converters.
LVDS/TTL Connectors	Ultra fast connectors for add-on boards (15 Gbit/s bandwidth).

Table 2.2: The current TRB 'v2c' components.

Short Component Description

- **FPGA.** The electronics throughout the detector often utilize FPGA chips for data acquisition and slow control. A good explanation on these chips can be found in [Sik01, Okl08a, Tin00]. In short, an FPGA (**F**ield **P**rogrammable **G**ate **A**rray) represents reconfigurable logic cells. Initially, it possesses a fixed number of these cells that can be (re)configured to realize Boolean functions [BV05] and hence allow digital logic circuits. Moreover, the wiring between the cells is also reconfigurable, normally using SRAM cells. The logic cells are mostly implemented as Look-up Tables (see [BV05] for details). This kind of electronics is extremely versatile, as it allows implementations of arbitrary digital components, from simple ROMs and FIFO modules up to microcontrollers, processing units, and much more. They can be easily programmed with hardware description languages, like VHDL [BV05] and Verilog⁴.
- **HPTDC.** TDCs (Time-to-Digital Converters) play an essential role in every particle detector electronics, as they convert time intervals triggered by analog inputs with a certain resolution into digital signals ready for processing. They are similar to ADCs (analog-to-digital converters), only they do not convert the analog signal directly, but they count the time between two successive analog signals above a certain threshold and digitize this value afterwards. The resolution is very high and usually below 1 ns allowing time-sensitive event tracking. In HADES experiments, a particular TDC module, the **H**igh **P**erformance general purpose **T**ime to **D**igital **C**onverter from CERN [Chr04] is being used.
- **LVDS.** **L**ow-**V**oltage **D**ifferential **S**ignaling is an IEEE standard since the year 1996 (IEEE std 1596.3-1996). It enables extremely high transmission speeds over inexpensive twisted-pair copper cables with very low power consumption. A more detailed and comprehensive description can be found in [Tex02].

TRB Architecture

The FPGA is wired directly to the ETRAX CPU allowing user-space control over a Linux operating system. While the powerful Xilinx Virtex 4 FPGA provides main calculations, some slow control data and monitoring signals can be utilized over the CPU interface. The complete TRB architecture can be observed in figure 2.3.

One major advantage of the TRBnet is the internal network communication based on fast optical links. The optical links have already contributed to saving a great amount of space within the detector, since they consist of plane wires with negligible small diameter, as shown in figure 2.4. Moreover, the noise in the communication channels has been significantly reduced, but the most important improvement is the 2 Gbit/s network bandwidth using the Texas Instruments TLK2501 transceiver.

⁴Both languages are IEEE standards. The first VHDL version is noted under IEEE 1076, while shortly after that, another version as IEEE 1164 has been released [BV05]. Verilog is known as IEEE 1364.

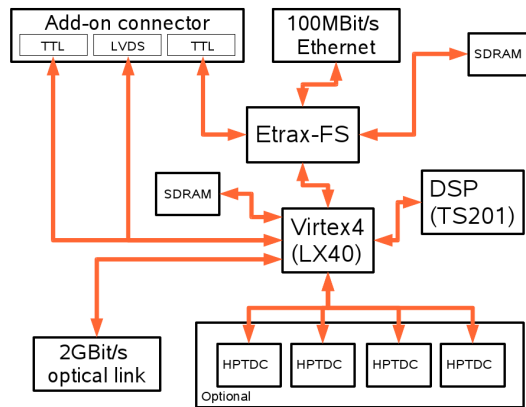


Figure 2.3: The entire TRB architecture. The ETRAX CPU provides direct access to the FPGA over two 16 bit lines. *Source:* [Fro08]

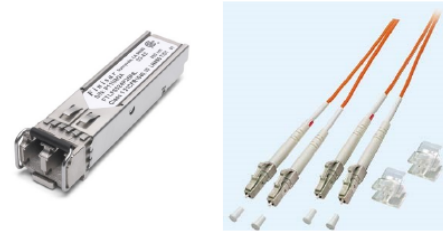


Figure 2.4: Optical link connector with its cables. *Source:* [Mic09]

The HPTDCs can be used for immediate signal digitization, allowing the TRB to function directly as a detector readout node. This is currently the case with the TOFinio and will also be performed in near future with the next upcoming sub-detector, Resistive Plate Chambers (RPC). The DSP is intended to reduce the FPGA workload in future experiments.

Although the TRB can be used as a stand-alone data acquisition component, it is also equipped with ultra fast LVDS connectors allowing additional boards to be attached to it. Many unique boards have been designed to support the new readout system. They serve mainly as bridges from the FEE to the TRBnet. While the FEE provide core data signals, the readout electronics acquire them and either forward the data directly through optical links to the next hub, or they send the data over the LVDS lines to the attached TRB for transmission. During this process, the data is converted into a TRBnet specific format⁵, as explained later in section 2.2.2. In this way, a clear separation between the detector technology and the bus system could be accomplished. More technical details on this issue are presented in chapter 3.

The optical link bandwidth amounts to 2 Gbit/s. The FPGA is running at 100 MHz and hence can forward the 16 bit wide data stream with 1.6 Gbit/s maximum. However, together with the entire TRBnet protocol, the total bandwidth averages around 1.2 Gbit/s, which is sufficient for future experiments [Mic08].

2.2.2 TRBnet Details and Triggering

One basic principle of the TRBnet bus system is that it transmits the data over four prioritized channels, as listed in table 2.3. The first level trigger is fired whenever the sub-detectors show high particle-activity. Since this is the most demanding part of the system bus, no other data but triggers are distributed over the first channel⁶. High particle quantity implies a central hit and therefore the data needs to be checked for rare events⁷.

⁵Some detectors, like MDC and RICH apply many data channels. In this case the data is converted directly on the FEE into the TRBnet format. The readout nodes only merge the numerous data channels, afterwards.

⁶The detector busy-activity can be directly ascertained here.

⁷Interesting events are most likely occurring in central hits, similar to those shown in figure 1.2

Channel	Data	Priority
1	Level one trigger	****
2	Level two trigger	***
3	Miscellaneous	**
4	Slow control and Monitoring	*

Table 2.3: TRBnet channel list. The first channel has top priority.

As mentioned in the previous chapter, HADES uses electron-positron pairs as probes for special events. They can be detected with the RICH- and the Pre-Shower-detector, therefore their data, together with TOF data, is an essential part of the second level trigger. This trigger inspects the data for specific patterns indicating di-electrons and decides if the event should be discarded or not.

Besides the TRB and the detector-specific readout boards, the network also contains **hubs**. These are yet other specific boards connecting up to 20 network nodes together. They are a substantial part of TRBnet and have their logic realized on two Lattice LFSCM25 FPGA chips per board, on top of which a TRB resides as an add-on for more controllability. Whereas the hubs provide constant data flow, there are some dedicated TRBs in the network performing all the calculations and trigger analysis. These particular boards with specific algorithms constitute the CTS and other data processing nodes (like the image processing unit and the matching unit, see section 3.3.3). All the network nodes are organized in a star-shaped arrangement, keeping the latency at minimum. The simplified network architecture is presented in figure 2.5.

Another aspect of TRBnet is its robustness and deadlock-freedom. The hardware is exposed to tremendous radiation and high temperatures, thus it tends to malfunction. However, if a TRB becomes inoperative, it should under no circumstances block the rest of the network. Therefore, the TRBnet has been designed to continue operating even if some vital parts are failing. More detailed explanations around TRBnet can be found in [Mic08].

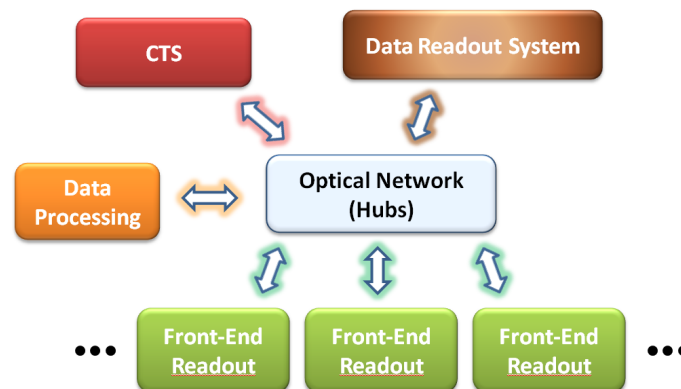


Figure 2.5: The abstract star-like architecture of TRBnet.

TRBnet Data Transport

The TRBnet bus system is based on the simplified Open Systems Interconnection reference model [Mic08]. FPGAs on every TRB in the network implement four layers of data transportation, as listed in table 2.4.

Layer	Data-packet
Application layer	Raw data ready for transport
Transportation Layer	Header, data and a termination tag
Link layer	Augmented data, plus additional end-of-buffer tags
Media interface	The augmented data from previous layers in 16 bit format

Table 2.4: TRBnet transportation layers according to the OSI layer model. Transmitter packs the data downwards, while the receiver unpacks it upwards through the layers.

The top layer provides the raw data, in other words, one or more applications residing on the FPGA may process the data and execute certain algorithms. When finished, they have to forward it for transport to the next network node. The raw data gets a header and a termination-tag in the transportation layer, so that the low-level logic can resolve when data starts and when it ends. Afterwards, in the link layer, the data is prepared for transport on the medium. There, it is marked with additional output buffer tags and moved into the buffers. Afterwards, in the lowest layer, it is finally sent through the optical link in 16 bit format to the next TRB or hub.

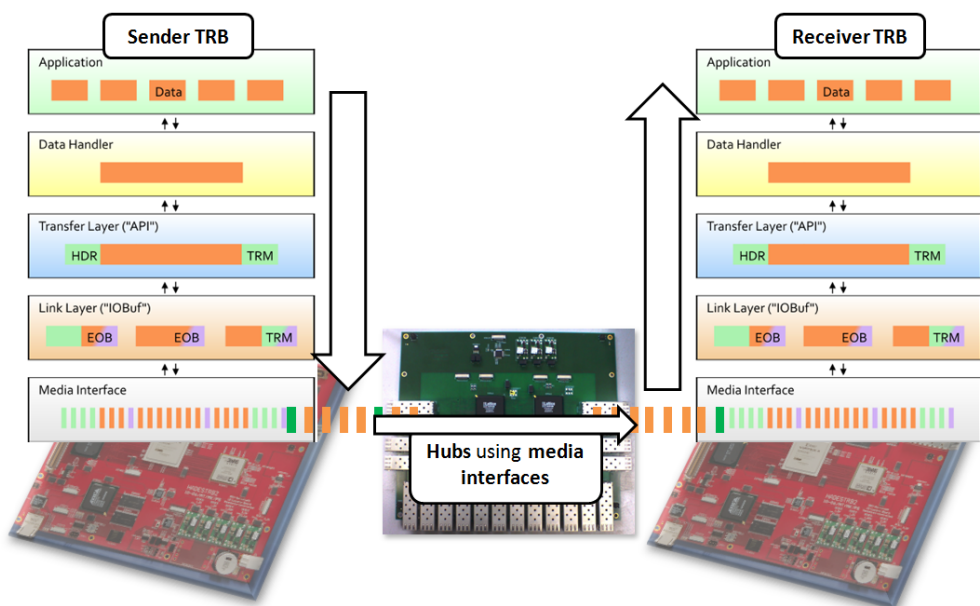


Figure 2.6: Data transportation through the OSI layer of TRBnet.

On the receiver side, the entire process is then reversed in order to obtain the data back. In figure 2.6, the data flow within the OSI-layers from one TRB to another is depicted. Every data packet contains 80 bits in TRBnet format, 64 of which are the raw data and altogether transmitted in five 16 bit words. An CRC checksum is also transmitted providing error detection. Figure 2.7 shows a scheme of the current network setup in combination with the data acquisition routines.

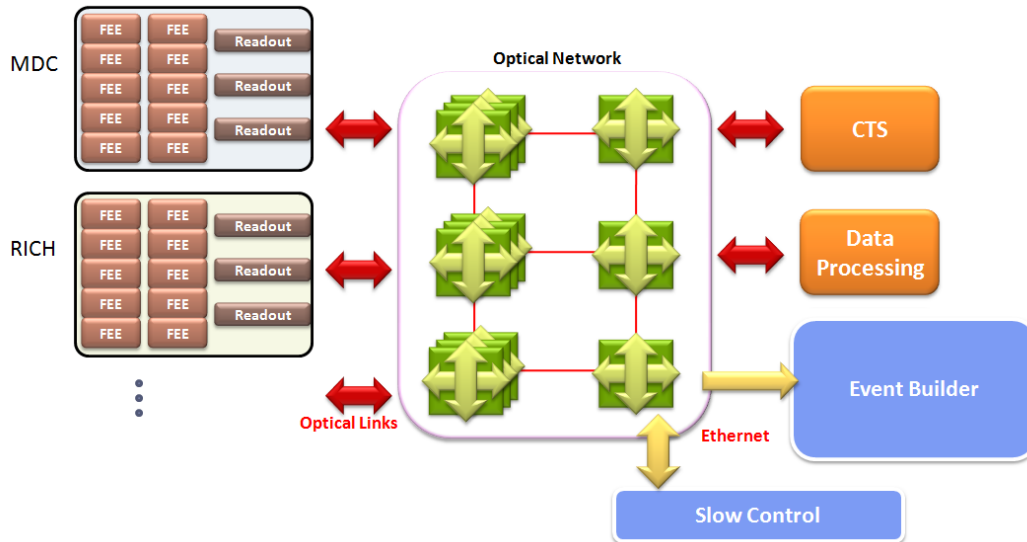


Figure 2.7: The entire scheme of the current TRBnet setup. The detector front-ends acquire data, which is transformed into TRBnet specific format and forwarded over the optical network hubs to the right hand-side for processing. Over Ethernet, the event data can be stored and the acquisition process controlled.

2.3 Read and Write Access

In order to communicate with the hardware, the TRBnet RegIO module can be used on any FPGA chip. It is able to interpret certain TRBnet datawords as *READ* and *WRITE* signals. Its purpose is, as the name implies, to allow register reads and writes (hence input and output). Every TRBnet node contains a unique address (the ID). Each ID is determined by the DS1820 temperature sensor on the chip and stored in a special register. This register is crucial for internal communication, however it is by far not the only register on the TRB and other boards. The RegIO entity can actually read any arbitrary register on the chip. Merely the register address needs to be provided. The entity moreover supports multiple reads and multiple writes to the same or consecutive network address(es). In case of a *WRITE* signal, the 32 bit data packet needs also to be given. In figure 2.8 the access to the monitoring system through the RegIO module is outlined. After decoding the media interface input, the RegIO initiates the *READ* or the *WRITE* and awaits some handshakes (unknown address, no more data, write acknowledge and data ready) together with the response data. If the read takes too long, a timeout signal is released and the operation canceled.

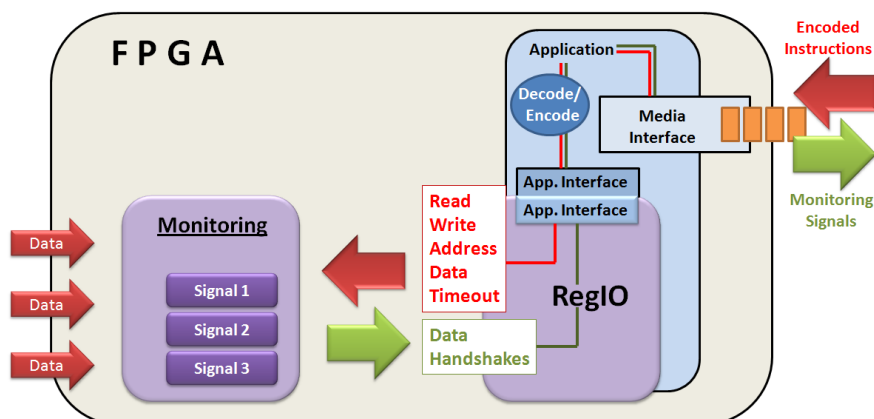


Figure 2.8: The monitoring facility can be read out exclusively through the RegIO module. The RegIO handles incoming read and write signals and forwards them straight to the monitoring system, as well as extract the monitoring signals.

2.4 Room for Monitoring

The purpose of the monitoring system is to monitor the state of the detector electronics and display it on the screen for easier evaluation and further analysis. Therefore, in order to obtain the most information, it should be embedded into the hardware as deep as possible. The detector front-ends are designed very differently. An implementation of the monitoring system for the front-ends would require different designs to suit each sub-detector. Since this approach is neither very flexible nor future-friendly, a more generic solution realized on the detector FPGA chips is favored. Since all detector data passes through the readout chips, the monitoring can be implemented there. Such a design is at the same time a new challenge, since the monitoring facility would have to be as generic and abstract as possible. A large variety of configurations is required to adapt the system to any arbitrary readout node and besides that, the FPGAs are already heavily loaded with algorithms leaving only little room for monitoring. Therefore an extensive analysis of every current readout system and all the sub-detectors will follow in the next chapter.

Other than that, TRBnet is still in development. However a large library of pre-designed entities for the FPGA chips has already been made available in VHDL. Applications can be loaded into TRBs, trigger simulated and some user commands executed via the Linux system on the ETRAX chip, however, the most interesting component for this thesis is the RegIO. The response can be easily encoded into the low-level media interface format and thus distributed throughout TRBnet directly. The RegIO is tailor made for acquiring monitoring signals from readout electronics and will be embedded into the solution in chapter 4.

3 Analysis and Design

This chapter is partitioned into two parts. The first part contains an in-depth sub-detector analysis and elaborates all the requirements on the new monitoring system. The second part introduces in section 3.6 an adaptive design fulfilling all of the presented constraints.

The HADES project currently applies eight distinct detector systems using approx. 80 000 data channels [Lor08, Pal08]. The first HADES experiment scheduled for the year 2010 is the gold on gold fixed target experiment. The experiment aims, among others, to demonstrate the effectiveness of the new readout system before the completion of the FAIR extension. The gold-ion beam can be maintained for several weeks, during which around 3 billion events can be collected. The vector meson decay rate into di-electrons is very sparse and, due to the background radiation, from all of these candidates, only maximum 100 000 can be used for in-medium hadron analysis. Therefore, in order to analyze such huge data loads appropriately, the trigger has to work with a frequency of 20–100 kHz minimum. After being upgraded, the TRBnet should be capable of forwarding the data stream at that speed [Mic08]. However, while waiting for trigger decisions with latencies of several events, the data needs to be buffered in the front ends and readout modules first. This is achieved through a trigger pipe system (see figure 3.1). Data is buffered in different pipes awaiting trigger decisions [Tra01]. Besides the meaningful events containing di-electrons, a minor fraction of minimum bias events needs to be included into the calculations as well, supplying comprehensive information about the detector.

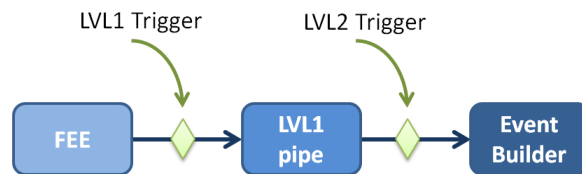


Figure 3.1: The most basic data flow scheme through the HADES pipe system. The level 3 trigger is currently in development.

The distribution of data has been briefly discussed in the previous chapter. In order to analyze deeper prerequisites on the monitoring system, a closer look on the detector electronics will be provided in this section. Actually, nearly all detectors follow the same scheme illustrated in figure 3.2. The front-ends are gathering raw analog signals in every HADES sector and forwarding it to the add-on boards and TRBs, which digitize the data and send it through TRBnet.

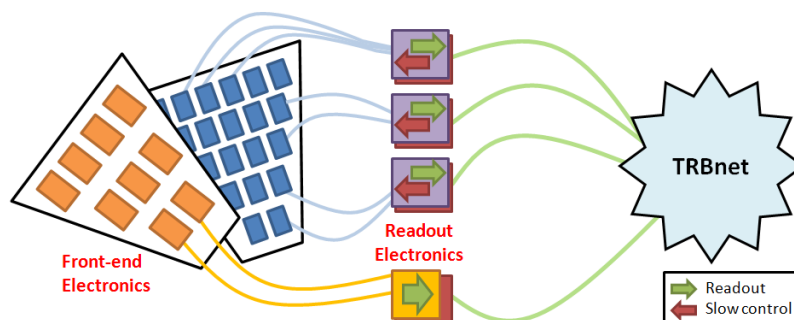


Figure 3.2: The FEE-to-TRBnet bridge. On the leftmost side are the detector planes depicted acquiring event data. The data is read out in the readout nodes consisting of TRBs with detector-specific add-on boards.

3.1 Basic Specifications

Since the monitoring system needs to fit in with the current HADES architecture, it has to be versatile and extensible, as the TRBnet itself. It should be designed to support all future experiments, regardless of which sub-detectors or which data they use. Therefore it needs to be developed directly on the FPGA chips of the readout electronics, FEE or TRBs. The FPGAs are already heavily loaded, so the monitoring system additionally needs to be very simplistic, keeping the resource consumption at minimum.

A high degree of customization is necessary in order to adapt itself to the many detector-systems and a large variety of configuration modes needs to be present. The monitoring facility additionally has to be data independent, in order to support all the future signals. Finally, it has to provide real-time access to the monitoring data and allow online configurability. Table B.1 summarizes these basic requirements.

Requirement	Description
Simplicity	Maximum efficiency at low resource consumption
Versatility	High degree of customization and data independence
Real-time access	Fast access and on-the-fly configuration
Extensibility	Support for future experiments

Table 3.1: A summary of basic requirements on the monitoring system.

3.2 Level 1 Trigger Electronics

The first level trigger plays the key role during data acquisition. It is used to downscale the event rate by a factor of 10 and reduce the data load significantly. However, it needs to be calculated extremely fast for every of the 1 000 000 events per second.

Its purpose is to check whether each collision is central enough to produce vector mesons. This can be done by simply counting the particle multiplicity in TOF and TOFino detectors. Central collisions, similar to those from figure 1.2, produce many scattering particles, hence by determining the charge in TOF and TOFino for each event and comparing it to a threshold value, the first level trigger can be calculated.

