Architecture and Prototype of a Real-Time Processor Farm Running at 1 MHz

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

vorgelegt von
Alexander Walsch, Master of Science
aus Günzburg

Mannheim, 2002

Dekan: Professor Dr. Herbert Popp, Universität Mannheim
Referent: Professor Dr. Volker Lindenstruth, Universität Heidelberg
Korreferent: Professor Dr. Reinhard Männer, Universität Mannheim

Tag der mündlichen Prüfung: 6. November 2002

Abstract

Triggers are key components in large scale high energy physics experiments. Nowadays experiments produce data at a rate of several TByte/s, which is more than any storage system can handle. Not all of the data produced is of importance and as a matter of fact only a tiny amount of all events are interesting to the experiment. Thus, data has to be reduced in real-time. This is commonly done by pipelined trigger systems.

The Level-1 trigger of the LHCb experiment is a hardware/software based system built around standard components whenever possible. The trigger is the second stage in the LHCb trigger pipeline having an average input of 1 MHz and a bandwidth requirement of more than 4 GByte/s. The input data is initially split amongst several input feeds with sub-events being as small as 128 Byte. Data have to be sent to a compute node which runs a track finding algorithm and produces a result message at a MHz rate.

The system uses a high-speed network available off-the-shelf which connects commodity PCs. The interface to the NIC is PCI.

The thesis gives an overview of the different networking requirements that have to be met to satisfy the LHCb boundary conditions. The requirements of the project do not allow to use common data transfer methods. However, based on a shared memory architecture a new method of transferring data is introduced. The method uses additional reconfigurable logic which allows to send data in bursts over the PCI bus directly into the network inducing almost no overhead.

Based on the Scalable Coherent Interface, tests concerning speed, throughput, latency, and scalability are presented. Based on the latest simulation results done at CERN, an approximate system size is calculated and a basic timing analysis of the system is given. The system is characterized by multiple nodes sending to one single receiver. Therefore, the Level-1 trigger is prone to network congestion since the receiving node can not handle the aggregate input data rate. However, a hardware based data scheduling mechanism, the TagNet, is introduced which avoids congestion in the system.

A 30 node prototype is presented which has been built around Linux PCs connected by an SCI network. The system is able to process data with a MHz rate. Sub-events have been chosen to be as small as 128 Byte. Data transfer has been scheduled by a basic implementation of the TagNet. The system has been used to prove basic functionality and to measure important input parameters concerning the system.

Zusammenfassung

Trigger Systeme sind Schlüsselkomponenten in modernen Hochenergie Physik Experimenten. Gegenwärtige Experimente erzeugen Datenraten, die einige TByte/s betragen. Kein Speicher kann diese Datenmenge aufnehmen. Zweitens sind auch nicht alle Daten relevant. Nur ein kleiner Bruchteil der Ereignisse sind für das Experiment von Bedeutung. Deshalb entwickelt man Trigger Systeme. Diese reduzieren die Datenmenge, indem sie nur die relevanten Daten selektieren. Am Ende ist die Datenrate so gering, dass man auf Band schreiben kann.

Trigger Systeme sind häufig mehrstufig aufgebaut. Der Level-1 Trigger des LHCb Experiments am CERN ist die zweite Stufe eines vierstufigen Systems und hat eine Eingangsrate von 1 MHz. Die Gesammtbandbreite, die das System bereitstellen muss, beträgt voraussichtlich 4 GByte/s. Die Eingangsdaten sind verteilt und nur 128 Byte im Durchschnitt. Alle Daten, die zu einem Ereignis gehören, müssen an einen bestimmten Rechner gesendet werden.

Das System benutzt ein Hochgeschwindigkeits Netzwerk, das kommerziell erhältlich ist. Als Schnittstelle zwischen Rechner und Netzwerk ist PCI vorgesehen.

Diese Arbeit beschäftigt sich mit den Anforderungen an das Netzwerk, die erfüllt werden müssen, damit es für LHCb in Frage kommt. Auf einer Shared Memory Architektur basierend wird eine neue Art der Datenübertragung erarbeitet, die externe rekonfigurierbare Logik verwendet und sehr gut zum verschicken von kleinen Datenblöcken geeignet ist.

Basierend auf der SCI Technologie werden Tests vorgestellt, die Geschwindigkeit, Durchsatz, Latenz und Skalierbarkeit diskutieren. Die Grösse des endgültigen Systems wird anhand Simulationsdaten berechnet und eine Zeitanalyse vorgestellt.

Da ein Datenempfänger eine Eingangsbandbreite von 4 GByte/s nicht empfangen kann, muss der Datentransfer in Stufen stattfinden. Deshalb wird ein Netzwerk vorgestellt, das die Daten orchestriert, um Netzwerküberlastungen vorzubeugen.

Die erarbeiteten Konzepte werden anhand eines 30 Knoten Prototyps vorgestellt. Der Prototyp besteht aus Linux PCs, die durch ein SCI Netzwerk verbunden sind. Das System kann Datenblöcke, die nur 128 Byte gross sind, mit einer MHz Rate verarbeiten. Der Datentransfer wird durch eine vorläufige Version des Orchestrierungs Netzwerks gesteuert.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Trigger systems are widely used in high energy physics experiments to reduce the tremendous amount of data by event selection and filtering in real-time. Taking the LHCb experiment as an example, 10^{12} $b\bar{b}$ pairs are created in one year. The total number of interactions can be calculated to be 4×10^{14}. However, due to the small branching ratios for interesting B decays of 10^{-5} or less and the limited detector acceptance the events that contain valuable physics information have to be extracted. This happens by pipelined trigger systems. The first trigger stage selects events based on part of the event data. All subsequent trigger stages take those events and apply different, specialized algorithms to reduce background events.

This thesis describes the requirements of the LHCb Level-1 trigger, the second of four trigger stages. A networked cluster of PCs is used to run the trigger algorithm. The farm has to process one event every microsecond with the total bandwidth required being about 4 GByte/s.

1.1 High Energy Physics Experiments

Accelerator experiments in high energy physics are characterized by interaction of particles, particle production, and particle decay. Production and decay have different probabilities which are described by Quantum Mechanics and observed in particle detectors.

Taking the LHCb experiment as an example, protons collide with a frequency of 40 MHz. The interaction point, the primary vertex, can be measured with a resolution of $49\mu m$ along the beam axis using a silicon vertex detector. The experiment is intended to measure decay probabilities of B-mesons. Those particles decay a few mm apart, at the secondary vertex, and are measured with a resolution of $225\mu m$ along the beam axis. About 10^{12} $b\bar{b}$ pairs are expected in one year of data taking ($10^7 s$) [1]. However, due to limited detector acceptance and small branching ratios of the order of 10^{-5} or less, only a small fraction of events are stored for offline analysis. The LHCb trigger system will select and filter this fraction of interesting events from the large number of events producing b quarks and other pp events in real-time.

1.2 High Energy Physics at CERN

CERN is the Organization for Nuclear Research and the world's largest particle physics center. The word "CERN" is a French abbreviation and stands for "Convention Européene de la Recherche Nucléaire". CERN has been founded in 1954 being one of the first European joint ventures. From the original 12 signatories to the CERN convention, membership has grown to 20. More than 6500 scientists, which amounts to more than half of the world's particle physicists, come to CERN doing their research. 500 universities and over 80 nationalities are represented. One of the collaborating institutes is the "Kirchhoff-Insitut für Physik" in Heidelberg, Germany. Its responsibility is, among others, to build a small, but important part for one of the upcoming accelerator experiments - the Level-1 trigger for the LHCb experiment.

However, the challenge imposed by the experiments is not only an attracting place for particle physicists. Computer scientists and engineers from around the world contribute to the electronics and computer systems required by the experiments to handle the huge amount of data. Sometimes even every days life is affected by results achieved by the people working at CERN. The most famous example is certainly the invention of the World Wide Web by Tim Berners-Lee in 1989. The original proposal can be found under [2].

Currently a new accelerator, the LHC[1], is installed in the place of the LEP[2]. Figure 1.1 shows an aerial view of the area. The large circle demonstrates the path which accelerated particles will take. Its circumference is about 27 km. CERN is located close to Geneva in Switzerland. However, more than half of the area covered by the accelerator ring is located in France.

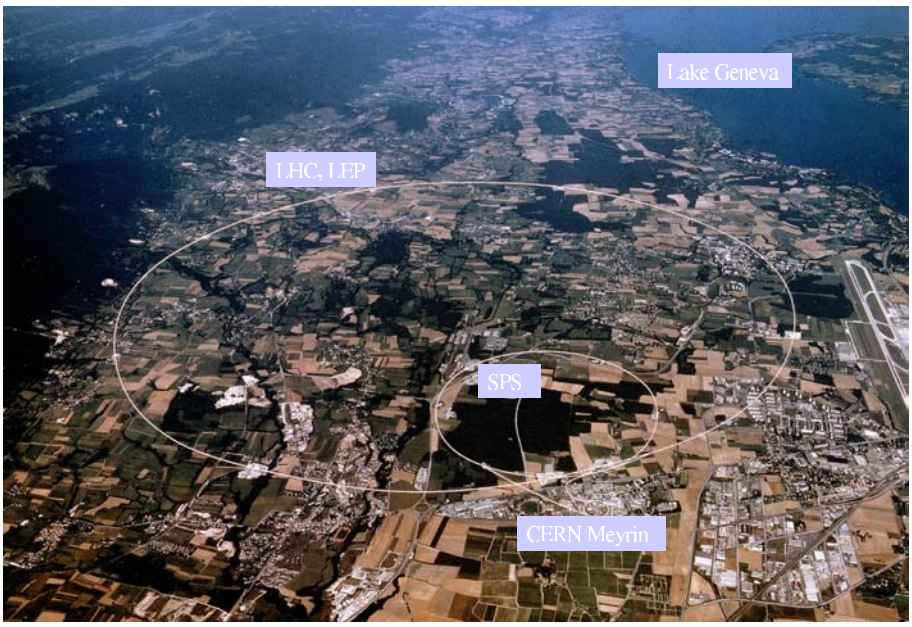

Figure 1.1: CERN as seen from above. Figure from [3].

[1]LHC: Large Hadron Collider.
[2]Large Electron Positron Collider

Originally the LHC experiments have been scheduled to take their first data in the year 2005. However, because of delays the experiments will start in the year 2007.

Four new LHC experiments, each having its own detector, are currently under construction:

- **ALICE**: The aim is to study the physics of strongly interacting matter at extreme energy densities, where the formation of a new phase of matter, the quark-gluon plasma, is expected.
- **ATLAS**: A general purpose experiment having the main effort to discover the Higgs particle[3].
- **CMS**: A general purpose experiment looking into physics beyond the Standard Model and into discovery of the Higgs.
- **LHCb**: Precision Measurements of CP violation and rare decays.

Figure 1.2 shows the location of the four LHC experiments. All detectors are located about 100 m underneath the surface. The LHCb experiment, which is the one of interest in this thesis, is depicted on the right bottom.

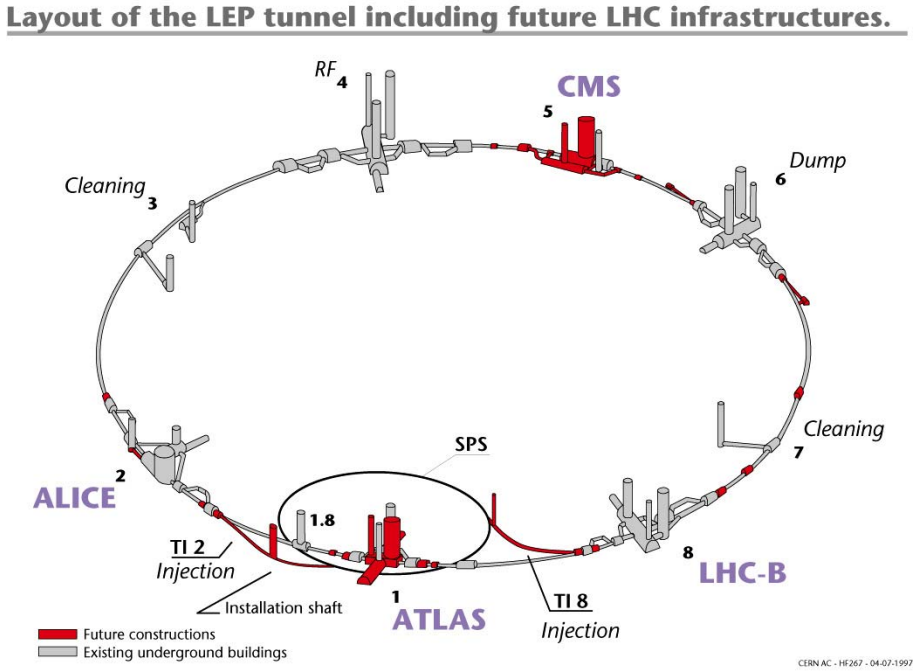

Figure 1.2: Location of LHC experiments. Figure from [3].

1.3 Trigger Systems

The data rate of nowadays experiments is in the TByte/s range exceeding the capabilities of storage systems today and in the foreseeable future. Secondly, but even more impor-

[3]The Higgs Particle or Higgs Boson is associated to the origin of mass.

tant, not all data are of interest for physics offline analysis. Therefore, the data has to be analyzed and successively reduced in real-time. Events that are valuable for physics analysis are kept whilst events which are not relevant are discarded. However, this makes the trigger system to the most crucial part in the experiment since any failure results in loss of valuable physics data or even in wrong interpretation of results. Triggering is commonly done by pipelined systems. Usually systems like that have three or four pipeline stages.

The first stage in the system selects events by analyzing a part of the total data stream coming from the detector. Only events that have been selected are analyzed by the other trigger stages. The following trigger stages reject those remaining events based on a filtering algorithm looking for specific characteristics. Therefore, the data rate is successively reduced such that final storage on tape is feasible. The last trigger stage usually analyzes all data associated with an event while the other stages deal with a subset. Every pipeline stage applies the same principle; data are kept in a buffer and a copy or subset of it are analyzed by the trigger logic. However, since buffers are not of infinite size a trigger decision has to be made in time to avoid a buffer overflow. Thus, the trigger decision has to be broadcast within a certain latency window.

Trigger logic can be implemented either in hardware or software. Software solutions are often desired to stay flexible during the runtime of the experiment. However, software is usually not feasible for front-end trigger stages since the latency requirements can not be met.

Upon a positive trigger decision the buffered data are forwarded to the next trigger stage. Since a large fraction is rejected both the overall amount of data and the data rate are reduced such that storage on tape for final analysis is feasible.

Figure 1.3 shows how data are successively reduced by the LHCb Level-0 and Level-1 trigger systems by applying the principle explained above.

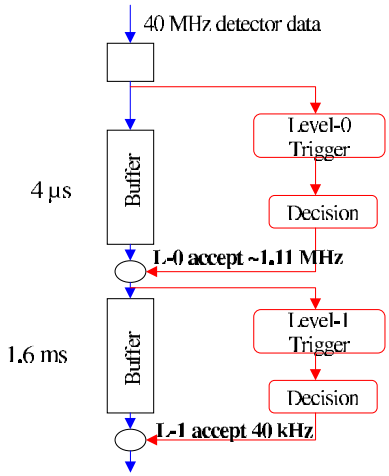

Figure 1.3: Sample trigger logic and buffering scheme.

Taking the LHCb experiment as an example the suppression factor of the first trigger stage, the Level-0 , is 40, reducing the event rate from 40 MHz to 1 Mhz. During analysis the Level-0 input data is kept in a buffer. However, the maximum latency is $4\mu s$. After a Level-0 accept the accepted events are analyzed by the Level-1 trigger which accepts events at 40 kHz. The data are buffered for 1.6 ms which is the total time that can be used by

hardware and software being in the Level-1 latency path. Therefore, two important tasks have to be fulfilled when designing a trigger stage:

- The hardware, e.g. networks, busses, have to cope with the data rate.
- The overall latency requirements have to be met.

The LHCb experiment will look at certain asymmetries in the decay of the B-meson. However, similar experiments are running at the moment. Before the LHCb trigger system is introduced in chapter 2 the trigger systems of the high energy physics experiments HERA-B [4] at DESY[4], and BaBar [5] at SLAC[5] are described briefly.

1.3.1 The HERA-B Trigger System

The HERA-B trigger system comprises four trigger stages which successively reduce the input rate and look for physics information. However, there is also a pre-trigger stage involved that does not reduce the data rate but rather chooses a region of interest.

The following stages are involved:

- The Level-1 Trigger (First Level Trigger, FLT) consists of a network of roughly 100 custom-built massively parallel and pipelined processors. It receives the initial detector data at a rate of 10 MHz and must deliver a suppression of 200 with a latency of $12\mu s$.
- The Level-2 Trigger (Second Level Trigger, SLT) handles an input rate of 50 kHz. It is a software based trigger running on a compute farm of 240 Linux PCs. It reaches a suppression factor of 100, which results in an output rate of 500 Hz. The average processing time is about 4 ms.
- The Level-3 Trigger (Third Level Trigger, TLT) is executed on the same compute farm and reduces the rate by a factor of 10. It is the first trigger stage that has access to all event data whilst the FLT and SLT see only part of the entire data stream. The latency has been set to 100 ms.
- The Level-4 Trigger (Fourth Level Trigger, 4LT) is implemented on a different compute farm using 200 CPUs. The data rate is reduced by a factor of 2.5 before data is stored on magnetic tape.

1.3.2 The BaBar Trigger System

The BaBar trigger is a two-level hierarchy trigger. The Level-1 trigger has been implemented in hardware whilst the Level-3 trigger is a software approach:

- The Level-1 trigger is configured to have an output rate of less than 2 kHz with triggers typically produced within $11 - 12\mu s$. The trigger data are processed by three specialized hardware processors.

[4]DESY: Deutsches Elektronen Synchrotron located in Hamburg, Germany.
[5]SLAC: Stanford Linear Accelerator Center located at Stanford University n Menlo Park, CA.

- The Level-3 trigger is running on an online computer farm with access to the complete event data. The output rate of the Level-3 trigger has been set to less than 100 Hz. There is no Level-2 trigger involved in data analysis.

1.4 Motivation of this Thesis

This thesis describes the requirements of the LHCb Level-1 trigger and the work that has been done to build a first prototype. The Level-1 trigger is the second of four trigger stages and has been planned as a software trigger to stay flexible during the runtime of the experiment. Usually experiments like this run for about 15 years.

The trigger receives a new event being more than 4 kByte in size every microsecond. Therefore, the trigger system requires a sophisticated network solution. A second boundary condition that has been set on the trigger stage is a tight latency requirement. In 1999 the latency had been agreed to be $256\mu s$. However, since events are queued both at the input stage and the processor the latency has been increased to 1.6 ms in 2001.

A software based trigger could be built around processors that compute a trigger result in parallel. However, additional communication overhead as imposed by MPI[6], e.g., suggests to favor a single processor solution. Therefore, the trigger is planned as a cluster of PCs with each CPU running a process that analyzes one complete event. However, delay in data analysis must be avoided. Delay of data analysis of one microsecond means one additional CPU and thus additional cost of 3k Euro. Upon completion of analysis a result message is sent to a central trigger supervisor.

The incoming event is initially scattered amongst some input feeds with the fragment size being not more than 200 Byte[7] on average. All fragments have to be delivered to a specific compute node. Upon arrival of the complete event the software starts to process the data. Since the total input data stream equals 4 GByte/s, the data must be scheduled since the receiving node can not handle incoming data at such rate.

The requirements on the trigger can be summarized as follows:

- A system has to be built that allows transmission of event fragments with a MHz rate. Since event fragments are small this part has been considered as the most crucial part of the system.
- Event fragments have to reach a specific target without causing network congestion.
- The system should use standard components whenever possible. The amount of custom built electronics should be minimized.
- The trigger issues an output message in chronological order with a MHz rate.

The network under investigation is a shared memory based solution, which can transfer messages without any software overhead; the Scalable Coherent Interface (SCI). However, to satisfy the MHz requirement of the LHCb Level-1 trigger a method which has been named hardware initiated DMA (HDMA) has been invented. HDMA uses an external

[6] Message Passing Interface

[7] This number has been used as assumption in 1999.

DMA engine, which is located on the same PCI bus, to push data into a shared memory region. This transfer scheme is also called device-to-device copy. Therefore, DMA logic has been implemented in an FPGA with PCI interface. The first FPGA that has been used was an ORCA[8] chip by Lucent [6]. Results looked very promising when ORCA FPGAs were used in tandem mode as explained in 5.2.3. However, when modern motherboards providing a 66 MHz/64-bit PCI bus arrived on the market using the ORCA chips was not making sense anymore since the internal logic could not be clocked at 66 MHz. Another drawback was the internal 32-bit data bus of the ORCA chip feeding the external 64-bit PCI bus. Thus, everything had to be transferred to faster FPGAs. It has been decided to use Altera APEX[9] devices. The vast majority of the results in this thesis is based on Altera chips.

[8] OR3TP12 FPSC
[9] APEX20KE

Chapter 2

The LHCb Experiment

The LHCb experiment studies CP violation on neutral B-mesons. A spectrometer is used to detect particle tracks. To get the best possible physics performance, it covers low polar angles up to 300 mrad. The detector reconstructs B-decay vertices with a resolution being in the micrometer range. Different sub-detectors provide particle information which is analyzed by the trigger system. The four stage pipelined trigger successively reduces the data rate from 40 MHz to 200 Hz such that it can be written on tape for offline analysis.

2.1 Physics Introduction

The LHCb experiment is dedicated to the study of CP violation in hadronic systems which originate from b-quarks. CP violation can be observed in decay asymmetries of the neutral B-meson. CP violation has first been observed in the neutral kaon system. In the Standard Model of particle physics the mixing of quarks is described by the Cabibbo-Kobayashi-Maskawa (CKM) matrix. The CKM matrix has four free parameters, three Euler angles, and one complex phase. The latter allows CP violation to occur within the Standard Model. CP violation in the kaon system is tiny and implies theoretical uncertainties. However, the Standard Model predicts large effects in the B-meson system.

2.2 The Large Hadron Collider

The Large Hadron Collider (LHC) is a high luminosity proton-proton collider with a center of mass energy of 14 TeV. Luminosity is a magnitude which corresponds to the number of particle interactions at the collision point. Compared to other experiments that are in operation or under construction, LHCb will be the richest source for B-mesons. Figure 2.1 shows the polar angles θ at which b and $\overline{b}$-hadrons have to be expected. The depicted maxima suggest that the LHCb detector should cover low polar angles. The polar angle is defined with respect to the beam axis in the pp center-of-mass system.

The LHCb experiment plans to run with an average luminosity of $2 \times 10^{32} cm^{-2} s^{-1}$. However, LHCb's luminosity is less compared to Belle ($4.6 \times 10^{33} cm^{-2} s^{-1}$) [7] and BaBar

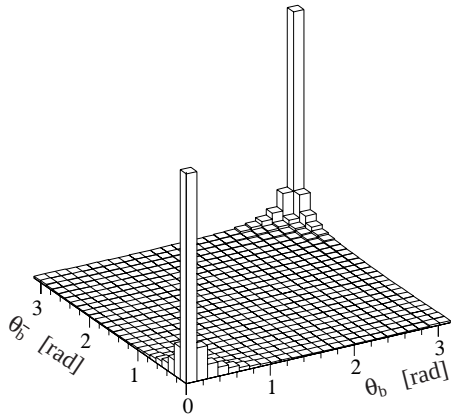

Figure 2.1: Polar angles of the b- and $\bar{b}$-hadrons calculated by the PYTHIA event generator. Figure from LHCb TP [1]

($7.25 \times 10^{33} cm^{-2} s^{-1}$) [5] since events with multiple interactions should be minimized. Events at LHCb are characterized by proton-proton (pp) interactions, which produce about $10^{12} b\bar{b}$ pairs in one year. The LHCb detector is designed to exploit the large number of b-hadrons produced in order to make precision studies of CP asymmetries and rare decays in B-meson systems. However, the amount of interesting events is in the order of 10^{-5} or less and thus, requiring an effective trigger system.

2.3 The LHCb Detector

The LHCb detector is a single-arm spectrometer with a forward angular coverage from 10 mrad to 300 mrad in horizontal projection and to 250 mrad in the vertical projection [1]. The layout of the spectrometer is shown in figure 2.2.

The detector can reconstruct B-decay vertices with a resolution in the micrometer range and provides particle identification for charged particles. A high performance trigger, which is optimized to select events with B-mesons efficiently, is foreseen. Based on particles with large transverse momentum and displaced secondary vertices a trigger decision is made.

LHCb comprises a number of different sub-detectors. Parts of the detector are outlined briefly:

- The beam pipe around the interaction point is divided into two conical sections. The first leads through RICH1 and has an opening angle of 25 mrad, the second section has a 10 mrad opening angle.
- The Vertex Locator (VELO) features a series of silicon stations placed along the beam direction. It is used to provide precise measurements of track coordinates close to the interaction region. It is discussed in more detail in 3.1.
- Two Ring Image Cherenkov Detectors identify charged particles over the momentum range 1-150 GeV/c. The upstream detector (RICH1) contains both a silicon aerogel and a C_4F_{10} gas radiator whilst RICH2, located downstream behind the magnet, contains a CF_4 radiator. Three different radiators are used to cover the full momentum range.

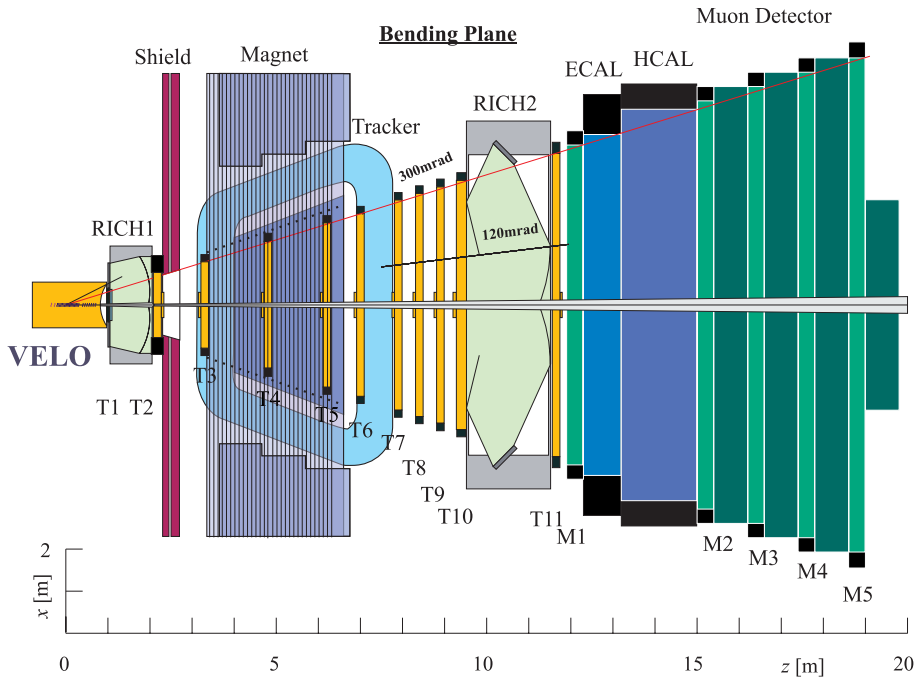

Figure 2.2: Schematic drawing of the LHCb detector as seen from above. Figure from the VELO TDR [8].

- The spectrometer dipole magnet is located close to the interaction region to minimize its size. Magnets are used to bend charged particle tracks depending on their momentum and charge.

- The tracking system (T1-T9) consists out of a series of stations containing Inner (IT) and Outer Tracker (OT) components. The tracker provides track reconstruction and precise momentum measurement for charged particles.

- The calorimeter system comprises a scintillator pad detector (SPD), a preshower detector (PS), an electromagnetic calorimeter (ECAL), and a hadron calorimeter (HCAL). The purpose of the calorimeter system is to provide identification of electrons and hadrons for trigger and offline analysis, with measurement of position and energy.

- The muon detector (M1-M5) provides muon identification and Level-0 trigger information.

The detector is presented as foreseen in the LHCb TP [1]. However, the latest developments in the LHCb experiment do foresee a detector which implies reduced material budget — the LHCb light detector. Changes to the detector will also affect the Level-1 trigger input since it is planned to have a magnetic field in the VELO to get momentum information. A special station called TT1 is also foreseen as Level-1 input. However, the amount of data and the data rate, which are very important boundary conditions, do not change.

