Evaluation of an FPGA and PCI Bus based Readout Buffer for the Atlas Experiment

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Diplom-Physiker Matthias Müller
aus Worms

Mannheim, 2004

Dekan:	Prof. Dr. Jürgen Potthoff, Universität Mannheim
Referent:	Prof. Dr. Reinhard Männer, Universität Mannheim
Korreferent:	Prof. Dr. Peter Fischer, Universität Mannheim

Tag der mündlichen Prüfung: 14. Februar 2005

Evaluation of an FPGA and PCI Bus based Readout Buffer for the Atlas Experiment

Abstract

This dissertation evaluates a readout buffer system for the ATLAS detector trigger and data acquisition system. ATLAS is a high energy physics experiment at the large hadron collider (LHC) with the aim to reach new frontiers in the investigation of the structure of matter. The high precision ATLAS detector produces a huge amount of data, 40 TByte/s, which is reduced by a three-level trigger system for online event data selection.

The readout buffer system acts as a data buffer while the second trigger level computes the trigger decision. ATLAS uses a sequential selection in the level 2 trigger which means that all event data required for the trigger decision is requested from the readout buffer component subsequently. This increases the complexity of the readout buffer device and its output event rate. Furthermore a region-of-interest (RoI) concept limits the amount of data necessary for the processing of one event inside the level 2 processor by defining the detector region with interesting data. Thus, approximately 10 kHz output rate have to be provided while feeding $\sim$1 kByte data packets with 100 kHz at the input.

The evaluated implementation of this readout buffer should be based on commercial "of-the-shelf" hardware. Thus a conventional Linux server PC with four PCI Bus segments has been used. This approach leads to uniformity in the ATLAS data acquisition system because all hardware beginning with the second trigger level is built of similar PCs. But a standard PC is not able to meet the previously mentioned requirements. Therefore it is extended (or accelerated) by a number of PCI based FPGA co-processor boards. Considering the above mentioned sequential selection and RoI concept, such a complex buffer component based on standard server PCs and FPGA co-processors has never been investigated before in high energy physics.

The FPGA co-processor is a simple component extending the PC for the time critical receiving and buffering of data. It is able to process data from four ATLAS detector links which allows the grouping of 12 to 16 links in one PC. Measurements show that this system is able to sustain the ATLAS requirements. Currently Linux OS, running on the PC system and handling the Gigabit Ethernet network I/O with the rest of the data acquisition system, is the main bottleneck. Improving this could be the subject of future investigations.

Evaluierung eines FPGA und PCI Bus basierten Auslesespeichers für das Atlas Experiment

Zusammenfassung

Die vorliegende Arbeit evaluiert einen PCI Bus basierten Auslesespeicher für das Datenerfassungssystem des ATLAS Detektors. ATLAS ist ein Hochenergiephysikexperiment am Large Hadron Collider (LHC), mit dem Ziel neue Erkenntnisse über die Struktur der Materie zu gewinnen. Der hochempfindliche ATLAS Detektor produziert eine große Menge an Ereignisdaten, etwa 40 Tbyte/s, die von einem dreistufigen Triggersystem in Echtzeit analysiert werden müssen.

Die in dieser Arbeit vorgestellte Auslesespeicherkomponente speichert die Ereignisdaten während die zweite Triggerstufe über deren weitere Verwendung entscheidet. Dabei werden alle zur Triggerentscheidung erforderlichen Daten nach und nach vom Auslesespeicher angefordert. Dies erhöht sowohl die Komplexität des Auslesespeichers wie auch die Anforderungen an seine Ausgangsrate. Um die Menge der notwendigen Daten zu reduzieren wird die Detektorregion, in der sich interessante Ereignisse befinden, bereits vom Level 1 Trigger erkannt und an die zweite Triggerstufe weitergegeben. Dadurch reduziert sich die Ausgangsrate auf 10 kHz, während Datenpakete in der Größenordung von 1 kByte mit 100 kHz an die Komponente übertragen werden.

Ein wesentliches Ziel ist die Implementierung des Auslesespeichers mit kommerzieller, weit verbreiteter „standart“ Hardware. Deshalb wurde ein konventioneller Linux PC mit vier PCI Bus Segmenten benutzt. Dies erhöht den Anteil an einheitlichen Hardware Komponenten im gesamten Triggersystem. Da ein solcher PC nicht in der Lage ist die hohen ATLAS Anforderungen an Eingangs und Ausgangsrate zu erfüllen, wurde er mit PCI Bus basierten FPGA Beschleunigerkarten erweitert. Unter Berücksichtigung der Komplexität der Komponente aufgrund der Verarbeitung von Level 2 Anfragen, ist dieser Ansatz für einen Auslesespeicher, bestehend aus Standart PC und FPGA Beschleuniger Hardware, einmalig in der Hochenergiepyhsik.

Die FPGA Beschleunigerkarte ist eine einfache Komponente, die den PC in zeitkritischen Aufgaben (Entgegennehmen und Speichern der Daten) unterstützt. Sie ist in der Lage, Daten von vier Detektorverbindungen gleichzeitig zu verarbeiten. Dies ermöglicht den Aufbau eines Auslesespeicher – PCs der 12 oder sogar 16 Detektorverbindungen verwalten kann. Messungen bestätigen, dass eine solche Komponente die ATLAS Anforderungen erfüllen kann. Der bandbreitenbegrenzende Faktor ist im Moment das Linux Betriebssystem. Dies hat unter anderem die Aufgabe die Gigabit Ethernet Verbindungen zu den Triggerprozessoren zu verwalten. Hier sind noch weitere Verbesserungen möglich.

Contents

Introduction

In today's picture of the structure of matter physicists still have a large number of open questions. Currently the physics of the ingredients of the matter, their properties, and mutual interactions is described by the Standard Model [PRSZ95]. It classifies particles in two basic types of fermions[1]: leptons and quarks and their interactions: electromagnetic, weak, and strong. Each particle of matter up to the atoms can be assembled and its properties explained out of the concept of leptons and quarks and the physics described by the Standard Model. But some assumptions of the Standard Model are not yet experimentally proved.

The electroweak interaction, which unifies the electromagnetic and weak interaction, is mediated by four gauge bosons[2]: three massive ($W^{\pm}$ and Z_0) for the weak interaction and one massless (Photon) for the electromagnetic interaction [PRSZ95]. The appearance of massive together with massless gauge bosons could be explained by the existence of a Higgs particle with a mass of up to 800 GeV [Col99]. This postulation of the Standard Model has not been experimentally proved yet.

If the existence of the Higgs particle can not be proved, a theory beyond the Standard Model postulates new supersymmetric structures at energies above 1 TeV with a whole set of new particles [Col99].

Only the Tevatron accelerator at Fermilab, with its centre-of-mass energy of 1.96 TeV, can currently reach these energy regions. To extend the opportunities for physics experiments in these energy regions, the LHC (Large Hadron Collider) project has been raised at the European Particle Research Laboratory CERN. LHC is a proton-proton collider with an energy level of up to 14 TeV well suited to investigate the above mentioned physical questions. Its interaction cross-sections for various physics events is increased by one order of magnitude compared to the Tevatron accelerator. This raises the probability to observe new phenomena around 1 TeV [Ses00].

Additionally a more precise measurement of the b quark and the latest observed top quark can be done with LHC allowing to measure rare decay channels and the CP violation of the $b\bar{b}$ decay (see [Ses00] for more details).

For a measurement of the proton-proton interaction a particle detector is needed. ATLAS [Col99] is one of four LHC detectors which will start in 2007. It has been designed to have sufficient physics performance to study the previously mentioned fundamental questions.

To achieve a reasonable precision a high effort is necessary. ATLAS consists of a number of sub-detectors to precisely record a high variation of particles created by the proton-proton

[1] Fermions are particles with spin 1/2, 3/2, 5/2 or more.

[2] Bosons are particles with spin 1, 2, 3 or more

scatter process and their tracks. Each event recorded by the detector generates approximately 1 MByte of data. Every 25 ns a new event appears in the LHC which leads to an event rate of 40 MHz, and a data volume of 40 TByte/s. Since this huge data volume can not be recorded, a pre-selection of physically reasonable and desirable events must be done within the detector's data acquisition system. This requires at least a partial analysis of the event which has to be done before the data is permanently recorded without disturbing further event detection. The detector's dead-time, the time in which no event data can be acquired, should not increase due to the online data analysis.

Thus staged hierarchical trigger systems are used for event pre-selection. Every stage receives event data with a lower rate and investigates it in more detail. Powerful algorithms have to be developed which analyse the event's physics and filter background or uninteresting events. A powerful computing architecture must be present together with facilities to move event data partly over long distances between the detector and the "counting rooms".

ATLAS uses three trigger levels to reduce the amount of data by a factor of 10^5 to a recordable event rate of 100 Hz [Col99]. Only the first of them processes events with custom hardware based on radiation-hard ASICs (Application-Specific Integrated Circuits) or FPGAs (Field-Programmable Gate Arrays). The other two trigger stages are based on large PC farms.

Due to the rapid increase of computing power and network technology in the past decade, more and more commercial products can be used even for time-critical tasks. This comprises computing nodes usable as trigger processors and network or data-bus technologies. Commercial "off-the-shelf" (COTS) components have a number of advantages compared with custom electronics components:

- Parts of the system can be exchanged against more powerful components without a redesign of electronics. This requires the compatibility of new components which is also a widespread issue in industrial applications.
- The manufacturers can give functional guarantees and support.
- The components are more cost effective then custom hardware.
- The components can be purchased in a short time. Thus recent hardware can be used at the experiment's start-up.

The ATLAS community has decided to use as many COTS components as possible within the trigger and data acquisition system [Gro03]. Nearly all components starting with the second trigger level are planned to be commercial products. Level 2 and level 3 trigger will use large PC farms. They are connected with Gigabit Ethernet [KH98] using commercial switches.

Another concept followed by the ATLAS trigger and data acquisition system is to keep the system as uniform as possible. The usage of similar PC technologies in the various subsystems reduces the administration effort and the amount of different software.

This thesis focuses on the boundary between the detector readout electronics and the commercial network to the second and third trigger level stage. The component present at this place, the readout subsystem (ROS), receives the detector data on 1600 point-to-point link connections with up to 100 kHz and forwards it to the trigger networks (Gigabit Ethernet) on demand.

Contrary to almost all other high energy physics experiments, the ATLAS level 2 trigger requests all event data required for the trigger decision from the ROS. Depending on the trigger

algorithm flow, event data from different sub-detectors is requested sequentially. Furthermore only data within a detector region, called region-of-interest and defined by the level 1 trigger, is asked. Both concepts reduce the network load at the input of the level 2 trigger, but complicates the ROS implementation and increases the event data output rate of this component. This makes the ROS architecture and implementation, evaluated within the thesis, a new and unique system in high energy physics.

Readout buffer components in the previous generation of high energy physics experiments have been less demanding. These components where based on the VME [3] bus in most cases combined with a large amount of custom hardware. Contrary, ATLAS follows the guideline to use as many standard, widespread commercial "of-the-shelf" components as possible. This thesis follows this idea and tries realising it within the ROS. The goal of this thesis is to implement a readout subsystem component with the following characteristics:

- Following the ATLAS community decision, the readout subsystem implementation should be based on commercially available PC hardware. Custom hardware development should be done as less and as simple as possible. Currently widespread PC systems provide 66 MHz / 64 Bit PCI buses which should be used for event data transport instead of the VME bus utilised by previous experiments.

- The readout buffer should concentrate many input links to one network output. This concept is suggested by the proportion between incoming data and the data required by the trigger farms. It helps to minimise the necessary amount of ROS hardware and level 2 and Event Building network switches and reduces the systems costs. But it puts also higher demands on each single component.

- The readout buffer should be the first instance of event building when a level 2 accept for an event occurs. It should perform partial event building on the data stored within all ROB's inside a crate and thus remove load from the ATLAS event building farm (SFI).

- The readout buffer should be tested in a realistic data-flow test environment.

The readout subsystem, presented in this thesis, uses a PC with multiple 66 MHz / 64 Bit PCI buses similar to the components used within the trigger farms. This generates uniformity between the components of the ATLAS data acquisition system.

Only the hardware interface to the detector front-end via SLink is done by a custom electronics component. Its first prototype has been based on the MPRACE FPGA co-processor [MPR]. MPRACE has been developed as a multi-purpose PCI based FPGA hardware. It is used in various physics and computer science applications:

- Acceleration of astrophysical simulations [Lie04].

- Image processing [Hez04].

- Trigger algorithm acceleration for High Energy Physics Experiments [Bro04] [HAK+04].

The board is flexible and can be equipped easily with four connectors for ATLAS detector readout links. This is an important issue to achieve cost-effectiveness and reduce the number

[3] Versa Module Europa

of PC components in the readout system. MPRACE is also a simple component comprising only an FPGA, memory, a PCI bridge, and the detector links.

The present thesis describes the development of a PCI bus based ATLAS readout buffer using the MPRACE FPGA co-processor. Test results show its usability. The thesis is divided as follows:

The first chapter introduces the ATLAS detector and its data acquisition system. It presents the ATLAS experiment and compares it to previous experiments. Furthermore this first chapter explains how detector data is acquired, processed, and recorded by the ATLAS trigger and data acquisition chain. The readout subsystem which is a central part of this chain gets also introduced here.

Chapter 2 focuses on the presentation of the ATLAS readout subsystem. It explains its task more detailed and shows the demands imposed by other ATLAS components. In the second part this chapter discusses various ROS implementation approaches and previous solutions.

Chapter 3 describes the MPRACE hardware and the FPGA firmware developed for the ROS application. It shows how data is received on four readout links, buffered, and made available on demand.

The software application running on the multi PCI bus host PC is described in chapter 4. The software design has major influence to the system performance. Chapter 5 shows test and measurement results and discusses them.

Finally chapter 6 discusses the approach presented within this thesis, points out possible improvements, and gives an outlook to readout systems in future experiments.

Chapter 1

Atlas Data Aquisition

The present chapter introduces the ATLAS experiment and the ATLAS data acquisition (DAQ) system. The ATLAS detector generates a huge amount of data which has to be pre-selected and recorded in real-time. This task is performed by the trigger and data acquisition system. It is constructed out of several processors, buffers, and data transportation elements. One of them is the readout subsystem (ROS). Its implementation is the main topic of the present thesis. The ROS position, task, and external boundaries in the ATLAS data acquisition system will be clarified here.

The first section of this chapter gives a general overview on the LHC accelerator and the ATLAS experiment. The second section introduces the ATLAS detector setup and the third section describes its data acquisition chain.

1.1 General Overview

To reach new frontiers in particle physics, experiments at a very high energy level are necessary. New particles, postulated by physicists as a prove or extension of the Standard Model [PRSZ95], are expected to appear at energies around 1 TeV [Col99]. The probability of physical processes to be investigated is very low [Col99] wherefore the rate of events has to be very high to observe sufficient target processes in a reasonable time.

Accelerator	Institute	Centre-of-Mass Energy [GeV]	Crossing [ns]	Event Rate [MHz]
PEP II	SLAC	10.58	4.2	238
KEK B	KEK	10.58	2	500
HERA	DESY	314	96	10.4
Tevatron (Run II)	Fermilab	1960	132	7.6
LHC	CERN	14000	25	40

Table 1.1: Centre-of-mass energy and event rates for various recent and future colliders [FRB+00] [Col95a] [Col02b] [Wol98].

These requirements characterise recent accelerator machines. Table 1.1 shows parameters of a selection of accelerators which are currently in use or under construction. The most

challenging accelerator LHC (Large Hadron Collider), which will start operating at 2007, will reach a new order of magnitude in centre-of-mass energy. By colliding two proton beams accelerated to 7 TeV a centre-of-mass energy of 14 TeV can be reached. This is substantially more then the present most powerful accelerator Tevatron at Fermilab and will extend the ability to study the physics mentioned above and in the introduction. LHC also provides an increase of the event rate compared to the Tevatron accelerator which raises the probability to observe rare physical events. Other accelerators (e.g. PEP II, KEK B, or HERA) have much lower centre-of-mass energies and are used for other physics studies.

To record rare physical events at these colliders, highly precise detectors are necessary. Typically those consist of a number of different sub-detectors to ensure that a large variation of particles can be observed precisely. Each sub-detector delivers the data in a high number of readout channels which transport either analog or digital data. Together with the high event rate provided by the accelerator this results in a large amount of data for each recorded event.

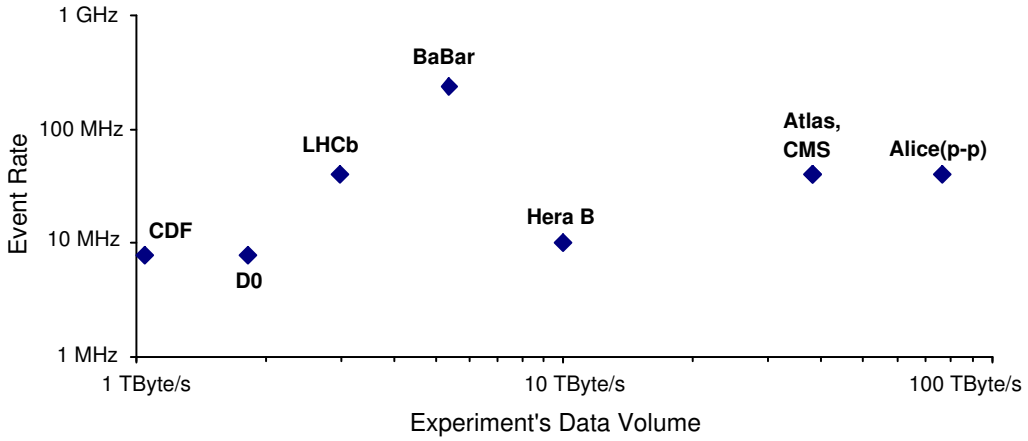

Figure 1.1: Overview of recent experiments at the PEP II (BaBar), Tevatron (CDF, D0), and HERA (HERA B) collider and experiments currently under construction at LHC (ATLAS, CMS, ALICE, LHCb) [Col95a] [Col02b] [Col02d] [Col02a] [Col02c] [Col04] [Col99] [Wol98].

Figure 1.1 shows the event rate against the generated data volume for various experiments at the above mentioned accelerators. The two Tevatron experiments CDF and D0 deliver a comparatively small data volume at an event rate of 7.6 MHz. The BaBar experiment, running at the PEP II accelerator, is one of the most demanding currently running experiments in terms of data acquisition and event processing. It delivers 5.4 TByte/s with an event rate of 238 MHz.

Compared to these already operating experiments, the four new detectors at LHC (ATLAS, CMS, LHCb, and ALICE) will run at a rate of 40 MHz which is more than four times the rate of Tevatron. Excluding LHCb, the data volume generated by these experiments will be one order of magnitude higher. ATLAS and CMS will deliver nearly 40 TByte/s, ALICE even 80 TByte/s. This will substantially increase the demand on the experiment's data acquisition compared to today's experiments.

But the basic operation of the data acquisition of all mentioned experiments is similar. Since the data volume of none of them can be permanently written to storage media like harddisk arrays or tapes, a pre-selection of events has to be done. To pre-select or "trigger" on single events, the analysis of at least parts of the detector event data is required to separate these from background or irrelevant events.

The latency for this process is a critical value for the efficiency of an experiment because no data can be taken meanwhile. Computing this trigger decision introduces an additional dead-time (a duration where no data can be acquired) into the data acquisition process [FRB+00].

To ensure that no physics process is lost during the trigger process, the dead-time has to be smaller than the rate of physical events to investigate. Its maximum allowable value has to be smaller than the event rate (25 ns in case of the LHC experiments). Thus trigger systems with a small execution time are required. Contrary high data reduction factors are needed to deal with the large event rates of modern accelerator machines. This requires a deep investigation of the event data with complex algorithms having long execution times.

To solve this contradiction, staged triggers are used which increase the data reduction factor, the complexity of the analysis algorithm, and the used fraction of event data with each stage [FRB+00]. The overall performance of a data acquisition system is influenced by:

- The efficiency of the analysis algorithm
- The performance of the trigger hardware
- The data transport between the trigger stages

The last two items mainly depend on technology choices and their efficient use inside the DAQ chain whereas the first one depends on the choice of the trigger algorithm and its implementation.

Trigger algorithms can be classified into [FRB+00] [Con84]:

- Fixed-flow triggers
- Variable-flow triggers
- Logical triggers
- Arithmetical triggers

Fixed-flow triggers need a fixed amount of time to compute an accept or reject decision. The processing time is independent of the event complexity. Most fixed-flow triggers are simple and fast algorithms used in the first trigger level. Contrary variable-flow triggers consume a variable amount of time which depends on the complexity of the event. This class of triggers contains counters, loops, and programs. Only a minimum and maximum execution time can be specified. Logical trigger are implemented with Boolean logic operations like AND and OR whereas arithmetical triggers use pulse height and counter values to calculate the trigger decision.

Variable-flow trigger processors can be distinguished into data-driven processors and program-driven processors [FRB+00]. Data-driven processors are pushed by the arriving of event data. The algorithm does not influence the quantity of data provided to the processor. The execution time does not depend on the event data. Contrary, in program-driven processors the program decides on the executed data. The execution time depends on the processed event data.

During the execution time of a trigger processor data must be stored in buffer elements. This requires memory which is large enough to keep all data arriving from the detector while the trigger decision is taking place. Furthermore, rejected events have to be sorted out within the buffer element.

Buffer elements attached to fixed-flow triggers can be implemented with simple pipelines (see level 1 trigger in [Col96]) or circular buffers (see level 1 trigger in [Col95a]) which are long enough to keep the event data for the fixed processing time. On trigger reject, event data leaving the buffer is simply not forwarded to the next trigger level.

Variable-flow triggers require a more flexible buffer mechanism. Depending on the execution time of the algorithm, only a maximum storage time is known per event. The buffer, typically fast SRAM, Dual-Port RAM [1], or similar (see e.g. [Col95a]), must be able to keep all events at least for this time. In case of an accept the event has to be localised inside the buffer and sent to the successive trigger level.

1.2 The ATLAS Detector

ATLAS is one of the experiments currently built at the new LHC accelerator at CERN (see 1.1). It is a general purpose detector to discover new physics at the boundary of current knowledge such as the Higgs boson or supersymmetrical extensions to the Standard Model. An international collaboration of approximately 2000 physicists from 33 countries participate in building up the experiment.

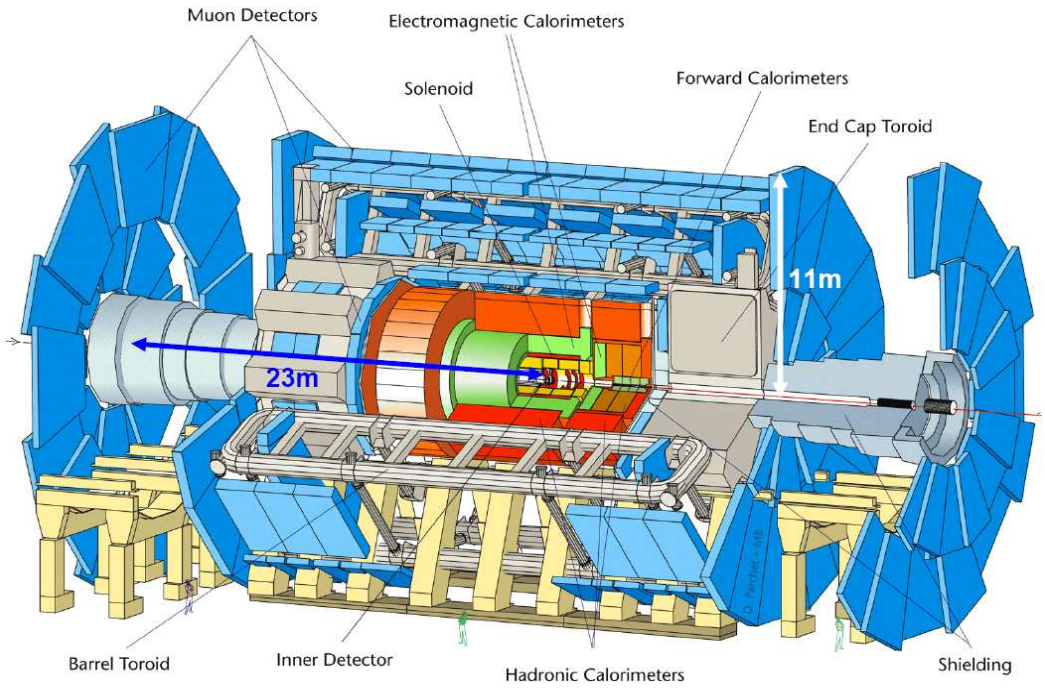

Figure 1.2: The Atlas Detector [Ses00].

To meet the physical goals the ATLAS detector has to recognize and track various particle signatures from electrons, γ, muons, jets, or missing energies [Sin00]. Therefore ATLAS comprises a number of sub-detectors which are placed in a number of layers around the proton beam interaction point. This is shown in Figure 1.2.

The area close to the interaction point, the innermost radius, is covered by the ATLAS inner detector which is divided again in three sub-detectors: Pixel, SCT, and TRT [Col99] [Ses00].

[1] Dual-Port RAM can be independently accessed for reading and writing.

It allows a precise recognition, track recording, and momentum measurement of particles. The whole inner detector is embedded in a solenoid magnet having a 2 T central field. External to this solenoid are the electromagnetic calorimeter to identify and measure electrons and photons, and the hadronic calorimeters to identify and measure hadronic particles. The outermost parts of the detector contain the muon spectrometer for a precise muon track recording [Col99] [Ses00].

All these sub-detectors deliver the data about the recorded events in up to $150 \cdot 10^6$ data channels resulting finally in approximately 1 MByte data for each event when permanently recorded. With an event rate of 40 MHz ATLAS generates a data volume of 40 TByte/s which has to be handled by the data acquisition chain.

1.3 The ATLAS Data Acquisition Chain

As already described in the previous chapter, the precision of the ATLAS sub-detectors results in a high data volume processed by the data acquisition with the experiment's interaction rate of 40 MHz. An overall data of 40 TByte/s is produced by ATLAS. To reduce this to a rate of 300 MB/s, which can be written to a permanent storage medium, an efficient trigger system is needed. This has to be flexible enough to adapted it to the various physics phenomena targeted by ATLAS.

Section 1.1 already introduced the concept of staged triggers in high energy physics to reduce the huge amount of data delivered by the experiment's detectors. ATLAS makes use of this concept too. Its DAQ system is implemented by three trigger levels, which reduce the event rate by a factor of 10^5.

Figure 1.3 shows the ATLAS trigger and DAQ system. Beside the three trigger levels various other modules are integrated into the ATLAS DAQ. They are described in the following sub-sections.

Level 1 Trigger

The level 1 trigger examines event data from the ATLAS calorimeter and muon sub-detector to form a trigger decision. The trigger algorithm analyses threshold information of energy and momentum to find signatures of possible particles. This level 1 arithmetic trigger (see 1.1) is implemented using synchronous, pipelined, parallel processors driven by the LHC 40 MHz clock. ASIC[2] and FPGA[3] technology is used for the processors which are partly placed very close to the detector to prevent signal synchronisation problems due to the wide spread area of ATLAS.

Computing the level 1 decision takes 2 µs, every 25 ns a new calculation must be initiated [Gro98a]. During this decision process the event data of all sub-detector is stored inside fast pipeline memories. On accept, event data is passed by the readout drivers to the readout buffers. Both are described later in this chapter.

The rate of level 1 accepts can be estimated by simulating the physical processes in the LHC proton-proton interaction together with the behaviour of the level 1 trigger processors [Gro98b]. The result is a table which assigns various particle signatures an estimated rate. This table is called "Trigger Menu" and gives a detailed report on the expected level 1 accept

[2] ASIC: Application-Specific Integrated Circuit
[3] FPGA: Field Programmable Gate Array

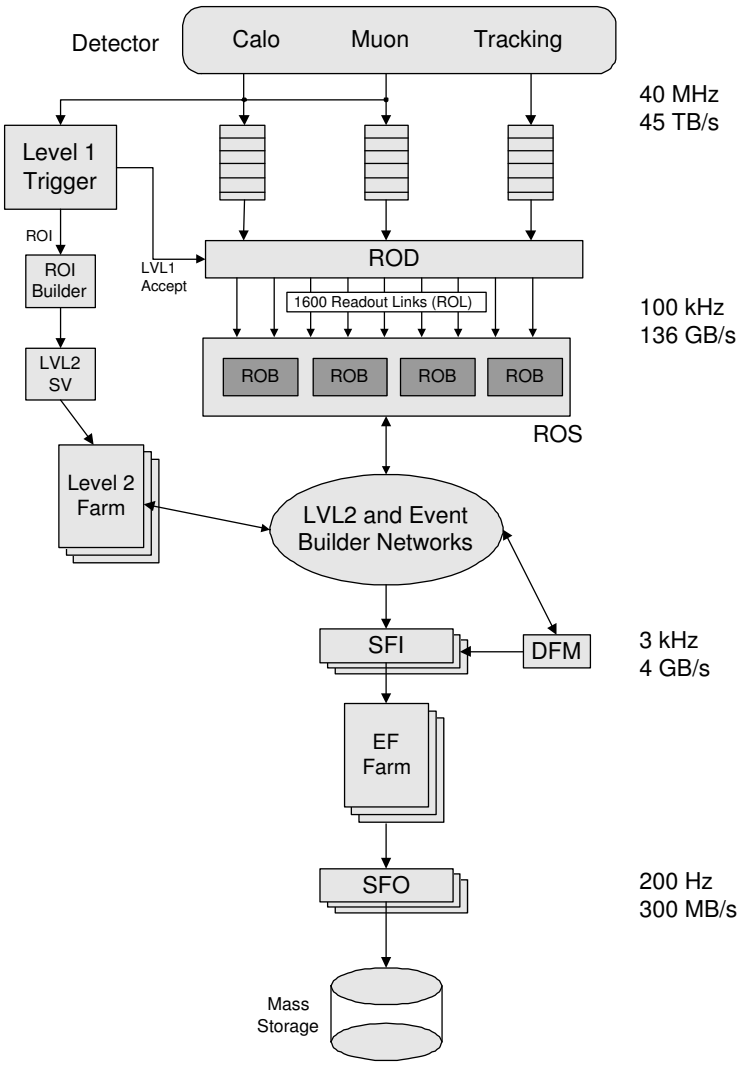

Figure 1.3: The ATLAS trigger and event data acquisition. [Col99]

rate and how it distributes on the various physical events. Figure 1.4 shows the most recent trigger menu for the initial LHC luminosity[4] of $2 \cdot 10^{33} \mathrm{cm}^{-2}\mathrm{s}^{-1}$ [VVT03].

According to this trigger menu a level 1 accept rate of 25 kHz is expected. Since this estimation does not contain any safety factor, a larger accept rate must be targeted within the trigger development. It has been decided to design the level 1 trigger and also all other instances of the DAQ chain for supporting an accept rate of 75 kHz. Furthermore an extension to 100 kHz is foreseen for a later experiment upgrade [Gro98a].

On accept, the level 1 trigger passes the detector area coordinates in which particles have been found to the next trigger level. This area, called region-of-interest (RoI), is defined as a pair of angles which describe a cone inside the ATLAS Detector with its top in the experiment's interaction point. It clearly shows the area where the next trigger stage (level 2) has to run the search algorithms. This substantially reduces the amount of data required by the level 2 trigger (see the level 2 trigger subsection within this section).

[4]The luminosity is the number of particles per time and per area within an accelerator beam. [FRB+00]

Selection	2 10^{33} $cm^{-2}s^{-1}$
MU20	0.8
2MU6	0.2
EM25I	12.0
2EM15I	4.0
J200	0.2
3J90	0.2
4J65	0.2
J60 + xE60	0.4
TAU25 + xE30	2.0
MU10 + EM15I	0.1
Others (prescaled, calibration, ...)	5.0
Total	~ 25

Figure 1.4: Level 1 trigger menu for the initial LHC luminosity of $2 \cdot 10^{33}\text{cm}^{-2}\text{s}^{-1}$. The notations of particle signatures are explained in [Col99]. The recent values have been obtained from [VVT03].

Readout Driver (ROD)

In case of a level 1 accept decision the event data is passed over to the readout drivers (ROD). 1600 Readout drivers organised in a number of VME crates collect the event data from the detector's data channels. Their task is to provide a general interface from the detector to the DAQ system, pre-format the event data by adding a header and a trailer (see section 1.4) or performing zero suppression [Gro98a], and de-randomizing the event data. The latter ensures that event data is sequentially transferred to the DAQ system. Furthermore a small buffer with a size in the order of 16 words is present.

Readout Links

Altogether 1600 readout links (ROL) transport the event data over a distance of up to a few hundred meters from the RODs to the readout buffers. The used technology is SLink a custom unidirectional point-to-point link standard developed at CERN [BMvdB97]. Its concept is shown in Figure 1.5.

SLink defines the interface between link implementation hardware and the user electronics. Data is always transferred in one direction from the SLink source card (LSC) to the SLink destination card (LDC). The interface to the source card prescribes 32 data signals and a clock signal with up to 40 MHz which have to be supplied to the LSC. Data arrives on a 32 Bit bus on the receiver (LDC) side together with the 40 MHz clock. Furthermore an error signal is raised when data has been corrupted during transmission.

The return channel (from the LDC to the LSC) plays a minor role and comprises 4 data bits and a flow control signal. The latter can be used to signal a link full condition to the user hardware.

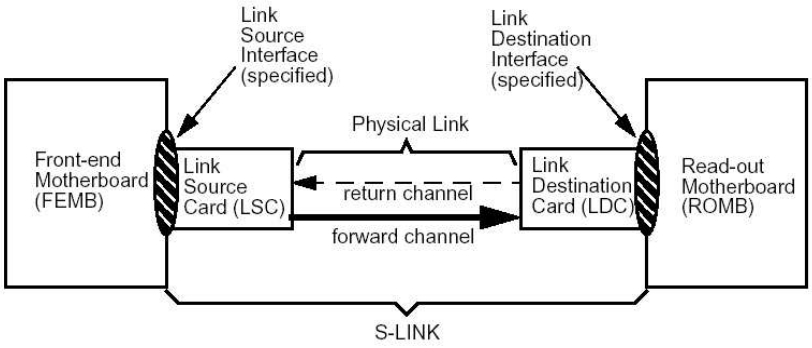

Figure 1.5: The CERN SLink standard. Only the link source and destination interfaces are specified together with the outline of the interface cards. The physical implementation is application dependent [BMvdB97].

The link technology is not fixed in the SLink standard. It has to be implemented on a mezzanine which uses the electrical signals from the SLink connector. The outline of the mezzanine and the SLink connector are determined by the SLink specification. Various physical link implementations are available: electrical parallel, electrical serial, and optical [vdBH].

The ATLAS experiment uses an optical implementation, which is called HOLA (High Speed Optical Link for ATLAS), between the RODs and the ROBs. It is based on a 2 GBit/s optical transceiver [RvdBH] with a throughput of up to 160 MByte/s. The source and destination cards use an FPGA to implement the SLink functionality. This FPGA controls the optical transceiver and the serialiser to transfer the data and control words and communicates with the SLink user through the standardised SLink connector.

Level 2 Trigger

The level 2 trigger is the second instance for trigger decisions in the ATLAS DAQ chain. It takes the RoI information from level 1 which has been collected from the level 1 trigger processors by the RoI builder. The trigger is implemented within a large farm of dual-CPU PCs connected over a Gigabit Ethernet network. A number of supervisor PCs control the level 2 farm and distribute the trigger tasks and the RoI information collected by the RoI builder.

Level 2 is based on program driven trigger processor (see 1.1). Furthermore it requests the required event data from the readout buffers. This might happen in several steps depending on the trigger decision algorithm ("sequential selection"). Each event accepted by the level 1 trigger is investigated by one level 2 processor. The RoI concept prevents that all event data is requested by the trigger with a rate of 100 kHz. Since the data of only a small number of readout buffers, covering the desired area of the detector, is required, the amount of data to the level 2 processor is very limited. In average the level 2 trigger requests only 7% of all event data arriving from the detector per ROL. This corresponds to 7 kHz in average for a level 1 rate of 100 kHz per readout buffer.

