Development of a Realtime Network Protocol for HADES and FAIR Experiments

Diplomarbeit

Jan Michel

Institut für Kernphysik

Goethe Universität

Frankfurt am Main

August 2008

Abstract

The High Acceptance Dielectron Spectrometer (HADES), located at the Gesellschaft für Schwerionenforschung (GSI) in Darmstadt (Germany), is used to investigate the electromagnetic structure of hadrons and dense hadronic matter. It is a fixed target experiment with several detectors for the measurement of leptons as well as hadrons. Constructed in 1996 - 2002, the detector electronics now are subject of a redesign. One part of the reconstruction is the replacement of the trigger distribution and data readout system.

The main topic of this thesis is the development of a new network protocol for trigger distribution, data readout and control functions of the detector. Based on the experience in past beamtimes, all these functions will be merged into one homogeneous system to allow for easy maintainability.

The main challenge for the network implementation are the different requirements of the respective data streams. Triggers must be distributed immediately to all detectors while the detector data needs a high bandwidth. Hence, the network is divided into several channels with different priorities allowing for quick switches by a special data format. A special feature of these channels is a locking behaviour: It forces the sender of a data packet to wait until all replies from the network nodes have been collected before it is allowed to send data on this channel again. This provides, as a replacement of the current busy-signal, a convenient way to detect the end of the detector dead time after a trigger signal.

A vital part of the data readout system is the merging of data streams. Transporting the data blocks from each front end board individually would cause a high fragmentation of the event data. Therefore, the data is merged into a single data stream in every network node that collects data from multiple origins. Interfaces to access the merged detector data stream are provided to enable preprocessing before the data is transported to the event building and storage system. Despite the strict requirements for the HADES experiment, the whole implementation is kept highly flexible so that the network protocol can easily be adapted to other purposes like FAIR experiments or the development of read-out electronics.

In spite of the fact that the protocol is media-independent, optical links form the backbone of the network. Programmable logic devices are used for the network nodes in particular for the front-end read-out cards. The code of the network protocol is based on VHDL, a common standard for configuring FPGAs and other programmable logic devices.

In this work, the implementation of the protocol that fulfills all requirements which are outlined first is specified. After a general discussion of the necessary elements, the implementation in hardware is described and perfomance measurements which have been done are shown.

Kurzfassung

Das High Acceptance Dielectron Spectrometer (HADES) an der Gesellschaft für Schwerionenforschung in Darmstadt ist ein Teilchendetektor, mit dem sowohl die Eigenschaften hadronischer Materie bei hohen Dichten als auch die elektromagnetische Struktur von Hadronen untersucht werden. Es handelt sich um ein fixed-target Experiment bei dem Protonen-, Pionen- oder Ionenstrahlen auf ein ruhendes Target geschossen werden.

Um die Eigenschaften der bei den Reaktionen erzeugten Teilchen zu vermessen, verfügt das Spektrometer über verschiedene Detektorsysteme. Die Unterscheidung zwischen Elektronen und Hadronen ermöglicht ein RICH (Ring Imaging Cherenkov Detector) in Kombination mit einem Pre-Shower-Detektor. Eine weitere Teilchenidentifikation ermöglichen zwei Flugzeitwände (TOF / TOFino). Letztere wird zukünftig durch eine RPC (Resistive Plate Chamber) ersetzt. Mehrere Vieldrahtkammern (MDC) ermöglichen zusammen mit einem Magneten die Impulsbestimmung der Teilchen.

Bisher wurden Messungen vor allem mit leichten Kernen, beginnend bei Proton-Proton-Reaktionen bis zu Ar+KCl Stößen gemacht. Um zukünftig auch Messungen mit schwereren Kernen, beispielsweise Au+Au, und höheren Energien durchführen zu können, muss das Datenauslesesystem erweitert werden. Unter Berücksichtigung der in vergangenen Experimenten gewonnenen Erfahrungen wird das Triggerverteilungs- und Datenauslesenetzwerk vereinheitlich und mit modernen Komponenten neu aufgebaut.

Der Schwerpunkt dieser Arbeit liegt dabei auf der Entwicklung eines Netzwerkprotokolls, das es ermöglicht, sowohl Triggerinformationen als auch Detektordaten über das selbe Netzwerk zu transportieren. Beide Arten des Datentransports haben sehr unterschiedliche Anforderungen an den Aufbau des Netzwerks: Während Trigger möglichst schnell zu allen Detektoren verteilt werden müssen, braucht der Transport der Detektordaten eine möglichst hohe Bandbreite, um große Datenmengen zu übertragen.

Diese Arbeit beschreibt zunächst die logischen Strukturen, welche für eine sichere und schnelle Datenübertragung sorgen, bevor dann auf die Umsetzung in Hardware eingegangen wird. Zuletzt werden HADES-spezifische Konfigurationen des sehr flexiblen Protokolls und Messungen zur Leistungsfähigkeit des Protokolls in verschiedenen Anwendungen vorgestellt.

Die Implementation des Protokolls erfolgt praktisch ausschließlich in programmierbarer Logik unter Verwendung der Hardware-Beschreibungssprache VHDL. Um die gegebenen Voraussetzungen zu erfüllen wird das Netzwerkmedium in mehrere logische Kanäle (Channels) unterteilt, die innerhalb der Hardware getrennt voneinander arbeiten. Dies ermöglicht ein schnelles Umschalten zwischen den verschiedenen Anwendungen um beispielsweise einen Datentransfer für die Übermittlung eines Triggers zu unterbrechen. Verschiedene Sicherungsmechanismen wie große Empfangspuffer und ein Handshake-Verfahren sorgen für die sichere und verlustfreie Datenübertragung.

Ein spezielles Feature des Protokolls ist der sogenannte Locking-Mechanismus. Dieser sorgt dafür, dass nach einem Transfer erst die Antworten von allen Netzwerkkomponenten abgewar-

tet werden müssen, bevor ein erneuter Transfer auf diesem Kanal erlaubt wird. Dies ist vor allem für die Behandlung von Totzeiten im Triggersystem von großer Bedeutung. Ein für die Datenauslese wichtiger Bestandteil ist das automatische Zusammenfassen und Bündeln von Datenströmen der einzelnen Detektoren. Möglichkeiten, einen Datenstrom während des Transports durch das Netzwerk abzugreifen und vorzuverarbeiten, sind ebenfalls vorgesehen.

Die Konfigurierbarkeit der erstellten Netzwerkkomponenten erlaubt sowohl die Implementierung in kleinen und kostengünstigen Bausteinen als auch die Erstellung von leistungsfähigen Netwerkknoten mit sechszehn und mehr Verzweigungen. Außerdem ist so eine leichte Anpassbarkeit an andere Anforderungen, zum Beispiel in FAIR Experimenten oder in der Entwicklung neuer Auslesesysteme gewährleistet.

Der Transport der Daten erfolgt in der momentanen Hardware-Ausstattung mit einer effektiven Datenrate von bis zu 1,2 GBit/s, wobei eine kostengünstige Umstellung auf bis zu 2 GBit/s mit momentan verfügbaren Komponenten möglich ist. Die Latenzzeit für die Verteilung der digitalen Triggerinformation über das komplette Detektornetzwerk beträgt weniger als 2,5 Mikrosekunden und hat so nur einen geringen Einfluss auf die gesamte Totzeit des Detektors.

Contents

List of Tables

List of Figures

1 Introduction

The High Acceptance Dielectron Spectrometer HADES, located at the SIS18 synchrotron at GSI in Darmstadt was built to measure e^+e^--pairs produced in collisions of protons and heavy ions[1] at energies of 1 to 2 GeV per nucleon. Here, only a brief description of the detector setup and the underlying physics will be given. Further details can be found in various publications [S$^+$95], [Gar98], [S$^+$04].

1.1 Physics Motivation

One interesting topic of todays nuclear physics is the better understanding of quarks, their interaction and finally their bound states, called hadrons. Due to the non-perturbative character of the QCD, the properties of these systems are not easily accessible by theoretical calculations. Here only numerical and approximative approaches like the lattice QCD, using discrete space and time dimensions, are possible.

To shed light into these questions, experiments have to be done that provide an insight into the properties of hadrons. The confinement of quarks prohibits a direct observation of single quarks in the ground state. At high temperatures the existence of a quark gluon plasma, the so-called deconfinement, is expected. Another topic are very dense hadronic systems, in which a change of mass for specific particles are predicted [BR91]. By the measurement of these modifications, one expects to gain more knowledge about how masses of hadrons are generated.

The timescale in which hot and dense matter exists in collision experiments is very short, typically about a few fm/c or 10^{-24}s. Thus, very short-lived probes are needed to directly probe the properties of the densest phase of the reaction. Vector mesons like the ρ have very short lifetimes ($\tau = 1.3$fm/c) and decay almost completely inside the fireball and are therefore predestined for such measurements.

Since the decay of vector mesons into pions has uncertainties due to final state interactions of the daughter particles, for the study of in medium effects of vector mesons their decay into e^+e^- has been proposed as the most promising probes.

Measurements of di-lepton mass spectra have been done by several experiments. The CERES [M$^+$07] and NA60 [Col06] experiment at CERN covered the energy range above 40 AGeV, while DLS [P$^+$97] (located at BNL) used low energies of a few AGeV.

[1] Up to now, runs with N+N, N+A, A+A and tests with pion beams have been done.

In the low invariant mass region a significant enhancement above the predicted production rates was observed, known as the "DLS-puzzle" [ECS]. To solve this puzzle and to further investigate, a second generation experiment, the HADES detector, has been built. With its much higher acceptance it is able to gather much higher statistics than the two arm spectrometer DLS.

In the HADES experiment energies of 1 to 2 GeV per nucleon for heavy ions and up to 3.5 GeV for protons are used [A$^+$07] [A$^+$08]. These energies are either below or only slightly above the production threshold of the ρ meson. The subthreshold production of vector mesons makes sure that they are produced via nucleon resonances in multi-step excitations and therefore require two collisions within a very short time. Due to the fact that this can only happen in the dense phase of the reaction this ensures only in-medium properties of the particles are measured. Opposed to that, in higher energy experiments vector mesons can be produced in later states as well.

For energies near production threshold, changes in the di-lepton mass spectrum due to in-medium effects have been proposed by theoretical calculations [BR91].

1.2 The HADES Spectrometer

The HADES spectrometer covers polar angles between 18° and 85° and is divided into six sectors, covering almost all azimuthal angles. It consists of several detector systems for particle identification and measurement of their energy and momentum.

The tracking of particles is provided by four layers of multi wire drift chambers, the MDCs, placed before and behind a magnetic field to measure particle trajectories and therefore their momentum. While passing through the gas filled chambers, every charged particle generates free electrons by ionizing the gas. Field wires produce a strong electric field to transport these electrons to the sense wires. The electric current generated there is then amplified and digitized. Each of the 24 MDC modules consists of six layers of sense wires, each tilted in a different direction to define the exact position of the charged particle.

TOF and TOFino are two time-of-flight detectors, each covering about one half of the polar acceptance range. A start detector placed close to the target provides the time reference. The time-of-flight combined with the measured momentum of the particle gives a method for hadron identification.

Moreover, a good and fast possibility for electron identification is needed. Two more detectors are added to provide this feature. The RICH[2] detector [B$^+$00] measures the Čerenkov radiation produced by fast particles when passing through a medium in which the speed of light is lower than their velocity. The radiator gas was chosen to be sensitive for particles with a Lorentz factor of $\gamma = 18$. In the HADES experiment only electrons can reach energies which are high enough while most hadrons remain below this threshold.

[2]RICH: Ring Imaging Č(h)erenkov detector

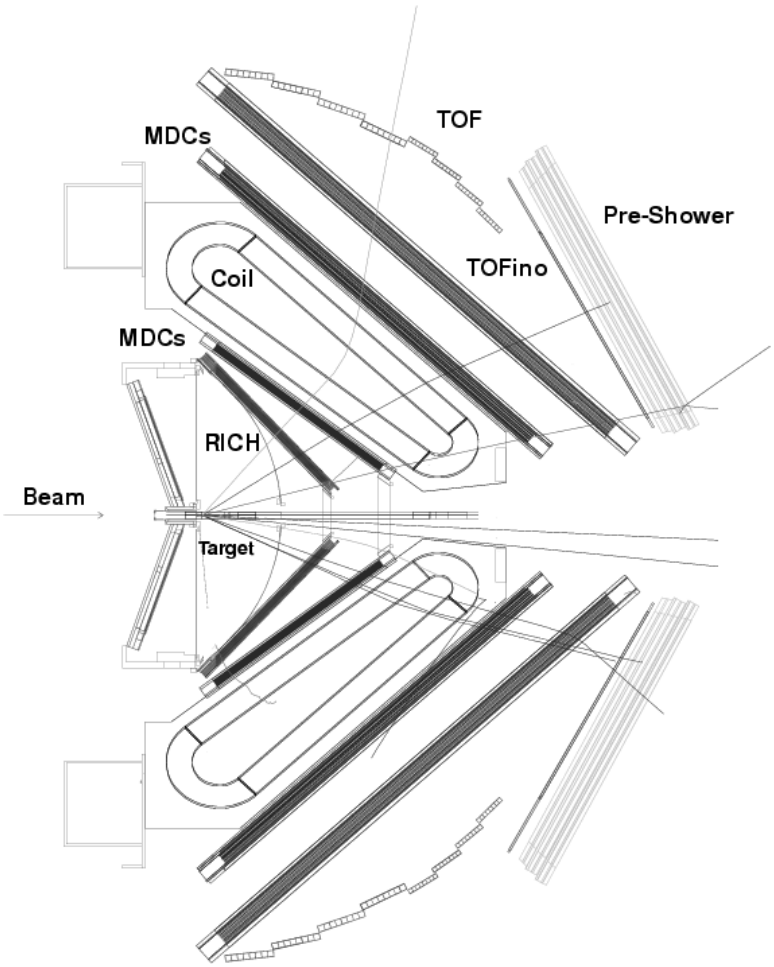

Figure 1.1: A schematic view of two of the six sectors of HADES [Tra01]. The different detectors in the first experiment setup are shown: RICH, MDC, TOF & TOFino and Pre-Shower. The newer Forward Wall (placed far to right) and the RPC (replacement of the TOFino) are not included. Several particle trajectories are drawn.

The photons are reflected by a mirror to an array of photo detectors in such a way that all photons with a fixed emission angle are focused to a ring on the focal plane. This is possible, since all electrons have a velocity of approximately the speed of light which causes the Čerenkov light to be emitted with a constant angle. Therefore, the lepton identification can be implemented by searching for rings with a fixed diameter on the pad plane.

The Pre-Shower detector consists of three layers of wire chambers separated by two lead plates [B+04]. The plates cause the electrons to emit bremsstrahlung which is then converted into e^+e^--pairs that are detected in the wire chambers. Since only electrons produce this shower of charged particles[3], it gives additional information for the identification of electrons.

Since the target is chosen to be small to reduce the rate of secondary collisions, most particles do not interact[4]. Thus, a veto detector placed behind the target detects non-interacting particles.

[3] Due to their higher mass, hadrons interact with the metal mainly via photoionization, but their energy loss is negligible compared to electrons.

[4] The interaction rate is about 1%.

2 The Current DAQ and Trigger System

The first step of the readout process is the digitization of the detector data which is initiated by a trigger signal. Depending on the the specific requirements for each detector, the raw analog data is pre-processed by amplifiers, signal shapers or converters before it is transported to digitization circuits. These converters are then able to measure e.g. the time, duration or height of each signal. Afterwards, data reduction algorithms like zero-suppression can be applied to the acquired data before it is further processed and eventually written to mass storage.

2.1 Trigger Concept

The high probability for events with a pure hadronic final state calls for effective trigger mechanisms to reject as many events without electron pairs as possible in a early stage. Otherwise the electronics would be blocked from the superior number of pure hadronic events and would not able to store the few interesting events.

The HADES trigger system is divided into two stages. The first trigger level is used to determine whether a reaction of interest took place. The second level trigger does a quick online analysis of parts of the detector data and checks for events with lepton candidates.

The LVL1 trigger is a trigger on the apparent multiplicity in the time-of-flight detectors. To simplify the detection, no reconstruction of specific particle trajectories is made but the hit multiplicity is encoded in a pulse height by analog summing units. A discrimination circuit is then used to select events with a high multiplicity which corresponds to more central collisions.

The time available for the trigger decision is very short: The LVL1 trigger signal is used as a common start for all TDCs. Hence, the LVL1 trigger signal must arrive at the TDC before the analog detector signal does. For that reason, delay circuits for the analog signals are employed to reach a time window of 100 ns for the trigger signal.