2.4 The LHCb Trigger System

The detector produces a total data rate of 40 TByte/s and thus exceeds capabilities of present network architectures and storage media. LHCb implements a selective four stage trigger system that analyzes the sub-detectors in different trigger stages. Events with B-mesons can be distinguished from other inelastic pp interactions by the presence of secondary vertices and particles with high transverse momentum p_T. Figure 2.3 shows the different stages of the LHCb trigger system.

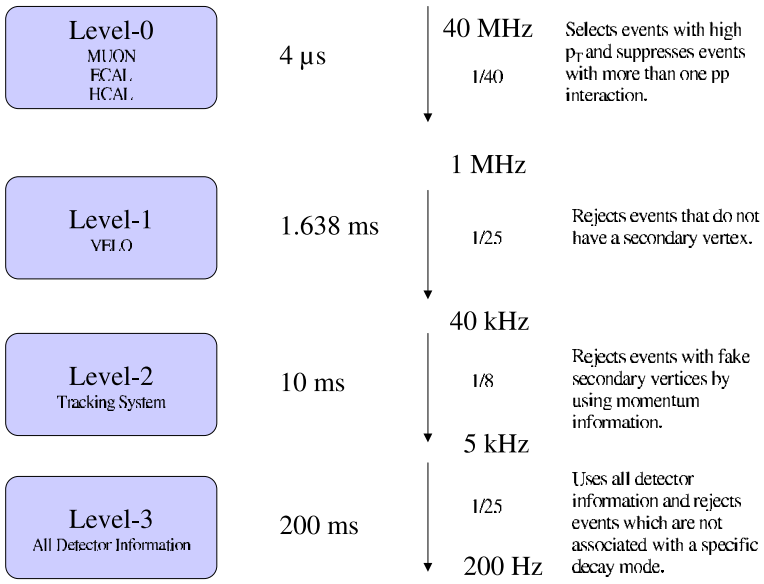

Figure 2.3: LHCb trigger system stages including latencies and suppression rates.

The pipeline comprises the following stages:

- **Level-0** comprises three high p_T triggers, which operate on muons, electrons, and hadrons. Additionally a pile-up veto is issued, which suppresses events with more than one pp interaction. The Level-0 operates on the bunch-crossing frequency of 40.08 MHz. The maximum Level-0 output rate is limited by the Level-0 derandomizer and accounts to 1.11 MHz. The latency of the Level-0 is set to be 4.0 μs.

- The **Level-1** selects events, which have a secondary vertex. The Level-1 operates at Level-0 output rate of 1.11 MHz and reduces the event rate to 40 kHz. The Level-1 overall latency has been set to 1.638 ms. The Level-1 trigger is discussed in chapter 3.

- The **Level-2** eliminates events with fake secondary vertices by using momentum information. The Level-2 operates at the Level-1 output rate and achieves a suppression factor of 8. Its latency is about 10 ms.

- The **Level-3** uses all detector information and selects events, which are associated with specific b-hadron decay modes. Level-3 has a latency of about 200 ms and accepts events at the data recording rate of 200 Hz.

A specific feature of LHCb is the fact that Level-0 and Level-1 trigger decisions are transmitted by the TFC. The TFC also distributes the LHC reference clock which runs at 40 MHz. The clock is used to drive all the electronics such that a synchronous readout is possible. Control commands which are used to reset the front-end electronics or to recover from an error condition are also transferred by the TFC. In case of a calibration event the TFC has to guarantee that triggers corresponding to calibration events are accepted. The TFC distribution network is based on the Trigger, Timing, and Control (TTC) network developed by the RD12 experiment [9]. The network transmits the information over an optical channel. If a device depends on timing information it must incorporate a TTC receiver chip (TTCrx).

Chapter 3

The LHCb Level-1 Trigger

The Vertex Locator is the data source for the Level-1 trigger. It is comprised of a number of silicon stations which allow precision measurements of charged particle tracks around the interaction region. The detector is read out by front-end chips in a radiation environment. The data gets digitized and preprocessed by the Off Detector Electronics and is finally sent to the input stage of the Level-1 processor farm. The Readout Units assemble sub-events out of data belonging to the same event and send that data to a specific compute node. The input rate is 1 MHz on average with peak values up to 1.11 MHz.

A track finding algorithm performs its track finding task once all sub-events have arrived. Upon completion of the algorithm a result message is sent to the Level-1 Decision Unit which produces the 40 kHz output signal.

3.1 Vertex Locator

The vertex locator is the data source for the Level-1 trigger. Furthermore, a second source of input will be part of the final system; the trigger tracker TT1 will provide momentum information. TT1 will be a full Si tracker composed of two half-stations being located between RICH1 and the magnet (see figure 2.2 for location of the RICH1 and the magnet). The final detector layout is changed at the time of writing. Some information which is more current than figure 2.2 can be obtained from a talk by Tatsuya Nakada given for the LHCC[1] referees [10].

The vertex locator (VELO) provides information for precise reconstruction of charged particle tracks around the interaction region. Displaced secondary vertices are a significant sign of b-hadron decay and thus vital to the experiment. The detector features a series of silicon stations which are placed along the beam axis. It is the only detector which also provides some backwards information which is used to separate events with more than one primary vertex. The location of the VELO within the LHCb detector can be seen in figure 2.2.

[1] Large Hadron Collider Committee

Since the geometry of the VELO is not final at the time of writing the design as described in the VELO TDR [8] is discussed instead. The final geometry will probably have less stations which reduces the amount of data that has to be handled by the Level-1 trigger. However, since the trigger tracker has been introduced as an additional input the final amount of data does not change significantly. The TDR VELO comprises 25 stations each providing an R and ϕ measurement. The arrangement of the disc-shaped stations is shown in figure 3.1.

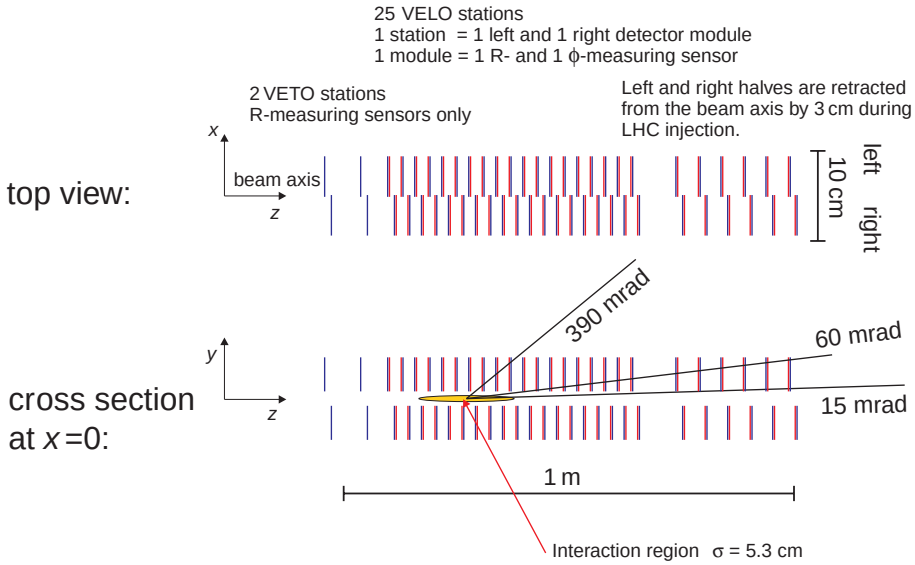

Figure 3.1: Arrangement of the detector stations along the beam axis. The interaction region ($\pm\sigma$) is depicted as well. Figure from VELO TDR [8]

The two stations depicted on the left are exclusively R measuring stations and part of the Level-0 pile-up veto system. The TDR VELO is located outside the magnetic field which implies straight tracks. However, in the final VELO design the detector will most likely be placed in a magnetic field to gain momentum information. Detector information is read out by analyzing the 204.800 analog readout channels. One VELO station is made of four half-circular silicon sensors covering the left and right part and measuring R and Φ respectively. Figure 3.2 shows a schematic view of a R- and Φ- measuring sensor.

The R-measuring sensor has azimuthal strips at constant radius whilst the Φ-measuring sensor has radial strips with a stereo angle to resolve track finding ambiguities. Both sensor types span 182° and have variable strip pitches. The R-sensor has strip pitches from $40\mu m$ up to $92\mu m$ whilst the Φ-sensor has strip pitches from $37\mu m$ to 98 μm. The innermost radius for both sensor types is 8 mm and the outermost radius 42 mm. The detector has a low occupancy of less than 1%. Test-beam measurements have shown R-sensor resolutions in between $3.6\mu m$ and $4.6\mu m$ depending on the strip pitch and the track angle[8].

Figure 3.2 also shows the 16 front-end chips mounted on one sensor. Every front-end chip will read out 128 channels in a radiation environment. Currently two candidates do exist that are under study:

- The SCTA_VELO, a chip derived from the SCT128A [11] developed for ATLAS [12].

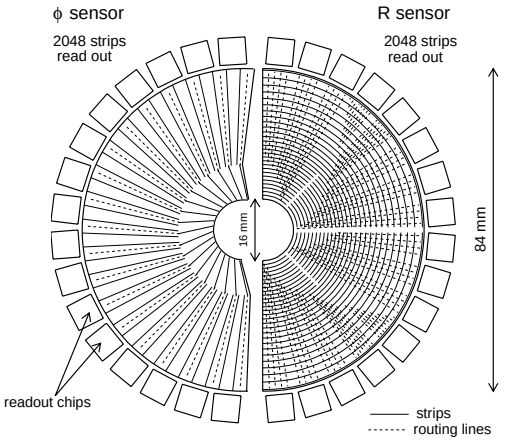

Figure 3.2: Schematic view of a R- and Φ- measuring sensor. Figure from VELO TDR [8]

- The BEETLE chip [13] developed in the ASIC laboratory of the Kirchhoff-Institut für Physik in Heidelberg [14].

Both front-end chips have to sample the detector information with the LHC bunch crossing frequency of 40 MHz. The data have to be stored in the chip for 4 μs until the Level-0 trigger decision is received. The Level-0 trigger accepts events with a maximum frequency of 1.11 MHz which corresponds to a readout time of 900 ns. Reading out the buffer with the Level-0 output frequency is achieved by multiplexing the channels in groups of 32 at 40 MHz. Thereafter, the analog data is transmitted via twisted pair cables to the Off Detector Electronics (ODE) located in the radiation free counting room. Figure 3.3 shows the architecture of the front-end.

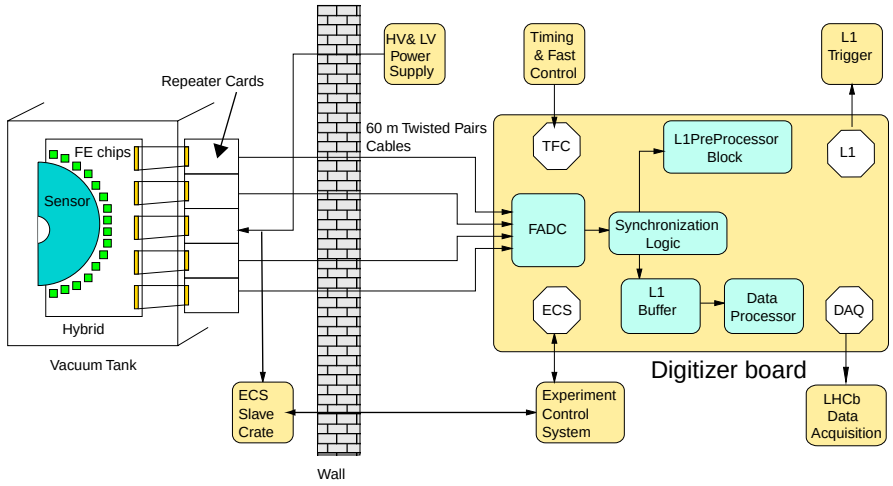

Figure 3.3: Front-end electronics architecture. Figure from VELO TDR [8].

3.2 Front-End Electronics

The Off Detector Electronics (ODE) board comprises the following functionalities:

- Digitization of the analog data coming from the front-end chips.
- Level-1 trigger preprocessing.
- Data Buffering during Level-1 latency.
- DAQ interface.

A schematic of the ODE can be seen in figure 3.4.

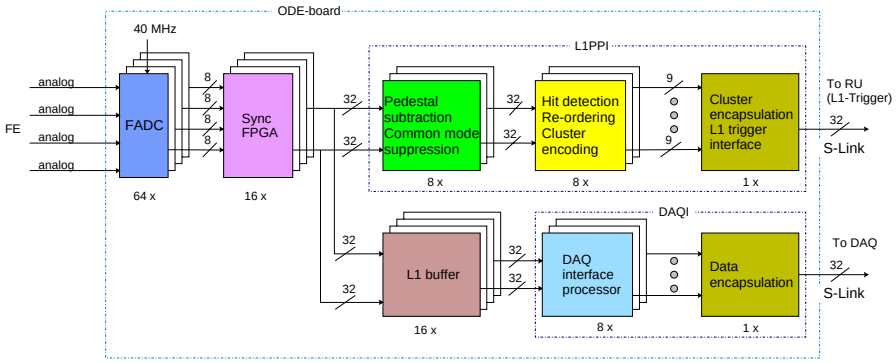

Figure 3.4: Block diagram of the off detector electronics. Figure from [15].

Each ODE board receives 64 analog signals from 16 front-end chips. Since one analog signal transmits data originating from 32 detector channels this accounts to 2048 detector channels total. The signals are digitized, processed, and packed with event header information. A Flash ADC (FADC) with 8-bit precision clocked at 40 MHz is used. Afterwards the data is shipped to the Level-1 trigger processor and the DAQ.

The ODE buffers the data for roughly 1.6 ms. In this time the ODE has to preprocess the data for the Level-1 trigger, the data have to be shipped to the Level-1 trigger, the Level-1 algorithm has to run the trigger algorithm, and a Level-1 trigger decision has to be made.

The Level-1 trigger preprocessor interface (L1PPI) performs the following tasks on the digitized data coming from the VELO:

- **Pedestal subtraction**

  Each channel of the detector has an offset (pedestal). The pedestal is measured in a special run and can be subtracted at first. The pedestal data, which is 1 Byte in size, can be downloaded by ECS[2] before the experiment starts.

[2]Experimental Control System

- **Faulty channel masking**

  If a detector channel is faulty it must be ignored. One mask bit per detector channel can be downloaded via ECS. This prevents wrong detector hit information in case of channel oscillations.

- **Common mode suppression**

  So far only linear common mode suppression has been analyzed. The implementation is flexible such that different algorithms can be implemented. For a more detailed coverage see [15].

- **Hit detection**

  A hit is defined as a channel which has an amplitude above a certain threshold value. The threshold value is an 8-bit value which can be downloaded via ECS.

- **Topological re-ordering (optional)**

  This feature only applies when the detector channels are read out in an order that does not reflect the geometrical sequence. A dual port memory is foreseen where data can be stored and read out in a different order.

- **Cluster encoding**

  The Level-1 trigger algorithm requires clusterized data. The following encoding scheme is proposed:

  Cluster of **one hit**: the cluster position is the strip position.

  Cluster of **two hits**: the cluster position is the first position; an extra bit signals a two strip cluster.

  Cluster of **three or more hits**: only two adjacent strips are taken to form a cluster. A cluster of four, e.g., is treated as 2 clusters of 2.

- **Cluster encapsulation**

  ODE encapsulates the data with a 32-bit header and trailer, respectively. Every data packet can be associated with an event by a 16-bit event ID which is monotonically increased by one. The interface between the ODE Level-1 preprocessor and the Level-1 trigger has been defined to be one S-Link [16] per ODE board. The cluster data from 8 preprocessors are combined in one S-Link. The S-Link width has been chosen to be 32-bit with a transfer frequency of 40 MHz. Event fragment length is strongly varying for different events. Therefore, an output FIFO buffers the data before it is transferred to the Level-1 . The maximum amount of hit clusters per ODE board is 127 with a hit cluster being 16 bits in size.

3.3 Readout Units (RUs) and Network

Figure 3.5 shows an abstract view of the LHCb Level-1 trigger.

After the hit cluster data has been preprocessed by the ODE as described above, it is sent to the input stage of the Level-1 trigger — the Readout Units. The number of RUs has been assumed to be about 25 in 1999. A detailed discussion will follow in chapter 6.

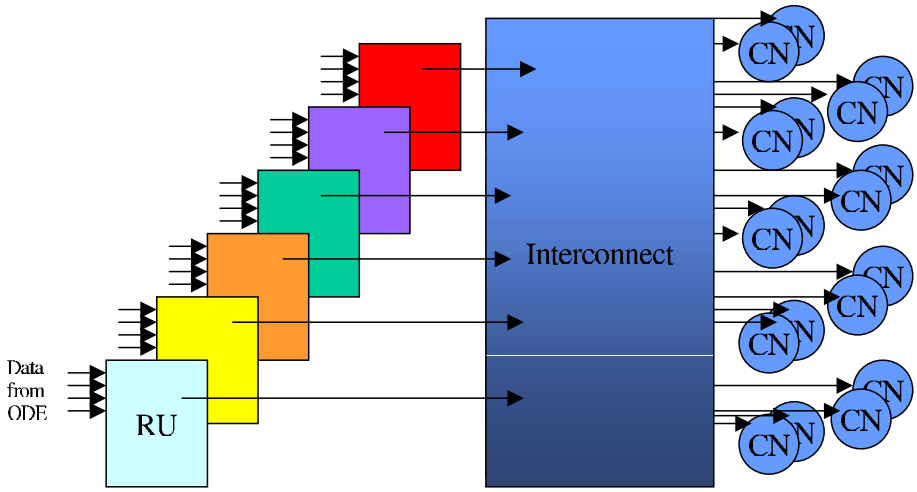

Figure 3.5: Abstract view of the LHCb Level-1 Trigger.

The input stage of the RU is equipped with FIFOs which buffer the incoming data from ODE. The FIFO size is variable between 4k and 128k × 32-bit. The data is merged to a so-called sub-event and stored in a sub-event buffer. Figure 3.6 shows a photograph of the RU [17] as implemented at CERN [18].

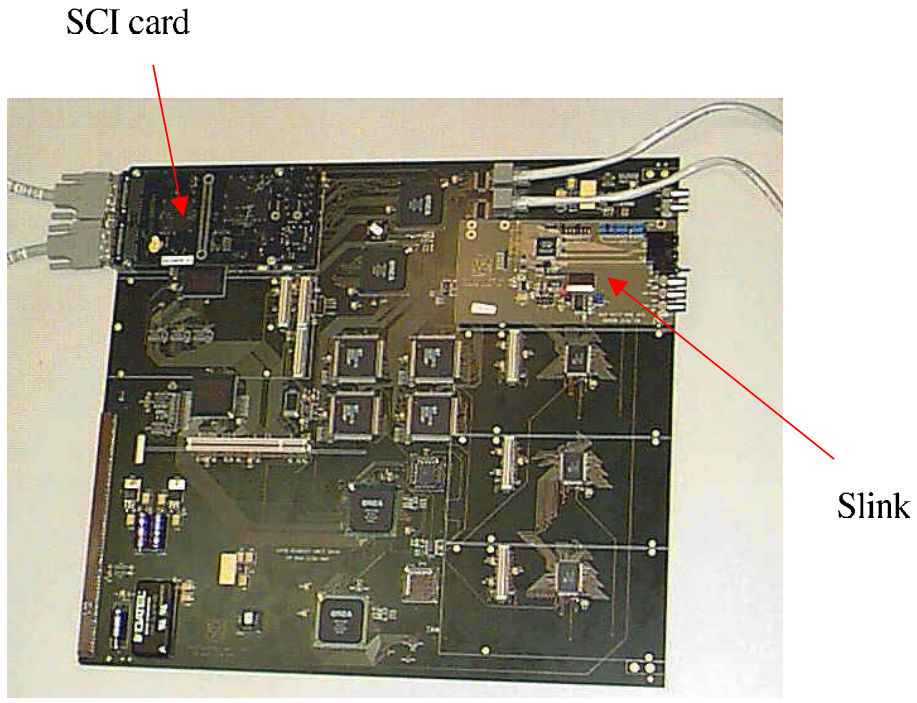

Figure 3.6: RU as implemented at CERN. The on-board processor is not shown. Figure from [19].

The RU forwards the sub-event to one of the compute nodes (CN) which is depicted in figure 3.5. To handle this task the RU is equipped with a 64-bit/ 66 MHz PCI bus, which is the interface to the network card, two FPGAs to implement any necessary logic, and an

on-board processor running an embedded Linux distribution. However, all data belonging to a specific event has to be processed by one compute node. The compute node has to be chosen right before event distribution and the sub-event data have to be orchestrated. The entire system is synchronized by a passive fiber optical clock distribution system (TTC) as mentioned in 2.4.

Since the RUs receive event fragments with a maximum frequency of 1.11 Mhz and the data belonging to one event is rather small[3] the network solution to the distribution problem has to copy data with a minimum of overhead. Although total event data size varies depending on present studies, it is certain that the overall network capacity will exceed 4 GByte/s. The CNs are COTS[4] PCs which are connected by a commercially available interconnect which plugs into the RUs PCI bus.

3.4 Track Finding Algorithm

Once all sub-events have been written into the memory of a compute node a software algorithm performs its pattern recognition task to do the track finding. The algorithm is looking at particle tracks traversing the VELO. If it finds indications for a secondary vertex the event could be an event containing B-physics.

How the final algorithm will look like is not sure at the time of writing. Different approaches are currently tested with simulated data. However, the following steps give an outline of the proposed algorithm:

- **2D Tracking**: B-mesons are typically produced below a polar angle of 200 mrad within the acceptance of the detector. The algorithm reconstructs all tracks in the rz plane. Thus, only R information is used at first. A track is defined as at least three hits on consecutive stations being on a straight line.
- **Primary Vertex Search**: The primary vertex which is the pp interaction point is determined by histogramming the 2D tracks.
- **3D Search**: Tracks with a significant impact parameter (IP) which is typically greater than 100 μm are chosen for 3D track reconstruction [20]. The IP is defined as the distance of the primary vertex to the track. Usually around 5 tracks are chosen.
- **Secondary Vertex**: The reconstructed 3D tracks are required to form a secondary vertex which is significantly separated from the primary vertex. The impact parameter information of those tracks is used to form a total B-event probability.

3.5 Trigger Output

The trigger produces a 128-bit output message for every event. It has been agreed on that the output of the Level-1 trigger must be in the same order as the input. The output

[3] It was assumed to be 200 Byte when I joined the project.
[4] commercial off-the-shelf

is sent to the Level-1 Decision Unit which produces a 40 kHz trigger signal. The trigger is distributed to the front-end by the TTC system. The exact content of the output message is not final yet. However, it will most likely contain the event ID and the B-event probability as computed by the track finding algorithm.

Chapter 4

LHCb Level-1 Trigger Networking

PCI is an I/O bus that can be found in any commodity PC. The Level-1 RUs and the CNs use PCI as an interface to the network. PCI is a synchronous bus with a peak bandwidth of 528 MByte/s in its fastest implementation. PCI performance of the sending node is an important issue since every microsecond a sub-event has to be transferred to the NIC. There are several methods of sending data to a remote node. A shared memory approach is chosen to transfer data since no additional overhead is involved. Therefore, movement of data is totally transparent.

PIO and DMA are used to perform I/O. PIO means that the CPU writes data whereas in DMA mode the data is fetched from memory by the NIC. However, a DMA controller has to be set up first which means an additional bandwidth requirement.

The Scalable Coherent Interface offers a hardware-based distributed shared memory solution. One implementation offers peak SCI link speeds up to 667 MByte/s and a sustained PCI bandwidth of 300 MByte/s.

4.1 LHCb Networking Requirements

The requirements to the LHCb Level-1 interconnect have been introduced in chapter 3. They can be summarized as follows:

- **PCI Interface**

  The interface between the interconnect and both the RU and the CNs is PCI as mentioned in 3.3. PCI is a local bus that can be found in any commodity PC which means minimization of cost. A requirement set on the interface is both availability at present for hands-on tests and in the near future when the final system will be built.

- **High-speed, low latency interconnect**

  The Level-1 trigger interconnect has to provide a total bandwidth of several GByte/s. In addition, the Level-1 network latency is part of the total Level-1 latency path.

- **Minimized overhead**

  The Level-1 interconnect has to satisfy the MHz requirement. This means that every microsecond a new transaction to a specific target has to be initiated. Additional overhead means additional traffic on both the PCI bus and the network links.

- **Scalability**

  The dimension of the network is not final. The peak input rate of the LHCb Level-1 trigger is 1.11 MHz with the average being 1 MHz. The number of input feeds is not final yet since it depends strongly on the detector geometry which is undergoing changes at the time of writing. The LHCb network must provide a bandwidth of 4 GByte/s independent of the number of input feeds and number of compute nodes. If the bandwidth requirement can not be met data can not be delivered.

- **Availability and Cost**

  The network should be available as of today since a prototype must be built. Network cost should be minimized since every large scale project like LHCb has to set a budget on its expenses.

4.2 The Peripheral Component Interconnect (PCI)

4.2.1 PCI Overview

The PCI Local Bus is a high performance bus for interconnecting chips, expansion boards, and processor/memory subsystems. It has been developed at Intel in the early 1990s as a standard method of interconnecting chips on a board. Later the PCI Special Interest Group, or "PCI SIG" has adopted it as an industry standard. Under the PCI SIG the definition of PCI has been extended to define a standard expansion bus interface connector for add-in boards.

First use of PCI in personal computers has been made in 1994 with introduction of the "Saturn" chipset and "Alfredo" motherboard for the 486 processor by Intel. With introduction of chipsets and motherboards for the Intel Pentium processor, PCI started to replace earlier bus architectures such as ISA, EISA, VL[1], and Micro Channel[2]. Both ISA[3] and EISA[4] expansion buses are synchronized by an 8 MHz bus clock signal with a bus width of 16-bit and 32-bit, respectively. The ISA bus has initially continued to coexist with PCI for support of "legacy" add-in boards that do not require the high performance of the PCI bus. But as legacy boards are redesigned, PCI is expected to completely replace ISA as well.

In 1998 the PCI SIG announced that Compaq, Hewlett-Packard, and IBM had submitted a new specification for review called "PCI-X". The new standard defines an increased PCI bus speed up to 133 MHz. It also includes changes in the PCI communications protocol

[1] VESA LocalBus

[2] Introduced by IBM. Micro Channel has been used as the primary expansion bus used in IBM's Personal System/2 (PS/2) and RS/6000 computers over the period from 1987 to 1995.

[3] Industry Standard Architecture

[4] Extended Industry Standard Architecture

affecting data transfer rates and electrical timing requirements. For details on PCI-X see [21].

4.2.2 PCI Bus Protocol

PCI is a synchronous bus architecture with all data transfers being performed relative to a system clock (CLK). The initial PCI specification permitted a maximum clock rate of 33 MHz allowing one bus transfer to be performed every 30 ns. Later, Revision 2.1 of the PCI specification extended the bus definition to support operations at 66 MHz. The vast majority of today's personal computers continue to implement a PCI bus that runs at a maximum speed of 33 MHz. However, high-performance boards as provided by Tyan[22] or SuperMicro [23] are built around high-end chipsets which offer fast 64-bit/66 MHz PCI buses for PC systems. Figure 4.1 shows a simplified view of a modern chipset architecture. The chipset provides an interface to the system bus, a memory controller, and interfaces to busses like PCI or AGP.

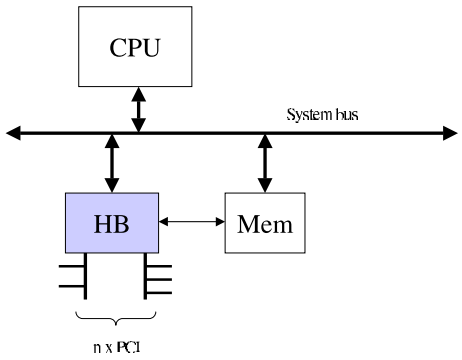

Figure 4.1: Modern chipset architecture with the host bridge (HB) being the external I/O controller.

PCI implements a 64-bit multiplexed Address and Data bus (AD[63:0]). The following details are based on 64-bit PCI buses since they have been used in the setups presented. However, 64-bit connector cards can be used in 32-bit slots since the wider bus is fully backwards compatible. At 33 MHz, a 64-bit slot supports a peak data transfer rate of 264 MByte/s, and a 66 MHz slot up to 528 MByte/s. These values have only academic character as presented in upcoming chapters. In real-life performance numbers like these can never be achieved.