Prior to that, the start of an event needs to be determined. For this purpose the Start/Veto detector system has been designed. Basically, it registers a collision between the beam and the target and provides thereby the start-signal for every sub-detector.

3.2.1 The Start/Veto Detector

The Start and Veto detectors regulate the start of the data acquisition operation [Kri08]. They are positioned in the center of HADES and a beam is set up to move through both of them, according to figure 3.3. Both are identical and consist of polycrystalline CVD¹-diamonds, with dimensions of $25 \times 15 \times 0.1 \text{ mm}^3$ (see figure 3.4). They are designed with good radiation hardness and can be operated at room temperature [HAD09].

If a particle is registered in the Start detector, and if none in the Veto detector is present afterwards, it must have been involved in a collision with the target on its way. Therefore, a hit has occurred and the data-taking can begin. They operate at the efficiency of 98.5 % with a timing resolution of around 29.2 ps [HAD09]. Altogether, event rates of $10^7/s$ can be detected [GSIa]. In this special case, a single TRB is responsible for readout and slow control, without any add-ons.

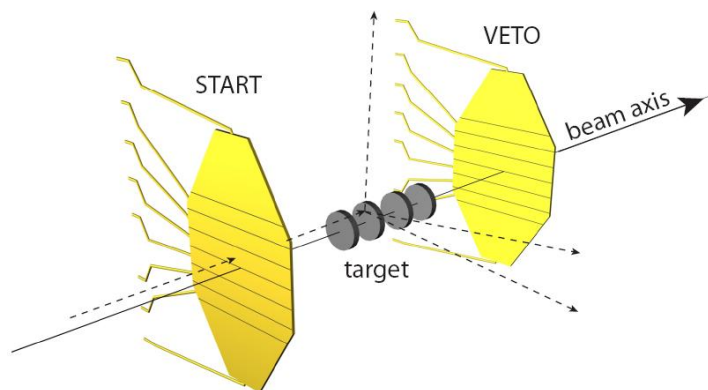


Figure 3.3: The identical Start and Veto detectors are placed in the middle of the HADES frame. If a particle disappears on its way, a collision with the target is most likely the reason, hence they determine the start of data acquisition. *Source:* [Lor08]



Figure 3.4: The Start/Veto detector in real life. Their height amounts to only 2.5. cm. *Source:* [Kri08]

¹CVD = Chemical Vapor Deposition

3.2.2 The TOF/TOFino System

After an event-start could be determined, the particle multiplicity needs to be checked. The best way to do this is to consult the **Time Of Flight** detectors, the TOF and TOFino. They are able to detect nearly every charged particle flying through. Moreover, they even contribute to some particle identification through energy losses in their plates. Since they are scintillators, they absorb particle energy and need to be positioned on the outermost shell of the HADES detector.

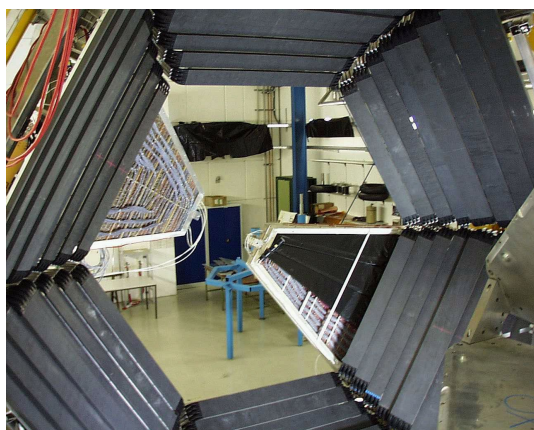


Figure 3.5: The Time Of Flight detector. It constitutes the outer most shell of HADES. Source: [Kri08]

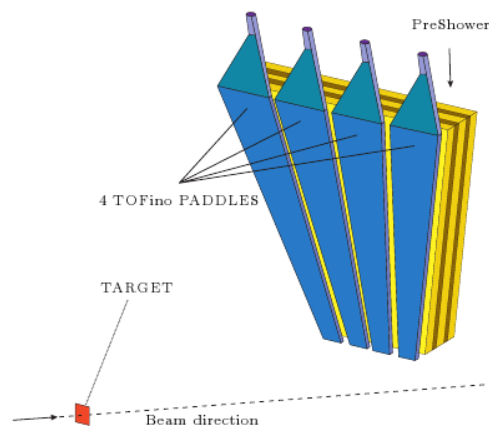


Figure 3.6: The trapezoidal TOFino paddles lean directly on the Pre-Shower. Source: [HAD09]

Technical Details. The TOF and the TOFino consist of BC408 plastic scintillator rods emitting light whenever charged particles fly through. This light induces secondary electrons in the Photo Multiplier Tubes (PMTs), which are then amplified and measured, providing good analog signals which reflect particle energy and allow timing measurements over the HPTDCs [Pal08, Ago02, HAD09]. The TOFino covers the smaller angles of 18 – 45 deg. and is supposed to register faster, more energetic particles usually emerging in lower angles of a fixed-target collision. Each of the six sectors contains four scintillator paddles providing moderate timing resolutions² of 400 ps. This value will be corrected in the future, as the TOFino is intended to be replaced by the Resistive Plate Chambers with a much finer timing resolution below 100 ps [Bel09]. The TOFino paddles are shown in figure 3.6 and are connected directly to the Pre-Shower. In this way, a quick lepton-hadron discrimination can be achieved. The TOF covers angles from 44 – 88 deg. and contains 384 scintillator rods in total. A picture of TOF is shown in figure 3.5. Each of the eight plates per sector contains eight scintillator rods, which can reach a timing resolution of less than 150 ps. The rods have variable size, increasing with the angle, from $20 \times 20 \text{ mm}^2$ diameter and 1 m length up to $30 \times 30 \text{ mm}^2$ and 2 m. This was necessary to keep the double-hit rate below 20% and minimize particle loss³.

²The TOFino alone can not be used for proper identification, but its data combined with the Pre-Shower detector achieves the desired results.

³This reference value, found in [Lan08], corresponds to $Au \rightarrow Au$ at 2 GeV.

The Readout. The TOF readout operation is performed using a TRB with the TOF add-on board (see figure 3.8). The add-on receives 128 TOF signals coming directly from the PMTs and reads them out with a time over threshold method [Pal08]. Additionally, NINO ASIC modules [Ang04] from CERN are used together with several amplifiers and discriminators to correct the signals and forward them to the TRB HPTDCs, where they are digitalized and packed into TRBnet format. The data can then be used for the CTS and for particle identification. The scheme from figure 3.7 sums up this procedure. The TOFino will be replaced and possesses an outdated readout mechanism.

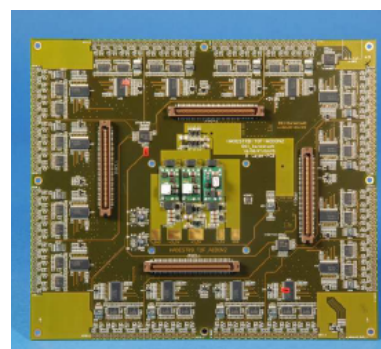
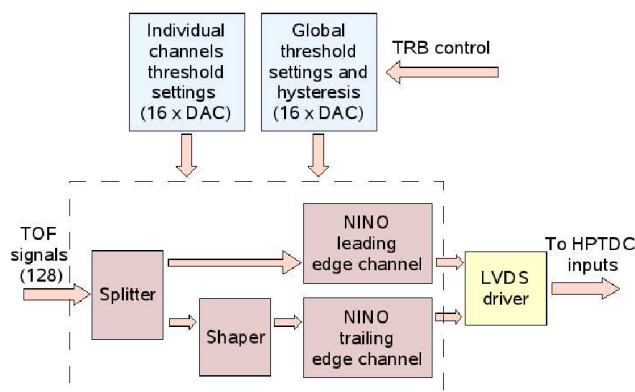


Figure 3.7: The TOF add-on operating scheme. TOF FEE provide the raw analog data on the left-hand side, which is forwarded to the TRB HPTDCs for digitization with time over threshold method. Source: [Pal08]

Figure 3.8: A photo of the TOF add-on board. Source: [Mic09]

3.2.3 Multiplicity Triggering

The main purpose of the CTS is to analyze TOF/TOFino trigger data [Fro08]. Simply all data from TOF and TOFino is gathered and compared to a certain threshold value. If the value is greater than the threshold, a trigger is released. As the decision needs to be generated very fast, a specific TRB with a trigger add-on board is analyzing the TOF/TOFino data online. The board is a part of the CTS and releases the trigger if the detectors show high particle activity. The TRBnet latency is very low (around 2-3 microseconds) [Mic08] and the board operates very fast, hence the trigger decisions can be created within few microseconds [Pal08].

3.3 Level 2 Trigger

If a large particle yield was present in the first step, a central hit has occurred. Therefore, additional sub-detectors distinguishing electron-positron pairs from other particles are necessary. The data awaits the confirmation in the level 1 pipe, but technically it can also be forwarded to the event builder and analyzed offline. This step will be committed in the future, the current

setup however needs to provide the second level trigger online. For this purpose, the CTS is once more needed, together with some additional data processing nodes.

3.3.1 The RICH Detector

The **R**ing **I**maging **C**herenkov detector [Zei99] is designed particularly for di-electron detection and is 'blind' for hadrons and other particles. It is a large optical device with many radiation-sensitive CsI photocathodes detecting Cherenkov radiation (see figure 3.9).

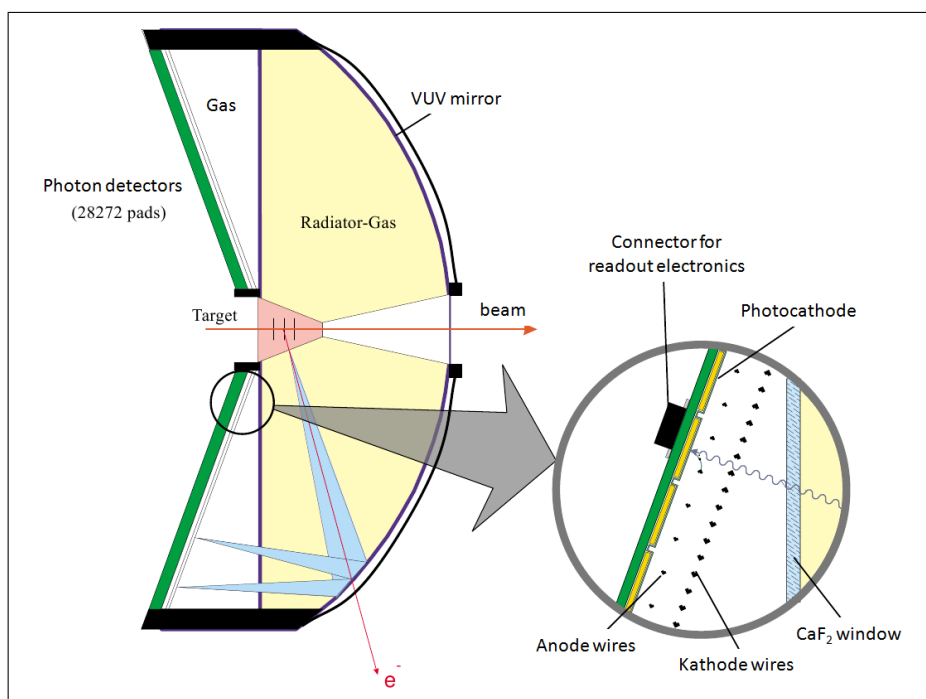


Figure 3.9: A cross-section through the RICH detector. The cone-shaped electron trail (light-blue) is reflected and focused into a circle on the photocathode-plane (green). *Source:* [Pac08b]

Technical Details. As it's name implies, the RICH uses the Cherenkov effect [Lan84] and is capable of detecting merely electrons and positrons [HAD09]. These very light particles are also the fastest in the HADES experiments. Therefore they leave a cone-shaped electromagnetic trail in the RICH gas-chamber, whereas all other, slower particles do not. The gas chamber is filled with C_4F_{10} gas that alters the speed of light inside it below the speed of di-electrons passing through. Therefore, the Cherenkov radiation can be emitted, which is reflected by the ultra-violet mirror. The mirror has 1.45 m radius and is designed in a special way, so that it reflects the cone-shaped trail back into a region of constant radius producing a circle contour. The reflection thus forms a circle due to the spherical mirror shape, which can be detected using CsI photocathode pads (see figure 3.9). Hence, whenever circle-like shapes are present, with very high probability they are caused by electrons or positrons. Additionally, at HADES experiments

all circles are of same size allowing perfect instrument calibration to support high resolutions. To achieve good results, 4712 pads per sector need to be read out, producing 28 272 data channels in total.

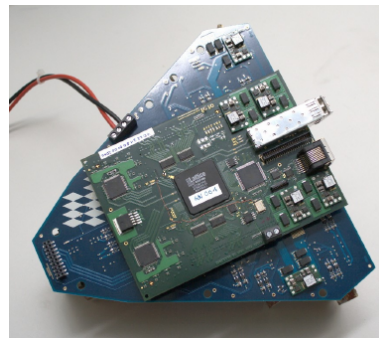
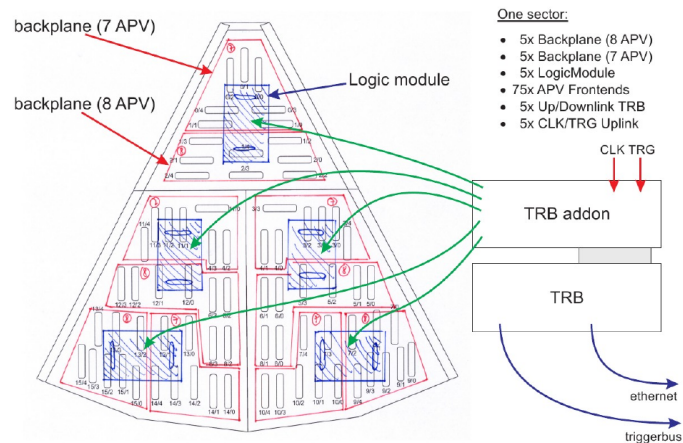


Figure 3.10: The RICH data acquisition layout. Recently, some changes have been made. The TRB add-on is a standard hub, which forwards the data over its optical links to TRBnet. Source: [Tra07]

Figure 3.11: A photo of an ADCM attached to a backplane. Source: [Mic09]

Readout. The front-ends are concerned with the acquisition and correction of photocathode signals [Boh99, Boh00]. The detector implements several distinct modules specifically designed to support the unusual geometry and the large number of channels. The adjusted readout electronics is connected to the FEE over five different backplanes [Pal08] (see figure 3.10). The RICH modules are then read out using customized ADC modules (ADCMs) attached to the backplanes.

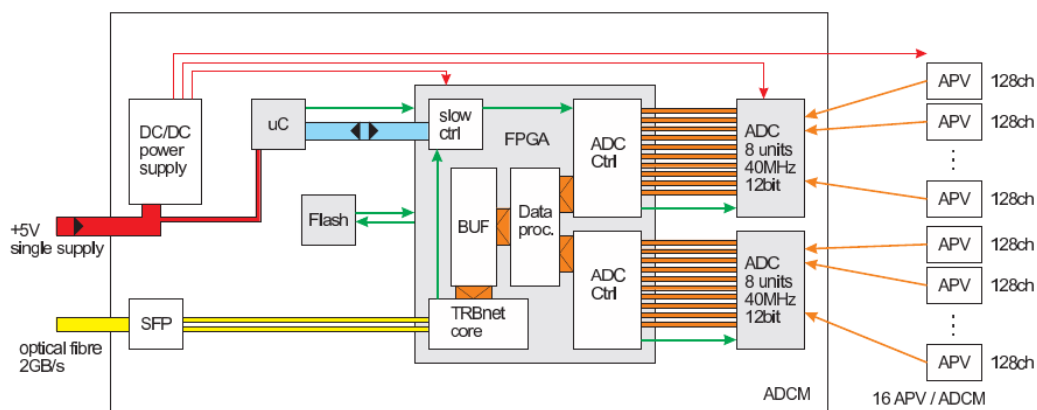


Figure 3.12: The RICH ADCM architecture. Source: [Pal08]

There are five ADCMs per sector, each responsible for 960 FEE channels making RICH the second most complex detector at HADES. The ADCM on its own is a complex system containing a Lattice FPGA, a RISC microcontroller, ADCs and an optical link (see figure 3.11). It serves at the same time as a bridge between the FEE and the TRBnet, implementing the TRBnet protocol and forwarding the data to hubs. The complete ADCM architecture is shown in figure 3.12.

3.3.2 The Pre-Shower Detector

The Pre-Shower sub-detector system [Bal04, HAD09] is used for electron identification in lower angles between 18 and 45 deg. At such low angles TOFINO alone does not suffice since particle velocities are nearly the same for leptons and hadrons. Therefore, another method of discrimination using the Pre-Shower is necessary.

Technical Details. The Pre-Shower contains three chambers filled with a mixture of argon and isobutane gas [Kri08]. The chambers contain wire-planes which can detect electromagnetic showers. The showers come from 1 cm wide lead-plates between the chambers. The first chamber is not exposed to any showers and detects merely the charge from the particle flying through. When an electron reaches the first lead plate, it starts emitting strong radiation in form of bremsstrahlung. Hadrons, on the other hand, do not. The radiation creates new electrons in forward direction, reaching the second lead plate, where the effect is intensified and recorded in the last chamber. Hence, if an electron moved through the Pre-Shower, an increasing current from the first chamber to the last can be registered, whereas hadrons produce constant current in all chambers. The effect is illustrated in figure 3.13.

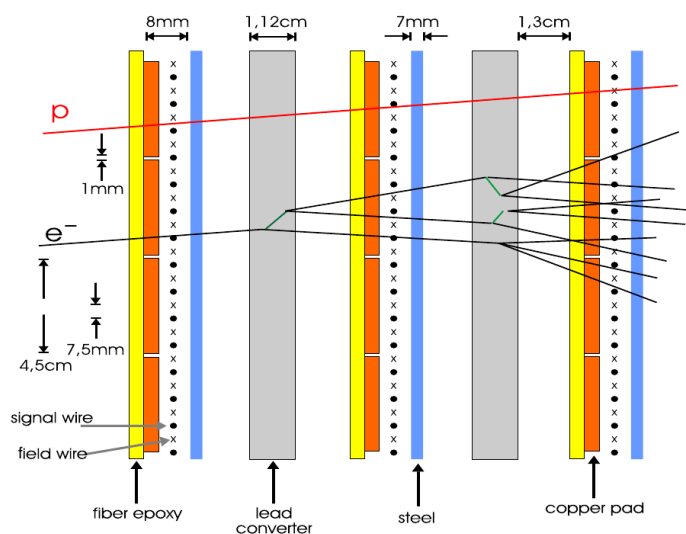


Figure 3.13: The Pre-Shower is able to enhance the charge generated by electrons and positrons flying through. They emit bremsstrahlung in the lead converters, producing further di-electron pairs. The electric charge in the chambers therefore gradually increases, whereas for hadrons it remains constant. *Source:* [Tra01]

Readout. The information on chamber status is forwarded straight to the variable gain amplifiers (VGAs) of the Pre-Shower add-on board [Pal08] (see figure 3.15). The add-on board contains 24 VGAs with four channels each, 12 ADCs (AD9212) and a Lattice LFE2-70E-5F900C FPGA⁴. The 10 bit ADCs operate on eight channels each and the FPGA is used for control. The add-on resides on a TRB, which provides the clock signal and some additional VGA controls over three DACs⁵. If the first level trigger arrives, the data from the Pre-Shower is amplified with VGAs, converted over the ADCs, moved to the TRB and then forwarded to the corresponding level 1 pipe, waiting again for the second trigger. To reduce the data load, only Pre-Shower values above a certain threshold are being stored. Besides the data-taking mode, the add-on board can additionally operate in calibration or maintenance mode. The entire procedure is illustrated in 3.14.

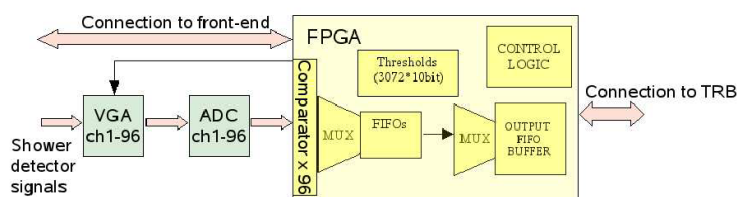


Figure 3.14: The Pre-Shower digitization process. The VGAs are controlled over the FPGA. *Source:* [Pal08]

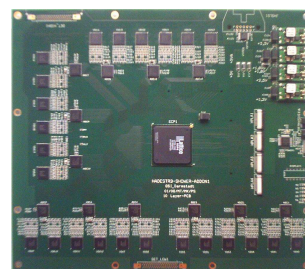


Figure 3.15: The Pre-Shower add-on board. *Source:* [Mic09]

3.3.3 Di-Electron Pattern Analysis

The most challenging data acquisition step is the determination of electron presence in the RICH, TOF and Pre-Shower detectors. This process currently needs to be performed online during measurements, and fast. The solution is to forward the event data through the network to a designated TRB-cluster operating as the Image Processing Unit (IPU) [Tra00]. There, large

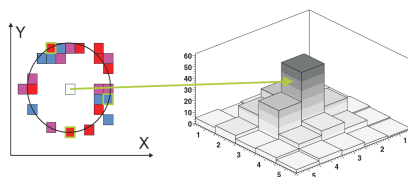


Figure 3.16: The RICH image processing algorithm uses Hough transformation to calculate the center. *Source:* [Lan08]

⁴A new version of the board is being designed, containing three ECP2M50 FPGAs and an optical link.

⁵DAC = Digital-to-Analog Converter.

buffers can be filled and image patterns developed. An example of a RICH calculation is shown in figure 3.16.

After that, the pattern data is forwarded to the Matching Unit (MU), which analyzes data from all detectors together and decides, for example, if the circle from figure 3.16 correlates with the TOF and Pre-Shower data, i.e. if the electron presence could be detected in all three detectors. Only if this is the case, the level 2 trigger signal is released, allowing the data to reach the event-builder. The entire triggering procedure is summarized in figure 3.18. More information on triggers can be found in [Tra01, Tra00, Pal08, Fro08].