Now the digitized data can be processed in the Image Processing Units (IPU). For the RICH a pattern recognition algorithm finds the fixed sized rings produced by electrons in the detector and gives a first estimate for the numbers of electrons to expect in the event [L$^+$03]. Due to the low complexity but high number of parallelizable operations, small FPGAs[5] are used. The TOF IPU calibrates the measured values of the TDC[6] and applies corrections for path lengths. Here, a high number of floating-point computations is needed. Hence, the algorithms

[5] FPGA: Field Programmable Gate Arrays - universal, programmable logic circuits.
[6] TDC: Time to Digital Converter.

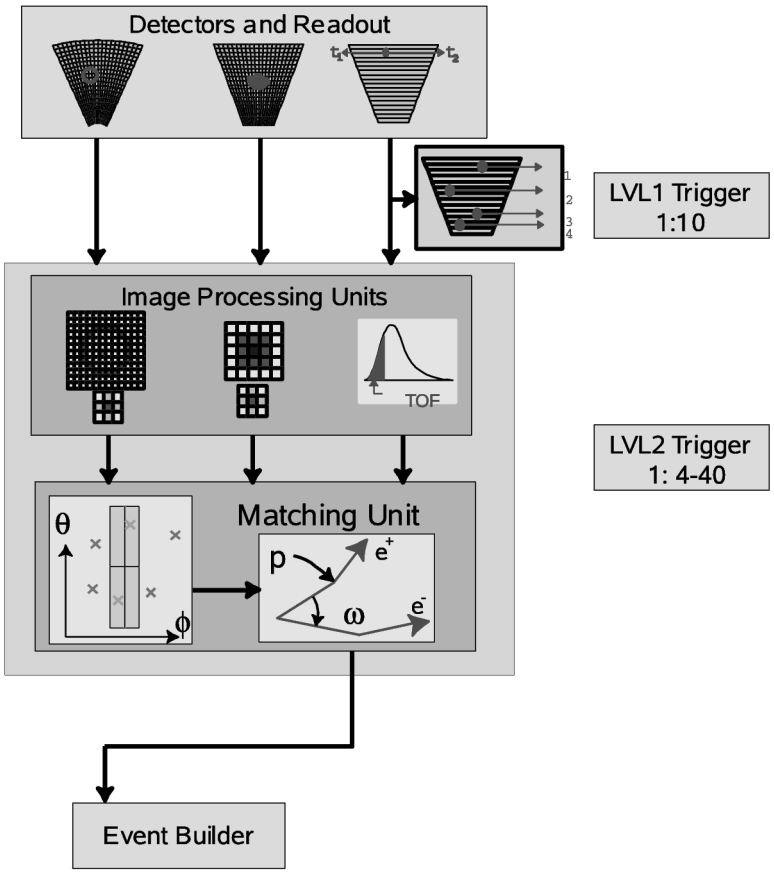

Figure 2.1: A schematic view of the HADES trigger system [Frö]. A two staged trigger system is used: The first trigger decision is based upon analog detector signals, while the second trigger is generated after parts of the data have been analyzed.

are implemented on DSPs[7].

The Pre-Shower IPU algorithm operates in two steps: First a search for local maxima in the first layer of the detector is performed. In the second and third layer then the deposited charge in a small area around each maximum is computed [Pet00]. The MDC data is not processed on-line, since unlike the other detectors it needs complex algorithms to deliver accurate information, which was not possible to be done in hardware with reasonable effort at the time HADES was built.

The LVL2 trigger decision is allowed to be made after a latency of several events as the detector electronics are capable of storing their data in internal memories. Here, the decision is based on the processed data of the IPUs.

The information of all three IPU subsystems is transported to the Matching Unit (MU) where the LVL2 trigger decision is made and distributed to all detectors. Upon a positive LVL2 trigger the data is transported to the event builder, a computer that combines all data of one event and passes it on to the data storage.

2.2 Current HADES Trigger Setup

The "Trigger Bus" is the backbone of the current trigger system and connects the central trigger system (CTS) with the detector trigger units (DTU). LVL1 and LVL2 triggers are transported on separate twisted pair ribbon cables carrying differential signals [Lin01]. Six signals are used to transport the trigger from the CTU to the detectors. Two are used as strobe signals, while four data lines transport trigger codes and trigger numbers. Another two lines are used to transport the status of the detectors back to the CTU. Here, the used RS485 signaling standard allows that all DTUs use the same cables to transmit their busy signals. The combination of the detectors signals is done by a so called "wired-or" technique: The bus line is in a logic high state as long as none of the boards actively pull the signal to the low state.

Each DTU receives the trigger information and starts the corresponding actions for the detector. The acquired data from the detector can then be tagged with the received trigger information. The data is transported to the image processing units (IPU), where the raw detector data is analyzed. In case of the Pre-Shower and TOF detectors, the IPU is combined with the readout electronics, while the RICH uses several VME boards.

The main limitation of the current DAQ setup are the differences of the trigger and readout systems between the detectors: The DTUs have to convert the data from the trigger bus to detector specific signals. Here, the internal data transport was developed independently for each detector and is not standardized as well. The data links between IPUs and the Matching Unit are again built up of independent data bus setups.

Due to the big number of different interface types and protocols, it is difficult to maintain the detector setup during runs, since there is no central monitoring port which supervises the whole

[7] DSP: Digital Signal Processor.

detector. Instead, several independent monitoring and debugging tools have to be used. This is mainly caused by the fact that the one network that is common to all detector parts, the trigger network, is not able to transport detailed status information but only a global busy signal. If the busy signal is not released, there is no direct way to determine which board causes the problem without employing other, detector specific monitoring tools.

Besides the maintenance issue, the vast number of electrial wires used for the data transfer affects the operation of the detector. There are fast switching digital signals on these lines, which introduce noise to the analog signals of the front end systems. Also the electrical connections from different points introduce ground loops that contribute to the noise level.

The matching algorithms are implemented on Digital Signal Processors (DSP) on VME boards. Compared with state-of-the-art systems they are slow (running at 40 MHz), not scalable and thus not capable of high event rates for heavy ion collisions. The used VME bus for the data transfer to the event builders is comparably slow. Parallelization of the readout by distributing the data to several VME crates would be a solution to speed up data transfer, but is very costly.

2.3 Changes to the DAQ and Trigger System

The modification of the trigger system is part of an upgrade program[8] for the whole HADES detector. The biggest change is the integration of two new detectors: In 2007 a forward wall was installed. It is placed at forward angles around the beam axis, detects spectator nucleons and gives information about the reaction plane of the event.

Moreover, the TOFino will be replaced by a Resistive Plate Chamber (RPC) [B$^+$06]. Its higher granularity improves the resolution at low polar angles and is not only needed in high multiplicity events in heavy ion collisions. At higher beam energies the scattering angles in the laboratory system become smaller and the fraction of particles hitting the RPC raises.

Therefore a multi purpose trigger and readout board (TRB) [Tra06] [F$^+$08] was built. It features a high speed optical connector, a large Virtex 4 FPGA and a TigerSharc Digital Signal Processor (DSP) as well as an Etrax FS CPU running Linux. A high speed connector allows to adapt the board to different requirements by attaching AddOn boards.

For instance the MDC readout system will be entirely replaced [Tar] with these boards: The front ends will be equipped with small FPGAs that are connected to a concentrator board that is built as an AddOn to the multipurpose TRB. In a second step, the connection between FEE and MDC-AddOn will be replaced by optical fibres.

[8] The HADES upgrade programm is supported by BMBF and EU.

3 The New Trigger and Readout Network

In view of the disadvantages of the current setup, a new network protocol as well as a new network layout had to be designed. In this chapter, the planned network layout and the requirements for the network protocol are described. A comparison with existing network protocols shows, whether it is possible to employ one of these for the HADES detector readout system.

3.1 Network Architecture

A brief view of the planned network layout is shown in figure 3.1. An optical network forms the backbone of the system that connects major parts of the readout and trigger system. Optical links have several advantages in the detector environment. Compared to electrical connections with the same bandwidth, the optical fibers are small and lightweight. They are not sensitive to electromagnetic influences and do not produce electromagnetic noise. Additionally, optical links avoid ground loops because all boards remain electrically uncoupled. Therefore their interaction with other electronics is negligible which leads to better signal-to-noise ratios in analogue signals and less bit errors in digital signals.

In the new system, the central trigger system (CTS) generates triggers that are distributed to the whole network, including the front end electronics (FEE). These perform the readout of the detector hardware and generate the event data. The data is then transported using the same optical network to the readout computers that process and store it.

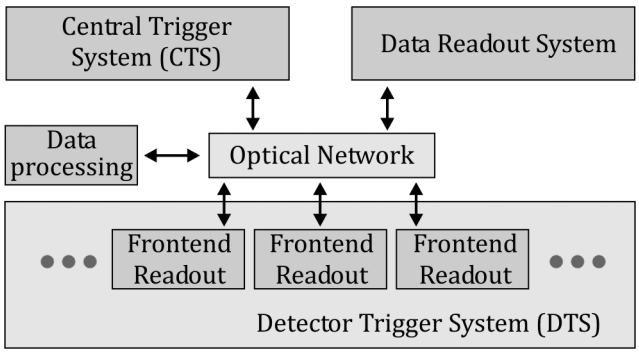

Figure 3.1: Network Layout. A central optical network connects all parts of the detector electronics like front end boards, the central trigger system and the readout system.

Processing units might be included into the network that acts on the data before forwarding it to the event builders. The use of dedicated hardware for this purpose can provide higher performance than algorithms running on universal computers.

3.2 Requirements for the New Network

In summary, the main aim of this work is to provide a multi purpose network which is able to transport detector data, triggers and slow control information. For all these tasks one common network protocol should be used throughout the whole detector, from the front end electronics to the central parts of the trigger system. The implementation of the protocol has to assure that every kind of application is able to use the network without having to cope with details of the implementation.

Since HADES is aiming for an event rate of more than 20 kHz, the response time for a trigger has to be in the order of a few microseconds only. Keeping dead times as short as possible is essential to reach high event rates without loosing too many events.

These strict timing requirements do not allow for an individual access to all readout boards to get the information about their busy status. For that reason, a fast way to determine when the whole system is ready to accept the next trigger is needed. This is done by merging all busy release signals from all boards automatically and forwarding them as a single data word to the central trigger system similarly to the old wired-or logic. This procedure can also be used to collect data from many readout boards and merge them into a single data stream.

Parts of the network links used for detector data transportation need to achieve the highest possible data rates. This requires a high speed connection that is capable of data rates of more than one gigabit per second. The peak data rate of the whole detector is estimated to be up to 3 GBit/s [Tra]. However, the network bandwidth can be smaller, since the readout will be located near the detector in terms of network topology and thus no part of the network is forced to handle all the data.

The data readout can be done in different ways: One possibility is to implement an Ethernet interface on one of the FPGA boards that converts the data received over the trigger network and transports them to a PC using a common network protocol. Another option currently being developed at GSI would be the use of a PCI-Express card which is equipped with a FPGA that can be programmed to receive data from the trigger and readout network and write it directly into the PCs memory.

Another extension compared to the current data acquisition system will be so-called “compute nodes”, FPGA based systems where all algorithms to process the data before it is delivered to the event building computers can be implemented [Kir07]. Therefore, the network has to provide a convenient interface to access data while it is transported over the network.

Moreover, the network protocol must be implementable in all kinds of hardware, from the newest PCI-Express card with dozens of Gigabits of bandwidth, to the smallest front end read-

out controller with only a few hundred Megabit bandwidth as needed for the new MDC readout for example. This calls for a highly modular and configurable implementation which can be adapted to any kind of hardware.

From the maintainability point of view, the network should be easy to be upgraded in the future. This includes the employment of mainstream hardware which will most probably be available for at least the next decade and to avoid to rely on special electronic devices.

Data transport must be reliable. There must be no arbitrary data loss under any circumstances, including environmental influences. Therefore handshaking and error correction must be part of the protocol.

3.3 Comparison to Existing Network Implementations

The most common solution for local high speed networks used today is Gigabit Ethernet. The hardware needed is not very expensive: Despite the optical transceivers it needs about 1500 lookup tables [Gao05] of resources inside a FPGA to implement the network controller. Additional logic has to be used to be able to transmit data according to higher level protocols.

Ethernet transports its data in so-called frames. Each frame consists of a header carrying information about the type of the frame and addresses followed by the data to be transported. How data is transported is defined by higher level protocols like IP[9] [Pos81a]. It provides a way to address network nodes not only using Ethernet but also over merely any other type of network. It therefore introduces another header containing again addresses and information about the data. The data, in turn, may consist of a datagram as defined in the user datagram protocol (UDP)[10] [Pos80]. A complete Ethernet frame containing IP and UDP headers is shown in figure 3.2.

When sending small packets, the protocol overhead is immense: A standard header for a UDP transfer has a length of 48 byte [Pos80] resulting in an overhead of 94% when sending one byte of data only. Being optimized for high data throughput rather than for low latency is the main disadvantage of Ethernet. Even a simple low level switch may introduce delays of tenth of microseconds before forwarding a packet.

Moreover, the IP-protocol does not include any kind of priority for special types of data. A mechanism to interrupt running transfers without data loss is not included as well. For that reason the transmitter would have to wait until the running transfer has been finished before sending a new frame in order to ensure data integrity. Such procedure would introduce latencies of up to 12 µs.[11]

Even though the protocol headers contain checksums to confirm whether the received data is valid, the transmitter has no guarantee that data has been received at all since there is no

[9] IP: Internet Protocol - A widely used protocol to address computers and send data over a network.
[10] UDP: User Datagram Protocol - A protocol to exchange data between computers connected through a network.
[11] Length of a maximum sized packet (1500 byte) at 1 GBit/s.

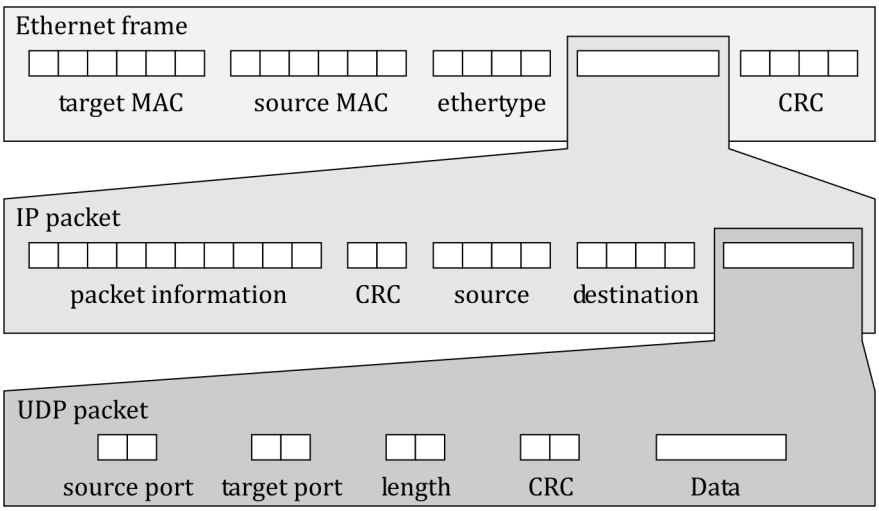

Figure 3.2: Ethernet frame transporting an IP header followed by an UDP packet. Each small box marks one byte of data.

handshake implemented. Therefore UDP is not suitable for the needs of HADES, and much more complex transaction based protocols like TCP[12] [Pos81b] would have to be applied in addition.

Another network solution frequently used for internal communications in high speed computer clusters is Myrinet, a proprietary network setup developed by Myricom.

The Myrinet-2000 specification [Myr05] foresees the usage of 2 GBit/s optical links with a very small protocol overhead of less than 2 per cent

Even though Myrinet would fit to most of our performance requirements, it is an expensive solution compared to others. The devices provided by Myricom must be used and the design of customized hardware is difficult, in particular on the front-end electronics where the boards have to be very small.

As a conclusion it can be stated that there are many different network implementations available on the market, but none of these fits completely to the requirements of HADES.

[12]TCP: Transmission Control Protocol - One of the main protocols used for internet, including handshakes to verify that all data has been received.

4 Network Concept and Data Format

Since none of the available network implementations is suitable for the HADES network, a new network protocol had to be developed. It will be mainly used to connect the universal trigger and readout boards (TRB) and therefore it is called TrbNet. In this chapter, an overview of the features of the protocol and its data format will be given.