The level 2 farm size can be obtained from a detailed model of the ATLAS DAQ chain. This is done within a C++ program [Ver] simulating the level 2 trigger, succeeding stages, and data transport and assembling delays. It is expected by this model that 496 PCs will be needed to process all events accepted from level 1 with the above mentioned trigger menu. The calculation is based on the assumption to have 8 MIPS (million instructions per second) CPUs

within the PCs available. The mean latency for computing the trigger decision is expected to be approximately 10 ms [Gro03] with a large variation from event to event. In the worst case single events may require many times the mean latency for decision while events may also be processed within a much smaller time. Finally 2-3% of all events accepted from level 1 also pass the level 2 trigger.

Readout Subsystem (ROS)

The readout subsystem (ROS) stores the event data received from the readout drivers during the level 2 decision. For each of the 1600 readout links, coming from the RODs, one readout buffer (ROB) has to be provided which is large enough to cover the level 2 latency of 10 ms with its large variation for single events.

Besides buffering of data, the ROS also serves requests for RoI data coming from the level 2 farm via Gigabit Ethernet. In case of a level 2 accept all data has to be passed to the event building process with a second Gigabit Ethernet interface. Thus approximately 10% (7% RoI requests + 2-3% level 2 accept) of all event data coming from the ROD leaves the ROBs towards the level 2 and event building farms.

Due to the large difference between input and output the main architecture concept for the ROS implementation is to group a number of ROL inputs and buffers to one network output. This guides the architecture design choice for the ROS which will be explained in detail within the next chapter.

Sub-Farm Input

On level 2 accept all fragments inside the 1600 ROBs are merged into one event data block for a final investigation by the third trigger level, the event filter. This merging is done within the sub-farm interface (SFI). A farm of approx. 59 PC's [Gro03], all of them interconnected via a Gigabit Ethernet switching network, is responsible for the collection of event data fragments from all ROBs. The farm merges them into one and passes this data block over to the event filter processors. The whole process is initiated and controlled by the dataflow manager (DFM) which assigns each accepted event one SFI node to perform the event building operation.

The SFI farm must reach an overall event building rate which is equal to the level 2 accept rate of 2-3 kHz (having a level 1 accept rate of 100 kHz). This requirement determines the farm size of 59 PC's which has been again calculated using the ATLAS DAQ model from [Ver]. Another value influencing the SFI farm size is the number of event data fragment sources. These are the ROB outputs. Their number depends on the proportion between ROB inputs and outputs. This will be explained in detail in the next chapter.

Event Filter

The event filter (EF) is the final trigger level of the ATLAS DAQ. It analyses the whole event in detail thereby reducing the data rate by a factor of 10-20. This is done using modified versions of offline algorithms which are normally used for the analysis of the already recorded data.

The EF is again implemented in a PC farm connected to the SFI farm via a Gigabit Ethernet switching network. On event filter accept the data leaves the DAQ chain through the sub-farm outputs (SFO) to be permanently recorded on a disk array. The final event rate at the SFO is

in the order of 200 Hz. A data stream of ∼200 MByte/s has to be written to the permanent storage medium.

1.4 The ATLAS Data Acquisition Event Format

The event data running through the previously described ATLAS data acquisition chain follows certain data format guidelines. These are specified in [BFM+04]. The data format has been developed to support various features (from [BFM+04]):

- The event data format contains information redundancy to allow self consistency checks of the event.
- The event format provides information whether the event has been corrupted during transmission within the data flow.
- The event formatting information does not exceed 20% of the typical ATLAS event data size.
- The event format is modular.
- The event format facilitates the identification of fragments.

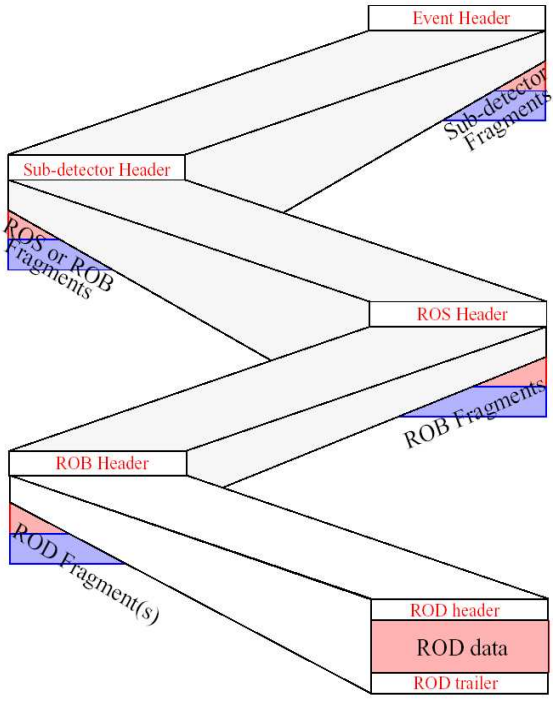

Figure 1.6: The ATLAS event format [BFM+04].

Starting at the readout drivers (ROD) each instance of the ATLAS DAQ adds a header, containing a set of information, to the event data. This is shown in Figure 1.6. The complete event consists out of a number of sub-detector fragments. Each of them again consists out

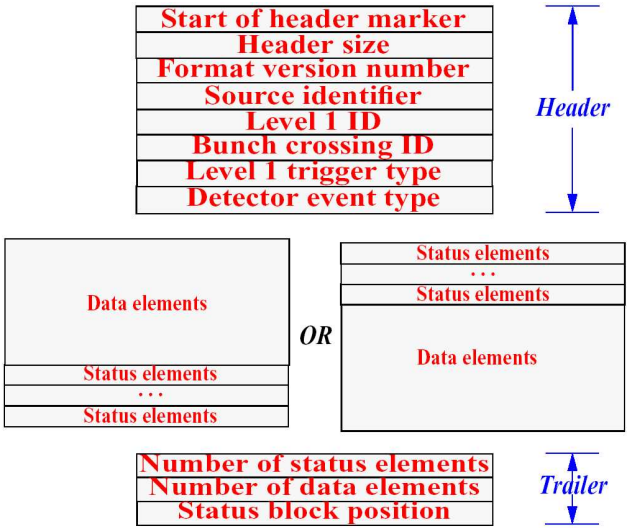

Figure 1.7: The ROD event fragment format [BFM+04].

of multiple ROS or ROB fragments. A ROS fragment is built on the level of a ROS, which groups a number of readout buffers (ROBs). Thus a ROS fragment contains a number of ROB fragments. The ROB fragment is built on the level of the readout buffer. Each ROB fragment contains one ROD fragment, supplied and formatted by the ROD out of the ATLAS raw event data.

Firstly, the ROD adds a header and a trailer to the data containing information about data sizes, event source, the level 1 ID (a unique number within a certain time assigned by the level 1 trigger), the bunchcrossing ID (a unique number within a certain time for each LHC event), and some information why level 1 has accepted this event. This ROD data format is shown in Figure 1.7.

Fragment Type	Start-of-Header Marker
ROD	0xee1234ee
ROB	0xdd1234dd
ROS	0xcc1234cc
Sub-Detector	0xbb1234bb
Full Event	0xaa1234aa

Table 1.2: The Start-of-Header Marker encodings for the various event fragment types up to the full event [BFM+04].

Secondly, the readout buffer adds a similar header to the event data containing redundant information from the ROD header and status information. The result is a ROB fragment. This continues on the level of the ROS (a composition of ROB fragments), the sub-detector (containing all ROS fragments of one sub-detector), and finally the complete event which is built by the SFI (see the previous section).

The first word of each header contains a value defining the type of the header. This allows the identification of the instance which has built the fragment. Table 1.2 shows these “Start-of-Header” markers used within the event data format. Upon this marker the data format and type of contained fragments can be determined.

Chapter 2

The Atlas Readout Subsystem Architecture

This chapter describes in detail the ATLAS readout subsystem (ROS) for which this thesis will present an implementation approach. The first section summarises the external dependencies of the ROS component. The second gives an overview of previous readout buffer implementation approaches and presents ATLAS ROS architectural choices.

2.1 Requirements for the ATLAS Readout Subsystem

The readout subsystem (ROS) is one of the central devices in the ATLAS data acquisition chain. It collects all data accepted by the level 1 trigger and makes them available to the level 2 trigger on demand. As it is placed in the main ATLAS data flow, it has to fulfil a number of requirements in terms of performance and usability.

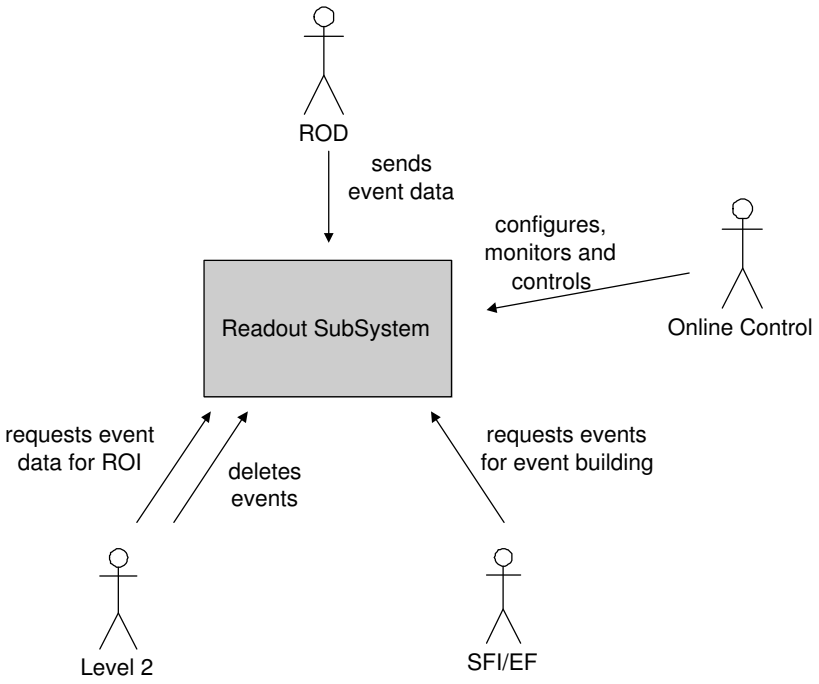

Figure 2.1: The ROS dependencies [RMR02].

Figure 2.1 shows the dependencies of the ATLAS ROS and external DAQ components. Input data is coming from the readout drivers (ROD) while the level 2 trigger and the sub-farm input (SFI) are located at the ROS output. Finally the ATLAS online control configures and controls the ROS component. All external dependencies are presented in more detail below complemented with a quantification of the load.

Readout Driver

The readout driver pushes event data, accepted by the level 1 trigger, into the readout buffer system. 1600 readout links, conform to the HOLA SLink implementation [RvdBH], are used to transfer the data of the seven sub-detectors to the ROS. Each of these links has a nominal bandwidth of 160 MB/s. Data arrives with the level 1 accept rate of up to 75 kHz upgradeable to 100 kHz (see 1.3).

Each of the sub-detectors delivers a different event fragment size via the SLink connection. This size is determined by the quantity of physical processes and also by the luminosity of the LHC accelerator. It can be estimated by a detailed examination of the physical processes inside the ATLAS detector. Table 2.1 summarises the results for the initial LHC luminosity of $2 \cdot 10^{33}\, cm^{-2}\, s^{-1}$ and the design luminosity of $1{\cdot}10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$ which will be achieved later by the experiment.

Sub-Detector	Number of ROLs	Initial Luminosity [Byte]	Design Luminosity [Byte]
Pixels	120	200	500
SCT	92	300	1100
TRT	232	300	1200
EM Calorimeter	740	752	752
Hadron Calorimeter	88	752	752
Muon precision	192	800	800
Muon trigger	48	380	380
CSC	32	200	200
Level 1 trigger	56	1200	1200
Total	1600	1.0 MByte	1.3 MByte

Table 2.1: The number of readout links (ROL) and the fragment sizes per link for the ATLAS sub-detectors. The experiment starts at the initial luminosity of $2{\cdot}10^{33}\mathrm{cm}^{-2}\mathrm{s}^{-1}$. Later the design luminosity of $1{\cdot}10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$ will be reached [Col99].

These numbers are based on the assumption that the RODs perform a zero suppression and compression of data for some sub-detectors (especially the electromagnetic calorimeter). This has not yet been decided. Thus data fragment sizes of up to 1.6 kByte may be possible for the electromagnetic calorimeter [Fra04].

Level 2 Farm

The level 2 trigger supervisor receives on level 1 accept a set of RoI definitions for further investigation within the level 2 farm. It assigns a farm processor which starts to collect event data fragments from the ROS. Due to the RoI principle of the level 2 operation, only event

data from a specific area of the detector is required (see section 1.3). Furthermore the data from a number of sub-detectors is taken into account and data may be requested subsequently in several steps by the trigger algorithm.

Thus estimating the event data rate which each ROB inside the ROS has to deliver to the level 2 farm needs a model of the level 2 algorithms and the knowledge of the ATLAS trigger menu. This modelling effort is done within the ATLAS modelling group [Atl]. A complete model of the ATLAS DAQ chain, starting with the level 1 trigger, is done within this group [Ver]. Its major assumptions are:

- The trigger menu from Figure 1.4 determinating the level 1 accept rate.
- The event fragment sizes from Table 2.1.
- Dual CPU PCs with 8 MIPS processors for the level 2 farm.
- A Gigabit Ethernet network to connect the level 2 farm with the ROS using the UDP/IP protocol [Pos80] [Pos81].
- A custom transport layer protocol on the top of UDP defined by the ATLAS DataCollection Group [Hau03].
- A bandwidth limit of 60 Mbyte/s to the SFI event building farm.

Sub-Detector	Low Luminosity		Design Luminosity	
	Average [kHz]	Max. [kHz]	Average [kHz]	Max. [kHz]
Pixels	0.67	0.98	0.99	1.42
SCT	0.53	0.74	0.79	1.08
TRT	0.19	0.22	0.04	0.05
EM Calorimeter	2.03	7.42	1.74	6.34
Hadron Calorimeter	1.36	1.99	0.89	1.30
Muon precision	0.10	0.20	0.29	0.57
Muon trigger	0.22	0.30	0.62	0.86

Table 2.2: Estimation of the level 2 request rates per readout buffer. One readout buffer stores the data from one ROL [Col99].

An estimation of the level 2 request rate per readout buffer (ROB), obtained using this model, is summarised in Table 2.2. One ROB stores the data arriving from one ROL. Table 2.2 presents a maximum and average request rate per ROB depending on the sub-detector. Together with the event sizes shown in Table 2.1 each ROB generates the data volume listed in Table 2.3.

Both tables show that the most request load is generated on Buffers handling the data from the calorimeter sub-detector. Up to 7.43 kHz or 5.58 MByte/s flow from the readout buffer to the level 2 farm at initial luminosity. This is caused by the level 2 trigger algorithm which needs in most cases data from the calorimeter first. At design luminosity this value decreases.

The readout buffers have to keep the event data at least for the level 2 processing time which is expected to be up to 10 ms with a large event by event variation (see 1.3). With an input rate of 100 kHz and the event fragment sizes listed in Table 2.1 a minimum memory size (without any safety factor) of 1200 kByte must be available in each ROB.

Sub-Detector	Low Luminosity		Design Luminosity	
	Average [MByte/s]	Max. [MByte/s]	Average [MByte/s]	Max. [MByte/s]
Pixels	0.13	0.20	0.50	0.71
SCT	0.16	0.22	0.87	1.19
TRT	0.06	0.07	0.05	0.06
EM Calorimeter	1.53	5.58	1.31	4.77
Hadron Calorimeter	1.02	1.50	0.67	0.98
Muon precision	0.08	0.16	0.23	0.46
Muon trigger	0.08	0.11	0.24	0.33

Table 2.3: The bandwidth needed for data requests per readout buffer. One readout buffer stores the data from one ROL [Col99].

SFI and Event Filter

On level 2 accept the event data of all ROBs within the ROS is requested by the sub-farm input (SFI) PCs for event building. In average 3% of all events accepted by level 1 are left at this point. With a level 1 accept rate of 100 kHz, this adds another 3 kHz to the output rate of each ROB. Depending on the sub-detector, between 0.6 and 3.6 Mbyte/s additional bandwidth is used per ROB for this process. All event data rejected by level 2 will no longer be used within the ATLAS DAQ and is deleted at the ROB level.

Online Control

Finally the ATLAS online control system is responsible for ROS configuration and control. This requires the ROS to provide an implementation of a generalised software interface. Having this, the online control can pass configuration data and is able to switch between various run levels. Errors are also reported through this interface [Gro03].

Event monitoring may also be requested by online control. This means that the ROS collects and delivers event data by a configurable set of parameters [Gro03]. Currently requirements and implementation details for event monitoring on the level of the ROS are not completely specified.

2.2 Readout Buffers in Other Experiments

The staged trigger architecture is widespread at various currently operational high energy physics experiments. The trigger strategy is in most cases similar. Only the number of trigger levels and their implementation differs from experiment to experiment.

The currently most demanding are CDF and D0 at Tevatron (Run IIb) and BaBar at PEPII (see 1.1). The data acquisition and trigger strategy of the two Tevatron experiments is very similar. There are three trigger levels to reduce the detector event data. The first two use custom hardware operating on partial event data. The third level, a PC farm, analyses the complete event after it has been reconstructed ([Col02b] [Col02d]).

The BaBar experiment has only two trigger levels. The first level trigger is built with custom hardware and on accept the event building process immediately starts. Finally a PC

farm analysis the complete event as it is done at D0 and CDF [Col95a].

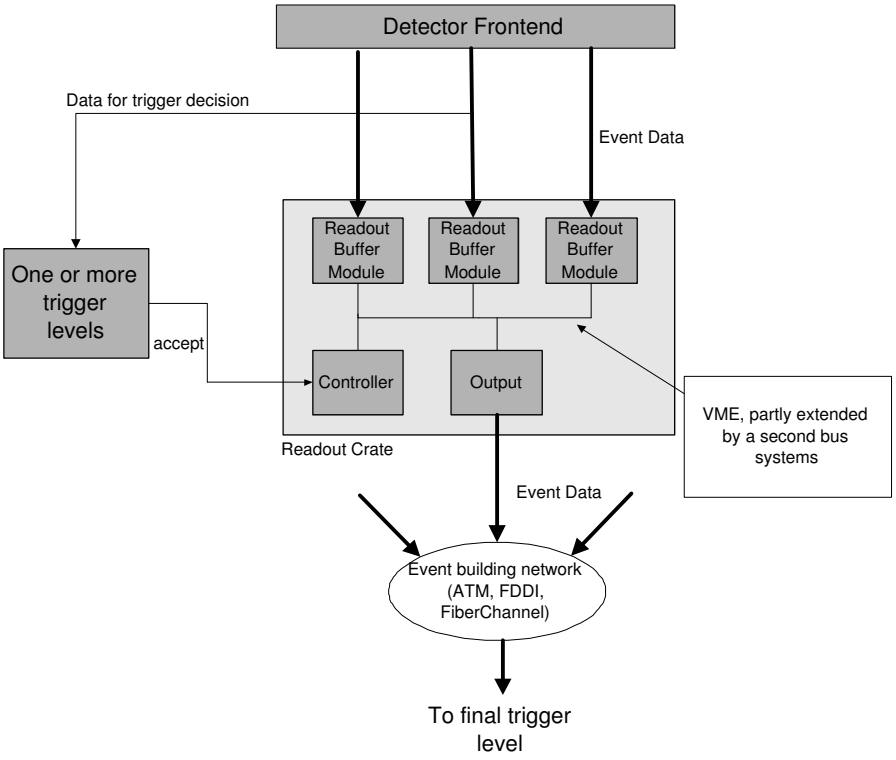

Figure 2.2: The generalised and simplified event data flow through trigger and readout buffer components used by many experiments. A similar system can be found in the data acquisition of CDF [Col02b], D0 [Col02d], BaBar [Col95a], [RMR02], and others.

In all experiments a readout buffer component stores the detector data prior to event building. The event data fraction required by the trigger is previously branched and directed into the trigger processor(s). If the level 2 (CDF, D0) or level 1 (BaBar) has accepted the event, data leaves the readout buffer component for event building and further analysis.

This data acquisition and trigger strategy is shown in Figure 2.2 with the focus on the readout buffer component. Its main operation in all above mentioned experiments is to read the event data from the detector, buffer it, and forward it on trigger accept. Therefore VME bus crates with a number of custom modules or single board computers are present. These readout modules provide the detector link destination and store the event data. On accept, data is transported over the VME bus to the output component and sent to the event building network. Partly the bandwidth of VME is extended by the use of a custom bus system (e.g. the “Magic Bus” in D0).

All LHC experiments, except ATLAS, follow the same trigger and data acquisition strategy. The main difference to previous experiments is the extended use of standard, “of-the-shelf” components. PC farms are used already in early trigger stages; in most cases already for level 2.

This is continued in the design of the readout buffer components used by the LHC experiments ALICE [Col04] and CMS [Col02c]. In all cases the bus system has changed from VME to PCI and the crate is a standard, industrial, high performance PC.

The readout modules are based on FPGAs and receive the event data on custom, high performance links. The ALICE readout buffer forwards this data directly into the memory of

the PC. At CMS memory is attached to the FPGA to store the event data received from the detector.

Only LHCb has decided to use a Gigabit Ethernet network approach instead of the PCI bus. There, a number of readout buffer modules handle the custom detector link and buffer the event data. Each of them has a Gigabit Ethernet port to deliver the data on trigger accept.

The ATLAS trigger and data acquisition is different to all of the above mentioned experiments. The level 2 trigger operates as a program-driven trigger processor (see 1.1). Furthermore it uses "sequential selection". This means that only that event data, required for the trigger decision, is requested from the readout buffers. Another specific characteristic is the RoI concept which reduces the amount of necessary data for the level 2 trigger decision. This is only rarely found at high energy physics experiments.

Sequential selection has a major impact on the readout buffer component since it must supply the event data on request. This puts additional complexity to the design of the readout buffer compared to all previously mentioned experiments.

One already operating experiment with a similar mechanism inside the level 2 trigger is HERA B. This experiment has a four level trigger architecture. The first level is implemented completely in custom hardware. Starting from trigger level 2 all computations are done by PC farms.

Figure 2.3 shows the HERA B event data flow with the focus on the level 2 trigger and buffer.

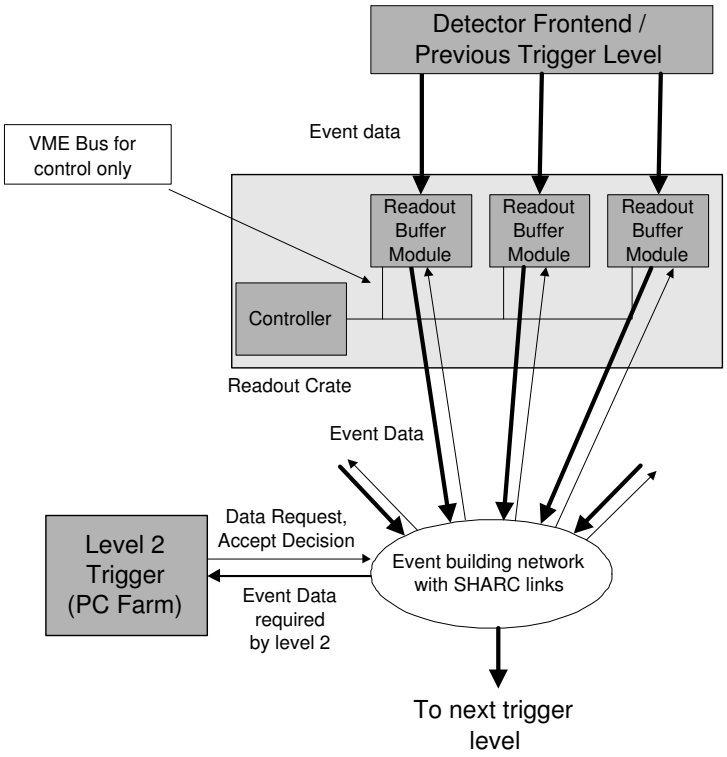

Figure 2.3: The HERA B event data flow with the focus on the second trigger level. This uses similar mechanisms than ATLAS [Wol98] [Col95b].

Detector data accepted from the first trigger level is stored in the readout buffer with a rate of 50 kHz. The readout buffer is based on SHARC DSP processors placed on VME crate modules. Each processor provides a number of 40 MByte/s serial links used for event

data transport towards the second trigger level and event building. The second trigger level requests and receives all required event data by theses SHARC links. Approximately 3% of all data is required for level 2. On accept, event building starts by sending the event data fragments from the various SHARC processors to one level 3 trigger node. This is done with the level 2 accept rate of approximately 2 kHz [Wol98].

Thus the ATLAS trigger and data acquisition with its level 2 trigger requesting the required event data from the readout buffers has a unique position within the high energy physics experiments. The architecture can not be compared to other LHC or Tevatron experiments. Only the HERA B trigger has similar conditions, but with a four times smaller detector event rate and using different technologies (SHARC links instead of Gigabit Ethernet). This comprises also the ATLAS readout subsystem. Contrary to all other experiments it has to provide the RoI event data to the level 2 farm on demand. This complicates the architecture and implementation and makes it a new development.

2.3 ATLAS Readout Subsystem Architecture

2.3.1 Architectural and Technology Choices

ROS grouping

For the ATLAS readout subsystem various architecture approaches have been discussed in the ATLAS community. Due to the high input and the low output rate, the concentration of a number of input links to one network output has been accomplished as a basic rule to all approaches. This principle can also be found by all other high energy physics experiments (see 2.2).

The number of combined input links are limited by the Gigabit Ethernet bandwidth and varies for different ATLAS sub-detectors. A first evaluation, only based on a simple comparison between the required input and output bandwidth (presented in section 2.1), estimates that between 14 and 70 inputs can be connected to one Gigabit Ethernet output depending on the sub-detector. This approach drastically decreases the number of Ethernet links between the ROS and the level 2 and SFI farm.

An extension to the combining of input links to one output is the possibility of local, partial event building inside the ROS. This means that a number of ROB packets are merged into one ROS packet as defined in the ATLAS Data Format specification 1.4. The result is a reduced number of messages and data packet frames exchanged between the SFI and ROS.

Estimations with the paper-model from [Ver] (already introduced in the previous chapter) show that this can reduce the SFI farm size. Without merging of fragments inside the ROS, 83 PCs have to be present in the SFI farm to meet the 2-3 kHz event building rate [1]. With a grouping of 16 inputs to one output and partial event building this decreases to 28 PCs. Thus approximately 50 PCs can be saved in the SFI farm. Also the level 2 farm decreases by a 5-8 PCs due to the reduced number of messages (again with a grouping factor of 16).

[1] The SFI farm size has been obtained from the model contrary to the values presented in the TDAQ TDR [Gro03] with a Gigabit Ethernet bandwidth of 100 MByte/s. The original bandwidth was assumed to be 60 MByte/s (see section 2.1).

Technology Choices

Regarding the other experiments described in section 2.2 a number of implementations for the ATLAS ROS can be considered and discussed.

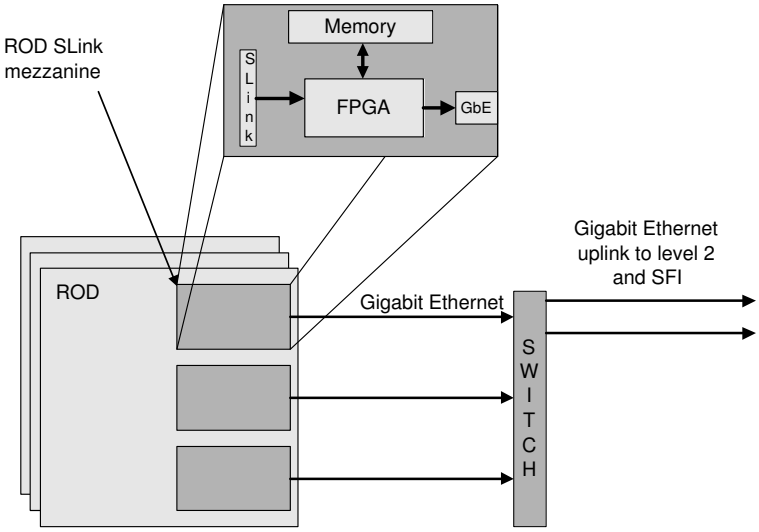

Figure 2.4: The ROB-on-the-ROD scenario. A readout buffer mezzanine replaces the SLink implementation mezzanine on the ROB VME board. [BBF+02]

One of the simplest scenarios for ATLAS uses one readout buffer module per detector ROL equipped with SLink input, some memory, an FPGA, and Gigabit Ethernet output. This is similar to the CMS approach (or HERA B, but without the SHARK links).

One possibility is to place this hardware directly on the readout driver replacing the foreseen SLink mezzanine hardware (see 1.3). This simple approach is called "ROB-on-the-ROD" concept [BBF+02](see Figure 2.4).

The fragments, sent by the readout driver (ROD), are buffered inside the on-board memory and delivered via Gigabit Ethernet to the level 2 or SFI event builder farm. A concentrator switch combines the Ethernet links of a number of modules into one or two up-links to the level 2 and SFI farm.

In a similar scenario the readout buffer modules could be placed on 3, 6, or 9 HU[2] boards hosted in a simple crate with only a power supply. Up to 5 buffers could be implemented on a 6 HU board each having one HOLA SLink input and one Gigabit Ethernet output, up to 8 on a 9 HU board and only 2 on a 3 HU board. The number of links is limited by the available space on the front of the board. One link (either Ethernet or HOLA SLink) is estimated to allocate 2 cm of the board's front panel.

Still one Gigabit Ethernet link per ROL is present which is grouped by a number of concentrator switches into a number of up-links towards the trigger and SFI farms. All scenarios mentioned up to now do not merge event data fragments. The number of Ethernet messages and thus the SFI farm size is not reduced.

This can be taken into account by increasing the logic of the above mentioned boards. This may increases the number of ROLs handled and reduces the number of messages between the level 2 / SFI farms and the ROS. Out of space considerations up to 8 links could be placed on a

[2]HU is a unit for the height of electronic boards. Each HU is equal to 1.75 inches. Thus 9 HU corresponds to 400.5 mm, 6 HU to 266.7 mm, and 3 HU to 133,35 mm.

6 HU, 14 on a 9 HU, and 3 on a 3 HU board while always 2 Gigabit Ethernet links are present. This approach requires a completely custom ROS design which is not intended by the ATLAS community.

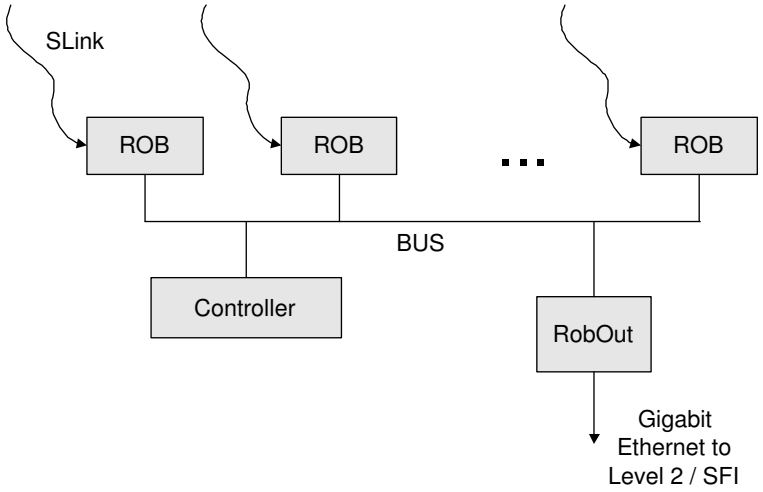

Figure 2.5: ROS Module with local event building via a standard bus (VME, PCI). A number of ROBs receive data from the ROL, buffer it, and send it to the bus on request. The network interface is attached to the module bus too.

Many other experiments (see 2.2) use commercial bus technologies like VME or PCI. This has also been discussed within the ATLAS community. A scenario where a number of simple custom or commercial boards receive the event data from the ROLs, buffer it, and deliver it via a commercial bus system is shown in Figure 2.5. The Gigabit Ethernet network connection, which is sitting on the module bus too, delivers the event data packets to the level 2 and event filter farm.

Multiple ROLs may be handled by one custom VME or PCI board independently, depending on the available space. A PCI board for example has space for up to four HOLA SLink inputs on the front side. A CompactPCI [3] system may carry up to seven 3 or 6 HU boards, each with four or eight links.

This bus-based scenario is very popular within the ATLAS community and a number of various prototypes have been developed and tested. They are summarised section 2.3.3. The bus-based approach is similar to the component used in other experiments. The main difference is the major technology update by using modern PCs and the PCI bus. Furthermore the ATLAS component has an increased complexity due to the level 2 event data requests (sequential selection).

2.3.2 Cost Modelling

Since the ATLAS readout subsystem will be a huge system, processing and buffering data from 1600 sources and answering requests from two large PC farms over Gigabit Ethernet, the overall system costs are an important issue. A number of architectural choices have been discussed within the community and several of them have been presented in the last paragraph.

[3] CompactPCI is a PCI standard variant for industrial applications. It defines a bus system equal to PCI but with a alternative connector used within 6 HU crates. CompactPCI boards may be 3 or 6 HU

Many different components have been proposed including a number of custom hardware components. Their costs have to be estimated with the goal to get a complete overview of the price of a full system for the various architectural options.

This cost modelling has been done in the ATLAS community [Mor03] and is summarised and partly extended here. It tries to calculate the prices for the scenarios presented in the previous section 2.3.1 which are:

- The ROB-on-the-ROD scenario
- The scenarios using 3, 6 or 9 HU ROB boards with one Ethernet output per ROL input in a simple crate with only a power supply. Each ROB board has one Gigabit Ethernet output per ROL input.
- The scenarios with "grouping" 3, 6 or 9 HU ROB boards in a simple crate with only a power supply. Each board has N HOLA SLink inputs (see previous section for the specific number), two Gigabit Ethernet outputs, and performs local event building on request data.
- The bus-based scenario with PCI based ROB boards in a standard PC (ROS-PC).

A VME based scenario, as used in many other experiments (see section 2.2), has not been taken into account because prototyping effort has shown that the ATLAS requirements can not be met (see next section 2.3.3).

The prices assumed for the various components are listed in Table 2.4. For all custom components prices have been obtained from [Mor03]. No difference is made between the price for the simple 3, 6, or 9 HU boards and the "grouping" boards which perform local event building. The 3 HU is assumed to be equal to a PCI ROB board.

Component	Price [EUR]	Source
ROBs		
ROB-on-the-ROD module	400	[Mor03]
3 HU board (with 2 ROBs)	1500	equal to PCI
6 HU board (with 5 ROBs)	4500	[Mor03]
9 HU board (with 8 ROBs)	7500	[Mor03]
PCI ROBIN board (4 SLinks)	1500	[Mor03]
Switches		
16-Port Switch (price per port)	80	[Alt]
Crates (simple, no bus system)		
6 HU (with power supply)	700	[RSC]
9 HU (with power supply)	800	[RSC]
PC		
standard PC with 4 PCI segments	1500	[Mor03]

Table 2.4: The prices for various components used for the different ROS scenarios 2.3.1.

This leads to the costing estimations for a complete system listed in Table 2.5. The presented model neglects a number of topics. It should only support the design choice with a raw price estimation. The omitted topics are:

- Influence of the scenarios to other parts of the ATLAS DAQ (SFI or level 2 farm size, number of switches for the level 2, SFI farm, ...).
- Cable costs.
- The cost of racks.

Scenario	ROB Boards	Crates PCs	Switch Ports	Estimated System cost [EUR]
ROB-on-the-ROD	1600	X	1800	784.000
Simple 3 HU	800	50	1800	1.379.000
Simple 6 HU	320	20	1800	1.598.000
Simple 9 HU	200	13	1800	1.653.100
Grouping 3 HU	540	20	x	837.200
Grouping 6 HU	200	13	x	909.100
Grouping 9 HU	133	9	x	1.004.700
PC ROS	400	133	x	799.000

Table 2.5: The estimated costing of a complete ROS for the different scenarios. The prices for the components have been taken from Table 2.4. 2.3.1.