In future, the level 2 trigger analysis will also be performed offline. For this occasion, FPGA boards based on ATCA crates [Min08] are being developed, which can simply replace the online level 2 trigger logic (IPU, MU, etc.) by connecting directly to the optical link network and storing the level 2 data for offline analysis. A prototype can be seen in figure 3.17.

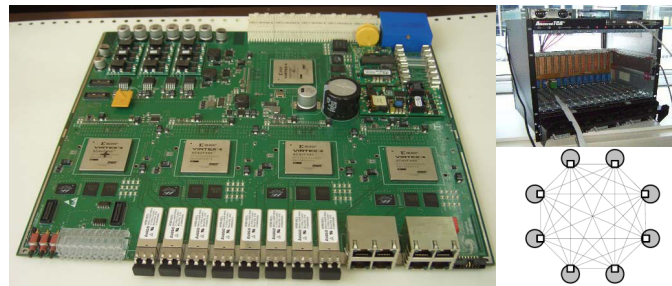


Figure 3.17: These boards will replace the level 2 trigger in the future. Ten of them will be connected over the ATCA crate with HADES. Source: [Min08]

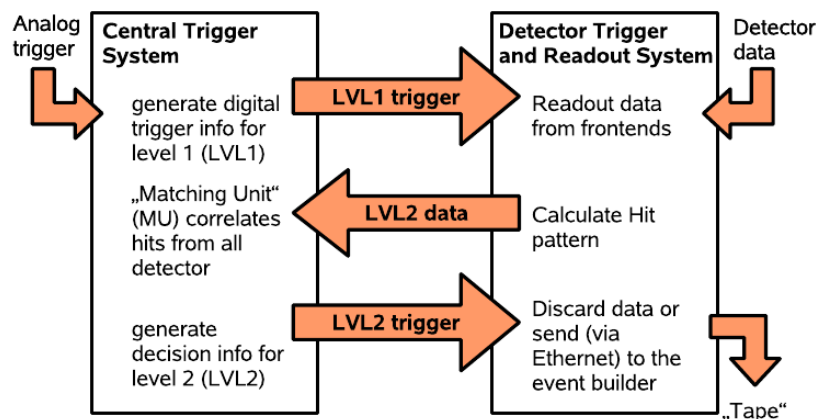


Figure 3.18: The HADES triggering procedure starts with the quick analysis of analog data from TOF. After that, the Pre-Shower and RICH are involved to generate the second level trigger. Source: [Fro08]

3.4 Particle Identification

All of the aforementioned systems are mainly used to generate trigger decisions on when an event should be stored, but one major detector is responsible for the actual identification of the numerous collision products - the Multiwired Drift Chambers (MDCs) [Mun04, HAD09]. Together with the data from other detectors (especially TOF), particle analysis can be performed correctly.

3.4.1 The MDC Detector

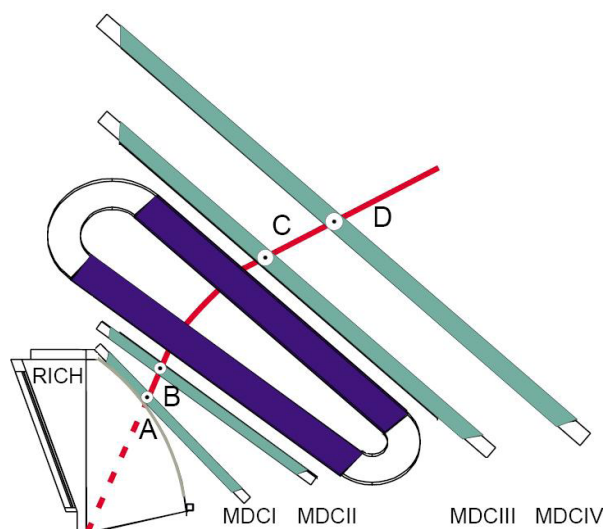


Figure 3.19: The HADES tracking system is using MDC planes and a magnet (max. 0.9 T) for momentum reconstruction. In order to operate at maximum precision, a calibration needs to be performed properly. Source: [Lan08]

The basic purpose of the MDC detector is the reconstruction of particle tracks. The MDC planes are positioned twice in front of a magnet, and twice behind it covering angles of 18 – 85 deg [HAD09]. The magnet is an essential element of any spectrometer. While a charged particle moves through the magnet, it gets diverted. The curvature of the track depends on how large its momentum is. Therefore, by detecting the position of the particle before the magnet, and then again after it, the degree of diversion can be determined and the particle-momentum calculated. An illustration of this process is outlined in figure 3.19. Based on their momenta, particles can be identified [GS08, BRR08].

Technical Details. There are four MDC planes with increasing size in every sector (24 MDC planes in total) and as the name implies, each one is made out of six layers of driftchambers each containing several layers of wires (see figure 3.20). Moreover, the space around the wires is filled with a certain gas mixture. If a charged particle flies through, it ionizes the gas molecules, creating an electric pulse for 1 ns in the wiring. By analyzing all six layers, the position of the

particle can be specified with a precision of 60 – 100 micrometer in azimuthal and 120 – 200 micrometer in polar direction. The wire layers are arranged in a 20 deg. displacement in such a way that they enhance the position detection in azimuthal area (i.e. for particles moving in polar direction).

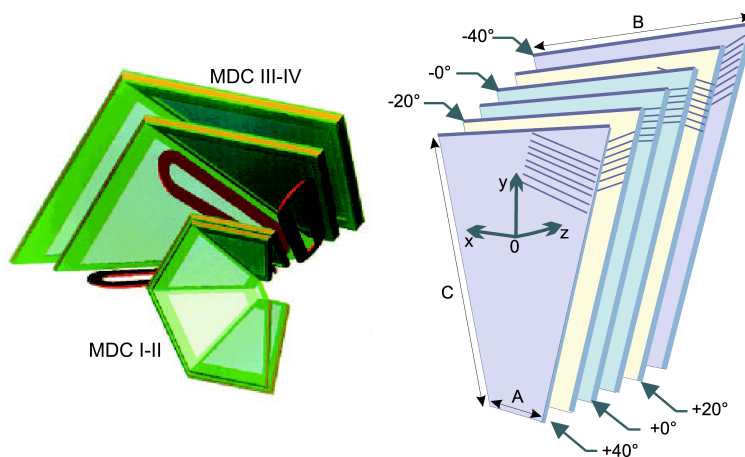


Figure 3.20: The MDC consists of 24 planes (left picture), each with six differently aligned wire-layers (right). *Source:* [Lan08]

Readout. With a contribution of over 27 000 channels, the MDC is the most demanding sub-detector in the HADES system. For appropriate readout, three different boards have been designed [Pal08, Tar08]:

- Daughter-Board
- Motherboard
- Optical End Point Board (OEPB)

They are all mounted directly on the back side of MDC planes, as shown in figure 3.21. The daughter-boards are responsible for analog signals only. They acquire the basic states of the driftwires, amplifying and discriminating the signals at the same time. The motherboards then digitize the data using TDCs. Several motherboards are connected to an OEPB, which acts as a driver card. The OEPB contains a Lattice FPGA (ECP2/M20) which is used for readout and slow control. Once mounted, the MDC FEE boards can not be accessed anymore without demounting the entire detector. It is therefore very important to keep the system stable as long as possible. This is the reason why two flash-memories are additionally located on the OEPB. While one runs a basic firmware allowing the FPGA to return to its basic configuration whenever needed, the other contains upgradeable firmware versions supplying the OEPB with newer logic.

The design of the OEPB has been a great challenge, as no space was left in the detector. Therefore, all of the electronics had to be realized on a tiny board of $4 \times 5 \text{ cm}^2$ size [Tar08]. The OEPB uses plastic optical fibers (POFs) for data transmission with 250 Mbit/s over the

Firecomm Fiber Optical Transceiver (FDL300T) out of the MDC planes [Pal08]. According to [Pal08], this measure was necessary in order to reduce the electromagnetic noise, the cabling size and also the costs. There will be around 400 such boards in the field for all 24 MDCs in total.

On the TRBnet receiver side, a TRB with an MDC add-on awaits the data. The add-on possesses 32 POF interfaces, 2 regular optical links and 3 FPGAs, capable of reading out two entire MDC chambers at the same time [Pal08]. This simplified readout scheme is presented in figure 3.21, and the entire readout electronics in figure 3.22.

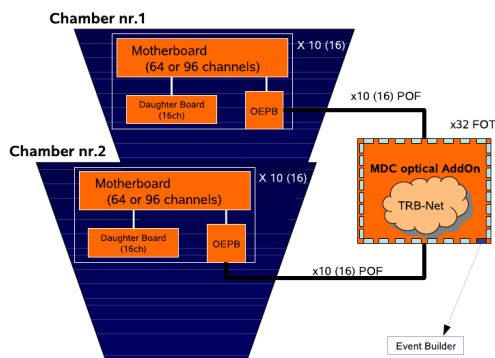


Figure 3.21: The MDC readout process involves three specific boards mounted directly on the detector. *Source:* [Tar08]

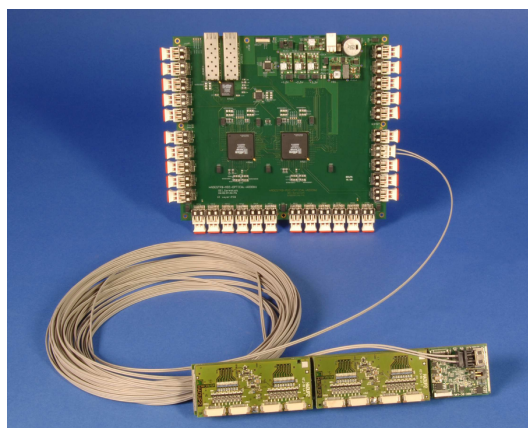


Figure 3.22: The entire MDC readout electronics. *Source:* [Tar08]

3.5 Analysis Results

Altogether, HADES utilizes following detectors in the current setup:

- **Start/Veto** – start of data-taking.
Technology: Beam measurement with CVD diamonds.
Readout: TRB.
- **RICH** – level 2 trigger, electron momenta.
Technology: Spherical UV mirror, gas chambers and CsI photocathodes.
Readout: ADCM, RICH add-on, TRB.
- **Pre-Shower** – level 1 trigger, electron detection.
Technology: Charge detection from bremsstrahlung in the scintillator-chambers.
Readout: Shower add-on, TRB.
- **TOF/TOFino** – level 1 trigger, time measurements.
Technology: Plastic scintillators, photo multiplier tubes.
Readout: TOF add-on, TRB.

- **MDC** – particle tracking and momenta.
Technology: Ionization of gas-molecules inducing electric current in the wire-layers.
Readout: Daughter-board, motherboard, OEPB, MDC add-on, TRB.
- **RPC** – TOFino replacement.
Technology: in development.
Readout: directly through the TRB.

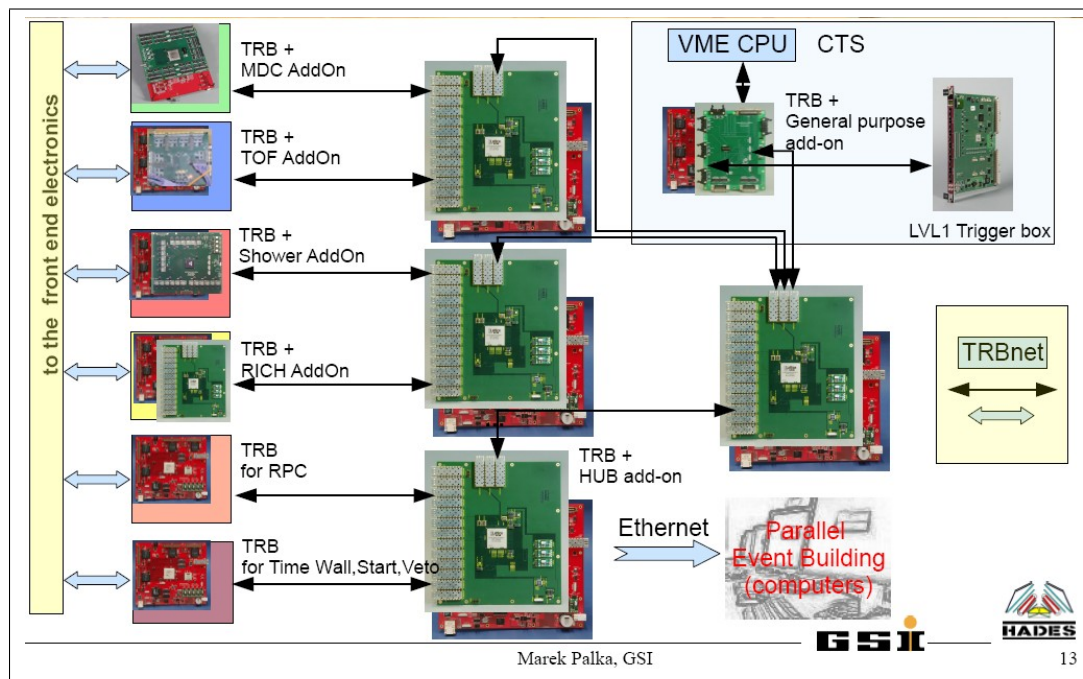


Figure 3.23: An exemplary TRBnet setup showing all the applied electronics. The CTS image in the top right corner is outdated and will be replaced by a TRB with a triggering add-on. *Source:* [Pal08]

Figure 3.23 shows the entire readout electronics in practice. The monitoring needs to be embedded deep into the hardware, FEE near. Since FEEs use analog signals only, to design a monitoring system for every sub-detector would go beyond the scope of this work and would be inadequate for future experiments. However there are many FPGAs present, supporting dynamic logic circuits. Some are available in the readout electronics and some are even part of the FEE. Therefore, the best suited place for monitoring are the FPGAs and the design will follow an approach to create a versatile system, ready for use on arbitrary TRBnet FPGA chips. Currently, the most important monitoring signals have been summarized in table 3.2.

3.6 Monitoring Design

In order to perform all-encompassing device monitoring, three questions regarding the monitoring signals need to be answered: what, where and when? First of all, the signal **type** needs to be

Signal	Usage
Temperature	Every detector, TRB or add-on
Voltage	MDC FEE
Channel busy time patterns	Hubs, determines the network flow and busy activity
States of finite state machines	Any readout-node, including FEE
Current settings (thresholds, etc.)	Any detector logic (MDC, Pre-Shower, etc.)
Statistics on which channel fires	MDC
Calibration status	Every detector requiring calibration
Average data statistics	Every detector, MDC especially
Number of token not received	MDC

Table 3.2: Currently, the most important monitoring signals are collected in this table. The list is not complete, as the detector is still in development.

determined by the monitoring system. For this occasion, a large variety of possible signal specifications needs to exist, distinguishing every signal coming from a different detector. Next, the signal **source** needs to be determined. It is very important to know where specifically the signal emerged from. The answer to this question is provided by the TRBnet itself, as every signal is always transmitted using the unique id of the network node which sends it. Finally, all data packets carrying monitoring signals must additionally contain a **timestamp**. Nearly 500 readout boards will be used in the experiments for the few detectors, hence the boards operate relatively independent from each other. Chronological ordering of monitoring signals needs therefore to be supported in order to avoid the imminent signal chaos (see figure 3.24).

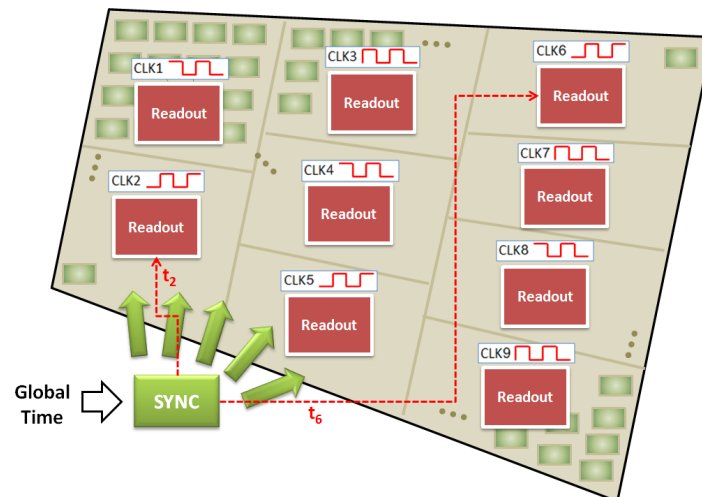


Figure 3.24: The TRBnet contains loads of FPGAs that can not be synchronized below a certain timer precision. In the example, it is evident that the signals t_2 and t_6 have different latencies, hence the global timer provides only a rough timing signal. Monitoring needs to support more accurate timers.

The monitoring system will be implemented for FPGA electronics directly in the hardware, embedded as deep in the FEE direction as possible. Therefore, two parts need to be developed – the first part makes signal monitoring on arbitrary FPGA hardware available, and then the second part gathers the data using TRBnet and transmits it to the software client outside the optical network (over Ethernet) for visualization.

It also needs to fit in with the effective readout mechanisms on the chips and to support the numerous signals (especially ones coming in the future). To achieve this goal, the monitoring system needs to be blind to any kind of signals coming from the detectors. The hardware part should simply provide an interface and treat all incoming signals as raw data, as illustrated in figure 3.25. The user needs to specify what kind of signal on which port is being used and visualize it in the software manually, later on. The hardware logic, however, does not include this signal information, e.g. if some busy times are stored in 12bit format, as well as states of a finite state machine, only monitoring of 12bit signals would be performed and the signals could not be distinguished by the hardware part. This step is necessary in order to support monitoring of arbitrary signals and to reduce the data load (since the signal definition does not have to be included in the package). The signals are only distinguished by their internal address.

Signal visualization can be easily performed using the open-source EPICS software. This step is explained in section 4.4 extensively.

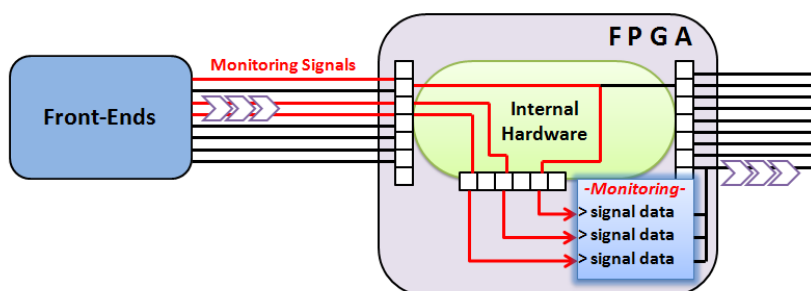


Figure 3.25: All monitoring signals need to be buffered as raw data. They are acquired over the interface to the internal hardware.

3.6.1 Signal Properties

As a first step, the monitoring system user needs to specify all signal properties in advance, in order to setup the system according to the current situation. The system needs to know what kind of resources it should allocate and how to treat the signals. Therefore, one main specification is the signal size, i.e. the **effective port width** of the interface between the monitoring system and the rest of the FPGA hardware.

Occasionally, statistics on certain detector parts will have to be acquired. In many cases, this step needs to be performed very fast, with (nearly) every clock cycle. The TRBnet readout mechanisms are not designed to send read-commands in every cycle. Therefore, in order to achieve the highest possible frequency and reduce the load on the slow-control channel, the

monitoring data needs to be buffered. For this occasion, customizable First-In First-Out queues (FIFOs)⁶ have been designed, containing the second main specification - the **FIFO depth** (i.e. the amount of data packets that can be maximally stored inside).

Now that FIFOs are needed, there is a huge number of different configurations which can be used. The FIFO approach, although necessary, actually opens up a huge variety of possibilities. First of all, the FIFOs can be filled only with a certain, predefined **frequency**. Further, all the data packs gathered in the same FIFO must contain a timestamp. Since one single detector applies many different FPGA chips, the timers need to be synchronized. However, this can not be accomplished with high precision. In figure 3.24, the global *SYNC* signal has different latencies for distinct readout nodes, therefore it can not be used for high-precision monitoring. For that reason, three different timers are present as summarized in table 3.3. Only the global timer is responsible for FPGA synchronization, while the other two work independently from one another. To achieve some appropriate chronological ordering with the out-of-sync timers, the **event-number** can be ascertained directly from the TRBnet. Every event distributed over the TRBnet contains a consecutive number (event ID). A part of the ID can be placed in the FIFO data packet, roughly labeling the signal with its trigger-source. In this way, a fine timer can distinguish high-frequency signals within a FIFO, while the event-number provides the global relationship on all independent FPGAs of the same detector system. Therefore, in order to specify the timer domain of a monitoring signal, a **timer-type** needs to be declared as well, which is one of the three timers from table 3.3.

Timer	Description
Global timer	The rough system time, approximately equal on all boards.
System timer	Generates the high precision timings, runs on all boards independently.
Trigger timer	The time since the last trigger on a particular board.

Table 3.3: Different timer types can be used to set the appropriate timestamp accuracy.

All the timers simply count the clock-cycles and provide in this way rough or fine signal timings on all monitoring chips. But the timing specification is not sufficient yet. With the above method, signals lying on different FPGAs can be chronologically combined together. But different signals on one particular board may require different timing granularity. They could contain the same timer type, but because of their varying frequencies, they might need a differing **timer resolution**. This property simply denotes from which bit on, the incoming timer signal should be used as a timestamp (see figure 3.26). While some signals count every cycle, others might increment their timer only after 32 cycles, for example.

Finally, the **sizes** of the raw data, timestamp and event-number need to be specified in order to extract the correct data from all of the datapack-bits later on. Figure 3.26 displays the data packet splitting and an exemplary timestamp generation inside the FIFO.

⁶A FIFO stores data inside a certain data structure (usually an array or a list). The first data packet that has been inserted will be read out first, independent on how many data packets have been written afterwards.