4.1 Generic Network Layer Model

In the late 1970s the "Open Systems Interconnection" subcommittee of the International Organization for Standardization (ISO) developed a model for describing network architectures, the "OSI Reference Model" (OSI) [Zim80], as shown in figure 4.1.

This model defines seven layers to describe all main tasks to be done when connecting to different applications over an inhomogeneous network. The descriptions are therefore very abstract to be applicable to most network implementations.

The layers can be roughly divided into three application and four transport layers. As their names indicate, the transport layers provide mechanisms to control the transport of data, whereas the application layers provide the interface for sending data.

The uppermost layer is the application layer that contains the interface for the user of the network. This is the only part of the network stack the user has to deal with. All other layers are hidden from the users points of view.

The sixth layer, as counted from the lowest, is the presentation layer. It decodes the received data, that has been encoded in a universal code for the network, to make it readable by the application again.

The session layer controls the data exchanged with another network member. It builds up connections to the receiver, manages data flow from and to this point, and closes the connection at the end of the transaction.

The fourth layer is the transport layer. As it is the highest layer of the transport section of this model, it marks the border to the actual network. Managing the network resources and optimizing the data to be sent effectively are its main purposes.

The network layer is the transport's counterpart to the session layer for the application. It manages the connection between two data transport interfaces independently from the data itself.

Finally, the data link layer controls and maintains the physical network connection, provided by the physical layer, that is used to send the data.

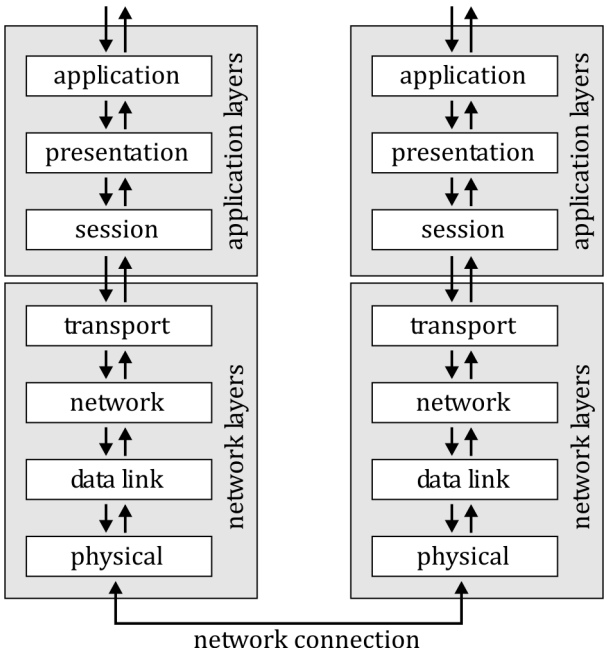

Figure 4.1: OSI layer model for network interfaces. It contains seven layers that provide all functions needed to connect different computer hardware with one network.

4.2 TrbNet Layer Model

For the TrbNet a simplified picture (see figure 4.2) that consists of four layers is used: The application layer, followed by session layer, link layer and the physical layer.

The session layer provides an interface for the application to send and receive data. It is responsible for an ordered data flow by controlling when and how data can be sent or received by the application. By this means, it hides the network from the application and prohibits the application from sending data when it is not allowed to. Additionally the session layer cares about addressing the data to the desired receiver[13].

The link layer arranges the data for being sent over the network connection. The main concern for this layer is the integrity of the data. Hence it contains mechanism to protect from data loss and corruption.

Finally, the physical layer provides the necessary logic to adapt the data to any available type of data connection to other network nodes.

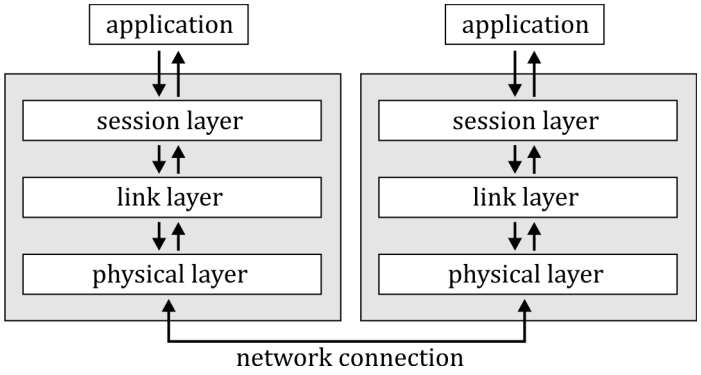

Figure 4.2: Layer model for TrbNet. Compared to the OSI layer model, a reduced architecture with four layers is used.

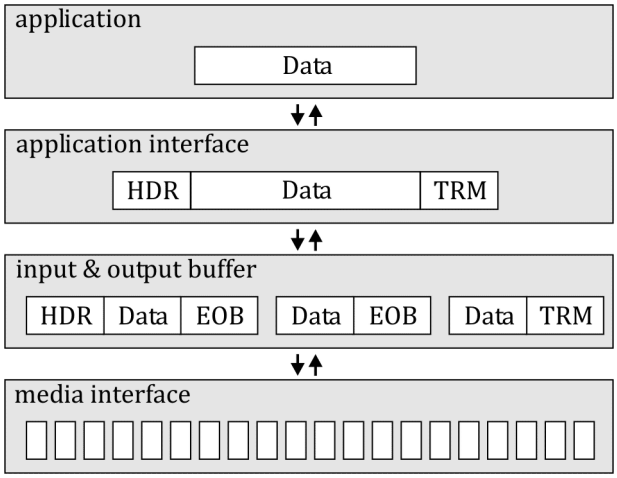

Figure 4.3: Dataflow through the network layers. A header and termination is added to the data from the application by the application interface. The I/O buffer organizes the data into buffers and adds handshake packets. The media interface prepares the data for sending over the data link.

4.3 Protocol Structure

The dataflow inside the network endpoints from the application to the network media and vice versa is depicted in figure 4.3. The data to be transmitted by an application via the network is first transported to the application interface (API), where a header and a termination are attached. Here, the header contains information like the address of the target endpoint while the termination carries various status informations (see table B.1).

In the link layer, a so-called output buffer (OBuf) divides the data into small bunches called "buffer" and special information is added for a handshake with the receiver as well as for error detection. The counterpart of the output buffer is the "input buffer" (IBuf) where the data is checked for validity. This process is described in detail in section 4.6.

The media interface provides the connection between the internal logic and any network hardware. E.g., the data coming from the internal logic is converted to a serial stream for an optical link or is sent over a parallel connection that runs at a different clock speed.

In the TrbNet a clear distinction is made between initial transfers and those that are sent as a reply. An initial transfer can only be made by a so-called "active application", while passive applications can only react on a transfer initiated by a different node.

In order to resemble the old busy logic all receivers have to reply to every initial transfer. The network remains locked until all replies have been collected by the original transmitter. Prior to that, no other initial transfers are allowed[14]. This implies that each session consists of two independent transfers, each starting with an header and finishing with a termination.

As already pointed out, the termination from the endpoints is used as a replacement for the busy logic of the current network setup: All receivers of the trigger must not send the termination until the processing of the event is finished. Consequently, the central trigger system is able to stop sending new triggers by reading the busy information via its API.

4.4 Priority Channels

The challenge for the design of TrbNet is that different demands on the data transport have to be handled. Slow control packets have the lowest requirements. They consist of small amounts of data and delivery is not time critical. Triggers are small packets as well, but have to be distributed over the whole network rapidly. The transport of detector data needs a large bandwidth but has no special timing constraints.

To take these different requirements into account, the network is divided into so-called channels. Each channel is assigned a priority based on its number, the lowest channel number has the highest priority. As explained above, the LVL1-trigger channel has to get the highest priority while the slow control takes the channel with the lowest priority.

[13] Network Addresses are described in section 4.8.

[14] N.B., that the network is not blocked completely since it is divided into independent channels as described in section 4.4.

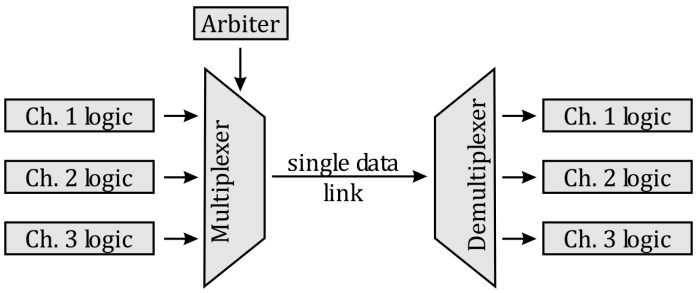

Figure 4.4: Data from the different channels is merged to be transported over a single data link and demultiplexed as it is received from the data link.

To allow the immediate transfer of LVL1-triggers, it must be possible to stall any other transfer, send this trigger and then resume the original transfer without any overhead or lost data.

Therefore, the data stream on the physical layer is demultiplexed into the different channels directly after the receiver (see figure 4.4). Inside the network node each channel uses distinct hardware resources. That allows to handle multiple transfers in parallel without interrupting each other.

In the other direction, towards the network, a multiplexer merges all the data from the different channels. This unit takes priorities into account: the included priority arbiter serves the channel with the highest priority first while all other channels have to wait for it to finish.

In addition, a round robin arbiter ensures that a high priority channel cannot lock the network by continuous data sending. Here, high priority transfers are interrupted from time to time to transmit low priority data as well.

To clearly distinguish between initial transfers and such that are sent as a response, the channels are further divided into an inital and a reply path.

4.5 Data Packet Format

To reach the required very low latencies for high priority packets, a high granularity of the data stream is necessary. Therefore, the data is divided into fixed sized packets. The multiplexer has to be able to switch channels after each packet, thus they have to contain the respective channel number. On the receiver side the demultiplexing of the packets can be done by extracting the channel number from each packet.

Since packets are the only allowed unit to be sent, special words like header or termination have to fit into one packet as well. To distinguish between them, up to eight packet types are defined. All packets have the same structure that is shown in table 4.1. However, the meaning of the data words which are named F1 to F3 depends on the packet type (see table B.4).

The packets have a fixed size to avoid additional overhead for the detection of the start of

BITS	DESCRIPTION
63 - 56	reserved[a]
55 - 52	Channel number (see section 4.4)
51	Init / Reply path (see section 4.4)
50 - 48	Packet type (see appendix B)
47 - 32	Data word F1
31 - 16	Data word F2
15 - 0	Data word F3

Table 4.1: The 64 bit packet format

[a] For example redundant bits on media interfaces

each packet. The size has been chosen carefully: If the packet size is too large, the latency for sending high priority packets is increasing. For small packet sizes the latency is very low but the ratio between overhead for packet type and channel number and payload gets worse.

The size of a packet was therefore set to be 64 bit, allowing 48 bit of data to be transported with 16 bit of overhead.

4.6 Data Buffers

As described in the section 3.2, each receiver has to be able to accept all transmitted data to avoid its loss. For that reason an input buffer is used that provides a FIFO[15] to store the data until it is further processed. To protect against buffer overflows, a handshake protocol is implemented to inform the sender after the buffer has been cleared. This information exchanged between both IOBufs is shown in figure 4.5 and the timing of the packets is depicted in figure 4.6. In addition, the size of this buffer is negotiated over the network.

The handshake consists of two special types of packets that are embedded into the data stream on the link layer: End of Buffer (EOB) and Acknowledge (ACK). After the transmission of a complete buffer the OBuf attaches an EOB. Consequently, the OBuf must stop transmitting data. On the receiver side, the data is read from the input buffer by the internal logic (e.g. an API or a hub). Here, when this last word has been read from the IBuf, it forces its OBuf via control lines to send an acknowledge. After the acknowledge has been received, the transmitter is allowed to continue sending data. This behavior is sketched in figures 4.5 and 4.6.

The EOB packet includes the number of words that have been sent while the ACK contains the actual size of the FIFO which may vary from implementation to implementation (see table B.4 for details). It should be pointed out, that these two types of words are only needed on

[15] First In First Out. A memory to temporarily store data which is read out in the same order as it was written to the memory before.

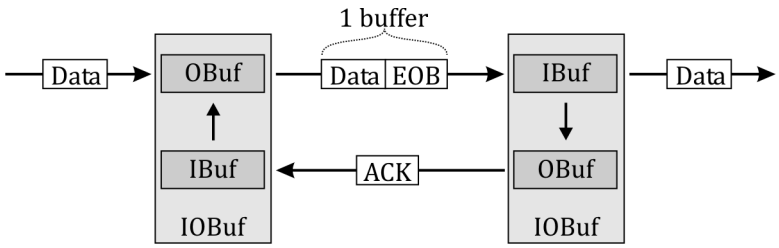

Figure 4.5: Handshake between two IOBufs transporting one buffer of data. Data is sent followed by an EOB packet. The receivers OBuf answers with an acknowledge, allowing the sender to send the next buffer of data.

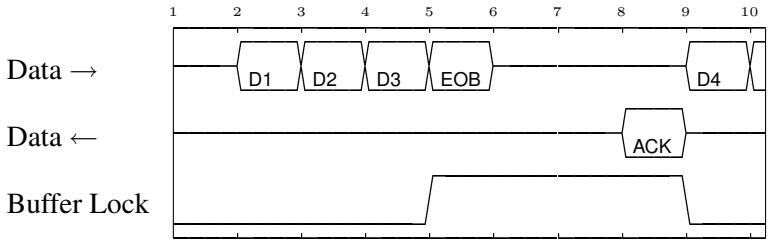

Figure 4.6: Handshake between two IOBufs. After sending one buffer of data (here: 3 packets of data), a EOB is added and the transmission is stopped until an acknowledge has been received.

the link layer and are not passed on to the application interface.

In order to increase the bandwidth, the sender is allowed to send two buffers of data before it has to wait for an acknowledge. Accordingly, the receivers FIFO must have twice the size of one buffer. This allows the transmitter to continue sending data while waiting for the acknowledge of the first buffer.

With our standard medium, an optical link running at 2 GBit/s and a FIFO size of 18 kB, this allows a contiguous data flow from one point to the other.

Additionally to the handshake mentioned above, the integrity of data is controlled by a checksum: Each EOB contains an checksum of all data that belongs to that buffer. As a checksum algorithm a 16 bit Cyclic Redundancy Check (CRC) is used.

From the mathematical point of view, a CRC [PB61] is calculated by treating the data stream and a defined generator vector as a polynomial. The transmitted checksum is then the remainder of the polynomial division of both polynomials. The hardware can perform this operation by simple bit-wise exclusive-or operations on every data word.

Hence, it does not need significantly more resources than a simple sum of all data words, but is capable to detect any odd number of bit errors and most errors with an even number of bit flips[16]. The input buffer checks the CRC by recalculating it. If a mismatch between both sums is detected, the appropriate bit in the termination packet (see table 6.1) is set to mark this transfer as corrupted.

4.7 Hubs and Data Concentrating

The IOBufs themselves are responsible to establish the point-to-point connections. For a network with dozens of nodes a special entity is needed that transports data between several endpoints. Such a device is usually called “hub”.

Due to the required locking behavior of the TrbNet channels, it is a common case that multiple endpoints send answers to a request. Hence, the hub includes a special functionality to merge all data from the different endpoints into one data stream before it is passed on.

The termination packet from every endpoint is handled separately from the data. It is read from all points and stored temporarily instead of forwarding it. The information from all terminations is merged into one termination packet using a logical or-function for every bit of the error pattern. This merged termination is then sent after all endpoints have terminated their transfer.

If the locking behavior is not used, another hub behavior can be implemented that does not wait for terminations from all endpoints and allows every point to send data at any time without using merged replies. In this case, the reply path is not used and only initial transfers are possible. This mode of operation could be very helpful for detector read-out systems where

[16]The implementation uses the IBM-CRC-16 generator polynomial $x^{16}+x^{15}+x^2+1$.

a trigger-less push architecture is used like it is planned for PANDA and the CBM-MVD[17] [S+07].

4.8 Network Addresses

Every network node needs to be addressable individually in order to be reached by messages on the network, which is needed during the configuration process. Moreover, the data has to contain an identifier of its source. All network nodes are therefore assigned to a unique 16 bit address.

The available address space is divided into three sections: Addresses from 0000 to EFFF are used as regular addresses in the final network setup. Addresses F000 to FEFF are to be used in test setups and must not be used in the detector network to avoid any ambiguous addresses during tests.

The remaining addresses from FF00 to FFFF are used as broadcast addresses. The highest address, with all bits set to one, is the widest broadcast that reaches every board.

For all other broadcasts, the last eight bits are used as a bitmask. The configuration of the endpoints allows to mark single bits as “do not care”-values. A broadcast is then accepted by these endpoints, even if the corresponding bit in the broadcast address is not set. Using an eight bit bitmask results in eight separate usable broadcast addresses. If necessary, this bitmask can be enlarged to 12 or more bits by the disadvantage of loosing addresses for network endpoints.