For the first four scenarios a large number of switches are necessary which connect the ROBs to the level 2 and SFI farm. Each of them is considered to have 16 ports. Two up-links are counted per 16-port switch and added to the 1600 ports necessary to connect the ROBs. This leads to a network with 1800 ports. No switches are added to the remaining three scenarios because the ports are intended to be already present in the level 2 and SFI farm switch (which is not taken into account here). Depending on the ROS grouping factor (the number of ROLs per ROS component) a number of ports, in the order of 200 - 300, must be present in the level 2 and SFI farm switch. This is equal to the previously mentioned number of up-links and makes the model consistent.

With this calculation the ROB-on-the-ROD scenario is the cheapest closely followed by the PC ROS. Since the PC ROS approach performs partial local event building, it reduces the SFI farm contrary to the ROB-on-the-ROD scenario. This gives the PC ROS an advantage even though it is not the most cost effective solution.

2.3.3 Previous Implementation Approaches

During the planning phase of the ATLAS DAQ system a number of approaches for the readout subystem has been investigated. All of them follow the bus based scenario presented in the last section. Three buses have been evaluated: VME, PCI and CompactPCI.

In all cases a custom ROB module with the evaluated bus interface has been developed which is called ROBIN (readout buffer Input). It is based in most cases on a microcontroller and a FPGA with additional buffer memory storing the event data from the readout links.

CERN VME ROS

The VME ROS implementation shown in Figure 2.6 has been developed at CERN [CFJ+00]. It is based on a number of CES RIO 8062 [CES] single board computers running LynxOS.

These CPUs are contained in a VME crate and have different tasks. There are CPUs acting as an interface to the level 2 farm (L2IF) and the SFI event builder farm (EBIF). Other CPUs act as ROBs receiving and buffering data from the RODs. Event data is moved by the VME bus between the ROB boards and the interface boards.

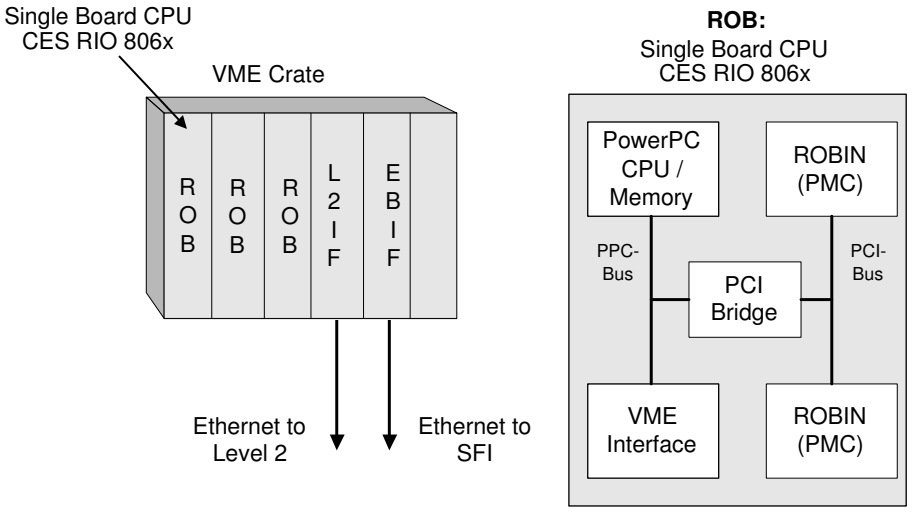

Figure 2.6: VME ROS implementation with single board computers in a VME crate [CFJ+00].

The RIO 8062 single board computer is based on a PowerPC 604e CPU and contains a PCI bus with two PMC (PCI Mezzanine Card) format slots. These are used to carry two ROBIN boards for receiving and buffering the data from one ROL. Two options have been tested: one based on another single board computer, the RIO MFCC 8441, and the UK-ROBIN. Both are schematically shown in Figure 2.7.

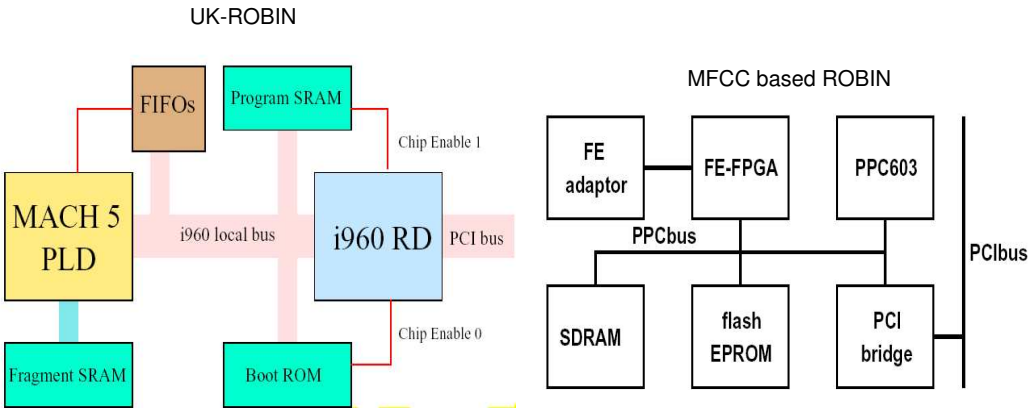

Figure 2.7: ROBIN options used within the VME RIO 8062 CPU. The MFCC is another PowerPC based single board computer from CES [CES]; the UK-Robin is a custom development based on an Intel i960 processor [CFJ+00].

The MFCC board is a commercial FPGA and PCI board from CES in PMC format which is designed to be applied to the VME CPUs of the same company. It is based on a PowerPC CPU and an FPGA which handles the incoming event data from SLink. Furthermore a PCI interface connects the module to the RIO CPU.

Incoming event data is stored in a 1020 byte page inside the SDRAM. This page and additional information is store in a FIFO where it is picked-up by the MFCC CPU. On request the CPU can pass the event data over PCI to the host CPU (the CES RIO 8062 Single Board Computer).

The UK ROBIN module has been developed at the Royal Holloway University of London (RHUL) and the University College of London (UCL) [GPR+00]. It is based on an i960 processor which also provides the PCI bus interface. One MByte of SRAM forms the event fragment data buffer.

The SLink input data stream is written directly into the SRAM with the help of control logic inside a MACH 5 PLD [4]. This is shown in Figure 2.8. Two FIFOs store empty and full pages of the SRAM event buffer; each page is defined to be 1024 Bytes. An incoming event fragment allocates a free page from the free-page FIFO and the control logic directs all data to the page address. The CPU is finally notified of the arrival of a new event data fragment using the used-page FIFO. The software part of the buffer management stores the page address in a hash table, where it can be requested or deleted. Event data leaves the hardware via the PCI interface which is integrated into the i960 CPU.

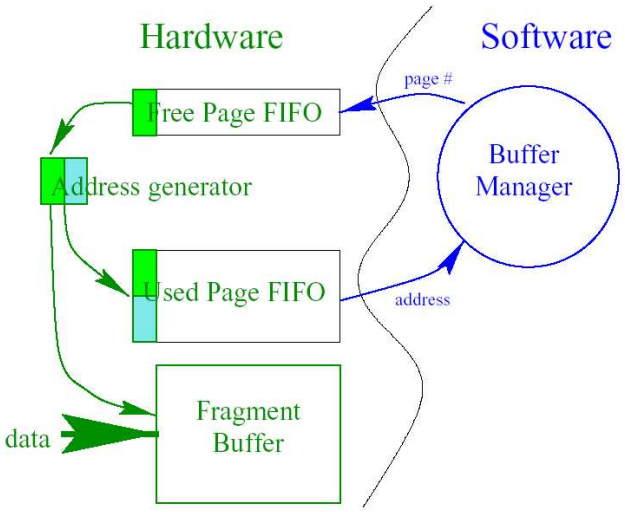

Figure 2.8: UK-ROBIN buffer management. Incoming event data fragments are stored in an empty page. The CPU is notified using the used-page FIFO. [GPR+00]

A special distributed software on the RIO CPUs uses a communication mechanism based on shared memory to distribute requests to the ROBIN modules and retrieve event data over the VME bus.

Performance measurements with this ROS implementation have shown that only with one or two ROBIN modules (either MFCC based or UK-ROBIN) the ATLAS requirements from section 2.1 can be fulfilled [CFJ+00].

Saclay CompactPCI ROS

The CompactPCI ROS approach, developed at the Commissariat à l'Energie Atomique, Saclay, [CGHM00] uses a 6 HU crate with two CompactPCI [Com97] backplanes on two 3 HU levels. This is shown in Figure 2.9. Each level has 16 slots to carry ROBINs (for receiving

[4] PLD: Programmable Logic Device

and buffering ROL event data), one ROB controller implemented using a CompactPCI single board CPU, and a network interface card (NIC). The 16 slot backplane provides two 8 slot PCI buses connected with a PCI-to-PCI bridge [Gro98c].

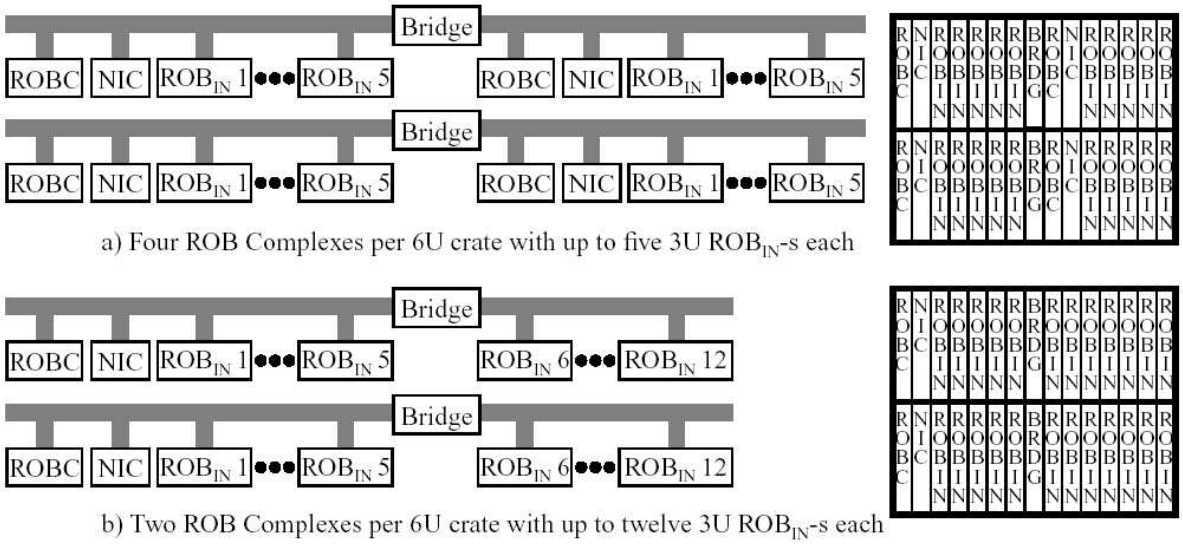

Figure 2.9: CompactPCI based ROS System using a 6 HU crate. The crate is divided into two 3 HU levels each having 16 slots. Each level has two CompactPCI buses connected by PCI-to-PCI bridges. Two configurations are shown: one where each of the four segments has one Ethernet interface (NIC) and one Controller (ROBC), and one where each level has a NIC and a ROBC. [CGHM00]

Saclay developed a PMC format ROBIN Module with a local Intel i960 processor for event management and request handling. Input from SLink is handled by an FPGA and stored in 8 Mbyte SDRAM. The event buffer management on the ROBIN is similar to the UK-ROBIN.

PCI I/O provides a PLX9080 bridge chip on the ROBIN module. The ROBIN PMC card may be used in a VME environment but the target is the use of a large CompactPCI system with a CompactPCI PMC carrier.

Since the Saclay group has left ATLAS in spring 2000 no detailed measurements are available and the project has been discontinued.

PC Based ROS

This approach proposes a standard PC with one or multiple PCI segments to be used as a ROS module. The ROS-PC is equipped with up to two PCI bus Gigabit Ethernet interfaces one for level 2 and one SFI farm connection. The processor within the PC handles the requests from the network and performs a local event building on the event data. A number of ROBIN (ROB input) boards are responsible for receiving and buffering the data. These boards are in most cases custom hardware developments. Various ROBINs have been developed and tested by different institutes:

- A SHARC DSP based ROBIN has been developed at NIKHEF [BJG$^+$00].
- A PCI variant of the previously mentioned UK-ROBIN [GPR$^+$00].

- A ROBIN based on a PCI FPGA Co-Processor has been developed at the University of Mannheim [Ris99] [BBW+01].

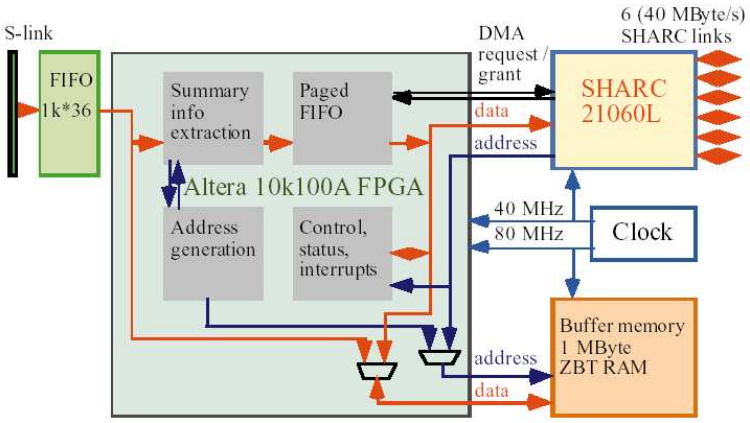

Figure 2.10: PCI ROBIN based on a SHARC DSP developed by the NIKHEF institute [BJG+00].

The SHARC based ROBIN is shown in Figure 2.10. It is based on an Altera 10k FPGA, a SHARC DSP processor, and 1 MByte ZBT [5] SRAM. The FPGA handles the input stream from one SLink ROL input; the processor is responsible for the buffer management and the request handling.

The event data buffer inside the 1 MByte SRAM is organized as a ring buffer with two pointers: one "fill"-pointer for the beginning of the empty buffer area and one "empty"-pointer placed at the end of the empty buffer area. The buffer management stores memory addresses of incoming event data inside a hash table. Care has to be taken that no event data is getting overwritten in the buffer. The hash value is generated out of the last 10 bit of the level 1 event ID.

The ROBIN is connected through the SHARC DSP links (each having a bandwidth of 40 MB/s) to a second SHARC board which makes the connection to the PCI bus. Up to four SHARC ROB boards can be combined to one PCI interface.

The FPGA ROBIN, developed at the University of Mannheim, is based on the commercial available multi-purpose FPGA co-processor microEnable [ea98]. It comprises a Xilinx 4000 series FPGA, a PLX9080 PCI interface, and 2 MByte asynchronous SRAM (see Figure 2.11).

Two firmware versions for this hardware have been developed the first one within a diploma theses [Ris99]. This was able to receive and buffer event data within the FPGA, and handle requests from the host PC over PCI.

The SRAM event buffer was divided in 2 kByte pages organised as a ring buffer. Each incoming event gets a new page with incremental start address. The page address is stored in a hash table using the last 10 Bit to generate the hash value. When all buffer pages are in use, old events get overwritten automatically. This first implementation has shown a very slow PCI interface. It could be substantially improved in a later version, which was the first step towards the development presented in this thesis.

Comparison measurements with other PCI based ROBIN prototypes have been done and are shown in Figure 2.12. This shows the maximum sustainable level 1 event rate, which is equal to the ROBIN input rate, depending on the ROBIN output request rate.

[5] Zero Bus Turnaround

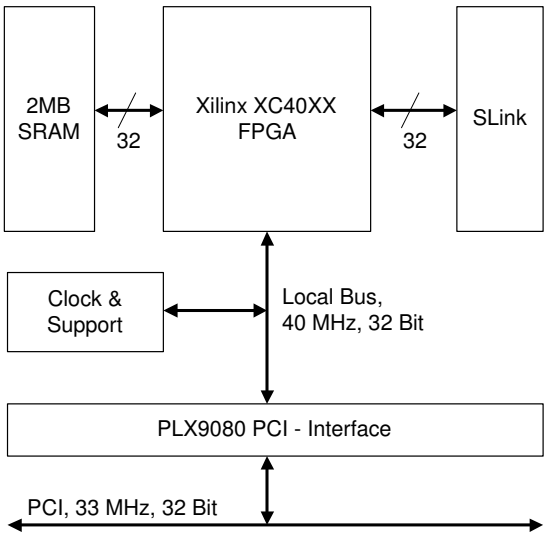

Figure 2.11: PCI ROBIN based on a the commercial FPGA Co-Processor microEnable [ea98] [Sil].

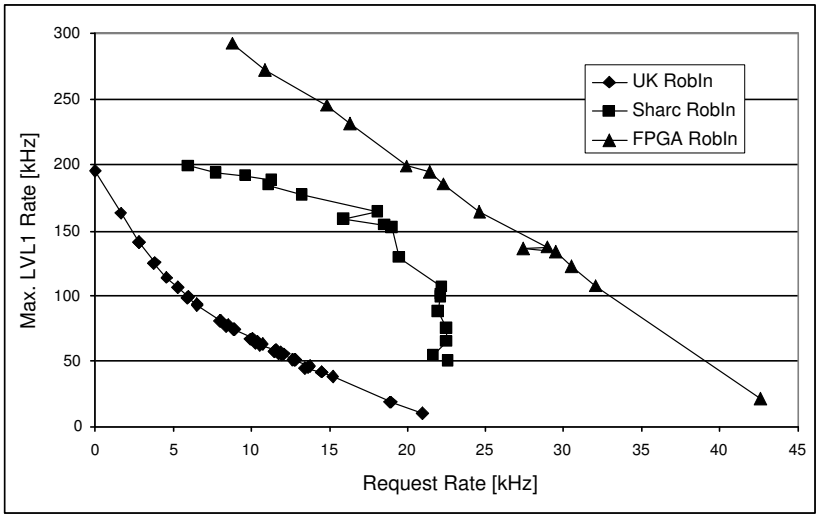

Figure 2.12: The performance of the three PCI ROBIN prototypes NIKHEF SHARC ROBIN, UK-ROBIN, Mannheim FPGA ROBIN, tested in a ROS-PC with equal software [FMTV].

Only the SHARC and the microEnable FPGA ROBIN are able to fulfil the requirements of 10 kHz request rate at a ROL input rate of 100 kHz. The FPGA ROBIN can sustain even three times the requirements, which is 10 kHz more than the SHARC ROBIN.

2.3.4 The ATLAS Baseline Architecture

Out of the number of discussed scenarios, the cost model, and the experiences of the previous ROS prototypes, a decision on the ATLAS ROS baseline architecture has been done within the TDAQ technical design report [Gro03].

One of the main aspects for the architectural choice is the system costs. Thus, regarding section 2.3.2, the PCI bus based scenario and the ROB-on-the-ROD scenario are reasonable candidates for the ATLAS ROS.

A VME solution, similar to the systems in other experiments (see 2.2), has been tested and found out to be too slow. A SHARC based solution with 40 MByte/s SHARK DSP links for communication and event building, similar to the HERA B experiment, (see 2.2) can also be excluded since the event building network must be more powerful and connected to a high number of PCs. The usage of a CompactPCI may also be an option, but has not been deeply investigated.

Thus the baseline choice, presented within the Trigger/DAQ Technical Design Report, is the bus-based solution with a standard, common-of-the-shelf PC. It is shown in Figure 2.13. Since a standard PC is not able to handle a reasonable number of ROLs it has to be extended by custom boards, called ROBINs (readout buffer input). Their task is to receive the ROL data and buffer it. The output of the ROBIN is done via PCI on demand of the host PC. Each ROBIN is foreseen to handle four ROLs. This is expected to be the maximum number of ROL SLink connectors which fit on a PCI board due to space considerations.

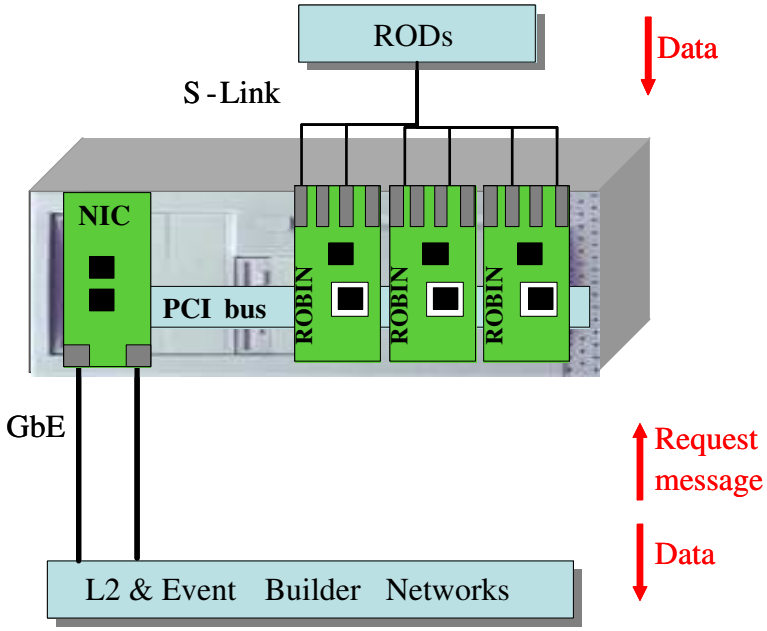

Figure 2.13: The ATLAS ROS baseline architecture [Gro03].

The PC hosts the ROBIN boards and listens to requests from the level 2 farm (ROI data requests) and the SFI event builder farm (EB request on a level 2 accept decision). An arriving request is forwarded to the ROBIN boards and a local event building is performed on the returning event data within the ROS-PC.

This approach has a number of advantages:

- It is a cheap solution
- A number of prototypes have been built implementing these scenario. It has been most of all tested.
- It has been proved that the performance requirements can be met using the ROBIN prototype presented within this theses. First preliminary measurements of this prototype have been used in the TDAQ TDR for this verification [Gro03].

- It uses a very common standard PC as basic component.
- It can be flexibly extended by re-organising the number of ROBINs in the PCs. One could start with grouping more ROBINs in the beginning phase of ATLAS.

Contrary, the PCs will use a lot of space. This is partly balanced by the saving of SFI farm PCs due to the local partial event building within the ROS PCs (see 2.3.1).

A number of ROBIN prototypes have been already investigated. All of them handle only one ROL and are mainly based on processors. This thesis will present in the next chapters a bus-based ROS development using a PC with multiple PCI segments and a ROBIN prepared to run with four ROL inputs. The developed ROBIN board is the first device which is able to handle four ROL inputs on one PCI board within ATLAS. Using this with a standard PC for a readout system and considering the additional requirement due to the level 2 trigger data request this component is new and unique in high energy physics. It will be shown within this thesis that the implementation meets the ATLAS requirements. First results of this work have been used as architecture verification in the ATLAS Trigger/DAQ technical design report [Gro03].

2.4 Summary

The ATLAS readout subsystem has to meet a number of requirements. These have been discussed in this chapter. The level 1 trigger accepts event data with up to 75 kHz including safety factors. This requirement may be upgraded to 100 kHz in a later phase of the experiment. The level 2 trigger requests "Regions-of-Interest" event data from the ROS with a rate of up to 7 kHz. This approach, combined with the high rates is very rare in high energy physics experiments. On level 2 accept, event building starts executed by a PC farm with a rate of 2-3 kHz.

A number of implementations and technologies have been discussed to implement a ROS which can meet these requirements. ATLAS has decided to use a implementation similar to other experiments, but with a major technology upgrade. A PCI bus approach, based on a standard, "off-the-shelf" PC system, together with PCI cards receiving and buffering event data from the multiple ROLs forms the baseline architecture.

This thesis implements the baseline ROS with a PCI card prepared to run with four ROL input links. Considering the sequential selection (requesting) of event data within the ROS, this is a new development and has never been used in a high energy physics experiment before.

Chapter 3

ROBIN Development with a Multi-Purpose PCI FPGA Co-Processor

The implementation of the ATLAS readout subsystem (ROS) baseline architecture, described in the previous chapter, requires ROBIN hardware with a maximum number of readout links (ROL). To develop a system with a low number of PCs and optimized cost, four ROLs per board have to be implemented and tested. None of the available PCI boards is prepared to do that; all previously presented ROBINs handle only one link per board.

The goal of this thesis is to evaluate a ROBIN device which is able to process the data from four links. This device should be as simple as possible to reduce overall cost and effort of custom hardware developments.

The evaluated component is the MPRACE FPGA co-processor. It is a continuation of the microEnable FPGA board presented in 2.3.3 which could been successfully used as a ROBIN device. MPRACE is widely used for other applications e.g. image processing [Hez04], high energy physics trigger acceleration [Bro04], or astronomy simulation [Lie04]. This shows its flexibility.

Contrary to the other ROBIN approaches, MPRACE is based on a FPGA only. No additional processor is present. All ROBIN tasks: ROL data processing, buffering, and the handling of requests has to be done inside the FPGA.

A FPGA has some major advantages compared to the processing inside a CPU. It is able to implement specialized logic for all ROBIN tasks. This operates usually faster than in a CPU. Furthermore several tasks can run in parallel inside a FPGA. A request from PCI, for example, does not disturb the processing of incoming event data. Both advantages have been already successfully used by the microEnable based FPGA ROBIN.

This chapter presents the MPRACE FPGA co-processor and the implementation of a ROBIN which is able to handle four ROLs. Additionally a host-PC has to store the ROBIN boards, receive requests from the level 2 and SFI event builder farm, forward them to the ROBINs, and perform the local event building. This requires software development which is presented in the next chapter. The software has to communicate with the ROBIN via the PCI bus. This is also described in the next chapter.

3.1 The FPGA Co-Processor MPRACE

FPGA (field-programmable gate array) components are re-configurable logic devices. They offer the possibility to implement custom logic circuits in a fast and flexible way. Complex electronic circuits with logical operations (AND, OR, XOR, etc.), RAM elements, registers or finite state machines can be designed and "programmed" into a FPGA device. After this reversible configuration process, which takes only milliseconds, a FPGA operates like a custom electronics device containing the designed circuit.

This allows the simple and fast development of special hardware for various tasks and algorithms. Data movement and formatting, link protocols, and specialized computation pipelines can be easily implemented. Furthermore the realisation of parallel processes is straightforward.

The MPRACE FPGA co-processor allows the usage of FPGA technology in a standard PC environment. It has been developed at the "Lehrstuhl für Informatik V" of the University of Mannheim [MPR].

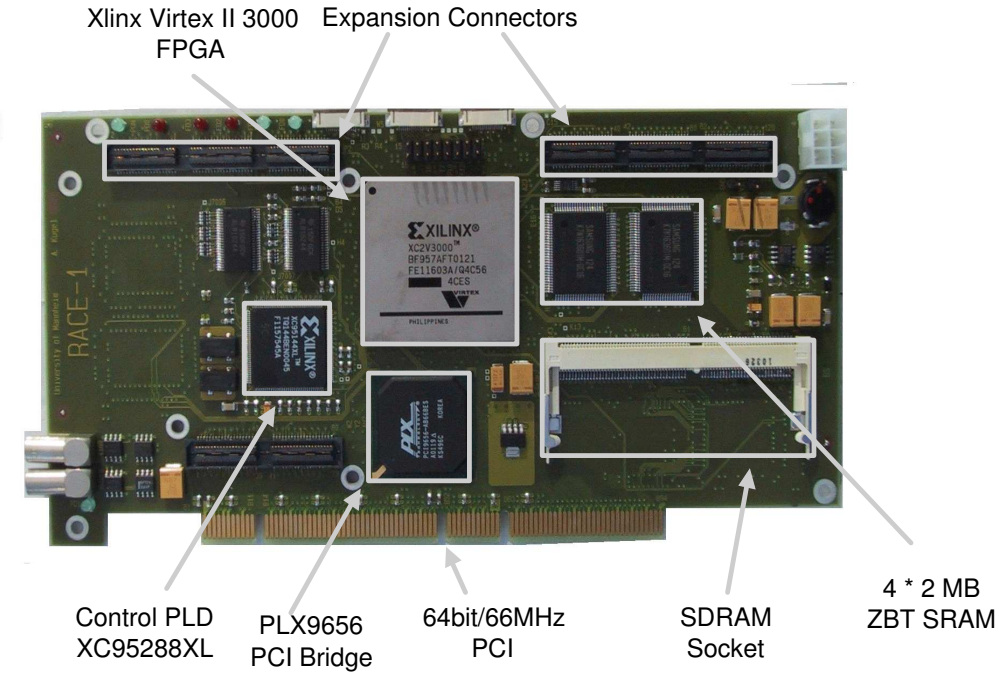

Figure 3.1: The MPRACE FPGA co-processor.

Figure 3.1 shows the MPRACE board with its main components:

- A Xilinx Virtex II 3000 FPGA with three million gates.
- A PLX 9656 PCI Bridge for a 64 bit/66 MHz PCI connection.
- Four independent ZBT[1] SRAM banks, each with 2 MByte.
- One carrier for SDRAM SO-DIMMS[2] (laptop style SDRAM modules).

[1] Zero Bus Turnaround (ZBT) SRAM needs no additional turnaround cycles when changing from read to write or vice versa.

[2] Small Outline - Dual In-line Memory Module

- Two expansion connectors for daughterboards with additional logic.
- A Xilinx XC95288 PLD for board control (clock frequency settings, FPGA configuration, etc.)

The main device is a Xilinx Virtex II 3000 FPGA in a BF957 package with three million gates and 684 available I/O pins. This FPGA provides 3584 configurable logic blocks (CLBs) each with 8 programmable 4-input function generators, a fast interconnection network, and 96 18 kBit RAM blocks (called BlockRAM) in special chip areas. Twelve digital clock managers (DCMs) can be used for clock generation and synchronisation.

The PLX 9656 PCI bridge is directly attached to the I/O pads of the FPGA. It provides a 32 bit / 64 MHz local-bus (see the block diagram in Figure 3.2) for communication. All data, exchanged between the FPGA and the PCI bus, has to pass this 264 MByte/s local-bus. Furthermore the MPRACE control PLD is attached to this local-bus. It contains a set of registers for an easy board control and the local-bus arbiter. The latter allows either the FPGA or the PLX 9656 to be a master on the MPRACE local-bus.

Various data transfer modes can be used by the host-PC to communicate with the MPRACE (see [Tec]):

Programmed I/O(PIO) In this mode the host-PC processor forces the read and write of data via the PCI bus. Therefore the target address area on the PCI bus is mapped into the virtual address space of the user process and a simple pointer operation executes the MPRACE access. The PLX 9656 translates each PCI access to a pre-defined address window into local-bus cycles targeted to the FPGA or PLD.

PLX - DMA The PLX 9656 provides a number of PCI bus-master DMA modes. In all of them the PLX bridge reads data from the PCs memory and writes them to a local-bus address or vice versa. The PC memory area may be a continuous block or a scattered number of areas (defined by a list of start addresses together with length information). A DMA flow control is also available on the local-bus which can delay the DMA operation if no more data is present (DMA-on-demand). It is asserted by a signal line between the FPGA and the PLX 9656, called DREQ (DMA Request). For all DMA operations a number of PIO accesses are required to program a set of registers inside the PLX 9656 chip.

Direct Master Direct master is another PCI bus-master DMA mode where the FPGA can directly read or write into the PC's memory. Therefore the PLX 9656 defines an address area on the local-bus for PCI I/O. Any read or write to this area is translated into PCI access cycles to a previously defined address range. The PLX 9656 acts as a window or gateway. This mode allows the FPGA to randomly access the memory of the PC.

All other MPRACE components, connected to the FPGA, are shown in the block diagram in Figure 3.2. The ZBT SRAM banks have a data width of 36 bit each and run with a clock frequency of up to 167 MHz. The SDRAM modules are connected with a 64 bit data bus. The frequency depends on the selected SO-DIMM module.

For extension purposes two high speed connectors are mounted on MPRACE. Each provides 94 signal lines directly attached to the FPGA. Currently available hardware extends the MPRACE by a HOLA SLink interface, a Gigabit Ethernet interface, additional SRAM or SDRAM, a PowerPC405, or a PowerPC440 processor.

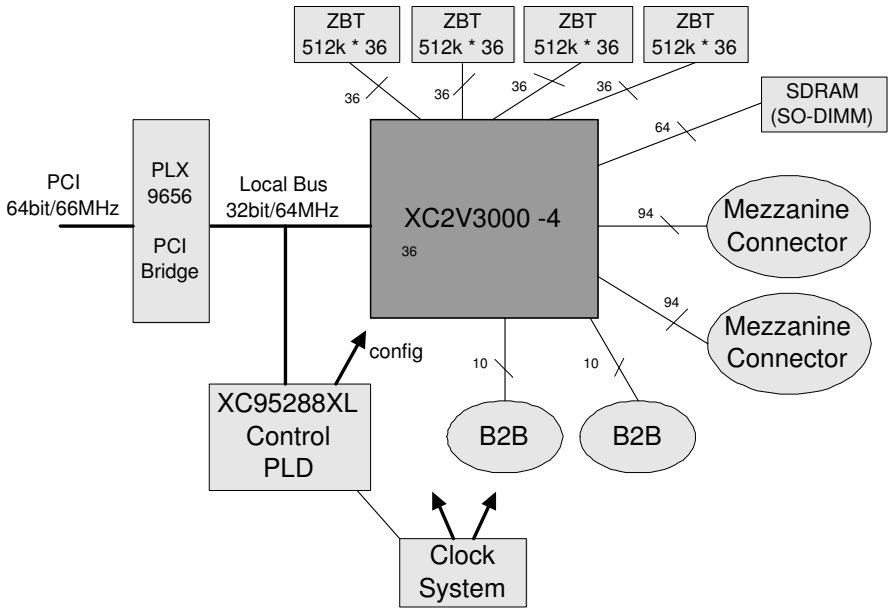

Figure 3.2: Block diagram of the MPRACE FPGA co-processor.

Two board-to-board connectors can be used to concatenate neighbouring MPRACE boards. Furthermore a XC95288XL PLD is used to control the MPRACE board. It provides mainly two services: FPGA configuration and clock control. The FPGA firmware can be uploaded via PCI bus to the board. Therefore the VirtexII device provides an 8 bit parallel configuration interface. The control signals for FPGA configuration are served by the PLD. The whole firmware upload typically takes only a few milliseconds.

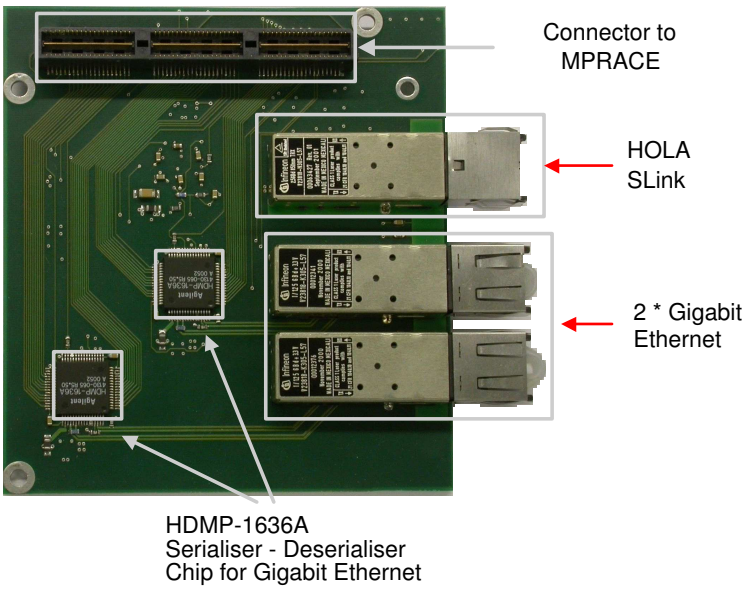

Figure 3.3: MPRACE HOLA SLink extension board. The mezzanine carries one optical 2.5 GBit/s transceiver and 2 optical Gigabit Ethernet transceivers. Each port needs a serialiser-deserialiser chip. Two of them, the HDMP 1636A for the Gigabit Ethernet, are shown. The TLK2501 for the HOLA SLink port is located on the rear side of the mezzanine.

MPRACE provides two clock domains. One is exclusively used by the PLX local-bus and can be set to a frequency of 8, 16, 32 or 64 MHz. The second clock domain is distributed to the FPGA and all other MPRACE components (SRAM, SDRAM, ...). It can be switched to 8, 16, 32, 64 or 125 MHz. Alternative frequencies have to be generated inside the FPGA using one of the DCMs.