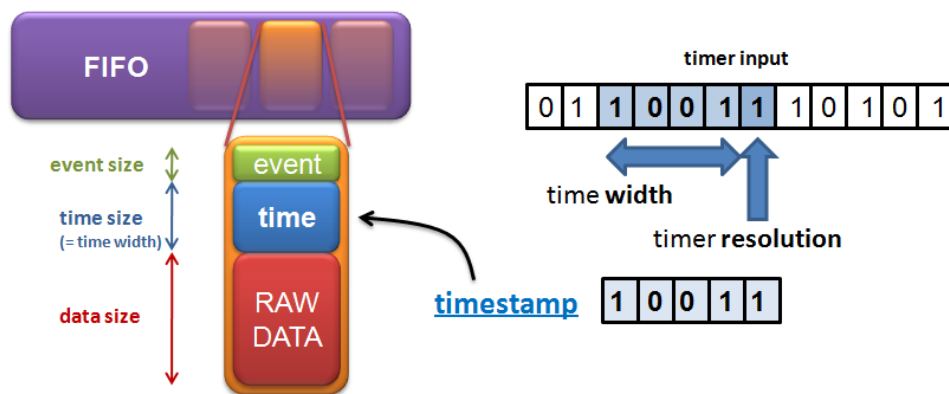


Figure 3.26: The FIFO data generation – raw data is encapsulated with the timestamp and/or an event ID inside the FIFO. The timestamp additionally supports different timer resolutions.

3.6.2 Orthogonal Design

Monitoring needs to be performed for every sub-detector, i.e. for many absolutely disparate systems, each with its own monitoring configuration. Some detectors need to monitor fewer signals, while others reach easily the space limitations on their FPGA chips. Hence if a monitoring specification of a particular sub-systems needs to be changed, it should under no circumstances have impact on the other systems. Monitoring on all chips needs to be performed independently from each another (since the timestamp and the event ID assure correct chronological ordering). This calls for an orthogonal design, where a change in one system does not affect the others.

In order to achieve this goal, the monitoring system needs to be customizable on every FPGA chip. Moreover, it needs to store its current configuration locally on the same chip. Every single signal definition together with its sizes, timer and frequency should be stored inside a ROM⁷ module, as presented in figure 3.27, enabling dynamic change-tracking. When the FPGA is being programmed, it contains a certain monitoring setup. The ROM needs to be generated automatically containing all the monitoring information from that particular setup.

The monitoring system should depend only on the information in the ROMs. Before the monitoring starts, the system has to gather all the ROM information and acquire the monitoring setups for all chips. Later, in the last step, the user can decide which signals in which way need to be visualized. Following such approach, there is no need for a central database, storing all monitoring information. The specifications are kept locally on every chip, instead.

3.6.3 Signal Readout

The signals are buffered in the FIFOs on FPGA chips, thus they need to be extracted over the TRBnet. The user should be able to read out every signal individually. Furthermore, an automatic readout instruction should be implemented to start the signal acquisition in preset time intervals. The client PC is located outside the TRBnet and the only way to communicate is

⁷ROM = Read Only Memory

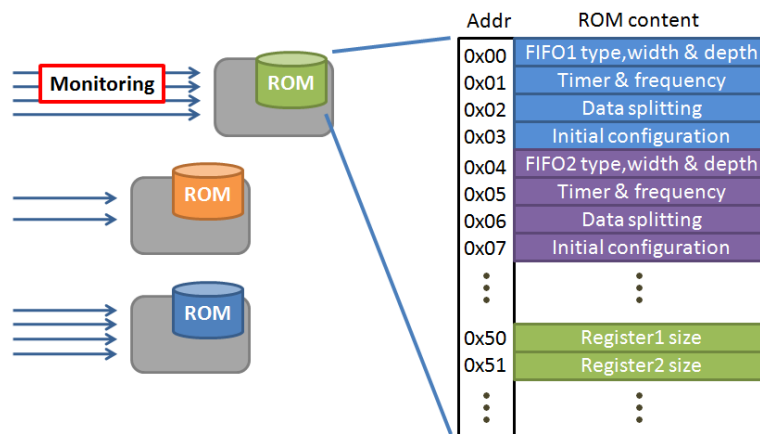


Figure 3.27: A ROM contains all monitoring configurations of a single FPGA chip. According to section 4.2.1, it is divided into FIFOs and registers.

through the Ethernet interface. Therefore, a TCP/IP server in a Linux shell needs to listen on one TRB in the network for incoming client calls. The server must decode the client instruction and connect to the FPGA to release the read or write signal through the entire TRBnet (on the least significant channel). This can be accomplished over the ETRAX interface located on every TRB. The server must use the two 16 bit lines from the ETRAX CPU to the FPGA and encode the request, or decode the response and send it back to the client.

3.6.4 Conclusion

To summarize the results of this chapter, the monitoring facility needs to be split into a simplistic hardware and a comprehensive software part. The idea is to apply highly configurable FIFOs to store the raw monitoring signals alongside their timestamps. On the least significant TRBnet channel, the FIFO contents can be gathered to a particular network node (residing anywhere inside the TRBnet) and then using a TCP/IP server transmitted to the client(s) outside the optical network. Figure 3.28 describes the entire monitoring procedure. The design needs strongly to support orthogonality, hence a ROM on each FPGA is read out first to obtain the current signal information and reconstruct the entire monitoring setup. Afterwards, the applied data can be visualized individually by any client.

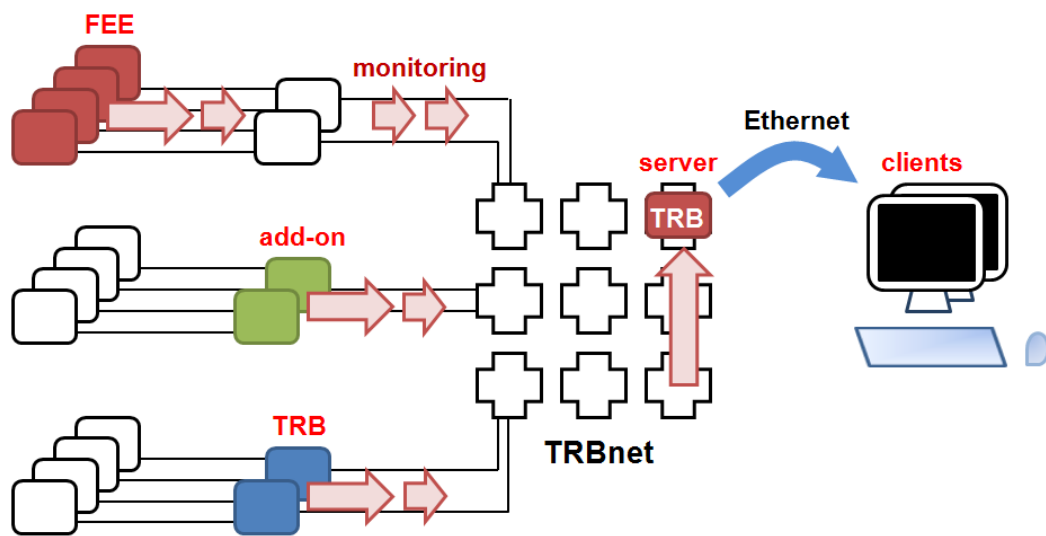


Figure 3.28: The Monitoring scheme inside the TRBnet. The monitoring is performed on the FEE, add-on boards or TRBs. Their data can be read out by a server and forwarded over the Ethernet to the clients.

4 Implementation

Under the consideration of all requirements disclosed in the recent chapter, firstly an implementation of the low-level hardware part will be presented, carried out in the VHDL programming language. Second, the high-level software counterpart is going to be elaborated, realized in C. Lastly, a practical use of the EPICS system for signal visualization will be introduced.

4.1 Monitoring System Layout

The monitoring should be not only performed for vital detector signals, but also used for generation of statistics, especially in terms of detector analysis. Busy times, channel loads, latencies and states of the system need to be examined properly. Since the applied technology is gradually evolving, all the experiments could be accompanied by many surprises. The monitoring system needs to support the constantly changing experimental setups by letting the detector developers themselves decide which parts need to be monitored. Such adaptive design needs firstly to supply an interface for the applied monitoring signals, and buffer them in highly customizable storage cells afterwards. The cells residing on front-ends or readout modules can be addressed and read out individually or automatically. One TRB in the network needs to implement a TCP/IP server. The server is responding to commands coming from outside of TRBnet, delivering the necessary monitoring data and controlling the entire procedure at the same time. On the client side, the user controls the monitoring process either over command line in a Linux shell or by using the EPICS control system accommodated to the monitoring facility, sending commands to the server and visualizing the results. Altogether, following parts have been implemented:

1. **Hardware part** - Signal storage system locally embedded on FEE, add-ons and some TRBs.
2. **Server side** - Software for controlling the monitoring procedure, running on one designated TRB with a fixed IP address.
3. **Client side** - Software executing commands specified by the user.
4. **GUI**¹ - EPICS control center using an embedded monitoring client.

4.2 Hardware Monitoring Section

Following the approach from figures 3.25, 3.26 and 3.28, corresponding VHDL entities have been implemented, enabling a synthesis on any FPGA chip. The implementation runs under

¹GUI = Graphical User Interface.

the name Real-Time **DE**vice **MON**itoring Framework (DEMON). The monitoring facility is intended to be used on many different detector FPGA chips during beam time at GSI and FAIR. The chips will be acquiring the corresponding monitoring signals in their internal hardware². They need to forward all the signals packed together over one or two huge ports to the monitoring system (see section 4.2.2 – ”The Input Interface”). After that, the monitoring can be accomplished automatically. The signals are first buffered and then read out over the TRBnet slow control channel.

FIFOs and Registers

The principal cell for storing the monitoring signals is the **FIFO** cell, easily found directly in the IP-cores³ of FPGA manufacturers. The main attention of this thesis lies in the variety of different FIFO configurations, required to hold any arbitrary detector signal, and the wide range of possibilities on how to read them out. The signal should not be further analyzed, but just buffered as raw-data together with its timestamp. Due to the many operational modes of a FIFO cell, a FIFO controller is needed for data flow supervision. FIFOs also possess configuration cells, storing the current mode and providing real-time controls. For further details, see section 4.2.3.

Not only FIFOs will be used as storage cells, but simple **registers** as well. The monitoring facility needs to be very resource friendly, hence some non-time-critical signals should be stored in plain D-flipflops serving as storage registers [Ok108b, Tin00]. Therefore, the monitoring system needs to support basic register access. The registers are extremely useful, as they allow a direct insight on the present state of the detector hardware. An illustration of the storage process is shown in figure 4.1.

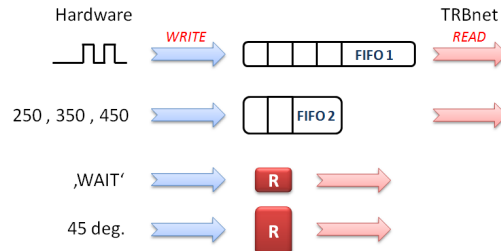


Figure 4.1: The basic monitoring principle. Data is acquired over an interface on the left-hand side and stored inside FIFOs or registers. The cells can be read out and configured over TRBnet.

²The term *internal hardware* is used to denote all the FPGA-internal logic devices other than the monitoring system.

³IP = Intellectual Property, predefined libraries enabling fast integration of basic components on FPGAs and other programmable devices.

Adaptability

Before DEMON can be integrated into the HADES system, the detector developer(s) must define all signal properties first, involving their size, timestamp generation mechanism and storage space they require. These specifications are kept central in one VHDL-File, as described in the following section. The file contains generics and constants configuring the entire hardware-related monitoring structure. After the configuration is carried out, the monitoring utility can be synthesized automatically. It appears that every sub-detector system will follow a different approach, hence there will be approximately N different DEMON setups, where N is the number of sub-detectors, and each setup will be run on several equal chips. For example, monitoring of the MDC detector can be performed directly on the front-end electronics and since there are approx. 400 such chips, they will all use the same DEMON configuration. Moreover, it is even possible to configure each chip individually with a different monitoring setup which is, however, due to the large number of chips not recommendable.

4.2.1 DEMON Configuration File

Since the monitoring system needs to be highly versatile, the user must specify how many signals, which FIFO types, timings, etc. are going to be used in the current experiment. Therefore the `demon_config.vhd` file is necessary. It is ready-made for VHDL-synthesis, requiring only slight adaptations to suit any monitoring setup (see Appendix A). Following configurations can be done:

- FIFO number (0–20)
- Register number (0–32)
- FIFO bus width (0–64)
- Register bus width (0–64)
- Size of the configuration cells (0–32)

The current version supports up to 20 FIFOs and 32 registers. The FIFO and register bus widths must be wide enough to carry the widest FIFO or register signal, respectively. Once set, the internal monitoring bus can not be changed anymore. The width should therefore be set wide enough to carry the maximal possible FIFO and register. For example, if a monitoring system requires FIFOs of widths: 16, 20 and 28 bit, than the FIFO bus width needs to be 28 bit wide. The same applies for registers. A detailed illustration is shown in figure 4.2. It is possible to use 64 bit for data transmission, although 32 bit is TRBnet standard. Therefore if using more than 32 bit, the monitoring data requires two read signals and the software part needs to handle these signals with care.

Additionally, each FIFO cell has to be customized separately. Following FIFO properties must therefore also be declared (an explanation can be found further below):

- Width
- Depth
- Logarithm of depth (automatic)

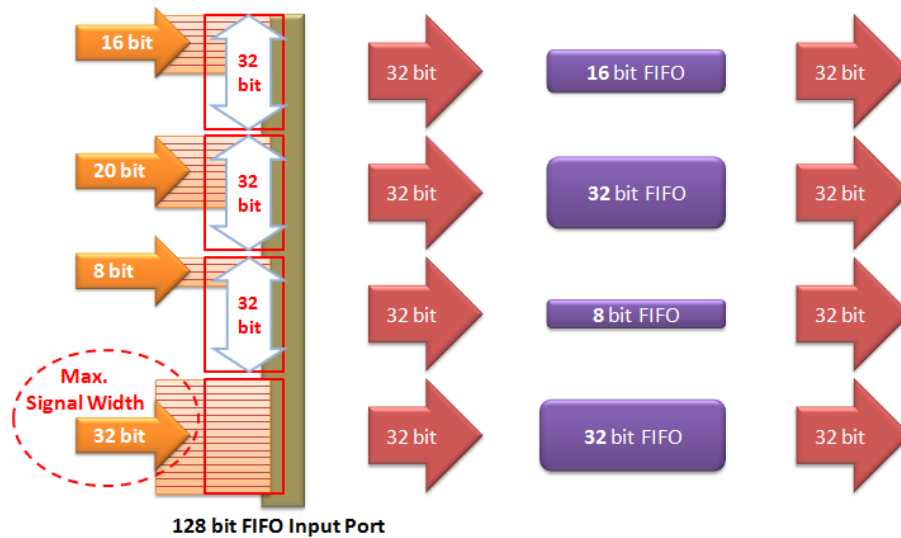


Figure 4.2: A description of the FIFO input port. All incoming signals are extended to 32 bit size due to the largest input signal. Therefore, the input port is 128 bit wide and the internal bus width is set to 32 bit. The FIFOs currently only support widths: 8,16,32 and 64.

- Control bits
- Monitoring type
- Frequency
- Timer type
- Timer resolution
- Timer size
- Data size
- Event size

The monitoring system strongly differentiates between the FIFOs and the much simpler register cells, as the latter consume much less resources. Therefore additional properties of every register need to be specified:

- Width
- Control bits

Lastly, the initial contents of a **configuration cell** can be defined. Every FIFO comes with a configuration cell and all configuration cells are of same size. They are used for online configurability of the monitoring system and are the only cells that can be written to. Every bit inside the configuration cell corresponds to a FIFO operation or an operating mode. The bits are internally connected to particular FIFO logic. Currently only four bits are connected. They allow following operations: reset, ringbuffer mode, input validation and halt/unhalt FIFO writes. At the moment, they seem sufficient to perform any high-level user command, but additional bits can also be used in future for more sophisticated instructions.

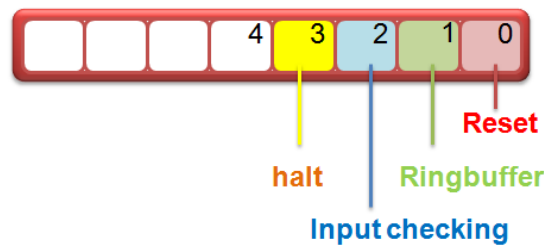


Figure 4.3: Predefined operations of the configuration-cell. These cells control the FIFO behavior online.

Explanations

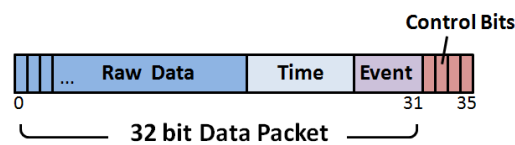


Figure 4.4: Each FIFO can have different data partitions. The control bits can be allocated and do not add to the data width (see section 4.2.2).

- **FIFO width** denotes the entire width of the data packet stored inside the FIFO. The entire data packet is composed of: raw data, timestamp and an event-number. As can be seen in figure 4.4, data size, timer size and event size partition the data packet according to the following equation:

$$data_size + timer_size + event_size = FIFO_width.$$

- **The FIFO depth** specifies how many data packets a FIFO should contain. Following values have been implemented so far: 16,32,512,1024,2048 and 4096. The logarithm of depth is needed for some additional functionality and will be calculated automatically.
- **The frequency** specifies with which speed the FIFO can be written to. Different signals need usually to be acquired with different frequencies. The FPGA runs with 100 MHz and the highest possible write resolution is therefore 10 ns (denoting every clock-cycle for $frequency = 0$). The actual frequency is $2^{frequency}$, so when specifying $frequency = 15$, the FIFO would be acquiring data every 328 microseconds. For a complete frequency table, see appendix B.
- **Timer type** and **resolution** have already been explained in section 3.6 and can be best understood with figure 3.26 and table 3.3. They define the timer domain and the time counting speed, respectively.

- **Control bits** allow optional controlling features, that the user needs manually to implement and are described in section 4.2.2.
- **The monitoring type** is one of the predefined settings described in section 4.2.3.

The ROM

In order to distinguish the raw input signals from each other and due to the discussion in section 3.6.2, signal specifications need to be stored locally on every FPGA chip. For this purpose, a ROM module is used which contains every signal being monitored alongside its properties (like width, depth, frequency, etc.). Its contents are generated automatically from the "demon_config.vhd" file and always follow the same scheme. Addresses 0–79 contain FIFO signals (4 addresses per signal, see figure 3.27), and the rest contains up to 32 register specifications.

The first step of the monitoring operation is to read out the entire ROM on every FPGA and acquire the specifications for every monitoring node. In this way each client PC located outside the TRBnet can create a map of addresses and signals that need to be handled.

4.2.2 The Input Interface

Monitoring signals are provided over an interface between the monitoring system and the remaining FPGA devices. The internal hardware needs to provide the signals, since they strongly vary between sub-detectors. The signals are subdivided into FIFO- and register-signals, to separate the two types from start on. Therefore, only two (but large) input ports are carrying all the

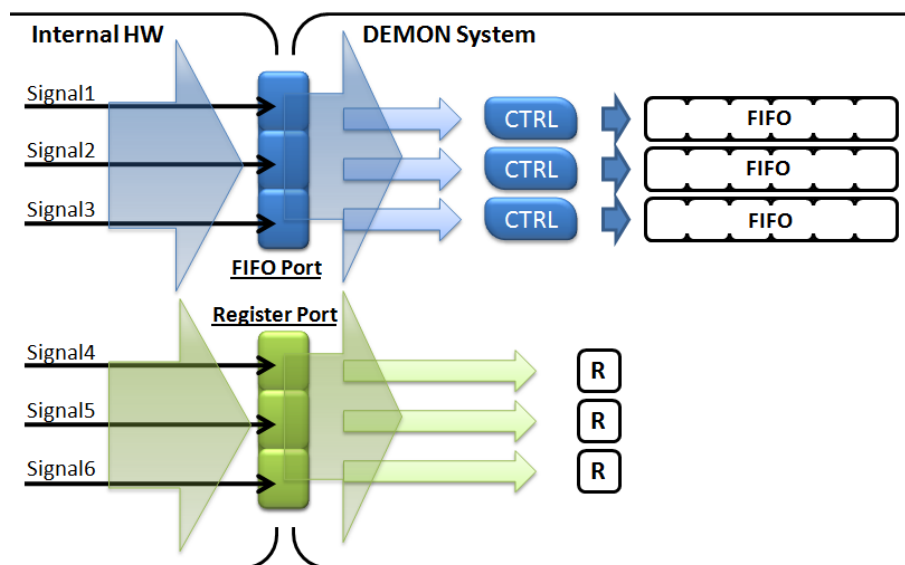


Figure 4.5: The monitoring signals are provided over the FIFO and the register port. The internal hardware needs to combine all inputs together into one huge port, which is decomposed by the DEMON system according to the preset bus width.

signals. The FIFO-port can be used for time-sensitive monitoring tasks (like statistics, behavior, analysis and similar), while the register-port for direct access to the hardware states (temperature, voltage, etc.). Both ports are composed of many individual input signals which partition the port in equally occupied segments (figures 4.2 and 4.5 show some examples). The width of each port-segment needs to be equal (see figure 4.2). The smaller input signals are inflated with zeros to meet the equality condition. Although it might seem wasteful, in this way the inner structure can be realized much easier, consuming less logic cells and enabling very simple multiplexers⁴ later on. The equal bus width for all FIFO and register signals inside the monitoring system is a premise.

All FIFO and register cells, which are numbered consecutively, simply gain access to the corresponding segment of their input port (see figure 4.5). For example, if the FIFO bus width is 32 bit and if there are 8 FIFOs in the monitoring setup, then the FIFO input port would be of 256 bit size and the first FIFO would gain access to the first 32 bits (i.e. 0–31), the second FIFO to bits 32–63, and so on. If one FIFO uses less than 32 bits, only the least significant bits will be used for monitoring.

Details

The reason for equal port splitting lies in the implementation technique. The design calls for a generic synthesis. At implementation time, the entities do not know how many FIFOs and input signals are there going to be used. Moreover the number of storage cells strongly varies, but one single source code needs to be able to generate arbitrary many cells. However, there is no completely efficient method to solve such versatile design in VHDL.