For larger networks, the available address space can be extended: The header contains two reserved bits to built up subnets. On the whole, with four subnet levels this allows for up to 2^{64} individually addressable nodes.

[17] MVD: Micro Vertex Detector. A silicon pixel detector for measuring the reaction vertex in the CBM experiment.

5 Implementation

After the network protocol has been specified its implementation in the hardware units has to be described. Starting with a short remark about the used programming language, this chapter is dedicated to this implementation which embodies the internal structure of the network layers mentioned before.

5.1 VHDL

Since the network protocol is implemented in FPGAs rather than off-the-shelf computers, a programming language that allows to describe basic logic functions has to be used. For this purpose two languages are available: Verilog and VHDL[18]. Here, VHDL has been chosen as it is more commonly used[19] than Verilog.

Logic developed in VHDL is organized in so called entities. In principle an entity is one logic unit, comparable to one device or one group of devices on a printed-circuit board. Like discrete devices, each entity has so-called "ports" to connect to other entities. In addition, configuration options can be used to change the structure of the entity.

Like it is done in any high-level programming language, the code has to be compiled; for VHDL this process is called synthesizing. During this process the described logic is mapped to the available FPGA resources and a programming file is generated that can be loaded into the FPGA.

These processes have to be done using special tools provided by the vendor of the devices. Since the code has to be compilable by different tools special effort has to be made to ensure portability of the code.

5.2 Data Ports

The overall design consists of different entities between which data is transmitted (see figure 5.1). These entities have to be exchangeable and combined in different configurations to adapt to different applications. Therefore, a standardized data interface is used. As outlined in table 5.1 each data interface consists of four ports. Two of them are transporting data and two are used for control signals.

[18] VHDL: Very high speed integrated circuit Hardware Description Language.
[19] In Europe mostly VHDL is used, while in the U.S. Verilog is preferred.

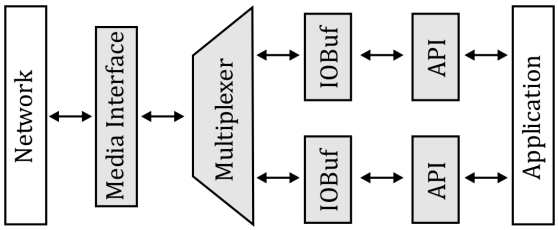

Figure 5.1: The main entities of the TRBnet logic. Here, a setup with two independent channels is shown.

NAME	DIRECTION	DESCRIPTION
Data	$\longrightarrow$	16 bit data vector
Packet Number	$\longrightarrow$	2 bit packet number vector
Dataready	$\longrightarrow$	Data is valid
Read	$\longleftarrow$	Receiver is reading the data

Table 5.1: The standard data port. Two ports with 18 bits in total are used for data, while two signals are used for a handshake.

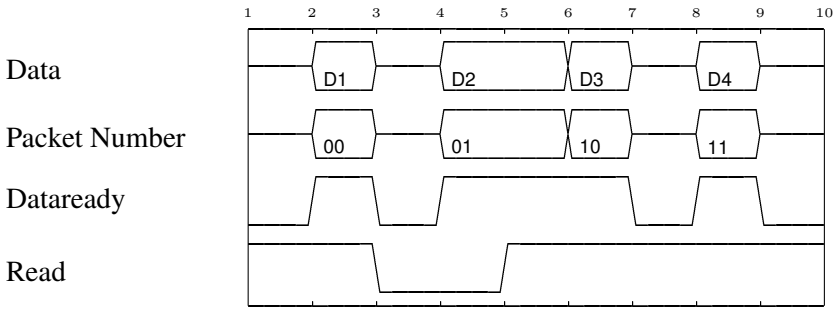

Figure 5.2: A typical transaction on a data interface. Data is read by the receiver under the condition that read and dataready signals are both enabled in the same clock cycle.

The packet size of TrbNet is 64 bit as defined in section 4.5. Internally, these packets are divided into four 16 bit words. There are two reasons for this splitting: one is that 64 bit wide data paths are to big to be handled efficiently. E.g., internal memory blocks can only handle 32 bit wide input and output vectors.

The other reason is that 16 bit words are compatible to most other hardware devices such as the optical transceivers. The 16 bit can easily be reorganized to interfaces that use 8 bit or 32 bit words.

There is no loss in bandwidth introduced by the splitting: The optical links work at 1.6 Gbit/s. With a FPGA clock frequency of 100 MHz this sums up to 16 bit per clock cycle, which exactly matches the chosen word size.

To mark which part of the packet is currently transmitted, two more bits are added to the port. The highest bits of each packet contain the most important information, namely the channel number and packet type, thus the highest 16 bit word is transmitted first.

On standard memory interfaces the read control signal is used in such a way that data is delivered one clock cycle after the read signal was set. Due to the low latency requirements, a free running handshake is used instead. This means, a data word is read in the same clock cycle as the read signal is set (see figure 5.2). Using this type of handshake eliminates the need for one waitstate inserted after setting the read signal before valid data is available.

5.3 Secure Buffers

With the free running handshake the time before offered data can be read is reduced by one clock cycle on one hand, but also introduces the need for a special way to register signals.

In most other cases it is sufficient to place a simple flipflop to synchronize a signal to the clock. Since the read signal may be deasserted at any time, the transmitter cannot rely on that the processed data is read in the subsequent clock cycle. Hence, the decision whether it can accept new data on its inputs or not in the next clock cycle cannot be done in time (Note that all outputs of each entity have to be registered, i.e. put through a flipflop). This will roughly be explained with the following example: The multiplexer generates a read signal to allow the output buffer to send data. If another higher priority output buffer has to send data, the multiplexer has to withdraw the read signal and allow the other buffer to send data.

There are several possibilities to avoid this possible data loss: The first is to use pure combinatorial outputs. However, this has big disadvantages: It increases the length of data paths between two flipflops and therefore reduces the maximum clock frequency.

The second possibility is to use a single register but allow to write data to this flipflop in every second clock cycle only. This additional clock cycle allows the receiver of the data to securely read out the data before it can be overwritten.The data bandwidth this type of buffer can handle is only half the data rate of the full version.

To be able to register signals nonetheless a secured buffer, the SBuf, is used. It provides a

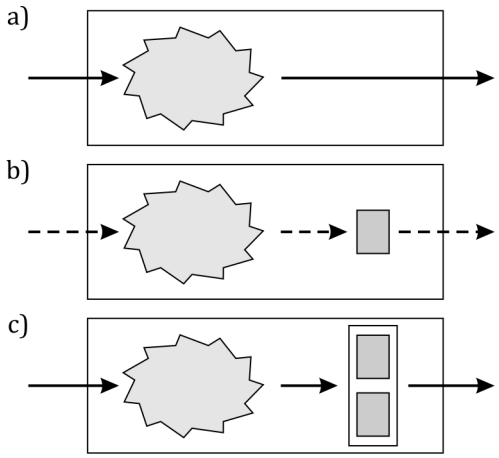

Figure 5.3: Secure Buffers provide three different kinds of data output for every entity: a) The combinatorial signal is directly forwarded to the next entity. b) A single register is used, but forces the entity to accept new data in every second clock cycle only. c) A full secure buffer provides two separate register.

combinatorial input and a registered output. In the case the subsequent receiver is not able to read the offered data immediately, a second register stage can be used to buffer one more data word. The buffer is now able to read data in all cases unless the second buffer stage is filled and the receiver is not reading. This gives the opportunity to generate a combinatorial read signal one clock cycle in advance: The buffer will be able to read data in the next clock cycle if the first buffer is empty or it is filled and the receiver reads the data. This read signal provides a reliable read signal for the entities inputs.

A schematic view of all three possible implementations is shown in figure 5.3. Due to the generic design it can be chosen individually for each buffer instance which of these possibilities is employed.

5.4 Media and Platform Independency

TrbNet should be able to run on nearly any kind of hardware. The implementation is therefore highly modular and configurable.

For each endpoint there are a lot of configuration options that can be changed without touching the core design. Hence, the logic resources in small devices might be reduced on the cost of bandwidth and / or features.

The code has been divided into two parts: A core design and hardware dependent parts. All code parts that rely on the hardware setup (Such as the readout of a specific temperature sensor e.g.) are easily exchangeable.

FPGA specific components are not used unless absolutely necessary. One exception are

VALUE	NAME	DESCRIPTION
0	ERROR OK	Transport link OK and running
1	ERROR ENCOD	Transmission Error, corrected by encoding
3	ERROR FATAL	Fatal error, link must be restarted
6	ERROR WAIT	Link is starting up, waiting for begin transmissions
7	ERROR NC	Media not connected

Table 5.2: Error states of the media interface. The names are defined as constants in the *trb_net_std* VHDL library.

FIFOs, for which automatic design tools are employed by default. Therefore, to adapt these entities to a new device it is necessary to generate the FIFOs specifically for this device.

5.5 Media Interfaces

The media interfaces provide the fundamental functions to transport data from point to point. The media interface can easily be exchanged, which makes TrbNet adaptable to very different hardware.

5.5.1 Common Media Interface Features

In addition to data transportation each media interface provides two things: The network needs an alignment to 64 bit boundaries in spite of the fact that smaller words are used to transport the data. Therefore a counter was included that assigns a number to each data word (see sec. 5.2). On the receivers side, the counter needs the possibility to resynchronize with the sender if one data word was lost.

This malfunction should be avoided as it leads to a lockup of the network. However, it is recommended that a media interface provides a method to send a global reset over the network. Since in this case the transport layer is stopped, a low-level signal has to be used.

The actual implementation of such a reset signal depends on the medium. On some types of interfaces an additional wire can be used. On serial connections like the optical link additional lines are not available. Here, a special bit sequence is sent that will never occur during normal network operation and thus can easily be recognized by every receiver.

The current status of the link is represented on a special port as described in table 5.2.

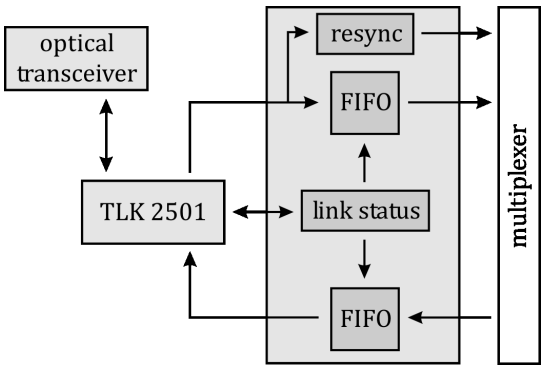

Figure 5.4: Implementation of the optical link interface using external TLK 2501 transceiver chip. The main purpose of the two FIFOs is to synchronize data flow between different clock domains. Additional logic provides the required features.

5.5.2 Media Interface: Optical Link

Synchronizing dataflow from and to the external TLK chip is not straight forward. In sum there are three different clocks involved: The internal clock of the FPGA, the recovered clock that is received over the optical link, as well as the transmission clock that is locally generated by an on-board oscillator. Hence, one dual ported FIFO is needed for each data direction to transport the data from one clock domain to the other (see figure 5.4).

The start up sequence of a connection should be reliable in all cases, whether one of the transceivers is reprogrammed, switched on or the cable is plugged in. When the transceiver starts to receive valid data, a counter is started. After 1.3 seconds[20] receiving data is enabled. After another 650 microseconds[21] the transmitter is enabled as well. Both time intervals are chosen comparably long even though the transceivers themselves are able to lock to a signal in less than one microsecond. The reason is that connecting a cable by hand takes some time until both optical fibers are connected accurately.

The required resynchronization is provided by sending four times the value 007F. In terms of the network protocol this is an illegal packet in one of the unused channels and is therefore not used during normal operation. After the fourth reception of 007F the packet counter is reset to zero, both FIFOs are cleared and the corresponding status bit is set. The transmission of the resynchronization sequence to the original sender of the sequence is triggered by the rising edge of the equivalent control bit.

NAME	DESCRIPTION
DATA	8 bit wide data lines
CARRIER	Asserted when valid data is offered on the link
PARITY	An even parity bit used to check integrity of data
CLOCK	Transfer clock
FIRST PACKET	Asserted when the first 16 bit word of a packet is sent
READY	Ready signal of the receiver. Used on high speed links only

Table 5.3: Connections used for a LVDS media interface

5.5.3 Media Interface: LVDS

A parallel LVDS[22] connection is used to transport data between a TRB and its AddOn board using the AddOn connector. The connection between a TRB with a General Purpose AddOn and an Acromag DX2004 module[23] uses this interface type as well.

The link uses eight data lines and four, respectively five, control lines as outlined in table 5.3. The transmission is done using a parallel, synchronous mode: The clock is transported along with the data.

There are two versions of the interface, a slow and a fast one. The slow one uses oversampling to detect edges in the clock signal while the fast one uses the received clock directly to recover the data. According to the Shannon-Nyquist theorem, the slow transmission clock must have less than half the frequency of the sampling clock, which is the internal FPGA clock. Therefore, the maximum frequency of the slow interface is 50 MHz for boards using a clock of 100 MHz.[24] Because of small variances[25] within both clocks, always one of these clocks is slower than the other, even when running at the same nominal speed. This limits the achievable transmission frequency to a value lower than 50 MHz. In the standard setup a 25 MHz transmission clock is used, because this frequency can easily be generated out of the 100 MHz internal clock. This limits the data bandwidth of the interface to 200 MBit/s.

The fast implementation directly feeds the received clock into a clocking network of the FPGA, which allows the clock to be used to directly drive flip-flops. Data is transported using a dual data rate (DDR) mode at 100 MHz. The bandwidth is then 1.6 GBit/s, the same as for the optical link.

Due to limitations of the FPGAs, the fast mode can only be used if one of the LVDS lines

[20] 1.3 s correspond to an overflow of a 27 bit counter if the device is running at 100 MHz.
[21] The overflow of a 26 bit counter.
[22] LVDS: Low Voltage Differential Signal. Every signal is transported differentially using two wires, one with an inverted signal, making the transmission more resistant against electromagnetic influences from the outside.
[23] A PCI card for VME CPUs featuring a Virtex 2 FPGA and 32 differential I/Os.
[24] Actually, the Acromag module is running with 99 MHz, since the on-board oscillator has a frequency of 33 MHz.
[25] In the order of 100ppm.

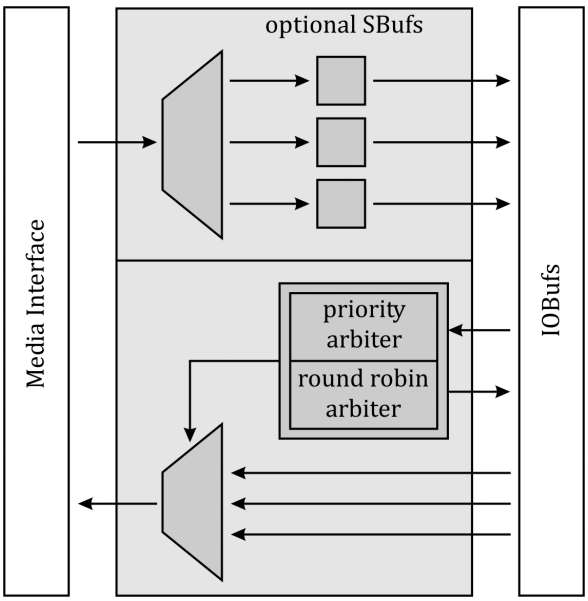

Figure 5.5: Implementation of the data multiplexer. Incoming data is divided into the different channels. Two types of arbiters control which channel is allowed to send data to the medium regarding channel priorities.

is connected to a clock input of the FPGA[26]. Otherwise the delay until the clock reaches the flip-flops is too high. A future implementation might use automatic alignment of the clock to make this mode available to interfaces without an dedicated clock input as well.

5.6 Multiplexer

The multiplexer merges the data input from the IOBufs of the different channels to one data stream that can be sent over the network using the media interface. Thus a priority control has to be implemented as discussed in section 4.4 and shown in figure 5.5.

The priority arbiter decides which channel will be served next by examining the incoming dataready signals: The channel with the highest priority that has data available is allowed to send data. Additional logic prevents the switching of channels during the transport of one packet and enables the priority arbiter only after four words from one channel, forming one packet, have been transported through the multiplexer.

A round robin arbiter allows to serve channels without respect to priorities by switching from one to the next active channel after every packet. The ratio of the round robin arbiter versus the priority arbiter can be configured.