For the MPRACE ROBIN implementation a HOLA SLink (see section 1.3) interface extension board was required. This has been developed within a diploma thesis [Fis02] and carries one HOLA SLink and two optical Gigabit Ethernet ports. Figure 3.3 shows a picture of this hardware. Beside the optical transceivers, the mezzanine carries the HDMP 1636A serialiser-deserialiser chips. Two of them convert the Ethernet link data from an 8 bit parallel bus into a serial data stream. The HOLA SLink data arrives on a 16 bit bus and is converted by a TLK2501 into the 2.5 GBit/s serial data stream.

3.2 The MPRACE ROBIN

3.2.1 Hardware Usage

The goal of this thesis is to employ the MPRACE hardware for the evaluation of a ROBIN which is able to handle four readout links (ROL). Event data from each of the HOLA SLink inputs has to be stored in buffer memory and provided to the PCI bus on request.

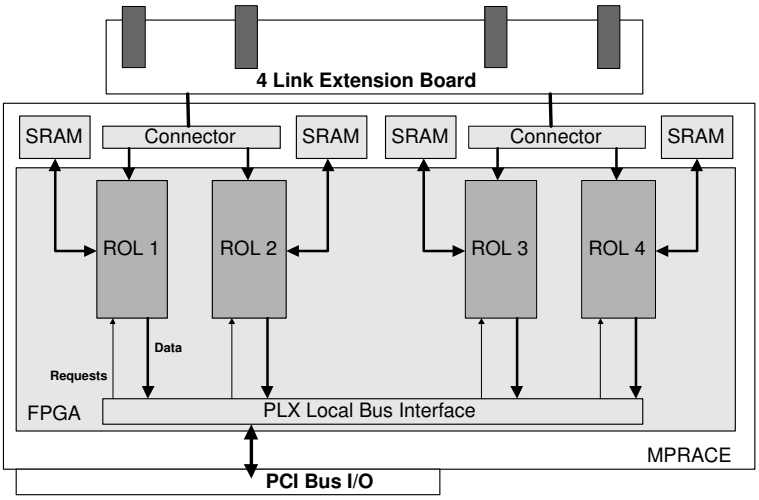

Figure 3.4: The mapping of the MPRACE resources to the ROBIN application.

Figure 3.4 shows the MPRACE components usage by the ROBIN application. The main functionality is implemented inside the FPGA. The HOLA SLink hardware is plugged on one of the MPRACE extension board connectors. Four independent ROL handlers process incoming event data and store it in the four MPRACE ZBT SRAM Banks. Messages (from level 2 or SFI), requesting a ROBIN service, arrive via the PCI and PLX 9656 local-bus interface. Event data is transmitted over the same interface in the opposite direction.

Since a four HOLA SLink mezzanine has not been available during the testing phase, the hardware presented in the last section had to be used. It provides only one link and shows that a HOLA SLink can be handled by the MPRACE FPGA firmware. Thus the data source

for the remaining three links of the MPRACE ROBIN had to be placed inside the FPGA. Measurements have proven that data input from a real HOLA SLink, compared to the internal data source, produces similar results (see section 5.2).

Designing a four HOLA SLink mezzanine for MPRACE is generally a minor problem. The ability to handle one link is proven by the present mezzanine. For a four link extension of this board two questions have to be considered:

- Are there enough electrical signals from the FPGA to drive four HOLA SLinks?
- Is there sufficient space to place four optical transceivers in a way that they can be accesses from outside the PC?

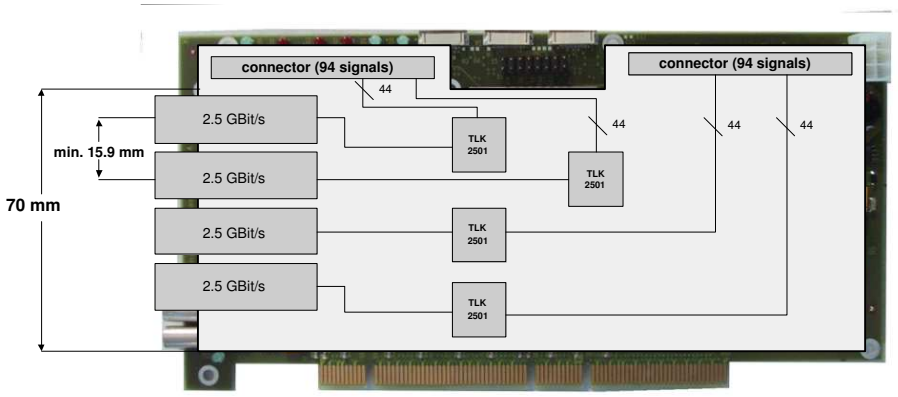

Figure 3.5: Outline of a four HOLA SLink mezzanine for the MPRACE board.

A possible schematic outline of the board is shown in Figure 3.5. Each HOLA SLink requires 44 signal Pins for the TLK2501 serialiser-deserialiser connection. Both MPRACE connectors provide 188 signal lines altogether. Thus sufficient electrical signal pins are available if both MPRACE expansion connectors are used. But one of them causes a long circuit path for two of the four HOLA SLink ports, which may create potential timing problems (see Figure 3.5). The TLK2501 serialiser-deserialiser should be placed close enough to the connector to reduce runtime differences between the 16 data signals from the connector to the TLK2501 chip.

Furthermore four optical transceivers have to be placed on the front panel of the mezzanine to make them available even if the PC is closed. The minimum pitch for the transceivers is specified with 15.9 mm [Inf02]. 70 mm is available on the mezzanine (see Figure 3.5), more than enough.

3.2.2 FPGA Firmware Overview

Together with the SLink extension board presented in the last subsection, MPRACE can be used as a complete ROBIN board for Atlas. Since only one HOLA SLink is present on the mezzanine input data for the other three has to be generated inside the FPGA. The MPRACE FPGA firmware implements the complete ROBIN functionality (see section 2.3.4). It

- Receives event data from the HOLA SLink on the mezzanine or from the internal data generator.

- Buffers them inside the MPRACE SRAM banks.
- Processes event data requests from PCI.
- Processes event delete requests from PCI.

The FPGA firmware for the MPRACE ROBIN has been developed using the VHDL language [3] and a number of tools for verification and design synthesis (see section 3.3).

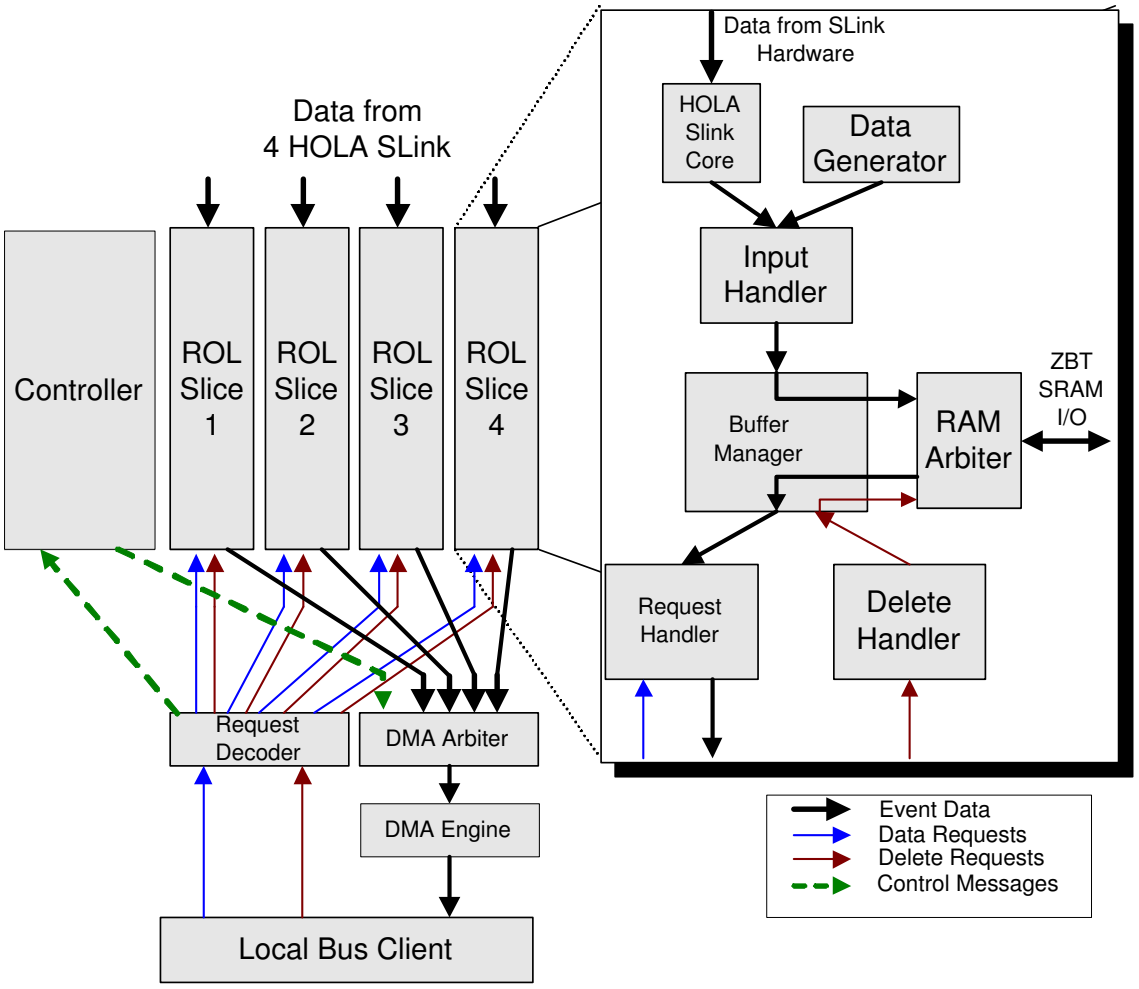

Figure 3.6: ROBIN FPGA firmware overview.

A block diagram of the firmware is shown in Figure 3.6. Each of the blocks represents a VHDL entity. The whole firmware is divided into four logical slices each responsible for one readout link (ROL). The MPRACE local-bus performs the host-PC communication. Three modules inside the FPGA firmware are valid for this communication. Finally a global controller module provides an interface for ROBIN control.

Each ROL slice is subdivided into a number of modules. Event data fragments, coming from the link hardware, have to be processed by the HOLA SLink module first. This module provides a SLink compatible interface (see section 3.2.4) which is also provided by the internal data generator module. Incoming event data is delivered to the input handler module. A buffer

[3] VHDL: VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

manager is responsible for the event buffer organization inside the MPRACE SRAMs. It is notified about incoming events and decides where they have to be stored.

An event data request from local-bus/PCI is processed by the request handler. It localises the event data within the buffer using the buffer manager module, retrieves the data, and adds the ROB header (see section 1.4). Finally the DMA engine transfers the formatted event data to the PC memory via the local-bus/PCI.

If a delete request has been received the delete module gets activated. It locates the event by its level 1 ID and ask the buffer manager to remove it.

Two Arbiter modules organize the access of the MPRACE SRAM and the DMA output. The ZBT SRAM is used by three modules of a ROL slice: the input handler, the request handler, and the delete handler. The DMA output is used by the four ROL slices and the controller module to communicate with the host-PC software.

3.2.3 Event Buffer Management

The buffer manager module organizes the MPRACE SRAM event buffer. This buffer organisation has been implemented by different approaches in previous prototypes (see section 2.3.3).

The UK-ROBIN and the MFCC-ROBIN divide the memory in pages of a pre-defined size (UK-ROBIN: 1 kByte, MFCC-ROBIN: 1020 bytes). Free pages are kept in a FIFO or queue; full pages plus additional information are passed to a FIFO to notify the ROBIN CPU. This finally performs the buffer management: updates a list of present event data, deletes events from the buffer on request, and fills the empty page FIFO (see [GPR$^+$00]).

Only the SHARC ROBIN uses continuous memory organised as a ring-buffer. A fill-pointer marks the next free buffer area and incoming event data is written to this location. The SHARC DSP stores the memory position of each new event data fragment in a hash table. A delete request to the ROBIN marks the corresponding event data buffer area as free. The end of the free buffer area is marked by an empty-pointer.

Is the end of the buffer reached a wrap-around causes the fill-pointer to start from the beginning of the memory. A check against the empty-pointer avoids an uncontrolled overwriting of event data within the buffer (see [BJG$^+$00]). This check also declares the buffer to be full if the fill-pointer does not start at a free area.

The disadvantage of this ring-buffer scheme is the fragmentation of the buffer memory. The buffer may be declared as full even if half of it is marked as free because free areas are distributed over the whole memory.

All above mentioned ROBINs use a hashing scheme based on a fraction of the lower bits of the event's level 1 ID.

Due to the disadvantage of the SHARC ROBIN buffer management, the MPRACE ROBIN implementation uses a page oriented buffer management scheme similar to the UK-ROBIN. But since no processor is present everything is done inside the FPGA. The full-page FIFO (see section 2.3.3) has been omitted since no communication with a processor is necessary. Furthermore the page size has been increased to 2 kByte to avoid the usage of two pages per event data fragment.

Figure 3.7 shows a schematic diagram of the MPRACE ROBIN buffer manager. One 2 MByte MPRACE SRAM keeps the event data from one ROL. The SRAM is divided into 2 kByte pages, whereas the first eight pages are reserved to store the hash table. Thus up to

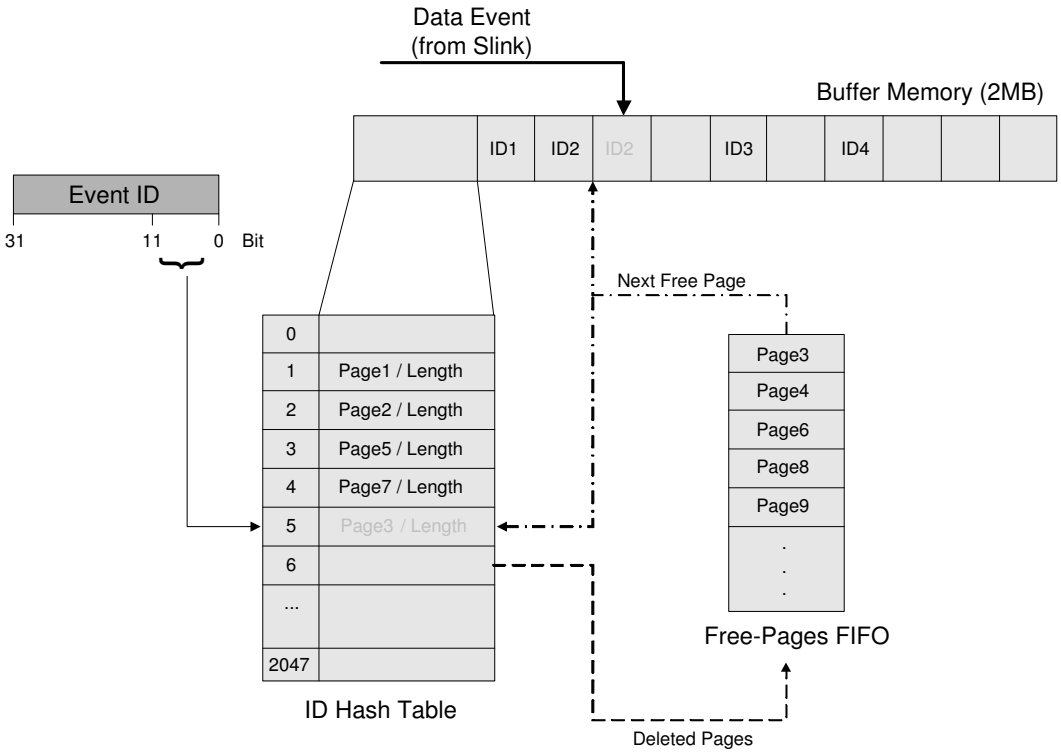

Figure 3.7: The MPRACE ROBIN buffer manager scheme.

1017 pages may store event data.

With a level 1 rate of 100 kHz events data can stay 10,17 ms in average inside the buffer. This is enough to store event data fragments for the mean decision time of the level 2 trigger (see section 1.1).

A 2048 entry free-page FIFO, implemented in VirtexII BlockRAM, contains the 12 Bit page number of all free buffer pages. Incoming fragments allocate one of these pages. After the event data is stored in the buffer, a hash table entry, containing the 12 Bit page number, the event data fragment length, and status information, is written to a hash table.

The hash key is generated out of the lowest 12 bit of the event ID. This hashing scheme is similar to all other mentioned ROBIN prototypes. With a monotone increasing level 1 ID the hash table entries get overwritten after 4096 events or 40,96 ms at 100 kHz level 1 rate.

If an event delete is requested, the corresponding page address is picked out of the hash table and pushed back to the free-page FIFO. The hash table entry is cleared by writing a value of zero (page zero of the buffer is not in use for event fragments).

This buffer management scheme limits the size of event data fragments to 2 kByte. It is sufficient for the fragments from all sub-detectors (see section 2.1). Any fragment larger than 2 kByte would be cut in the end and an error flag would be stored within the hash value (see section 3.2.5).

3.2.4 SLink Input and Event Data Generator

Event data from a HOLA SLink source card (the readout driver output) has to be handled by the MPRACE ROBIN implementation (see section 1.3). Since a HOLA SLink receiver

card can not be plugged on MPRACE (no matching SLink connector is present) the already mentioned mezzanine (see Figure 3.3) has been developed. This contains only the physical link implementation. The SLink protocol engine, which is inside a FPGA on the original hardware, has to be added. Therefore a VHDL module, provided by CERN, is placed inside the MPRACE FPGA.

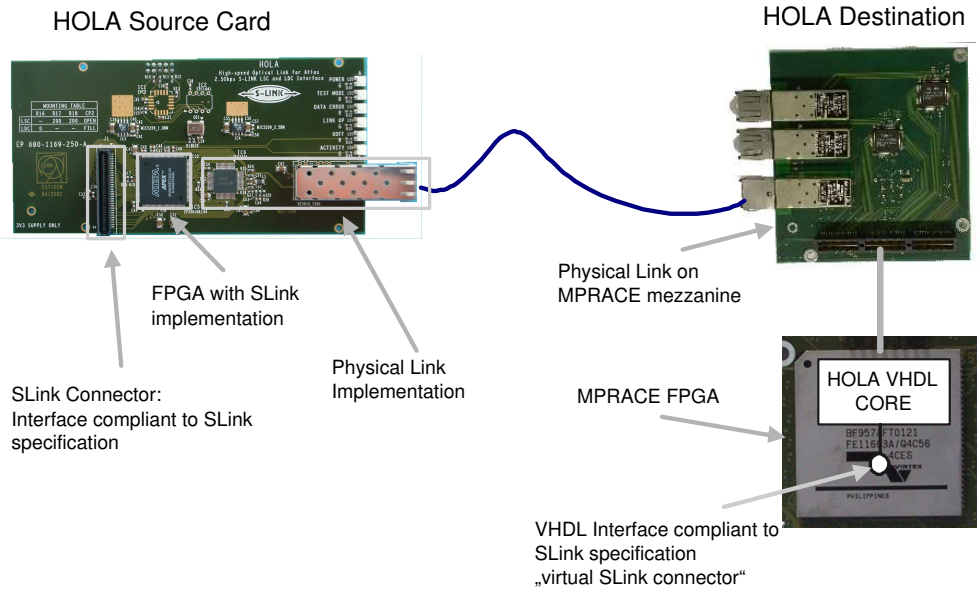

Figure 3.8: Setup of a SLink connection between the MPRACE ROBIN using the HOLA SLink mezzanine and a standard HOLA SLink source card. The link functionality is divided into two parts: a physical link (2 GBit/s optical transceiver plus serialiser) and a SLink protocol and interface engine (available as VHDL module).

Figure 3.8 shows the connection between a HOLA SLink source card and MPRACE. The original hardware provides a specified electrical interface connector. The MPRACE implementation uses a HOLA VHDL module to create a ”virtual” SLink connector within the FPGA. This can be used by the connected modules of the ROBIN firmware.

Beside this HOLA VHDL module an internal data generator supplies event data via a compatible interface. A multiplexer allows switching between both data sources: the HOLA SLink input and the data generator. The data generator allows a programmable event data size and rate. This has turned out to be very helpful for the tests and performance measurements presented in chapter 5.

3.2.5 Input Handler

The input handler processes the incoming data from the SLink interface (or the data generator) and adds them to the event data buffer with the help of the buffer manager. Additionally a simple sanity check of the incoming event data fragment is done. The collection of statistics and monitoring data which would be required by the online control software is currently not supported.

The input handler consists of two parts: a 512 word deep FIFO and a finite state machine running the main algorithm. A FIFO is essential to ensure that no data gets lost when the SLink flow-control signal (XOFF) has been asserted. The runtime of this signal over the fibre

back to the source together with the reaction latency causes a delay between the flow-control assignment and the abort of the input data stream. During this period the MPRACE ROBIN must still be able to receive data. This is guaranteed by the input FIFO.

The flow-control signal is derived from the filling state of this input FIFO. SLink data transfer is interrupted if the FIFO is half-full or more. This allows the transmission of additional 256 words after the flow control signal has been raised. Out of this FIFO margin the maximum fibre length can be calculated using formula 3.1 and the HOLA hardware specification [RvdBH]:

$$L = \frac{FM * DTR - LSC}{UFD * 2} \tag{3.1}$$

where:

L	=	fibre length [m]
FM	=	FIFO margin (256 words)
DTR	=	data transfer rate (16 ns/word)
LSC	=	SLink source reaction time to stop transmitting data after XOFF received (320 ns)
UDF	=	unit fibre delay - time for light to travel 1m in fibre (approx. 6 ns/m)

The result is a maximum fibre length of 314 m, enough for ATLAS purposes.

Figure 3.9 shows the algorithm of the input process. It is triggered by the arriving of new event data via SLink which causes the input FIFO to get filled. As a first step the availability of a free buffer page is checked by testing the status of the free-page FIFO inside the buffer management module.

If a new page can be allocated, the first data word is read from the input FIFO. According to [BFM+04], appendix A, the ROD starts a new event data fragment with a control word indicating a begin-of-frame. This maker carries in the upper 16 bits a 0xB0F0, while the lower 16 bits are reserved for transmission error control bits. The begin-of-frame marker causes incoming data to be copied to the selected buffer page.

Again the end of an event data fragment is signalled by an end-of-frame control word (similar to the begin-of-frame, only the upper 16 Bit change to 0xE0F0). If this is received before the buffer page is full the hash table entry is generated, written to the hash table (see buffer manager paragraph), and the algorithm waits for new data.

If the buffer page is filled up before the end-of-frame marker has been received, the event fragment is cut and the hash table entry is generated with an additional error flag raised. Finally the input FIFO is read until either an end-of-frame or new begin-of-frame is found.

3.2.6 Message Decoder

The input handler moves each incoming event data fragment into the MPRACE buffer memory. To make this event data available for the level 2 or SFI farm it has to be requested with a four word message via PCI bus. The level 1 ID is used to identify the event data fragment inside the ROBIN. The same mechanism is used to delete event data fragments form the ROBIN buffer. A message containing a set of level 1 IDs initiates the delete procedure of all corresponding fragments. Furthermore messages can be used for ROBIN debug and control purposes. Currently only the possibility to read the event buffer memory is implemented.

To interpret incoming messages a decoder engine has been implemented. This checks the type of the request, activates the corresponding process in the target ROL slice or control module, and passes additional data.

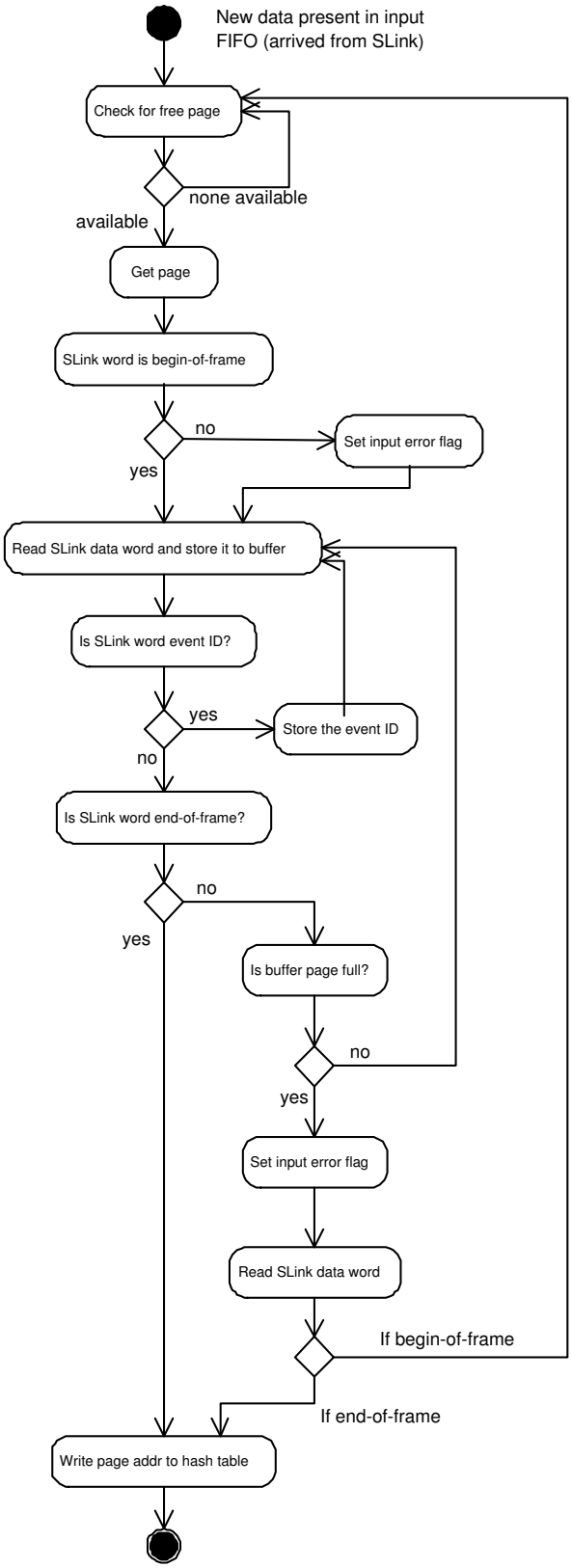

Figure 3.9: The algorithm of the input process. Incoming data is written into a buffer memory page and some basic sanity checks on the data format are done.

An incoming message arrives from the local-bus and is first written into a 256 word FIFO. This unblocks the local-bus during the processing of the message and simplifies the MPRACE ROBIN access by the host-PC software (see section 4.2).

The format of a message is shown in Table 3.1. It has been defined by the ATLAS ROS community for a usage with PCI bus and Ethernet based ROBIN devices [GGK02]. The first word, the request field, is divided into three values which names the requested service, the target ROL slice, and carries a sequence number. The service value specifies the function requested from the ROBIN hardware. It is an eight bit number. Table 3.2 shows the currently supported services. The ROL ID determines the target ROL for the requested operation. The MPRACE ROBIN supports input from four ROLs. Thus this field may carry a number between one and four. Finally the sequence number is a sequential number which is incremented for each message. It is considered to be used by a ROBIN which is directly connected to Gigabit Ethernet to detect lost request messages. For the current MPRACE ROBIN implementation it is not in use.

The second and the third word of the message, address field and extended address field, may carry the target address for ROBIN replies. Only the address field is used by the current MPRACE ROBIN implementation. The extended address field is reserved for the future transmission of 64 bit target addresses or 48 bit target Ethernet MAC addresses.

	Byte 3	Byte 2	Byte 1	Byte 0
Request Field	ROL ID		SEQ	SRV ID
Address Field	Address			
Ext. Address Field	Extended Address			
Data Field 1 .. n	1 .. n Data Words			

Table 3.1: The format of ROBIN request messages.

The remaining words of a message form the data field. Its format and the contained information depends on the requested service and are documented in Table 3.2. In case of an event data fragment request the data field contains only one word: the requested level 1 event ID. In case of a delete message a whole set of level 1 IDs is carried by the message. In this case the first word of the data field determines their quantity followed by each level 1 ID for deletion.

SRV ID	Service(Request) Type	Data Field
0x01	Get Event Fragment Data	1. Event ID
0x10	Delete Event Fragment Data	1. Number of Event IDs 2. Event ID 1 3. Event ID 2 ... N. Event ID n
0x1F	Delete all event fragment data	none
0X48	Write Event Data Buffer	1. Buffer Event Buffer Address 2. 1024 Data words
0X49	Read Event Data Buffer	1. Buffer Event Buffer Address

Table 3.2: Service / request types.

Beside the PCI bus messaging mechanism a set of simple registers provides control and statistics information and can be accessed directly from PCI. It is intended to access this functionality with messages too in the future to unify the MPRACE ROBIN access.

The VHDL message decoder algorithm reads the message data word by word out of the FIFO and interprets the values. The first word contains all initially required information to determine the target ROL slice and the requested service (event data request, delete or control request). The interpretation of service ID and target ROL leads in the activation of the target module inside the target ROL slice. The next two words, containing the reply address, are passed to the activated module as parameters. The activated module has to acknowledge its activation. After this the data field words are forwarded to the engine. This finalizes the message processing.

3.2.7 Request Handler

Requests for event data fragments, decoded by the above described message handler, are forwarded to the request handler module. The message decoder passes the event ID of the desired event and the target address for the reply to the request handler of the target ROL slice. This consists of two parts: two 16 word FIFOs and the main processing engine (a finite state machine). The module is shown in the schematic diagram in Figure 3.10.

The two FIFOs store up to 16 requested level 1 IDs and target addresses temporary. This allows the message handler in most cases to continue immediately even if the request handler engine is already busy.

In the next step the main processing engine picks an event ID and target address pair out of the FIFOs and locates the data fragment in the buffer memory by the lower 12 Bit of the level 1 ID (see section 3.2.3). The result is the start address of the event data inside the buffer.

According to the ATLAS data format, described in section 1.4, a 15 word ROB header has to precede the event data fragment coming from the buffer. Its format is shown in Figure 3.11. A 16 word FIFO inside the main processing engine is used to build this header. In parallel, data is loaded from the event buffer and stored temporary in another 256 word FIFO.

Finally the ROB header and later the event data from the buffer is sent to the DMA engine. A multiplexer switches the output of both FIFOs controlled by the empty flag of the ROB header FIFO. As long as there are still ROB header words left they are passed to the DMA engine. Only if the ROB header has completely left the processing engine the event data starting with the ROD header gets transmitted. The DMA engine needs two additional parameters when activated: the target address and the complete data count. The first arrives from the message decoder supplementary to the event's level 1 ID. The complete data count is calculated out of the data size in the event buffer plus the 15 word ROB header.

The major effort is the construction of the ROB header. It contains fix values, data from the ROD header which is part of the event data inside the buffer, and ROBIN status information. The first word of the ROB header is a fix value of 0xDD1234DD (see Figure 3.11). It is followed by the total size of the event fragment (event data size obtained from the buffer manager + size of the ROB header) and the ROB header size (15 words). The format version and the source ID are contained in the event data's ROD header. These words are copied by pushing them into the ROB header FIFO when they arrive from the event buffer.

Two status words are foreseen: one for the fragment status and one for the ROB status. The first reports if a fragment has not been properly received or even if it is not present in the

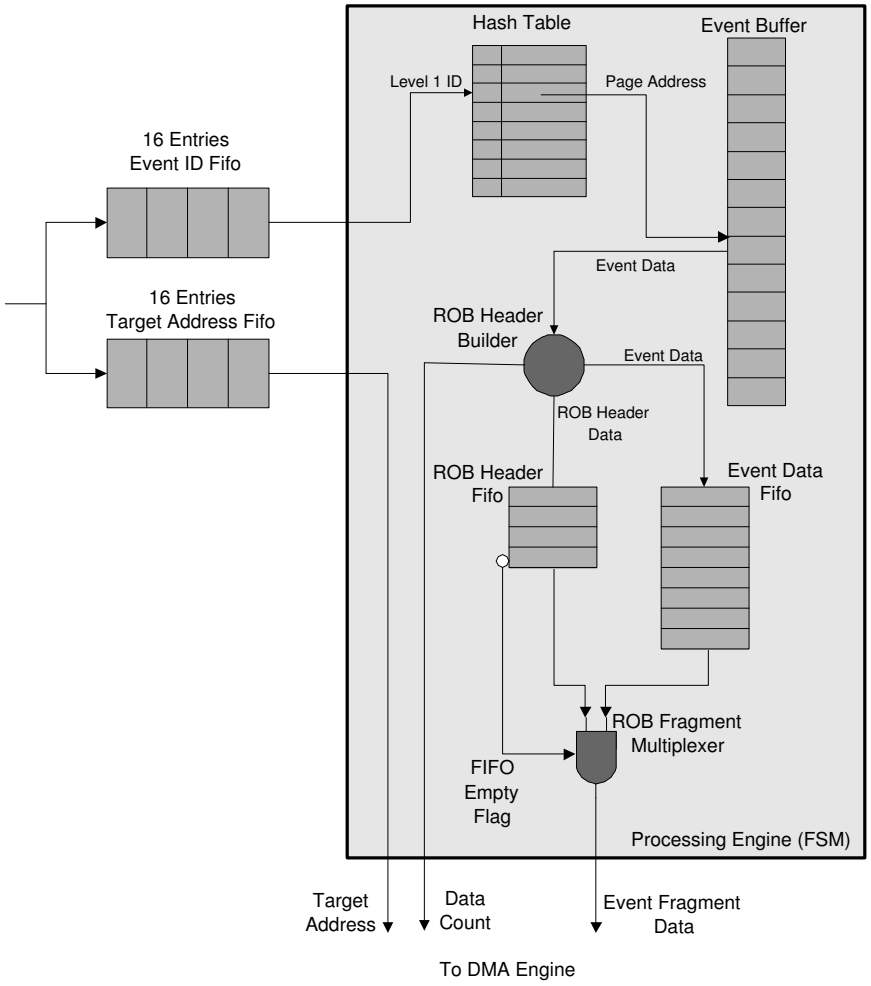

Figure 3.10: The MPRACE ROBIN request handler process.

buffer at all. The second allows the ROBIN to pass internal status information to the requestor. This is currently not in use, but could provide information about the filling state of the event buffer or the latest incoming event ID.

The offset element describes the offset address at which the event fragment, received by the ROD, starts. Finally some fragment specific entries get added which are again obtained from the ROD header when they arrive from the event buffer:

- The level 1 ID.
- The bunchcrossing ID which counts each event from the accelerator machine.
- The level 1 trigger type with information why this event has been accepted by level 1.
- The event type which is mainly used to specify calibration events.

If the requested event could not be found by its level 1 ID, a default ROB header is generated and sent back to the host-PC. The uppermost bit of the first status word inside this default header is enabled to signal the “fragment not found” error condition.

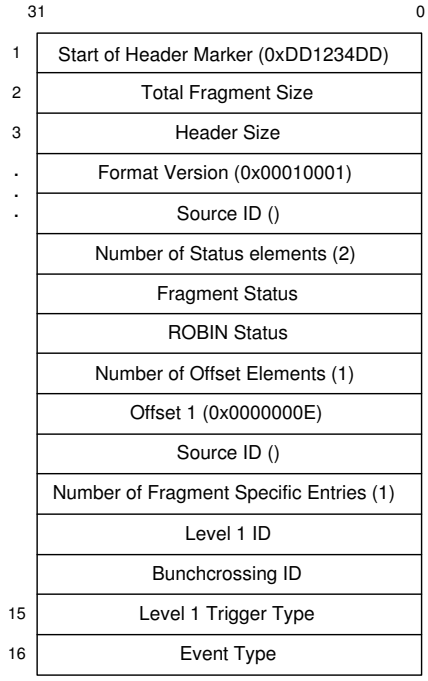

Figure 3.11: The ROB header format.

3.2.8 DMA Engine

The transmission to the PC's Memory is implemented inside a separate module: the DMA engine. Two alternative versions of this engine have been tested:

- The first one expects a pre-initialized PLX9656 DMA-on-demand and uses the flow control signal DREQ to start the transmission (see section 3.1). When data is available the DREQ signal is asserted and the PLX 9656 starts to fetch the event data. It is written sequentially into the DMA buffer inside the host-PC memory. This first version is a pull-scenario in which the FPGA is the passive and the PLX 9656 the active component. It is the initial implementation ported from the microEnable ROBIN (see section 2.3.3).