PCI defines support for both 5 Volt and 3.3 Volt signaling levels. A "keying" scheme is implemented in the PCI connectors to prevent inserting an add-in board into a system with incompatible supply voltage. Although used most extensively in PC compatible systems, the PCI bus architecture is processor independent. PCI signal definitions are generic allowing the bus to be used in systems based on other processor families. PCI includes strict specifications to ensure the signal quality required for operation at 33 and 66 MHz. Components and add-in boards must include unique bus drivers that are specifically designed for use in a PCI bus environment. This restriction along with the high bus speed dictates that most PCI devices are implemented as custom ASICs. However, current FPGAs[5] offer solutions which are not based on ASICs anymore. For a brief discussion see

[5]Field Programmable Gate Array

5.2. The higher speed of PCI limits the number of expansion slots on a single bus to no more than 3 or 4 for 33 MHz buses and 2 slots for 66 MHz architectures.

To permit expansion buses with more slots, the PCI SIG has defined a PCI-to-PCI Bridge mechanism. PCI-to-PCI Bridges are ASICs that electrically isolate two PCI buses while allowing bus transfers to be forwarded from one bus to another. Each bridge device has a "primary" PCI bus and a "secondary" PCI bus. Multiple bridge devices may be cascaded to create a system with many PCI buses. However, modern chipsets are mostly composed of a host bridge with multiple independent PCI buses originating from it as depicted in figure 4.1.

There are three PCI address spaces which are supported. The PCI Configuration Address Space is mandatory to configure the device. I/O Address Space and Memory Address Space are customary.

- **Configuration Space**

  PCI supports an auto configuration mechanism. Each PCI device possesses a 256 Byte block which is called PCI Configuration Address Space. However, only the first 64 Byte are predefined by the PCI specification — the Configuration Header Region. The PCI configuration space contains entries which specify the device. Other registers allow configuration of the device's I/O addresses, memory addresses, interrupt levels, etc. The device is configured by configuration transactions which is done by the OS and the BIOS on startup.

- **I/O Space**

  Intel x86 processors do have a separate I/O space next to their memory space. The PCI I/O space provides the capability to read and write from I/O addresses. I/O address ranges have to be configured at startup time. However, I/O regions can not be mapped to the virtual address space of a process. Any access to I/O space requires a system call which implies additional latency.

- **Memory Space**

  PCI memory space is mapped into the physical address space of the processor if required by the device. Figure 4.2 shows how a 32-bit physical address space is seen by the processor. PCI memory address regions can be accessed by load and store operations of the processor. A PCI memory address region can be mapped to the virtual address space of a process.

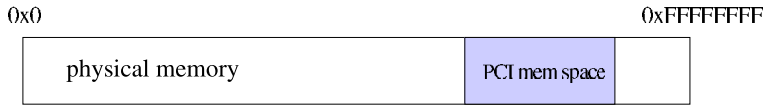

Figure 4.2: Example physical address space.

The following printout lists two PCI devices including their address spaces on a desktop computer. The information can be retrieved by reading /proc/pci on a Linux system:

```
Bus  0, device   8, function  0:
  Class 0200: PCI device 1113:1211 (rev 16).
    IRQ 11.
    Master Capable.  Latency=64.  Min Gnt=32.Max Lat=64.
    I/O at 0xe400 [0xe4ff].
    Non-prefetchable 32 bit memory at 0xe8000000 [0xe80000ff].
Bus  1, device   0, function  0:
  Class 0300: PCI device 1002:4742 (rev 92).
    IRQ 11.
    Master Capable.  Latency=64.  Min Gnt=8.
    Non-prefetchable 32 bit memory at 0xe4000000 [0xe4ffffff].
    I/O at 0xd000 [0xd0ff].
    Non-prefetchable 32 bit memory at 0xe6000000 [0xe6000fff].
```

As shown by the printout, PCI devices are listed by their location which is specified by bus number, device, and function. Devices which have different bus numbers are either separated by a PCI bridge or the host bridge. Some contents of the PCI Configuration Space, e.g. the master capability, are displayed as well.

Bus Arbitration

A bus which is shared by many devices needs an arbiter that decides which device is allowed to drive the bus next. A PCI device which is able to demand bus ownership is said to have bus master capability. However, if a device is able to drive the bus is determined by its implementation. Figure 4.3 shows two transactions and the REQ# and GNT# signals which are used for bus arbitration.

The bus master, which initiates a transaction, arbitrates for bus ownership by asserting a REQ# signal to a central arbiter. The arbiter grants ownership of the bus by asserting the GNT# signal. REQ# and GNT# are unique on a per slot basis allowing the arbiter to implement a bus fairness algorithm. Arbitration in PCI is hidden in the sense that it does not consume clock cycles. The current initiator's bus transfers are overlapped with the arbitration process that determines the next owner of the bus.

The timing diagram in figure 4.3 can be analyzed as follows:

- Cycle 1 - Master#1 has already asserted REQ#1 asking for the bus.
- Cycle 2 - GNT#1 is asserted as response to REQ#1. In addition Master#2 is asking for the bus by asserting REQ#2.
- Cycle 3 - A data transaction is initiated. For more details regarding the signals see 18.
- Cycle 4 - Data transaction. Master#2 is still asking for the bus and receives a GNT# from the arbiter.
- Cycle 5 - Master#1 has finished its transaction.
- Cycle 6 - REQ#2 is deasserted and Master#2 is initiating a data transfer.

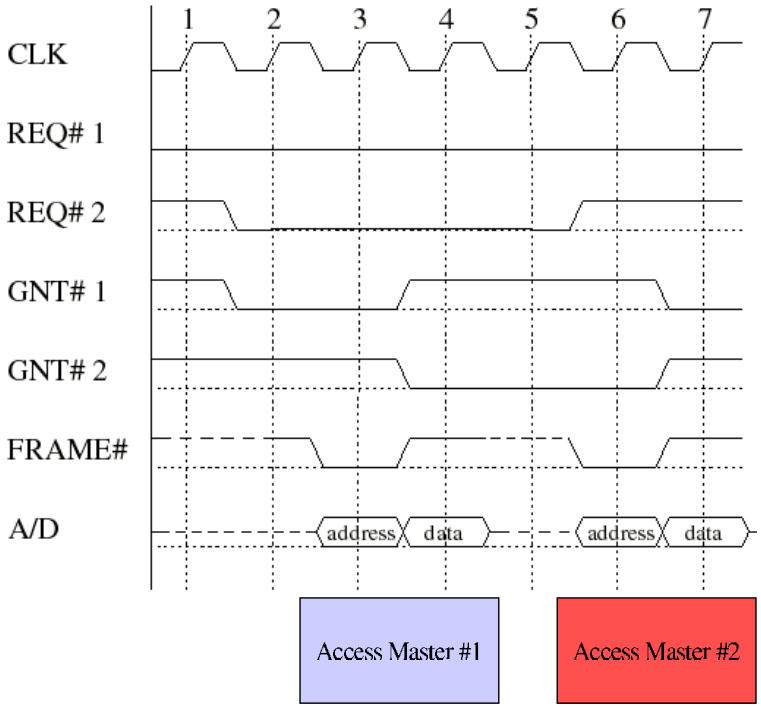

Figure 4.3: PCI bus arbitration. Figure from [24].

- Cycle 7 - REQ#1 has been asserted all the time. However, since the arbiter is fair Master#2 has been able to transfer data.

Data Transfer

The multiplexed Address and Data bus allows a reduced pin count on the PCI connector that enables lower cost and smaller package size for PCI components. PCI bus cycles are initiated by driving an address onto the AD[63:0] signals during the first clock edge called the address phase. The address phase is signaled by the activation of the FRAME# signal and the C/BE[3:0]# signals. The latter signals the type of transfer (memory read, memory write, I/O read, I/O write, etc.). Although it is not widely implemented, PCI supports 64-bit addressing. Unlike the 64-bit data bus option which requires a longer connector with an additional 32-bits of data signals, 64-bit addressing can be supported through the base 32-bit connector. Dual Address Cycles are issued in which the low order 32-bits of the address are driven onto the AD[31:0] signals during the first address phase, and the high order 32-bits of the address (if non-zero) are driven onto the AD[31:0] signals during a second address phase. The remainder of the transfer continues like a normal bus transfer.

A PCI bus transfer consists of one address phase and any number of data cycles. I/O operations that access registers within PCI targets have only a single data phase. Memory transfers that move blocks of data usually consist of multiple data cycles that read or write multiple consecutive memory locations. A transfer like that is called a burst transaction. Burst transactions make best use of the bus resource. During the data phase the C/BE[3:0]# signals serve as byte enable to indicate which data bytes are valid. Both the

initiator and target may insert wait states into the data transfer by deasserting the IRDY# and TRDY# signals. Valid data transfers occur on each clock edge in which both IRDY# and TRDY# are asserted. Both the initiator and target may terminate a bus transfer sequence at any time. The initiator signals completion of the bus transfer by deasserting the FRAME# signal during the last data phase. A target may terminate a bus transfer by asserting the STOP# signal. When the initiator detects an active STOP# signal, it must terminate the current bus transfer and re-arbitrate for the bus before continuing. If STOP# is asserted without any data cycle completing, the target has issued a retry. If STOP# is asserted after one or more data cycles have successfully completed, the target has issued a disconnect.

4.2.3 PCI Timing Diagram

The following timing diagram illustrates a write transaction on the PCI bus:

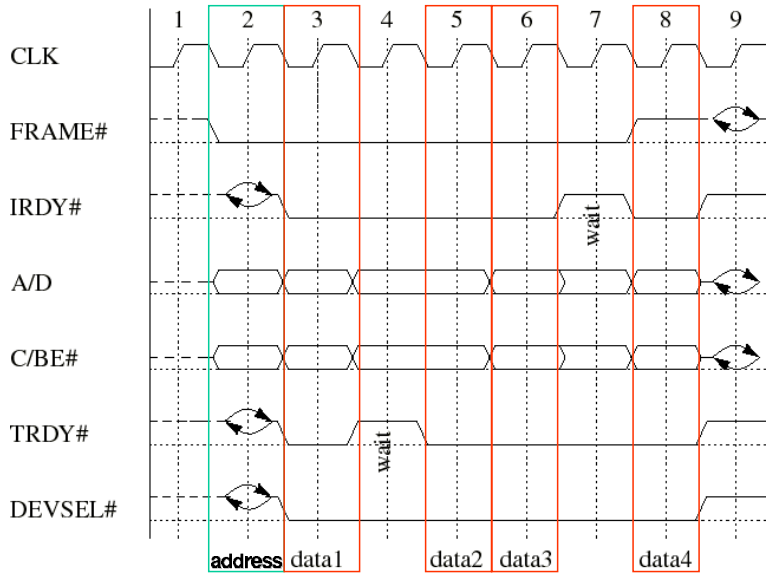

Figure 4.4: PCI write transaction. Figure from [24].

- Cycle 1 - The bus is idle.
- Cycle 2 - The initiator asserts a valid address and places a write command on the C/BE# signals. This is the address phase. Turnaround cycles can be seen for IRDY#, TRDY#, and DEVSEL#. Turnaround cycles are inserted when one agent stops driving the bus and a different agent starts to drive the bus.
- Cycle 3 - The initiator drives valid write data and byte enable signals. The initiator asserts IRDY# low indicating valid write data is available. The target asserts DEVSEL# low as an acknowledgment it has positively decoded the address (the target may not assert TRDY# before DEVSEL#). The target drives TRDY# low indicating it is ready to capture data. The first data phase occurs as both IRDY# and TRDY# are low. The target captures the write data.

- Cycle 4 - The target deasserts TRDY# indicating it is not ready to capture the next data.

- Cycle 5 - The initiator provides the same data and byte enables as in cycle 4. This time both IRDY# and TRDY# are low and thus the data phase valid. The target captures the write data.

- Cycle 6 - The initiator provides new data and byte enables. The second data phase occurs as both IRDY# and TRDY# are low. The target captures the write data.

- Cycle 7 - The initiator deasserts IRDY# indicating it is not ready to provide the next data.

- Cycle 8 - The initiator provides new data and byte enables. The target captures the write data. Since this is the last data phase FRAME# is deasserted.

- Cycle 9 - The transaction has been finished. IRDY#, TRDY#, and DEVSEL# are deasserted.

4.2.4 Interrupts

Any interrupt capable PCI device can generate interrupt requests to request servicing from the system software. After the system startup configuration software executes, the system initialization begins. The system or the operating system must provide a device-specific interrupt service routine for each interrupt driven device in the system. Furthermore, the operating system must build an interrupt jump table in memory. Each entry in the interrupt table must contain the start address of a device-specific interrupt service routine associated with a particular device. Most interrupt capable devices come with a loadable kernel module which is loaded during startup of the operating system. The operating system calls the initialization code within the driver, which probes the bus for the card. The driver contains the device's interrupt handler and is responsible for placing the start address of the handler into the proper entry of the interrupt table. Every time an interrupt request occurs the processor interrupts its current task, pushes the contents of specific registers into stack memory, and executes the interrupt handler which has been associated with the interrupt issued. Thus, every time an interrupt occurs the current process is suspended and the interrupt handler code is executed. For more information on PCI interrupt-related issues see [25].

4.3 High-speed, Low Latency Interconnects

The LHCb Level-1 trigger requires a high-speed low latency network card which is available for hands-on experience and prototyping. A second important fact is the cost of the network since the LHCb budget does not foresee equipment in the upper price range. The following subsections point out important boundary conditions with respect to the Level-1 trigger interconnect.

4.3.1 Scalability

Scalability is a property which describes how well a network is suitable for small and large size networks. Very often a small system is set up and increased in size if more compute power is needed. However, the probability of saturation for some network paths increases the more sending nodes participate in network traffic. The parts of the network where saturation effects occur are called hot spots.

A nearly ideal network will scale linearly in total bandwidth the more network nodes are added with the latency staying almost the same. An example of a none-scalable interconnect is a bus — the maximum bandwidth which is shared among all participants prevents a scalable system. Latency in a torus topology will increase since the distance between the nodes also increases. However, a switch could help to compensate that effect. Switches have the disadvantage that they get expensive for large cluster solutions. In addition, switches create a central hot spot. Any network traffic has to traverse the switch. Therefore, the traffic pattern observed on the output depends highly on the intrinsic properties of the device.

Scalability is a crucial part of the LHCb interconnect since it has high bandwidth requirements and must have a latency which can be predicted.

4.3.2 Input/Output Models

There are three main ways to perform I/O for a PCI device:

- **Programmed I/O**
- **Interrupt-driven I/O**
- **DMA-driven I/O**

Programmed I/O (PIO)

The CPU explicitly sends and receives data to and from the device. Therefore, CPU cycles have to be used to perform I/O which is usually avoided since the overhead for the CPU is too significant. However, PIO is an attractive way to transfer data in embedded systems where no other work has to be done while an I/O device is being used. Figure 4.5 shows the data path for programmed I/O.

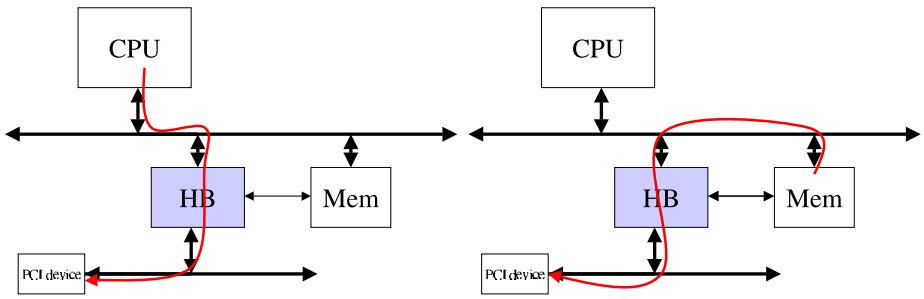

Figure 4.5: Left: Data path for programmed I/O. Right: Data path for DMA.

Interrupt-driven I/O

The CPU initiates an operation with a device. The user process that has initiated the request is blocked and some other process runs. When the device has finished the task an interrupt is generated. The interrupt handler either makes the user process runnable if all I/O has been completed or it schedules the next I/O request if there is more work to be done. This transfer scheme enables the CPU to process other jobs while I/O is taking place. The interrupt response time can be up to $20\mu s$ and depends heavily on the architecture. However, in case of an Intel based machine running standard Linux the interrupt latency for a PCI interrupt handled by the Dolphin SCI driver has been measured to be $8.4\mu s$. Additionally, interrupts cause performance loss of other tasks.

DMA-driven I/O

The term DMA stands for Direct Memory Access and requires hardware support in the form of a DMA controller on the PCI device. The DMA capable device can access the main memory of the system. Figure 4.5 shows the path of the data for DMA-driven I/O. Before a DMA transfer can be issued the DMA engine has to be programmed. The control registers of the DMA engine typically include an address, a length, and commands like read or write. Depending on the DMA engine the control registers offer space for one or multiple transfers. A setup where the DMA engine is programmed to transfer multiple independent data blocks is called chained mode DMA.

The key feature of DMA is the fact that the I/O device can directly access memory, and therefore transfer data between controllers and memory without CPU intervention. However, the DMA engine has to be set up in an initialization phase which is not efficient for small block sizes. DMA controllers use physical addresses. Thus, when programming a DMA controller from a user space program the virtual address of a buffer must be translated to its physical counterpart. This is done by some underlying software like a device driver. The memory pages used by DMA controllers have to be pinned to memory to avoid the they are swapped during a DMA operation.

4.3.3 Protocol Overhead

The LHCb data flow does foresee that a sub-event has to be transferred by the network card (see 3.3) every microsecond. Since it has been agreed on that data should traverse the PCI bus of the RU at first it can be calculated that one data transfer is allowed to utilize 66 clock cycles assuming a 64-bit/66 MHz PCI bus. Assuming PCI bursts such that there is no wait cycle in between PCI data cycles and assuming furthermore an average packet size of 128 Byte a maximum overhead of 50 clock cycles can be calculated.

The LHCb Level-1 trigger should transfer data using as little overhead as possible. There will be some overhead on the network link concerning routing, reliability of transfers, and error detection.

Reliability of data is a crucial issue. Corrupted or lost messages can not be tolerated. In traditional LANs some protocol software took care of reliability of data transfers. Data has been kept in a buffer, messages have been acknowledged, and corrupted or lost data

has been retransmitted if necessary. However, this must not be done in software since a software protocol stack can not be executed in the sub-microsecond range.

Figures 4.6 and 4.7 show PCI bus scenarios as observed when using PIO and DMA, respectively.

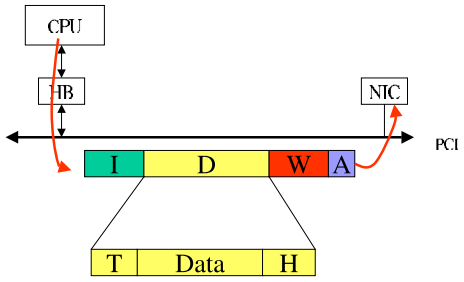

Figure 4.6: Overhead PIO mode. The CPU is sending data to the NIC.

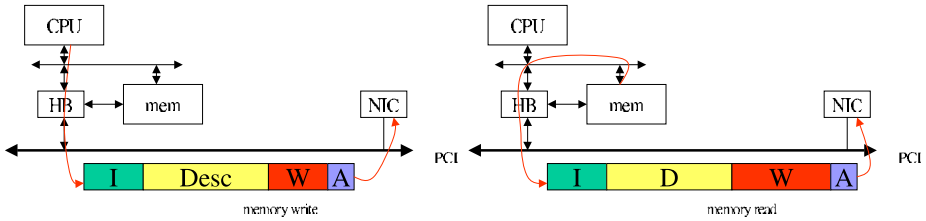

Figure 4.7: Overhead DMA mode. Left: Setup of the DMA controller. The CPU is sending a DMA descriptor to the NIC. Right: The NIC fetches data out of memory which implies more wait cycles on the PCI bus than in case of writing to memory.

The scenario in figure 4.6 shows a PCI transaction where data is pushed to the network card by PIO. Concerning overhead two kinds can be observed and must be distinguished. The PCI bus protocol implies little overhead. The figure shows an address phase (A), some wait cycles (W) and one idle cycle (I). The idle cycle in between data transfers is not necessary if the same agent issues back-to-back transfers. PCI overhead of less than 10 clock cycles would be extremely fast assuming a fast response time of the PCI target and transactions being issued almost back-to-back by the chipset. This is certainly hard to achieve with common chipset implementations. The figure also shows that the data payload on the bus can have overhead as well. This is depicted by a possible trailer (T) and header (H). However, if T and H could be minimized or even vanish this scenario could be a solution to the LHCb transfer problem if nearly back-to-back transfers were possible.

Figure 4.7 shows the PCI bus scenario when data is transferred using DMA. As depicted there are two transfers that have to be initiated. The first transfer is the setup of the DMA engine. A descriptor is pushed into the network card at first. After a period of time depending on the network card a PCI memory read transfer is fetching the data out of the memory. The advantage of DMA is certainly visible for large block sizes (e.g. > 512 Byte) but rather a throttle for small messages as the Level-1 trigger is concerned with.

4.3.4 Memory Models

There are two memory models supported when talking about cluster interconnects; the shared memory and the message passing (distributed memory) model. In a shared memory system accesses to remote locations are transparent since a common global address space is used. Therefore, virtual addresses have to be translated to the physical address counterpart. If the memory page that has to be accessed reflects a remote location it is mapped to the local network card. It looks up the destination node in its internal address translation tables and transfers the data to the destination. In a shared memory environment load and store operations can be done without any software overhead. Figure 4.8 shows the data flow in a shared memory environment assuming that the NIC (Network Interface Controller) is located on the PCI bus of the system.

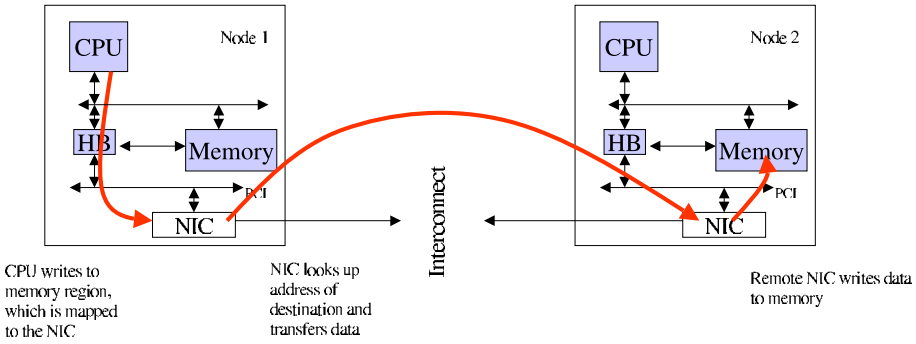

Figure 4.8: Write operation to a remote memory region.

In a message passing system the processors communicate by sending messages like send and receive API calls. However, this usually implies overhead since at least the receiver of the data has to be specified. An example given is the TCP header. The 192-bit TCP header specifies source and destination port amongst other things. To transfer the TCP header on the PCI bus three clock cycles have to be used in addition.

4.4 The Scalable Coherent Interface (SCI)

The IEEE Standard for Scalable Coherent Interface [26] provides computer bus-like services. However, instead of a bus, it uses a collection of fast point-to-point unidirectional links to provide the high bandwidth needed for high-performance multiprocessor systems. SCI supports distributed shared memory with optional cache coherence scalable up to 64k nodes. Like other system area network (SAN) solutions as Myrinet [27], the primary objective of SCI is to provide high bandwidth in the Gbit/s range, low latency in the microsecond range, and low CPU overhead for communication operations.

SCI has been designed to address scalability in many respects. Some examples are presented:

- Scalability of **performance** - the aggregate bandwidth grows with increasing number of system nodes;

- Scalability of **interconnect distance** - SCI networks offer link distances from centimeters to hundreds of meters depending on the link media and physical layer implementation.
- Scalability of **memory systems** - the number of processors supported is not limited. This is especially important when cache coherency is implemented.
- The **addressing capability** is limited to support 64k nodes which by far exceeds any existing compute cluster.

4.4.1 Concepts of SCI

- The SCI interconnect uses unidirectional **point-to-point links**. The number of links grows as nodes are added to the system. Every additional node provides its own elasticity buffers increasing the number of buffers in the system. Therefore, the aggregate bandwidth scales if the traffic pattern is chosen such that link segments do not get saturated. The links allow concurrent data transfers.
- The **signals** that are transmitted are low voltage differential signals. An LVDS standard for SCI [28] has been developed that defines low-voltage differential signals with a voltage swing of 250 mV.
- SCI has been designed to connect up to 64k **nodes**. A node can either be a workstation, a server, a single processor with its cache, a memory module, I/O controllers and devices, or bridges to other interconnects or buses.
- **topology independence**; SCI allows a variety of different topologies. Some are shown in figure 4.9

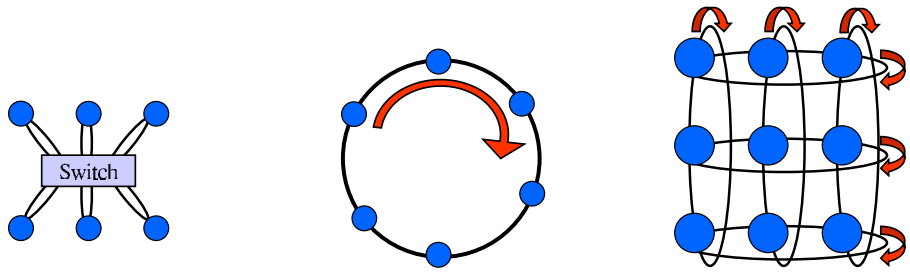

Figure 4.9: SCI topologies.

The basic SCI topology is a ringlet. However, both switched and multi-dimensional tori are feasible.

- **Fixed addressing scheme**. SCI uses a 64-bit fixed addressing scheme, which has been defined by the Control and Status Register (CSR) standard [29]. The 64-bit SCI address is divided into two parts. The most significant 16 bits specify the node ID, the remaining 48 address bits are used for addressing within the nodes.
- SCI utilizes a 64-bit **hardware-based distributed shared memory (DSM)**. The distribution of the memory is transparent to software and processors. A remote memory access initiated by the processor is handled by the SCI hardware. A processor

can invoke remote memory operations by load and store operations. The operating system does not need to be involved in transactions since memory operations can be executed from user-level. This results in latencies in the low microsecond range.

- **Bus-like services**. SCI defines transactions to read, write, and lock memory regions. This functionality is well known from computer buses. Additionally message passing and global time synchronization are supported. Interrupts can be delivered remotely and broadcasts are foreseen. Transactions can be tagged with four different priorities. Bandwidth allocation has been developed to assign transfer bandwidth to nodes that are willing to send. This can be compared to bus arbitration. A queue allocation protocol has been developed to facilitate traffic to nodes, which are heavily loaded.

- **Split transactions**. SCI splits its transactions into request and response phase. Therefore, link utilization increases since multiple outstanding transactions per node are possible. Thus, transactions can be pumped into a network with a very high rate using the interconnect in a pipelined fashion.

- SCI defines an **optional cache coherence** protocol, which is based on a distributed-directory approach. However, the cache coherency protocol is highly sophisticated and complex. Therefore, three different subsets of cache coherency have been defined — the minimal set, a typical set, and the full set. SCI network controllers attached to an I/O bus like PCI can not participate in cache coherency actions since local memory access can not be snooped. However, cache coherence is not needed for LHCb since data is not shared between nodes.

- **Reliability in hardware**. Error detection based on a 16-bit CRC[6] code is done in hardware. Transactions and hardware protocols are provided that allow a sender to detect failure due to packet corruption. A receiver can inform a sender that it is not capable of accepting further packets. Since actions are taken on a per packet basis SCI does not guarantee in-order delivery of packets.

- **Layered specification**. Three different layers are foreseen by the SCI standard: the physical layer, the logical layer, and the optional cache coherency layer. The logical layer is described in more detail in 4.4.2. Three physical layer models have been described in the standard. One parallel electrical link has been foreseen to cover short distances of a few meters. Two serial links have been specified to work for longer distances.

- **C code**. The basic concepts of SCI are distributed as C code. The major reason for this approach is the fact that C is unambiguous and that the specification can executed as a simulation. Packet formats and the physical layer specification are not distributed as C code.

4.4.2 SCI Logical Layer

The logical layer describes types of transactions, packet types and formats, packet encodings, the standard node interface structure, bandwidth and queue allocation protocols,

[6] Cyclic Redundancy Check

error processing, addressing and initialization issues, and SCI-specific CSRs.