The demultiplexer distributes the incoming data to the input buffers. It reads the first word

[26]This is the case for the AddOn connector, but not for the connection on the Acromag module.

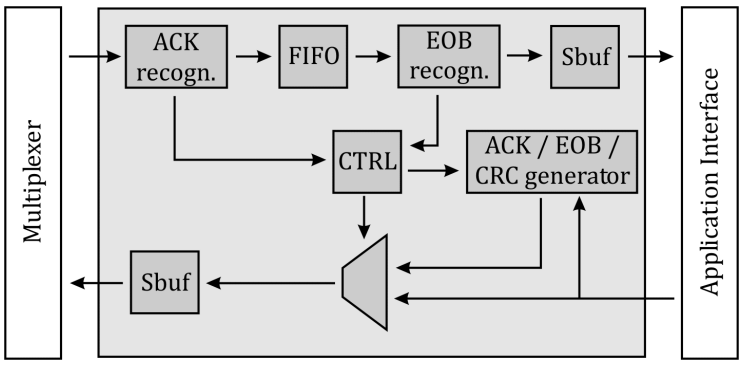

Figure 5.6: Implementation of the IOBuf. Incoming data is checked for handshake packets, stored in a FIFO and send to the application interface. Outgoing data is sent regarding the handshake protocol.

VALUE	BUFFER DEPTH	REAL DEPTH
0	no FIFO / no data can be received	
1	2	4
2	4	8
3	8	16
6	127	255
7	infinity	

Table 5.4: Sizes of buffers and real FIFO depths, given as the number of 64 bit packets that can be stored. Values of up to 3 are implemented in Lookup-tables used as RAMs, while for size 6 internal memory blocks are employed.

of each data packet, interprets the channel number and sets the dataready signals accordingly. SBufs may be used on every output. Since this would require eight SBufs, one for each channel and every path, the logic utilization is much higher. The standard configuration does not use SBufs which is sufficient because the input buffer by definition must always be able to accept data.

5.7 Input and Output Buffer

The IOBuf manages the handshake protocol and generates checksums to assure data integrity. As already outlined in section 4.6, the IOBuf is internally divided into two parts: The IBuf and the OBuf. The main task of the IBuf is to provide a buffer for the data and pass it on to the application interface or the hub logic. The OBuf manages the data flow from the API to the

network (see figure 5.6).

Data forwarded by the multiplexer is checked first whether it is an acknowledge. In such a case, this information is provided to the OBuf whereas all other data is written into a FIFO. When the data is read from the FIFO by the application interface, all end of buffer packets are recognized and cleared, while all remaining packet types are transported further on.

The OBuf provides the second part of the functionality. It sends the data from the application interface to the network. EOB and ACK packages are generated and inserted into the data stream whenever necessary. In the case that the buffer of the IBuf on the receiver side is full (recognized by the OBuf by counting sent EOB and received ACK), data transmission is stopped.

To check the data integrity, both IBuf and OBuf contain a checksum generator. The checksum calculated by the OBuf is transported in each EOB packet. The IBuf calculates the checksum of the received data and is therefore able to validate the received data. If the received checksum does not match the calculated one, a bit in the termination is set to mark the transfer as corrupted.

The size of the input buffer is encoded in 3 bits as shown in table 5.4. A depth of 7, which means infinite buffer size, can of course not be implemented in hardware. It is set for devices that are always able to read data and therefore don't need a buffer. Another option is to send this buffer size as an acknowledge to a transfer that is discarded and hence allowing the sender to omit all EOB packets until the end of this transfer.

5.8 Application Interface

The application interface (API) connects on one side to the IOBuf and on the other side to the users application (see figure 5.7). It encapsulates the network protocol so that its details are hidden from the application.

When the API receives data it first checks in the header, if the target address either matches its own address or is a broadcast[27]. If this is not the case, it clears the data and replies with a short transfer[28]. Otherwise it writes the data into a FIFO from which the application can read the data.

For the data in- and output, standard data interfaces are used. Additional signals are used to control the transfers as shown in figure 5.8 and described in appendix D.

The API can be of two types: An active one that is allowed to initiate transfers, and a passive version that can transfer data only as a reply to incoming requests. The API includes a state machine to control this behaviour according to the selected API type. In the HADES case most of the network nodes will only have a passive API. Only a few central systems are allowed to actively send data.

[27] Broadcasts are accepted according to the bitmask set with BROADCAST_BITMASK as described in section 5.13.
[28] A short transfer is a termination only, without header and/or data packets.

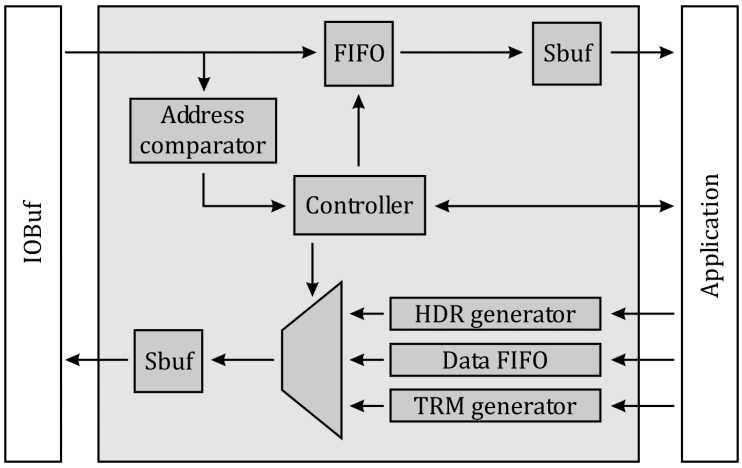

Figure 5.7: Implementation of the application interface (API). If an incoming header contains the applications address, the data is forwarded to a FIFO from which the application can fetch the data. The application controls outgoing transfers with several signals. The API generates header and termination packets for all sessions and sends the data received from the application to the output buffer.

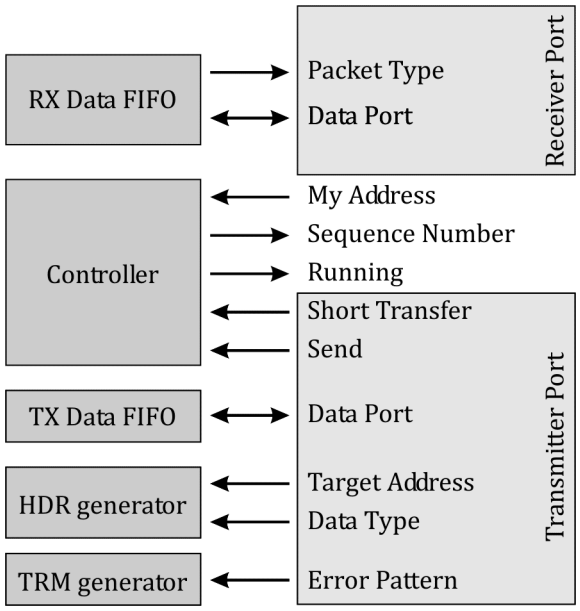

Figure 5.8: Ports of the application interface. From the applications point of view, the application interface acts like two FIFOs with additional control lines. The data ports include handshake signals as described in sec. 5.2.

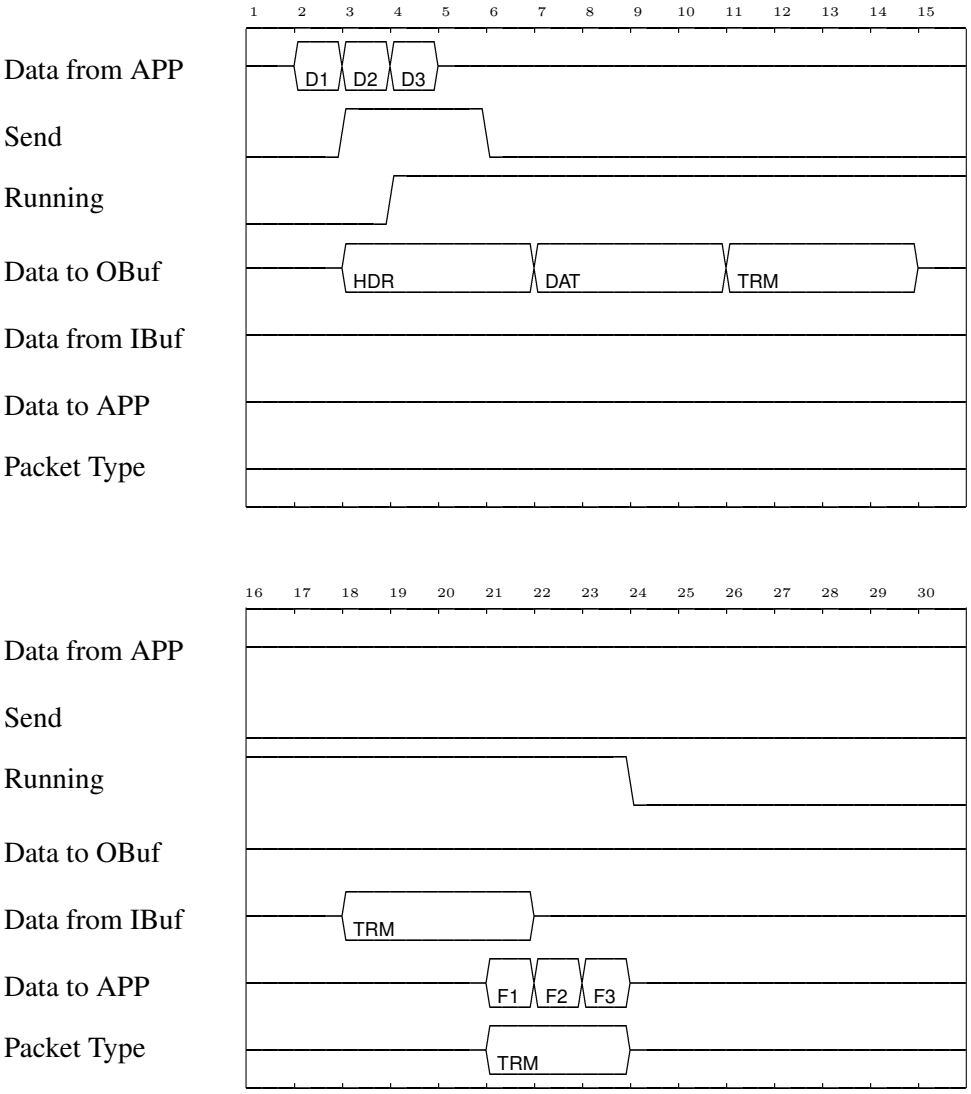

Figure 5.9: A typical transfer done by an application. In the upper part the application writes data into the FIFO, then raises the send signal that causes the API to form a header, send the data and termination to the OBuf. Later, shown in the lower part, the reply is received through the IBuf and passed on to the application. After the last word has been read, the API indicates that the channel is free for the next session by releasing the running signal. The ports for target address, data type and error pattern have to be supplied as well but are not shown here.

When the application makes a transfer, it has two options: sending real data or a short transfer only.

If the application has to send data, it is filled into the API's FIFO using the data interface. The "send" signal can be raised at any time and causes the API to start the transfer. Consequently, a header with the provided target address is formed and then data transmission is started. The data has to be provided along with a packet counter, marking the four words of each packet. Since the first word of each packet is not generated by the application but by the API, the counter has to start with one, then count up to three and start again with one.

If a short transfer should be executed, the "short transfer" signal of the application interface has to be enabled without writing data to the FIFO. Due to the lack of an header with addresses, a short transfer on the init channel is always a broadcast to all devices. On the reply channel, the short transfer is as usually only forwarded to the node that initiated the transfer.

After all data has been written to the FIFO, the "send" signal can be released. The API then keeps on sending all data remaining in the FIFO and finishes the transfer with a termination, including the provided error pattern, afterwards.

When data is received by the API, it can be read out by the application using a standard data port. The API can be configured[29] to either send all four words of each packet to the application or to forward only the three data words and provide the current packet type on a separated port.

Figure 5.9 shows a typical transfer where one packet of data is sent, and a short transfer is received.

5.9 Hubs

The hub is one of the most versatile parts of TrbNet, which can be used in two cases: It can be implemented on a dedicated hardware hub or it can be used to distribute data internally on any board. An example is the case that only the TRB is connected to the optical network but not its AddOn board. If both FPGAs contain an application, a three ported hub is needed on the TRB with one connection to the AddOn, one to the optical network and one to the internal endpoint. The only difference between both implementations is the integration of application interfaces into the hub.

The hub therefore has a vast number of configuration options. Despite the number of media interfaces, the parameters for all IOBufs can be set and the number and channel of optional application interfaces can be defined. Refer to table D.4 for a complete list of options.

5.9.1 Basic Hub Setup

The ports of the hub are connected to media interfaces. All required entities are instantiated inside: On each input a multiplexer and an IOBuf as in every other network node is employed.

[29] By setting APL WRITE 4 PACKETS to yes respectively no.

Their outputs are connected to the main hub logic as shown in figure 5.10. Optional application interfaces can be connected directly to the central logic as well. They provide access to the network for applications that reside in the same hardware device without using additional buffers.

A so-called RegIO component (described in section 5.11) is connected to the hub to control all media interfaces and to readout the current status of all channels. In the case of a locked network channel, one can easily locate the source of the malfunction using these registers.

The central hub logic is needed once per data channel and is the main entity for data transfer and merging of terminations (see figure 5.11. In other networks like Ethernet usually switches are used that are able to route data from one port to another without involving the remaining ports. In this first hub version the more elementary concept of a hub is used: If data is received on one port, it is distributed to all other ports.

According to the specification of the network protocol, the data on each channel is further divided into an initial (init) and a reply channel (see section 4.4). Data on the init channel needs to be sent to all ports of a hub, since no address based routing is done. When data arrives on the reply channel the data flow can be optimized: Since the source port of the init transfer has been stored before, the data on the reply channel needs to be sent to this one point only.

This semi-routing implies that data sent as a reply needs fewer network resources than data on the init channel. Hence, big data transfers should not be sent actively, but should be requested by the receiver of the data.

5.9.2 Merging of Data Streams

The reply arbiter is selecting the interface from which data is read on the reply channel. There are two possible working modes. The simplest way is to read data from one endpoint until it terminates the transfer, move to the next one and so on until all points have been read. However, if longer breaks in the data transfer from one point occur or the corresponding media interface has a lower speed than others, bandwidth remains unused.

Therefore, the hub is also able to switch between reply ports in the middle of a data transfer as shown in figure 5.12. The result is a better bandwidth usage in case of media with different speeds but also special effort must be made to mark every packet with the matching sender address. This is done by storing the transfer header from each interface and resend it upon a change from one interface to the other.

In network setups with more than one active endpoint per channel, data collisions may occur even though channels are locked during a transfer. For example this can happen if two endpoints start a transfer at the same time. Therefore, a future option is to resolve possible data collisions. In this case, one of the transfers must be interrupted by either deleting its data or storing it in an internal memory to pass it on later. If such an error occurs, it will be marked by setting the appropriate bit in the termination packet. The initiator of this transfer then decides whether, if possible, to resend the data. For the current HADES network setup plans this is not

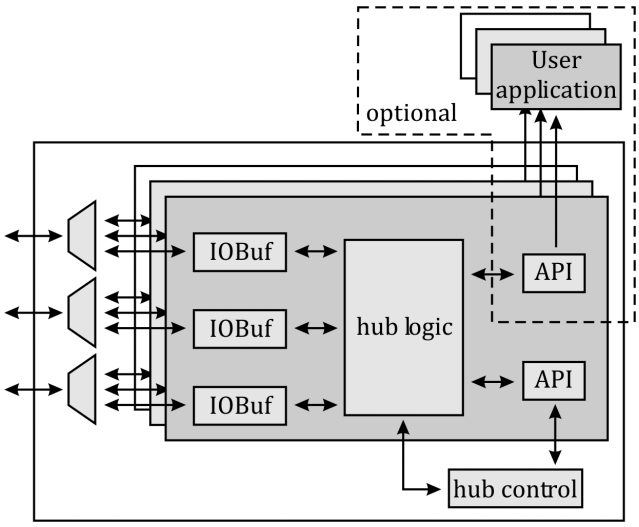

Figure 5.10: Implementation of the hub. The data received from the connected media interfaces is splitted by the hub into the different channels. It is then transported by the central hub logic to one or all other media interfaces. The hub logic can not only be connected to media interfaces using IOBufs, but also to application interfaces. These are implemented to connect a control application as well as any other kind of aplication to the hub.