The cells need to be read out using one RegIO module afterwards and therefore multiplexers are required (as the RegIO can handle only one address at a time). In case of the FIFO cells, one single multiplexer is used to distribute the read signals to the FIFOs and multiplex the responses. If the signals that need to be multiplexed vary in size, the multiplexer needs additionally to store the signal widths in some internal registers in order to handle the signal properly.

A much simpler solution is to use a universal bus width, in order to generate arbitrary many cells from one single source code. By reading the configuration file, the system can ascertain how many cells it needs to create and, more important, how to connect them. This can be implemented by a simple *generate*⁵ statement. The multiplexer does not need any additional information, as it simply divides the combined input port into equal segments (the bus width). Based on the address received from the RegIO, it selects the signal. The generate statement simply uses the number of FIFOs and the FIFO bus width from the configuration file to create and connect all cells to the FIFO input port and to the FIFO multiplexer. The register cells are generated in the same way. The source code and additional information is given in appendix C.

⁴Multiplexers are digital components responsible for signal switching. One single output signal is connected to one of the multiple input signals. The selected input signal is adjusted according to an additional control port. They hence enable basic signal selection. [Tin00]

⁵In VHDL, the generate statement can be used to apply dynamic hardware generation. In this case, the statement determines how many FIFOs and registers are defined in the configuration file and uses a FOR-loop to generate each of them with their own properties. In this way, the number of cells may vary, but it does not have any impact at all on the source code.

Control Bits

One additional aspect greatly contributes to the extensibility of the monitoring system. Besides the two input ports, additional control bits can be set by the internal hardware. They can be used to accomplish various tasks, however manual implementation is required. They are completely optional and can be used to pass some controlling options coming from the internal hardware directly to the monitoring facility. These controls can perform arbitrary algorithms which the user needs yet to implement (and which depend on the situation). For example, if the internal hardware reaches a certain state, the monitoring might need to stop or some values have to be discarded. This behavior can thus be passed on immediately to the monitoring system over the control port.

Moreover, the currently used monitoring signal can be marked with a small bit pattern. The control port can be used in this case to label the data packets in their storage cells. For this reason, the FIFO- and register-cells can contain 'control bits'. The bits from the control input port can regulate a mechanism (like a finite state machine, for instance) which sets the control bits pattern on top of every data packet according to the current situation. The marking is stored inside the FIFO or register. Moreover, a readout mechanism can be easily implemented which distinguishes monitoring signals based on their markings. An internal data processing mechanism can be accomplished in this way, like for example when a high temperature is present and it is known that the hardware tends to malfunction at these temperatures, the control bits can mark the inconsistent packets or discard them instantly to achieve better statistics. The possibilities are open to any developer to implement their own algorithms, as needed. Figure 4.6 displays this process. The control bits do not interfere with the actual data, however their number is limited. Table 4.1 in the following section displays the limitations for every FIFO type, whereas register control bits are not limited.

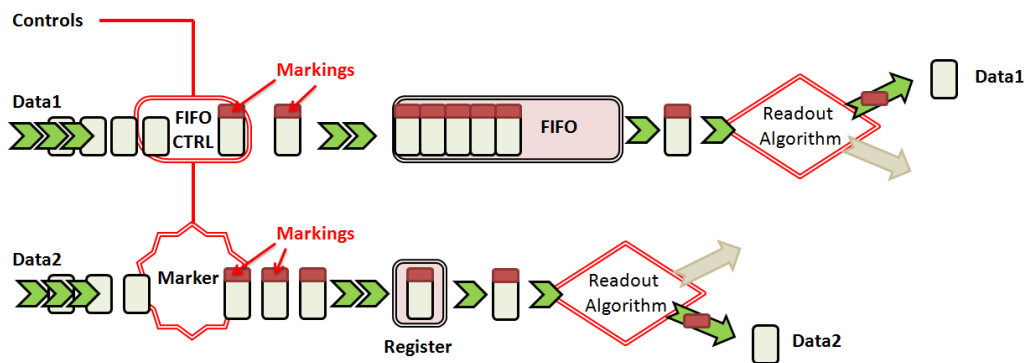


Figure 4.6: The control port can be used to implement additional algorithms. The upper part shows a FIFO, while the lower describes the marking procedure inside a register. The control bits can be set automatically by the hardware to perform arbitrary monitoring operations.

4.2.3 The FIFO cell

The entire FIFO cell is a complex system, containing a FIFO controller, the actual FIFO and lots of logic.

The FIFO Controller

The FIFO controller is responsible for input filtering. It receives the monitoring signal from the corresponding interface segment and has three important functions. First one is the **frequency regulation**. The controller uses an internal counter to determine when the preset frequency has been reached. When the counter is equal to $2^{\text{frequency}}$, a *WRITE* signal is released to the FIFO (whereas the input data is passed all along). In this way, the FIFO can write signal data only at certain frequency. The controller can furthermore perform **input-validation**, storing only differing signals. If a signal did not change since the last *WRITE*, it will not be written again. This behavior can be switched on and off using the corresponding configuration bit (see figure 4.3). One more interesting feature of the monitoring facility is the internal marking of data packets, as explained in section 4.2.2 – ‘Control Bits’.

Figure 4.7 shows a simplified FIFO controller structure. It can be divided in two parts, the top and the bottom part. The logic in the lower part is responsible for correct frequency handling. The upper part can be avoided if the *VALIDATE* signal coming from a certain configuration bit is set to *low*. Otherwise, if it is set to *high*, the current input data is compared to the previous data stored in the register and blocked as long it does not differ. While data is blocked, the *WRITE* signal is also disabled (not depicted here).

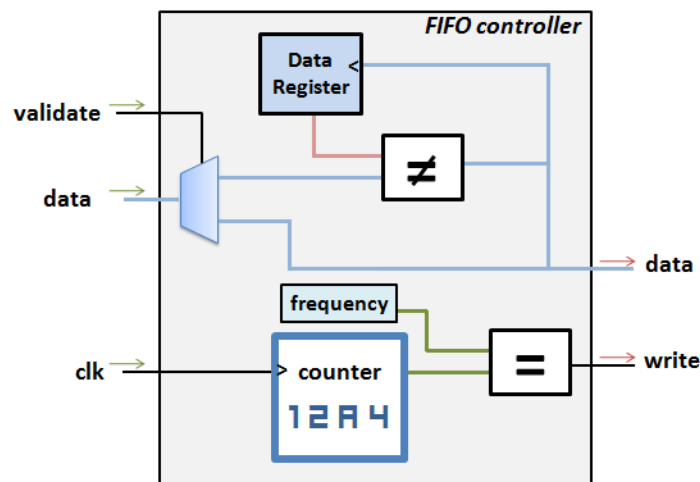


Figure 4.7: Basic FIFO controller logic. The upper part can be switched off using configuration cells connected over the *VALIDATE* signal.

The FIFO cells

For every FIFO module, a bit-matrix has to be allocated inside the FPGA hardware. Since the FIFO widths and depths strongly vary, the resource consumption can become quite large. In order to suit every experiment in near future, a large collection of possible FIFO types has been pre-designed to minimize the resource consumption and adapt itself to the current situation. Usually, a proper FPGA chip contains many Block RAMs (BRAMs) and has its logic implemented in Look-up Tables (LUTs). Therefore two types of FIFOs can be implemented. If no logic cells are left, a large FIFO can be instantiated using one or several Block RAMs. However, if no Block RAMs are free, the smaller LUT-FIFOs can be synthesized, consuming some remaining logic-cells (LUTs). Therefore, all FIFOs are either implemented on Block RAMs or Look-up Tables and synthesized on the Xilinx and Lattice FPGA chips of the FEE or readout boards. The LUT-FIFOs do not support control bits in the current version. Table 4.1 summarizes all implemented FIFOs.

Type	Size	Resource Consumption	Control Bits
Block RAM	16 × 1024	1 Block RAM	2
Block RAM	16 × 2048	2 Block RAMs	2
Block RAM	16 × 4096	4 Block RAMs	2
Block RAM	32 × 512	1 Block RAM	4
Block RAM	32 × 1024	2 Block RAMs	4
Block RAM	32 × 2048	4 Block RAMs	4
Block RAM	64 × 512	2 Block RAMs	8
Block RAM	64 × 1024	4 Block RAMs	8
Look-up Table	8 × 16	8 Look-up Tables	–
Look-up Table	8 × 32	16 Look-up Tables	–
Look-up Table	16 × 16	16 Look-up Tables	–
Look-up Table	16 × 32	32 Look-up Tables	–
Look-up Table	32 × 16	32 Look-up Tables	–
Look-up Table	32 × 32	64 Look-up Tables	–
Look-up Table	64 × 16	64 Look-up Tables	–
Look-up Table	64 × 32	128 Look-up Tables	–

Table 4.1: All of the implemented FIFO types are listed here. The control bits come free with every Block RAM (they are the FIFO parity bits) and can be used for additional information. The resource consumption gives merely a reference value (on a Xilinx chip), since additional resources for logic and wiring are required.

FIFO Modes

Since the FIFO acquires data packets with constant frequency, it tends to become full before its results can be read out. When full, the FIFO throws away incoming data packets. This behavior may lead to unwanted side-effects, therefore another FIFO mode has been implemented – **the**

ringbuffer mode. When a FIFO reaches its nearly-full boundary, it discards the oldest data packet, i.e. a 'fake read' is performed as presented in figure 4.8. In this way, the FIFO contents are always up-to-date. It can also happen that new data is written with higher frequency than it is read out. In this case the FIFO first needs to be halted and then read out to prevent data inconsistency.

The ringbuffer mode, once turned on can also be switched off again using the appropriate configuration bit. Every FIFO can be instantiated in the ringbuffer mode, however it uses slightly more resources than the plain FIFO in the standard mode. Hence, if it is known that the ringbuffer mode is not necessary, an alternative, simpler FIFO can be built with lower resource consumption. This implies that every FIFO type from table 4.1 possesses two modes exactly – the **standard** and the **ringbuffer mode**. Where the ringbuffer mode can mimic the standard mode, it does not work vice versa.

The data count limit for the nearly-full behavior of the ringbuffer is dynamic and depends on the FIFO size. It is calculated according to the simple equation:

$$fifo_depth - \log_2(fifo_depth) = limit$$

Tests have shown that it is safe to set the boundary to a static value in future (see figure 4.8). When the data count reaches the limit, a *READ* signal is triggered and the *t1* flag denotes it. In the next clock cycle, the FIFO is offering one data packet. After a *t1* signal, the *t2* flag is always triggered in the following cycle, denoting that the FIFO data is fake and needs to be discarded. Hence the offered data packet is never going out of the storage cell if the *t2* flag is high. If the limit has been reached and a real *READ* signal is pending, then the *t1* flag will not be released. Altogether, the *t1* flag marks a fake *READ* and the *t2* flag discards the data packet. In this way

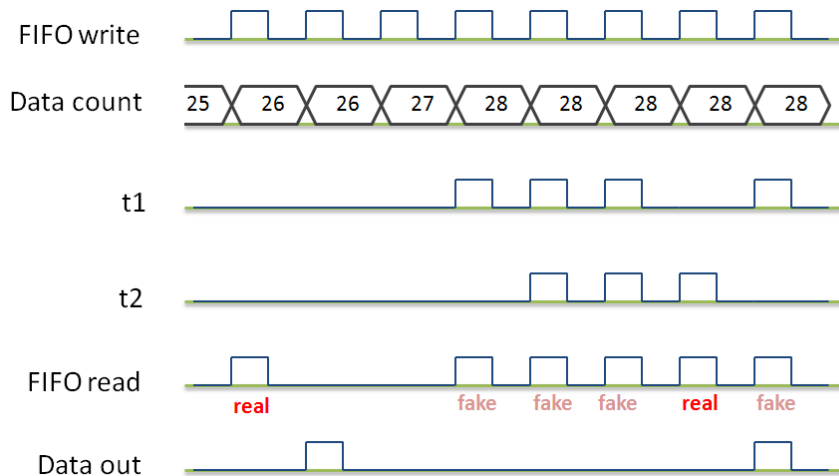


Figure 4.8: The ringbuffer function of a FIFO cell is depicted here. In this example, the FIFO depth is only 32 and hence the limit is 27 data packets. The FIFO write signal and the clock cycle are the same. When the limit is passed, the cell starts initiating fake reads, thus the data gets discarded to make room for the newer signals. The fake reads are set internally, however if a real read comes from the RegIO, the *t1* and *t2* flags are not set and the packet is read out.

a ringbuffer functionality can be easily realized in any FIFO. As can be seen in figure 4.8, even when *WRITE* signals are coming in every clock cycle, the ringbuffer never tends to get full. The flags can also be released in every cycle and even distinguish real and fake *READ* signals with no problems.

4.2.4 Address Space

Now that data can be stored inside the cells, the monitoring system needs to access each of the cells individually through the slow control channel. As part of TRBnet, each board contains a unique address. Therewith each FPGA chip in the network can be determined and every single monitoring setup accessed. Inside the facility, however, in order to distinguish the cells among each other, the RegIO module is needed. As explained in section 2.3, it has been designed as part of TRBnet in a related work [Mic08] and governs the entire address range within the FPGAs. It was initially designed to directly read from or write to certain registers, but since *READ* and *WRITE* signals and the response can be sent in the media interface specific format, it will be also used for monitoring control. Its basic functionality has been already discussed in figure 2.8.

Every storage cell containing monitoring signals automatically undergoes an internal numbering within the chip. The cell rank is added to the current address space to obtain the cell address, e.g. the address of FIFO 1 is the first address in the FIFO address range, thus $0x2000$ and the 4-th register operates under the address $0x3003$. The addressing remains the same on every chip and can be derived directly from the ROM. Table 4.2 shows all currently used address ranges.

Component	Address Range
ROM module	$0x1000 - 0x106F$
Configuration cells	$0x1800 - 0x1813$
FIFO cells	$0x2000 - 0x2013$
Register cells	$0x3000 - 0x301F$

Table 4.2: The DEMON implementation supports only the listed addresses. It has been limited to 20 FIFOs and 32 registers, and the ROM contains 112 entries. If higher addresses are not in use, more cells can be realized in the future. An inappropriate address on the slow control channel results in an 'unknown address' response and terminates the request.

Attached to the RegIO, another TRBnet entity is present – the **RegIO bus handler**. It has been designed to partition the entire address scope of the RegIO into smaller parts. In case of the monitoring system, four segments are being used according to table 4.2. The cell access is realized in two steps. First, the RegIO bus handler determines the correct segment for the access, based on the most significant bits of the cell address. Afterwards, a multiplexer selects the corresponding cell using the least significant bits and transmits the read/write request. Only the ROM does not possess a multiplexer and can be read out immediately. Therefore, during a request the incoming address is resolved twice, except for the ROM. The multiplexer also handles the handshakes with the RegIO and is responsible for sound readout of large datawords (larger than 32 bit). More details can be found in section 4.3.2 – "Controlability".

4.2.5 DEMON Summary

The monitoring signals arrive over two input ports. They are stored either in FIFO- or register-cells. All FIFOs contain a controller which regulates the writes, and a configuration cell storing the current state and mode of operation. Another port for additional, optional and automatic hardware controls can also be used, but needs to be implemented manually.

The actual acquisition of monitoring signals is performed over the TRBnet slow control channel. The requests consisting of a cell address and the control signals are resolved in the RegIO modules and forwarded to the multiplexers. After that, the cell can be read out and data is forwarded through TRBnet to the monitoring server.

The entire monitoring scheme is shown in figure 4.9. The signals are provided on the left-hand side. The first monitoring step is to read out all the ROMs and determine the DEMON configuration on every chip.

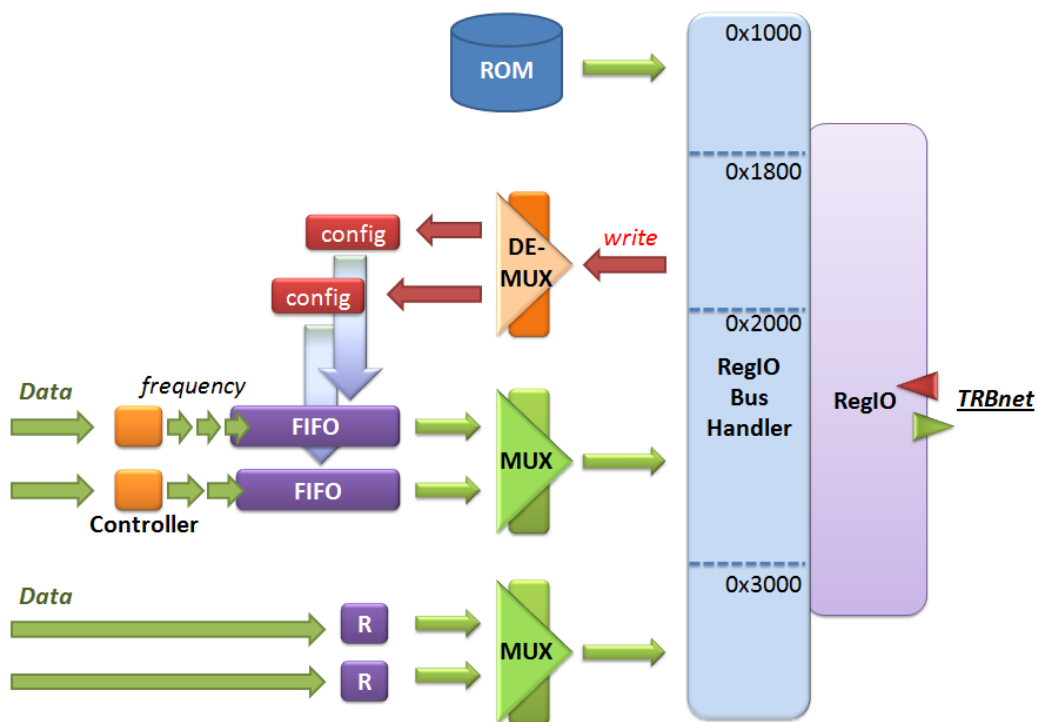


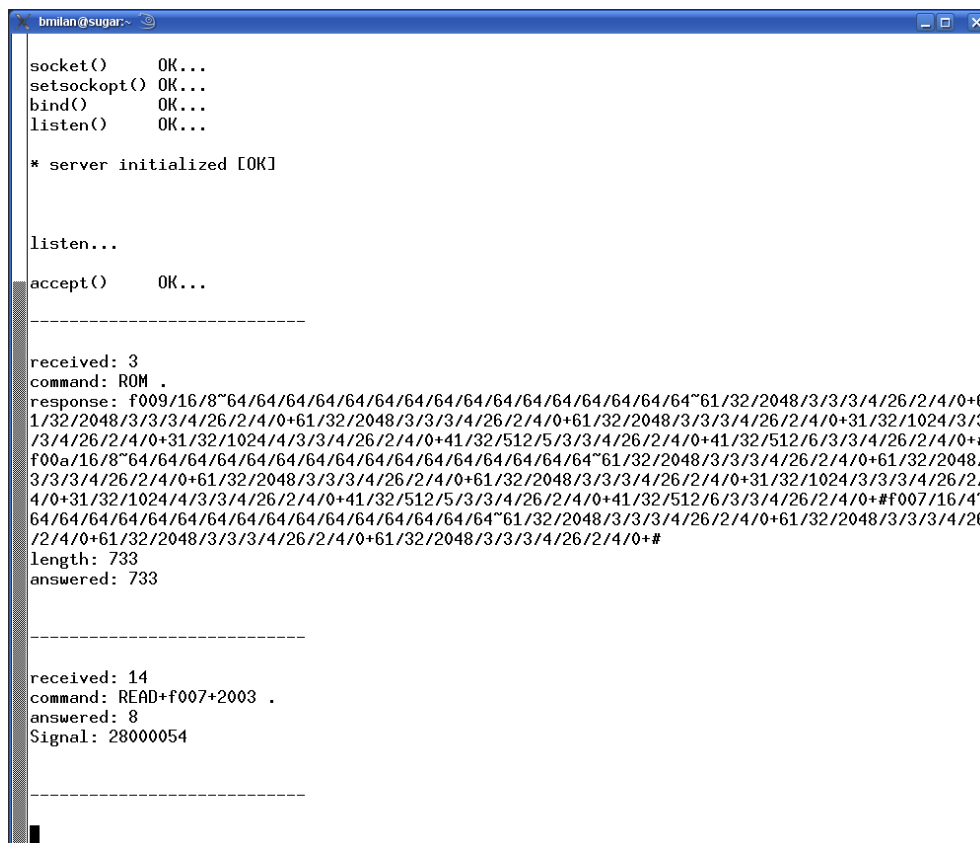
Figure 4.9: The entire hardware part. Configuration cells directly control the FIFOs and their controllers. The ROM contains all signal properties, whereas raw data is either stored in registers or buffered in FIFOs with additional timestamps. The number of FIFOs, registers, FIFO controllers and configuration cells varies from setup to setup, as well as their properties.

4.3 Software Monitoring Section

In order to acquire the monitoring data over TRBnet, a software counterpart is used to gather the buffered hardware signals. A TCP/IP server is additionally necessary to extract the data over the Ethernet. Outside the TRBnet, a client sends monitoring instructions specified online by the user and gathers the responses. In this way, the client on any PC can use the server to control the monitoring system over the Ethernet line. Both software parts, the client and the server, are written in C programming language.