Transactions

Transactions are split and are made of a request and a response sub-action. Therefore, the nodes involved are called the requester and the responder. A transaction is initiated by the requester, which sends a packet to the responder. Subsequently a response packet is sent from the responder to the requester. However, this only applies for transactions with response sub-action. Packets carry addresses, command and status information, and data. The latter depends on the type of transaction. One node can have up to 64 outstanding transactions.

Both request and response sub-action consist of a send packet which is generated by the sender and an echo packet returned by the receiver. Figure 4.10 shows the different phases of a transaction.

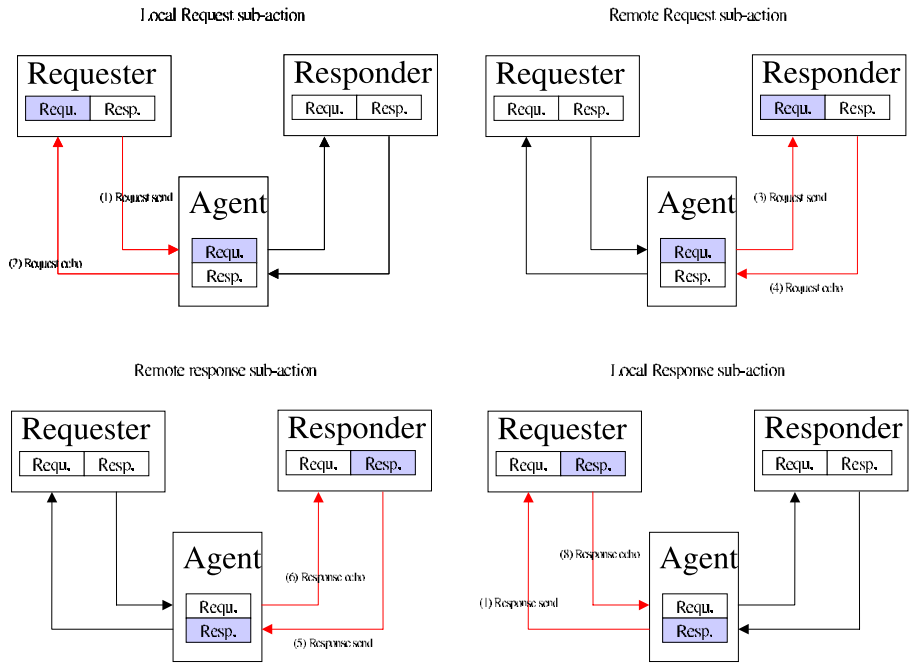

Figure 4.10: SCI transaction phases.

The echo packet tells the sender if a packet has been accepted properly or rejected. If a packet has been accepted it is stored in the receivers input queue. In this case the sending node can discard its send packet. If a packet can not be accepted the echo packet will cause retransmission of the send packet. The latter can occur if the input queue of the receiving node is filled.

A consequence is that SCI can not guarantee in-order delivery of send packets. A packet which is rejected can be overtaken by a later packet, which finds an empty buffer slot.

Figure 4.10 also shows an intermediate node labeled agent. Agents could be bridges or switches coupling two or more SCI ringlets. An agent is responsible for the echo sub-action. If the agent takes a send packet off a ringlet to forward it to the target ring it has the responsibility to acknowledge this by sending an echo packet back to the sender.

Therefore, echo packets are not end-to-end confirmations but local ring acknowledgments. End-to-end confirmation is provided by response sub-actions.

Transaction Types

Three transaction types can be distinguished:

- transactions with responses (**read**,**write**, and **lock** transactions)
- **move** transactions
- **event** transactions

Figure 4.11 shows an overview of possible transaction types.

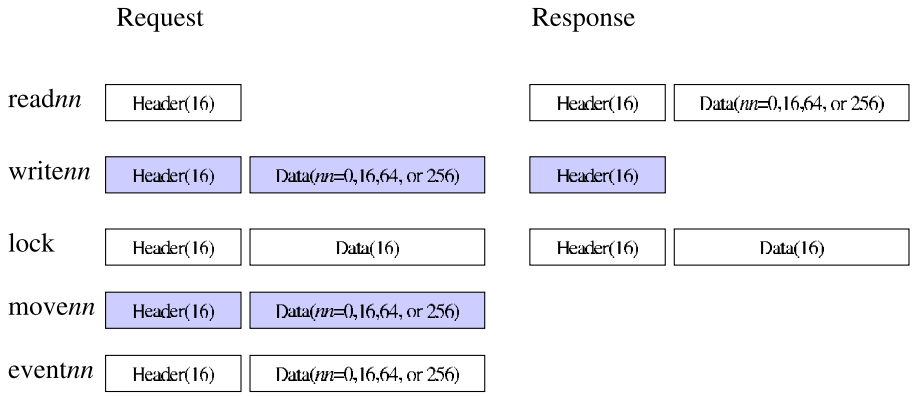

Figure 4.11: Transaction types.

Read transactions copy data from the responder to the requester with 0, 16, 64, or 256 Byte being transferred. In a 16 Byte transaction (readsb; selected-byte read) the size of the transferred data can differ between 1 and 16 Byte. 0 Byte is used for cache coherency purposes and not discussed here.

Write transactions transfer data from the requester to the responder. Possible data sizes can be seen in figure 4.11. The 16 Byte write transaction (writesb; selected-byte write) has variable data size.

The lock transaction is an atomic operation that copies data to a memory location and delivers the original value back. The new value is transferred in the request packet whereas the old value is returned in the response packet.

Move transactions are used to copy data to a memory location when no response sub-action is desired. This could be because of real-time applications where timeliness is more important than guaranteed delivery. However, flow control is still present since echo packets are in use. Two different kinds of move transactions are specified; direct move (dmove) and broadcast moves. However, broadcasts are specified as options.

Event transactions lack both flow control and response sub-action. They are intended for special purposes like delivering a time stamp for global time synchronization.

Packets

There are four basic packet types: request send, request echo, response send, and response echo packets. During the initialization phase and during re-synchronization packets like init and sync are in use also. Figure 4.12 shoes the format of a request send packet.

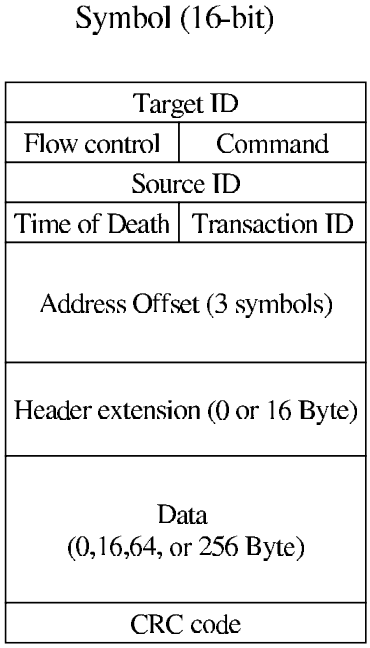

Figure 4.12: Format of an SCI request send packet.

SCI packets are made of a sequence of 16-bit symbols. The packet header usually comprises seven symbols (14 Byte) and the trailer (CRC) one symbol. The meaning of the different fields of the packet is as follows:

- The target ID holds the address of the destination node. If a node receives a packet the target ID is inspected and a decision is made whether the packet is taken off the ring or forwarded to the output link.
- The flow control field contains information for the bandwidth allocation. The command field specifies the type of the request transaction.
- The source ID specifies the address of the originator of the packet.
- Time of death specifies when the packet has to be discarded. The 6-bit transaction ID allows in conjunction with the source ID 64 outstanding transactions per node.
- Three symbols are foreseen to carry address offsets at the responder.
- Header extensions applies to some cache coherency transactions only.
- The packet can carry either 0,16,64,or 256 Byte of data.
- The 16-bit CRC code protects the packet content excluding flow control information.

Compared to the request packet introduced, response send packets carry status information instead of the address offset. Echo packets are four symbols in size whereas the special packets for initialization and control are eight symbols long.

Packet Encoding

The 16-bit symbols are the basic units for packet encoding. However, two other signals are needed. The clock signal determines symbol boundaries whereas the flag signal marks start and stop of packets. In order to enable SCI links to run continuously and at high speeds, the space between packets is filled with idle symbols. Idle symbols serve two purposes:

- they allow SCI nodes to synchronize the incoming data stream to the local clock.
- they transfer allocation and other network control information

At least one idle packet has to be transferred on a link in between regular packets. Only special packets may be transported back-to-back. Idle symbols are created whenever a node takes a packet off the ring and are replaced when a node inserts a packet. Control information stored in the idle symbols has to be reinserted properly such that the allocation protocol can work properly.

Allocation Protocols

Two allocation protocols are foreseen by the SCI standard. The queue allocation protocol ensures that the input queue of the receiver reserves some space such that retransmitted packets get eventually accepted. The bandwidth allocation protocol eventually guarantees some bandwidth to a sending node such that it can send its packets. However, a more detailed discussion on those topics will be provided in 4.4.3 when the Dolphin implementation is discussed.

The Scrubber

Corrupted packets having a wrong target ID, e.g., will not be taken off the ringlet by any node. Therefore, packets would circulate forever and waste bandwidth. However, the SCI protocol does foresee a special node to take care of this and other tasks — the scrubber. It monitors ringlet activity and is responsible for maintaining network functions like deletion of corrupted or stale packets and idle symbols. It also handles packets with addressing errors.

4.4.3 The Dolphin Implementation

Link Controller 3

Dolphin Interconnect [30] has released its PCI 64/66 SCI Adapter based on the SCI link chip LC3 and the PCI bridge chip PSB66. The card promises SCI link speeds of

667 MByte/s, a sustained PCI bandwidth of more than 300 MByte/s, and an application-to-application latency less than ten microseconds. This section focuses on the Dolphin implementation and points out some implementation details.

The SCI link chip LC3 provides high speed SCI links, sends and receives SCI packets, and provides an interface to Dolphin's back interface B-link. Figure 4.13 shows a block diagram of the LC3 chip.

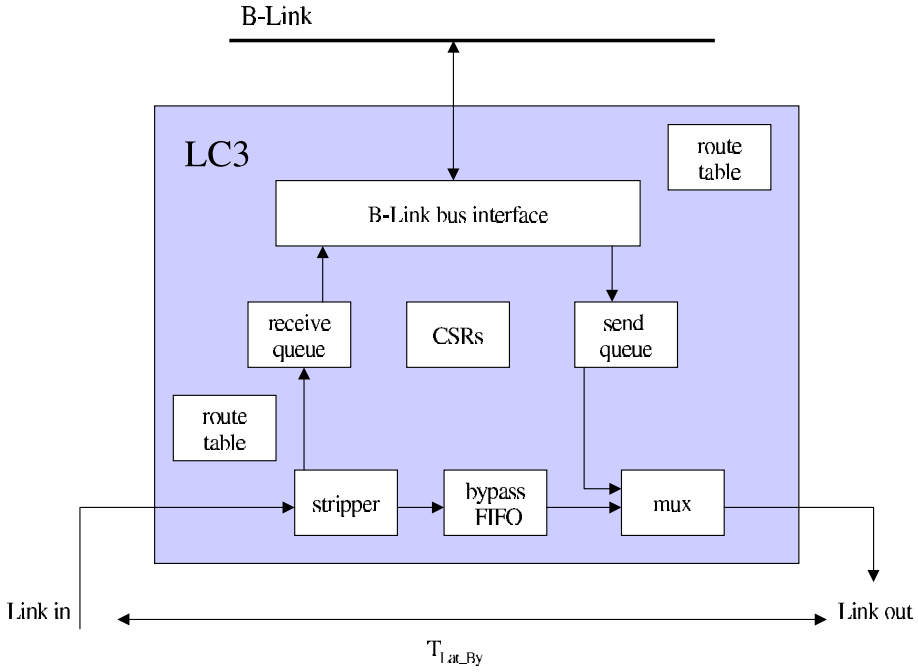

Figure 4.13: Block diagram of the link controller chip LC3.

The chip comprises high speed SCI input and output links, queues for sending and receiving packets, lookup tables for packet routing, a bypass FIFO to store an incoming packet if necessary. This is only the case if the node is sending a packet from its send queue. In addition some upper level protocol management for transaction handling has been implemented.

When a packet is received the stripper block takes the packet off the ring. The target ID field is compared with the criteria that have to be met to strip the packet off the ringlet. Criteria are that the packet has to be routed or that its final destination has been reached. If the receive queue has space available the packet is stored and forwarded to the B-Link afterwards. If no space is available in the receive queue the packet is discarded and a echo busy packet is sent to the sending node. Both send and receive queue can store up to 8 packets. However, one buffer space for the opposite send packet type is reserved. This means that a maximum of 7 request-send packets can be stored since one slot is reserved for a response-send packet and vice versa. Scheduling between request and response packets is done automatically. If the packet uses the node just as a bypass node the packet is forwarded into the bypass FIFO. The time it takes to bypass a node is depicted in the figure. T_{Lat_By} will be determined in 5.1.4.

The LC3 can check one packet at a time. After sending a packet it has to check its bypass FIFO. Therefore, the bypass FIFO must be large enough to buffer the largest possible

packet size. The LC3 uses the basic packet types introduced in 4.4.2. Table 4.1 shows the packets used by the LC3.

Table 4.1: LC3 packet types as described in [31].

Packet Type	Description
request-send	read or write request sub-action content
request-echo	request sub-action local acknowledgment
response-send	read or write response sub-action content
response-echo	response sub-action local acknowledgment
sync	data path deskew packet
init	hardware init reset, stop, and clear packets

Figure 4.14 shows an SCI tracer screen shot. The LVDS signals of the SCI link are converted into TTL, which can be analyzed using a logic analyzer. The transfer shows an nwrite64 packet. This is a send-request packet having a data payload of 64 Byte. The target and destination IDs are highlighted. The signal F1 is the SCI flag signals which delimits packets. The clock signal does not apply since the logic analyzer has been used in state analyzing mode. The chronological order of the symbols is from top to bottom. Therefore one symbol is transmitted in between target and source ID (compare figure 4.12).

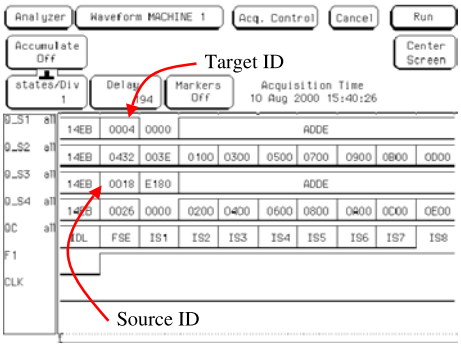

Figure 4.14: SCI trace of a request-send packet.

The LC3 supports all SCI data packet sizes besides the 256 Byte packet. A 128 Byte SCI packet has been defined instead.

Bandwidth Allocation

The LC3 implements a fair-only bandwidth allocation protocol. The bandwidth is allocated through idle symbols. The SCI standard defines low and high-type priority levels [26]. However, only low-type idles have been implemented by Dolphin. Therefore, no node can be privileged. A node can only send a packet if the packet can be postpended to an idle symbol on the ringlet with a specific bit, the idle.lg, set. Thus, sending of packets can be blocked if the bypass FIFO is emptied at the moment or no idle symbol with its idle.lg bit set is on the link. Bypass traffic is always prioritized. Therefore, if the packet

frequency on the link is such that the idle interval in between the packets is less than the bypass FIFO size a node could undergo starvation. To avoid this a starving nodes begins to issue flow control by clearing idle.lg[7] bits to prevent other packet producing nodes from sending.

Queue Allocation

Queue allocation as defined by the SCI standard [26] is a reservation scheme. A node can be in four different queue allocation states. If a packet can not be transmitted due to a filled receive queue the internal state of the node switches to a state where queuing space is preallocated for certain packets. At the same time the requester is notified by an echo busy packet to send with a different priority. The so called A/B aging protocol avoids life locks and has been implemented according to the SCI standard [26].

The PSB66 - The Interface to PCI

The PSB66 is Dolphin's interrupt capable interface to the PCI bus. The bridge supports fast 64-bit/66 Mhz busses according to the PCI Local Bus Specification, revision 2.1 [32]. Only those features are discussed in this section that play a significant role in the result section. Figure 4.15 shows a simplified block diagram of the PCI chip.

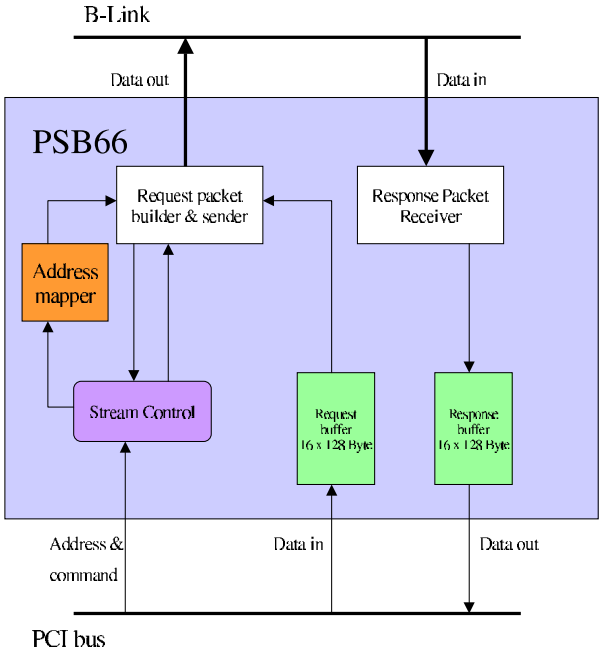

Figure 4.15: Simplified block diagram of the Dolphin PSB66.

The PCI chip keeps track of outstanding transactions with a stream model. Whenever a PCI transaction causes an SCI request a stream buffer is occupied until the SCI transaction is completed. For nread*nn* and nwrite*nn* transactions this becomes true when the

[7] This is one specific bit of an idle symbol.

response send packet is received. As the block diagram shows, 16 streams have been implemented for reading and writing each. Every stream entry can hold 128 Byte of data. Streams can be filled and emptied in the most efficient way by nread*128* and nwrite*128* transactions. However, also nread*64* and nwrite*64* transaction are used if the amount of data exceeds 64 Byte but is less than 128 Byte. The remaining data is sent using selected byte transactions. However, only one outstanding transaction is allowed for partially filled buffers. Therefore, the most efficient way to send data is the usage of 128 Byte data payload packets. Dolphin guarantees that they are translated into 128 Byte SCI request-send packets and eventually are written in a 128 Byte PCI burst transaction on the remote PCI bus. Partially filled buffers are flushed when one of the following events happen:

- The address crosses a 128 Byte boundary.
- A Timeout exceeds a limit, which can be set through a CSR register. The minimum value is $0.96\mu s$.
- Non-continuous addresses.

There are other ways to flush the stream buffers. However, that requires access to certain CSRs.

A DMA engine is implemented. Since some results are presented, which make use of the DMA engine the key features are summarized:

- DMA push mode pushes data from PCI to SCI.
- DMA pull mode pulls data from SCI to PCI.
- Single and chained mode capability.
- Operates with 8 Byte granularity.
- Up to 256 DMA Control Blocks (DMACB) can be processed.

The PCI 64/66 SCI Adapter Card

The basic board which allows simple ringlet topologies comes with one LC3 chip. A second SCI link chip can be added by mounting a daughter card. Figure 4.16 shows a block diagram of a Dolphin SCI card carrying a daughter card to allow two-dimensional topologies.

Dolphin's B-Link, a 64-bit/80 MHz bus, is used as a backend interconnect. It can connect up to eight LC3 SCI link chips and allows data rates of up to 610 MByte/s. The B-Link packet format is a superset of the SCI packet format.

When using a two-dimensional topology data has to be routed to reach its final destination. The current default setting uses dimensional routing such that data first traverses the network in direction of the daughter card cabling, eventually gets taken off the ringlet and is sent in direction of the main card cabling. Internally this is accomplished by using a table lookup. However, this is the default routing scheme. Others have been implemented, but must be configured before usage.

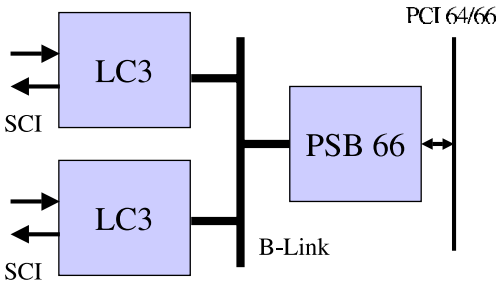

Figure 4.16: Block diagram of the Dolphin SCI card. In the sketch two LC3s are connected by the B-Link bus which allows a two-dimensional topology.

Software Interface

The SISCI[8] API is available at Dolphin's Internet site, which provides support for creating and exporting local memory segments, and connecting to and mapping of remote memory segments. Figure 4.17 shows how the shared memory is set up.

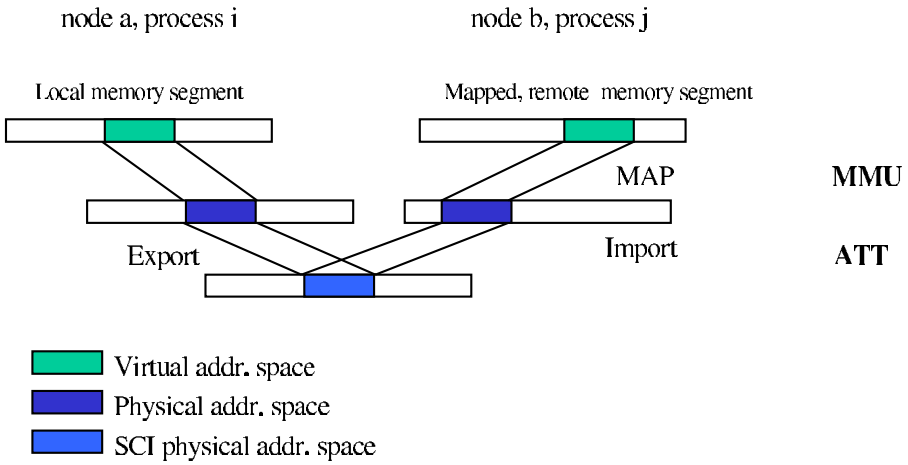

Figure 4.17: Address spaces and address translations in SCI clusters.

A node which wants to share memory with other nodes allocates a certain amount of memory and exports it into the SCI address space. Other nodes import the region into their physical address space and eventually map it to the virtual address space of the calling process. The translation between the local physical addresses and the global SCI addresses is maintained by on-board address translation tables (ATTs).

Once the memory mappings have been set up communication between nodes can be established by using CPU load and store operations. Thus, the importing process uses a store operation to a virtual memory address in case of a write operation. The virtual address is translated to its physical counterpart by the MMU. However, by importing the memory segment the region has been mapped to the SCI NIC as demonstrated in figure 4.8 and the data are transferred to the remote location via the SCI NIC.

[8]Software Infrastructure for SCI

4.5 Traffic Shaping and Network Congestion

The LHCb Level-1 trigger network has to provide an aggregate bandwidth of 4.2 GByte/s. Event data being 4.2 kByte in size is sent to a specific CN within a microsecond. Since neither present nor future CNs which are based on PCs can stand an input requirement like this, traffic has to be scheduled to avoid congestion in the receiver. Neither memory bandwidth nor the PCI bus can satisfy the bandwidth requirement. However, the first buffers that will overflow are the input buffers of the NIC.

A network can be understood as a network of queues. Every node attached to the network has an input and an output queue. If the rate at which packets arrive exceeds the rate at which packets can be transmitted the queue size grows increasing the latency for the network path. Even if the packet arrival rate is less than the transmission rate the queue size will grow as the arrival rate approaches the transmission rate. The reason is that a node does not only receive and pass packets. It also has to make a routing decision on every packet received. If a node reaches a saturation point which means that no buffer space is available anymore it could do one of the following things: drop the packet or issue some kind of flow control. Packets have to be resent which could be happen by a hardware protocol as foreseen by SCI or in software as widely implemented in TCP solutions. However, issuing flow control can quickly congest the entire network. If one node restricts the packet flow coming from another node, the other node's input buffer will fill up and so on. To prevent congestion on the entire network the traffic has to be scheduled when packets enter the network. A method will be introduced in chapter 6. Applying a congestion control mechanism when packets have already entered the network usually does not apply.

Figure 4.18 shows the buffers that are present in an SCI network using Dolphin's network equipment.

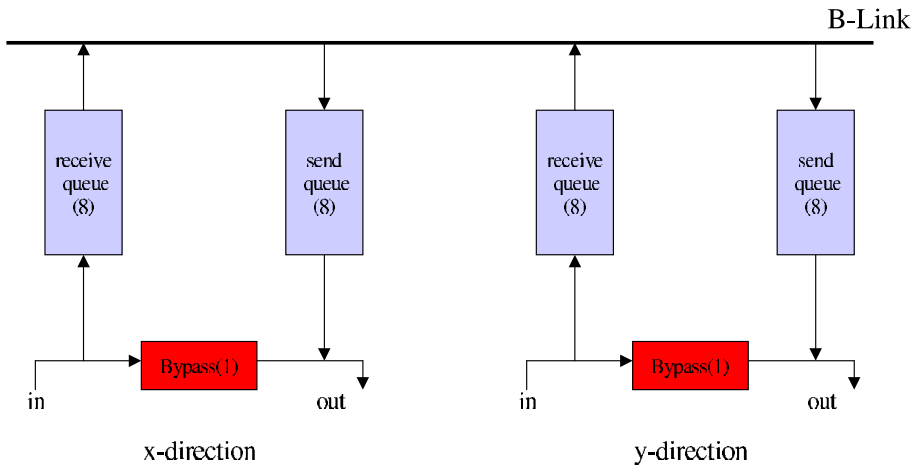

Figure 4.18: Packet buffers in SCI cards.

Every receive queue can hold 8 packets total. However, it always reserves one buffer slot for the opposite packet type. This means that a maximum of 7 request-send packets can be stored in the queues since one queue space is reserved for a response-send packet.

Queuing space in case of the Dolphin SCI card can be summarized as follows:

- There is space for one 128 Byte packet in the bypass FIFO.
- If a packet has to be routed there is space for 14 request-send packets at the most.
- A receiving node can store up to 7 request-send packets.

A CN can hold up to 7 request-send packets. If traffic is not scheduled packets coming from the network feeds would arrive at the same time causing buffer overflow and therefore retry traffic. Chapter 6 introduces the TagNet which schedules the transfers in the Level-1 compute farm such that a receiving CN can handle the input rate without buffer overflow.

Chapter 5

Performance of the Scalable Coherent Interface

Dolphin's SCI implementation is analyzed in means of throughput, interrupt capability, and latency. This is done to get a general understanding of the hardware. Both software initiated and hardware initiated transfer modes are presented. Software initiated DMA offers bandwidth performance up to 248 MByte/s for large block sizes whilst PIO dominates for small data sets. However, since a global shared memory can be set up transferring data directly by hardware seems to be advisable for applications that require better performance. Hardware initiated DMA offers a way for excellent PCI bus utilization and thus minimization of overhead caused by PCI idle cycles. Whilst the non-interleaved mode uses one data source and seems to be sufficient for most applications, the interleaved mode offers best possible PCI bus utilization. Both methods can transfer small data blocks of 192 Byte and less with a rate beyond 1 MHz. Data packet latency is determined and a the maximum PCI-to-PCI latency for different sized tori is calculated. Calculations and performance measurements on large ringlets show that no network congestion caused by increased path length has to be expected when staying underneath 107 nodes for an idle ringlet.

5.1 Software initiated Data Transfer

5.1.1 Point-to-Point Bandwidth

The point-to-point bandwidth between two adjacent SCI nodes is measured in a six-node ringlet. The PIO measurements use the C language library memcpy function to copy the data into a memory region which has been imported by the sending node using the SISCI library. Copying the data is totally transparent since the C programmer uses a virtual target address and is not aware on which node the address is located. Software initiated DMA (SDMA) measurements differ since the DMA engine of the PSB66 has to be set up. The DMA engine can be used in single or chained DMA mode. The following results were obtained using chained DMA mode, which is especially appropriate for small block

sizes. The maximum number of different DMA descriptors that has been loaded into the PSB66 is 256. All measurements use DMA push mode unless stated otherwise, which means that data are pushed to a remote location. All measurements have been done using a 64-bit/66 MHz PCI bus unless stated otherwise.