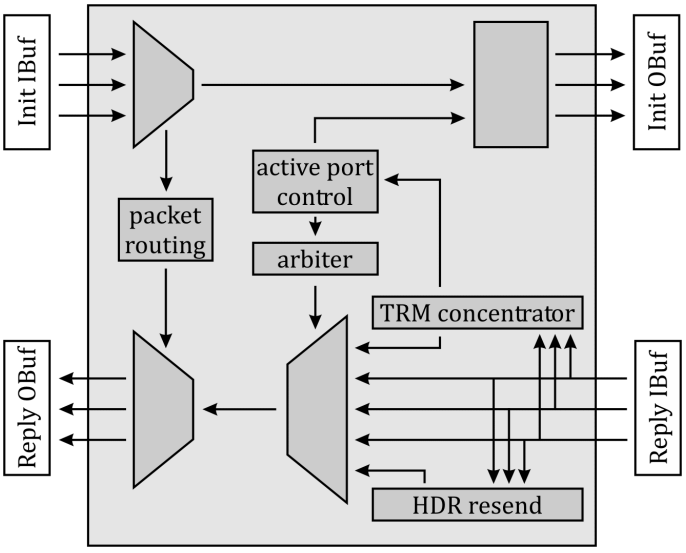

Figure 5.11: Implementation of the hub logic. Data received on the init channel is transported to all other init output buffers. The data streams received from all endpoints on the reply channel are merged to one single data stream, organized by an arbiter, and sent to the interface from which the init transfer came. All terminations are merged into one packet and sent after all data has been forwarded.

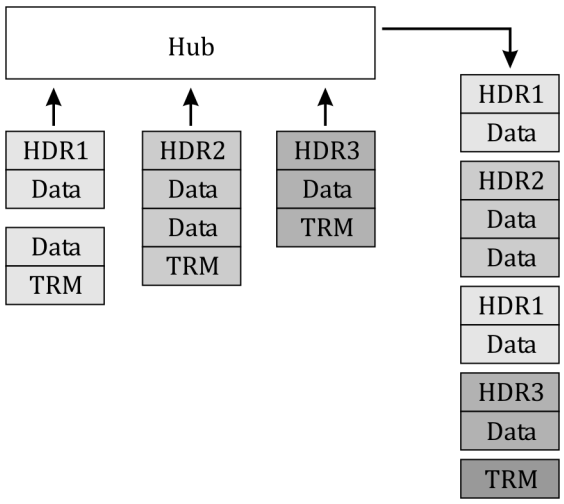

Figure 5.12: A hub merges the data transfered from different endpoints into one data stream. The termination from all points is merged to a single termination packet at the end of the stream. Note that the hub divides the transfer from the first endpoint into two bunches to achieve the highest possible data rate. The header of this endpoint is repeated to tag all data with the correct source address.

needed since there will be only one active endpoint per channel.

5.10 Streaming API

One of the future plans for the HADES DAQ is to integrate parts of the image processing algorithms into new hardware. The most obvious way to implement this functionality is that the processing application itself sends a request to the frontends and asks for data. Since the processing algorithms are splitted into many devices, this would cause a lot of distinct transfers.

The streaming API provides an efficient way to handle this. It allows to access a data stream, to modify it and to transmit it further after the changes have been applied (see figure 5.13a).

Another possibility to use the streaming API is to transport data to a different target as the original request node. This setup is shown in figure 5.13b: The streaming API controller reads the initial request to decide whether the answer should be redirected or not. When the reply arrives, it generates a new transfer using another API on an other channel or a different media interface. Only a short transfer is sent to the original target to deliver some status information.

This functionality can be used to remove big data transfers from TrbNet to be passed on over gigabit ethernet to send it to a readout computer.

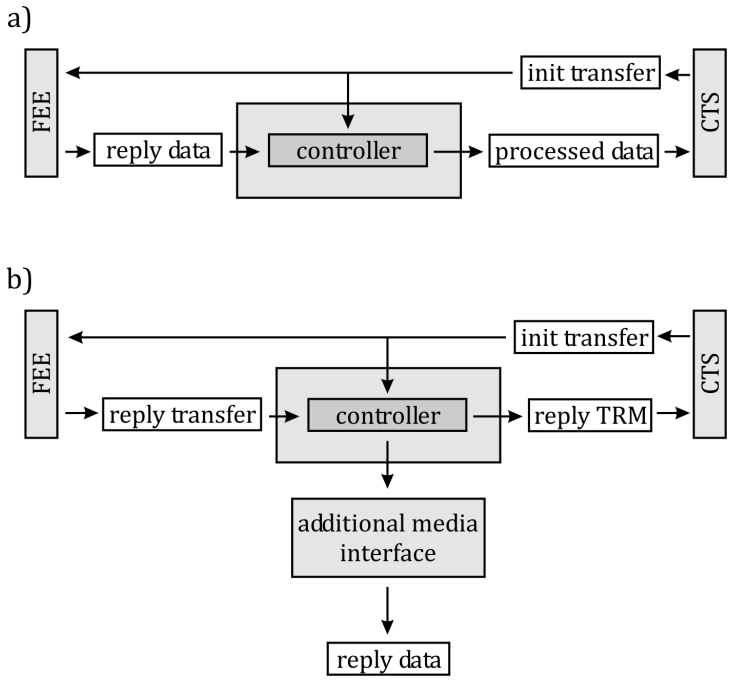

Figure 5.13: A streaming API is used to (a) process detector data or (b) to redirect data to a different target using an additional data interface.

5.11 Reading and Writing Registers

The RegIO component handles all data received on the slow control channel. Its structure is depicted in figure 5.14. It provides status registers that are common for all boards, user defineable status and control registers, and stores information about the hardware. As already mentioned, these common registers are needed to provide a universal monitoring tool for all parts of the detector.

Each register has one 16Bit address assigned as shown in Table 5.5. Each address corresponds to one 32 Bit register or data word, their description can be found in table B.2.

To access all registers and other information, a simple command structure is defined. The protocol is quite simple: For each access a 48 bit data word is sent as shown in Table 5.6. The DTYPE of the packets carries the desired operation, followed by the first 16 bit word with the register address and, in case of a write access, 32 bits of data. Altogether there are four different operations implemented: Single and multiple reads and writes from or to the same address. The multiple access is only supported on the internal data port and can be used to read data from a FIFO.

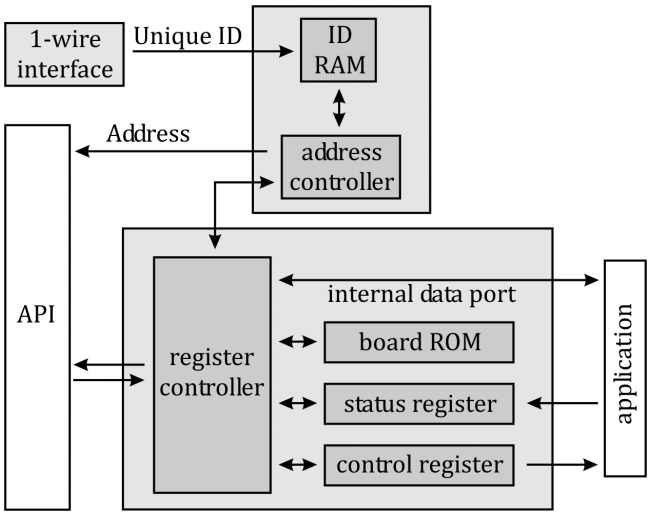

Figure 5.14: Implementation of the register read / write and address controllers. Data received from the application interface is interpreted by a central controller that access various registers as requested. An address controller manages the network addresses and the board ID.

ADDRESSES	IMPLEMENTATION STYLE	DESCRIPTION
0 - 1F	registers, internally writable	common status
20 - 3F	registers, externally writable	common control
40 - 4F	ROM	board information
50 - 7F		
80 - BF	registers, internally writable	user status
C0 - FF	registers, externally writable	user control
100 - FFFF	forwarded to internal data port	

Table 5.5: Definition of Register Addresses

DTYPE	DESCRIPTION	F1	F2	F3
8	read register	address	0	0
9	write register	address	data(31..16)	data(15..0)
A	read multiple	address	length	0
B	write multiple	address	data(31..16)	data(15..0)
F	network administration		see section 5.13	

Table 5.6: Register read/write protocol

	Packet 1			Packet 2		
DESCRIPTION	F1	F2	F3	F1	F2	F3
Set Address	SETADDR / ENDPOINTID	unique ID				Address
Ack. Address	ACKADDR	BoardInfo		—		
Read UID	READUID	—		—		
Send UID	UID				BoardInfo	

Table 5.7: Address assignment and board identification protocol. For all commands, the DTYPE is set to F.

5.12 Temperature Sensors

A temperature sensor (DS18S20 [DS1]) is available on every custom-built board for HADES[30] and used to provide the unique ID of every network node as well as the board temperature. The sensor uses the 1-wire bus protocol which allows to access multiple devices with only one wire despite a ground connection and an optional voltage supply. Depending on the board layout, the ID can be read out either directly from the FPGA using a 1-wire protocol implementation or, if the sensor is not connected directly to the FPGA, from any other source using a simple data port.

The 1-wire interface is not implemented with all features but supports two operations needed for TrbNet: Readout of the sensors unique ID and the current temperature. Even though the 1-wire protocol supports multiple devices on one bus, this is not implemented here.

The readout of the sensor is running all the time. At startup, the ID of the sensor is read out first. After about 1 ms the ID is provided to the network node which now can be assigned an address. Now the interface starts the temperature measurement. About 750 ms after the measurement has been started, the result is read out and provided on a status register.

5.13 Addresses and Unique IDs

Each node of the network needs a 16 bit address. Using hard coded (i.e. a fixed address inside the VHDL code files) addresses is not an option since this would need to many different design files. Hence, when starting up the network, the addresses are assigned to each endpoint by a central address arbiter. It uses a database including the configuration of all boards which in addition allows to check for their presence in the network.

To identify each board before addresses are assigned, a unique ID (UID) of 72 bits length is used. The first 64 bits of this ID can be retrieved from the temperature sensor that is available

[30] Boards without this sensor need a hardcoded identifier or an id register that can be written independently from TrbNet.

on all boards.

The remaining eight bits of the unique id make it possible to use multiple endpoints on the same board. For example, if a hub and an application are present in the same FPGA or there are two FPGAs on one board, both need different addresses and therefore different unique ids.

The transportation of the 72 bit unique ID needs two data packets to be sent. Since two packets are able to transport 96 bits of data, 24 bits remain unused. These bits are used to store basic information about the board that can be evaluated for future optimizations of the network. They include information about the kind of the endpoint, which channels it is connected to and more.

In order to readout the unique id and to set the network address of the board, a set of commands is defined as shown in table 5.7.

6 Specific Network Setup for HADES

6.1 Channel usage

The distinct logic for each channel leads to a high demand on available FPGA resources. Therefore only four of the possible sixteen channels are used in the new HADES DAQ concept. The first channel is, as mentioned before, used by the LVL1 trigger and the channel which has the lowest priority is used for slow control. In between there are two channels to transport the LVL2 trigger combined with detector data as explained below.

The LVL1 trigger channel is used to transport triggers only and no data. It contains the busy logic for the whole detector.

The second channel transports the IPU data, i.e. it provides the data link between the detector system and the IPUs. The processed data of the IPUs is then transported to the Matching Unit. Both parts of the data transfer are done within the same network session by implementing the IPUs with streaming APIs into the network. Some detectors are not included in the LVL2 trigger decision yet (for example the MDC), hence, these detectors are not using the IPU data channel.

The second level trigger is transported on the third channel. Upon a negative decision, the detector data is deleted. In case of a positive decision the data is transported to the readout system as reply to the LVL2 trigger information.

How this transport is done depends on the detector electronics: It can be read out directly from the detector front ends, or, if the data has already been transferred to the IPU, the data can be stored there, thus freeing the buffers of the front ends earlier.

6.2 Channel Configuration

Some features TrbNet provides are not necessary for a reliable operation of the LVL1 trigger channel. For every trigger, only one packet is sent in each direction. The locking behavior of the channels prohibits sending another trigger before all replies have been received. The handshake of the IOBufs can therefore be omitted for the first level trigger channel since no buffer overflow can occur and it does not provide additional information about the status of the system. The merged termination from all endpoints is sufficient to make sure that every front end board received the trigger.

As outlined in table 5.4, there are different possibilities to implement data buffers. The

smallest possible buffers would be sufficient here, but in hubs as well as most of the endpoints, the big FIFOs in internal memory blocks will be used. The difference in hardware consumption for both implementations is, despite the used memory block, nearly equal. Only in devices with very few memory resources the use of the small FIFOs will be necessary.

Almost all application interfaces will be implemented with passive behavior. There is only one active endpoint for each channel. The data and trigger channels are controlled by boards of the central trigger system. The slow control channel can be accessed from an application running for example on a VME CPU.

6.3 Common Registers

A set of common status registers is used to allow for a quick overview of the condition of the whole detector. Like other registers, they can be readout using the register read/write protocol described in section 5.11.

A network supervisor application running on a PC can for example readout these registers from all network nodes in defined time intervals (e.g. 1 Hz) the overall status of the network and inform the DAQ operator in case of problems. Since the registers are required for all nodes, the readout can easily be done by a single broadcast.

The definition of the registers is given in table B.2. In the first register, four bits are used to give a rough information about the boards condition, ranging from warnings to serious errors that need immediate intervention. Other bits show whether the internal trigger counter might not match the received one. In this case, the data from different events most probably got mixed up and a resynchronization of the trigger counters is needed. In case of a mismatch, the second register can be read out to see the current status of the internal trigger counters.

On some boards the temperature sensor can give additional information about the state of operation, so the temperature information is included as well.

Since these registers are only ment to provide an overview of the status, special queries depending on the individual board, have to be made, to find out more details about the source of an error.

6.4 Error Pattern

The error pattern is formed by 32 bits that are transported with each termination and is merged from all endpoints. The standard value of all bits is 0. The pattern is divided into two blocks of 16 bit: The first one is common for all channels while the second one is channel dependent. These bits are listed in table 6.1.

The first bit in the termination is generated by an addressed endpoint. When a long transfer is sent back, the source address is transported with the header and allows to determine if the packet actually reached its destination and was interpreted correctly. In case the answer consists

BIT	SHORT NAME	DESCRIPTION
0	endpoint reached	Data was received by addressed endpoint
1	collision detected	A hub detected a data collision. This data transfer has been dumped
2	word missing	Mismatch between received amount of data and data count
3	checksum error	Mismatch between received and calculated checksum
4	don't understand	The addressed endpoint was not able to process the data
5 - 11	reserved	
12	hardware error message	An hardware error has occured
13	software error message	An software error has occured
14	warning message	A non-vital problem has occured
15	info message	Information about a possible problem
16 - 31	reserved for channel specific information	

Table 6.1: Common Errorbits definition

of a short transfer only, this bit will be set by the addressed endpoints to acknowledge that the data has been received.

When using multiple active endpoints on one channel, data conflicts might occur in a hub when two endpoints initiate a transfer at the same time. If such a collision happens, a special bit in the error pattern is set to show that the data was not transported properly and most likely is lost.

Another four bits are used to see whether there was an error somewhere in the detector hardware that should be handled. Normally, these error states should be read out from each board individually using the common registers, but in some cases the additional information transported with every termination can help to solve an error more quickly.

6.5 Data Types

The data type (DTYPE) is a four bit information sent with every header that allows to distinguish between different kinds of data. On the slow control channel, the data type is used to select the kind of register operation that should be done. The two trigger channels use it to specify the trigger code (see table B.1 for the complete list of data types).

Data sent by the front end electronics could be marked by the data type to distinguish between different data formats. There is no need to insert the trigger code here, since data is always sent as a reply and the trigger code is already included in the initial transfer.

7 Performance Measurements

To determine the performances of TrbNet a test setup containing up to 6 endpoints on 6 different boards was built up as shown in figures 7.1 and 7.2. The achievable data rates and transport latencies have been measured and compared to calculations.

7.1 Effective Data Rates

The theoretical achievable data rate can be calculated by two simple considerations. The first is the amount of data transported with each data packet. The size of a data packet is defined to be 64 bit of which 48 bit are used to transport data, whereas 16 bit are needed for channel and packet information. Therefore, the maximum data rate for data packets only is 75% of the medium datarate.

The second part of overhead is caused by the handshake of the network protocol. For long transfers, the contribution of header and termination is negligible or at least small. By using a buffer size of 6, corresponding to a memory capable of storing 127 packets per buffer, every 127th packet has to be an EOB. Hence, it adds roughly one percent to the overhead.