- The second DMA engine module uses the direct master feature of the PLX9656 (see section 3.1). Here the FPGA actively writes to the PCI bus through the pre-defined PLX 9656 address window. Thus the FPGA can randomly access the DMA memory within the host-PC. The target address for the data write operation is defined inside the request message.

In both versions, the software running on the host-PC has to detect the end of an event data fragment transmission. Therefore two mechanisms can be used by the software: interrupts or polling. Due to performance reasons (which are discussed in 4.2) the polling mechanism has been chosen.

Figure 3.12 shows the flow diagram of the DMA-on-demand DMA engine. On arrival of data the DREQ flow-control signals the PLX 9656 the presence of data. This starts to fetch (read) the data from the engine. The first transmitted word is the data count value. This enables the host-PC software to calculate the position inside the DMA buffer where the last data word is expected. Now all data words are transmitted. Finally a predefined, fixed value

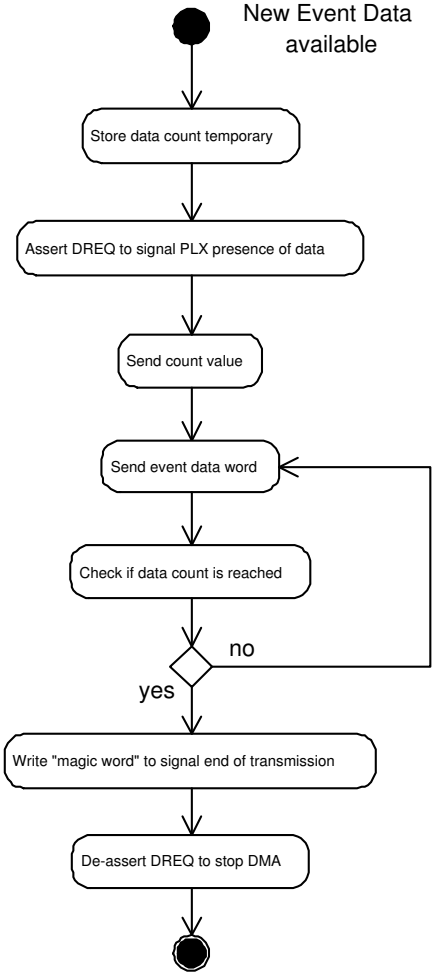

Figure 3.12: DMA-on-demand DMA engine flow.

word ("magic word") finalized the transfer. The host software can poll for this at the calculated end-address of the DMA transmission.

The main disadvantage of this first DMA scheme is the complex DMA buffer management inside the host-PC software It has to prepare the buffer properly by setting each word to zero (see section 4.2).

A simpler mechanism is implemented in the second approach. Here the FPGA writes actively into the PC's memory using the direct master feature of the PLX 9656 (see section 3.1). This allows the writing of the first word of a transmitted data packet in the end. The software knows the start address in advance and only needs to poll there. The flow diagram of this alternative second engine is shown in Figure 3.13.

Initially the target address and the total word count are stored temporary to make them available during the whole data transmission process.

The first word, the ROB start-of-header marker (see section 1.4), is also stored temporary. Instead a value of one gets written into the DMA buffer. This signals the host-PC software that the requested transmission has started which is useful for debugging purposes. Now the whole data packet (except the first word) is transmitted to the target DMA buffer until the total word count is reached. In the end the first word, which has been temporary stored before, is

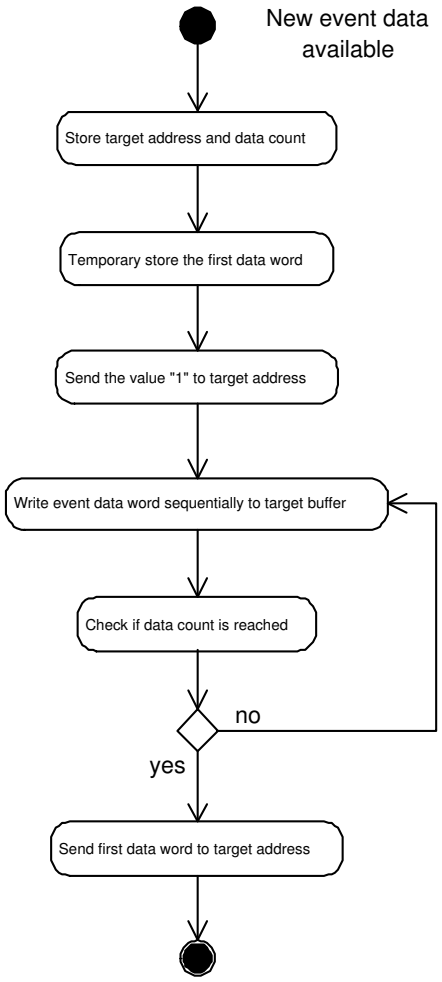

Figure 3.13: Direct master DMA engine flow.

sent to signal the end of the transfer.

This mechanism allows the host-PC software to define the address inside the DMA buffer where the requested event data has to be placed. The arrival of data is simply checked by polling on the first word of the buffer which has to be loaded to zero in advance.

A further discussion on the MPRACE ROBIN DMA transmission schemes from the software's point-of-view will be done in 4.2. Both methods have been tested with the MPRACE ROBIN and the result will be shown in chapter 5.2.

3.2.9 Delete Handler

The delete handler removes event data from the ROBIN's buffer management. It is activated upon an incoming delete message by the message decoder which passes a set of level 1 event IDs to identify the target event data.

Deleting events is a two step process. Figure 3.14 shows a schematic diagram of the delete handler implementation. The event IDs arrive in a block of several ten or hundred. These are temporary stored inside a FIFO with 128 entries. The deletion process is now done in two steps:

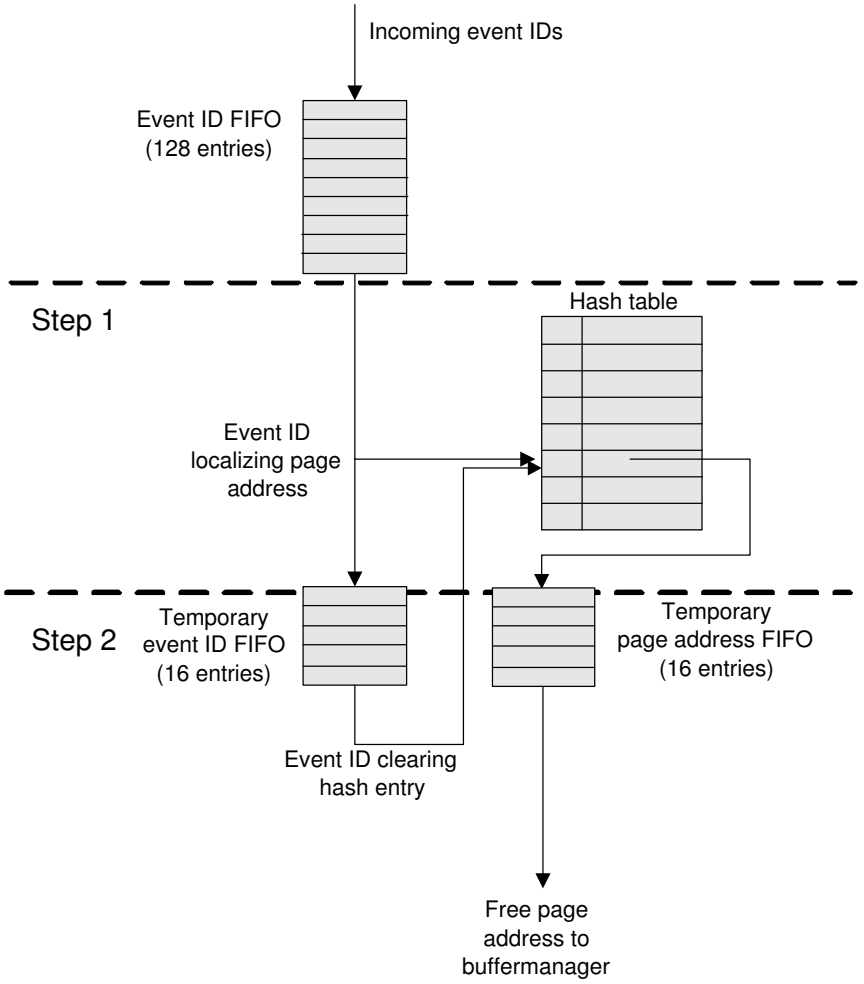

Figure 3.14: MPRACE Fragment delete engine.

Step 1: The engine picks a single level 1 ID from the FIFO and reads the corresponding page address from the hash table. If the hash table entry is zero the event is ignored and no further action is taken. Otherwise level 1 ID and page address are temporary stored in two parallel 16 entries deep FIFOs. This process continues until the temporary FIFOs are full.

Step 2: Event ID and page address are picked from the temporary FIFOs and a second access to the hash table invalidates the table entry by writing a zero. Finally the free page address is passed back to the buffer management and added to the free page FIFO.

To simplify this procedure a reply message upon a successful or un-successful execution is not passed to the requesting host-PC. This saves logic and bandwidth, but makes control and debug mechanisms impossible. Furthermore a matching of the level 1 ID is only checked using the hash key. Since the hash key for a number of event IDs is equal, this may lead into a deletion of a wrong event. A real check would require looking into the buffer and checking the level 1 ID stored within the ROD header of the event data fragment.

3.2.10 RAM and DMA Arbiter

The ZBT SRAM and the DMA output are accessed in the MPRACE ROBIN design by different modules at the same time. The SRAM, used as event data buffer, is required by the input handler, the request handler, and the delete handler. Data reads or writes may be requested by two of these modules or by all of them at the same time. The same situation is present at the DMA output to the local-bus / PCI bus. Several of the four slices and/or the controller module may want to send data to the host-PC.

In all of the above cases a number of users require the usage of one resource. This makes an arbitration of this resource, SRAM or DMA, necessary. Therefore two arbiter modules have been developed. They consist of a multiplexer and a control logic which resolves access conflicts.

The conflict resolution is based on a rotating priority scheme. Always the requesting channel with the highest priority gets a grant signal and the priorities are shifted after the client has released the resource again. This leads to a fair arbitration.

In case of the RAM arbiter each client has a guaranteed number of clock cycles available (currently 512) for RAM I/O. If this credit has expired the client may loose the grant and the RAM is assigned to another channel.

The DMA output arbiter is not able to remove the grant signal after a number of clock cycles because the DMA engine has to complete a data package transfer before a new can be started (see section 3.2.8). Thus a client is allowed to use the DMA resource to transmit one packet. After this, the DMA is re-scheduled to another client.

3.2.11 The Controller Module

The MPRACE ROBIN controller module comprises a set of registers and counters accessible via PCI. They monitor and control all other parts of the design. Furthermore an in-situ debug core is part of the controller module which is able to monitor signals after a pre-defined trigger event has occurred. It consists of a large FIFO to sample up to 128 signal values. They can be read via the PCI bus. The statistics and monitoring counters provide information about:

- The number of received and processed event data fragments.
- The number of received erroneous event data fragments.
- The number of SLink flow control activations (XOFFs).
- The number of free pages in the buffer management.
- The number of data request messages received via PCI.
- The number of delete messages received via PCI.
- The number of control messages received via PCI.

Within the controller module the parameters for the internal event data generators (see section 3.2.4) can be adjusted. This comprises the size of the event fragments and their generation frequency.

A very useful debug option, provided by this module, is the direct event buffer access. Any page, including the hash tables, may be dumped to PCI.

3.3 VHDL Design, Synthesis and Verification

All MPRACE ROBIN modules have been implemented and tested using the VHDL hardware description language. It has been selected due to the availability of various design tools and the local competence with this language within the Mannheim FPGA group. Developing the MPRACE FPGA firmware with VHDL is a five step process:

1. Development of the VHDL design description.
2. Verification.
3. Synthesis.
4. Place & Route.
5. Design Test.

Several iterations of this design flow have been necessary.

ActiveHDL from Aldec has been used for the first two steps of the design flow: development and verification of the VHDL description. This tool provides a useful integrated development environment for VHDL including project management.

For the design verification the development of a testbench is necessary. In case of the MPRACE ROBIN a ZBT SRAM and a PLX local-bus simulation engine was used. A simple version of the PLX local-bus simulation engine was already available from [Sim04] and has been extended and improved (to simulate the PLX DMA-on-demand and direct master DMA mode). SLink data input was not added to the simulation test bench because the internal data generator module provides a sufficient substitution.

After the verification, a netlist is generated within the synthesis step using the VHDL synthesiser Synplify from Synplicity. Finally this netlist is mapped to the FPGA architecture by a place & route tool from the FPGA manufacture Xilinx.

Number of ROLs	Slices	Block Rams	4 Input Logic	Flip-Flops
1	3.027 (21%)	25 (26%)	4.053 (14%)	2.941 (10%)
2	4.692 (32%)	31 (32%)	6.377 (22%)	4.609 (16%)
3	6.477 (45%)	37 (38%)	8.709 (30%)	6.351 (22%)
4	7.386 (51%)	43 (44%)	10.877 (32%)	7.942 (27%)

Table 3.3: FPGA Design Resource Utilisation for the MPRACE ROBIN firmware handling 1 to 4 ROLs reported by the Xilinx Place & Route tool.

This tool also reports the FPGA resources utilization. The values for the MPRACE ROBIN design are shown in Table 3.3 for one, two, three, and four readout links. The number of slices describes the main utilisation of the Xilinx Virtex II FPGA. A slice is the main element comprising two 4-input function generators, carry logic, arithmetic logic gates, wide function multiplexers, and two Flip-Flop storage elements. Four slices form a configurable logic block (CLB) which is connected to other CLBs by a fast switch matrix [Xil03].

The MPRACE ROBIN with four ROLs uses 51% of all available slices. In average 1468 are used per ROL, 1680 for all other logic (local-bus access, message decoder, DMA engine +

arbiter control logic). The number of 4 input logic elements show that not all of the slices are completely utilized. Otherwise this value would be twice the number of the used slices. Each ROL uses 6 BlockRAMs for FIFOs, 19 are required by the other parts of the design.

In the last step of the FPGA design flow the FPGA firmware is tested with the MPRACE hardware. A number of debugging facilities have been added to the design to assist this step. They are mainly placed in the controller module and have been already described in 3.2.11.

Two types of debugging aids are present. The statistics counters give general information about the data flowing through the design (number of incoming packets, messages, etc.) and are the first level of debugging. The second level enables to watch the signals within and between the various design modules. Buses and control signals can be monitored for a short period using the FIFO sampling tool described in 3.2.11. Finally an external logic analyzer has been frequently used to observe the PLX local-bus signals.

The current FPGA firmware runs with 96 MHz. Only the local-bus module is limited to the maximum MPRACE local-bus frequency of 64 MHz. This allows a PCI bus bandwidth of 256 MByte/s and a SRAM bandwidth of 384 MByte/s. Both are sufficient for ATLAS.

3.4 Summary

To implement a PC based ROS System which is able to handle a high number of ATLAS readout links a hardware component for input and event data buffering is required. The prototype development of such a component, called ROBIN, has been presented in the last chapter. It is based on a multifunctional PCI FPGA co-processor with the physical links on a mezzanine daughter board. To maximize the number of ROLs in the ROS PC, four receivers including buffers have been implemented and it has been shown that a mezzanine daughter board can be developed.

The FPGA firmware, which implements the main ROBIN functionality, is designed with VHDL and uses approximately 50% of the current FPGA. All four MPRACE SRAM banks are used to buffer event data arriving on four readout links. An average level 2 decision latency of up to 10,17 ms is possible with the current buffer. This matches the requested value, but may have to be extended to guarantee a save operation.

Incoming event data is delivered through the PCI bus interface on demand or deleted on request. The required software on the host-PC has to implement the ROBIN message passing and handle the network I/O to the rest of the data acquisition system. It will be discussed in the next chapter.

The current version of the firmware seems to provide sufficient I/O bandwidth to satisfy ATLAS needs. The verification of this is presented in chapter 5.

Chapter 4

ROBIN Messaging and Readout System Software Design

The MPRACE based ROBIN hardware, presented in the last chapter, provides the functionality to receive and buffer incoming event data on four ATLAS readout links. This event data has to be made available via PCI bus to the level 2 and SFI event builder farms on demand.

According to the ATLAS ROS baseline architecture (see section 2.3.4) multiple of these PCI ROBIN boards are placed in a host-PC system. This forwards requests to the ROBINs and performs local event building on the returning event data fragments.

This chapter describes the host-PC system and its software components: the main ROS application and the MPRACE ROBIN PCI bus message passing.

4.1 ROS Host PC

The host-PC system contains a number of MPRACE ROBIN boards. It has to

- receive data or delete requests via Gigabit Ethernet,
- pass theses requests to the ROBIN boards,
- combine the returning event data fragments,
- pass these combined fragments to the requestor via Gigabit Ethernet, and
- detect, control, and initialize the ROBIN hardware.

To test a reasonable number of MPRACE ROBINs (at least three or four) a PC with 4 – 6 PCI slots or more has to be used. Two main options are imaginable for that PC:

1. An “industrial” PC with a large number of 64 Bit / 66 MHz PCI slots implemented with PCI-to-PCI bridges[1]. These systems typically use a passive PCI backplane and a single-board computer inside one PCI slot. Backplanes with up to 11 high performance PCI slots are available [TPH].

[1] A PCI-to-PCI bridge is a transparent device to concatenate several PCI buses. It is used in most cases to increase the number of slots on a mainboard.

2. A server PC using a mainboard with multiple PCI buses connected to the main processor bus. Systems with up to 6 PCI slots distributed on four independent PCI buses are widespread and based on Intel's or Serverwork's Xeon processor chipsets. A 19 inch case may be used to make such a PC rack-mountable.

The first option allows a large number of ROBINs in a system. But since a number of PCI-to-PCI bridges concentrate the traffic to only one single PCI bus, the full 66 MHz / 64 Bit bandwidth of all buses can not be completely exploited. Beyond this, it is not clear at this point if a single system can handle such a high number of boards and readout links.

Thus the second option has been used as evaluation platform within this thesis. It is sufficient for evaluation tests and a system has already been available.

The PC, available for the tests, was based on a dual-Xeon mainboard (X5DP8-G2) from Supermicro with an Intel EZ7501 chipset. It provides four PCI bus segments with six 64 Bit / 66 MHz or faster PCI slots. The slot and bus configuration is shown in Figure 4.1.

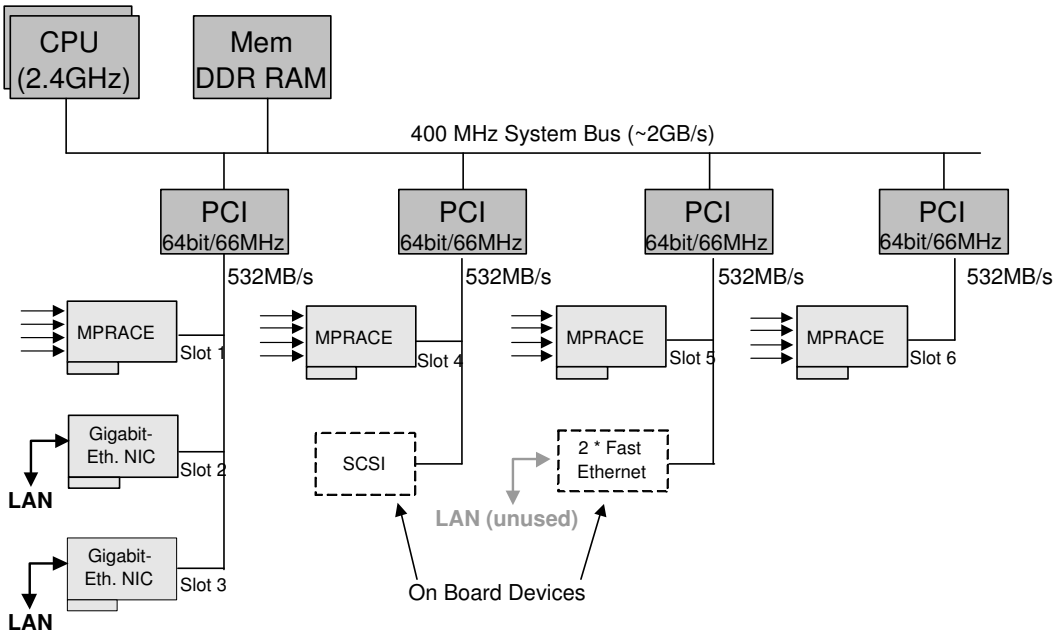

Figure 4.1: The ROS PC configuration used with the MPRACE ROBIN.

Three slots (1 - 3) share the same bus segment while the others are single slots distributed on different buses. Two internal component share the PCI bus with a single slot. There are two Fast Ethernet and one SCSI adapter present, but recent mainboards already come with two Gigabit Ethernet interfaces [sup]. Thus the PC can be equipped with up to four MPRACE boards and two Gigabit Ethernet adapters.

All PCI buses lead into the 400 MHz system bus which is able to sustain 2 GByte/s [Gor02]. The evaluation PC, used for testing, was equipped with two Xeon 2.4 GHz CPUs running the main ROS application.

4.2 ROS PC Software

4.2.1 Atlas Software Overview

The ATLAS trigger and data acquisition (DAQ) software has a number of different tasks:

- calculate trigger decisions.
- move event data and additional information.
- control the ATLAS trigger and DAQ system.

All of them are very demanding. Controlling the trigger/DAQ system requires the supervision of a huge amount of different hardware. There are huge number of processors which are standard PCs in most cases, networks, switches, and also custom electronics (in the level 1 trigger). Moving data also causes a huge effort due to the large number of nodes and high data rates. Finally the trigger software contains a number of algorithms to detect various physics events.

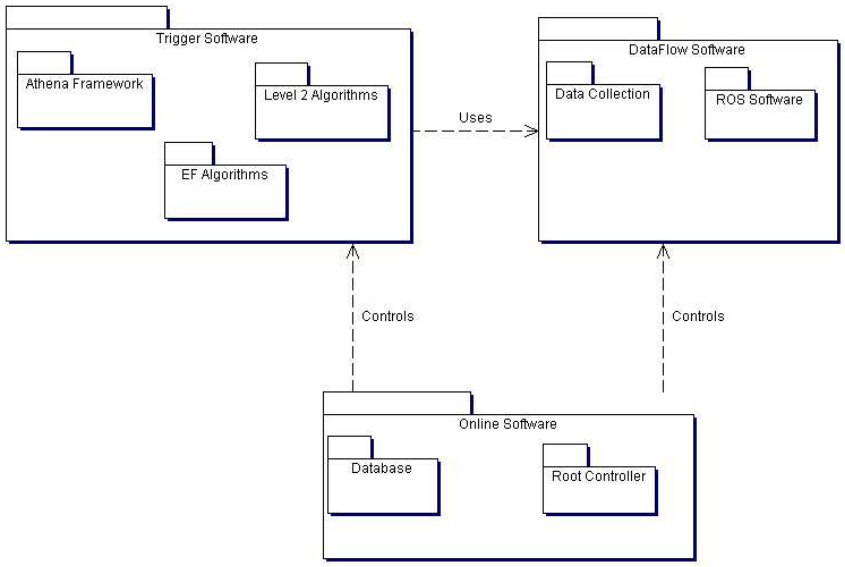

Figure 4.2: The ATLAS software packages and dependencies.

Figure 4.2 shows the ATLAS software packages, their dependencies, and the most important sub-packages. Three main packages are present: trigger software, dataflow software, and online software.

The trigger software package contains all trigger algorithms for level 2 and event filter (EF). The level 1 trigger is implemented in hardware and therefore not shown. All algorithms inside this package use the Athena [Ath01] framework for common tasks (e.g. memory allocation, data parsing and formatting, ...).

The trigger software requires the dataflow package for all data movements between the different DAQ components. It comprises the data collection software for network interface

access and safe network message passing and the ROS software. The latter contains all software for the readout subsystem and will be explained later in this chapter.

Finally the online software package contains the main ATLAS trigger and DAQ control software. This comprises a database for all components and a root controller which communicates with local controller applications running on each DAQ system node via Ethernet network.

As shown in Figure 4.2, the ROS software sub-package, used on all ROS PCs, is part of the dataflow package. It will be explained in the next sections.

4.2.2 ROS Software Layer Model

The application running on the ROS host-PC performs all ROS PC task mentioned in section 4.1. It is contained in a software package, the ROS software (see previous paragraph), which has been developed at CERN [ea02]. This package comprises the main application and libraries for memory management, online software connection, network I/O, multithreading, and ROBIN communication through an abstract, hardware independent interface. This interface has been implemented to communicate with the MPRACE ROBIN.

The ROS software requires Linux OS. It can be divided in several layers shown in Figure 4.3. Various interfaces are defined for the communication between these layers.

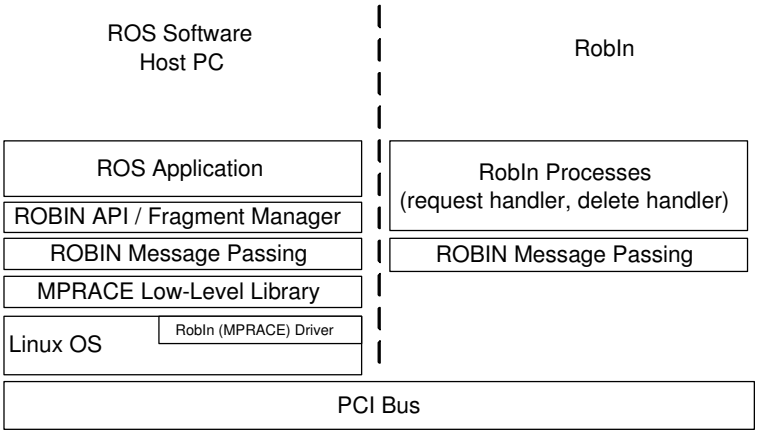

Figure 4.3: Software layers.

The uppermost contains the multithreaded ROS application. It executes the request distributing to the ROBIN boards and the event data fragment collection algorithm. This layer has been developed at CERN. Only some modifications have been done within this thesis.

The second layer implements the abstract interface "*FragmentManager*" [ea02] for the MPRACE ROBIN communication. The interface provides basic ROBIN functions: hardware configuration, event data requisition, and event deletion.

The implementation of this FragmentManager interface is done with the MPRACE ROBIN Message Passing mechanism. This comprises the sending and receiving of messages to or from the hardware via PCI. It is the third layer of the ROS software.

The low-level communication with the MPRACE board is implemented within a library which forms the next layer. It provides an interface for basic MPRACE read and write functionality via programmed I/O or direct memory access (DMA). Additionally the complete

MPRACE hardware control is available: loading FPGA firmware, programming clock frequencies, etc. .

Some MPRACE access operations need assistance of a device driver embedded in the Linux OS. These are: basic PCI hardware detection, mapping of PCI bus addresses to the user process virtual address space, and allocation and preparation of memory for DMA operations.

The fundamental layer is finally the PCI bus which "connects" the software part in the ROS PC with the VHDL processes inside the MPRACE FPGA hardware. The latter can also be abstracted and expressed in a similar layer model. This is also shown in Figure 4.3. The MPRACE ROBIN firmware can be divided into two layers: one for ROBIN message passing and one containing the main ROBIN processes.

The ROBIN message passing layer corresponds to the VHDL message decoder and DMA engine module (see section 3.2.6 and 3.2.8). The ROBIN processes layer comprises all other VHDL modules for request handling, delete handling, input handling, and buffer management. It can be compared to the application layer in the ROS software.

The next sections describe the various layers of the ROS software in more detail starting with the application layer.

4.2.3 Main Application and Fragment Manager Interface

Overview

The main ROS application, the first layer in the previously described model, is integrated into the ROS software framework [ea02]. This framework provides various modules for memory management, online software connection, network I/O, multithreading, and ROBIN communication. The ROS software framework has been developed in C++; the various modules are object-oriented class libraries.

The ROS PC application is based on a multithreading concept which is intended to optimize I/O extensive parts. A number of threads, responsible for network I/O and ROBIN communication, are executed and share the ROS PC's CPU. If one has to wait for data from a hardware component (e.g. the MPRACE ROBIN) it forces a task switch and releases the CPU. The various tasks, their dependencies, and communications are shown in Figure 4.4.

There are four types of threads. Each may have multiple instances running on the PC system.

TriggerIn The TriggerIn thread watches the Gigabit Ethernet connection for incoming event data or event delete requests from the level 2 trigger or SFI event builder Farm. The requests are handed over to the request handler thread(s) using a Mutex[2] protected queue object.

Request Handler The request handler thread executes the incoming requests from TriggerIn. In case of an event data request, the target MPRACE ROBIN boards are accessed and event data request messages are sent via PCI. All returning event data fragments are combined to one ROS Fragment (partial local event building) and passed over to the DataOut thread. In case of a delete request all ROBINs in the system are asked to delete the set of level 1 IDs contained in the request message. Multiple of these threads are

[2] A Mutex (mutual exclusion) protects a shared resource in a multithreaded environment. It takes care that only one thread can access the resource at the same time.

present. If one of them has to wait for a ROBIN reply another has the chance to get access to the CPU.

DataOut The DataOut thread executes the output to the Gigabit Ethernet network. It takes the data queued by a request handler thread and passes it to the Linux operating system via the network socket interface. Linux adds the protocol headers (UDP and IP) and activates the Gigabit Ethernet network adapter hardware.

Control and Error This thread manages the control, monitoring, exception and error handling by sending messages and status reports to the ATLAS online control software. In case of a critical error the application may get terminated.

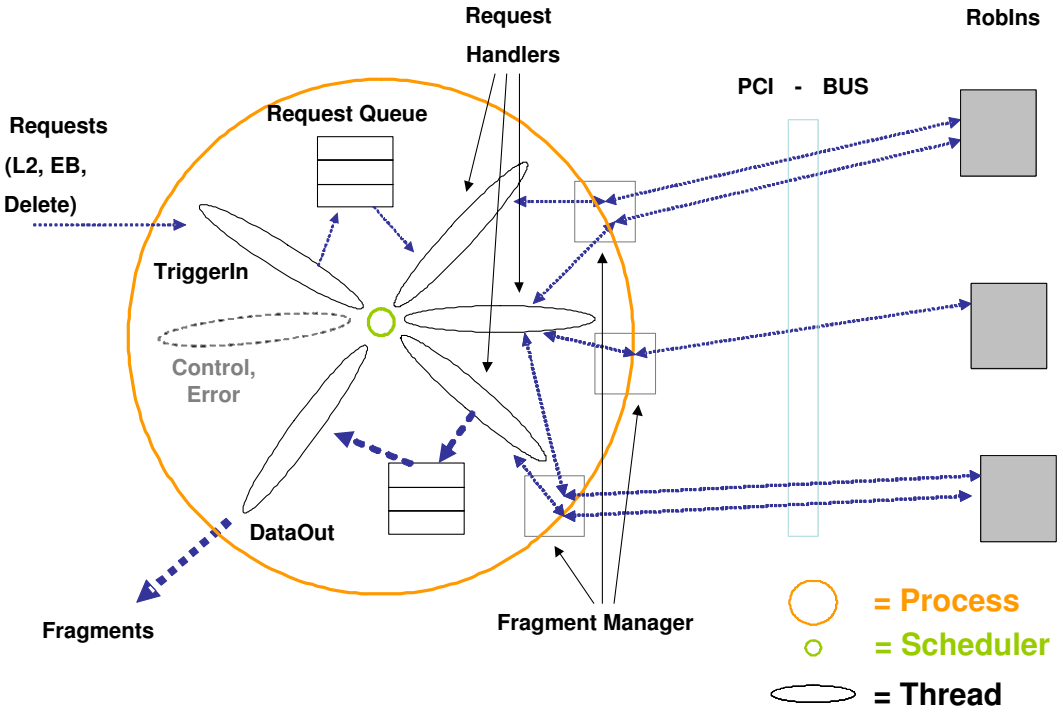

Figure 4.4: The threads of the ROS software main application.

Only the request handler threads perform MPRACE ROBIN accesses. Therefore the *FragmentManager* interface is present with a set of functions for the basic ROBIN operations: board configuration, event data request, and event data delete. This interface is described in the next paragraph.

ROBIN Access with the Fragment Manager Interface

The fragment manager is a C++ class library for ROBIN access. It has been introduced in the software layer model in Figure 4.3.

An abstract base class *FragmentManager* defines an interface to request the different ROBIN services independent of the used hardware. For each ROBIN hardware flavour a new class has to be developed which inherits the interface from the *FragmentManager* base class and implements its methods.

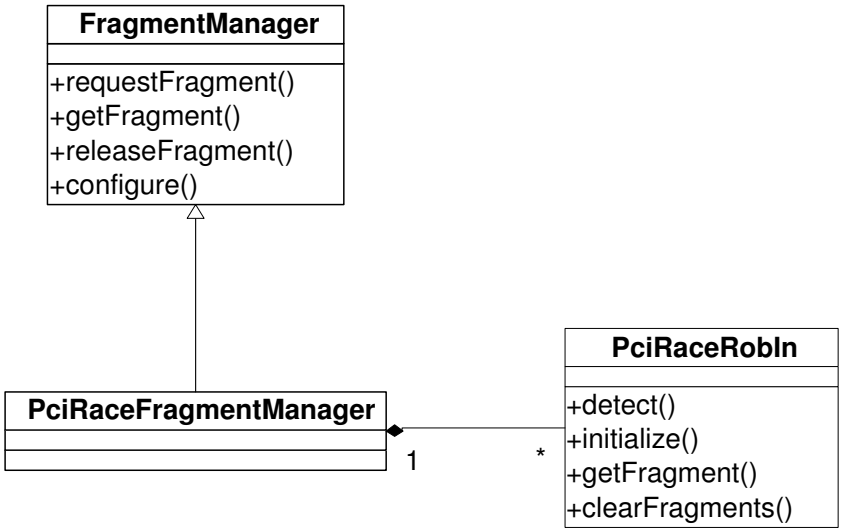

Figure 4.5: The PciFragmentManager.

This has been done for the MPRACE ROBIN hardware. Figure 4.5 shows the resulting class diagram. The specialized class *PciRaceFragmentManager* has been inherit from the *FragmentManager* base class for the MPRACE ROBIN. A third class, *PciRaceRobIn*, is used to implement the four *FragmentManager* methods. It encapsulates the hardware access and provides methods for each message type to the MPRACE firmware (see section 3.2.6). Furthermore it prepares the DMA memory to enable MPRACE reply messages.

The present *"FragmentManager"* methods are:

void configure(DFCountedPointer<Config> configuration)
Creates an object of the class *PciRaceRobIn* to enable MPRACE ROBIN hardware access. Configures the board by loading the FPGA firmware file, setting the input source (SLink or data generator), and programming the data generator if necessary. This method also allocates DMA memory and registers it on MPRACE (sets the PLX 9656 Direct Master window).

int FragmentManager::requestFragment(int eventID)
Requests an event data fragment from the MPRACE ROBIN. This method allocates a target buffer for the event data fragment reply and sends a request message to the hardware. Requests are stored in the MPRACE ROBIN internal request queue with space for up to 16 event level 1 IDs and the target addresses (two FIFOs, see section 3.2.7). The method must take care that this queue never exceeds 16 outstanding requests. In the end a ticket, which identifies the request, is returned. The method does not wait for any ROBIN reply. It is non-blocking.

ROBFragment FragmentManager::getFragment(int ticket)*
Waits for an event data fragment previously requested by *requestFragment*. This method checks the complete arrival of an event fragment from the ROBIN hardware. If the data has not been completely transmitted, the method forces a task

switch to give other threads a chance to execute their code on the CPU. Otherwise the status value inside the event's ROB header is analysed and in case of a valid event data fragment a pointer to the buffer is returned. In case of an empty fragment a zero-pointer is given back.

void FragmentManager::releaseFragment(const vector<unsigned int> level1Ids)*
This method sends an event delete message with a list of event IDs to the ROBIN. Typically 100 event IDs are deleted at once. The MPRACE ROBIN firmware uses a FIFO to store the event IDs until the delete instruction can be executed. The method returns immediately when the message has been sent. Currently no error check is supported.