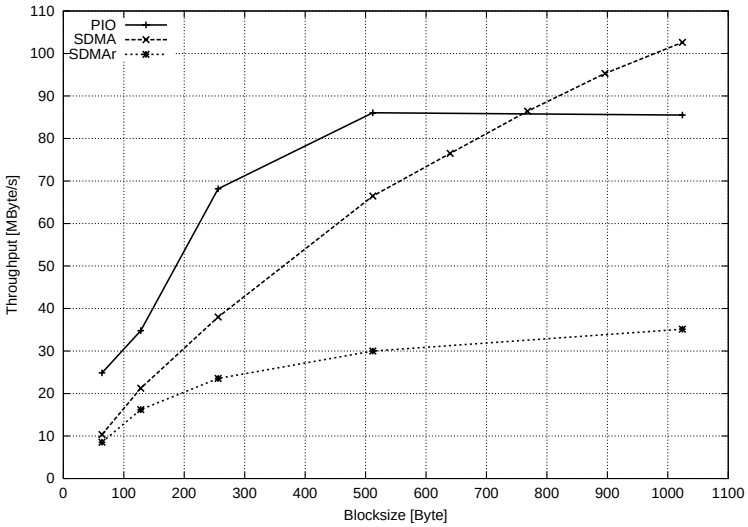

Figure 5.1: The point-to-point bandwidth results obtained with PIO and SDMA for block sizes less than 1024 byte.

Fig. 5.1 shows the results for block sizes less than 1 kByte. Time measurements for software initiated transfers are done on the sending node using the gettimeofday() call. It provides an accuracy in the order of microseconds. However, to insure proper functionality timing has been double checked with a PCI analyzer. One characteristic of the PIO curve is that maximum performance is very poor. The data curve reaches its maximum of 86 MByte/s at a 512 Byte block size. PCI analysis on the sending node reveals that the maximum PCI burst length is only a few data cycles. More important is the fact that the host bridge only forms 32-bit wide data packets on a 64-bit bus. A possible explanation is that CPU-to-device data movement, which is a PCI memory write transactions, is not well supported. However, this result may vary depending on the chipset used. The chipsets used in these section are ServerWorks IIIHE-SL and IIIHE chipsets [33] with the write combining feature obviously not enabled. The maximum burst length observed has been restricted to not more than 5 clock cycles. To reach high throughput, DMA capability of the PCI card is assumed by the chipset manufacturers. The SDMA data curve is rising and does not reach its maximum in the plotted data range. However, PIO guarantees higher throughput for block sizes smaller than 768 byte. The advantage of PIO is its true zero software overhead while SDMA has the disadvantage that the DMA engine of the SCI card has to be programmed prior to transfers. The header(H) and trailer(T) depicted in figure 4.6 does vanish in case of the SCI card. Since DMA is capable of PCI bursts when fetching data from host memory, its performance outranges PIO for larger block sizes. The SDMAr labeled plot reads from a remote location. Performance is very poor compared to writing since data is fetched out of remote memory. However, the major reason for performance loss is the fact that the SCI card issues only one outstanding read at a time. Idle times of 2.8μs in between memory reads can be observed on the remote

side. Since the maximum performance is only about 41 MByte/s, reading is not shown in the upcoming plot for block sizes greater than 1 kByte.

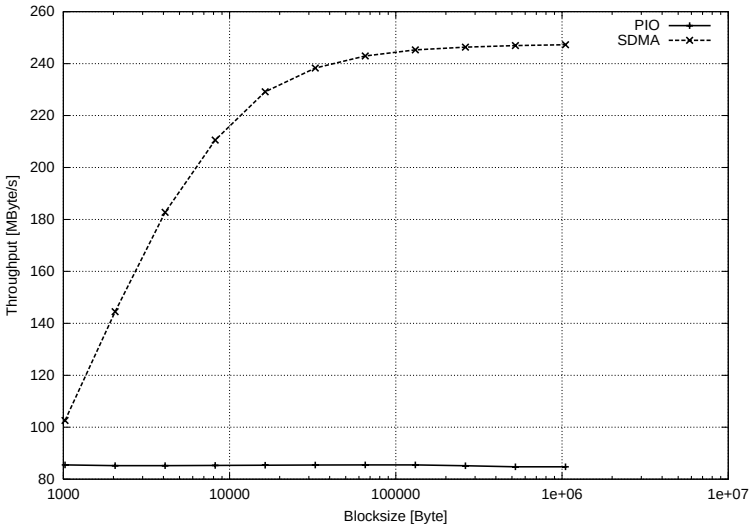

Figure 5.2: The point-to-point bandwidth results obtained with PIO and SDMA for block sizes between 1024 byte and 1 MByte on a logarithmic scale.

Fig. 5.2 shows the results for blocks greater than 1 kByte. While the PIO data curve does not climb above 86 MByte/s, the SDMA data points show a point-to-point bandwidth of up to 248 MByte/s. Moving block sizes greater than 1 MByte shows that the curve really reaches its maximum.

5.1.2 Maximum Performance Tests

Maximum performance tests have been conducted in a 10×3 node torus using SDMA. One node has been chosen to act as data sink and a variable number of nodes have been chosen to act as data sources. One test utilizes only nodes that are located in the same horizontal ringlet and therefore analyzes the maximum performance if no packet routing takes place. The other test has been conducted using the entire system. Data block sizes are 128 Byte, which corresponds to one SCI data packet, 1 kByte, 10 kByte, 100 kByte, and 1 MByte. A PCI analyzer in the target node calculates the total bandwidth.

Ten node ringlet - no routing

Fig. 5.3 shows the results of the ringlet topology. For one source, the result is similar to the plots already discussed; two sources push the maximum bandwidth up to 315 MByte/s. An analysis of the PCI bus shows that the back-to-back transmission gap between PCI bursts on the receiving side can be as low as 9 clock cycles. All data on the PCI bus is transferred using bursts of 128 Byte in size. The maximum bandwidth can be achieved by two nodes acting as senders. Pushing the target node to the limit by adding data sources decreases the aggregate bandwidth by 15 MByte/s at the most. Five data sources can push a maximum data rate of 298 MByte/s. Analysis of the sending nodes reveals that bandwidth is not distributed to equal amounts. The sending node, which is sitting

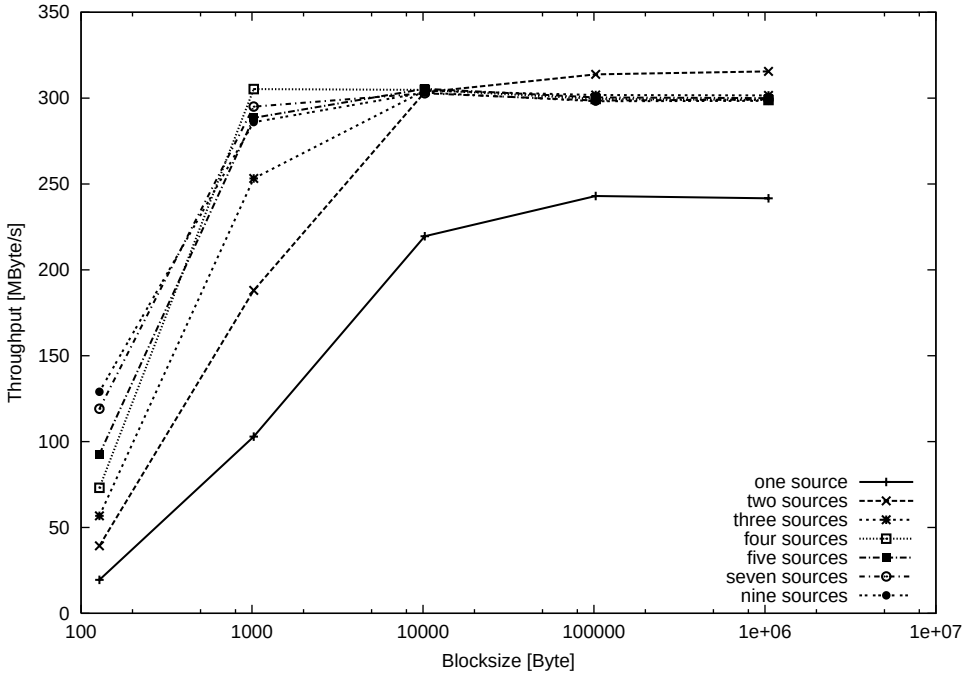

Figure 5.3: Performance measured on the receiving node with different numbers of data sources.

upstream at the most and thus does not have a bypass FIFO filled by the other sending nodes, gets allocated the most bandwidth. For a more detailed discussion on that topic see 7.9.

30 node torus

Similar performance tests have been made in the torus topology shown in Fig. 5.4. To build a torus, each node has to be equipped with an additional SCI mezzanine card, which sits on top of the SCI PCI card. Thus, each card is able to send data in x or y direction, to receive data from x and y links, and to reroute packets from x into y direction.

Fig. 5.5 shows one of the results of the torus topology. The results with two data sources differ depending on the combination of sending nodes. The maximum bandwidth for two data sources equals 315 MByte/s. However, figure 5.5 also shows a combination with a maximum throughput of 232 MByte/s in the target node. This is even less than the result obtained for one sender and one receiver. Combinations have been chosen as follows:

- **two sources i**: Node #00 and node #20 are sending nodes. Data coming from node #00 has to be routed at node #09. Data originating from node #20 is routed on node #29.
- **two sources ii**: Node #00 and node #10 are sending nodes. Data from node #10 does not have to be routed. Target and source share the same torus row.
- **two sources iii**: Node #09 and node #20 are sending nodes. Data originating from node #09 does not have to be routed. Target and source share the same column.

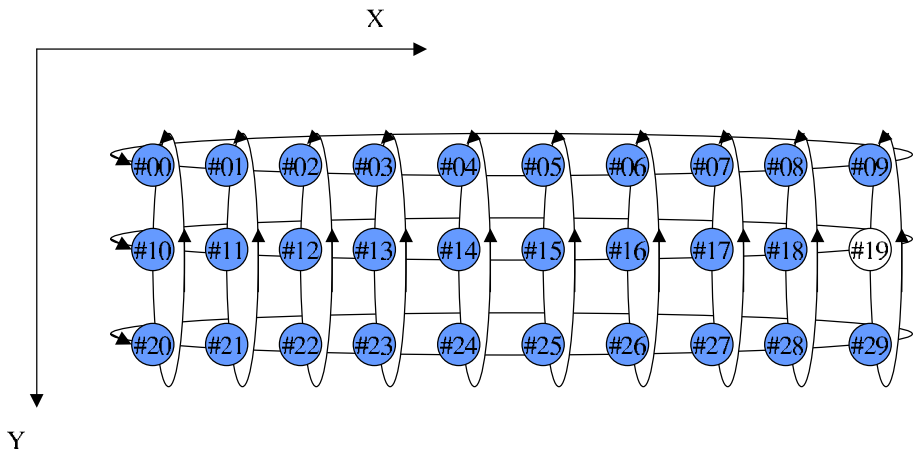

Figure 5.4: 2D-torus as used for the benchmark tests presented. Always the same node has been used as receiver. It is depicted as a blanc circle. Direction of SCI data packets is depicted by the arrows.

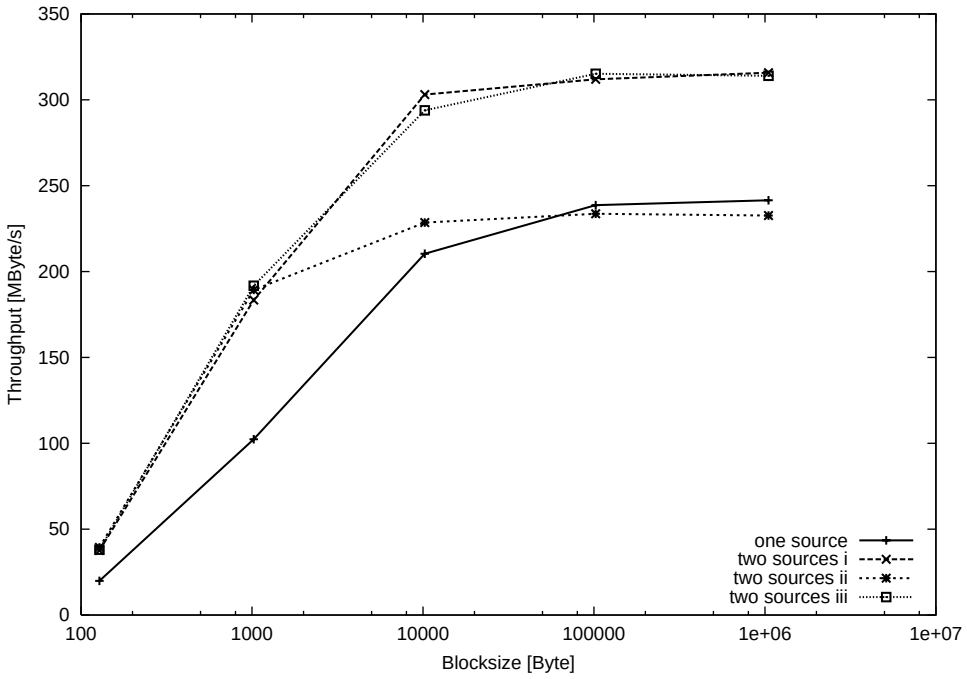

Figure 5.5: Maximum performance measured on node #19, the receiving node, with one data sources and two data sources.

Combination ii shows poor performance. It is the only combination with both x and y SCI receive queues being filled at the same time. Figure 4.18 shows the two receive queues being present in Dolphin SCI cards. Obviously queue read out via the B-Link is not fair. Since one receive queue is preferred the other queue is not read out frequently enough making the bandwidth drop.

However, even more data sources have been added. The results are shown in figure 5.6.

The source nodes in figure 5.6 have been chosen as follows:

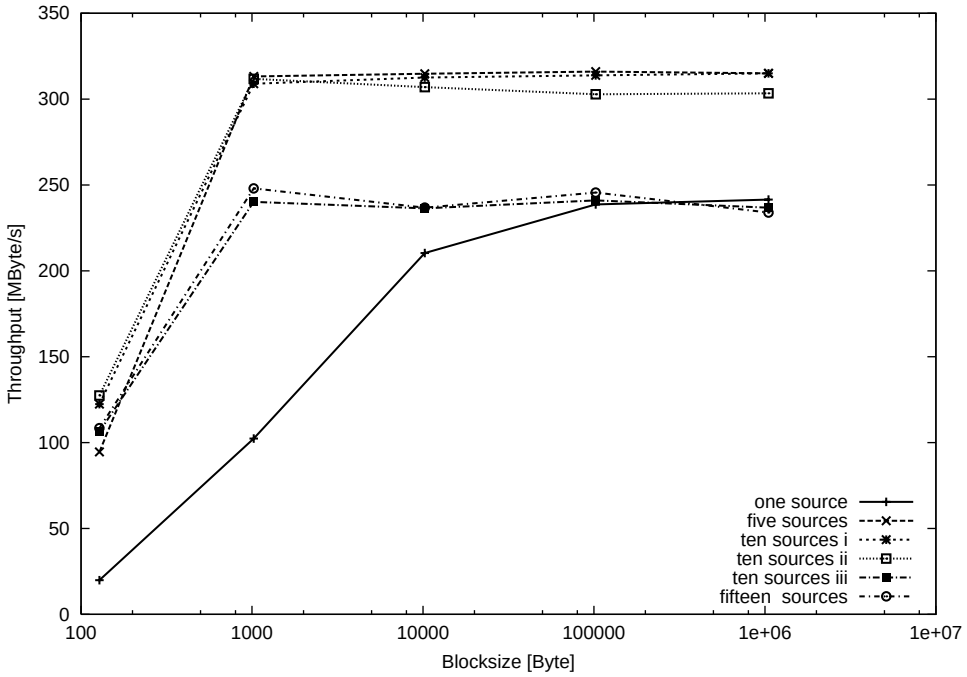

Figure 5.6: Maximum performance measured on node #19 with a variable number of data sources.

- **five sources**: all data sources are located in the upper most 10 node ringlet of figure 5.4.
- **10 sources i**: top row in figure 5.4.
- **10 sources ii**: 5 nodes are located in the top row, 5 in the bottom row.
- **10 sources iii**: 5 nodes are located in the middle row, 5 in the bottom row.
- **15 sources**: 5 nodes are chosen from a row, respectively.

Five sources all located in the same horizontal ringlet have been chosen. The resulting curve reaches its maximum at 1 kByte with a maximum of 315 MByte/s, which equals the maximum value obtained for two sending nodes.

Three different combinations have been chosen, which use 10 sending nodes. Combination i, giving the best result, uses 10 nodes located in the upper most row of figure 5.4. Therefore, all packets are taken off the horizontal ringlet by node #09, which inserts them into the vertical ringlet to reach node #19. Combination ii uses two different route nodes. However, the packets originating from the bottom row have to traverse the bypass FIFO of node #09 after they have entered the vertical ringlet. Since a node does not send data during bypass traffic this accounts to a drop in aggregate bandwidth of 12 MByte/s. The third combination uses 5 nodes from a row, respectively. The bandwidth drops significantly to 236 MByte/s. The fraction of packets that originates from the middle row is not routed. Therefore, the receiving node #19 has to accept data packets both from its x and y SCI links at the same time. The same comes true for the result utilizing 15 nodes. This finding has already been observed in figure 5.5.

Taking the results obtained into account it can be stated that a network with one receiver that accepts packets from more than one sending node shows a peak PCI bandwidth of 315 MByte/s. However, there are two situations that have impact on performance. If a receiver has to accept data packets both from x and y direction the bandwidth drops by 30% regardless of the number of sending nodes. Nodes that either send or route data are influenced by bypassing traffic. This can be observed by a 4% drop of aggregate bandwidth at the receiving node.

5.1.3 Synchronization by Remote Interrupts

An SCI card is able to fire an interrupt on a remote node. This is commonly used to inform a remote node that a transaction can be started or is finished. The time for asynchronous remote process synchronization has been determined with the following setup. Two processes have been started on separate nodes that notify each other with the help of the remote interrupt service provided by SCI. The communication has been set up on such a way that the node that has been interrupted sent a remote interrupt to the node that interrupted it. This results in a ping pong like communication pattern. The rate at which two nodes can communicate using the remote interrupt service has been measured. The time, t_{sync}, it takes to send a remote interrupt to a user space process is shown in table 5.1. However, this value includes the SCI network latency, the interrupt latency of the operating system and architecture. The synchronization time t_{sync} is measured on an idle node, which means that its PCI bus is totally free of traffic, and on a busy node, which means that the PCI bus shows maximum traffic. To keep the node busy, two SCI nodes send large blocks of data to guarantee a sustained bus utilization of about 320 MByte/s.

Table 5.1: Average synchronization time in busy and idle mode.

mode	t_{sync}
Idle	$22.6\mu s$
Busy	$35.2\mu s$

Analysis of the PCI bus interrupts shows that there is only negligible fluctuation between interrupts in the idle state of the node. Differences in t_{sync} of up to $2\mu s$ can be seen in the busy state.

5.1.4 Minimum Data Packet Latency

Packet latency is measured from PCI-to-PCI bus using two PCI analyzers. Each one triggers on the transferred data pattern. Both trigger signals are analyzed on an oscilloscope and a time difference is calculated. Fig. 5.7 shows a Number of hops vs. time curve for routed and not routed data packets. The not routed data points have been taken using a single one-dimensional ringlet whilst the routed data points correspond to data traversing two dimensions. A data receiver and a data sender are exchanging 128 Byte packets of data. The number of nodes residing in between are denoted hops. A hop can either be a simple hop where data is forwarded or it can be a route node. On a route node data

is taken off the ringlet and injected into the other one. Both sides trigger on the first PCI data cycle since delays caused by the chipset when accessing host memory are not of interest.

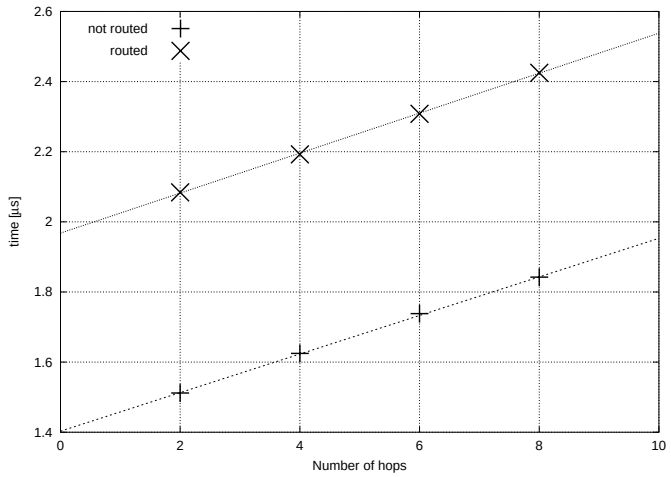

Figure 5.7: PCI-to-PCI latency measured as explained in the text. Paths, which include route nodes are labeled routed data.

Four data points have been taken for routed and not routed paths, respectively. Fit curves are straight lines. Line parameters are summarized in table 5.2.

Table 5.2: Line parameters for the fit lines shown in figure 5.7. The parameter x denotes the number of hops.

data	fit line
not routed	$0.06 \times x + 1.40$
routed	$0.06 \times x + 1.97$

By analyzing the fit data three important parameters can be extracted. The slope of the lines equals the bypass latency of a node which is depicted in figure 4.13 and accounts to $T_{Lat\_By} = 60ns$. The difference in ordinate values, which can be calculated to be 570 ns is the routing latency T_{Lat_Rt}. The PCI-to-PCI latency T_{Lat_PCI} can be calculated to be 1.40 μs being the y-axis intercept of the not routed data curve. Table 5.3 lists a summary of the findings.

Table 5.3: Results of Packet Latency measurements as shown in figure 5.7.

latency	result
T_{Lat_PCI}	$1.40\mu s$
T_{Lat_Rt}	$570ns$
T_{Lat_By}	$60ns$

Taking the results above into account the maximum PCI-to-PCI latency for a $n \times m$

torus can be calculated to be $T_{Lmax} = T_{Lat\_PCI} + T_{Lat\_Rt} + (m + n - 4) \times T_{L\_By}$. This calculation assumes dimensional routing. Figure 5.8 shows a torus with the longest path highlighted.

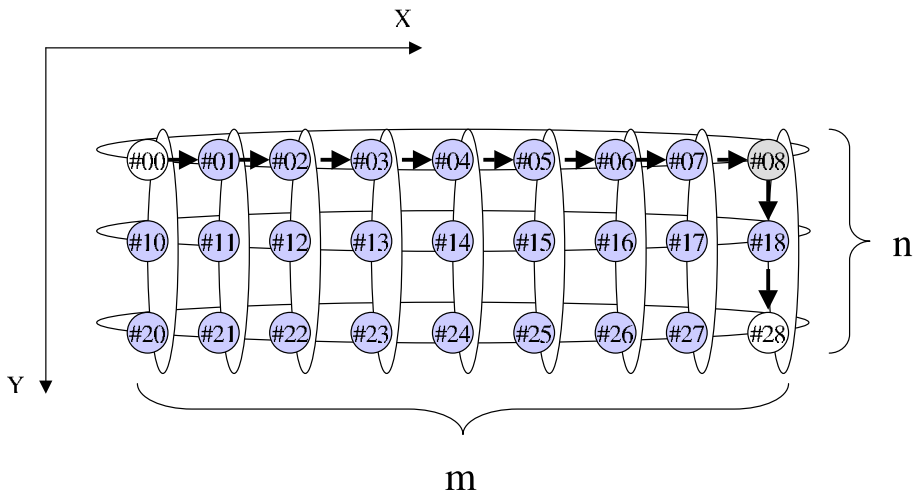

Figure 5.8: Longest data path in a 9×3 torus. Data is sent from node #00 to node #28 via route node #08. The arrows depict the flow of the SCI data packet.

Taking the dimensions of the torus in the figure into account a maximum latency of $2.44\mu s$ can be calculated. However, this calculation assumes that no bypass or route node is sending data itself. An SCI 128 Byte data packet comprises 72 symbols (see 4.12) assuming no header extension. The Dolphin SCI link transfers a symbol every 3 ns. Thus, the time a packet has to wait in the bypass FIFO accounts to 216 ns. Therefore, assuming that the packet has to wait whenever it bypasses a node a maximum latency of $4.17\mu s$ can be calculated for a congestion free system. The SCI cabling in the figure does not reflect the cabling in the real world since long cables are avoided.

5.2 Hardware initiated Data Transfer

5.2.1 Increasing Performance by using FPGAs

When using PIO a process pushes data to a virtual address. The address is translated to its physical counterpart which has been mapped to the SCI NIC earlier. Eventually the CPU writes into the SCI card's input buffer via the PCI bus. To get access to the PCI bus the system's host bridge becomes bus master and issues memory write cycles towards the SCI card, which acts as PCI target. The SCI card transfers the data and writes it to a memory location of the remote node where it can be accessed by a remote process. This is depicted in the top figure of 5.9. However, the results presented in 5.1 show poor performance since burst length is limited to five clock cycles with data cycles being restricted to 32-bit in width.

However, once the shared memory has been set up during the initialization phase, data can be pushed to a remote location by writing to the physical address associated with the imported memory region. This region is also visible to another PCI device. Therefore, any PCI device can access the input buffers of the SCI card by issuing PCI memory write cycles. A DMA engine implemented on a FPGA provides the necessary functionality to access the network card without data having to traverse the hostbridge. This transfer mode is referred to as Hardware initiated DMA (HDMA).

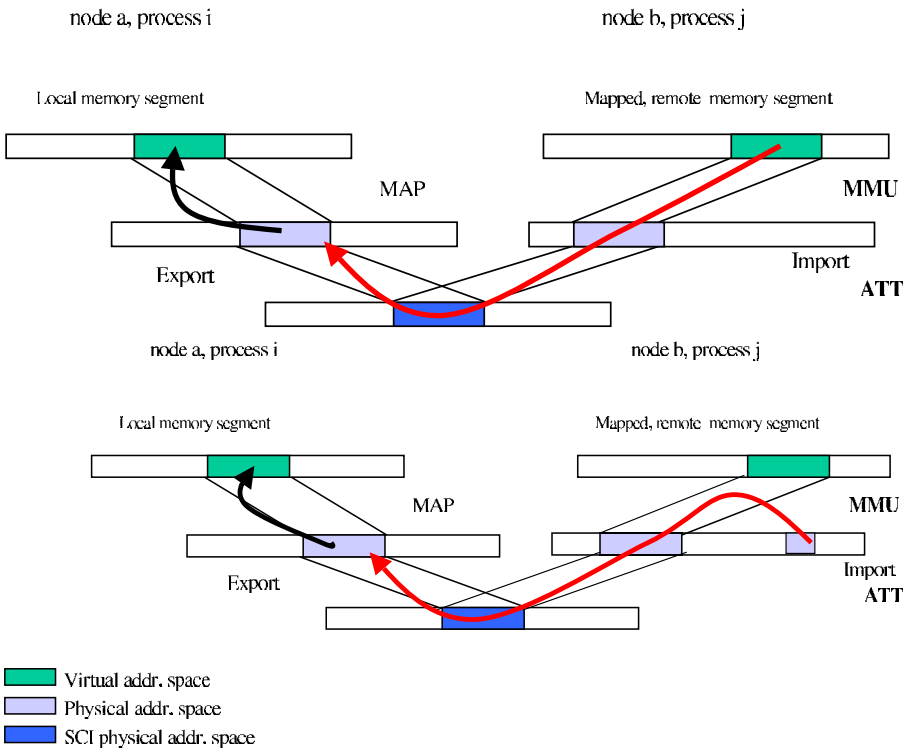

Figure 5.9: Data transfer using HDMA (bottom) compared to common transfer (top). Local and remote physical memory regions are of equal size. The size of the physical memory region associated with the HDMA engine which is depicted on the right of the bottom figure is not important for the data transfer since it is only used to configure the HDMA engine via PCI.

An FPGA (Field Programmable Gate Array) is a reconfigurable integrated circuit that

contains many identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. Designs can be implemented using an abstract description language like VHDL [34] or Verilog [35]. They allow to describe logic components like finite state machines or single gates. Specific software provided by the FPGA vendor is used to create an FPGA configuration file. The results presented in this section have been created by implementing a DMA engine in an Altera APEX20KE [36] device and a Lucent OR3TP12 FPSC [6]. Both FPGAs come with custom made PCI cores. A more detailed explanation of the logic blocks can be found in 7.2.2.