To conclude at this point, for long transfers and large buffers, the usable bandwidth is about 74% of the network bandwidth. For the optical link using a speed of 1.6 Gbit/s[31] this corresponds to a data rate of 1.187 Gbit/s.

[31] Link running at 2 Gbit/s, but using 8b/10b encoding and therefore effectively loosing 20% bandwidth by itself.

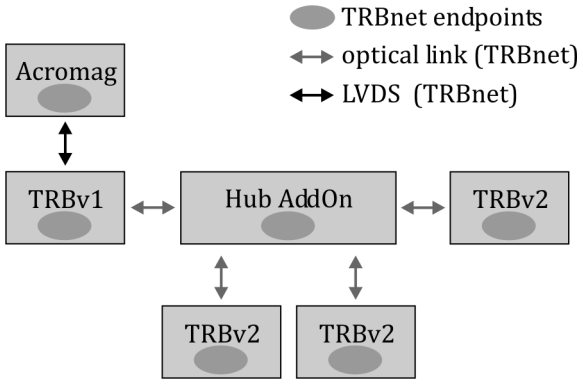

Figure 7.1: Schematic view of the network test setup

Figure 7.2: Network Test Setup.

Compared to other networks like Ethernet, this percentage is rather low. UDP packets for example have about 68 byte overhead for each packet of up to 1500 byte size, and therefore 4.3% overhead. However, these values cannot be compared directly, since one has to take into account that for UDP additional overhead is needed for flow control and checksums, whereas this feature is included in the TrbNet calculation.

The amount of data made up by the ACK packets in the opposite direction is not taken into account, since they do not influence the data rate. The bandwidth used for ACK packets is the same as for EOB and TRM packets combined and therefore add up to less than one percent of the bandwidth used for sending data.

The major part of the overhead is caused by the high granularity of TrbNet. By increasing the packet sizes by a factor of two it might be reduced from 25% to 12.5% while the usable bandwidth would be increased by 17%. This would allow 1.38 Gbit of data to be transported per second in case of optical links.

Nonetheless, the latency for a high priority packet to be transported increases by a factor of two from a maximum of 50 ns to 100 ns, which might not fit to the requested response times for triggers.

The real data rates have been measured on a single channel point-to-point connection, using two boards with TLK2501 chips connected with 10m long optical fibers. Both implementations used FIFOs with a depth of 127 packets. The number of bytes sent by one application was

TRANSFER SIZE (BYTE)	TIME (µS)	DATA RATE (GBIT/S)
12	1.11	0.22
480	4.17	0.94
4800	33.1	1.16

Table 7.1: Measured data rates using an optical link with TLK2501 chips and a varying transfer size.

varied while the response always consisted of 12 data bytes. The time between the start of the header packet and the end of the termination was measured. The results are shown in table 7.1.

For a long transfer, the achieved data rate is almost at the theoretical limit. It should be pointed out, that the unused bandwidth when doing shorter transfer is not lost, but can be used for transfers on other channels.

7.2 Transmission Latency

To get an estimate for the maximum transmission delay for the whole network, the delays introduced for transporting data have to be taken into account, as well as the latency of 50 ns per link introduced by the granularity of TrbNet packets.

The test setup contains two boards, both being equipped with Xilinx Virtex 4 FPGAs [Xil07], TLK2501 serial transceivers [TLK03] and optical transceivers [Fin05]. The connection between both is done by 10m optical fibers.

Usually, the way to measure the transmission delay between two boards is to measure the time between writing a word to the media interface of one board and retrieving this word from the media interface of the receiver. Unfortunately this method was not possible due to the lack for sufficient debug connectors.

The measurement was therefore made by measuring the time between sending an EOB and receiving the ACK packet on the data in- and output of the senders media interface toward the multiplexer.

Opposed to the initial setup, this measurement gives two times the transmission delay from media interface to media interface plus one time the delays introduced by multiplexer and IOBuf. The former delays can be calculated based on the devices data sheets while the latter can easily be derived from simulation or by studying the source code.

The transmission latency of the TLK2501 is given [TLK03] as 34 to 38 bit times, the receive latency is 76 to 107 bit times. A bit time corresponds to 0.5 ns, so the combined delay is about 70 ns.

All latencies sum up to about 240 ns as shown in table 7.2. The additional measured delay in the multiplexer and IOBuf is 60 ns, so the expected response time is 540 ns.

DESCRIPTION	TIME (NS)
Media Interface RX - Delay from internal data input to data output	50
Output time from internal signal to FPGA I/O pad	10
TLK transmit latency	20
10m optical fiber transport delay[a]	60
TLK receive latency	50
Media Interface TX - Delay from data input to internal data output	50
Total data latency	240

Table 7.2: Detailed list of transmit and receive latencies for an optical link using external TLK2501 transceivers.

[a]The speed of light in an optical fiber is between 0.4 and 0.6 times the speed of light in vacuum

The measured value varies between 500 ns and 530 ns because of the jittering alignment between the involved clock domains. Assuming a latency of 240 ns between two media interfaces therefore seems to be a good estimate.

According to the data sheet of the Lattice FPGAs [Lat08a] the delays for its internal SERDES are higher than for the external TLK2501. The transmit latency is given to be about 10 internal clock cycles, the receive latency is about 30 clock cycles, summing up to 400 ns when running at 100 MHz. These numbers were confirmed to be accurate by measurements in a test setup consisting of one hub addon and one TRB.

Assuming that no additional media interface is needed and the output time to the optical transceivers is negligible, the total transport latency between two Lattice FPGAs is about 460 ns and by this means twice as high as for a connection using TLK2501 chips.

To calculate the data latency for the full network setup, two more delay sources have to be taken into account: The latency for a hub to transport data from one media interface to the other is less than 10 clock cycles and the IOBuf and API need less than 7 clock cycles to transport data between media interface and application.

For the HADES setup, two layers of hubs are needed to distribute triggers to all detector parts. In case of the MDC another hub on the MDC AddOn is used to connect to the vast number of front end boards. This adds up to four optical connections[32] plus a shorter delay for the connection between TRB and MDC AddOn. Four of the involved FPGA will be Lattice devices while the other three are Xilinx FPGAs with external optical transceivers. The latency between CTS and the MDC FEE, which is assumed to be the longest network connection, is therefore about 2.1 µs for each direction. Since the trigger transmission occurs during the time the detector needs to read out the electronics, its latency does not contribute to the total detector

[32]A new version of the MDC readout will use optical fibers instead of the current electrical cables.

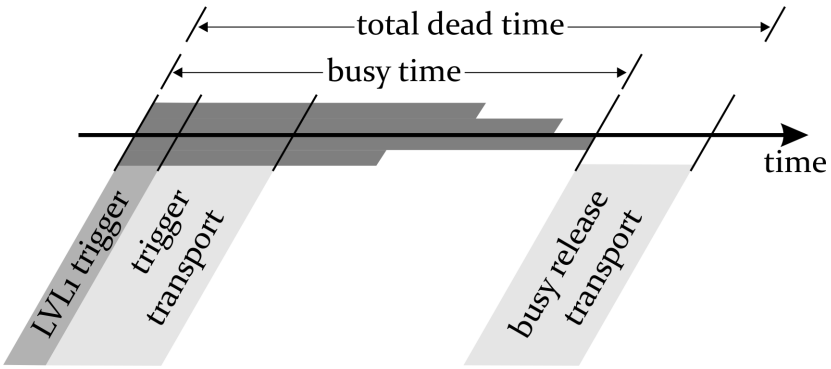

Figure 7.3: The total detector dead time is determined by the busy time of the detector electronics and the time needed to transport the busy release signal, whereas the trigger transport does not contribute to the dead time.

dead time whereas the reply transfer does. As can be seen in figure 7.3, the total dead time is the dead time of the slowest detector part plus the latency for the busy release signal from this point to the CTS.

Therefore, the TrbNet adds about 2.1 µs to the detector dead time. Since the latency is fixed by the network setup, the real latency between busy release and the next trigger can be shortened: The busy release can be sent as soon as it is sure that the detector is able to process the next trigger 2 µs later.

7.3 Logic Resources Usage

Another important point is the logic consumption of the implementation in each FPGA. The implementations can roughly be divided into three different setups: Standalone hubs, endpoints and combined implementations.

The size of a hub is mainly defined by the number of media and application interfaces it provides. All other configuration options are more or less fixed by the requirements for the HADES network.

Table 7.3 shows the different implementation sizes for various implementations. The sizes for an implementation on a Xilinx Virtex4 LX40 [Xil07] and a Lattice SCM25 [Lat08b] are shown. In all cases the optical media interfaces are included. The different number of used logic blocks is caused by the different media interfaces.

The configuration application is the only part of the hub that exists once without respect to the number of connected interfaces. All other parts are instantiated for each channel individually. Therefore, the amount of resources occupied by the hub logic scales almost linearly with the number of interfaces with an offset of about 1000 slices.

The available size of the Virtex4 LX40 is about 36k LUT, which allows to implement up to

Ports	Xilinx Slices	LUT	FF	Lattice Slices	Both RAM
2 (1 channel)	1200	2050	1450	1785	8
2	3500	6350	3700	3750	14
4	6000	10800	6100	6300	26
8	11000	19600	10800	11100	50
12	16000	28700	15550	$\approx$ 16200	74
16	20900	37000	20300	$\approx$ 21000	98

Table 7.3: Logic resources needed for hub implementation with different number of ports. Designs for Xilinx and for Lattice devices have been evaluated, however they are not directly comparable, as both implementations need different media interfaces. Despite the first entry, all sizes are measured for a full four channel setup.

14 media interfaces. Nonetheless, this setup will not be needed, since on all boards with many media interfaces Lattice chips are used. The current hub board contains a SCM25 chip that has about 12.7k logic blocks, corresponding to a hub design with up to 8 ports. Currently a new hub board is being developed featuring a Lattice ECP2M100 FPGA [Lat08a] with 48k logic blocks. For a 16 ported hub about half of its resources will be occupied.

The size of an endpoint is determined for a full configuration that handles all four channels. This is, receiving LVL1 triggers, sending and receiving data on the two data channels and having register read and write capabilities using the slow control channel. The EOB/ACK handshake was switched of for the LVL1 trigger channel as described in sec. 6.2. The management of TrbNet addresses and the readout of a temperature sensor was included as well. Since the sizes of the design do not differ much, the calculations were done for a Xilinx Virtex4 FPGA only and are shown in table 7.4.

PART	NUMBER	SLICES	FF	LUT	BRAM
optical Media Interface	1	250	350	360	2
Multiplexer	1	170	90	320	0
IOBuf[a]	4	240	240	450	1
full API	3	280	300	510	2
trigger API	1	30	40	50	0
Slow Control	1	360	320	620	0
Full Endpoint	1	2050	2300	3600	10

Table 7.4: Logic resources needed for a full featured network endpoint with four channels and slow control capabilities. The numbers given for the individual components do not add up to the given number of the whole endpoint due to optimization processes done by the synthesis tool and differences in the configuration used on different channels.

[a] Using BlockRAM FIFOs and including checksum generation

8 Summary and Outlook

The TRB network inherits the same logical structure that was used for the old trigger bus but does so by very different means. A network protocol has been developed that is capable of connecting all Types of detector read-out boards and is able to perform all data transfers needed. This improves the overall efficiency by increasing bandwidth and decreasing noise levels while the simplified maintenance will assure shorter downtimes upon failures of single detector devices.

The performance measurements have demonstrated that the achieved performance meets the requirements in every respect. The latency of the network is around 2 µs, but its impact on the detector dead time can be further reduced with little effort. The data rate of 1.18 GBit/s is sufficient for high event rates in experiments with heavy ions. The consumption of logic resources is low enough to implement the protocol in small FPGAs on front-end electronics.

Even though the design of the TrbNet protocol was primarily based on the specific needs for the HADES experiment, TrbNet provides a number of configuration options to be adaptable to the needs of other experiments. TrbNet can for example be used in any kind of detector test setup where a secured transfer of data from several units to a central system is needed.

If necessary, a routing algorithm can be implemented inside the hubs to reduce the network load and improve the available bandwidth. The data format includes spare bits that can be used to divide the network into subnets which can be addressed independently and therefore permit the use of a vast number of network nodes.

The number of different channels can be increased to allow more different applications to be active at the same time. The current packet format forsees a maximum of sixteen channels, but this number can also be enhanced by changing the packet format slightly.

The locking behavior needed for data readout and trigger distribution can be omitted for other applications. Despite the lower usage of logic resources this also provides the possibility to use multi-master network setups with an unlimited number of active endpoints. Hence, this opens up the option to use TrbNet in free running, triggerless experiments like the PANDA detector or the readout of the CBM micro vertex detector.

In conclusion one can state that TrbNet is a versatile network that fits to the needs of the HADES experiment but also allows for numerous extensions and adaptions to future requirements.

A List of Files

Table A.1 provides a list of most of the files needed for TrbNet. Each file contains one entity with the same name. The number "16" in some of the file names refers to their implementation to be specific for 16 bit wide words and 64 bit packets. Entities without this number are able to handle words with different wides or do not rely on word widths at all.

Table A.1: List of design files used for TrbNet.

NAME	DESCRIPTION
trb_net16_hub_base	Top entity for any hub. Instantiates all necessary buffers, applications and the hub logic
trb_net16_hub_func	Library with functions used for hub configuration
trb_net16_hub_logic	Main hub logic, including data concentrating and routing
trb_net16_iobuf	Top entity for the input and output buffers
trb_net16_ibuf	Input buffer, instantiated by the IOBuf
trb_net16_obuf	Output buffer, instantiated by the IOBuf
trb_net16_term_buf	Termination buffer, used instead of iobuf in not used channels to terminate any incoming transfer
trb_net16_term_ibuf	Special ibuf for channels where only short transfers are received
trb_net16_io_multiplexer	Data multiplexer and demultiplexer used to merge and separate the data channels
trb_net_pattern_gen	Decoder from an address bus to single enable signals
trb_net_priority_arbiter	Priority arbiter for hub and multiplexer, includes optional round robin pattern
trb_net_priority_encoder	Generates bit pattern needed for priority arbiter
trb_net16_med_tlk	Media interface for an optical link using external tlk chips
trb_net_med_8bit_slow	Media interface for a synchronous, parallel data transfer with 8 data and 5 control bits
trb_net16_api_base	Application interface. Connected to IOBuf and user application

trb_net16_regIO	Provides register read and write access over the network. Connected like an application to the API on the slow control channel.
trb_net16_addresses	Address assignment logic. Instantiated by trb_net16_regIO
trb_net_onewire	1-wire interface to read out temperature sensors
trb_net_sbuf	Sbuf is used instead of simple flipflops due to the free running handshake
trb_net16_sbuf	Sbuf version for 16 bit words
trb_net_fifo	General FIFO entity
trb_net_dummy_fifo	A dummy FIFO. Can be used instead of a FIFO, when no bigger storage capability is needed
trb_net16_dummy_fifo	16 bit version of dummy FIFO
trb_net_fifo_16bit_bram_dp	A blockram FIFO with two independent clocks
trb_net_ram_dp	Generic RAM description for arbitrarily sized memories, with two read and one write port
trb_net_ram_16x8_dp	Dual ported RAM with a fixed size of 16 words with 8 bit, with support for init values
trb_net_ram_true_dp	Arbitrarily sized RAM with two read and two write ports
trb_net_std	Library with basic functions and constant declarations
**device*_fifo_18x16*	Device specific code for a FIFO with 18 bit width and a depth of 16 words
**device*_fifo_18x32*	Device specific code for a FIFO with 18 bit width and a depth of 32 words
**device*_fifo_18x64*	Device specific code for a FIFO with 18 bit width and a depth of 64 words
**device*_fifo_18x1k*	Device specific code for a FIFO with 18 bit width and a depth of 1k words
**device*_fifo_dp_18x1k*	Device specific code for a FIFO with 18 bit width and a depth of 1k words with two independent clocks

B Register and Bitfield Definitions

Data Types

Table B.1: Definition of data types for each channel

TYPE	TRG LVL 1	IPU DATA	TRG LVL 2	SLOW CONTROL
0 - 7	trigger code	data type	trigger code	
8				register read
9				register write
A				multiple read
B				multiple write
C				
D				
E				
F				network administration

Slow Control Registers

Table B.2 shows the definition for common status and control registers for all network endpoints.

Hub Control and Status Registers

Packet Types

The definitions of all network packet types are given in table B.4.