4.3.1 The Server

The main purpose of the TCP/IP server is to control the FIFOs, gather all monitoring signals and export them outside of the optical network. It accomplishes this task by receiving the corresponding instruction from the client as a string. Currently, only three instructions are necessary: *ROM*, *READ* and *WRITE*.



```
bmilan@sugar:~  
socket()      OK...  
setsockopt()  OK...  
bind()        OK...  
listen()      OK...  
  
* server initialized [OK]  
  
listen...  
accept()      OK...  
-----  
received: 3  
command: ROM .  
response: f009/16/8~64/64/64/64/64/64/64/64/64/64/64/64/64/64/64/64~61/32/2048/3/3/3/4/26/2/4/0+6  
1/32/2048/3/3/3/4/26/2/4/0+61/32/2048/3/3/3/4/26/2/4/0+61/32/2048/3/3/3/4/26/2/4/0+31/32/1024/3/3  
/3/4/26/2/4/0+31/32/1024/4/3/3/4/26/2/4/0+41/32/512/5/3/3/4/26/2/4/0+41/32/512/6/3/3/4/26/2/4/0+#  
f00a/16/8~64/64/64/64/64/64/64/64/64/64/64/64/64/64/64/64~61/32/2048/3/3/3/4/26/2/4/0+61/32/2048/  
3/3/3/4/26/2/4/0+61/32/2048/3/3/3/4/26/2/4/0+61/32/2048/3/3/3/4/26/2/4/0+31/32/1024/3/3/3/4/26/2/  
4/0+31/32/1024/4/3/3/4/26/2/4/0+41/32/512/5/3/3/4/26/2/4/0+41/32/512/6/3/3/4/26/2/4/0+#f007/16/4~  
64/64/64/64/64/64/64/64/64/64/64/64/64/64/64/64~61/32/2048/3/3/3/4/26/2/4/0+61/32/2048/3/3/3/4/26  
/2/4/0+61/32/2048/3/3/3/4/26/2/4/0+61/32/2048/3/3/3/4/26/2/4/0+#  
length: 733  
answered: 733  
-----  
received: 14  
command: READ+f007+2003 .  
answered: 8  
Signal: 28000054  
-----
```

Figure 4.10: The server output in a Linux console. The 733 characters long stream represents the contents of three ROMs. The ROMs contained 4–12 FIFOs that needed to be encoded correctly.

ROM. The ROM command allows the server to execute a call to all monitoring FPGAs and acquire their ROM contents. A large stream is returned by the TRBnet containing an address and the contents for each ROM. Based on this stream, an internal C structure is filled that holds the entire monitoring setup. The applied number of chips alongside their addresses, FIFOs and register properties are first stored in this structure. Afterwards, the server generates an ASCII stream. The structure is encoded into a string and transmitted back to the client. Separators split different chips and FIFOs in the stream. The '#' symbol separates two consecutive FPGAs, whereas '~' distinguishes the three parts of one single FPGA stream: the header, the register part and the FIFO part. The header part contains the address and the numbers of FIFOs and registers. Each value is always separated with a '/' symbol. The register part contains register sizes only and the FIFOs in the FIFO part are additionally separated using the '+' sign. Figure 4.10 shows some server responses, where the first instruction is the ROM command.

READ. The read command can be used for a single read or a blockwise readout. In both cases, additional parameters are decoded from the instruction. The complete command looks more like: "READ+F003+2001". The string is split according to the '+' character and the first value (F003) is the hexadecimal TRBnet address of the chip, where the second (2001) resembles the hexadecimal storage cell address on that particular chip.

For a blockwise read, the number of data packets can be additionally provided together with the option to increase the address range with each read, or leave it as it is. By increasing the address automatically, several consecutive storage cells can be read out at once, for example "READ+F003+2001+3+1" would acquire monitoring data from the TRBnet address 0xF003 and the cells 0x2001, 0x2002 and 0x2003. If the last option in the example is a '0', only three reads to the cell 0x2001 would be performed.

WRITE. Contents of a configuration cell can be altered using this command. Again, a single or a blockwise write instruction can be released into the TRBnet. Parameters are again the chip and the cell addresses. This time the data needs to be supplied as well, thus the command syntax is:

1. WRITE+F003+1802+0+ON
2. WRITE+F003+1802+3+OFF
3. WRITE+F003+1802+11101011

The first example resets the FIFO (bit '0' is set to *high*), the second unhalts it if it was halted and the third sets the entire configuration vector⁶. If a number is provided after the cell address, followed by an 'ON' or 'OFF', then a single write to the specified bit is executed. Since bit '0' controls the reset, if it is turned on, the FIFO will reset itself (first example). After a successful execution, a confirmation is sent back to the client.

Server Operation

The server is activated on a particular TRB in the Linux-shell and listens on a certain port non-stop for incoming calls. It uses the two 16 bit lines connected directly to the FPGA to commu-

⁶In this example the configuration cell is 8 bit large.

nicate with the hardware. The software relies on an implementation of the TRBnet *libtrbnet* classes developed in Munich. The tool already provides basic TRBnet support over a Linux shell and grants access to the RegIO. These commands are applied as high-level instructions to acquire monitoring data and write to the configuration cells.

It is possible to send a request to every TRBnet node at once by using the address *0xFFFF*. In this case, the response is a huge array with the requested signal coming from all network nodes. The incoming data is packed and transmitted to the client afterwards. Current version does not support this function due to the manual grouping option which is more secure.

4.3.2 The Client

The monitoring client uses a TCP/IP interface to communicate with the server. It can be started from command line in a Linux shell, in which case the user selects predefined control commands to send to the server. The command line menu currently contains nine commands, as shown in figure 4.11.

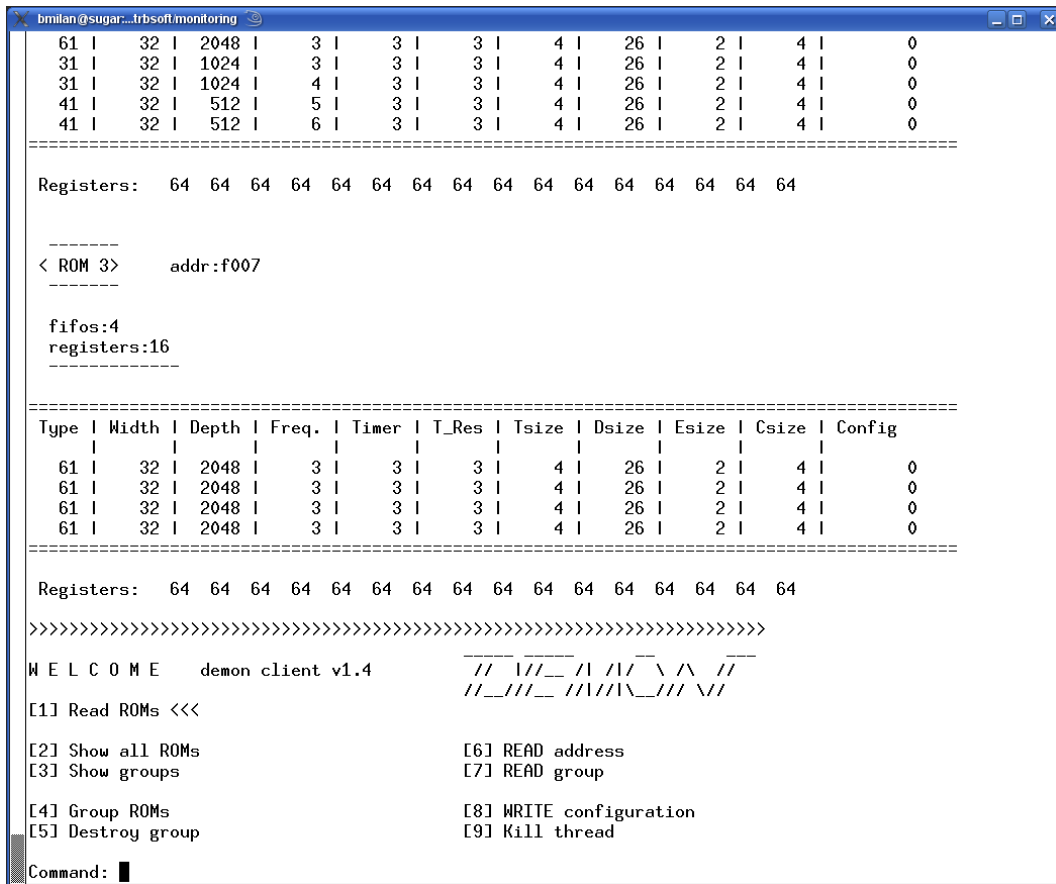


Figure 4.11: The client uses a simple menu to initiate user commands. Currently, the contents of a ROM are displayed on the screen. ROM3 has the TRBnet address *0xF007*, four FIFOs and 16 registers with all the properties from the lower table. All the registers are 64 bit wide, in this example.

The first command is to read out the ROMs. The client receives the stream containing every ROM content in an augmented ASCII string. Every monitoring chip is separated by the '#' sign and contains additional separators, as explained in section 4.3.1. The client parses this string and stores the monitoring setup in an internal datastructure. After the initial step is performed, the monitoring can begin.

The user has the possibility to display the entire system on the screen and group individual chips. The entire setup will be used to monitor different detectors, therefore related setups can be grouped together to read them out with a single command. The entire group is handled by sending one *READ* or *WRITE* command to all the individual addresses in the group. This procedure can also be initiated in a POSIX thread. In this case the user may specify the readout frequency, after which the client starts to gather all signals constantly in a loop, until its thread is dismissed. The user can always choose addresses, cells and options from a list, which is generated at run-time according to the current setup.

Controlability

The client contains the entire setup and needs therefore to perform some additional validity checks. First of all, after each write, the client awaits a confirmation. If it is received, the contents of the configuration cell are updated in the internal C structure. In this way, the user can always determine the current FIFO state. Second, if a cell needs to be read out and its size exceeds 32 bit, its contents are read out twice. The current RegIO version can not handle datawords larger than 32 bits. Therefore, larger monitoring signals are buffered in the FIFO- and the register-specific multiplexers. The multiplexer sends the large dataword in two segments. During the first read, only the lower 32 bits are sent back. Afterwards, the multiplexer awaits the second read to transmit the second part, without reading the FIFO or the register out again. The client combines both data chunks in the end. The client additionally needs to support correct readout approach. The read out data needs to contain sane values. Since the internal latency amounts to 10 clock cycles, data can not be read out faster than that. If a packet is read out, a new one can be written to the FIFO in its place. Thus if the write frequency is higher than 10 cycles, the packets inside the FIFO get fragmented and inconsistent. The write frequency has been slowed down to 10 cycles and might lead to undesired effects, i.e. to data loss. In order to maintain consistent data sets, the FIFO can be halted before it is read out blockwise. The client checks the frequency on blockwise readout requests and informs the user to halt the writes, prior to reading the FIFO out.

Another important feature is the grouping of similar readout nodes. The user may specify which nodes should be grouped together. A list of all TRBnet addresses is displayed and the set can be created. Afterwards, the collective read of a group can be performed. All nodes behave as they were only one single chip. Therefore the user simply specifies the cell and the readout method (normal, blockwise and/or threaded) and acquires the monitoring signals of the whole group.

In practice, the command-line tool is interesting only for quick debugging. Appropriate detector analysis and more convenient monitoring can only be performed using graphical user interfaces or at least data visualization frameworks. Therefore, the client has been completely embedded into an EPICS API, as presented in the next section.

4.4 EPICS Application

The Experimental Physics and Industrial Control System (EPICS) [EPI] is a widespread open source software platform. It combines a vast number of different applications to provide a generic infrastructure and facilitate the operation of scientific instruments throughout the world. It basically creates a distributed control system, i.e. a network with servers providing physical data which are stored internally in so-called **process variables** (PVs), and clients⁷ that can process the PVs in many different ways and even visualize them. There are various client tools available and the possibilities are practically endless, however in this thesis, the EPICS system is only partially used specifically for data visualization and GUI control.

4.4.1 Brief Introduction

The EPICS software has been developed to regulate extremely large and complicated systems and to grant access to the usually large device network combined with them. It creates a new software bus for data interchange and allows in this way communication between the user(s) and the hardware. The network protocol is called **channel access** (CA). It provides high bandwidth for internal communication. The servers usually have direct access to the hardware devices. The data therefore originates from real-world physical systems (accelerators, telescopes, etc.) and is published to the CA over the servers which are also termed **input/output controllers** (IOCs). Figure 4.12 sums up the whole procedure. The IOCs provide PVs into the CA and afterwards the client-side applications visualize the PVs gathered from the CA. Once a CA is created, arbitrary clients may attach to it, however security measures can be initiated to prevent the access to some PVs. In this way, any software infrastructure can be created to support the experiments. With the EPICS base installation, a large library of C/C++ headers is also included. EPICS is preferably designed to support PV access with any C application.

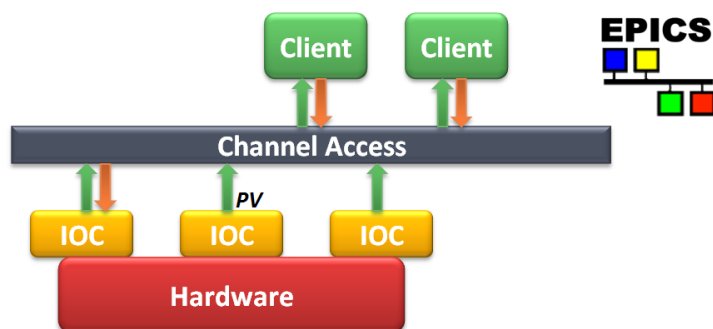


Figure 4.12: The basic EPICS system contains client(s), server(s) and the CA. All clients can execute different visualization software and the IOCs operate independently as well. Practical setups can involve hundreds of clients and IOCs. On the top right side, the EPICS logo is shown.

⁷The client is also referred to as the Operator Interface (OPI)

Before the CA can be used, an internal database containing all applied signals and procedures needs to be created. The database structure is very simplistic and contains merely **records**. A record on the other hand is a very complex object. It is defined over its fields, which denote besides its name and type what kind of processes the record can start, which PVs it contains and what are their scan intervals, value ranges and other attributes. The records are the active parts of an EPICS system and provide the CA functionality.

4.4.2 GUI Realization

There are many possibilities to use the EPICS system for monitoring. It is equipped with native monitoring support and the monitoring server from section 4.3.1 could be integrated into the EPICS API as well. This would facilitate the signal readout, but it would still require the server-client connection over the Ethernet, thus a communication overhead would be added. The data would be encoded into PVs and the server would listen to the CA for broadcasts. Therefore, the communication overhead has been minimized by applying a manual implementation of a TCP/IP server. On the other hand, all EPICS applications operate on PVs only. If an EPICS GUI is used for data visualization, the EPICS protocol needs to be implemented on the client side to support the PV access. Thus a simplified EPICS setup is used in this case, without the EPICS hardware support, as shown in figure 4.13.

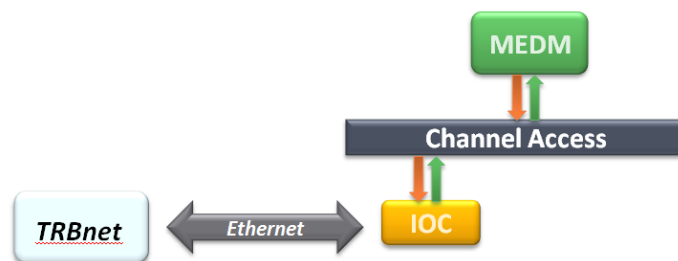


Figure 4.13: The simplified EPICS setup used for monitoring contains no hardware device. Instead, a modified monitoring client (embedded in the IOC) connects directly to the TRBnet TCP/IP server.

The final goal of this thesis is to develop a visualization platform for the acquired monitoring signals, but since the detector is still in development, no comprehensive information on signal types is known. Thus the GUI can not be developed effectively. Another aspect is also the variety of detectors and end-users. It is not feasible to create one single comprehensive GUI regulating all system types, as the values they monitor tend to differ, as well as their visualization method. But this is also one of the main reasons why EPICS should be applied. The API is highly modular allowing all users to customize their own settings, adapting itself to arbitrary monitoring setups. To demonstrate the compatibility between the DEMON system and EPICS, the API has been tested with a well known EPICS GUI, the Motif Editor and Display Manager (**MEDM**). However, the final version can only be created when more information on the experiments is known and a more complex TRBnet structure can be created. In following, only a proof of principle is provided and not the final GUI implementation.

MEDM is a versatile GUI containing a large library of components, frequently used for device monitoring and control (meters, bars, plotting utilities, text editors and many more). The components are controlled using internal PV interfaces. Each component has some basic properties, like size, range or timing intervals, but also a PV linked to it. Monitoring components display the meters and bars by reading from the CA, while control components write PV values into the CA.

After an IOC has been created, the EPICS database needs to be set. When the IOC is running, it automatically provides a CA derived from the database records. The IOC can be started directly from the MEDM GUI over a control button ('shell command'), so the user only has to start MEDM to gain full control. A minimal component arrangement has been created in MEDM to support this process. As previously mentioned, the IOC does not have direct hardware access. Instead, it implements a modified version of the monitoring client from section 4.3.2 to communicate with TRBnet.

Implementation Details

In the final version, the user will control the modified client over MEDM control buttons and dynamic menus. In the current implementation, only basic functionality has been realized to demonstrate the compatibility between the monitoring system and EPICS. An implementation of the full GUI is not feasible due to time limitations of this thesis. However, there have been exemplary source codes generated to facilitate the implementation of the GUI later on.

The IOC contains a precompiled C application. The application contains a callback function which performs following steps:

1. Get an instruction from a special PV (pv_1)
2. Send the instruction as a string to the server
3. Gather the server result
4. Store the result in a predefined PV (pv_2)

After a click on the appropriate button in MEDM interface, a shell command is executed to start up the IOC on the client Linux PC. While starting, the IOC executes a script, loading all the necessary records from the database and putting pv_1 in a *subscription* state. When the PV is in the subscription mode, the CA automatically monitors changes to that PV, after which it executes the callback function from above to start the monitoring process. After the initialization script, the IOC is ready and listens for user commands.

The commands are initiated by the user in MEDM. They can be typed directly in a textfield on the screen (the blue input field in figure 4.14). When the user presses the *SEND* button afterwards, pv_1 changes its value, initiating the callback function due to the subscription method. Thus the command, written by the user in the textfield, can be forwarded to the server. The server reply is then displayed on the screen over pv_2 . A demonstration of the minimalistic GUI is shown in figure 4.14.

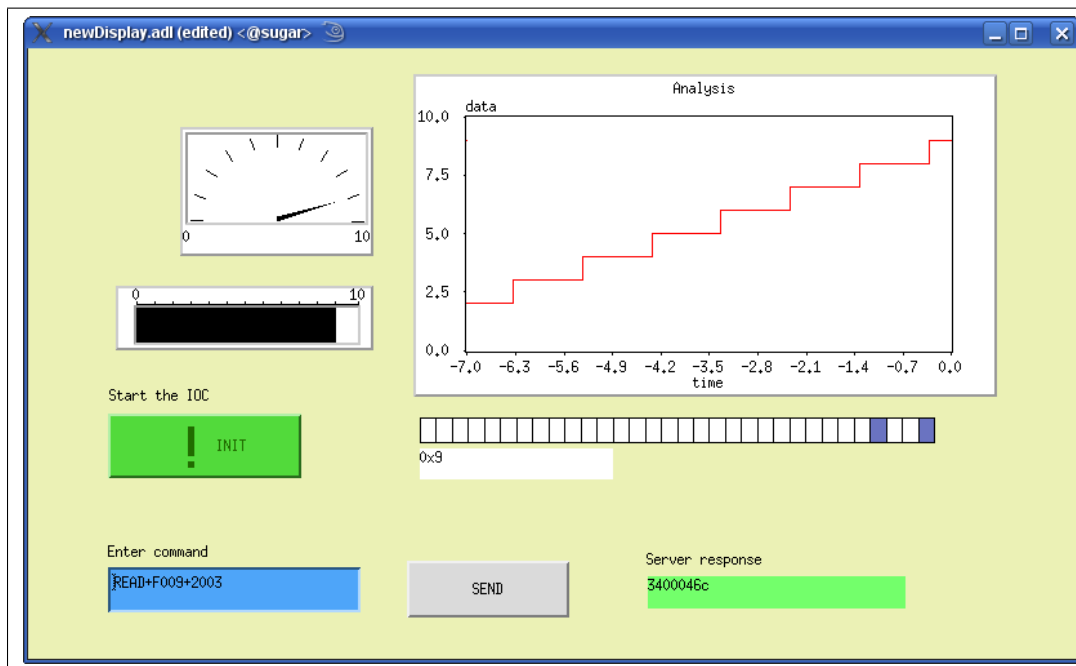


Figure 4.14: A minimalistic EPICS GUI. The meters, bars and fields in the upper part are just examples and do not display real monitoring values. The *INIT* button should be pressed first to start the IOC and set up the CA. In the lower part, the user may execute a command typed in the blue textfield and send it to the server by pressing *SEND*. The server response is displayed in the green field.

4.5 Summary

In figure 4.15, the entire monitoring architecture can be observed. Multiple input signals are acquired through detector readout nodes or FEE. The monitoring data is stored locally on the chip, with a suitable timestamp. The storage process runs in parallel and an increased number of FIFOs and registers does not affect the performance. Over a RegIO interface the system can be controlled and read out. After a *READ* signal, the FIFO contents are sent to the monitoring server in 32 bit format. The server interprets basic RegIO functions (single/multiple *READ/WRITE*) coming by the client over the Ethernet line. On the client PC, the user may either specify which instruction to send over a Linux console, or by applying an EPICS GUI (which is currently still in development). As shown in figure 4.15, the client can group similar chips and access them all at once. Also an automatic refresh of a given address or a group is supported.

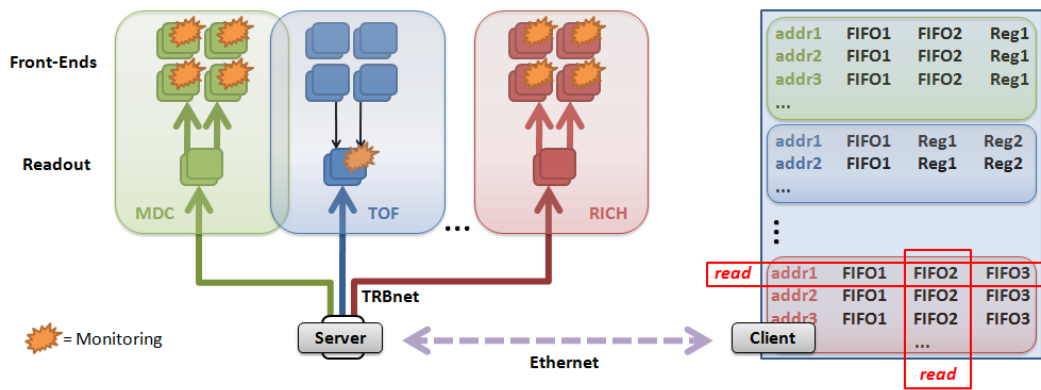


Figure 4.15: A scheme of the entire monitoring procedure. Monitoring can actually be performed on any TRBnet FPGA, besides the server. The client allows different readout methods to be performed.