Data are transferred directly from the FPGA to the memory mapped input buffers of the SCI card located on the same PCI bus. Afterwards the data are sent to their remote destination. This is depicted in the bottom figure of 5.9. This method of operation is also known as device-to-device copy. The data to be transferred has to be written into a buffer located on the FPGA board in an initialization phase. For test reasons data can just be generated on the fly. Since no host bridge is involved except for PCI bus arbitration, this mode promises very high data rates even for small packets. This method of data transaction can be implemented in two different ways. One method uses two FPGAs, each having its own PCI slot on the same bus, in tandem mode which means that the FPGAs operate interleaved. The other method, which is easier to implement but less efficient because of the increase of PCI idle cycles in between bursts, is to use only one FPGA. Both setups are sketched in figure 5.10.

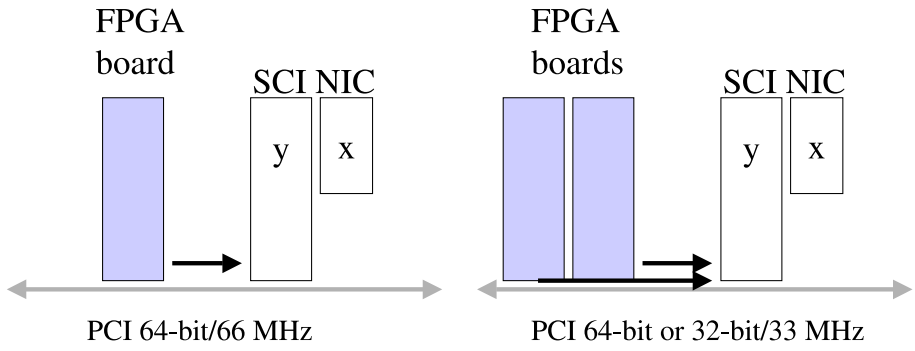

Figure 5.10: Device-to-device copy. Non-interleaved mode is depicted on the left side, the tandem mode operation solution on the right side.

However, in both methods the FPGA board requests bus mastership and send data in a single burst to the SCI card which starts a data transaction to a remote location. The address which is needed for the PCI transactions have to be stored in an HDMA descriptor buffer. Afterwards transfers are controlled by state machines implemented in the FPGA and no software is necessary.

5.2.2 Non-Interleaved Transfer Mode

In the setup presented an Altera APEX20KE FPGA board [36] and a Dolphin 64/66 SCI card [30] are located on the same 64-bit/66 MHz bus of a standard Linux PC with a

ServerWorks IIIHE chipsets. Figure 5.11 shows the results of a point-to-point connection benchmark with variable data packet size.

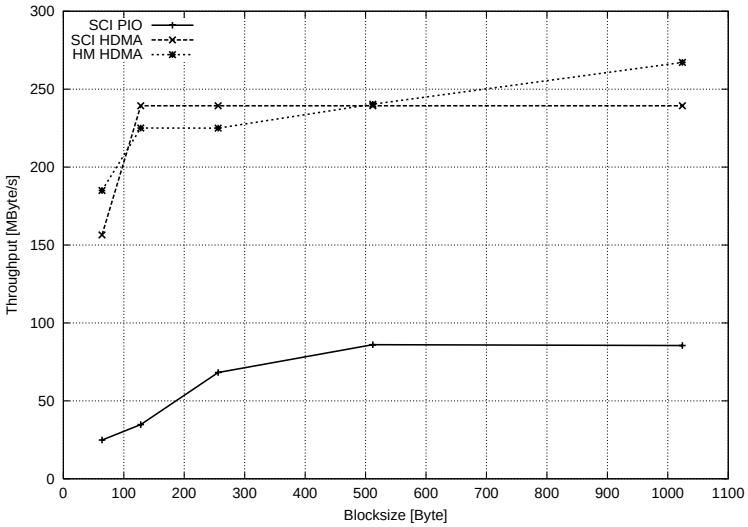

Figure 5.11: FPGA initiated transfer vs. PIO for block sizes less than 1024 Byte as measured in a point-to-point connection. The different curves are explained in the text.

Three curves show the difference between HDMA and PIO initiated by the CPU. The data curve labeled SCI PIO is identical to the data presented in figure 5.1. The HDMA curves have been measured by using a PCI tracer in the sending node. A process on the sending side initializes and triggers the FPGA, which pushes the data to the address specified. After the transfer has been completed the receiving process is notified and starts checking the received data stream.

The data labeled HM HDMA shows data written into local host memory. HM HDMA performance depends on the chipset since the FPGA tries to send one burst per transfer. Since target retry and target disconnects slow down transmission the maximum data rate observed has been 267 MByte/s. The data labeled SCI HDMA shows the performance of an SCI point-to-point connection. The curve reaches its maximum at 128 Byte since the SCI card issues a target disconnect type C on reception of 128 Byte. Both a target disconnect and a new transfer require a new bus arbitration. Since the FPGA we use can not have more than one outstanding transaction its internal PCI core state machine requires to be in its initial state before a new arbitration request can be issued. Figure 5.12 shows a transaction as observed on the sending node.

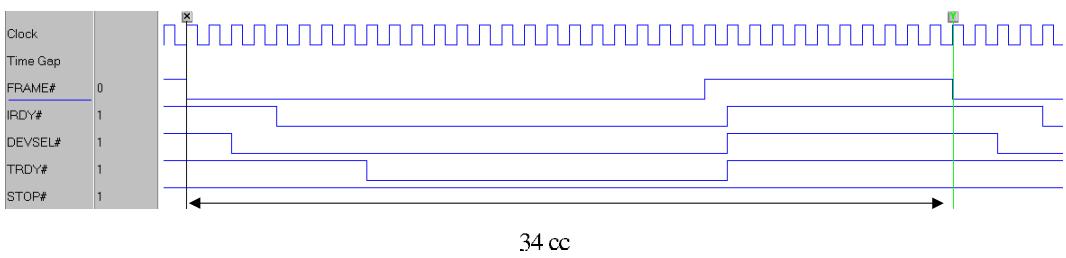

Figure 5.12: PCI trace of a 128 Byte HDMA transfer as seen on a 64-bit/66 MHz bus. The total transaction time equals 34 clock cycles.

Figure 5.13 shows a sketch of a PCI transfer depicted in figure 5.12 in terms of clock cycles. The target setup time of the SCI card is seven clock cycles. This time is fixed since it is due to the SCI card's PCI interface chip PSB66. The burst-to-burst gap of 10 clock cycles can be minimized by choosing a different FPGA type or by a method introduced in the following section.

Figure 5.13: Sketch of the PCI cycles of a DMA transfer as seen on a 64-bit/66 MHz bus. The cycles can also be seen on the PCI trace in figure 5.12

However, the minimum number of PCI cycles required for one 128 Byte burst can be calculated to be 25 assuming one idle cycle in between bursts. Thus, a maximum bandwidth of 325 MByte/s for an SCI card used as target can be calculated which is 61 % of the theoretical bandwidth of the PCI bus.

Figure 5.14 shows the packet frequency for block sizes less than 1 kByte.

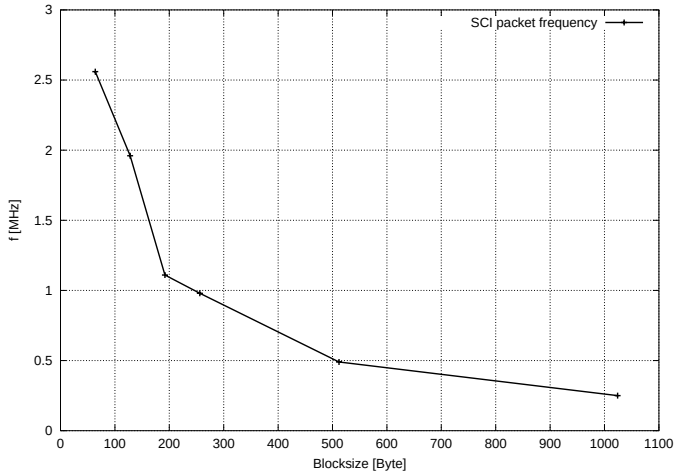

Figure 5.14: Packet frequency in MHz as measured for block sizes less than 1024 Byte on the sending node.

SCI HDMA data measured and presented in figure 5.11 has been analyzed in terms of frequency. A maximum frequency of 2.56 MHz can be extracted for sending 64 Byte bursts. Even for data bursts of 192 Byte in size the frequency is beyond 1 MHz.

However, figure 5.11 shows odd behavior of the HM HDMA curve for the 128 Byte and 256 Byte data points. Data transfer to local memory seems to be less efficient than data transfer to remote memory via the SCI card. PCI cycles have been analyzed for SCI transfers on the sending and receiving side, and for HM HDMA transfers. Figure 5.15 shows the result of the analysis.

The left part of the figure is also shown in figure 5.13. However, the 34 clock cycles total on the sending side are distributed differently on the receiving side. The PCI command issued is MWI (memory write and invalidate) which results in one wait cycle before data is actually sent. The MWI PCI command is semantically identical to a memory write transaction. However, it guarantees a minimum transfer of one complete cache line. After

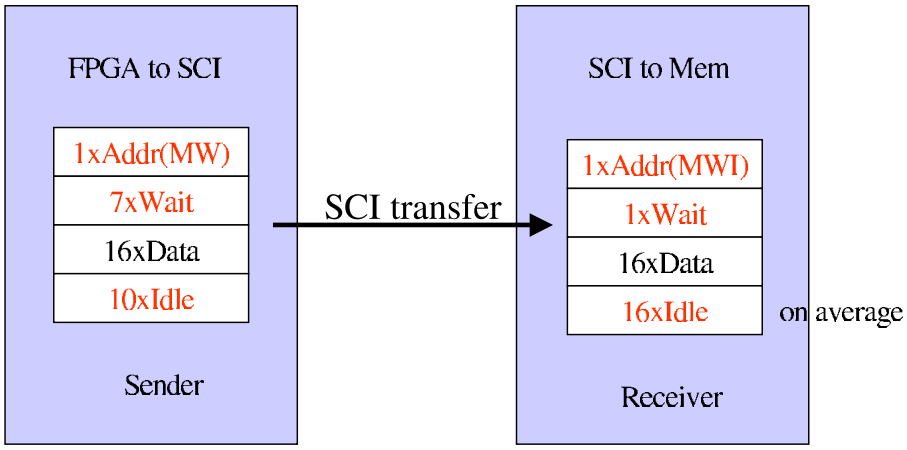

Figure 5.15: Sketch of PCI cycles as measured on receiving and sending node of an 128 Byte SCI transaction.

a complete burst transfer on the receiving side 16 idle cycles can be observed on average. Neither retry traffic nor disconnects have been observed on the receiving side.

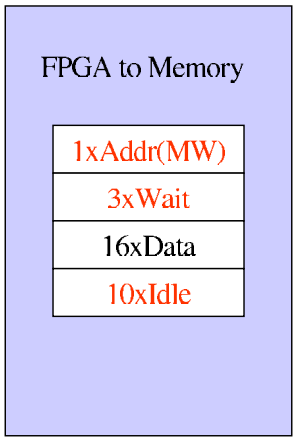

Figure 5.16: Sketch of PCI cycles of a 128 Byte transaction into local memory.

Figure 5.16 shows the PCI cycles involved on an HDMA transaction into local memory. The node used is the same as the receiving node depicted in figure 5.15. However, the PCI command issued is memory write which results in three wait cycles instead of one in the previous case. The DMA engine always sends the following burst after ten clock cycles which amounts to 30 clock cycles total. When looking for retry traffic and disconnects it becomes clear that odd performance in figure 5.11 is caused by frequent retries on the PCI bus when the FPGA pushes data into local memory.

Figure 5.17 shows the reason of the performance loss.

The hostbridge buffers the incoming data such that it can utilize the faster system bus best. The buffer is flushed either when it is filled or after a timeout applies. The host bridge does not accept data when it flushes its buffer. In case of 128 Byte and 256 Byte transfers the FPGA can not fill the buffer in one complete burst. The hostbridge eventually writes the buffer content into host memory and does not accept any incoming data anymore.

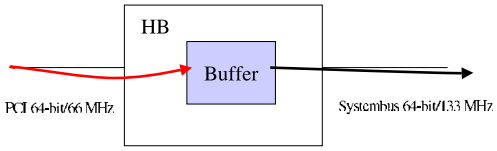

Figure 5.17: The hostbridge buffers the incoming data such that it can issue a burst on the system bus.

The FPGA sends data with 10 idle cycles in between bursts whereas the SCI card issues 16 idle cycles on the receiving side. The hostbridge buffer is not flushed yet when new data arrives in case of HM HDMA resulting in retry traffic. The SCI card being the slower PCI initiator sends its data when the hostbridge is ready to accept a new burst resulting in better performance on the PCI bus.

In case of larger block sizes the buffer can be filled in one complete burst utilizing the hostbridge best in case of HM HDMA. The SCI card does not accept bursts larger than 128 Byte which decreases performance compared to the hostbridge for larger block sizes.

5.2.3 Interleaved Transfer Mode

Interleaved mode has been tested in a 32-bit/33 MHz PCI environment since all COTS[1] PCs come with only two 64-bit/66 MHz PCI slots. The FPGAs, which have been used to demonstrate the capability of interleaved transfer mode (Lucent ORCA OR3TP12 FPSC) have an internal 32-bit bus and a PCI core that is not pipelined and therefore do not allow more than one outstanding PCI request. A sketch of the setup can be seen on the right of figure 5.10. At the time of benchmarking Dolphin's SCI cards built around the link chip LC2 and the PCI bridge chip PSB64 have been used. Since the LC2 supports a maximum of 64 Byte data payload per SCI packet the DMA burst packet size has been chosen to be 64 Byte, as well. However, the results in tandem mode are very promising. Sending 64 Byte packets to the SCI card a rate of 1.51 MHz with a back-to-back burst gap as low as 1 PCI clock cycle has been observed. Figure 5.18 shows a PCI trace of an interleaved transaction on a 32-bit/33 MHz PCI bus.

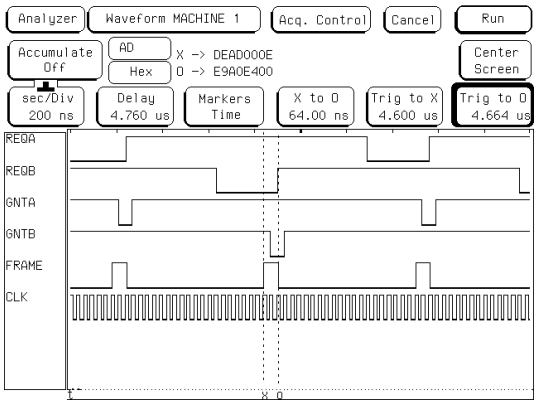

Figure 5.18: PCI trace of an interleaved transaction. The idle time in between bursts is reduced to a minimum of 1 clock cycle.

[1] commercial off-the-shelf

A sketch of the PCI bus cycles can be seen in figure 5.19. Figure 5.18 also shows how the devices take turns in accessing the PCI bus. This can be seen by looking at the two PCI REQ and GNT signals. The meaning of the signals have been explained in 4.2.

Figure 5.19: PCI cycles for one DMA transfer as seen on a 32-bit/33 MHz bus using the PCI 64/33 SCI card.

The four wait cycles in between the PCI address cycle and the data cycles are due to the target setup time of the SCI card and can not be minimized.

5.3 Large SCI Ringlets

Most of the work presented in this thesis has been done of moderate sized ringlets of up to 10 nodes and tori. However, some tests have been made using a large ringlet of 30 nodes. Emphasis has not been to look at performance but rather on stability and scalability.

Large ringlets are prone to network congestion if the number of nodes exceeds a certain number. The combined network latencies of request-send and request-echo packet must not exceed a time limit which is set by the packet frequency on the sending node. Send queue slots of the LC3 are flushed when the request-echo packet has been received. The packet buffers are depicted in figure 4.18. The bypass latency of a node has been measured in 5.1.4 to be 60 ns. Assuming a ringlet of N nodes, 7 slots in the send queue, and a packet frequency of 1.11 MHz which corresponds to the peak LHCb frequency, the non-equation $7 \times 900\ ns > N \times 60\ ns$ has to hold true.

N can be calculated to be 107 with these simple assumptions. However, if a request-send packet has to wait in a bypass FIFO because the node is sending itself an additional delay of 216 ns has to be added for one hop (compare section 5.1.4). Therefore, $7 \times 900\ ns > N \times (60\ ns + 216\ ns)$ must hold true, resulting in 22. Echo packets are not end-to-end confirmations but local to ringlets. Therefore, even large size tori can be realized.

To get a real hands-on example, traffic in a 30 node ringlet has been chosen such that data packets have been sent by two nodes equipped with FPGAs to 14 nodes each. The 30 node ringlet appeared to be stable since there has not been any network problem after 12 hours. No retry traffic has been observed on the feeding PCI busses which have been monitored with logic analyzers. Additionally some tests using software initiated transfers have been made successfully.

Chapter 6

Level-1 Trigger General Architecture

The basic architecture of the Level-1 vertex trigger system is presented and a preliminary timing analysis of the system is outlined. The system is based on PCs connected by SCI in a 2D torus topology. The input data stream is initially scattered between several RUs, which have to send data to a specific node in the network. On reception a software algorithm performs data analysis and sends a result message to the Level-1 Decision Unit Interface. The Level-1 Decision Unit Interface reorders the events and forwards them to the Level-1 Decision Unit. A maximum latency is set on the overall data path through the Level-1. To avoid network congestion at the receiver, the data are orchestrated by the TagNet. The TagNet is a scheduling network which provides a flexible congestion control mechanism. The TagNet is managed by the TagNet scheduler that keeps track of events currently processed and CPUs which are available to accept new data. The network farm size is calculated to be 17 rows and 18 columns.

6.1 Level-1 Trigger System Architecture Overview

The results already presented suggest that SCI can be used as baseline solution for the LHCb Level-1 trigger. Figure 6.1 shows the basic components of the Level-1 trigger.

The figure shows a 2D topology which connects a certain number of CNs. The network feeds, the RUs, are connected by a congestion control or scheduling network. This chapter focuses on the following topics:

- The data packet latency depends on the length of the network path. Therefore, an estimate on the network size with respect to number of input feeds and number of compute nodes is given.
- SCI allows a different network topology. However, the network topology chosen must provide an aggregate bandwidth of more than 4 GByte/s and must allow to

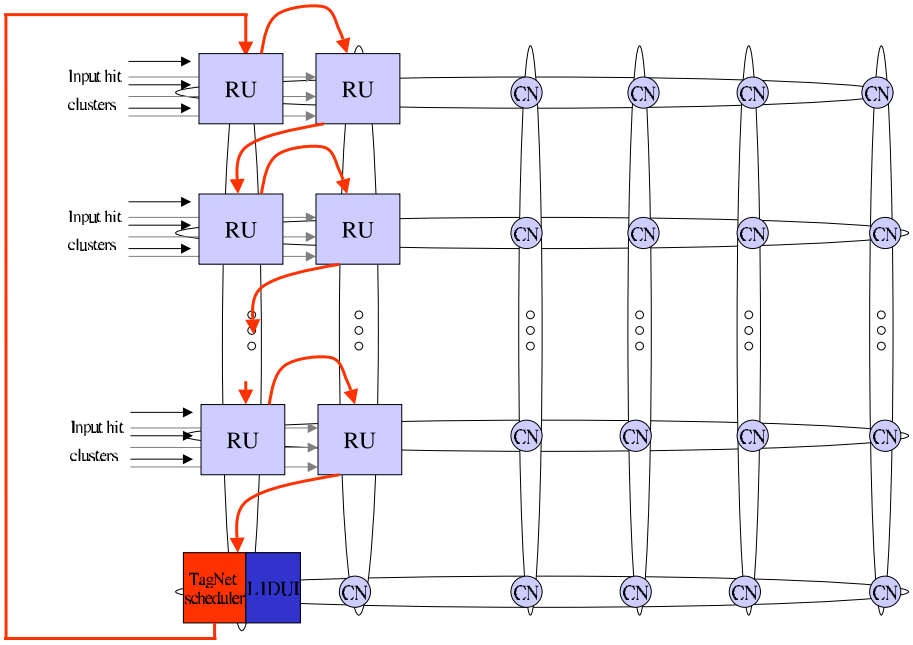

Figure 6.1: The Level-1 architecture as discussed in this chapter.

add both RUs and CNs easily without causing network congestion. Additionally, central congestion must be avoided.

- The CNs can not receive data with a rate of more than 4 GByte/s. Therefore, data transfer has to be scheduled to avoid congestion at the receiver.
- All hardware that has to be traversed by the data in between Level-0 accept and the time the algorithm starts to process the event contributes to the Level-1 latency. Therefore, a timing estimate is given.

6.1.1 Network Feeds (RUs)

The number of RUs is determined by the amount of data that has to be expected. However, the technique how data are transferred is given by the event rate.

The absolute maximum Level-0 trigger accept rate is defined by the readout time of the Level-0 derandomizer. The average rate over a large time window is 1 MHz, which has been chosen as a compromise between the physics performance of the experiment and the cost and complexity of the front-end electronics. Some triggers will be rejected by the central readout supervisor when there is a risk of overflowing buffers in the front-end electronics or overloading the Level-1 trigger system. For more information on LHCb electronics see [37, 38, 39]. However, the maximum Level-0 trigger rate can be determined to be 1/900 ns being 1.11 MHz . Thus, the only way how this can be accomplished with standard components is an approach using HDMA as explained in 5.2.

The buffer depth of the off-detector electronics (ODE) can accommodate a maximum latency of 1820 events [40]. Thus, taking the maximum Level-0 rate into account the maximum time an event is allowed to stay in the Level-1 system can be determined to be 1638 μs .

Figure 6.2 shows the latest results on VELO clusters per ODE (13 on average) at the time of writing, Noise, spill-over and common mode corrections are not included in this simulation result.

The data corresponding to an event is distributed between the R and ϕ sectors of the VELO. The number of VELO stations is not final yet. However, to use some hands-on numbers the following calculations are based on 25 stations as outlined in the VELO TDR [8].

A hit cluster corresponds to a 16-bit value. Figure 6.3 shows an average number of 1286 cluster hits per event. However, the average number of cluster hits is assumed to be 1800 since noise is not included [41]. Therefore, the data payload of an event is about 3600 Byte on average. Given the average input rate of 1 MHz this amounts to 3.6 GByte/s.

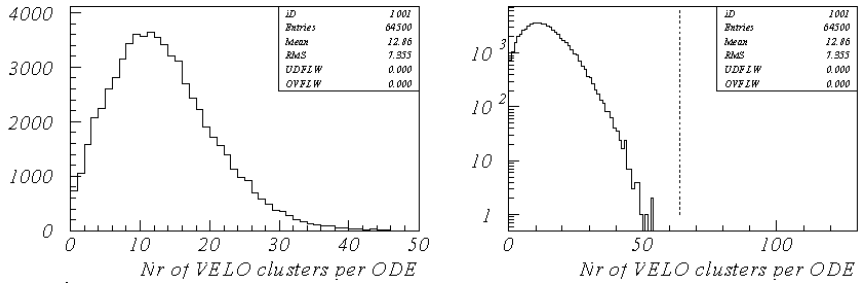

Figure 6.2: Number of VELO clusters per ODE (13 on average). Minimum bias events with Level-0 decision applied have been used as input data for simulation. Noise is not included. Data has been taken from [42].

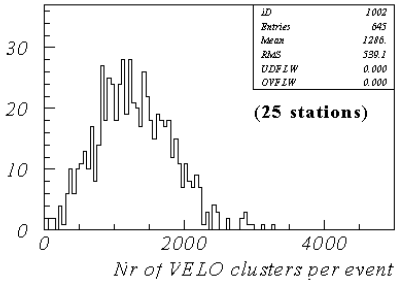

Figure 6.3: Number of VELO clusters per event (1286 on average). Noise is not included. Data has been taken from [42].

One ODE board reads out one of the 100[1] 180° segments of the VELO. The number of ODE boards depends on the detector geometry since one board can read out 2048 detector channels. This amounts to 26 Byte data payload on average per board. Currently three ODE boards are planned to feed one RU, which corresponds to 34 RUs total. Therefore, the data volume that has to be expected adds up to 78 Byte of data payload per RU. Data that is shipped to a CN by the RUs also gets a 4 Byte source identifier per ODE link and an additional 8 Byte overhead per RU. Therefore, every RU has to send additional overhead amounting to 20 Byte. The current Readout Unit implementation is discussed in detail in [17]. About 34 RUs which have to send a data payload of 98 Byte each are required. However, this value depends strongly on the amount of input data. The overall event size is more likely to increase rather than decrease. In addition, overhead has to be

[1]The number is based on 25 stations.

transmitted (see 6.5). Therefore, the system should be able to handle an overall bandwidth of 4.5 GByte/s. The RUs should also be capable of sending up to 256 Byte with a MHz rate.

6.1.2 Compute Nodes (CNs)

Another input required to design the system is the amount of computing needed for the track reconstruction task. There have been various studies on this subject [43, 44]. However, all of them can give only a rough estimate of the computing cycles required, since the algorithms are changing and depend heavily on the event size. It is therefore mandatory to design a system that is scalable in respect of the computational and bandwidth needs of the experiment.

Due to the inherent combinatorial aspect of the tracking the algorithms used up to now show non-linear behavior. The left plot in figure 6.4 shows the dependency of the time it takes to search for 2D tracks on the number of R clusters for $B \rightarrow \pi^+\pi^-$ events. The scattered data points in figure 6.4 are pile-up events which will not be selected by the Level-0 trigger. Pile-up events have more than one primary vertex and need more compute time.

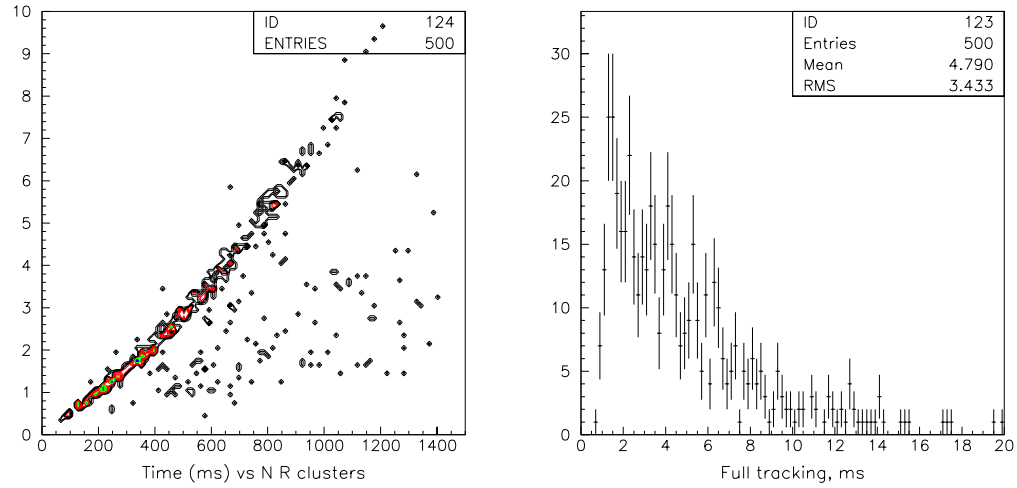

Figure 6.4: Left: Processing time for 2D track search vs. number of R clusters for $B \rightarrow \pi^+\pi^-$. Right: Time for full tracking. The plots have been taken from [45].

An average processing time can be extracted by analyzing a realistic mix of background and physics events. However, at the time of writing only simulation data for physics events has been available — $B \rightarrow \pi^+\pi^-$ events. For those events the time estimated for the entire algorithm including search for 3D tracks is around 4.8 ms on average[45]. This timing analysis is shown on the right of figure 6.4. Minimum bias events are expected to show a very similar distribution. However, taking both improvements to the algorithm and changes to the detector geometry into account the average processing time is believed to be reduced to 2 ms.

Since total event sizes are small the entire event will easily fit into the level 1 cache of commodity PCs. The level 1 cache runs with processor speed and therefore results obtained

for processing times can be scaled by a factor of four to reflect equipment available in three years [46].

The number of nodes is given by the average processing time and the input rate. With the given numbers and an extrapolation of compute power accessible when the experiment starts about 250 twin processor nodes are required. Driven by costs the system should use commodity components wherever possible. PCs are used for computing since they are the most cost effective systems on the market.

Due to the complexity of track finding it has been decided to use a software based approach. However, to speed up the track finding task a hardware preprocessor card could be added if necessary.