Table B.2: Register Definitions

NAME	ADDRESS		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Common Register 1	0000	l	error messages				trg mismatch		tbd									
		h	tbd				temperature											
Common Register 2	0001	l	trigger level 1 counter															
		h	trigger level 2 counter															
Board ROM 1	0040	l	compile time															
		h	compile time															
Board ROM 2	0041	l	compile version															
		h	compile version															
Board ROM 3	0042	l	Board information															
		h	Board information															

Table B.3: Additional Slow Control Registers for Hub Control and Monitoring

NAME	ADDRESS	BIT 15-0	BIT 31-16
Port Status 0	0080	Busy Ports (Channel 0)	
Port Status 1	0081	Busy Ports (Channel 1)	
Port Status 2	0082	Busy Ports (Channel 2)	
Port Status 3	0083	Busy Ports (Channel 3)	
Hub Status	0084	Overview of Hub Status (tbd)	
Port Control 0	00C0	Active Ports (Channel 0)	
Port Control 1	00C1	Active Ports (Channel 1)	
Port Control 2	00C2	Active Ports (Channel 2)	
Port Control 3	00C3	Active Ports (Channel 3)	
Hub Control 0	00C4	General Hub Control (tbd)	
Hub Control 1	00C5	Channel 0 Control (tbd)	Channel 1 Control (tbd)
Hub Control 2	00C6	Channel 2 Control (tbd)	Channel 3 Control (tbd)

Table B.4: Packet type definitions

NAME	VALUE	DESCRIPTION	F1	F2	F3
DAT	0	Normal data word	Data		
HDR	1	Header / Source change	Source Address	Target Address	sequence number / data type
EOB	2	End of Buffer	0	Data count	Checksum
TRM	3	Termination	Errorbits		sequence number / data type
EXT	4	Extended data word	checksum, other error detection		
ACK	5	Acknowledge		Length of Buffer	
—	6	—	—		
ILL	7	Illegal word	ignore		

C Port Definitions

The following tables show the ports of all main TrbNet entities.

- All data ports as described in section 5.2 are abbreviated to only one entry.
- The standard inputs CLK, RESET and CLK_EN for the clock, reset respectively clock enable are not shown.
- Status and control ports used for debugging only are not shown.
- The underlines used in the source code have been omitted for better readability.

Multiplexer

Table C.1: Ports of the data (de-)multiplexer

PORT	WIDTH	DESCRIPTION
MII	–	Media independent interface as defined in table C.2
INT DATA OUT PORT	–[33]	Internal data output to IBuf for each channel and path
INT DATA IN PORT	–[34]	Internal data input from OBuf for each channel and path

[33] One data port for every channel and path, i.e. 8 ports with standard setup
[34] One data port for every channel and path, i.e. 8 ports with standard setup

Media Independent Interface

Table C.2: Ports of the Hub

PORT	WIDTH	DESCRIPTION
MED DATA IN PORT	–	Data port to transport data to from the medium to the internal logic
MED DATA OUT PORT	–	Data port to transport data from the internal logic to the medium
MED ERROR	3	Error status of the medim interface as defined in table 5.2

IOBuf

Table C.3: The input and output buffer ports

PORT	WIDTH	DESCRIPTION
MED INIT DATA IN PORT	–	Data input from the multiplexer for the init channel
MED REPLY DATA IN PORT	–	Data input from the multiplexer for the reply channel
MED INIT DATA OUT PORT	–	Data output to the multiplexer for the init channel
MED REPLY DATA OUT PORT	–	Data output to the multiplexer for the reply channel
INT INIT DATA IN PORT	–	Data input from the application interface for the init channel
INT REPLY DATA IN PORT	–	Data input from the application interface for the reply channel
INT INIT DATA OUT PORT	–	Data output to the application interface for the init channel
INT REPLY DATA OUT PORT	–	Data output to the application interface for the reply channel

API

Table C.4: The application interface ports used to send and receive data

PORT	WIDTH	DESCRIPTION
TRANSMITTER		
DATA IN PORT	–	Data input from the application.
SHORT TRANSFER IN	1	The transfer should not contain any data and is a short transfer consisting of TRM only
SEND IN	1	Start sending of data with rising edge and ends transmission with falling edge
DTYPE IN	4	The type of data that is sent
ERROR PATTERN IN	32	Errorbits that are sent with the next TRM packet
TARGET ADDRESS IN	16	The address an active API sends the data to
RECEIVER		
DATA OUT PORT	–	Data output to the application.
TYP OUT	3	Type of network packet the data belongs to
RUN OUT	1	Active while a transfer is in progress
SEQNR OUT	8	Sequence number of current transfer[35]
MY ADDRESS IN	16	Address of this network endpoint[36]

[35]The sequence number is used for example to transport the trigger number

[36]The network address is output by RegIO respectively by the network administration entity.

Hub

Table C.5: Ports of the Hub

PORT	WIDTH	DESCRIPTION
MED INTERFACE	–	A complete media independent interface as defined in table C.2. The width of every port is multiplied by the number of used media interfaces
API INTERFACE	–	A complete API interface as defined in table C.4. The width of every port is multiplied by the number of used application interfaces.

D Configuration Options

IOBuf

Table D.1: The configuration options for the input and output buffer

NAME	VALUES	DESCRIPTION
IBUF DEPTH	0 - 6	size of data buffers
IBUF SECURE MODE	yes / no	activates SBuf on the IBuf's data output
USE ACKNOWLEDGE	yes / no	use EOB/ACK on this channel
USE CHECKSUMS	yes / no	send and validate checksum with EOB
USE VENDOR CORES	yes / no	chooses whether generic FIFOs or device dependent implementations should be used
INIT CAN SEND DATA	yes / no	implement init OBuf with capability of sending data. Otherwise it can send ACK only
REPLY CAN SEND DATA	yes / no	implement reply OBuf with capability of sending data. Otherwise it can send ACK only[37]

[37] On channels that use EOB/ACK every endpoint must be able to send data on the reply path

API

Table D.2: The configuration options for the application interface

NAME	VALUES	DESCRIPTION
API TYPE	passive / active	Selects the API to show passive or active behaviour
FIFO TO APL DEPTH	1 - 6[38]	Sets the depth of the fifo used to store data from the network to the application.
FIFO TO INT DEPTH	0 - 6	Sets the depth of the FIFO used to store data from the application before it is transported to the OBuf
FORCE REPLY	yes / no	selects whether an answer to a request is required or optional
USE VENDOR CORES	yes / no	chooses whether generic fifos or device dependent implementations should be used
SECURE MODE TO APL	yes / no	implements an sbuf on the data output towards the application
SECURE MODE TO INT	yes / no	implements an sbuf on the data output towards the OBuf
APL WRITE 4 PACKETS	yes / no	selects whether the application receives the whole 64 bit packet or the three data words only
BROADCAST BITMASK	00 - FF	sets the broadcast address bitmask

[38] A dummy FIFO cannot be used in this place due to the address recognition needs at least a three packets deep FIFO

RegIO

Table D.3: The configuration options for the register read and write component

NAME	VALUES	DESCRIPTION
NUM STAT REGS	1 - 6[39]	Address width used for user status registers
NUM CTRL REGS	1 - 6	Address width used for user control registers
INIT CTRL REGS	–[40]	Standard values for user control registers
USED CTRL REGS	–[41]	Switches control registers on or off
USED CTRL BITMASK	–[42]	Switches single bits in used control registers on or off
NO DAT PORT	yes / no	Activates data and address port to user application
INIT ADDRESS	0000 - FFFF	Initial network address after startup
INIT UNIQUE ID	96 bit	Initial unique id on startup before temperature sensor is read out
COMPILE TIME	32 bit	Compile time stored in board information ROM
COMPILE VERSION	16 bit	Consecutive number for each code version stored in board information ROM
HARDWARE VERSION	32 bit	Hardware version stored in board information ROM

[39] A single register cannot be selected due to restrictions in xilinx XST synthesis tool.
[40] Vector with 2^NUM CTRL REGS·32 width
[41] Vector with NUM CTRL REGS width
[42] Vector with 2^NUM CTRL REGS·32 width

Hub

Table D.4: The configuration options for the hub and data concentrator

NAME	VALUES	DESCRIPTION
MUX SECURE MODE	yes / no	Use SBufs at the output of the demultiplexer
HUB CTRL DEPTH	1-6	depth of the FIFOs in API on control channel
HUB USED CHANNELS	yes / no[43]	Switch off channels that are blocked by this hub
IBUF SECURE MODE	yes / no	Use an SBuf for the data output of the input buffer
INIT ADDRESS	32 bit	Initial network address before real address is assigned
INIT UNIQUE ID	96 bit	Initial unique id on startup before temperature sensor is read out
COMPILE TIME	32 bit	Compile time stored in board information ROM
COMPILE VERSION	16 bit	Consecutive number for each code version stored in board information ROM
HARDWARE VERSION	32 bit	Hardware version stored in board information ROM
MII NUMBER	2 - 16	number of media interfaces connected to the hub entity
MII IBUF DEPTH	1 - 6[44]	Selects the depth of the IBuf for each instance
API NUMBER	1 - 8	Number of application interfaces connected to the hub entity
API CHANNELS	–[45]	Selects the channel each application interface is connected to
API TYPE	act / pas[46]	Selects the type of application interface
API FIFO TO INT DEPTH	1 - 6	Depth of FIFO for each application interface for sent data
API FIFO TO APL DEPTH	1 - 6	Depth of FIFO for each application interface for received data

[43] Vector with one entry for each channel
[44] array with one entry per channel and media interface
[45] array with one entry per application interface. Values are all valid channel numbers
[46] array with one entry per application interface

Bibliography

[A+07] G. Agakichiev et al. Dielectron production in C-12 + C-12 collisions at 2-AGeV with HADES. *Phys. Rev. Lett.*, 98:052302, 2007.

[A+08] G. Agakichiev et al. Study of dielectron production in C+C collisions at 1 AGeV. *Phys. Lett.*, B663:43–48, 2008.

[B+00] M. Böhmer et al. Lepton identification with the CsI based RICH of HADES. *Nucl. Instrum. Meth.*, A471:25–29, 2000.

[B+04] A. Balanda et al. The HADES Pre-Shower detector. *Nucl. Instrum. Meth.*, A531:445–458, 2004.

[B+06] D. Belver et al. The front-end electronics for the HADES RPC wall (ESTRELA-FEE). *Nucl. Phys. Proc. Suppl.*, 158:47–51, 2006.

[BR91] G. E. Brown and Mannque Rho. Scaling effective Lagrangians in a dense medium. *Phys. Rev. Lett.*, 66:2720–2723, 1991.

[Col06] NA60 Collaboration. NA60 status report. 2006.

[DS1] *DS18B20 Programmable Resolution 1-Wire Digital Thermometer Manual*.

[ECS] C. Ernst *et al.*, Phys. Rev. C58, 447 (1998); W. Cassing *et al.*, Phys. Rep. 308, 65 (1999); K. Shekhter *et al.*, Phys. Rev. C68, 014904 (2003); M. Cozma *et al.*, Phys. Lett. B640, 150 (2006).

[F+08] I. Fröhlich et al. A General Purpose Trigger and Readout Board for HADES and FAIR-Experiments. *IEEE Trans. Nucl. Sci.*, 55:59–66, 2008.

[Fin05] Finisar. *FTLF8519P2xNL 2.125 GB/s Short-Wavelength SFP Transceiver - Product Specification*, September 2005.

[Frö] Ingo Fröhlich. priv. comm.

[Gao05] Jon Gao. 10 100 1000 mbps tri-mode ethernet mac. `http://opencores.org/projects.cgi/web/ethernet_tri_mode/overview`, 2005.

[Gar98] C. Garabatos. The HADES dilepton spectrometer. *Nucl. Phys. Proc. Suppl.*, 61B:607–612, 1998.

[Kir07] Daniel Kirschner. *Level 3 Trigger Algorithm and Hardware Platform for the HADES Experiment*. PhD thesis, Justus-Liebig-Universität Gießen, 2007.

[L+03] J. Lehnert et al. Performance of the HADES ring recognition hardware. *Nucl. Instrum. Meth.*, A502:261–265, 2003.

[Lat08a] *LatticeECP2/M Family Handbook HB1003*, April 2008. `http://www.latticesemi.com/documents/HB1003.pdf`.

[Lat08b] *LatticeSC/M Family Handbook HB1004 Version 02.0*, March 2008. `http://www.latticesemi.com/documents/DS1004.pdf`.

[Lin01] Erik Lins. *Entwicklung eines Auslese- und Triggersystems zur Leptonenidentifizierung mit dem HADES-Flugzeitdetektor*. PhD thesis, Justus-Liebig-Universität Gießen, 2001.

[M+07] A. Marin et al. Dilepton measurements with CERES. *PoS*, CPOD07:034, 2007.

[Myr05] *Myrinet Overview*, June 2005. `http://www.myricom.com/myrinet/overview`.

[P+97] R. J. Porter et al. Dielectron cross section measurements in nucleus nucleus reactions at 1.0-A-GeV. *Phys. Rev. Lett.*, 79:1229–1232, 1997.

[PB61] W. W. Peterson and D. T. Brown. Cyclic Codes for Error Detection. *IRE*, 49, 1961.

[Pet00] Markus Petri. *Entwicklung eines kombinierten Auslese- und Echtzeit-Triggersystems zum Nachweis von Elektronen/Positronen-Signaturen in einem elektromagnetischen Schauerdetektor*. PhD thesis, Justus-Liebig-Universität Gießen, 2000.

[Pos80] Jon Postel. *RFC 768 - User Datagram Protocol*, August 1980.

[Pos81a] Jon Postel. *RFC 791 - Internet Protocol*, September 1981.

[Pos81b] Jon Postel. *RFC 793 - Transmission Control Protocol*, September 1981.

[S+95] P. Salabura et al. HADES: A High Acceptance DiElectron Spectrometer. *Nucl. Phys. Proc. Suppl.*, 44:701–707, 1995.

[S+04] P. Salabura et al. Studying in-medium hadron properties with HADES, 2004.

[S+07] Christoph Schrader et al. A readout system for the cbm-mvd demonstrator. *GSI Scientific Report*, 2007.

[Tar] Attilio Tarantola. The upgrade of the multiwire dirft chamber readoutof the hades experiment at gsi. Abstract #2106 submitted for IEEE 2008 Dresden conference.

[TLK03] *TLK2501 - 1.5 to 2.5 Gbps Transceiver*, July 2003.

[Tra] Michael Traxler. Daq upgrade overview. Hades Wikipage `http://hades-wiki.gsi.de/cgi-bin/view/DaqSlowControl/DaqUpgradeOverview`.

[Tra01] Michael Traxler. *Real-Time Dilepton Selection for the HADES Spectrometer*. PhD thesis, Justus-Liebig-Universität Gießen, 2001.

[Tra06] Michael Traxler. A general purpose trigger and readout board (trb) for hades and fair-experiments. *GSI Scientific Report*, 2006.

[Xil07] XilinxInc. *Virtex-4 User Guide 2.3*, August 2007.

[Zim80] Hubert Zimmermann. Osi reference model - the iso model of architecture for open systems interconnection. *IEEE Transactions On Communications*, April 1980.

Danksagung

Mein Dank gilt allen, die mich während des Studiums und der Arbeit an der Diplomarbeit unterstützt haben, im Besonderen:

- Prof. Dr. Joachim Stroth für die Aufnahme in seine Arbeitsgruppe und die Möglichkeit, an diesem interessanten Projekt arbeiten zu können.
- Dr. Ingo Fröhlich für die sehr gute Betreuung während der gesamten Arbeit und die viele Mühe beim Lesen und Verbessern dieser Arbeit.
- Dr. Christian Sturm für seine fortwährende Hilfe bei jedweder physikalischer Fragestellung sowie das Korrekturlesen dieser Arbeit.
- Dr. Michael Traxler für die Bereitstellung der Hardware für die Tests des Netzwerks.
- Marek Palka für die Arbeit an der Konfiguration der optischen Links.
- Christoph Schrader für die Unterstützung bei der Suche nach dem ein oder anderen versteckten Bug in Code und Hardware.
- Der gesamten Frankfurter HADES-, CBM- und R3B-Gruppe für zahllose interessante Gespräche und Vorträge über Arbeitsgebiete und viele andere Themen, sowie die zeitweise Ablenkung beim gemeinsamen Arbeiten an anderen Problemen.
- Meinen Zimmergenossen Christoph und Samir für die angenehme Arbeitsatmosphäre in Büro und Labor.
- Meinen Kommilitonen, insbesondere Meike und Daniel, ohne die das ganze Studium sicher nicht so angenehm gewesen wäre.
- Meinen Eltern, nicht zuletzt für die finanzielle Unterstützung während des gesamten Studiums.