For each of the four readout link (ROL) in a ROS PC, attached to a MPRACE ROBIN, one *PciRaceFragmentManager* object is created on start-up. To initiate an action on the hardware, a request handler takes a reference to the dedicated PciRaceFragmentManager object and calls then interface methods (see Figure 4.4).

The Request Handler Thread

The request handler thread is a part of the first ROS software layer. It is one of the core elements of the main application. The request handler algorithm makes extensive use of the *FragmentManager* interface presented in the previous paragraph.

The thread picks request messages received by TriggerIn from the protected queue object. Each of them asks either for event data from multiple readout links, distributed over one or more ROBINs, or to delete event data on all ROBIN boards. Since each ROL has one corresponding *PciRaceFragmentManager* object, the request handler algorithm takes all required objects and uses the previously described interface to execute the requests.

The performance of the ROS PC system depends mostly on the event data request and delete execution. These arrive in most cases during normal operation. Besides network and software effects, the ROS performance is heavily influenced by the ROBIN hardware access.

Deletes are simply sent with the appropriate message to the ROBIN hardware. Event data requests are more complicated because the software has to wait for the reply of one or more ROBIN boards. The request handler algorithm should, for an optimized ROS performance, use this delay for other tasks. It should not poll until the event data reply from the ROBIN has finally arrived. This goal is supported by the non-blocking *FragmentManager* interface and the MPRACE ROBIN level 1 ID request queue. Both allow sending a request message to the hardware, do something else, and come back from time to time to see whether the event data has already arrived. This allows both, the ROS PC software and the ROBIN, to work in "parallel".

Thus the target of the request handler thread should be to keep the ROBIN hardware as busy as possible by filling its hardware request queue. Two types of request handler algorithms have been tested within this thesis. Both try to follow the above mentioned guideline, but with two different approaches.

1. The first request handler approach is contained in the ROS software as it is provided by CERN. It handles a number of requests from TriggerIn (outstanding requests) in parallel threads.

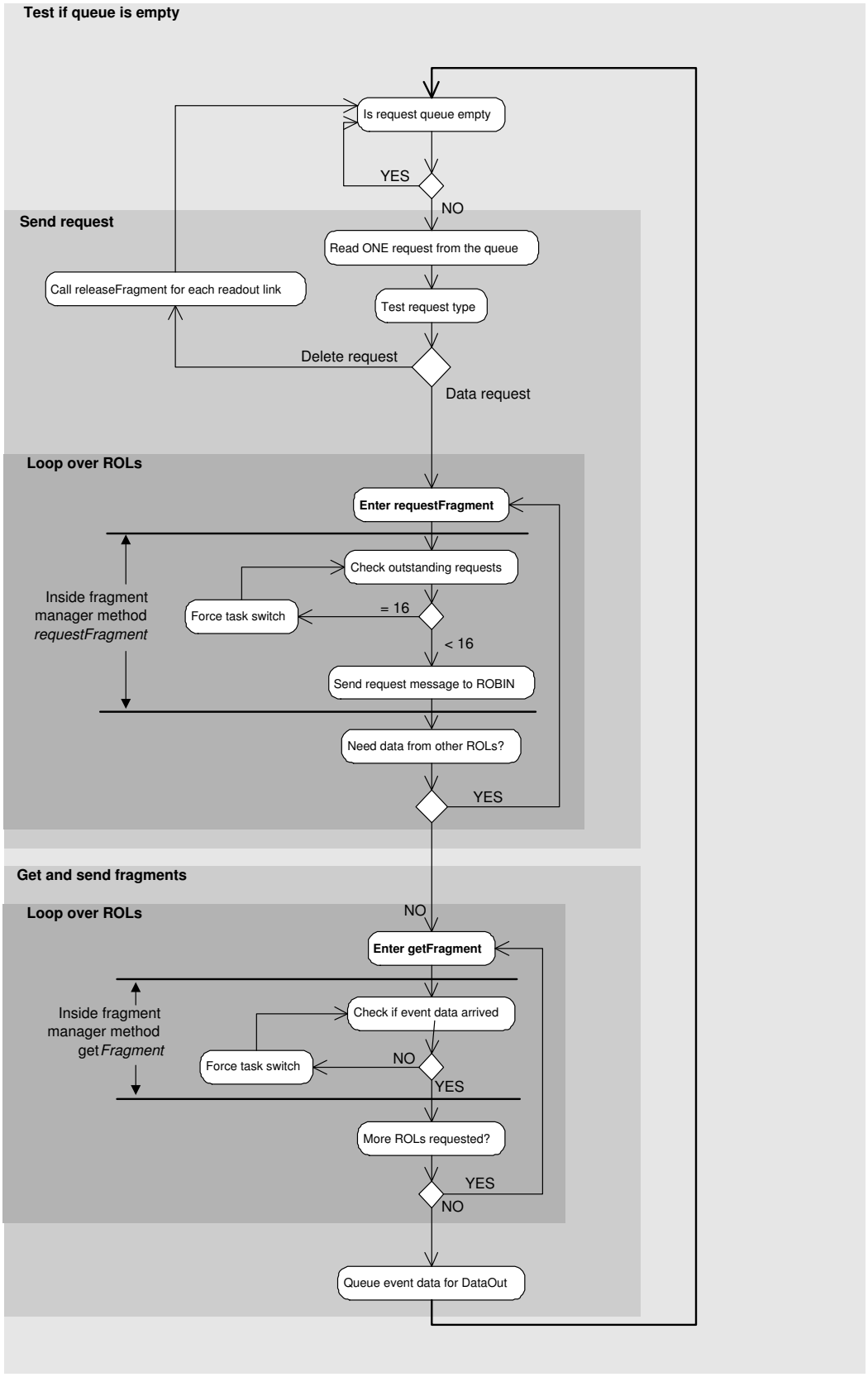

Figure 4.6: The original CERN request handler algorithm.

2. The second approach has been added to the original ROS software within this thesis. It is executed by only one thread, but with a more complex algorithm (compared to the original CERN request handler) to optimize the ROBIN hardware utilisation.

Both versions differ only in the execution of event data requests. Event deletes are processed in the same way.

Figure 4.6 shows the request handler thread algorithm originally included in the CERN ROS software. It is divided into two main parts: the sending of the request to the ROBIN hardware and the retrieving of the reply.

The first part is entered if the queue from TriggerIn is found not empty. In case of an event data request all required ROLs on the MPRACE ROBINs are asked for the event data using the *FragmentManager* method *requestFragment*. When entering *requestFragment*, a check of the number of outstanding requests is done first to prevent a congestion of the MPRACE hardware queue. If this exceeds 16 the algorithm forces a task switch to another thread. Otherwise the event data request is sent to the ROBIN board. The second step checks for the arrival of event data from each requested ROL. If data is not present, again a task switch is forced. Finally all fragments are combined to one ROS fragment and queued for DataOut.

Multiple of these algorithms are running as parallel threads in the ROS system. The forced task switch when waiting for the ROBIN permits other request handler threads to use the CPU for new ROBIN hardware requests, processing of replies, or handling Gigabit Ethernet I/O. This allows the above mentioned parallelism between ROBIN and ROS PC software. The main disadvantage of this multithreading approach is the task switch overhead and the impossibility to control the program flow between the request handler threads. It can not be determined after any task switch which thread is executed next.

The second algorithm approach, shown in Figure 4.7, eliminates these disadvantages. It is executed only in one thread. Requests from TriggerIn are accessed as long as there is sufficient space in the MPRACE ROBIN hardware queue. Again the *FragmentManager* method *requestFragment* is called in a loop over all required ROLs. If the maximum number of outstanding requests has been reached (or if there are no more requests from TriggerIn) the algorithm starts to check for ROBIN replies. This part, shown in the right half of Figure 4.7, is equal to the original CERN code. The *getFragment* method is called and waits for a message containing event data. Now for each returning event fragment the algorithm tries to send a new request as soon as possible to keep the ROBINs request queue filled. The result is that no task switches have to be done while waiting for reply data. Since it is tried to fill the ROBINs hardware queue first, the check for reply data is done on requests which have been sent some time ago. This decreases the wait duration and the time inside the polling procedure. But if *getFragment* has been entered the algorithm waits until the ROBIN reply has been arrived. No other thread (DataOut, TriggerIn) can get access to the CPU until the thread's time slice[3] has expired.

Both request handler algorithms have been tested within this thesis and the results are shown and discussed in the next chapter.

4.2.4 ROBIN Message Passing

The *FragmentManager* interface used by the previously described request handler threads has been implemented to communicate with the ROBIN hardware. This communication is based

[3]The Linux scheduler assigns each thread and process a fixed time slice. If this has expired another task gets access to the CPU.

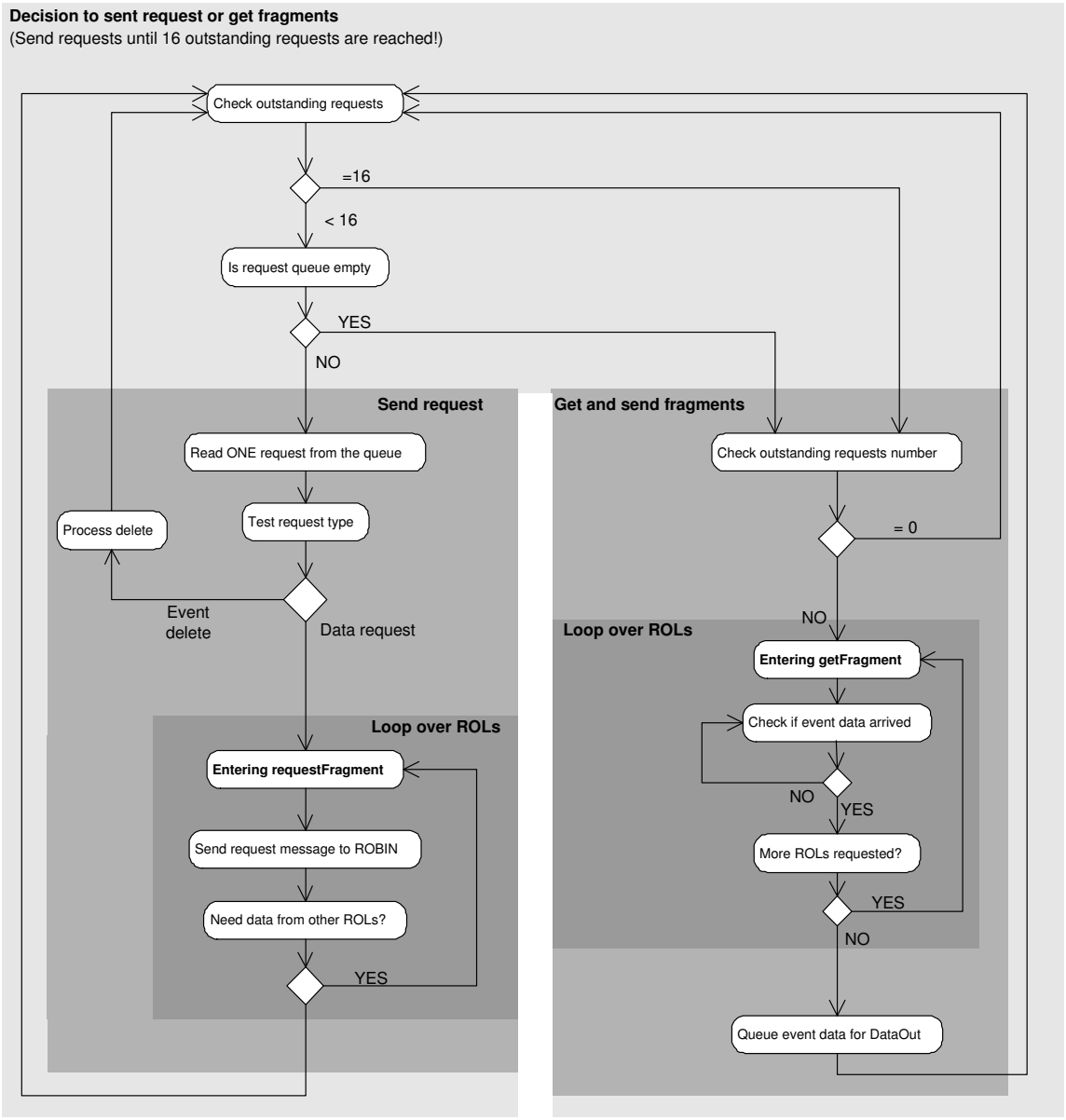

Figure 4.7: The modified request handler algorithm.

on the exchange of messages via the PCI bus. Messages from the ROS PC to the MPRACE ROBIN are used to request a service. Their format has already been described in 3.2.7. Messages from the ROBIN to the ROS PC contain event data fragments. They are transferred using PCI bus master DMA initiated by the ROBIN's DMA engine (see section 3.2.8). Two different approaches to transfer the ROBIN reply messages have been implemented and tested. The main difference between them is the component executing the DMA operation. In one case the PLX 9656 DMA-on-demand mode is used to fetch the message data from the FPGA in the other case the FPGA is the master component and can arbitrary accesses the PCI bus.

The following paragraph describes how messages are sent to the ROBIN. Both options for the reverse direction are presented in the two subsequent paragraphs.

Sending Messages from the Host-PC to the MPRACE ROBIN

Sending messages to the MPRACE ROBIN hardware can simply be done with programmed I/O (PIO) (see section 3.1). The message has to be constructed in advance according to the format described in 3.2.6. Then the resulting data block is written word by word to the 256 word message-input-FIFO in the MPRACE hardware (see section 3.2.6).

Alternatively the PLX 9656 DMA engine can be programmed to fetch the message from the PC's memory (see section 3.1). This requires at least four PIO accesses for programming the PLX 9656 DMA registers [Tec].

The PIO option is only intended for short messages (e.g. event data requests) since it prevents the host-PC's CPU from other tasks. For larger messages the second option, the PLX DMA engine fetching message data from the host-PC's memory, should be used. Due to the programming overhead it is only efficient for message sizes of several 100 words or more.

To assure that the message FIFO on the MPRACE FPGA does not overflow, its filling state is connected to the PLX DMA-on-demand flow-control (see section 3.1). This suspends the DMA operation if the FIFO contains more then 192 words.

But this flow-control is only active during DMA operations. In case of PIO an overflow is still possible. Since messages are continuously processed by the design and even after the DMA has already been suspended, the FIFO has still sufficient space for additional 16 event data request messages. This may be the reason why messages have never been lost during the tests, even under high load. Nevertheless a control mechanism should also check the FIFOs fill level when data is sent with PIO accesses.

MPRACE ROBIN Reply Messages with "DMA-on-demand"

The return channel from the ROBIN hardware to the host-PC is more extensive. Messages containing event data fragments are transferred from the MPRACE ROBIN hardware to the ROS PC.

Since this happens only after a prior request message the simplest solution would be to read the message data from the ROBIN hardware via PIO. This is very ineffective because it engages the host CPU. Furthermore PIO measurements on the above presented PC do never exceed a bandwidth of 4 MByte/s.

Another option would be to program the PLX 9656 DMA engine to transfer the reply message for each event after the request message has been sent. This adds the DMA initiation overhead to the message and event data fragment transfer time. Furthermore it necessitates the host-PC software to wait until the reply has been arrived and the DMA operation has been finished. Measurements, which will be presented in 5.2, show that this approach is very ineffective for the target message size of approx. 1 kByte.

Thus two methods have been developed which enable the MPRACE ROBIN to sent messages with event data fragments into the PC without or with only a small intervention of the host-PC software. This removes additional overhead per reply message transfer and makes the previously described request handler thread algorithms possible.

The main problem of these transfer mechanisms is to notify the ROS PC software about the data arrival. Two options have been considered for this:

1. Using a PCI interrupt.

2. Polling the DMA Buffer memory.

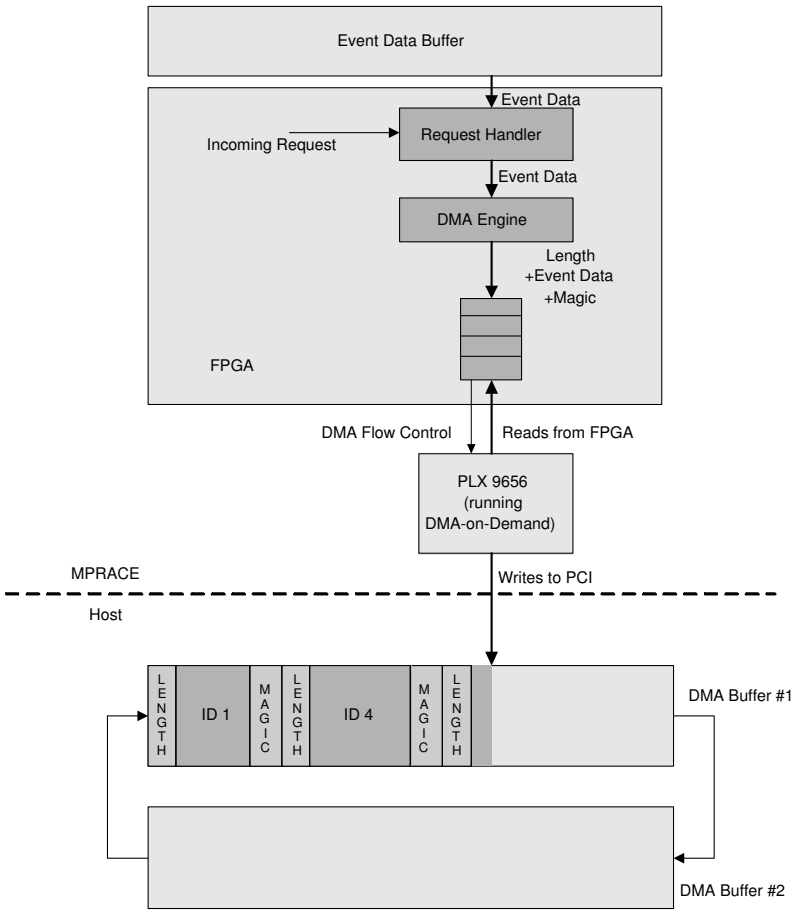

Figure 4.8: Data transfer between MPRACE ROBIN and host using DMA-on-demand.

The interrupt method is the common approach to tell PC software about an incoming event from a PCI device. In this case the hardware raises an interrupt signal which forces the host-PC CPU to break its current process and run a special interrupt service routine. This can signal the user software the concurrency of the interrupt condition.

Contrary the poll method looks permanently on the DMA memory for a data change. This can be used by the hardware to signal a condition to the user process (e.g. the arrival of data). The poll method requires setting the memory content to a pre-defined value in advance to properly detect the signal. Furthermore it consumes a lot of CPU resources.

Tests have shown that the frequency of PCI interrupts is in the order of 100 kHz [Kug04]. This is not sufficient for a ROS, handling 12 or more readout links. The requirements, presented in chapter 2.1, demand 10 kHz event data rate per ROL. This creates in 120 kHz event data rate flowing over PCI in a system with 12 ROLs.

Thus a DMA memory polling mechanism has been developed for the ROBIN return channel. It depends substantially on the MPRACE DMA features. As already mentioned before, two DMA variants, based on different features of the MPRACE PLX 9656 DMA engine, have been tested. One uses the DMA-on-demand feature of the MPRACE PLX 9656 PCI bridge chip (see section 3.1). The resulting communication mechanism is described in this paragraph. The second variant is based on the more flexible PLX 9656 Direct Master DMA. It will be explained in the next paragraph.

Figure 4.8 shows the co-operation between the MPRACE and the host-PC for the first alternative with DMA-on-demand. The host-PC initializes the DMA operation on a large buffer. It is able to take several hundred messages containing event data packets. The FPGA firmware uses the DMA flow-control signal to suspend the data transfer if no message has to be transported.

When event data has been requested by the host-PC the ROBIN's request handler module fetches event data out of the MPRACE buffer and forwards it to the DMA engine (see section 3.2.6 and 3.2.8). The latter adds the complete data length at the beginning and a pre-defined "magic" word at the end and pushes everything into a small 32 word FIFO. This activates the PLX 9656 DMA with the FIFO "not empty" condition.

On the host-PC, data arrives sequentially in the DMA memory. Since the size of messages vary, it is necessary to track the DMA memory and calculate the start and endpoint of each message. Therefore the size of each message is transmitted in the first word. This allows the calculation of the position where the final data word of a message will appear. The next message will start at the succeeding word.

For a proper recognition of arriving data each word in the DMA memory has to be loaded with a pre-defined value prior to the DMA initiation. Thus a simple loop fills the complete memory with zero before it is used.

If there is not sufficient space left in the DMA memory to transfer another message from the ROBIN hardware, the PLX 9656 DMA operation is aborted. After this, the present memory can not be immediately used for DMA again. This would require to completely overwriting it with zero which would also delete all recently arrived data. Thus a second Buffer has to be used.

This concept has some advantages and disadvantages for performance and usability:

- Data transmission from the ROBIN to the host-PC has no overhead per reply message. No DMA registers have to be programmed except when a DMA buffer has become full. This distributes the DMA initialisation overhead to a number of several hundred reply messages.

- The mechanism does not require any action by the host-PC except watching the buffer and re-initiating the DMA from time to time.

- No polling on PCI is required.

- No interrupts are required.

- But the DMA buffer has to be erased (filled with zeros) before it can be used. This reduces the reply message transfer rate because it has to be done frequently.

- Data packets appear sequentially inside the DMA buffer. The host-PC and also the ROBIN have no influence on the position of a data packet within the memory.

The last two items, which are the main disadvantage for the DMA-on-demand message passing scheme, disappear with the second communication approach.

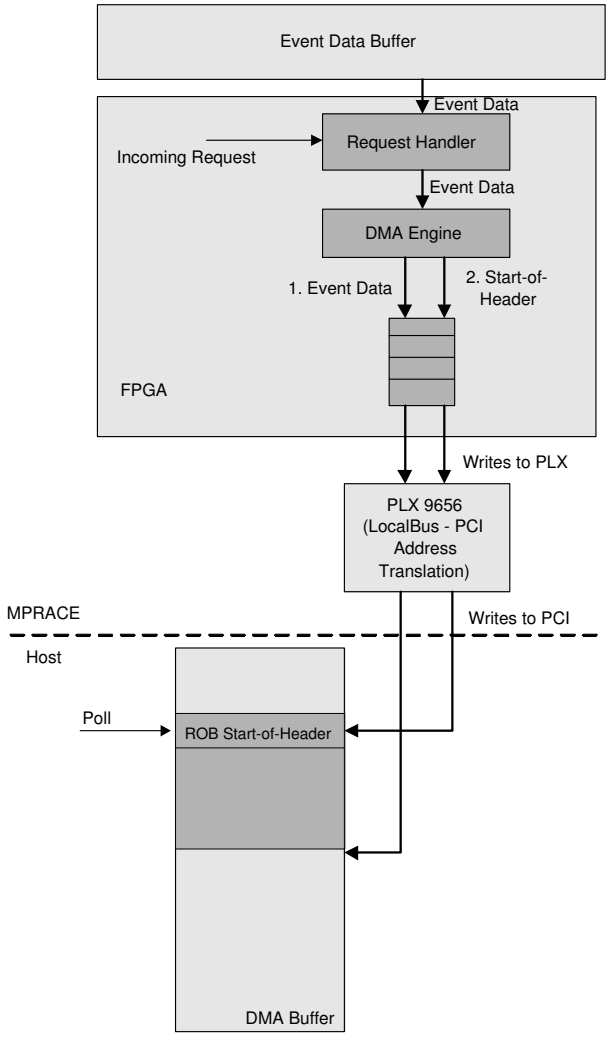

Figure 4.9: Data transfer between MPRACE ROBIN and Host using Direct Master DMA.

MPRACE ROBIN Return Messages with ”Direct Master” DMA

The second approach to transfer the MPRACE reply message uses the Direct Master DMA feature of the MPRACE PLX 9656 PCI bridge (see section 3.1). It allows the FPGA, as a master device on the MPRACE local bus, to randomly access the main memory through the PLX. Reply messages, containing event data fragments, must not sequentially arrive in the PCs DMA buffer. They can be placed at any location defined by the software and submitted as a parameter within the service request message.

Figure 4.9 shows the communication mechanism. Upon an incoming request message the request handler module inside the MPRACE ROBIN FPGA fetches the required data from the MPRACE event buffer and passes this to the DMA engine (see section 3.2.6 and 3.2.8). There, the DMA transfer process is divided into two parts. The first part forwards the reply message to an output FIFO. Thereby the first word is replaced by a value of one. In the second part the first word gets overwritten with the original value to signal the host-PC the completion of the data transfer.

The software on the host-PC polls for the arrival of the reply message and the contained event data. Therefore it sets the first word of the target DMA buffer location to zero. If this

value changes to the ROB Start-of-Header marker (see section 1.4), which introduces a event data fragment from the ROBIN, the ROBIN reply has been completely arrived.

This scheme does not need any DMA buffer preparation or DMA initialisation during the run time of the ROS software application. Everything can be done in advance. Only the first word of the target buffer for the reply message must be set to zero before sending an event data request.

It will be shown in the next chapter that this results in a major performance improvement compared to the previously presented messaging scheme.

4.2.5 Low-Level MPRACE Library and Device Driver

The two lowest layers in the model in Figure 4.3 contain the basic libraries for the MPRACE hardware access and a device driver embedded into the Linux kernel.

The MPRACE low-level library covers the programming of basic hardware functionalities. These are:

- Detection and initialisation of the MPRACE hardware.
- Uploading the FPGA firmware via PCI.
- Programming the on-board clock system.
- Data read and write to the FPGA co-processor via programmed I/O (PIO).
- DMA memory allocation and programming of the PLX DMA registers.

The library is implemented as object oriented code written in C++. To get access to a MPRACE board an object of the top-level interface class *race1* has to be created. It provides a set of methods for all frequently used operations.

PCI bus I/O within this library is done with memory mapped I/O. Therefore a PCI address area is mapped into the virtual address space of the user's application process. This can only be done in the restricted "kernel" mode of the Linux OS [RC01]. A device driver has to be loaded to the Linux kernel for these operations. The driver developed within this thesis supports:

- PCI hardware detection and allocation.
- Mapping of PCI address areas to the virtual address space of a user process for memory mapped I/O.
- Allocation of continuous memory for DMA.

The driver has been developed in C as a Linux kernel module [RC01] and can be loaded or unloaded during Operation system's run time.

4.3 Summary

The last chapter introduced the PC system used for the ROS application. It contains the MPRACE ROBIN boards, mediates between the level 2 and SFI event builder farm, and performs local event building on the event data fragments coming from the readout links.

A dual Xeon PC with four PCI segments and six PCI Slots has been chosen as a test system. Up to four MPRACE ROBINs can be plugged into this system. Two Gigabit Ethernet adapters provide the connectivity to the trigger and event builder farms.

The software application running on the PC uses a software framework provided by CERN. This ROS software comprises the main application code and libraries for network I/O, event memory allocation, multithreading, and a hardware independent interface for the ROBIN access. The main application uses multithreading for its different tasks: trigger input, ROBIN request handling, and data output. The handling of incoming requests is originally done in a number of concurrent threads. To remove overheads due to task switches, a modified single threaded request handler has been introduced. Test results from both request handlers will be presented in the next chapter.

The abstract ROBIN software interface *Fragment Manager*, used by the ROS software to communicate with the ROBIN hardware, has been implemented to support the MPRACE ROBIN. It sends messages to the hardware and enables the receiving of replies (event data fragments in most cases). For the reply channel, from the ROBIN hardware to the host-PC software, two DMA communication mechanisms have been implemented and tested. One of them is a port from the previous microEnable ROBIN prototype, the second has been developed for the MPRACE ROBIN approach. Test results for both schemes will also be presented in the next chapter.

Chapter 5

Results and Analysis

To get information about the performance and usability of the MPRACE ROBIN hardware, the PC system, and the software component, a set of tests and measurements has been done. These are presented and discuses within this chapter.

One major point is the test of the ROBIN functionality. This comprises the evaluation of the PCI bus message passing, influence of input via HOLA SLink, and the efficiency of the ROS software request handler. Furthermore the performance of a complete ROS PC with event data input and Gigabit Ethernet output is presented and discussed. This will show that the proposed ROBIN hardware together with the PC test system can fulfil the ATLAS performance requirements.

5.1 The Test Setup

For all measurements, presented in this chapter, one test setup has been used. Figure 5.1 shows this setup. It comprises four PC's:

- One contains four MPRACE cards and two Gigabit Ethernet adapters. It executes the ROS software described in the previous chapter and represents one ATLAS ROS module.
- Another acts as a HOLA SLink source. Therefore a SLink source card, called SOLAR [IvdB02], has been plugged into one PCI slot. It sends event data fragments via one HOLA SLink with programmable size and frequency.
- Two requestor PCs are connected via one Gigabit Ethernet switch to the network adapters of the PC. These send event data requests or delete messages to the ROS PC. One emulates the level 2 farm and the other the SFI event builder farm.

The ROS PC system has already been described in section 4.1. It is a dual CPU system with two 2.4 GHz Xeon processors and four PCI buses. This PC can be equipped with up to four MPRACE ROBIN boards and two Gigabit Ethernet network adapters. One of the MPRACE board carries the HOLA SLink mezzanine described in 3.1. This allows tests with real SLink input. In all other cases the internal event data generator has been used (see section 3.2.4).

The PC with the SOLAR SLink source is a 866 MHz Pentium III. Furthermore two different requestor PCs are used: a 2.4 GHz Pentium 4 and a 2.66 GHz Xeon system. Each has a

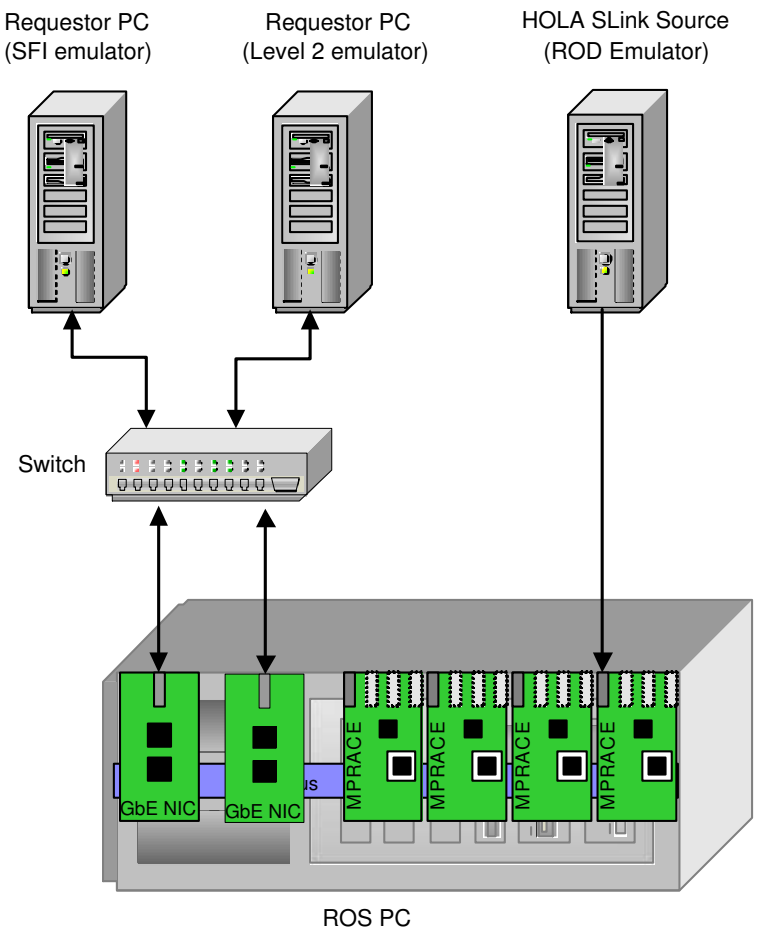

Figure 5.1: Test setup for performance measurements.

Gigabit Ethernet interface already on board. Finally the Gigabit Ethernet switch is an Allied Telesyn AT-9410GB with 10 ports.

Not all components of this setup are activated for each measurement. Many tests, presented in the next section, are performed without Gigabit Ethernet I/O. Thus the two requestor PCs do not operate in this case. The HOLA SLink source PC is also not used and therefore deactivated in many cases.

5.2 System Test Results

The previously described test setup has been used to run a number of measurements. These are presented in this section. The major focus of the first test series is the MPRACE ROBIN performance inside the ROS PC. Initial results of the event data request performance archived with a simple test application are presented. Subsequent tests add more complexity. First events are deleted in addition. SLink input is added next.

The next step changes the focus from the ROBIN hardware investigation to the software efficiency. The main ROS application and the efficiency of the request handler algorithms (see 4.2.3) are investigated. Finally a complete ROS PC with up to four MPRACE ROBINs and 16 readout links is tested with and without Gigabit Ethernet I/O.

5.2.1 MPRACE Request Performance

The aim of the first measurement is to determine the efficiency of event data requests to the MPRACE ROBIN. Therefore a simple test application sends permanently a four word event data request message via programmed I/O to the MPRACE and checks for the reply. After a certain amount of time the MPRACE ROBIN answers each request with the desired event data fragment. This time specifies the MPRACE event data request latency and is the value to be measured. Only the *FragmentManager* interface with the methods *requestFragment* and *getFragment* (see 4.2.3) are used by the test application to get event data fragments from one readout link (ROL).

In total 100 000 event requests have been executed and the total duration has been determined by the test application. Therefore a high resolution timer, based on the CPU clock pulse, measured the time between the first and the last requested event. This allows the calculation of the request latency per event. The procedure has been repeated 20 time to get statistical information about the measured value.

No input load has been generated on the SLink port during the measurement and no delete messages have been sent to the ROBIN. Instead event data was pre-loaded into the MPRACE event buffer memory.

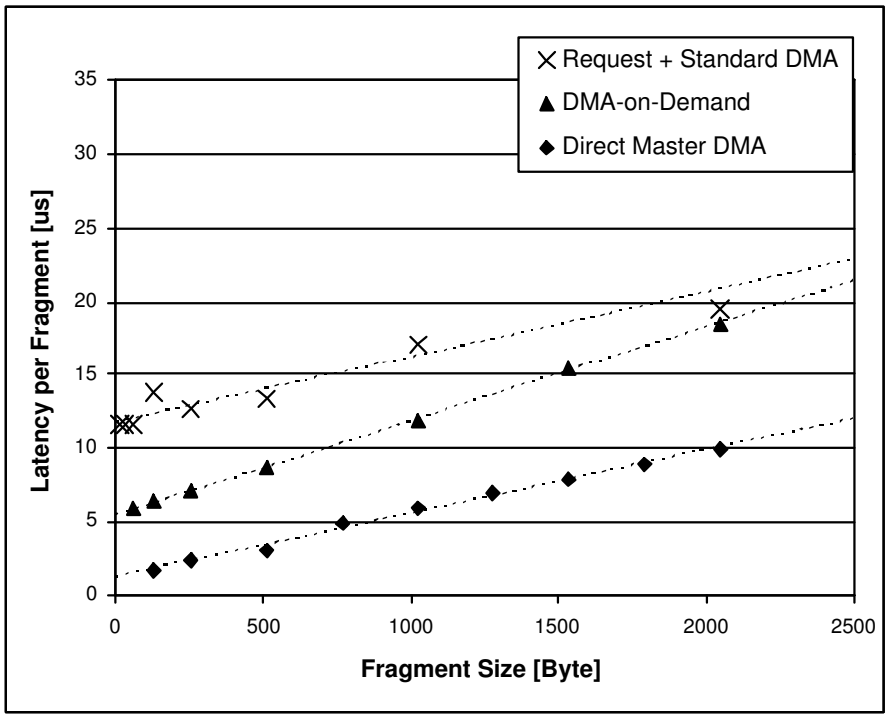

Figure 5.2: MPRACE ROBIN event data request latency for three different DMA mechanisms. Errors are in the order of 1% and therefore not shown.

Figure 5.2 shows the resulting request latencies for one MPRACE ROBIN depending on the event fragment size for three different communication variants. Two of them have been acquired with two different mechanisms to transport the ROBIN reply messages containing event data fragments. These variants, DMA-on-demand and direct master have already been described in section 4.2.4.

The third curve in Figure 5.2 shows a standard DMA for comparison. For this measurement another test application programs the PLX 9656 to read a block of data with a size between 16 and 2048 Bytes and records the time for this operation. No event data request is sent to the MPRACE ROBIN.