6.2 Network Topology and Size

The trigger requires a massive parallel system which has to meet several characteristics. The system should scale easily since the number of network feeds is not final yet. The network architecture has to provide guaranteed delivery in hardware since the CPU load of handling loss of data in software is prohibitive. The RUs have to send event data within 900 ns. The term sending must not be understood as network latency but rather as the time it takes the RU to write a data packet into the NICs input buffer.

Transferring data by using the shared memory concept is advisable, since there is no software overhead involved and a message can be transmitted in a single burst on the PCI bus. SCI has been chosen to be the baseline network technology. The shared memory concept is implemented in hardware, delivery of data packets is guaranteed on the hardware level, and writing data into the SCI NICs input buffer is feasible in less than $1\mu s$. However, the concept presented does not exclusively require SCI but rather a network technology which translates a write to a physical address to a remote memory write operation. A 2D-torus has been chosen to be used as baseline network topology. Network solutions using external switches increases cost dramatically in large networks. Additionally, large switches imply central hot spots since all data has to traverse the internal bus of the switch. A 2D solution is depicted in figure 6.1. Two RUs share the same horizontal ringlet to make better use of the available bandwidth. The SCI link provides a theoretical peak bandwidth of 667 MByte/s. However, one RU sending 128 Byte packets with a MHz rate accounts to 122 MByte/s. Placing two RUs in one row increases the link utilization.

The number of torus rows can be calculated to be 17 assuming 34 RUs as demonstrated in 6.1.1. In section 6.1.2 the number of CNs has been estimated to be 250.

The maximum bandwidth in both horizontal and vertical ringlets will be restricted by the maximum bandwidth in a link segment, B_{Link}. The results in section 5.1 show that a node with its bypass FIFO filled can not send with full rate. Therefore, it is advisable to account for this by defining B_{Link} to be the maximum aggregate bandwidth on a link with no sending node on the ringlet being throttled. Defining B_{tot} to be the total aggregate bandwidth in the Level-1 network farm, the following non-equation must hold true for a column:

$$B_{Link} \geq \frac{B_{tot}}{n_{col}}$$
$$B_{Link} \geq \frac{B_{Link} \times n_{rows}}{n_{col}}$$

with n_{col} being the number of CN columns in the torus. Finally, this yields:

$$n_{rows} \leq n_{col}$$

Taking the numbers calculated above into account it can be assumed that the Level-1 trigger will be composed of 17 rows and 17 columns equipped with CNs.

6.3 Network Traffic

6.3.1 Routing

In a 2D network data can reach a node on various ways depending on the routing strategy used. The Dolphin SCI implementation allows basically two different strategies for routing; driven by a table lookup and driven by a rule based procedure. In both cases the address of the destination node is used. Since tables, node IDs and procedures are configurable the system provides flexibility which allows to implement some degree of fault tolerance. However, the default routing scheme for this application is, as illustrated in figure 6.5, dimensional routing.

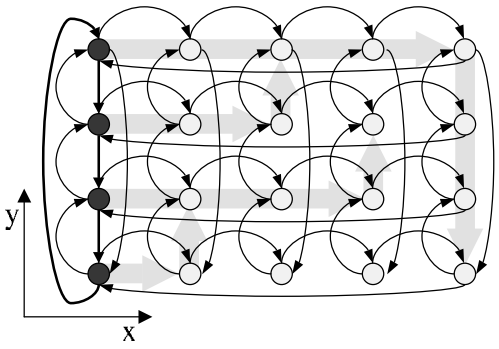

Figure 6.5: Dimensional routing in a 5×4 torus.

Data is first sent along the horizontal ring until it reaches its target column. Eventually, data packets are routed to the vertical ring and reach the destination node. Several concurrent routes, which are in agreement with the scheduling rules are shown.

6.3.2 Scheduled Data Transfer

The boundary conditions of the Level-1 network farm require that an event is sent within a microsecond. However, since the total event size is 3.6 kByte in size taking current

simulations into account, the CN has to cope with a data rate of 3.6 GByte/s if all sub-events arrive at the same time. The event size will most likely grow up to 4.5 kByte giving a data rate of 4.5 GByte/s. However, data must not arrive at the same time since neither event size can be handled by the receiver. Therefore, transfers have to be scheduled.

Scheduling of transfers and sending data are coupled closely. To handle the high rate the data has to be moved from the RUs without software intervention. This is achieved by moving the assembled sub event via a DMA engine from the local buffer via the PCI bus to the SCI network interface card. The DMA engines will be implemented in FPGAs located on the RUs as described in [17]. To control the global traffic pattern generated by the sources all RUs have to be synchronized to a certain degree. This synchronization is implemented with the help of the TagNet, that connects the DMA engines, and the TagNet Scheduler, that assigns events to nodes that are available in such a way that no congestion in the network is possible. There are two separate hot spots in the network. One is the receiving CN that can not handle the data rate. The second source of congestion is traffic directed to one vertical ringlet. The aggregate bandwidth in one vertical ringlet must always be less than B_{Link}. However, the latter is achieved by the scheduler which assigns events to CNs. Therefore, Every node has to send an availability message to the scheduler after an event has been processed. A schematic of the RU logic can be seen in figure 6.6.

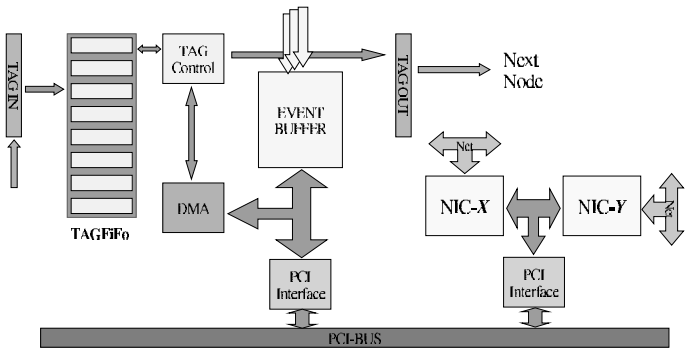

Figure 6.6: Under the control of the TagNet data is moved by a DMA engine from the event buffers to the SCI NIC card without CPU intervention.

The RUs transmit cluster hit information originating from the same event to a selected CPU of the farm. A 14-bit destination ID is distributed to all RUs within the ring. Upon reception of a destination ID a table lookup gives the physical address of the corresponding buffer in the target node and an SCI transaction is initiated. Thereafter the tag is forwarded to the next RU in the TagNet ring with the forwarding scheme being flexible and programmable by a register addressable via PCI. The TagNet is supervised by the TagNet scheduler, which receives sorts and queues node IDs of idle nodes before transmitting them. Additionally, the scheduler monitors the status of the events being processed. However, to prevent data corruption, messages on the TagNet are Hamming encoded to protect the data integrity which is of the uttermost importance in this part of the system. A corrupted target address, e.g., could lead to an event being moved partially to different nodes. A separate TagNet note focusing on implementation issues is prepared by a group at CERN. For more information see [18]. Figure 6.7 shows RUs and the TagNet scheduler as connected by the TagNet.

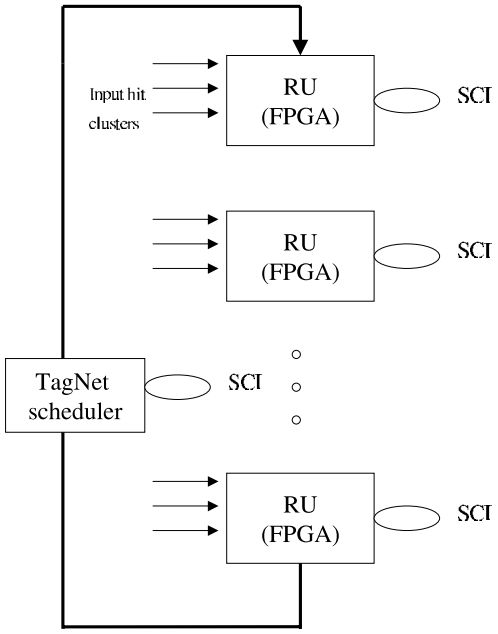

Figure 6.7: The TagNet connecting the RUs and the TagNet Scheduler. Both TagNet slave and TagNet scheduler logic is implemented in FPGAs.

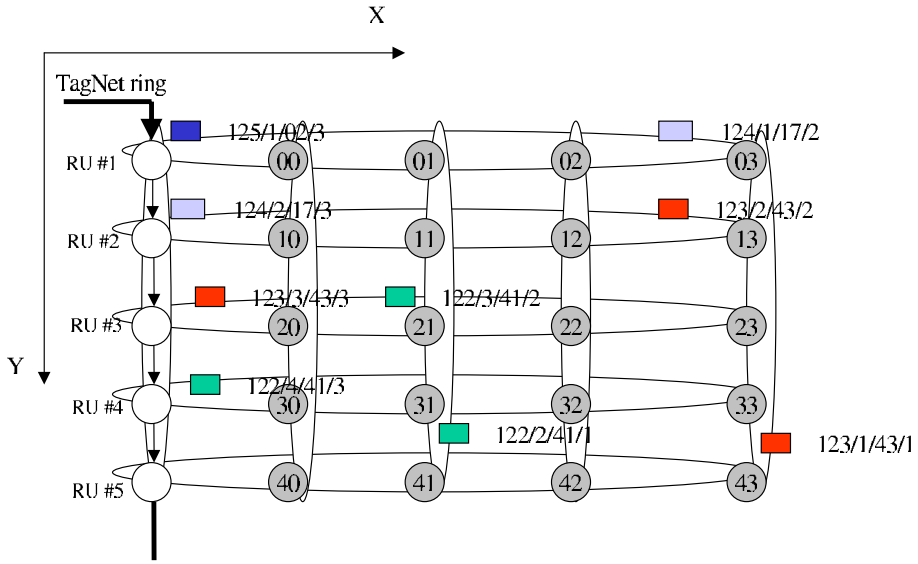

Figure 6.8: Several fractions of a few events moving through a part of the torus. Data belonging to the same event is depicted in the same color. The packets are tagged by event/RU/Dest/timestep giving the ID, source RU, destination node and the time step at which they have entered the network. The scheduler is not shown.

To understand the scheduling principle it is helpful to go through the process of several events passing through the network. In figure 6.8 the location of data packets corresponding to several events are shown at different time steps.

The event number, sending RU, destination node and time step are encoded in the form event/RU/Dest/timestep. Event fractions originating from different events are sent at the same time. However, since under the control of the TagNet sub-events belonging to the same event are not sent at the same time no congestion at the receiving CN is possible.

6.4 Level-1 Decision Unit Interface

For every event processed by the system a result is produced. Therefore, the trigger has an asynchronous output of 1 MHz. This small message of 128-bit in size has to be forwarded to he Level-1 Decision Unit within the allowed maximum latency time. Since the Level-1 Decision Unit expects the events to arrive in order in which they entered the vertex trigger system, the events have to be reordered. The total data traffic originating from this action is negligible compared to the event data stream.

The content of the Level-1 vertex trigger result message is not final yet. The Level-1 Decision Unit Interface must be aware when events have entered the system because the messages have to be forwarded to the Level-1 Decision Unit within the Level-1 maximum latency. This could either happen by receiving Level-0 trigger information or by setting up a dedicated communication between the TagNet scheduler and the Level-1 Decision Unit Interface.

The result messages are written into the Level-1 Decision Unit Interface buffer, which is addressed by the event ID. A read pointer always points to that table entry, which has to be forwarded next. If the message is received within the maximum latency the message is forwarded to the Level-1 Decision Unit. In case of a maximum latency violation a time out message is created. The physical connection between to the Level-1 Decision Unit is S-Link, a protocol engineered at CERN and described in [16]. The node hosting the Level-1 Decision Unit Interface is equipped with an SCI card that allows memory mapped access to the table that has been mentioned above.

6.5 Timing

To illustrate the amount of time that is spent during the different stages of processing it is instructive to look at a specific sample configuration of the system. The timing values used are either taken directly from tests with prototypes, or have been extrapolated from those tests.

As outlined in 6.1 the average event is expected to contain about 1800 cluster hits with noise hits and additional detector information included. A hit is stored in 2 Byte. A fixed amount of overhead is needed for organizing the event. Currently an additional overhead of 20 Byte per RU is assumed. Due to the fact that the data is split into a large number of small fragments this overhead accumulates to 680 Byte on average assuming a system with 34 RUs as data sources.

Data can be sent in packets of 128 Byte, 64 Byte, and 16 Byte on the SCI link. Assuming an average load of 128 Byte per RU one 128 Byte SCI packet will be sent. Therefore, the total expected average event size that has to be transported roughly corresponds to 4.2 kByte. However, since neither the final detector architecture nor the data that is used to feed the track finding algorithm is final data can easily increase up to 4.5 kByte per event on average.

The time to process an event in the system can be divided into the following main phases:

1. The transport phase T_t
2. The processing phase T_p (see 6.1.2)

3. The notification phase T_n

Transport Phase T_t

For a detailed description of the different modules involved in the data flow from the VELO detector to the RUs see chapter 3.

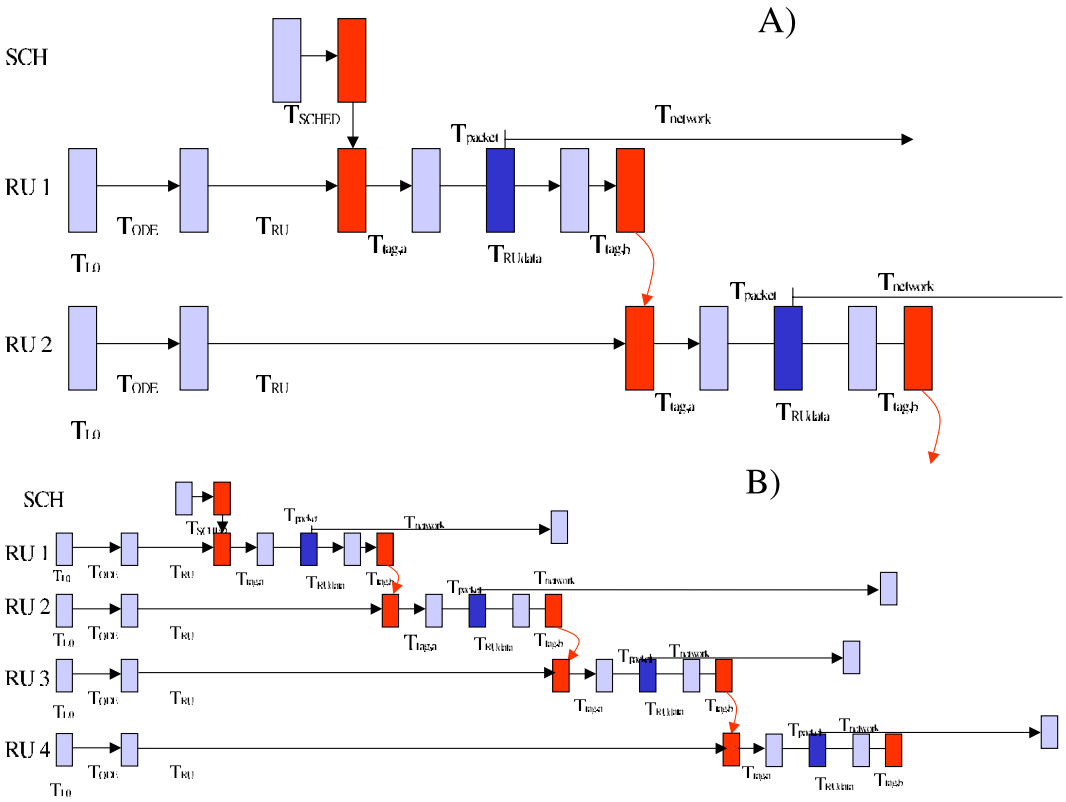

Figure 6.9: A) The sequence of certain operations during the transport phase for the first 2 RUs. Deviating from the text T_{RUdata} has been split into the time to transport the first packet (T_{packet}) and the remaining time. T_{tag} is divided into two parts, T_{taga} for decoding the tag and setting up the remote DMA transfer and T_{tagb} for forwarding the tag. B) Looking at the emerging pattern for 4 RUs the accumulation of the non-overlapping components becomes clear. Since order of delivery is not guaranteed packets might overtake each other.

The transport phase is the time needed for the data to be transferred from the ODEs to the RUs and finally to the buffer in the processing node. For this the following assumptions are based on measurements, simulations and estimates. Figures 6.9A and 6.9B illustrate certain aspects of the timing:

- T_{L0}: Data movement to the RUs starts after the Level-0 trigger signal.
- T_{ODE}: The ODE has to move the data into the RUs input buffer. T_{ODE} roughly contributes $10\mu s$.
- T_{RU}: The sub-event has to be assembled inside the RU and directory structures have to be filled. The event stays in the RUs until it is processed by the DMA engines.

Figure 6.9 shows that T_{RU} depends on the location of the RU within the TagNet chain. T_{RU} can be estimated to be between $4\mu s$ and $38\mu s$ assuming 34 RUs ignoring additional queuing time in case of very large events.

- T_{sched}: After notification by the Level-0 the scheduler assigns the event to a free CN. This will be done in hardware using an FPGA. The scheduler will need less than $1\mu s$ for this.

- T_{tag}: The tag created by the scheduler travels through the TagNet and has to be analyzed by every TagNet client. T_{tag} accounts for the time it takes to analyze the tag, setup the DMA engine, and forward the tag. T_{tag} can be assumed to contribute 150 ns.

- T_{RUdata}: The data has to be transported to the network interface card. This is the sub-event size including headers and overhead divided through the net PCI throughput including any applicable wait states and bus turn around cycles. First measurements have been presented in section 5.2. The upper limit for T_{RUdata} is 900 ns.

- $T_{network}$: Transport through the network. This time depends on the amount of data to be transported and the latency inherent to the network. The network latency depends on the distance between source and target in units of network nodes in between and route nodes that have to be traversed. The latency of the last packet originating from the last RU determines the component of the latency that is not hidden by overlapping transfers. Due to the scheduling of the transfer it can be assumed that all nodes are idle when receiving a packet. Numbers have been presented in section 5.1.4. In addition to this latency the time to transport the data through the network contributes with $4\mu s$/kByte as measurements on the feeding PCI bus have shown.

- T_{queue}: time to wait for the processor to finish the previous event. This time depends on the buffer scheme chosen. In a single buffer setup the event gets processed immediately whilst in a double buffer scenario immediate processing can be delayed by the previous event.

Since events can stay for a non predictable time in the RUs input buffers an exact timing analysis is impossible. An example calculation is shown assuming a $m \times n$ torus and an event size of Es. T_t can be calculated as follows:

$$\begin{aligned} T_t &= T_{L0} + T_{ODE} + T_{RU1} \\ &\quad + n(T_{tag} + T_{RUdata}) \\ &\quad + T_{network} + T_{queue} \end{aligned}$$

For immediate forwarding, two RUs per horizontal ring, and the timing analysis presented

in section 5.1.4 the equation reduces to:

$$\begin{aligned} T_t &= 10\mu s + T_{RU1} \\ &\quad +n * 1.05\mu s \\ &\quad +1.40\mu s + 570ns \\ &\quad +(m+n-4) * 60ns \\ &\quad +\frac{Es}{2n} 4\mu sec/kByte \\ &\quad +T_{queue} \end{aligned}$$

or n = 17, m = 18 and assuming an event of 4.0 kByte in size the largest expected T_t is:

$$T_t = 32.15\mu s + T_{RU1} + T_{queue}$$

Assuming immediate processing and $4\mu s$ for T_{RU1}, T_t amounts to $36.15\mu s$.

Notification Phase

As described above, two messages have to be transmitted during the notification phase. Both originate at the processing nodes after finishing the processing steps. One message, the trigger result, is sent to the Level-1 Decision Unit Interface that sorts the result messages according to event numbers and forwards the message to the Level-1 Decision Unit. The earliest moment at which the front end buffers can be released is after the global Level-1 has distributed the decision in case of a reject and after the data has been transmitted out of the front-end buffers in case of an accept. This has to happen within the maximum latency time. The second message, that has to be sent, indicates that the node is ready to accept another event. This has to be sent to the scheduler. This message is sent outside the latency window and thus the time this transaction takes is only of concern for the efficiency but not for the latency. Both messages can be assumed to be transferred in less than $5\mu s$. However, if both entities are implemented in the same node the messages can be combined.

Chapter 7

The Level-1 Trigger Prototype

A prototype of the Level-1 vertex trigger system has been implemented. The system is capable to send small messages with a rate of more than 1 MHz as required by the LHCb experiment. However, the TagNet version implemented is very basic and only allows static assignment of compute nodes. The system comprises three RUs, 26 CNs, and the Level-1 Decision Unit Interface which amounts to 30 nodes total. Mockup data has been analyzed in the receiving CN. A packet loss has never been detected. However, since the system is based on a standard Linux distribution the analyzing process might be suspended such that events could be missed. A frequent occurring reason are interrupts. However, an overall system analysis shows that almost 100% of the events are analyzed.

There are two important parameters that go directly into the system architecture — the maximum link bandwidth and the maximum bandwidth that the B-Link can handle. Both are presented. B_{Link} has been determined to be 432 MByte/s (75% of maximum SCI net bandwidth) for a system that does not use displaced RUs. For a system using displaced RUs B_{Link} increases at least to 478 MByte/s which is 83% of the maximum SCI net bandwidth. The maximum B-Link bandwidth has been determined to be 450 MByte/s (88% of maximum B-Link net bandwidth).

7.1 Baseline Architecture

The concept presented in chapter 6 has been implemented as basic version which does not include the TagNet scheduler. The TagNet has to fulfill two different tasks. Traffic shaping prevents network congestion and can be implemented by using one bit. Load balancing and resource management require full system information since no buffer must be overwritten before the previous buffer content has been analyzed. Therefore, the TagNet scheduler is of importance for the latter whereas a prototype showing a MHz performance without network congestion can be realized using a simpler solution. In fact, for the current prototype it is assumed that the time for event analysis does not differ and therefore event buffers can be overwritten after a certain amount of time. The shared memory regions that have to be addressed by the network feeds are not chosen dynamically. A static

setting has been implemented with target addresses being used in a fixed order. At the time of writing the Level-1 track finding algorithm is optimized for efficiency and not used in this implementation. Therefore, every compute node runs a process that checks for data packet loss prior to sending a message to the Level-1 Decision Unit Interface. As outlined in chapter 6 the Level-1 trigger will be a real-time cluster of about 300 twin processor nodes. In order to demonstrate 10% of the full scale system, a 30 node Linux cluster is used to demonstrate the basic concept. Figure 7.1 shows a picture of the cluster nodes.

Figure 7.1: 30 nodes out of 32 Linux computers are connected to a 10×3 torus. The picture on the right shows the SCI cabling.

24 nodes are equipped with ServerWorks IIIHE-SL , the remaining 6 nodes with ServerWorks IIIHE chipsets [33]. Both chipsets support fast 64-bit/66 MHz PCI buses, which is essential for fast I/O via the PCI local bus. All nodes are equipped with two Pentium III processors. 24 run 800 MHz CPUs, 6 of them 733 MHz processors. Currently the nodes are equipped with 512 MByte RAM, a current SuSE Linux distribution, an up-to-date kernel version, and sufficient disk space (40 GByte). All compute nodes utilize Dolphin PCI 64/66 PCI SCI cards built around the SCI link controller chip LC3 [30] located in a 64-bit/66 MHz PCI slot. The nodes are connected, as depicted in figure 7.2, forming a two-dimensional torus. However, the figure does not reflect the real cabling since long cable lengths should be avoided. In the real setup the nodes are preferribly connected interlaced. Three nodes serve as input feeds (RUs) and send data packets using hardware initiated DMA as described in section 5.2. Currently 26 nodes receive data packets from the sending nodes. Those nodes are referred to as compute nodes in the following sections. Upon reception of the data packets coming from the RUs, a 128-bit result message is sent to the result node, the L1DU Interface (Level-1 Decision Unit Interface). Sending the data to the CNs is synchronized by a basic TagNet implementation. However, the output produced by the compute nodes is not synchronized and not necessarily in order. All results presented in this chapter refer to this setup unless stated otherwise.

Input feeds need additional logic to implement the DMA engine and the TagNet logic. Altera Apex PCI boards[1] are used in the current prototype since no sufficient amount of RUs as planned for the experiment is available. However, to use the terminology introduced in chapter 6 the input feeds are termed RUs. The Altera board is located on the same PCI bus as the Dolphin SCI card. Thus, the data can be written into the

[1] based on the APEX20KE FPGA

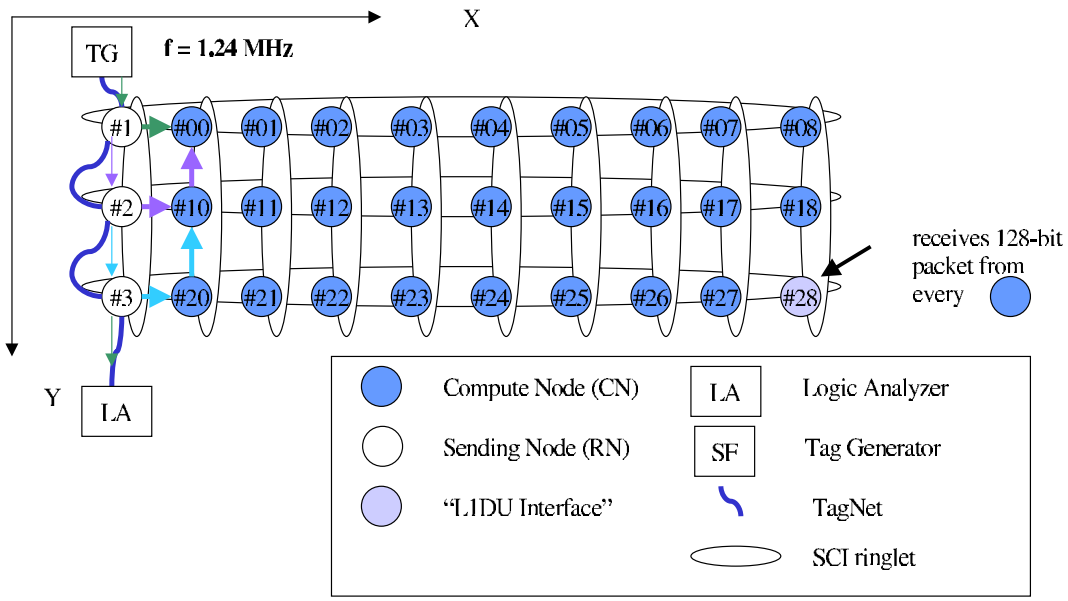

Figure 7.2: This figure shows the current setup in Heidelberg. 30 nodes are connected by the Scalable Coherent Interface and form a two-dimensional torus topology. The three nodes to the left are mockup input feeds connected by a basic implementation of the scheduling network TagNet. The node labeled L1DU Interface receives 128-bit result messages from every compute node.

SCI card directly without crossing a PCI bridge or the hostbridge. Major characteristics of the logic implemented are a PCI interface to the 64-bit/66 MHz PCI bus, an internal descriptor buffer, a mockup data generator, and the DMA engine, which sends data to the physical address specified by the entries in the descriptor buffer.

Currently x-y routing is used with y and x-axis depicted in the figure.

7.2 Network Feeds

7.2.1 Data Scheduling and TagNet Prototype

A basic TagNet without a scheduler has been implemented. This is sufficient for current tests focusing on system performance as outlined in 7.1. The sending nodes are connected by the scheduling network as depicted on the left of figure 7.2. The implementation uses 1-bit signals and does not have an input queue. However, this is sufficient to build a prototype running at 1 MHz which has been considered to be the most crucial part of the system. A handshake signal is required to avoid loss of tags. Additionally, the signal is used to monitor the system utilization. This busy logic has been implemented for the prototype only and will be replaced by an input queue in future implementations. Since the upcoming implementations are managed by the TagNet scheduler overflow of TagNet queues is avoided.

However, the current version is simple and sufficient to study the network traffic and performance of an SCI cluster which is fed at 1 MHz.