5 Evaluation of the Work

The proposed solution has been extensively tested under normal conditions (no beam time). A simple TRBnet setup has been created and the endpoints have stored dummy data in their FIFOs and registers. Several different monitoring setups have been analyzed this way.

5.1 DEMON Tests

At first, only two TRBs were connected together. One TRB was acting as the TRBnet endpoint, while the other as the monitoring server. The server was able to directly read out the monitoring signals of the endpoint through a Linux console. The dummy data has been composed to count clock cycles. After acquiring it, the cycles have been incremented according to the defined FIFO frequencies. Therefore, the FIFO controller is operating correctly reflecting exactly the preset frequency.

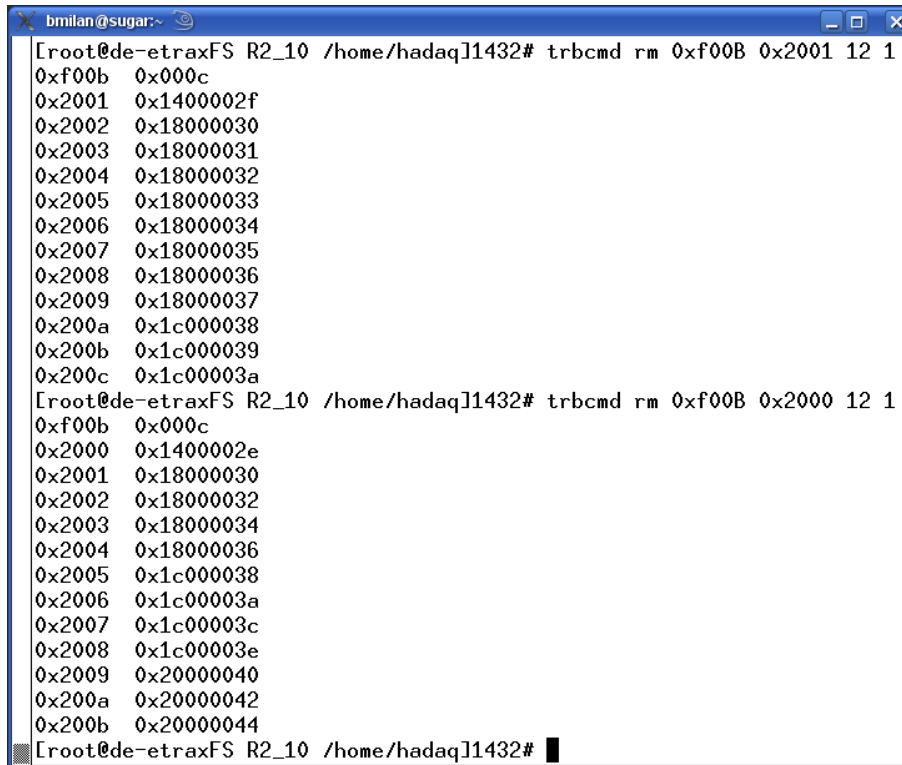
FIFO	Frequency	Data Test	Ringbuffer
BRAM 16×1024	high	passed	OK
BRAM 32×512	highest	passed	OK
BRAM 32×512	high	passed	OK
BRAM 32×1024	high	passed	OK
BRAM 32×2048	highest	passed	OK
BRAM 32×2048	high	passed	OK
BRAM 32×2048	moderate	passed	OK
BRAM 64×512	highest	passed	OK
BRAM 64×512	high	passed	OK
BRAM 64×1024	high	passed	OK
LUT 32×16	high	passed	OK
LUT 32×32	high	passed	OK

Table 5.1: All the cells have been examined carefully with a logic analyzer. All modes were operational and produced valid output.

Nearly all FIFO types were tested over the console, using 1–3 predefined frequencies: highest (every clock cycle), extremely high (every second clock cycle) and moderate (every 16-th clock cycle). The test results are shown in table 5.1. All FIFO contents have been analyzed carefully using a logic analyzer. The ringbuffer-, as well as the standard-mode of the presented FIFOs are operating correctly. Lastly, the bus width has been varied to analyze 32 bit and 64 bit behavior. As expected, when accessing the 64 bit FIFOs, two consecutive reads are necessary to obtain the

complete data packet. Again, the entire data set inside the storage cells was intact and could be acquired without any difficulties.

Having the hardware part fully operational, the server and the client were tested next. The server-software was able to access and control the individual FIFOs with no problems. The data could be transmitted to the client and client commands were interpreted correctly. So far, neither data loss nor errors could be detected. The ROM has been read out and contained sound values allowing the exact reconstruction of the monitoring setup. FIFO and register data could be printed to the console on the client side, as shown in figure 5.1, and contained exact values stored in the cells. The configuration bits have been set to test different FIFO modes afterwards.



```

bmlan@sugar:~
[Root@de-etraxFS R2_10 /home/hadaq]1432# trbcmd rm 0xf00B 0x2001 12 1
0xf00b 0x000c
0x2001 0x1400002f
0x2002 0x18000030
0x2003 0x18000031
0x2004 0x18000032
0x2005 0x18000033
0x2006 0x18000034
0x2007 0x18000035
0x2008 0x18000036
0x2009 0x18000037
0x200a 0x1c000038
0x200b 0x1c000039
0x200c 0x1c00003a
[Root@de-etraxFS R2_10 /home/hadaq]1432# trbcmd rm 0xf00B 0x2000 12 1
0xf00b 0x000c
0x2000 0x1400002e
0x2001 0x18000030
0x2002 0x18000032
0x2003 0x18000034
0x2004 0x18000036
0x2005 0x1c000038
0x2006 0x1c00003a
0x2007 0x1c00003c
0x2008 0x1c00003e
0x2009 0x20000040
0x200a 0x20000042
0x200b 0x20000044
[Root@de-etraxFS R2_10 /home/hadaq]1432# █

```

Figure 5.1: The main hardware test has been performed over a Linux console. The current FIFO contents are printed on the screen. The upper FIFO is operating at the highest frequency, while the lower one allows writes every second clock cycle (the most significant bits can be ignored, since they hold the timestamp and the event ID).

After the minimal setup, which proved the correctness of the hardware part, a more complex network has been created. One TRB was again acting as a server, while connected to a hub with three additional TRBs simulating the front-ends according to figure 5.2. Its purpose was to gather and analyze the monitoring data of several different monitoring setups. By doing so, all client functions have been verified.

The initial "libtrbnet" classes required a slight adaption in order to support the new hub electronics. After that, however, the client could obtain the monitoring values from each TRB suc-

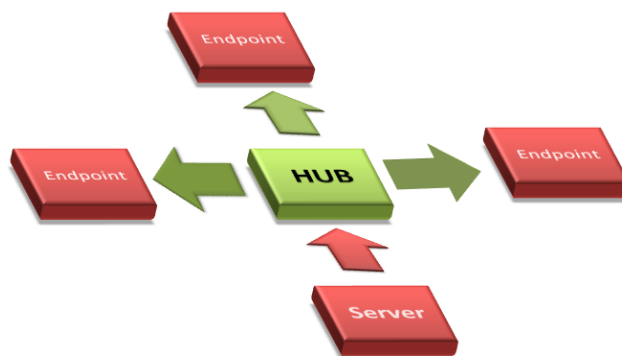


Figure 5.2: The second test involved a hub which connected three endpoints simulating monitoring signals with a TRB in server mode.

cessfully. The endpoints have been arranged in several different groups and in each case, the data has been extracted correctly.

5.1.1 Performance Measurements

One of the key aspects of the monitoring system (and even the main argument for its design) is the low resource consumption. The detector parts are constantly under high load and the FPGA logic nearly filled to the maximum. Hence, monitoring can only be performed if not interfering with the readout hardware. To create a comprehensive image, many different monitoring configurations have been synthesized on the Xilinx chip (xc4vlx40). The FPGA contains 38 864 Slices¹ and in some cases, they are over 90% occupied. In table 5.2, the results of the analysis are shown. The column 'Slices' denotes the total logic cell occupancy and plays the key role in the analysis. The Block RAM usage is another important aspect, without which the system could not be realized. In the first row, only the TRBnet setup has been generated without the monitoring system. The minimal system needed for monitoring consists of a full application (implements all four TRBnet layers) with the RegIO and the media interface. The consumption of this minimal system can be subtracted from every following result to obtain the net monitoring consumption. In the end, the internal hardware will already contain the TRBnet protocol and the media interfaces, so the resource consumption will tend to get much lower, approximately near the net consumption.

As can be seen, the basic setup containing 4 mixed FIFOs and 4 registers should be able to run on any detector chip. Its net consumption is around 7% of the Xilinx resources. However, such a setup is not advisable. When 4 Block RAM FIFOs are used, the net consumptions shrinks down to merely 4%. Even 12 BRAM FIFOs can then be used until the 10% boundary is reached.

There has been a huge difference observed between the consumption of the LUT- and the BRAM-FIFOs. As a result, it is advised to use the LUT FIFOs only if really necessary, since

¹One slice contains two logic blocks on the Xilinx Virtex chips. Each logic block consists of one Look-up Table and a D-flipflop (as well as some additional logic circuits).

DEMON setup	Slices	Flipflops	LUTs	BRAMs
–NONE– just the basic TRBnet entities without DEMON	1974 (10%)	2464 (6%)	3065 (8%)	9 (9%)
4 mixed FIFOs, 4×32 bit Registers 32×16(LUT),32×32(LUT),32×512,32×1024 Ringbuffer	3212 (17%)	3707 (10%)	4669 (12%)	13 (13%)
4 mixed FIFOs, 4×32 bit Registers 32×16(LUT),32×32(LUT),32×512,32×1024 No ringbuffer	3059 (16%)	3686 (9%)	4384 (11%)	13 (13%)
4 BRAM FIFOs, 4×32 bit Registers 32×512, 32×1024, 32×2048, 32×4096 Ringbuffer	2750 (14%)	3443 (9%)	4065 (11%)	21 (21%)
4 BRAM FIFOs, 4×32 bit Registers 32×512, 32×1024, 32×2048, 32×4096 No ringbuffer	2739 (14%)	3426 (9%)	4027 (10%)	21 (21%)
4 BRAM FIFOs, 16×32 bit Registers 32×512 (4) Ringbuffer	2710 (14%)	3343 (9%)	4061 (12%)	14 (14%)
4 BRAM FIFOs, 16×64 bit Registers 32×512 (4) Ringbuffer	2733 (14%)	3360 (9%)	4112 (11%)	14 (14%)
12 BRAM FIFOs, 4×32 bit Registers 32×512 (6), 32×1024 (3), 32×2048 (3) Ringbuffer	3741 (20%)	4738 (12%)	5542 (15%)	34 (35%)
12 BRAM FIFOs, 4×32 bit Registers 32×512 (6), 32×1024 (3), 32×2048 (3) No ringbuffer	3708 (20%)	4689 (12%)	5431 (14%)	34 (35%)
12 LUT FIFOs, 4×32 bit Registers 32×16 (6), 32×32 (6) Ringbuffer	5775 (31%)	6230 (16%)	7923 (21%)	10 (10%)
12 LUT FIFOs, 4×32 bit Registers 32×16 (6), 32×32 (6) No ringbuffer	5739 (31%)	6168 (16%)	7868 (21%)	10 (10%)

Table 5.2: Several different monitoring setups have been composed to determine the resource consumption on the Xilinx XC4VLX-10FF1148. The LUT FIFOs should be used with caution and only as a last resort. BRAM FIFO can be applied in ringbuffer mode with no problems.

they consume too many logic cells. The BRAM FIFO should always be preferred, assumed that enough Block RAMs are present on the chip. One Block RAM is always automatically used for the ROM, thus several Block RAMs need to be available to perform adequate monitoring.

It appears that the native FIFO mode and the ringbuffer mode consume nearly equal amount of resources. The ringbuffer requires merely three more slices per BRAM FIFO. LUT FIFOs however consume much more in the ringbuffer mode. Lastly, the registers can be widely used with almost no restriction. The resource consumption of 32 bit and 64 bit registers is nearly the same. The FPGA logic seems to generate them without any difficulties and it is advised to use them in large numbers for debugging, testing and monitoring.

5.2 Possibilities of the Monitoring Facility

A scheme of TRBnet from a different perspective is shown in figure 5.3. Its inner nodes are mostly hubs and readout boards, while the leaves represent the front-end electronics. The monitoring should be ideally performed on the lowest parts of the tree (the front-ends), as mentioned in the previous chapters, since they allow the most possibilities. The largest information about the current state of the detector can be found there. If for some reasons DEMON can not be integrated on the FEE level, then the readout boards need to compensate that loss of information. However, not only the lower levels can be used to hold the monitoring facility, but just any TRBnet node as well, especially the hubs. Analyzing the data flow through the hubs can reveal some crucial data acquisition properties, e.g. which channels fire at most, what are the busy times and how the data load evolves through the optical network. On the lower levels, again, the detector states can be analyzed better, the front-end behavior examined and optimization routines tested.

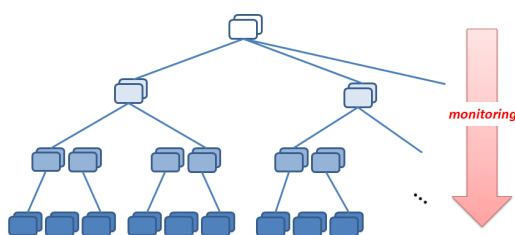


Figure 5.3: The alternative TRBnet structure. The lowest level represents the front-ends, while the next level the readout nodes. After that, in upper parts of the tree, the hubs reside. The actual tree does not have a root and the hubs can all be connected together, in this case however one single TRBnet node operates as the monitoring server (the root).

The generic nature of the monitoring facility allows unlimited applications. The facility can be even used for some online **hardware testing** purposes. The hardware can be programmed to generate a certain bit pattern, which is well known to the detector developers. If the pattern remains the same without any modifications, no hardware errors should be present. The pattern can be stored inside the FIFOs and the corresponding data packets marked with the control bits over the control port. The **control bits** indeed support any arbitrary algorithm inside the monitoring system that can be used for additional data routing and manipulation. In this case, the

data can be manipulated internally and fully automatically by the hardware, like the front-ends for example, without user intervention. The DEMON user simply needs to link the corresponding control signal coming from the front-end to the control port (and of course implement the marking algorithms). Another good use for the control port is that an algorithm can be easily written to control the FIFO controller from the hardware side. If the FIFO needs to be kept empty until a certain trigger has been released, the control port can block the FIFO writes. After the trigger, the FIFO can be written to, enabling time sensitive and highly accurate monitoring tasks, without user intervention.

The current implementation is (strongly) constricted. The number of maximal FIFOs and registers is bound to 20 and 32, respectively, and the FIFOs have been predesigned supporting only fixed widths (8, 16, 32, 64) and depths (16, 32, 512, 1024, 2048, 4096). An automatic FIFO generation of arbitrary sizes could not be successfully implemented, due to the software limitations of the FPGA manufacturers. The current release only supports the Xilinx and Lattice chips, but regarding the upcoming experiments, the monitoring facility seems to have enough to offer. Additionally, it is highly **extensible**. All aspects regarding this issue are shown in figure 5.4. New FIFO types can be easily created following the current approach from the corresponding VHDL-file ("data_cell.vhd") and relying solely on the IP-Core generators of the manufacturers. The new FIFO just needs to be inserted into the configuration file afterwards, in order to include it into the ROM and to provide the system with the necessary information (like width, depth, frequency, etc.). The extension of the internal bus beyond 64 bit is, however, rather inadvisable. Correct handshakes need to be present between the multiplexers and the RegIO module. More than 64 bit would require three or more consecutive reads to the same cell, hence in order to read one single packet, many read signals are necessary. It would seem more convenient to use two parallel 64 bit FIFOs for input signals larger than 64 bit, and combine their contents later on in the software. On the other hand, larger input chunks can be broken into smaller ones and stored in the same FIFO, so that, again, several reads can be used to obtain back the huge data word with the current implementation. Hence large input signals can be handled with minor adjustments very swiftly. The readout of 32 bit signals is nearly twice as fast than for larger signals, as the RegIO can not transmit more than 32 bit in one response. The limitation to 20 FIFOs and 32 registers has been chosen randomly, as the VHDL source code must possess some clear boundaries. In future, when more powerful FPGAs should be applied, the limit can be increased. The monitoring system is perfectly scalable to any given size.

It is very important to keep the system running without user intervention. Therefore, the monitoring signals can be acquired absolutely automatically. The preset FIFO frequency is used to limit the number of writes to the FIFO. The FIFO can be set to always contain either the oldest or the newest values. The latter can be achieved with the ringbuffer mode. Therefore, any FIFO can be preconfigured to best suit any given situation. The software part supports an automatic data refresh mode, where a certain frequency can be set (and also changed freely by the user) to initiate an automatic signal gathering procedure. The client and the server enter a loop and communicate in a thread to keep the values updated on the screen.

The signal acquisition needs to be handled with care. If the FIFOs do not get read out regularly, they lose information². The RegIO module together with the internal DEMON latency

²When the FIFO is full, either new data can not be written or the oldest data gets discarded (ringbuffer mode).

allows two consecutive reads only every 10 clock cycles. If the FIFO frequency is higher, the FIFO first needs to be halted and then read out, otherwise new data packets would not be stored in order of their arrival. The client already implements routines to support this issue. Data taking can be triggered internally by the hardware using the control port, as explained above. However, the user can reset the FIFO and start from the beginning, after providing the right signal. Additional controls over the slow control channel can be implemented using the configuration cells. The corresponding cell-bit just needs to be linked to the proper logic component it is meant to control.

One important issue is the chronological ordering of stored data packets. As explained in section 3.6, the FPGAs of one sub-detector system operate independently from one another, but they gather the same monitoring signals. In order to combine the data from all chips together, the timestamps need to get synchronized. For this purpose there have been three timers integrated with different precisions. Additionally, an event number (or a part of it) can also be pulled into the data packet for further association. During beam time, these measures should suffice to gain a comprehensive look inside the detector and all of its parts.

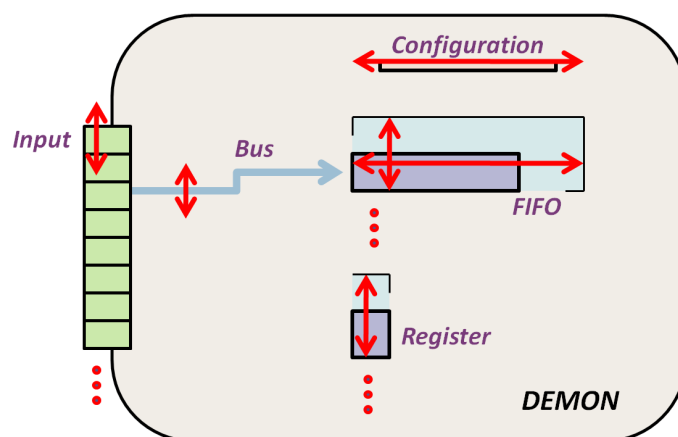


Figure 5.4: The hardware implementation is very extensible. The number of input signals and applied FIFOs and registers can be arbitrarily increased. The new structure is generated automatically based on the configuration file. A larger bus width, as well as the sizes of FIFOs, registers and configuration cells can also be implemented easily.

5.3 Evaluation

As concluded in the recent section, the monitoring system has met following requirements:

Resource friendliness When subtracting the TRBnet resource consumption on the chip, the monitoring tool shows reasonable results. As a final conclusion inferred from table 5.2, it seems that there can be much accomplished within 10 % of the large Xilinx chip resources. The basic monitoring system containing 4 FIFOs and 4 registers should be able to run

on any TRBnet chip. The DEMON system is therefore considered as lightweight and resource friendly. A broad use of registers and BRAM FIFOs is encouraged. However, if no Block RAMs are available, the system tends to become very expensive. In that case, only a minimal setup containing few small FIFOs should be used. The design should concentrate more on the registers.

Adaptability The system can be configured arbitrarily to suit any given detector. It can hold all currently applied detector signals in relatively large buffers. The readout mechanisms support the large variety of signals and handle them appropriately. The additional presence of control bits and configuration cells allow the system to adapt itself to any given situation. The possibilities are open to the developers to implement their own, enhanced algorithms inside the monitoring facility. Currently two operation modes of a FIFO are implemented (standard and ringbuffer) and they suffice to acquire all detector signals adequately. The global synchronization on all boards is achieved through different timer domains, event IDs and timestamp modulation. Moreover, each board can hold a different monitoring setup due to the orthogonal design and the applied ROM modules. Since only raw data is monitored, the user can select the visualization method freely.

On the other hand, high adaptability implies that the user performs all the customizations manually. Depending on the system, this step tends to get very time consuming.

Extensibility The system is absolutely scalable, supporting indefinitely many storage cells. The FIFO sizes can be enlarged easily, as well as the internal bus width. Current version contains some limitations which can be, however, continuously expanded. All the entities are derived automatically from the configuration file. If the number of signals increases, the multiplexers are extended accordingly, as well as the entire internal structure. The ROM also contains enough free addresses to hold more than 112 entries (20×4 FIFO properties + 32 register sizes).

Controlability On the hardware side, the monitoring system operates on its own. The hardware can automatically perform some internal actions over the control port and the FIFOs buffer signals with preset frequencies. On the client side however, the user can read out the cells and write their configurations. The only way to control the FIFOs and change their operating modes is over the configuration cells. They can be up to 32 bit in size. The client additionally offers more functions. The user can group similar boards and read/configure them collectively, as well as initiate reads in a threaded loop. The following data visualization leans on the large EPICS libraries and provides a high degree of freedom.

Usability The hardware part first needs an appropriate configuration, before the entire system can be used. The configuration process requires that the user understands the basic principles of storage cells and the internal bus. Every FIFO possesses many properties and they all need to be set manually. To accelerate and facilitate the configuration process, many constants are defined in the configuration file. After the main configuration, the user still needs to link the detector signals to the corresponding ports. In the end, a visualization scheme for each signal can be selected in the EPICS API.