The tests result in Figure 5.2 shows that the direct master DMA method is the fastest. It has the lowest event data fragment request latency. The time between sending the request message and having the data in the DMA buffer is linear increasing with the fragment size. For small fragment sizes (128 Bytes) a latency of $1.7 \pm 0.02\,\mu$s can be observed. For large event fragments (2048 Bytes) this ends up to $10 \pm 0.1\,\mu$s.

Using the DMA-on-demand method increases the request latency to $5.8 \pm 0.05\,\mu$s in average for 64 Byte data fragments and $18.51 \pm 0.2\,\mu$s in average for 2048 Byte data fragments. This reflects mostly the time for the DMA buffer clean process and the re-initiation of the DMA which happens frequently after a number of transferred event fragments when the PLX 9656 DMA operation expires. The time for both operations is spread over the number of event data fragments, transferred into the DMA buffer, and appears as additional overhead.

Finally the standard DMA offers the largest latency for getting a data packet. This is the result of the large overhead required for a single DMA initiation. Thus even small data packets of 64 Bytes require $11.7 \pm 0.1\,\mu$s of transfer time.

A linear fit of the measurement results in Figure 5.2, allows an estimation of the event data request overhead and the bandwidth of the event data DMA transfer. The latter is the inverse gradient of the fitted line.

	Request Overhead [μs]	Used Bandwidth [MByte/s]
Direct master DMA	1.27 ± 0.01	232.6 ± 2
DMA-on-demand	5.45 ± 0.05	156.4 ± 2
Standard DMA	11.7 ± 0.1	223 ± 2

Table 5.1: Request overhead and DMA bandwidth for event data requests form the MPRACE ROBIN.

The resulting values are summarised in Table 5.1. The lowest request overhead is achieved with the direct master DMA message transfer method. At least $1.27\,\mu$s is required to transmit an event data package even if it would have a size of zero. All durations which do not depend on the fragment size are combined in this value: the time to load zero to the first word of the DMA target memory, the time to sent the four request words to the MPRACE hardware, and the processing time inside the MPRACE ROBIN. This value increases for the DMA-on-demand method and the standard DMA due to the additional overhead for memory preparation and DMA initiation.

Regarding the bandwidth values in table 5.1 shows that event data can be transferred with 232.6 ± 2 MByte/s into the PCs memory. This is valid for the direct master DMA and also nearly matches the standard DMA. The determined bandwidth almost matches the MPRACE local-bus limit of 256 MByte/s.

Only the DMA-on-demand method has a reduced bandwidth of 156.4 MByte/s. This can be explained with analysing the recurrent overhead for DMA initiation and memory preparation. Both appear more often in case of large event data fragments and are spread over all transferred data packets. The result is a fragment size dependent overhead which is expressed in the bandwidth reduction.

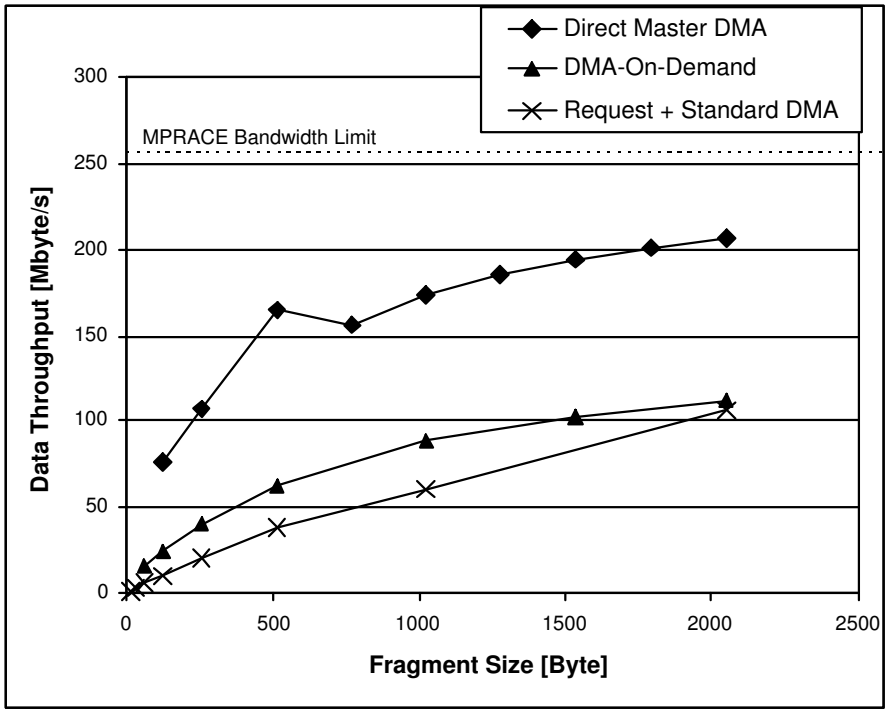

Figure 5.3: Throughput of MPRACE ROBIN event data request. Three different DMA mechanisms have been used to transfer data. Errors are in the order of 1% and therefore not shown.

The average throughput of event data requests from the MPRACE ROBIN is shown in Figure 5.3. Contrary to the above mentioned DMA bandwidth, this throughput includes the duration for sending request message and the MPRACE latency. It has been calculated from the values in Figure 5.2 for all three transfer schemes. This figure shows again the advantages of the direct master transfer mechanism for small fragments between a few words and 2 kByte. Almost twice the throughput of the other two mechanisms can be reached.

Since the direct master DMA has shown up to be the best method for transferring reply messages from the MPRACE ROBIN board to the Host PC it will be used in all future measurements.

5.2.2 Influence of Delete Messages

If event data is not required any more, the ROS and thus the ROBIN are instructed to remove it from the ROBIN buffer. This is carried out by sending a delete messages to the MPRACE ROBIN which instructs the hardware to remove the event with the submitted level 1 ID from the event data buffer (see 3.2.6).

To quantify the influence of these delete messages to the MPRACE ROBIN event data request performance, the application of the previous test has been extended. Each requested level 1 ID is deleted in a later message. To reduce the number of delete messages, 100 IDs are grouped together to form a 412 Byte message.

Since event data is pre-loaded into the MPRACE event memory for this test, the delete message can not be finally executed in the ROBIN hardware. Otherwise no more data would be available after a short time. Thus the event delete is emulated. The messages are executed in the ROBIN hardware up to the point where the hash table entry in the ROBIN buffer

management is loaded and the "free" memory page is written into the free-page FIFO (see 3.2.9). Compared to a real delete, the required SRAM write and free-page FIFO push operation are executed without an asserted write-enable signal. Thus the delete emulation consumes identical time but performs no action.

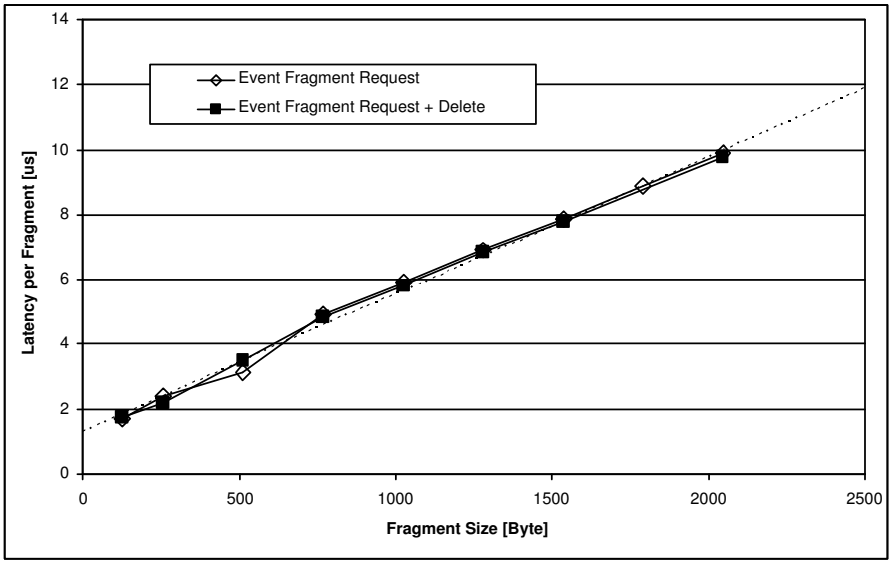

Figure 5.4: MPRACE ROBIN event data request latency with and without sending of event delete messages. Errors are in the order of 1% and therefore not shown.

Figure 5.4 shows the resulting request latency. The previous result with direct master DMA from Figure 5.2 is added for comparison. No major difference and penalty is created by the addition of delete messages. The delete messages do not disturb the event data transmissions via PCI bus. The delete processing is executed inside the ROBIN hardware in parallel to other tasks. A linear fit to the curve with including delete messages shows an overhead of $1.3 \pm 0.02\,\mu$s. This corresponds to the result without delete messages when considering the 1% measurement error. The event data fragment transfer between the MPRACE ROBIN and the PC memory is as fast as without deletes.

5.2.3 Influence of SLink Input

Adding continuous SLink input to the MPRACE ROBIN completes the picture of the MPRACE performance. An external SLink data source, the previously mentioned SOLAR board (see section 5.1), sends event data packets to the ROBIN input via a 2.5 GBit/s fibre. This data is processed as described in section 3.2.5, and stored in the MPRACE SRAM buffer memory. Each incoming event gets requested and deleted by a PCI bus message. Therefore the test application from the previous measurement has been used again.

The achieved event data request latency on the PCI bus interface is shown in Figure 5.5. Three curves are plotted: the request latency with continuous input from SLink to the MPRACE ROBIN, the request latency with continuous input from the internal data generator (see 3.2.4), and the request latency without input from the previous paragraph for comparison.

It can be seen that input from SLink and from the internal data generator is in most cases equal, except for small fragment sizes. Here the results with real HOLA SLink show an unexpected high value. This can not be observed with the data generator.

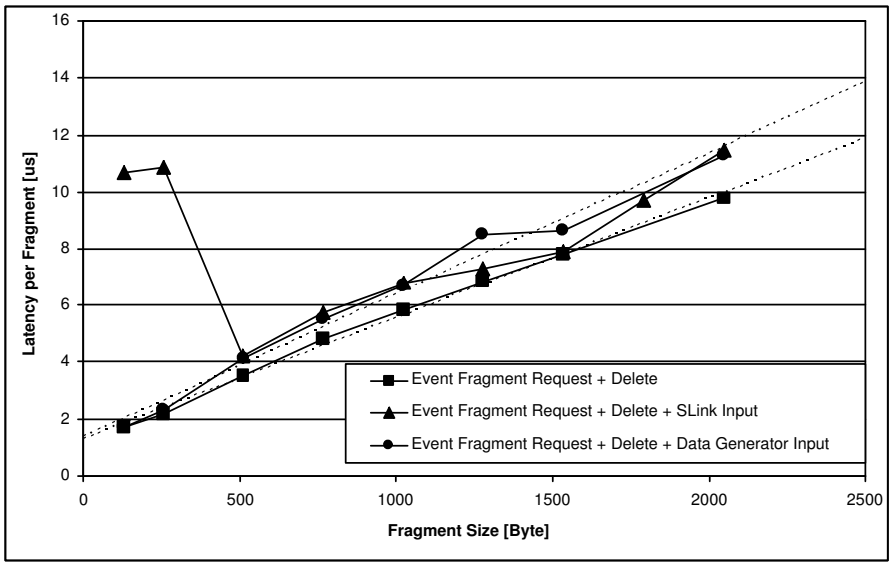

Figure 5.5: MPRACE ROBIN Fragment Manager performance with continuous SLink input.

Since event data, coming from the internal data generator, is processed reasonably, the problem might by inside the SLink HOLA core, the MPRACE SLink receiver hardware, or the SOLAR SLink source (see 3.2.4). If some effect in these components decreases the input rate, the PCI bus request rate must also decrease. But this needs further investigation. Using a different HOLA SLink source hardware could exclude an odd behaviour of this component. Furthermore a more detailed debugging of the HOLA SLink core, provided by CERN, or the receiver hardware could give a reasonable explanation.

Comparing all three results with event data input, again no major difference can be observed. This shows that input load to the SLink port of the MPRACE ROBIN does not have a major influence to the request performance on the PCI bus. The internal resources (event buffer RAM, event buffer management) can be efficiently shared between the input channel and the output to PCI.

A linear fit to all three results shows again the bandwidth used for the event data transfer on the PCI bus. With SLink or data generator input a reduced value of 199 MByte/s can be observed. Without input 233 MByte/s has been archived (see 5.2.2).

This reduction is caused by the access to the event data buffer memory on the MPRACE ROBIN. The bandwidth between the FPGA and the event data RAM is limited to 384 MByte/s and has to be shared between input from SLink and output to PCI. Due to the fair arbitration between both channels and since all event data arriving to the ROBIN is also requested through PCI, the bandwidth is getting halved. Input can be done with up to 192 MByte/s and output to PCI is limited to 192 MByte/s which nearly matches the measurement result.

5.2.4 Multi-Threaded Versus Single-Threaded ROS Software Design

The last sections have presented the native request and delete performance of the MPRACE ROBIN by using the *FragmentManager* software interface (see 4.2.3). This interface is normally used by the request handler threads within the ROS application to access the ROBIN hardware (see 4.2). This ROS application processes input requests from the trigger farms via

Gigabit Ethernet, distributes these to the MPRACE hardware, collects all required event data fragments, and returns them to the requestor over Gigabit Ethernet.

A number of request handler threads within this ROS application are responsible for the MPRACE ROBIN access and the event data collection. Their efficiency is crucial for the performance of the whole ROS. Two variants have been developed and presented in section 4.2.3. It is the aim of this paragraph to test and compare both.

All measurements have again been performed with the setup presented in 5.1. Input to the MPRACE has been generated by the internal data generator during the test. On the output side the ROS application has controlled and processed event data requests or delete requests. These can be either received form a Gigabit Ethernet interface or generated by an internal request-generator algorithm. The first option requires external requestor PCs and complicates the measurement procedure. The second option allows measurements in a "standalone" mode where the Gigabit Ethernet interaction gets emulated with the above mentioned ROS internal software component. To compare the request handler thread algorithms, network I/O is not essential. Thus the second option has been chosen for the measurements in this paragraph.

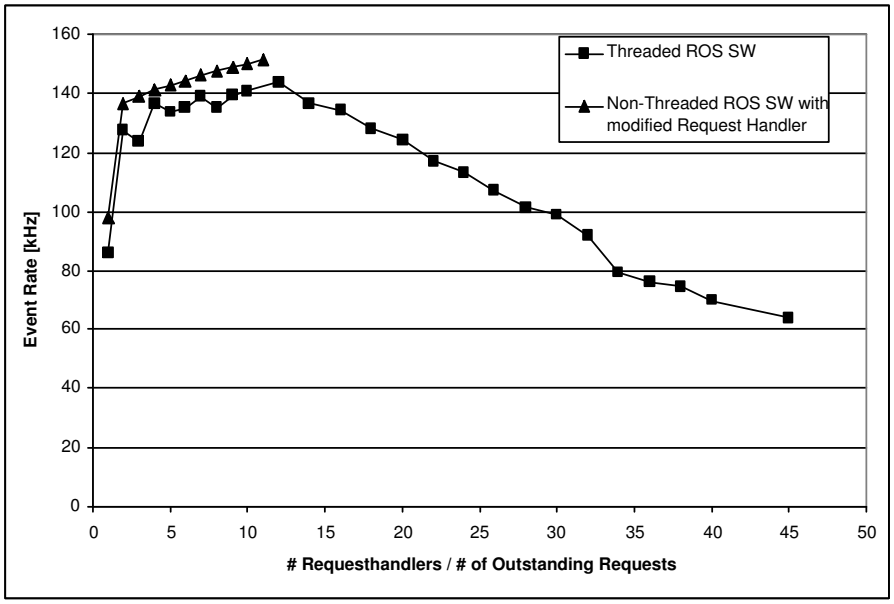

Figure 5.6: The ROBIN PCI request rate depending on the number of request handler threads for the CERN request handler or depending on the number of outstanding requests for the modified non-threaded request handler. Fragment size is 1024 Bytes.

For each event arriving at the ROBIN's input the ROS software request-generator triggers an event data and delete request. This is processed by the TriggerIn thread and later by the request handler thread. The latter accesses the ROBIN hardware and collects event data or sends a delete message. Finally the DataOut thread simulates the output of the collected event data to the Gigabit Ethernet network. The ROS software counts all events, processed during the test, and reports the event rate.

This result is shown in Figure 5.6. Both types of request handler algorithms have been tested and are plot in Figure 5.6. The CERN version of this algorithm is based on multiple threads each handling only one request at the same time. Multiple threads are used to optimise and re-use MPRACE hardware I/O delays. The test varied the number of threads and measured

the number of events processed by the ROS system. The result is shown in the curve with the square measurement points. It rises for small thread counts, since the MPRACE I/O delays are better utilized with each additional thread. This reaches a maximum with 14 threads. More threads add overhead due to the more frequent task switches. The event rate decreases again.

The alternative request handler algorithm is executed in only one thread. It tries to optimise the use of the MPRACE hardware request queue by maximizing the number of outstanding requests. This maximum number of outstanding requests is also the parameter varied during the test. It determines the highest filling state of the MPRACE request queue. This filling state is tried to be sustained by the algorithm.

The test shows that the alternative one-thread request handler algorithm increases the achievable event rate by 5 to 10% for a fragment size of 1024 Bytes compared to the original CERN implementation. The system benefits from the lower number of task switches and I/O delays can be used more efficiently for the ROS software tasks.

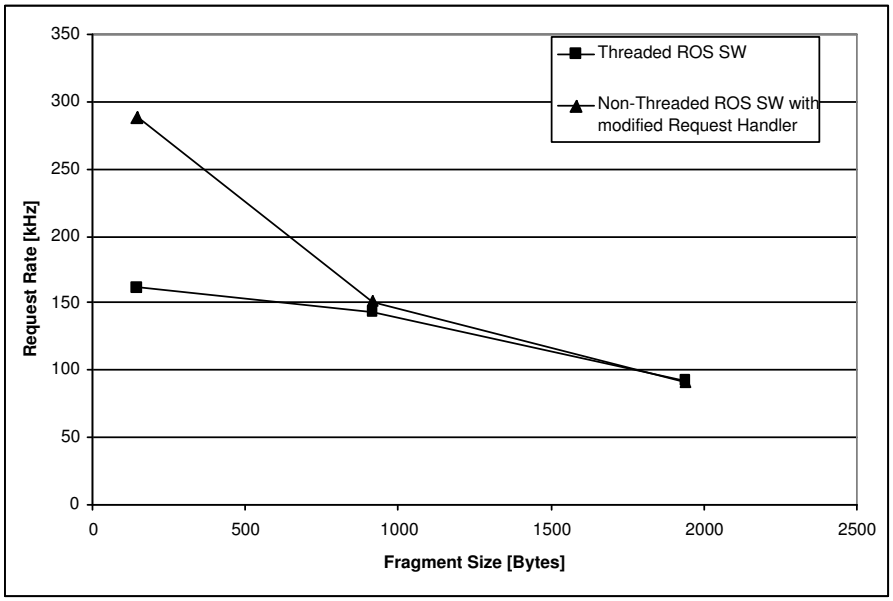

Figure 5.7: Size dependency of the ROBIN PCI request for the two request handler options.

The fragment size dependency of the event rate is shown in Figure 5.7. The benefit of the alternative algorithm increases for small event data fragments (128 Byte). For large fragments (2048 Byte) no advantage can be observed any more.

Figure 5.8 illustrates the reason for this behaviour. On the left side the event processing of the original CERN algorithm is schematically displayed. On the right side the alternative approach is shown.

The first operation of both algorithms is the sending of the request message. After this the polling mechanism starts. Since the fragment is not immediately present in the PCs DMA memory, a task switch is forced in the multithreaded request handler's polling loop. Depending on the Linux task scheduler, the polling is continued after an arbitrary amount of time.

Contrary to this, the alternative request handler checks returning event data not immediately after the request. First a number of other messages are sent and previously requested data is processed. After this the polling of the already requested event data starts.

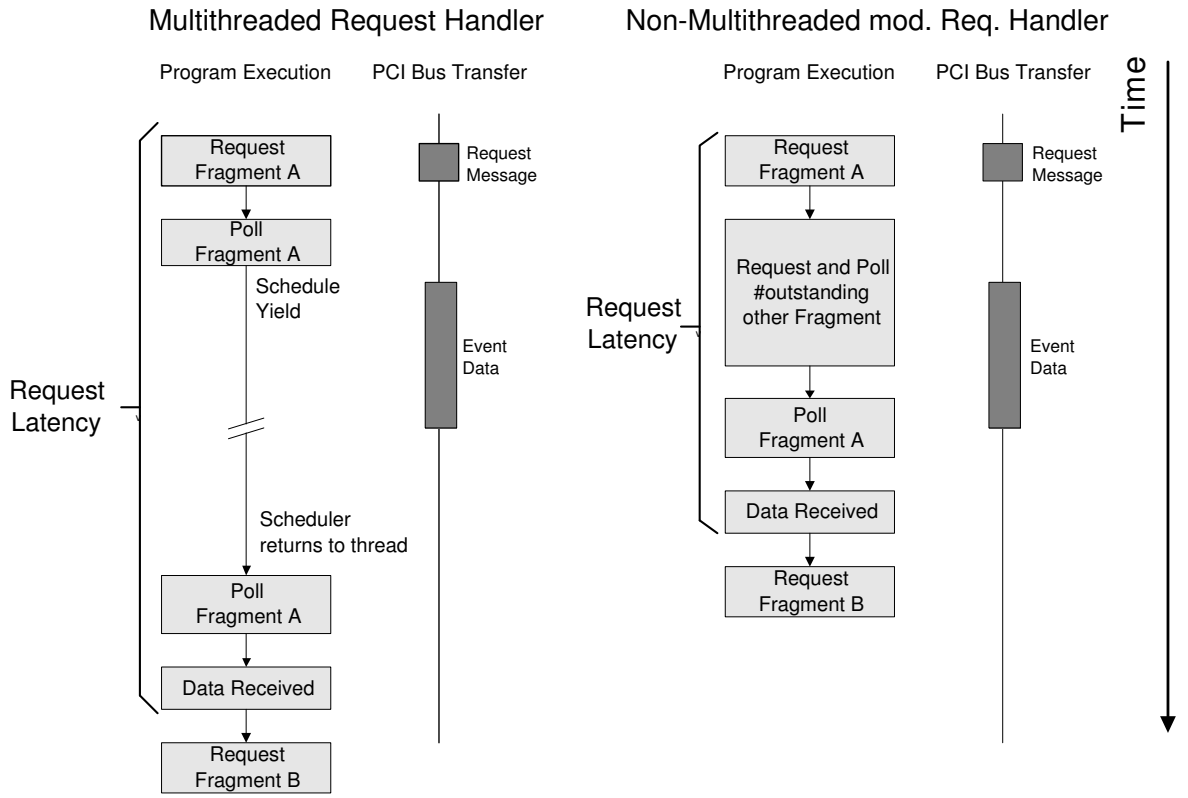

Figure 5.8: Execution difference between original and modified ROS Software request handler.

In case of small fragment sizes and therefore short transfer durations, the alternative request handler algorithm is able to detect a transmitted event data packet earlier than the original approach. The latter needs more time to return to the polling position inside the code since another thread may use the CPU for a comparatively long period. In case of large fragments, which have long transfer durations, this difference decreases to zero. Both algorithms have to wait for the data arrival.

Due to this observation, all further measurement results have been produced with this alternative non-multithreaded request handler. It is at least equal, in many cases better than the original CERN implementation.

5.2.5 Performance of a ROS PC with Multiple MPRACE ROBINs

In the ATLAS ROS baseline architecture, presented in section 2.3.4, it is intended to use a ROS PC system which is able to receive event data on at least 12 readout links. This requires the usage of a number of MPRACE boards. Up to now all measurements have been done with only one board and only one of the four ROLs activated. Measurements with up to four boards and 16 ROLs will be presented in this paragraph.

Again the ROS software and the setup described in section 5.1 have been used for the measurements. Input data to the MPRACE ROBINs has been generated with the internal event data generator and a rate of 130 kHz. The fragment size was set to 916 Bytes resulting in a total size of 1024 bytes when requested via PCI (generated event fragment + ROB header). The ROS software internal request generation and also the DataOut thread emulated network

I/O during the tests. Furthermore the alternative single-threaded request handler has been used and run with up to 12 outstanding requests.

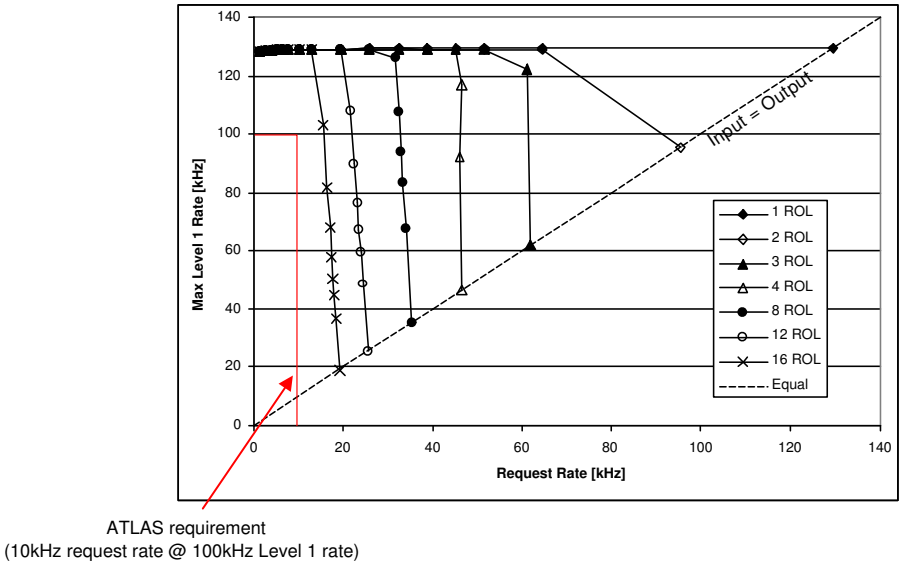

Figure 5.9: MPRACE ROBIN performance without DC I/O.

Figure 5.9 shows the measurement results. It plots the maximum sustainable level 1 accept rate (which is the ROBIN's input rate) depending on the event data request rate on PCI for 1, 4, 8, 12, and 16 readout links on up to four MPRACE ROBINs. Three boards have been plugged into PCI slots on separate buses. The fourth ROBIN has been placed on the same bus as the third.

Each data point of a curve represents a different fraction between requested data and incoming data. All points on the line "Input=Output" correspond to a fraction of 100%. This means that all incoming data is also requested and deleted by the ROS PC. This fraction decreases with each further data point on a curve. In the end only 1% of all data is requested, all other data is only deleted.

With the decreasing of the fraction between incoming and requested events also the request rate (plotted on the x-coordinates) decreases. This is produced by the request-generator of the ROS software. It generates, with a fix frequency, delete and event data requests. Thus, for smaller fractions also the request rate decreases. The maximum sustainable level 1 rate, plotted on the y-axis in Figure 5.9, saturates for small fractions and thus for lower request rates at the level of the ROBIN event data generator frequency.

When using only one ROL, the performance is sufficient to request and delete all incoming event data fragments at the selected input frequency of 130 kHz. With two ROLs (both are on the same ROBIN board) the possible level 1 accept rate decreases for high PCI request rates down to 95 kHz. In this case the data from two ROLS have to pass the MPRACE PCI interface which decreases the maximum output performance. This continuous with an increasing number of ROLs. For all measurement curves the maximum request rate can never exceed a value which is determined by the latency of the MPRACE ROBIN and the performance of the ROS software. Comparing the results in Figure 5.9 with the ATLAS requirements in 2.1 shows that even with 16 ROLs handled by the ROS PC these can be met.

Calculating the PCI-Bus throughput for the maximum request rates depending on the num-

ber of ROLs gives information about the PCI bus utilization with one and more MPRACE boards. This is shown in Figure 5.10.

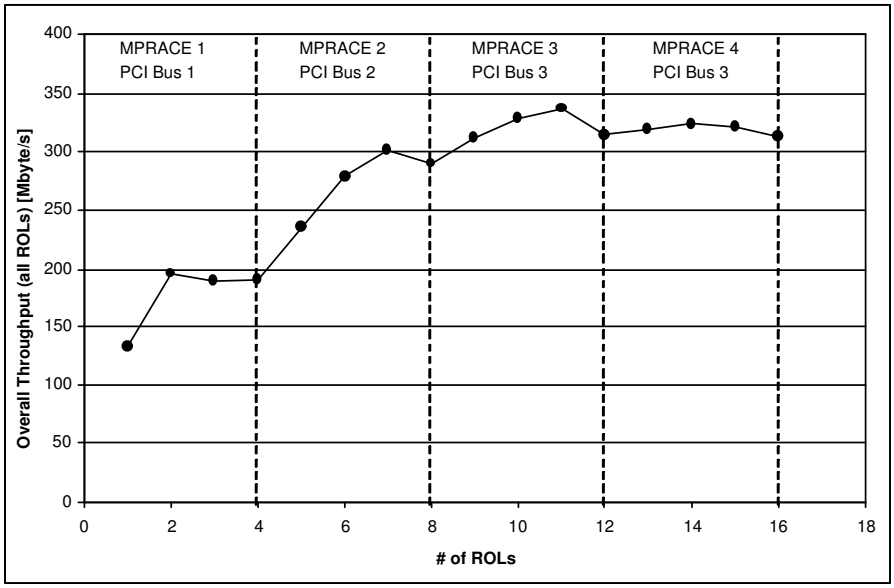

Figure 5.10: MPRACE ROBIN PCI bus throughput without Gigabit Ethernet network I/O.

The throughput with only one activated ROL can be compared with the measurements in Figure 5.2.1. The value in the previous measurement is about 40 MByte/s larger than the present value. Since Figure 5.4 has shown that the additional delete requests have no influence on the event data request performance, the throughput decrease must be a result of the additional input load and the more complex ROS software.

With two ROLs on one MPRACE ROBIN the PCI throughput increases to 200 MByte/s. This shows that the request handler can use the delay in which the request of the first ROL is processed inside the MPRACE hardware to send the second request message.

But for three and four ROLs the bandwidth decreases again by 5-10 MByte/s. This could be explained by arbitration effects on the MPRACE local-bus (see 3.1). Request messages are written by the PLX 9656 to the MPRACE local-bus. This has high priority and the local-bus arbiter breaks any ongoing access of the FPGA. Any re-arbitration adds a few clock cycles before and after the breakdown. When the maximum throughput of MPRACE has been already reached, this effect may decrease it by a small value for any additional ROL.

Adding a second MPRACE ROBIN board on another PCI bus increases the useable bandwidth by a factor of two. Thus it is expected that an activation of one ROL on this second board immediately raises the throughput by 135 Mbyte/s (the throughput when requesting data from a single ROL). This can not be observed. Instead the volume of requested data increases only by 40 Mbyte/s. This effect is a result of the request handler algorithm. It has to access five ROLs instead of one which means that a single one is accessed more rarely.

This continues with two and three ROLs on the second board. With each ROL an increased throughput has been observed. The maximum of 300 MByte/s is reached with three ROLs on the second board. Again with four links the rate somewhat decreases due to the above mentioned arbitration effect.

Adding another board again increases the bandwidth and the third board can transfer data

in parallel to the first two boards. Again the throughput raises, saturates and decreases a bit.

Finally the fourth board is plugged into the same bus as the third. This does not increase the bandwidth and also the throughput do not increase any more.

5.2.6 Influence of Network I/O

Finally the ROS system has been tested with one and two activated Gigabit Ethernet network interfaces for I/O to level 2 and SFI event builder farm. The setup has been already described in section 5.1. It consisted of the following components:

- A ROS PC with up to four MPRACE ROBINs handling up to 16 ROLs.
- Two PCs for request generation. One is emulating the level 2 trigger farm and one the SFI event builder farm.
- A Gigabit Ethernet switch in between.

Event data input to the ROBINs have been again supplied by the internal data generators which run at 130 kHz and produce event data fragments with a size of 916 kByte.

During the tests, the requestor PCs have asked each event, previously arrived at the MPRACE ROBINs inside the ROS PC, and generated a request to delete it later. These requests have been sent over Gigabit Ethernet using the UDP/IP protocol and a message passing library developed at CERN [Hau03].

The ROS system queues these requests and sends them to the MPRACE ROBIN hardware. Again each data request is forwarded to all ROBINs and all ROLs inside the PC. The ROS system finally combines all event data fragments from the different ROBINs to a large ROS fragment and sends this via the network interface to the requestor PC. Each fragment, coming from the MPRACE ROBINs, has a size of 1024 Bytes during the tests presented in this paragraph. This is a representative for the largest event data fragments arriving at a readout link (see table 2.1).

To request and delete each incoming event data fragment on each ROL is an overestimation compared to the real operation conditions because in case of a level 2 request only event data from specific ROLs within the pre-defined region-of-interest (see section 1.3) are required.

The achieved request rates per readout link (ROL) are plotted in Figure 5.11. The lower curve shows the results with one Gigabit Ethernet network adapter and the upper curve with two. Comparing the result with Figure 5.9 each point corresponds to the values on the “Input=Output” line.

This comparison shows that the activated network I/O decrease the request rates per ROL by a factor of 2.5 to 3. But the ROS system presented in this thesis can still sustain the worst case event data request rate of 10 kHz (see 2.1) with up to 12 readout links distributed on three MPRACE boards.

Calculating the throughput dependent on the number of readout links leads to Figure 5.12. The presented values include the data volume required for the Ethernet frame header (20 Byte), IP header (24 Byte), UDP header (8 Byte), and the header added from the CERN data collection protocol suite [Hau03]. With one activated Gigabit Ethernet link up to 100 MByte/s can be achieved, with two 125 MByte/s. This is the data volume flowing over the Gigabit Ethernet interface(s).

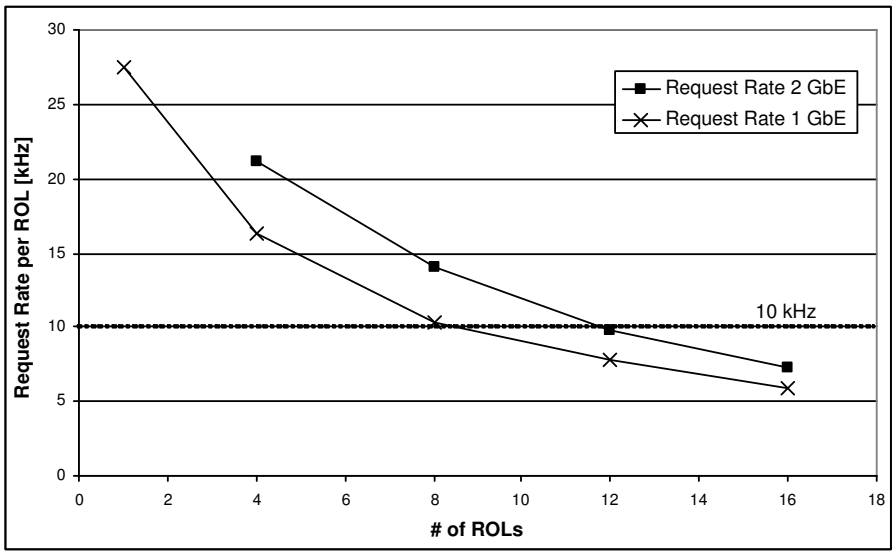

Figure 5.11: ROS system performance with MPRACE ROBIN and network I/O to level 2 and SFI farm emulator.

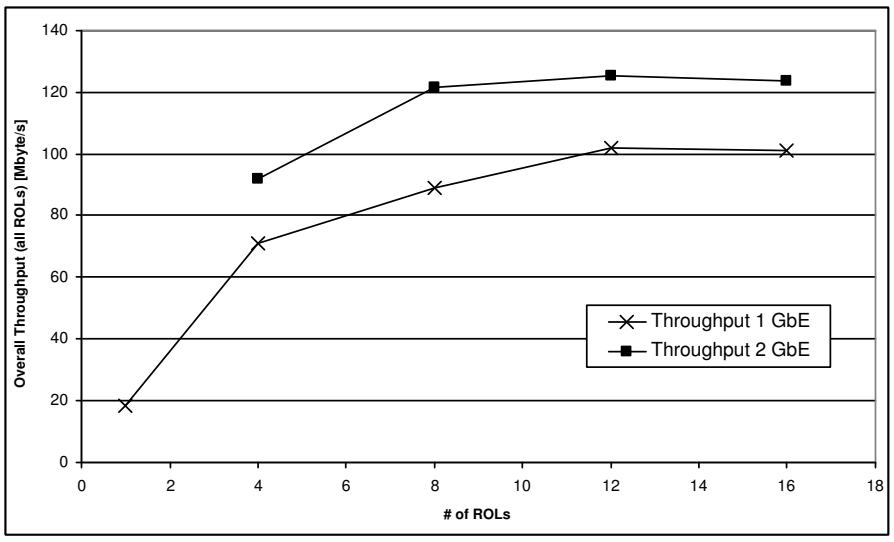

Figure 5.12: ROS system throughput with MPRACE ROBIN and network I/O to level 2 and SFI farm emulator.

This result shows that only a small part of the increasing bandwidth with two Gigabit Ethernet adapters can be used by the ROS to improve throughput and request rate. Watching the process CPU utilization during the measurements with the application "top" shows that approximately 65% of the system processor time is used by the Linux operating system. The remaining CPU time, only 35%, is utilized by the ROS application processes. Out of these indications, the 2-3 times larger request rates without network I/O and the high system utilisation with network I/O, it can be assumed that the ROS, presented in this thesis, is limited by the operating system. Linux is not able to exploit two Gigabit Ethernet interfaces while also handling the MPRACE ROBIN boards.

The reason for this limitation of Linux OS in case of heavy network traffic can be found in the Linux Kernel [Lin]. Each Ethernet frame, a block of 1538 Bytes of data, is constructed

within the kernel by copying data from the user's buffer to a kernel buffer, adding the protocol headers, and passing this to the network hardware driver. This forces all event data to be copied between two memory buffers. Since the event data is already present inside a DMA capable memory, this copy procedure is unnecessary in this case.

This matter of fact also explains the factor of three in the comparison between the measurement results with and without network I/O. With network interaction all data has to be move three times: from the ROBIN hardware to the PC memory, within the PC memory from one buffer to another, and finally to the network adapter.

5.3 Conclusions

The measurements have shown that the MPRACE PCI ROBIN delivers a good performance. It is able to sustain 160 MByte/s input on four readout links, while handling event data requests with 200 MByte/s on PCI.

A multiple PCI bus ROS PC, equipped with MPRACE ROBIN boards, running an improved CERN ROS software, is able to fulfil ATLAS worst case requirements: a request rate of 10 kHz with up to 12 readout links distributed on three boards. This requirement has been estimate with assuming a level 1 accept rate of 100 kHz. This assumption includes a safety factor of four which can be carried to the request rate requirement.

Additionally it could be proved that up to 16 readout links on four boards can be successfully handled. Grouping 16 ROLs in one ROS PC does not match the ATLAS worst case requirements any more, but may be connected to lower demand sub-detectors.

Currently the limiting factor of a PC based ROS with MPRACE ROBINs is the handling of the network connection to the level 2 farm and the SFI event builder farm. Even if sufficient network bandwidth is available, the ROS PC is not able to exploit it.

It has been shown that this is not caused by the MPRACE ROBINs. A ROS system with several MPRACE ROBINs and without network I/O reaches three times the performance then an equal ROS with network I/O. Instead, the limitation has been localized within the network handling of the Linux operating system.

Chapter 6

Conclusion and Outlook

6.1 Summary

The goal of this thesis was to evaluate a readout subsystem for the ATLAS detector. It should be a cost-effective solution and mainly based on widespread commercial "of-the-shelf" components with a minimized effort for custom hardware development.

A high performance server PC with four PCI bus segments has been chosen as the target platform. Since this standard PC is not able to process the event data coming from the readout drivers on a reasonable number of links it has been extended with a number of MPRACE FPGA co-processor boards, called ROBIN in this context. The FPGA co-processor, a 64 Bit / 66 MHz PCI card with an FPGA, memory, and expansion connectors for the detector link, performs the task to receive and buffer the event data in the presented approach. It is a simple device which has been used for many other applications before. The PC is responsible for Gigabit Ethernet network I/O to the level 2 and SFI event builder farm. It runs an optimized multithreaded application, which handles the network I/O, assisted by the Linux OS. Furthermore it collects the required event data from the PCI boards with a hardware abstraction interface and performs local event building. Thus, the ROS PC is a first stage of event building in case of a level 2 accept decision. This concept moves load from the SFI farm to the ROS.

The functionality of the level 2 trigger makes this system unique in high energy physics. Only a few experiments (e.g. HERA B) have a level 2 trigger processor which requests all necessary event data from the buffer. This characteristic has also a major influence to the complexity of the ROS and increases its output load.

The measurements in the last chapter show that the proposed system can be used in the ATLAS data acquisition. It has been proven that the FPGA co-processor based PCI bus ROBIN device is able to handle event data from four ATLAS readout links with an input rate of 100 kHz and a bandwidth of 160 MByte/s. Three boards with altogether 12 readout links grouped in one PC could successfully provide 10 kHz event data rate on Gigabit Ethernet. This meets the worst case requirement estimations for the level 2 and SFI data rates.

The main bottleneck of the current design is the handling of the Gigabit Ethernet network I/O within the Linux OS. Even if sufficient network bandwidth is provided by using two interface adapters, the event rates towards the level 2 and SFI farms could not be improved. The reason has been identified to be the Linux internal processing of the network sockets and the

assembly of the event data Ethernet frames.

Since the evaluated system is still a prototype a number of further issues (advantages and disadvantages) have to be considered and discussed. This is the aim of the first section within this chapter. In the subsequent section the final readout buffer system technology decision will be presented. Finally an outlook to future data acquisition systems in high energy physics experiments finishes this thesis.

6.2 Discussion of the Current Design

6.2.1 The MPRACE ROBIN

General Remarks

The current MPRACE ROBIN design, evaluated within this thesis, has advantages but also disadvantages. The component is simple and cheap. It consists only of an FPGA, four memory banks, the HOLA SLink input, and the PCI bridge. All this can be placed on a standard format PCI card which makes the hardware production much easier and cheaper compared to a 6 or 9 HU multi-layer board. This remains true even when considering that a 6 or 9 HU board can handle more readout links.

Using an FPGA requires a number of tools for firmware development. The typical design language, VHDL, needs special knowledge and is not a widespread programming language. This makes maintenance or a later upgrade more difficult. But C-like programming languages as Handle-C could help an ordinary C programmer to develop FPGA firmware faster. The disadvantage of complex programming is balanced by the speed of an FPGA and the ability to execute several processes in parallel.

ROBIN Event Buffer

The ROBIN's event buffer and its management is important for the performance, usability, and error-proneness of the device. The event buffer must be large enough to keep all incoming event fragments while its level 2 trigger decision is computed (in average 10 ms). The ROBIN can not delay the receiving of incoming event data from the ROD hardware because the latter can only store a few data words. The XOFF signal, which may be used by the ROBIN hardware to stop transmission from the ROD, is only intended to be a flow-control signal and should only be raised for a few clock cycles. Otherwise event data may get lost due to the missing buffer capabilities in the ROD.

The present SRAM event buffer on the MPRACE, with a size of 2 MByte per ROL, is just sufficient. To give the ROBIN more headroom to react to extraordinary situations, a large buffer would be reasonable. Therefore a number of options can be taken into account:

- Equip the MPRACE SDRAM connector with an SDRAM module.
- Expand and enlarge the four SRAM memory.
- Replace the SRAM memory by larger SDRAM components.

The first option enables the FPGA to use one SDRAM module with a 64 Bit data bus at up to 133 MHz. It allows a memory bandwidth of 1064 MByte/s, 266 MByte/s per channel. This

is approx. 70% of the current event buffer bandwidth. Regarding the more complicated access of SDRAM this may not be enough.

Another option would be to add more or larger SRAM components. This would be a simple extension to the current hardware and has the advantage that SRAM access is very simple from the FPGA's point-of-view. But a large amount of SRAM is very expensive.

An alternative would be to replace the SRAM by SDRAM components within an MPRACE hardware upgrade. They offer a cheap implementation of a much larger buffer, e.g. 32 or 64 Mbyte. But contrary SDRAMs need a more complex access control [Sam03a] [Sam03b]. They requires refresh cycles to keep the memory contents and care must be taken when accessing the banks, rows, and columns of the component to gain the optimal bandwidth. This makes a SDRAM less handy then SRAM, but with the great advantage of a larger amount of memory.

For accessing SDRAM from an FPGA commercial logic cores are available [XIP]. Since in most cases sequential addresses are accessed (for reading and writing 1 kByte event data) the SDRAM can be accessed in burst mode in most of the times which offers optimal performance. Having a clock frequency of 96 MHz (as in the current ROBIN implementation) and a SDRAM width of 32 bit, a bandwidth of close to 384 MByte/s, equal to SRAM, could be reached [Kra04].

MPRACE ROBIN Buffer Management

The current management of the MPRACE ROBIN event buffer is simple, but suitable for the ATLAS application. But some situations have not been taken into account yet (see chapter 3):

- Only one buffer page, which is currently 2 kByte, can be used for an event data fragment. Data is cut otherwise.
- SLink transmission error flags are not recorded.
- The length of the event fragment arriving from the SLink is not compared to the event size information inside the ROD header (see 1.4).
- A missing SLink end-of-frame is not detected and may combine two event fragments. This may also exceed one page inside the event buffer. A detection of this problem implies that the event size information of the ROD header is analysed and taken into account.
- An already existing hash table entry is not checked before a new entry is going to be written. If an event stays in the buffer for a long time it may get overwritten and lost.
- The real level 1 ID (the one contained in the ROD header) is not checked when requesting an event data fragment. It is only identified by the hash key. Thus the MPRACE ROBIN may return wrong event data.
- The MPRACE Delete Handler identifies an event data fragment only by its hash key too. Thus it may delete the wrong fragment from the hash table.
- The MPRACE Delete Handler does not notify the user when the deletion of an event has failed. This would be important for operational monitoring.

One of the most complicated issues may be the handling of event data fragments with a size larger then the current page size of 2 kByte. This problem should not appear during normal operation because the event data fragments per ROL are far below 2 kByte (see 2.1) for all sub-detectors. But in case of an unexpected behaviour of the ATLAS DAQ System, or to support the possibility of future upgrades this should be considered.

An optimal solution would be a linked list with an infinite number of pages per level 1 ID. This implementation is very flexible but it would require a complex FPGA implementation or an additional microcontroller. Another option is to increase the page size to 4 or 8 kByte. This also increases the fraction of unused memory because many event data fragments need only a small part of a memory page. Furthermore this only moves the fragment size limit to a larger value.

A good compromise is to use a limited number of pages for one level 1 ID. This solution is similar to the enlargement of the page size, but omits the memory dissipation. Depending on the maximum number of pages per event the level 1 ID hash table increases by the same factor. This solution may be the best for a FPGA based buffer management as it is done with MPRACE. The first approach is more flexible, but better done in a CPU.

Most of the other deficiencies are simple improvements by adding additional checks (e.g. comparing lengths or event IDs), setting of error flags and counting of error concurrencies. The SLink error flags may be added to the buffer and inserted into the ROB header on request. This notifies the next data acquisition instances about the SLink transmission problem.

The arriving of corrupted event data fragments via SLink can be handled similar. There may be fragments which are too long, too short, or at which the SLink start or end-of-frame identifiers have been missing. These have to be buffered and marked by an error flag.

If a new event data fragment attempts to overwrite an existing one another complicated situation rises. This situation appears only if the level 2 trigger needs more then 40,96 ms for the decision on one an event (with an assumed level 1 accept rate of 100 kHz), four times more then the expected average level 2 decision time of 10 ms (see 1.3). But due to the large variation of the level 2 decision time, which may be several times the average, this case may rarely occur. A handling of this exceptional case must be considered.

A possibility would be to copy the present fragment to the PC's memory before it gets overwritten. A mechanism checking the host PC for an old fragment, which could not be found at the ROBIN, could be used at the PC to get the event data fragment later.

Another improvement would be to acknowledge the event delete requests, extended by a list of level 1 IDs which could not be deleted by the MPRACE ROBIN. This helps to early detect and identify problems.

6.2.2 ROS PC and Software

The selection of the PC System (see section 4.1) hosting the MPRACE ROBIN boards has turned out to be a good choice. It is able to carry up to four boards and provides sufficient network bandwidth for the level 2 and SFI connection.

The measurements have shown that the handling of the network interface is currently the main bottleneck of the ROS System with a modern, high performance PC. The Linux OS is not able to move the event data fast enough to the network hardware because of the slow Ethernet frame assembling. But up to 12 readout links distributed on three boards can fulfil ATLAS requirements. More may be used at lower demand sub-detectors.

One possible improvement for this situation would be to use a faster PC. Instead of the 2.4 GHz system, 3.6 GHz PCs are already available. It is expected to have even faster CPUs in a two years timescale when the ROS PCs will have to be purchased. This can make a system with 16 or even 20 readout links possible which is able to fulfil the ATLAS requirement of 10 kHz request rate per ROL.

Another approach would require an extensive change in the network handling of the ROS PC. As already described in 5.2.6 the single Ethernet frames are build within the Linux kernel out of the ROS event data fragments by copying them together with the header into a buffer which will then be used by the network hardware for DMA. The Ethernet interface access and the DMA operation is finally done inside a Linux device driver.

Most Ethernet cards provide the possibility to gather the Ethernet frame data from different memory locations. Since the event data arriving from the MPRACE boards is already in a DMA capable buffer, this could be used to send the data directly to the Ethernet hardware without memory copy. Only the network headers have to be build in a separate memory area.

This requires the modification of the present network driver by adding the possibility to supply Ethernet frames without using the Linux kernel. Since in most cases no information on the hardware access of recent Gigabit Ethernet cards is provided by the manufacture, this will get the main problem during the implementation. Furthermore the network protocol suite (currently UDP/IP) has to be present in the ROS application to form the header. Therefore open source software could be used [Dun].

The measurements in chapter 5 also show that a ROS PC with more then six slots and substantially more then 16 readout links, as it has been considered in (see 4.1) with the usage of an “industrial” PC, is currently not reasonable. This is also valid for a CompactPCI approach with up to 7 boards, each with 4 or 8 readout links (see section 2.3.1). Only the form factor of such a system differs to the PC used within this thesis. The technology (a PC with PCI bus) is equal. Thus the presented measurements have shown that such a system, a CompactPCI PC handling 28 or even 56 readout links, can currently not meet the ATLAS requirements.

6.3 The Final ATLAS Readout System

Upon the evaluations and the development presented in this thesis, the PCI bus based ROS System has been chosen as the baseline architecture of the ATLAS ROS System. Preliminary results have been used in the “ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report” [Gro03] to prove the effectivity of this choice.

Out of the above mentioned disadvantages of the MPRACE ROBIN in terms of error and exception handling, and also because of the small event buffer memory, it has been chosen to develop a new ROBIN PCI card. This development comprises all advantages of the MPRACE ROBIN and takes the various other approaches, presented in 2.3.3, into account.

To add the option of a later upgrade path for bandwidth improvements, this ROBIN also provides a Gigabit Ethernet interface for a direct connection to the level 2 or SFI farm PCs. This is implements the idea of a Gigabit Ethernet based ROS presented in 2.3.1.

Figure 6.1 shows the final ATLAS ROBIN board. Instead of four readout links it will carry only three plus an electrical Gigabit Ethernet interface. The PCI interface and the FPGA are equal to MPRACE. The main event data flow is routed through the FPGA, as it has been done by MPRACE, because of performance issues.

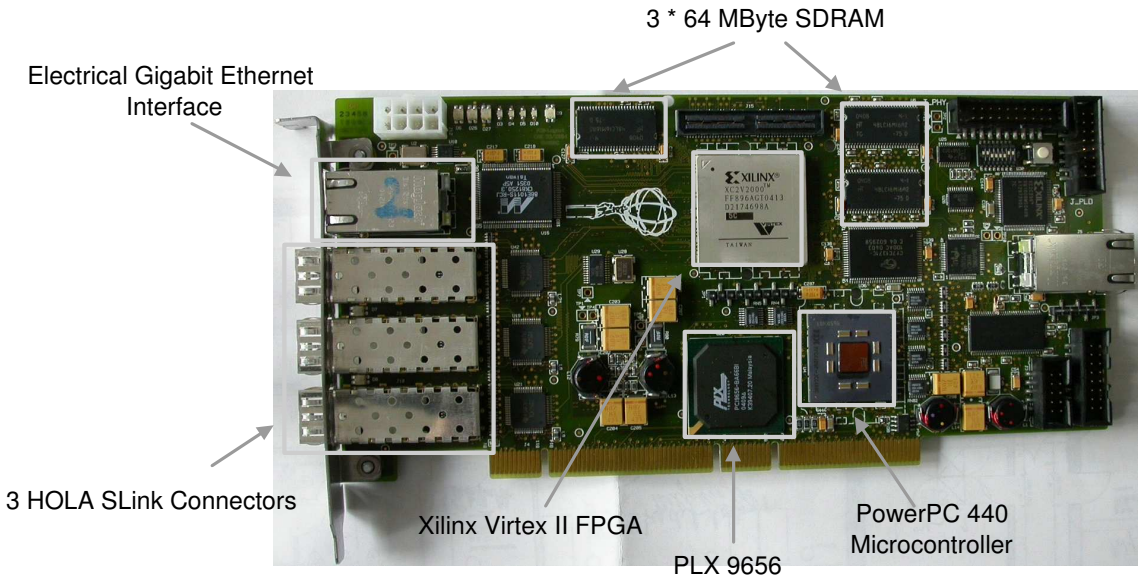

Figure 6.1: The final ATLAS ROBIN board.

For the buffer management, request handling, and error and exception management the FPGA is assisted by a PowerPC 440 microcontroller. Various ideas from the UK-ROBIN approach are re-used in the processor part of the buffer management. The microcontroller also processes the decoding of messages form PCI and network. This increases flexibility and removes all problems mentioned in 6.2.1.

Three banks of SDRAM, each having 64 MByte, are available for event data. This provides huge headroom as it has been discussed above (see 6.2.1).

This ROBIN is used similar to the MPRACE board in the multi PCI bus PC as it has been presented in 4.1. Also the ROS software is re-used. First tests with a prototype show a performance equal to the MPRACE based ROBIN. This development solves most of the problems mentioned in the MPRACE ROBIN discussion above. Only the limitation of the network handling remains because the PC system is still unchanged.

6.4 Future Experiments

The next generation of high energy physics experiments will again increase the bandwidth demands on the data acquisition system. To investigate new physics, the experiments will run at much higher energies as LHC and ATLAS and will be more precise. This increases the amount of data supplied by the next generation detectors.

Furthermore the trend to use as many standard, “of-the-shelf” hardware will continue. Since the computing power of these components evolve very fast, more complex and also more flexible trigger algorithms can be executed on-line during data acquisition. This can even be done on a complete event since data transport technologies are able to transfer data at least 10 times faster then Gigabit Ethernet (e.g. 10 Gigabit Ethernet).

Thus future trigger and data acquisition systems may immediately build the complete event when it has been accepted by a first hardware trigger level and sends the full event data to a large computing farm [Du02]. This puts high demand on the readout buffer system.

To meet these larger requirements new technologies have to be considered. The currently used PCI bus is attempted to be replaced in the next five years. PCI-Express is a popular upcoming candidate. Contrary to PCI, PCI-Express is a 2.5 GBit/s serial point-to-point interface with a switch fabric similar to modern networks. It uses a variable number of channels to adapt to various hardware bandwidth demands. This makes data rates of several Gigabyte per second possible which enables the usage of the presented architecture in future high energy physics experiments with higher bandwidth demands.

Also the network connection bandwidth has to be improved to transport a higher amount of data. The next Ethernet generation with 10 GBit/s per interface is already available. Also InfiniBand is an upcoming connection standard, mainly for PC clusters [Ass04].

The ROBIN and its components also have to fit to increasing bandwidth demands. Memory with more bandwidth are already standard within current PC market. There are double data rate (DDR) RAM modules which carry data at the raising and falling edge of the clock. New FPGA families will provide a decrease structure size and will allow higher clock frequencies.

Furthermore processor cores, directly implemented in silicon, can be used [XIL]. A similar approach for adding a processor to a ROBIN design would be to use logic core which includes it into the FPGA firmware. Thus less and faster components will be available for future ROBIN implementations.

Appendix A

Glossary

ASIC	Application-Specific Integrated Circuit
ATLAS	A Toroidal LHC Apparatus
CERN	Conseil Européen pour la Recherche Nucléaire
CLB	Configurable Logic Block
CPU	Central Processing Unit
DDR RAM	Double Data Rate RAM
DFM	DataFlow Manager
DMA	Direct Memory Access
DREQ	DMA Request
EB	Event Builder
EF	Event Filter
FIFO	First In First Out (Memory)
FPGA	Field-Programmable Gate Array
HOLA	High Speed Optical Link for ATLAS
LHC	Large Hadron Collider
LDC	Link Destination Card (SLink)
LSC	Link Source Card (SLink)
MAC	Media Access Control
MIPS	Million instructions per second
OS	Operating System
PCI	Peripheral Component Interconnect (Local Bus)
PLD	Programmable Logic Device
PMC	PCI Mezzanine Card
RAM	Random Access Memory
ROB	Readout Buffer
ROBIN	Readout Buffer Input
ROL	Readout Link
ROS	Readout Subsystem
SCT	SemiConductor tracker
SDRAM	Synchronous Dynamic Random Access Memory
SFI	Sub-Farm Input
SFO	Sub-Farm Output
SRAM	Static Random Access Memory

TRT Transition Radiation Tracker
VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VME Versa Module Europa (Bus)
ZBT Zero Bus Turnaround

List of Figures

List of Tables

Bibliography

[Alt] *ALTERNATE Online Componter Shop* [online]. Available from World Wide Web: `http://www.alternate.de`.

[Ass04] InfiniBand Trade Association. *InfiniBand Specification* [online]. 2004. Available from World Wide Web: `http://www.infinibandta.org/specs`.

[Ath01] *Athena – User Guide and Tutorial*, August 2001. Available from World Wide Web: `http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/Tech.Doc/Manual/2.0.0-DRAFT/AthenaUserGuide.pdf`.

[Atl] *Atlas Modelling Web Page* [online]. Available from World Wide Web: `http://www.nikhef.nl/pub/experiments/atlas/daq/modelling.html`.

[BBF+02] C. Bee, H.P. Beck, P. Farthouat, D. Francis, M. LeVine, J. Strong, and J. Vermeulen. *ATLAS ROB-on-ROD Recommendations of the TDAQ ROB-on-ROD Taskforce Final Report*. Technical report, 2002. Available from World Wide Web: `https://edms.cern.ch/file/345700/1/ROBonRODfinal.pdf`.

[BBW+01] R. Bock, J. A. Bogaerts, P. Werner, A. Kugel, R. Männer, and M. Müller. *The Active Rob Complex: An SMP-PC and FPGA based solution for the Atlas Readout System*. In Proc. IEEE Realtime Conference, pages 199–203, Valencia, June 2001.

[BFM+04] C. Bee, D. Francis, L. Mapelli, R. McLaren, G. Mornacchi, J. Petersen, and F. Wickens. *The Event Format in the Atlas Data Acquisition*. CERN ATLAS Note ATL-DAQ-98-129, CERN, February 2004. Available from World Wide Web: `https://edms.cern.ch/file/445840/2.4/ATL-D-ES-0019v24.pdf`.

[BJG+00] H. Boterenbrood, P. Jansweijer, G.Kieft, R. Scholte, R. Slopsema, and J. Vermeulen. *A SHARC based ROB Complex : design and measurement results*. CERN ATLAS Note ATL-DAQ-2000-021, CERN, May 2000.

[BMvdB97] O. Boyle, R. McLaren, and E. van der Bij. *The S-LINK Interface Specification*. Technical report, CERN, 1997.

[Bro04] O. Brosch. *A Kaon Trigger for FOPI*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, May 2004.

[CES] *CES - Creative Electronic Systems* [online]. Available from World Wide Web: `http://www.ces.ch`.

[CFJ+00] G. Crone, D. Francis, M. Joos, J. Petersen, and S. Veneziano. *Read-Out Buffer in DAQ/EF prototype -1*. CERN ATLAS Note ATL-DAQ-2000-053, CERN, July 2000.

[CGHM00] D. Calvet, O. Gachelin, M. Huet, and I. Mandjavidze. *A Scheme of Read-Out Organization for the ATLAS High-Level Triggers and DAQ based on ROB Complexes*. CERN ATLAS Note ATL-DAQ-2000-014, CERN, March 2000.

[Col95a] The BaBar Collaboration. *Technical Design Report*. SLAC, March 1995.

[Col95b] The Hera-B Collaboration. *Hera-B – An Experiment to Study CP Violation in the B System Using and Internal Target at the HERA Proton Ring – Design Report*. DESY, January 1995. DESY-PRC 95/1.

[Col96] The CDF Run II Collaboration. *THE CDF II DETECTOR – TECHNICAL DESIGN REPORT*. Fermilab, October 1996. FERMILAB-PUB-96/390-E.

[Col99] The Atlas Collaboration. *ATLAS DETECTOR AND PHYSICS PERFORMANCE Technical Design Report, Volume II*. CERN, May 1999. CERN/LHCC 99-14.

[Col02a] LHCb Collaboration. *Data Acquisition and Experimet Control – Technical Design Report*. CERN, September 2002.

[Col02b] The CDF Run IIb Collaboration. *THE CDF IIb DETECTOR – TECHNICAL DESIGN REPORT*. Fermilab, September 2002.

[Col02c] The CMS Collaboration. *CMS – The TriDAS Project Technical Design Report, Volume 2: Data Acquisition and High-Level Trigger*. CERN, December 2002. Available from World Wide Web: `http://cmsdoc.cern.ch/cms/TDR/DAQ/daq.html`. CERN/LHCC 02-26.

[Col02d] The D0 Collaboration. *DØ Run IIb Upgrade Technical Design Report*. Fermilab, December 2002. FERMILAB-PUB-02/327-E.

[Col04] The ALICE Collaboration. *ALICE – Technical Design Report of the Trigger, Data Acquisition, High-Level Trigger, and Control System*. CERN, January 2004. CERN-LHCC-2003-062.

[Com97] PCIMG PCI Industrial Computers. *CompactPCI Specification*. Technical report, September 1997.

[Con84] S. Conetti, editor. *A Review of Triggers and Special Computing Hardware for the Fermilab Fixed-Target Program*, Guanajuato, Mexico, 1984. Proceedings of the Symposium on Recent Development in Computing, Processor and Software Research for High Energy Physics.

[Du02] P. Le Du. *Trigger and Data Acquisition for collider experiments – Present and future*. Presentation on the 8-th International Conference on Instrumentation for Colliding Beam Physics, March 2002. Available from World Wide Web: `http://www.inp.nsk.su/events/confs/instr2002/talks/020306/ledu.pdf`.

[Dun] A. Dunkels. *lwIP - A Lightweight TCP/IP stack* [online]. Available from World Wide Web: `http://www.sics.se/~adam/lwip/index.html`.

[ea98] R. Lay et. al. *MICROENABLE – A RECONFIGURABLE FPGA COPROCESSOR*. In Proc. Fourth Workshop on Electronics for LHC Experiments, pages 402–406, Rom, 1998.

[ea02] B. Gorini et al. *ROS Software architecture document*. Technical report, July 2002. Available from World Wide Web: `https://edms.cern.ch/file/364343/1.3/softWareArchitectureV13.pdf`.

[Fis02] K. Fischer. *Entwicklung eines Gigabit Ethernet Interfaces für einen FPGA Prozessor*. Diploma Thesis, Universität Mannheim, May 2002.

[FMTV] D. Francis, "M. Müller", L. Tremblet, and J. Vermeulen. *Summary of ROS system tests*. Available from World Wide Web: `http://agenda.cern.ch/askArchive.php?base=agenda&categ=a02164&id=a02164s5t2/transparencies`.

[Fra04] D. Francis. *Private communication*, 2004.

[FRB+00] R. Frühwirth, M. Regler, R.K. Bock, H. Grote, and D. Notz. *Data Analysis Techniques for High-Energy Physics*. Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology, 2000. ISBN: 0521635489.

[GGK02] B. Green, G.Kieft, and A. Kugel. *ATLAS TDAQ/DCS ROS Protoype-RobIn Software Interface*. CERN EDMS Note ATL-DQ-EN-0003, CERN, September 2002. Available from World Wide Web: `https://edms.cern.ch/file/356332/2.2/swid.pdf`.

[Gor02] Benedetto Gorini. *Private communication*, 2002.

[GPR+00] G.Boorman, P.Clarke, R.Cranfield, G.Crone, B.Green, and J.Strong. *The UK ROB-in: A prototype ATLAS readout buffer input module*. CERN ATLAS Note ATL-DAQ-2000-013, CERN, May 2000.

[Gro98a] ATLAS Level-1 Trigger Group. *Level-1 Trigger Technical Design Report*. CERN, June 1998.

[Gro98b] ATLAS/Trigger Performance Group. *ATLAS Trigger Performance Status Report*. Technical report, November 1998.

[Gro98c] PCI Special Interest Group. *PCI to PCI Bridge Architecture Specification, Rev. 1.1*. Technical report, December 1998.

[Gro03] ATLAS HLT/DAQ/DCS Group. *ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report*. CERN, 2003.

[HAK+04] C. Hinkelbein, A.Khomich, A. Kugel, R. Männer, and M. Müller. *Using FPGA coprocessor for improving execution speed of TRT LUT – one of the feature extraction algorithms for ATLAS LVL2 trigger*. page 247, February 2004.

[Hau03] R. Hauser. *Design of the Message Passing Interface*. Technical report, 2003. Available from World Wide Web: `https://edms.cern.ch/file/391517/0.5/DC-013.ps`.

[Hez04] S. Hezel. *FPGA-basiertes Template-Matching mit Distanztransformierten Bildern*. PhD thesis, Universität Mannheim, February 2004.

[Inf02] Infineon. *V23818-N305-L57(*) Small Form Factor Multimode 850 nm 2.5 Gbit/s OC-48 2x5 Transceiver with LC™ Connector*. Datasheet, 2002.

[IvdB02] W. Iwanski and E. van der Bij. *32-bit S-LINK to 64-bit PCI interface – Users Guide*. CERN, February 2002. Available from World Wide Web: `https://edms.cern.ch/file/249657/1/userguide.PDF`.

[KH98] W. Kemmler and M. Hein. *Gigabit-Ethernet – Der Standard – Die Praxis*. Fossil Verlag, 1998.

[Kra04] E. Krause. *Private Communication*, 2004.

[Kug04] Andreas Kugel. *Private communication*, 2004.

[Lie04] G. Lienhart. *Beschleunigung Hydrodynamischer Astrophysikalischer Simulationen mit FPGA-Basierten Rekonfigurierbaren Koprozessoren*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, July 2004.

[Lin] *Cross-Referencing Linux – Linux Kernel Sources* [online]. Available from World Wide Web: `http://lxr.linux.no/`.

[Mor03] G. Mornacchi. *Architecture, deferrals and costing*. ATLAS Week Presentation, February 2003. Available from World Wide Web: `http://agenda.cern.ch/askArchive.php?base=agenda&categ=a03192&id=a03192s0t4/transparencies`.

[MPR] *The MPRACE Board* [online]. Available from World Wide Web: `http://www-li5.ti.uni-mannheim.de/fpga/?race/`.

[Pos80] J. Postel. *User Datagram Protocol*. RFC 768, August 1980. Available from World Wide Web: `http://www.ietf.org/rfc/rfc0768.txt?number=768`.

[Pos81] J. Postel. *INTERNET PROTOCOL – DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION*. RFC 791, September 1981. Available from World Wide Web: `http://www.ietf.org/rfc/rfc0791.txt?number=791`.

[PRSZ95] Povh, Rith, Scholz, and Zetsche. *Teilchen und Kerne*. Springer Verlag, 1995.

[RC01] A. Rubini and J Corbet. *Linux Device Drivers, 2nd Edition*. O'Reilly, 2001. ISBN: 0-59600-008-1.

[Ris99] R. Rissmann. *Implementierung von Vorverarbeitungsalgorithmen fuer den ATLAS Level 2 Trigger auf dem FPGA-Prozessor microEnable*. Diploma Thesis, University of Heidelberg, February 1999.

[RMR02] R.Cranfield, M.LeVine, and R.McLaren. *ROS Requirements*. Technical report, 2002. Available from World Wide Web: `https://edms.cern.ch/file/356336/1.0.0/ros_urd_1000.pdf`.

[RSC] *RS Components, Electronic Components* [online]. Available from World Wide Web: `http://www.rs-components.com`.

[RvdBH] A. Ruiz, E. van der Bij, and S. Haas. *HOLA – High-speed Optical Link for Atlas*. Available from World Wide Web: `http://hsi.web.cern.ch/HSI/s-link/devices/hola/datasheet.pdf`.

[Sam03a] Samsung. *SDRAM Device Operations*. Technical report, 2003. Available from World Wide Web: `http://www.samsung.com/Products/Semiconductor/DRAM/TechnicalInfo/sdram_device_operation_full_version_200401.pdf`.

[Sam03b] Samsung. *SDRAM Timing Diagram*. Technical report, 2003. Available from World Wide Web: `http://www.samsung.com/Products/Semiconductor/DRAM/TechnicalInfo/sdram_timing_diagram_20040205.pdf`.

[Ses00] M. Sessler. *Algorithms on CPUs and FPGAs for the ATLAS LVL2 Trigger*. PhD thesis, Universität Heidelberg, February 2000.

[Sil] *Silicon Software GmbH* [online]. Available from World Wide Web: `www.silicon-software.de`.

[Sim04] H. Simmler. *Private Communication*, 2004.

[Sin00] H. Singpiel. *Der ATLAS LVL2-Trigger mit FPGA-Prozessoren*. PhD thesis, Universität Heidelberg, November 2000.

[sup] *Supermicro mainboard X5DPE-G2* [online]. Available from World Wide Web: `http://www.supermicro.com/products/motherboard/Xeon/E7501/X5DPE-G2.cfm`.

[Tec] PLX Technology. *PCI 9656 Data Book*. PLX Technology. Available from World Wide Web: `http://www.plxtech.com`.

[TPH] *The Trentonprocessors Home Page* [online]. Available from World Wide Web: `http://www.trentonprocessors.com/products/`.

[vdBH] E. van der Bij and S. Haas. *CERN S-LINK homepage*. Available from World Wide Web: `http://hsi.web.cern.ch/HSI/s-link/Welcome.html`.

[Ver] J. C. Vermeulen. *Atlas Papermodel Version 2.1* [online]. Available from World Wide Web: `http://www.nikhef.nl/pub/experiments/atlas/daq/Paper2003/Papermodel21.tar.gz`.

[VVT03] J. Vermeulen, Valerio, and S. Taprogge. *Beauty and the Beast aka PESA and the ROS*. Presentation, ROS I/O Path Meeting, December 2003.

[Wol98] T. Wolf. *Die Systemsoftware für den First Level Trigger des HERA-B Experiments*. PhD thesis, Universität Mannheim, August 1998.

[XIL] *Xilinx Web page* [online]. Available from World Wide Web: `http://www.xilinx.com/`.

[Xil03] Xilinx. *Virtex™-II Platform FPGAs: Complete Data Sheet*. Technical report, 2003.

[XIP] *Xilinx IP Cores* [online]. Available from World Wide Web: `http://www.xilinx.com/xlnx/`.

Acknowledgments

I would like to thank all the people who have supported me during the development and writing phase of this thesis.

First of all I want to thank my family and my wife for the patience and support.

Furthermore I like to thank Prof. Dr. R. Männer, who gave me the possibility to work in his research group. Many thanks also to the people of the FPGA group: Andrei, Maoyun, Christian, Gerhard, Stefan, Oliver, and especially Andreas Kugel. Many people at CERN have to mentioned too here for their support and help: David, Benedetto, Jorgen, Markus, Jos, and Per. Finally I would like to thank Andrzej (RHUL) for cross reading the manuscript.