The final customization step is also the most arduous one. First, the EPICS database has to be created to store all the monitoring signals. After that, each signal requires its own visual component on the screen. Some signals may require additional calculation algorithms, before they can be plotted or displayed. Altogether, much work is required by the user. This is at the same time the largest drawback in such a generic design. The system is extraordinarily adaptable and can be tuned to any desired operation, but the user has to do all the work.

The entire system is running very stable. The TRBnet performance is very good in the smaller test systems. Memory consumption of the server is sufficiently low. The server does not need to store large values and thus operates very efficiently on the TRB Linux system. Even if all TRBnet nodes are involved, the memory overhead should not exceed several megabytes. Every instruction is handled separately for each client. Only after the server sends back the response and empties its buffers, the next instruction can be executed. The only situation that should be avoided is to start many clients in parallel and to start acquiring ROM information on all of them at the same time. The memory could easily exceed the 128 MB on-board limit in that case. However, since the *malloc()* function is used appropriately, even then no problems should occur. The client should have access to much more RAM on the PC outside the TRBnet, therefore no difficulties should be expected there either (even not when using many threads).

Despite the great outcome so far, there is still some room left for improvement. When in future a more complete signal list can be acquired, the information on how to visualize each signal can also be stored inside the ROM (encoded in the signal type). The EPICS GUI would then be able to provide some core visualizations automatically, if needed. This step would significantly reduce the administrated user effort.

One huge limitation of the monitoring system which results in resource dissipation is the unified bus width. All FIFO- and register-cells are connected to the multiplexers and the input interface over equally wide drivers³. Therefore, if some signals are smaller than the bus width, the bus nevertheless drives many zeros through the system and wastes resources. There are two different approaches to solve this matter, however none of them has been considered, since they both require additional resources, so that the gain would not be noticeable in the end. One way is to use more complex multiplexers and create each driver differently, the other is to use many separate input ports with differing sizes in the input interface. The first proposal requires registers in the multiplexers and some more complicated controls, while the second calls for many additional multiplexers. Thus the bus width remains as it is.

With the proposed monitoring tool, digital hardware states can be acquired and visualized with some user intervention. To reduce the necessity of manual intervention into the source code, further work is required. More importantly, the current HADES setup is not completely developed. The performance of the tool could change when it is used in an environment with 500 boards. Monitoring data is transported on the least prioritized channel and could be blocked or delayed by triggers and event data. The laboratory tests are not sufficient, although they prove the correctness of the monitoring facility.

³In this context, the driver denotes the VHDL driver signal, carrying the actual monitoring load. The driver is the bus.

5.4 Closing Remarks and Future Work

There is yet a lot to accomplish, until the first beam time and practical application. This thesis provides the basis for a comprehensive detector monitoring facility. The design has aimed for prolonged applicability. Therefore it can be adjusted to store any signal and visualize it in any way the EPICS library permits. After the first experiment it is advisable to store the EPICS GUI for further work. The user needs to do all the adjustments, but in EPICS it can be done relatively fast and after a first skeleton application, it can be used as a template for further experiments.

Due to the large complexity of the EPICS API, no skeleton GUI could be created within the scope of this work. There are many ways to create one, but this would require many fine adjustments, so this is left to the end-user after more information about the experimental setup can be acquired.

As a proposal for the future application, some basic monitoring signals should be added in an internal signal library. The library could govern the monitoring signals and provide a method to create the EPICS records in the database automatically. New signals from following experiments should be added to the library, as well as their visualization components and settings. In this way, a comprehensive particle detector monitoring framework can be realized providing core signal information for future experiments.

Bibliography

- [Ago02] C. Agodi et al. The HADES time-of-flight wall. *Nucl. Instrum. Meth.*, A492:14–25, 2002.
- [Ang04] F. Anghinolfi et al. NINO: an ultra-fast and low-power front-end amplifier/discriminator ASIC designed for the multigap resistive plate chamber. *Nuclear Instruments and Methods in Physics Research Section A*, 533(1-2):183–187, 2004.
- [BRR08] W. Blum, W. Riegler and L. Rolandi. *Particle Detection with Drift Chambers*. Springer Verlag, 2nd edition, 2008.
- [BV05] S. Brown and Z. Vranesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill, 2nd edition, 2005.
- [Bal04] A. Balanda et al. The HADES Pre-Shower detector. *Nucl. Instrum. Meth.*, A531:445–458, 2004.
- [Bel09] D. Belver et al. The HADES RPC inner TOF wall. *Nuclear Instruments and Methods in Physics Research Section A*, 602:687–690, 2009.
- [Boh00] M. Böhmer et al. Lepton identification with the CsI based RICH of HADES. *Nucl. Instrum. Meth.*, A471:25–29, 2000.
- [Boh99] M. Böhmer. Das Auslesesystem für den Ringabbildenden Čerenkovdetektor im HADES Spektrometer. Master’s thesis, Technical University of Munich, 1999.
- [Chr04] J. Christiansen. High Performance Time to Digital Converter. *CERN/EP-MIC*, 2004.
- [EPI] Experimental Physics and Industrial Control System. <http://www.aps.anl.gov/epics/>.
- [Fro08] I. Fröhlich et al. A General Purpose Trigger and Readout Board for HADES and FAIR-Experiments. *IEEE Trans. Nucl. Sci.*, 55:59–66, 2008.
- [Fro09] I. Fröhlich et al. Future perspectives at SIS-100 with HADES-at-FAIR. *Invited talk at 47th International Winter Meeting on Nuclear Physics*, 2009.
- [GS08] C. Gruppen and B. Schwartz. *Particle Detectors*. Cambridge University Press, 2008.
- [GSI] GSI Helmholtz Center. <http://www.gsi.de>.
- [GSIa] CVD-Diamond Detector Applications in Heavy-Ion Measurements. <http://www-wnt.gsi.de/detlab/cvd/CVD-Applications.htm>.

- [GSIB] Ion-Beam Radiotherapy. http://www.gsi.de/portrait/Broschueren/ionenstrahlen_e.html.
- [GSIC] Facility for Antiproton and Ion Research. <http://www.gsi.de/fair>.
- [GSID] High Acceptance Dielectron Spectrometer. <http://www-hades.gsi.de>.
- [HAD09] G. Agakishiev et al. The High-Acceptance Dielectron Spectrometer HADES. *The European Physical Journal A, Volume 41, Issue 2*, pages 243–277, 2009.
- [Kri08] F. Křížek. *Study of inclusive electron-positron pair production in collisions of Ar+KCl at 1.756 A GeV*. PhD thesis, 2008.
- [Lan08] S. M. Lang. *Analyse der Elektronpaarproduktion im Stoßsystem Ar + KCl bei 1,76 AGeV*. PhD thesis, Frankfurt, 2008.
- [Lan84] L.D. Landau et al. *Electrodynamics of Continuous Media*. Pergamon Press, 1984.
- [Lor08] M. Lorenz. *Geladene Kaonen Produktion in Ar+KCl Reaktionen bei 1.756 AGeV*. Master's thesis, University of Frankfurt, 2008.
- [Mic08] J. Michel. *Development of a Realtime Network Protocol for HADES and FAIR Experiments*. Master's thesis, University of Frankfurt, 2008.
- [Mic09] J. Michel. *Upgrade of the Data Read-Out and Control Systems of the HADES Detector*. *HADES Collab. Meeting, Sesimbra*, 2009.
- [Min08] M. L. Lang et al. ATCA-Based Computation Platform for Data Acquisition and Triggering in Particle Physics Experiments. *Field Programmable Logic and Applications FPL 2008*, pages 287–292, 2008.
- [Mun04] C. Müntz et al. The HADES tracking system. *Nucl. Instrum. Meth.*, A535:242–246, 2004.
- [Ok108a] V. G. Oklobdzija, editor. *Digital Design and Fabrication*, chapter 9. CRC Press, second edition, 2008.
- [Ok108b] V. G. Oklobdzija, editor. *Digital Design and Fabrication*, chapter 7. CRC Press, second edition, 2008.
- [Pac08a] Y. C. Pachmayer et al. Dielectron Production in C+C Collisions at 1 GeV/u and the Solution to the DLS Puzzle. *J. Phys.*, G35:104159, 2008.
- [Pac08b] Y. C. Pachmayer. *Dielektronenproduktion in $^{12}\text{C} + ^{12}\text{C}$ Kollisionen bei 1 GeV pro Nukleon*. PhD thesis, Frankfurt, 2008.
- [Pal08] M. Palka et al. The new data acquisition system for the HADES experiment. *Nuclear Science Symposium Conference Record, NSS '08. IEEE*, pages 1398–1404, 2008.
- [Sik01] A. Sikora. *Programmierbare Logikbauelemente*. Fachbuchverlag Leipzig, 2001.

- [Tar08] A. Tarantola et al. The Upgrade of the Multiwire Drift Chamber Readout of the HADES Experiment at GSI. *Nuclear Science Symposium Conference Record, IEEE NSS '08*, pages 2146–2149, 2008.
- [Tex02] *LVDS Application and Data Handbook*. Texas Instruments, 2002.
- [Tin00] R. F. Tinder. *Engineering Digital Design*. Academic Press, second edition, 2000.
- [Tra00] M. Traxler et al. The 2nd level trigger system of the HADES detector. *IEEE Transactions on Nuclear Science*, 47:376–380, 2000.
- [Tra01] M. Traxler. *Real-Time Dilepton Selection for the HADES Spectrometer*. PhD thesis, Gießen, 2001.
- [Tra07] M. Traxler. HADES Data Acquisition Upgrade. *HADES DAQ Meeting*, 2007.
- [Zei99] K. Zeitelhack et al. The HADES RICH detector. *Nucl. Instrum. Meth.*, A433:201–206, 1999.

A Explanation of '*DEMON_config.vhd*'

```
constant FIFO_NUM      : integer range 0 to 20 := 8;  
constant LOG_FIFO     : integer range 1 to 5  := 3;  
constant FIFO_BUS_WIDTH : integer range 8 to 64 := 32;  
  
constant REG_NUM      : integer range 0 to 512 := 12;  
constant LOG_REGISTERS : integer range 1 to 9  := 4;  
constant REG_BUS_WIDTH : integer range 2 to 64 := 64;  
  
constant CFG_SIZE     : integer range 4 to 32 := 4;
```

Listing A.1: The first three rows define the basic FIFO specifications. In the example, 8 FIFOs with 32 bit effective port width will be used. After that, the registers are specified (in this case 12, with 64 bit bus width). The logarithm always needs to be rounded upwards.

```
constant c_BRAM16x1024  :  
                std_logic_vector(7 downto 0) := "00100000";  
  
constant c_BRAM16x1024R :  
                std_logic_vector(7 downto 0) := "00100001";  
  
...  
  
constant c_LUT64x32    :  
                std_logic_vector(7 downto 0) := "10110000";  
  
constant c_LUT64x32R   :  
                std_logic_vector(7 downto 0) := "10110001";  
  
constant c_NULL       : std_logic_vector(7 downto 0) := "00000000";
```

Listing A.2: All FIFO types are encoded, so they can be used as Strings inside the configuration file (see listing A.4). The user should not change this part, unless providing new FIFOs definitions.

```
type cell_generic is record
    width : integer range 1 to 256;
    depth : integer range 1 to 16384;
    log_depth : integer range 1 to 14;
    control_bits : integer range 0 to 8;
    monitoring_type : std_logic_vector(7 downto 0);
    frequency : std_logic_vector(7 downto 0);
    timer_type : std_logic_vector(7 downto 0);
    timer_resolution : std_logic_vector(7 downto 0);
    time_size : std_logic_vector(7 downto 0);
    data_size : std_logic_vector(7 downto 0);
    event_size : std_logic_vector(7 downto 0);
end record;

type cell_generics is array(1 to 20) of cell_generic;

type register_generic is record
    width : integer range 0 to 256;
    control_bits : integer range 0 to 8;
end record;

type register_generics is array(1 to 32) of register_generic;

type cfg_generic is record
    init : std_logic_vector(CFG-1 downto 0);
    number : integer range 1 to 20;
end record;

type cfg_generics is array(1 to 20) of cfg_generic;
```

Listing A.3: *The FIFO is defined as a VHDL record, nevertheless some properties still have to be encoded binary (in the current version). Registers and the initial configuration are defined in the same way. This part should not be modified.*

```

constant fifo1:  cell_generic :=
    (32,2**11,11,4,c_BRAM32x2048R,"00000001","00000011",
      "00000011","00000100","00011010","00000010");
...
constant fifo20: cell_generic :=
    (32,2**11,11,4,c_BRAM16x2048,"00000000","00000011",
      "00000011","00000100","00011010","00000010");

```

Listing A.4: After the definitions, the instantiation has to be performed. Every FIFO, register and configuration has to be set individually. All values are written to each FIFO in the right order (width, depth, log, ctrl, type, freq, timer, t_res, t_size, d_size, e_size). All data is written to the ROM if the FIFO type 'c_NULL' or the register width '0' is not stated (which stop the chain).

```

constant fifo_generics : cell_generics :=
    (
        fifo1 , fifo2 , fifo3 , fifo4 , fifo5 ,
        fifo6 , fifo7 , fifo8 , fifo9 , fifo10 ,
        fifo11 , fifo12 , fifo13 , fifo14 , fifo15 ,
        fifo16 , fifo17 , fifo18 , fifo19 , fifo20
    );

constant reg_generics : register_generics :=
    (
        r1 , r2 , r3 , r4 , r5 , r6 , r7 , r8 , r9 , r10 ,
        r11 , r12 , r13 , r14 , r15 , r16 , r17 , r18 ,
        r19 , r20 , r21 , r22 , r23 , r24 , r25 , r26 ,
        r27 , r28 , r29 , r30 , r31 , r32
    );

constant cfgs : cfg_generics :=
    (
        cfg1 , cfg2 , cfg3 , cfg4 , cfg5 , cfg6 , cfg7 ,
        cfg8 , cfg9 , cfg10 , cfg11 , cfg12 , cfg13 , cfg14 ,
        cfg15 , cfg16 , cfg17 , cfg18 , cfg19 , cfg20
    );

```

Listing A.5: After the values are set, they are kept inside the records and used to generate the entire architecture during the synthesis process. This fragment requires no modification.

B FIFO Frequency Map

f	f (binary)	Write Delay
0	00000000	10 <i>ns</i>
1	00000001	20 <i>ns</i>
2	00000010	40 <i>ns</i>
3	00000011	80 <i>ns</i>
4	00000100	160 <i>ns</i>
5	00000101	320 <i>ns</i>
6	00000110	0.64 μs
7	00000111	1.28 μs
8	00001000	2.56 μs
9	00001001	5.12 μs
10	00001010	10.24 μs
11	00001011	20.48 μs
12	00001100	40.96 μs
13	00001101	81.92 μs
14	00001110	163.84 μs
15	00001111	327.68 μs
16	00010000	0.655 <i>ms</i>
17	00010001	1.31 <i>ms</i>
18	00010010	2.62 <i>ms</i>
19	00010011	5.24 <i>ms</i>
20	00010100	10.48 <i>ms</i>
21	00010101	20.97 <i>ms</i>
22	00010110	41.94 <i>ms</i>
23	00010111	83.88 <i>ms</i>
24	00011000	167.77 <i>ms</i>

f	f (binary)	Write Delay
25	00011001	0.33 <i>s</i>
26	00011010	0.67 <i>s</i>
27	00011011	1.34 <i>s</i>
28	00011100	2.68 <i>s</i>
29	00011101	5.36 <i>s</i>
30	00011110	10.72 <i>s</i>
31	00011111	21.44 <i>s</i>
32	00100000	42.88 <i>s</i>
33	00100001	1.43 <i>min</i>
34	00100010	2.86 <i>min</i>
35	00100011	5.71 <i>min</i>
36	00100100	11.43 <i>min</i>
37	00100101	22.87 <i>min</i>
38	00100110	45.74 <i>min</i>
39	00100111	1.52 <i>h</i>
40	00101000	3.05 <i>h</i>
41	00101001	6.1 <i>h</i>
42	00101010	12.2 <i>h</i>
43	00101011	24.4 <i>h</i>
44	00101100	2.03 <i>d</i>
45	00101101	4.06 <i>d</i>
46	00101110	8.13 <i>d</i>
47	00101111	16.26 <i>d</i>
48	00110000	32.52 <i>d</i>
49	00110001	65.05 <i>d</i>

Table B.1: The FIFO frequency map. These frequencies can be set in order to achieve the desired write delay.

C Generate-Statement for the FIFOs

```
gen_fifos : for i in 1 to FIFO_NUM generate
  the_FIFO : data_cell
    generic map(
      width => fifo_generics(i).width ,
      depth => fifo_generics(i).depth ,
      log_depth => fifo_generics(i).log_depth ,
      control_bits => fifo_generics(i).control_bits ,
      monitoring_type => fifo_generics(i).monitoring_type ,
      frequency => fifo_generics(i).frequency ,
      timer_type => fifo_generics(i).timer_type ,
      timer_resolution => fifo_generics(i).timer_resolution ,
      timer_size => fifo_generics(i).time_size ,
      data_size => fifo_generics(i).data_size ,
      event_size => fifo_generics(i).event_size
    )
    port map(
      CLK          => CLK,
      RESET        => RESET,
      CLK_EN       => CLK_EN,
      DATA_IN     => FIFO_DATA_IN( (i*FIFO_BUS_WIDTH)-1 downto
                                   (i*FIFO_BUS_WIDTH)-FIFO_BUS_WIDTH ),
      READ_IN      => combined_fifo_read(i-1),
      DATA_OUT    => combined_fifo_data((i*(FIFO_BUS_WIDTH+1))-1
                                         downto (i*(FIFO_BUS_WIDTH+1))-(FIFO_BUS_WIDTH+1)),
      READY_OUT    => combined_fifo_dataready(i-1),
      NO_MORE_DATA_OUT => combined_fifo_no_more_data(i-1),
      CONFIG_IN   => combined_cfg_data_out( (i*CFG_SIZE)-1 downto
                                             (i*CFG_SIZE)-CFG_SIZE ),
      TIME_IN     => fifo_timer_bus((i*32)-1 downto (i*32)-32),
      EVT_IN      => EVENT_NUMBER_IN,
      CTRL_IN     => CTRL_IN( (i*4)-1 downto (i*4)-4 )
    );
  end generate;
```

Listing C.1: The generics are mapped according to the configuration file. The port `DATA_IN` is decomposed and each '`FIFO_BUS_WIDTH`'-wide portion connected to the corresponding FIFO. The code for the register cells is very similar.

```
gen_fifo_timer_bus : for i in 1 to FIFO_NUM generate
  gen_ftimetype1 : if fifo_generics(i).timer_type = "00000001"
    generate
      fifo_timer_bus((i*32)-1 downto (i*32)-32) <= GLOBAL_TIME_IN;
    end generate;
  gen_ftimetype2 : if fifo_generics(i).timer_type = "00000010"
    generate
      fifo_timer_bus((i*32)-25 downto 0) <= LOCAL_TIME_IN;
      fifo_timer_bus((i*32)-1 downto (i*32)-24) <= dummy24;
    end generate;
  gen_ftimetype3 : if fifo_generics(i).timer_type = "00000011"
    generate
      fifo_timer_bus((i*32)-1 downto (i*32)-32) <= TRIGGER_TIME_IN;
    end generate;
end generate;
```

Listing C.2: *The correct timer is connected according to the 'timer type' from the configuration file.*

D Source Files

Name	Description
CFG_cell.vhd	The simple configuration cell. It controls the FIFO state.
CFG_MUX.vhd	The multiplexer for the configuration cell address range.
data_cell	The generic FIFO cell. It contains a large number of predefined FIFOs and the internal connection.
FIFO_controller.vhd	This entity regulates the FIFO writes and provides additional controls.
FIFO_MUX.vhd	The multiplexer for the FIFO address range.
monitoring_unit.vhd	The top entity containing all others. It generates the DEMON architecture and the wiring.
register_cell.vhd	The register cell.
register_MUX.vhd	The multiplexer for the register address range.
ROM.vhd	This entity generates the ROM contents based on the current configuration.
tcpserv.c tcpserv.h	The TCP/IP server source file and C header. It is able to start up the server and interpret user commands.
tcpclient.c tcpclient.h	The command-line TCP/IP client. It connects to the server and allows the user to gain access to the monitoring facility.
EPICS_client.c	The version of the monitoring client adapted to the EPICS API. It contains additional process variables and callback functions, as well as additional revised functions.
DEMON.db	The EPICS database supporting the basic monitoring system.

Table D.1: All created source files are presented in this table.

Acknowledgments

As first I would like to say many thanks to my dear mother, Slavica Grsic and my entire family for the support throughout my studies. Especially I thank my uncle Nikola Ćorković who knows much more about particle physics than myself and who supported my interest through the last few years. Additionally, I would like to thank Dr. Ashraf Abu Baker for his constant support throughout the recent years of my studies, the great job opportunity and the start into the research business and congratulate him once more on his new title. Suits you well!

At the institute, I would first like to thank Dr. Ingo Fröhlich and Jan Michel for extremely good support and extraordinary mentoring. Thanks to you, I have learned so much. Additionally, Dr. Prof. Joachim Stroth has all of my gratitude, who accepted me in his institute and granted me this great opportunity to contribute to a worldwide project, as well as for his great lecture on nuclear physics, that woke my interest and supported my decision to undergo this thesis. Also I thank many times Dr. Rer. Nat. Uwe Brinkschulte for the excellent support and the acceptance of this external thesis.

Christian Trageser, Dennis Doering and Kathrin Göbel I would like to thank for the comfortable working atmosphere and the many interesting talks and lectures on particle physics and beyond. My gratitude goes also, of course, to the rest of the HADES and CBM collaboration in Frankfurt for all the interesting questions, talks and presentations (Selim, Attilio, Samir, Christoph, Khaled, etc, etc, etc). Michael Traxler deserves additionally my gratitude for the great support and the initial engineering, as well as Sergey Yurevitch for his support. The rest of the DAQ team, I thank for building and maintaining the circuits I could use to build this thesis upon. Great design, good work!