Since the order in which the CNs are addressed is fixed the tag is a single pulse latched by the tag receiving node on reception. Four signals have been implemented. Naming

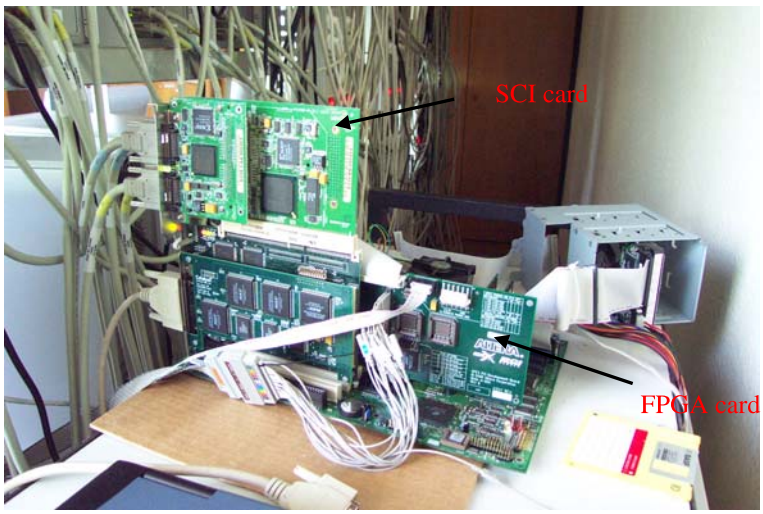

Figure 7.3: The picture shows the setup for a mockup RU as used in the system presented.

conventions are introduced in table 7.1.

Table 7.1: TagNet signals, direction, and meaning.

Signal	Direction	Description
TIn *xx*	IN	tag in
TOn *xx*	OUT	tag out
BIn *xx*	IN	busy in
BOn *xx*	OUT	busy out

The parameter *xx* is either 01,01 or 02 in the current implementation. The signals are also depicted in figure 7.4. All signals are negative logic. Since the FPGAs[2] provide 16 RX and TX LVDS channels the four signals have been implemented using one channel each. Since no clock signal is transmitted, and therefore the clocks of transmitter and receiver are not synchronized, the tag line is asserted for 4 clock cycles. Given a design clock frequency of 66 MHz this time period is more than sufficient to satisfy setup and hold time requirements of the tag receiving register. A schematic view of the current TagNet chain can be seen in figure 7.4.

RU#1 (see figure 7.2) receives a tag and starts its DMA transfer. Besides the TIn and TOn lines a busy logic has been implemented such, that a DMA engine signals its busy state to the prior node in the TagNet chain. After a node has finished its DMA transfer it forwards the tag to the downstream node, which pushes its data to the same remote node as the one before. This produces the same traffic pattern as shown in figures 6.8 and 6.9. However, forwarding a tag is only possible if the adjacent node does not assert its busy signal. Upon tag reception a node asserts its busy line to signal an ongoing transfer. Every node has a pipelined output that allows to store and forward one tag while a new transfer is already processed. If the output queue is full the BOn signal is not deasserted

[2] Altera APEX20K400E

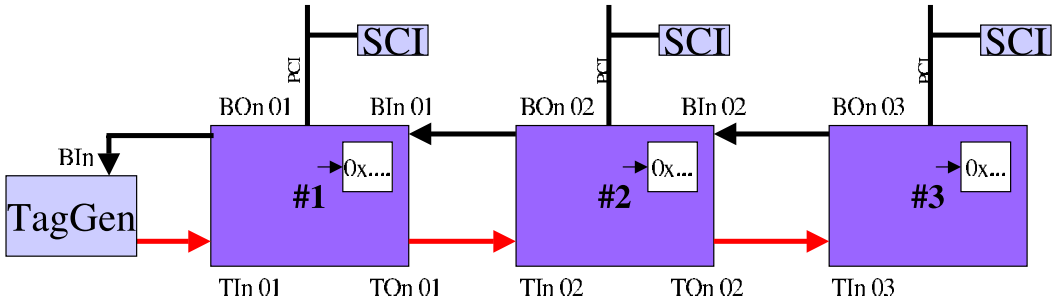

Figure 7.4: The three sending nodes (RUs) also depicted on the left in figure 7.2 are connected by the TagNet links. The node numbers refer to the numbers introduced in figure 7.2

immediately. A state machine diagram as depicted in figure 7.5 shows the four states of the TagNet implementation.

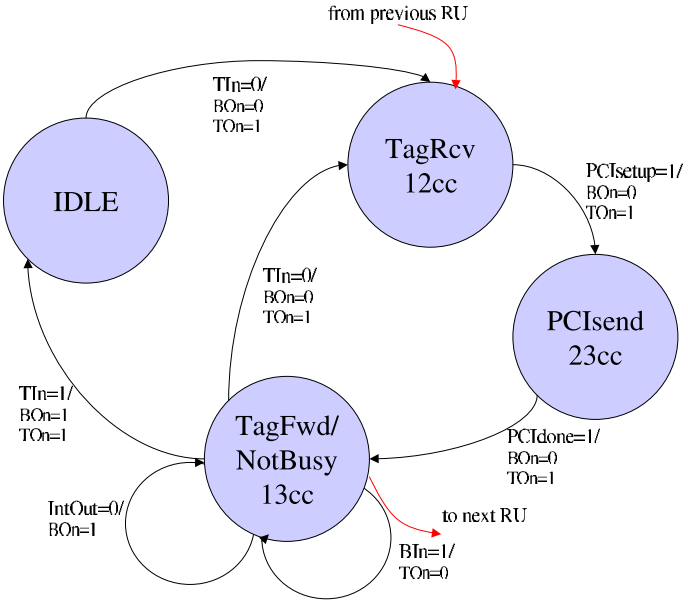

Figure 7.5: Finite state machine of the TagNet logic. Using the terms introduced in 6.5, T_{taga} accounts to 12 clock cycles and T_{tagb} to 13 clock cycles. The time spent in the PCIsend state is identical to T_{RUdata} with its duration being defined by the assertion period of the PCI FRAME# signal. Only when the internal output buffer is empty (IntOut=0) the BOn signal is deasserted, thus avoiding buffer overflow.

State transitions depend on the status of the external TIn and BIn and the internal PCIdone, IntOut, and PCIsetup signals. PCIdone signals that a PCI transaction has been finished successfully. After forwarding the tag the system can either return into its idle state or start a new PCI transaction upon reception of an external TIn pulse. The time in terms of clock cycles has been measured for the states. Tag reception and PCI transfer do not overlap and account to 35 clock cycles for a 128 Byte transfer. The internal latency of the FPGA and tag forwarding do overlap and account to a total of 13 clock cycles. Figure 5.11 shows HDMA results which have not been controlled by the TagNet. The given TagNet implementation applies an additional overhead of 14 clock cycles per transfer.

The system is stimulated by an external LVDS tag generator implemented in an FPGA. The stimulation frequency can be programmed by writing into a CSR register accessible from PCI. If the first node in the TagNet chain deasserts its busy out signal it receives a tag from the external tag generator. The TagNet signals are analyzed with a logic analyzer. The resulting picture can be seen in figure 7.6.

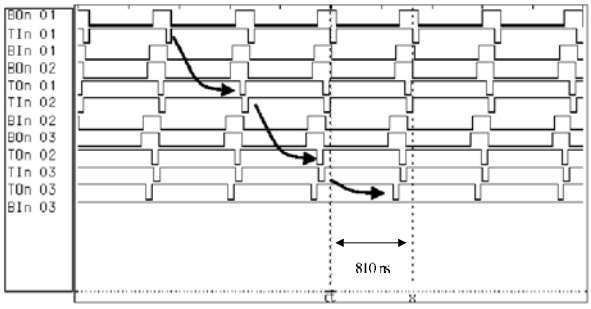

Figure 7.6: This logic analyzer screen shot shows the tag flow as measured with three RUs. The first RU receives a tag (TIn 01), asserts its busy signal (BOn 01) during an ongoing transfer, and forwards the tag to the adjacent node (TOn 01) after the PCI transaction has been finished. The nodes are connected as depicted in figure 7.4

The logic analyzer screen shot shows 128 Byte PCI transactions happening at a rate of 1.24 MHz. Whenever a tag is received a PCI transaction is initiated. The tag flow is depicted by arrows. For this setup the BIn signal of the last node in the TagNet chain is statically deasserted such that the last node can always forward its tag.

The busy logic allows to monitor the system utilization during run time. For highest performance BOn must be deasserted as briefly as possible. In the best of all cases a new tag is received one clock cycle after BOn has been deasserted. The screen shot in figure 7.6 shows a system running not at full speed.

7.2.2 DMA Logic

As outlined in 5.2 a DMA engine has been implemented in an FPGA, which allows to push data to a specified physical address. A schematic of the design can be seen in figure 7.7.

The major building blocks are the PCI interface, which is provided by Altera, the master interface, the target interface, and the DMA engine module itself. The figure also shows the TagNet logic component, which has been described in 7.2.1. Master and target interfaces are state machines that manage PCI master and target transactions, respectively. The design is capable of performing PCI target memory read/write and PCI master memory write transactions. There has been no need to implement master read functionality or transactions concerning PCI I/O space. An FPGA configuration file is created by using the Quartus II [47] software provided by Altera [36]. A screen shot of the compilation process is depicted in figure 7.8.

Each individual DMA transfer is characterized by a descriptor which is shown in figure 7.9. One descriptor holds four 32-bit entries which are target address, blocksize, and two unused fields for future use. Hence, one descriptor uses 16 Byte of buffer space.

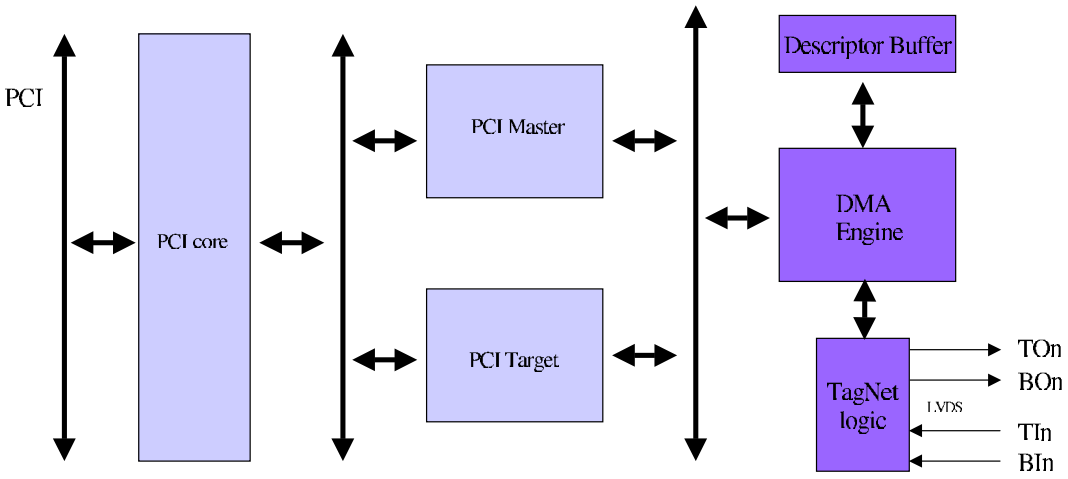

Figure 7.7: Schematic of DMA logic.

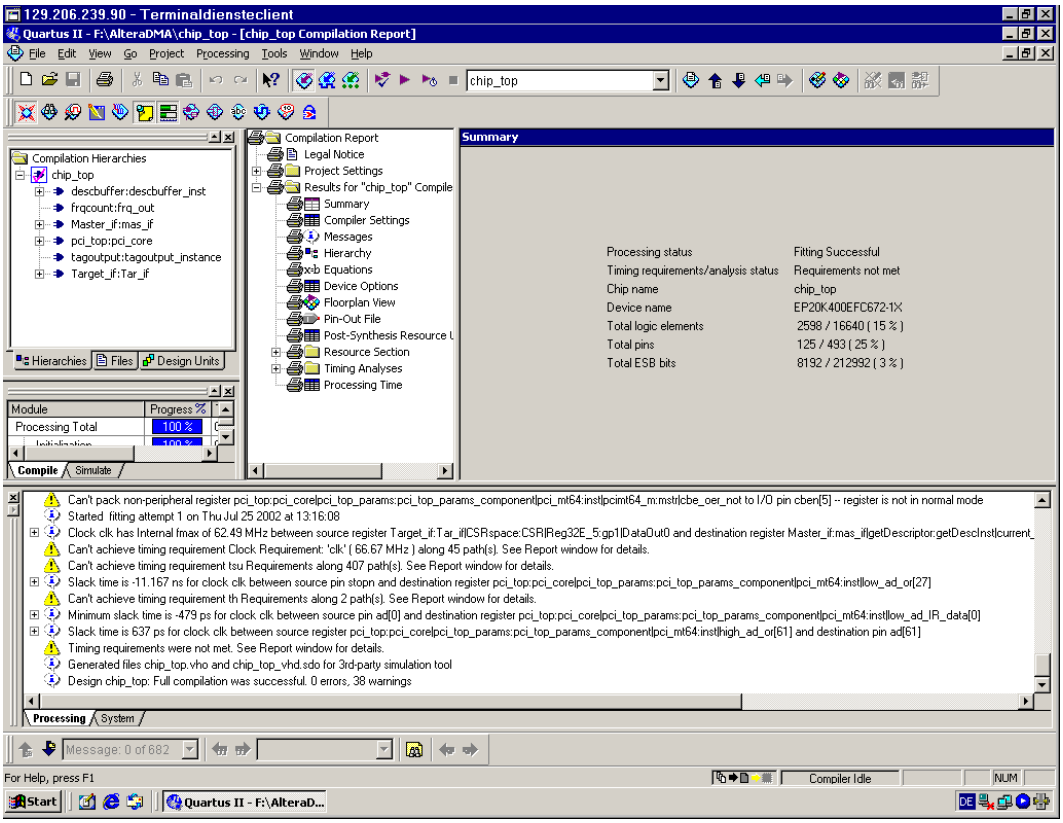

Figure 7.8: Output of the design software as provided by Altera. It reveals that only 15% of the logic is used.

The field target address holds a 32-bit PCI address whilst the entry blocksize holds the data block size to be transferred in units of PCI data cycles, which means that an entry of 0x4, e.g., is equivalent to a block size of 32 Byte on a 64-bit PCI bus. Two additional fields have been implemented. One field is foreseen to hold the offset into the data buffer where event data is stored. However, since mockup data is sent at the moment this specific

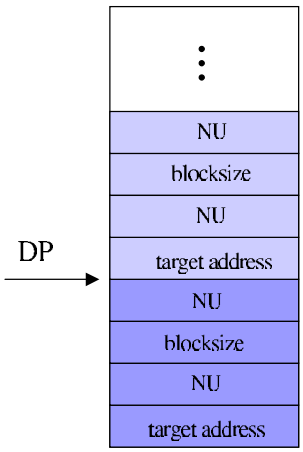

Figure 7.9: Layout of the descriptor buffer. Up to 64 different descriptors can be stored at this point which is sufficient for the current system size. After each transfer the descriptor pointer (DP) is incremented.

entry is also unused.

Each DMA transaction can be triggered by either a tag received by the TagNet TIn signal or a software trigger issued by a local process. DMA transactions are programmable in means of number of transactions per trigger and number of descriptors that are actually stored in the descriptor buffer. One trigger can cause anything in between one and an infinite number of DMA transactions. However, the maximum number of descriptors has been limited to 64, which is sufficient for the current test environment. After each transfer a descriptor pointer as shown in figure 7.9 is incremented such that it points to the target address field of the following descriptor. Software triggering has been used for debugging and benchmarking purposes as presented in 5.2.

In case of the Level-1 prototype one descriptor is read from the descriptor buffer on tag reception. The PCI master state machine requests the PCI bus and becomes bus master. A PCI burst, which is characterized by the entries of the descriptor, is directed towards the SCI card located in the same bus. On reception of the data the SCI card issues a remote data transfer into a memory region, which has been exported by the remote node in an initialization phase. After the PCI transfer has been finished the descriptor pointer is incremented and the tag forwarded. The latter is only done immediately if the BOn line of the downstream node is not asserted.

The descriptor buffer is a memory mapped region and thus can be accessed by PCI read and write memory cycles. Additionally four 32-bit registers accessible from PCI have been implemented. The registers are used to configure the device and to get information used for trouble shooting and status.

Table 7.2: Address space of DMA logic.

Region	Offset	Name	Access	Size
BAR0	0x100	DMACNTRL	R/W	32-bit
BAR0	0x200	FRQ	R/W	32-bit
BAR0	0x300	DESCS	R/W	32-bit
BAR0	0x400	TAGS	R	32-bit
BAR1	0x0	DESCBUF	R/W	256 × 32-bit

- **DMACNTRL**:

Bit field	Name	Description
31:11	Transfers	Number of transfers
10:8	Resv1	Reserved
7:6	Mode	Transfer Mode
5	TagNetEn	Enable DMA triggered by tag
4	Resv2	Reserved
3	Clear	General clear
2	Resv3	Reserved
1	Status	DMA status
0	Start	Start DMA transaction

DMACNTRL.Transfers: DMACNTRL.Transfers is the number of DMA transfers, which is processed after reception of a tag or a software trigger. In case of the Level-1 prototype this bit field is set to one.

DMACNTRL.Mode:

The design can be programmed in terms of how many transfers it should issue per trigger. In case of the Level-1 trigger prototype only one transfer is initiated per tag. However, for trouble shooting and stability various settings are possible:

0X: As many DMA transfers are processed as shown in DMACNTRL.Transfers. However, the maximum number is DESCS + 1.

11: As many transactions are performed as set in DMACNTRL.Transfers regardless of DESCS. If DMACNTRL.Transfers is greater than DESCS + 1 the descriptor buffer gets processed again from the beginning. This setting has been implemented if number of desired transfers is greater than the number of descriptors that can be stored.

10: This mode has been used for stability tests. The logic loops through the descriptor buffer contents until it is stopped by software.

DMACNTRL.TagNetEn: The design can be stimulated both by an external signal and by software. This bit enables an external stimulation of the DMA engine by the TagNet.

DMACNTRL.Clear: The DMACNTRL.Clear bit sets the descriptor pointer and the TagID as shown in 7.13 to zero.

DMACNTRL.Status: The status of the design can be monitored by reading a status bit. On completion of a transaction the bit is set.

DMACNTRL.Start: The start bit is set by software process and is redundant if the DMA engine is triggered externally. It initiates a DMA transfer as specified by the entries in the descriptor buffer.

- **FRQ**: The tag generator has been implemented in the same design. However, only the first client in a TagNet chain is connected to the output of the generator. The frequency of the tag can be set by writing to this register. The tag generator is shown in figure 7.4.

- **DESCS**: Not all entries of the descriptor buffer have to be occupied. The value in this register is the actual number of descriptors − 1 that are stored.

- **TAGS**: Every TagNet client can be checked with respect to the number of transfers it has issued so far. This register has been very helpful for trouble shooting since the number of processed tags has to be identical for every TagNet client.

- **DESCBUF**: Memory mapped descriptor buffer, which can store 64 descriptors in its current implementation as shown in figure 7.9.

Software Interface

To access both registers and the descriptor buffer from PCI a software library has been used [48]. PCI commands like memory read and write can be used from user space. As outlined in 4.4.3 the SISCI library provided by Dolphin allows to import remote memory regions. However, the addresses retrieved are virtual addresses and can not be used by the DMA logic directly. To translate virtual addresses into their physical counterpart a function callable from user space has been implemented into a Linux kernel module.

To initialize the device to be used correctly in Level-1 trigger prototype mode the following steps have to be executed.

- First the system is reset by DMACNTRL.Clear. This is done every time the system is used to stop any potential ongoing transfers.

- The physical addresses of the imported memory regions and the size of the data to be transferred is written to DESCBUF. Since the SISCI API delivers a virtual address, the physical address has to be acquired by a call which has been implemented in a Linux kernel module [48]. More details can be found in the appendix A.3.

- DESCS has to be initialized depending on how many descriptors should be used. In case of a 26 node cluster having two data buffers each, this number would be 52.

- The DMA engine has to be programmed to initiate one transfer per incoming trigger in case of the LHCb Level-1 trigger prototype. This is done by using the DMACNTRL register.

7.3 Global Shared Memory

If compute nodes want to exchange data using the shared memory concept a memory region accessible from all participating nodes is setup. No difference to common memory operations can be seen since user space processes use virtual addresses to access remote memory. Thus, a totally transparent programming model is utilized. In our environment all nodes used as data receivers export m physical memory regions of s kByte each into the global SCI address space. Both m and s are variable and can be set on startup. Every RU imports all the regions and therefore will end up with a total shared memory chunk of $m \times s$ kByte per CN and $m \times N$ addresses with N being the number of compute nodes. N is limited by the maximum amount of imports which is 16k in the current implementation of the PSB66.

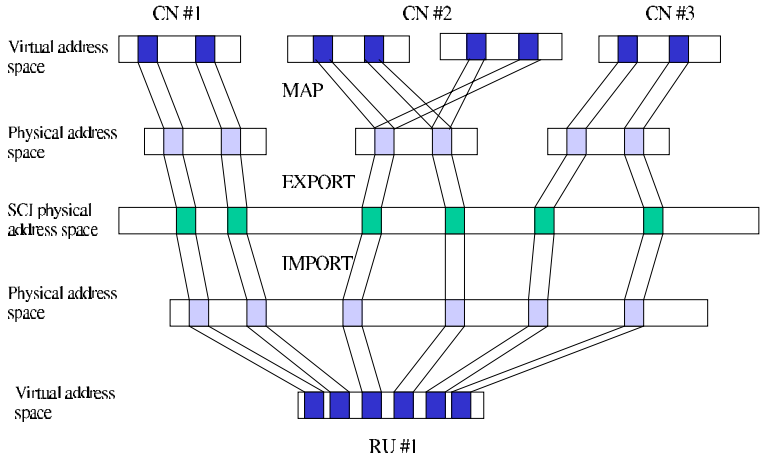

Figure 7.10: Every single compute node can export m chunks of memory. The figure shows three compute nodes exporting two memory regions each as an example. Afterwards the RUs import all exported regions. CN #2 runs two processes which map the memory regions into their virtual address space to have access to data sent via SCI.

Figure 7.10 shows three nodes exporting two physical memory regions each into the global SCI memory space and one RU importing the regions afterwards. The import/export functionality has been implemented in C using the SISCI API, a programming interface provided by Dolphin. When a region is imported and mapped to user space, a virtual address is returned to the calling process. However, to use this address with hardware devices the virtual address must be translated into its physical counterpart. Every RU runs a process that imports the regions at system initialization time. The virtual addresses are stored in an array and translated to physical addresses one after the other. After the address translation has been finished every RU holds an array of physical addresses pointing to remote memory regions. In the current test environment every compute node exports two memory regions of 4 kByte each which is sufficient for the event sizes used in the prototype. As depicted in figure 7.2 one node is used to function as the Level-1 Decision Unit Interface, which receives the 128-bit result messages. Thus, in the largest possible setup 26 compute nodes are used.

All physical addresses are transferred to the descriptor buffer of the DMA engine via the PCI bus. Besides the addresses the data size to be transferred has to be stored. Figure

7.11 shows the content of some descriptor buffer entries after initialization time.

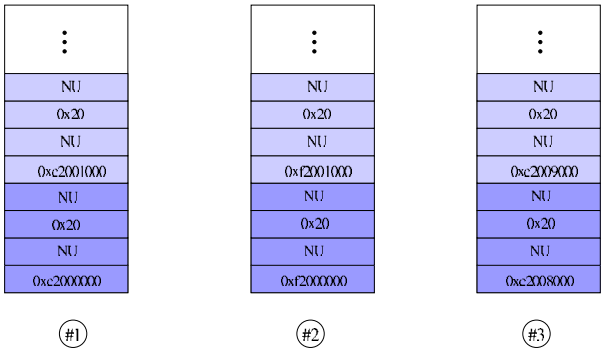

Figure 7.11: During the initialization phase every RU imports the memory regions exported by the compute nodes. This is done by software and results in 52 virtual addresses when 26 CNs export two memory regions each. After those addresses have been translated to physical addresses they are transferred to the DMA engines' descriptor buffer located in the FPGA. The figure shows a snapshot of some descriptor buffer entries after initialization. Descriptor entries labeled NU are not in use.

Every descriptor holds four 32-bit data words; the address, the data transfer size in units of 4 Byte, and two fields that are currently not in use. If node #1 sends to address 0xc2000000, e.g., a remote data transfer of size 128 Byte via the SCI interface card is initiated.

7.4 Data Transfer

7.4.1 Transfer Order

The current implementation uses a static address buffer since the scheduler has not been implemented yet. Thus, the order, in which CNs are addressed is determined by the order of addresses stored in the descriptor buffer. Memory regions are imported and their base addresses stored in the descriptor buffer row by row starting from the upper left corner of the torus sketched in figure 7.2. Since every CN exports two memory regions, the second region is filled in a second iteration. Figure 7.12 gives an overview of the final order of base addresses in the descriptor buffer.

Once the system starts transferring data RU #1 will transfer its data to CN #1 buffer #1. After the transfer has been finished RU #2 will transfer the second data chunk to that compute node. In the meantime RU #1 has received a new trigger command and transfers data with the target being in the second column of the torus. Thus, after three trigger signals have been issued a simultaneous transfer can be observed. A compute node is addressed again after N trigger signals with N being the number of compute nodes.

7.4.2 Data Format

The data, which is transferred in a PCI burst, is variable in size and can be set during the initialization phase. If *datasize* indicates the size of the data block measured in byte and *baseaddress* marks the address of the remote buffer as stored in the descriptor buffer

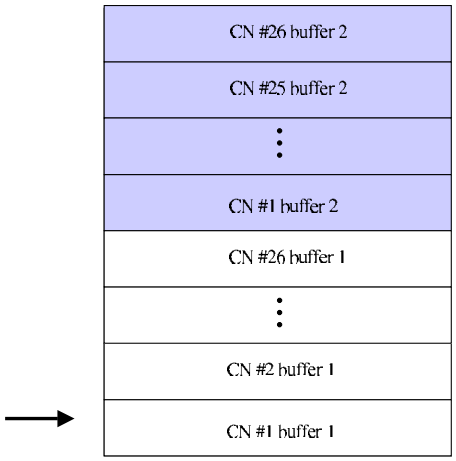

Figure 7.12: Since no scheduler has been implemented so far the order in which remote nodes are accessed is set by the order of descriptors. The figure shows the descriptor order in a buffer. The read pointer is depicted to the left. It is incremented every time a transfer has been finished.

of RU#1, the RUs write to the buffer sequentially starting at *baseaddress*. Thus, for *datasize* equal to 128 Byte the write offsets are set to 0 Byte, 128 Byte, and 256 Byte for RUs #1, #2, and #3, respectively.

The data being transferred are mockup data, which hold an identifier unique to a transferred event. The tag ID is incremented for every new event whilst the index field is incremented every PCI data cycle. Thus, every data word holds an identifier, which is unique during a certain time frame, e.g. more than $53ms$ in case of a 16-bit wide counter and a transfer rate of 1.24 MHz. This simplifies troubleshooting in case of problems. For 128 Byte bursts 16 64-bit data words are transferred. Figure 7.13 shows a data word as transferred in one data cycle on the PCI bus.

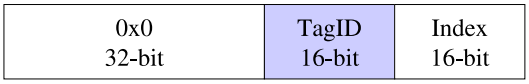

Figure 7.13: A 64-bit mockup data word as transferred every PCI data cycle. The least significant bit is to the right.

7.5 Data Integrity

The current TagNet loop is not closed since the CNs do not report back to the TagNet scheduler after processing an event. Therefore, a statistical proof of event reception has been chosen. A data integrity process running on every node checks the incoming data and creates some statistics. However, a standard Linux distribution is used which does not guarantee that the process is active all the time. The process can be suspended because of context switches or interrupts that have to be processed.

Data is written into the host memory of the compute nodes and is overwritten again after $m \times N$ transfers. The time this takes depends on the sending frequency of the RUs and the number of event buffers in the system. For the results presented here the setup chosen comprises 26 CNs having two event buffers each and three RUs sending a data packet every 810 ns. Since data transfer happens in a round-robin scheme it can be calculated that

every buffer is overwritten after $42.12\mu s$ ($m = 2$). The time period in between memory accesses must be used to check if all packets have arrived.

As explained above every compute node receives three packets coming from different sources. The current test environment focuses on sending 128 Byte packets, which is equivalent to the SCI packet size. Therefore, checking one 32-bit word every packet is sufficient to check for lost data packets. Every compute node exports two memory regions, event buffer 1 and event buffer 2, which are addressed by the sending nodes. After data has been written to buffer 1 of a node a certain time period passes by where all other nodes receive their data packets. When the node is addressed again data is written into the second buffer, buffer 2. This time is $21\mu s$ in the current setup. Figure 7.14 shows a data flow diagram of the algorithm which checks for data integrity.

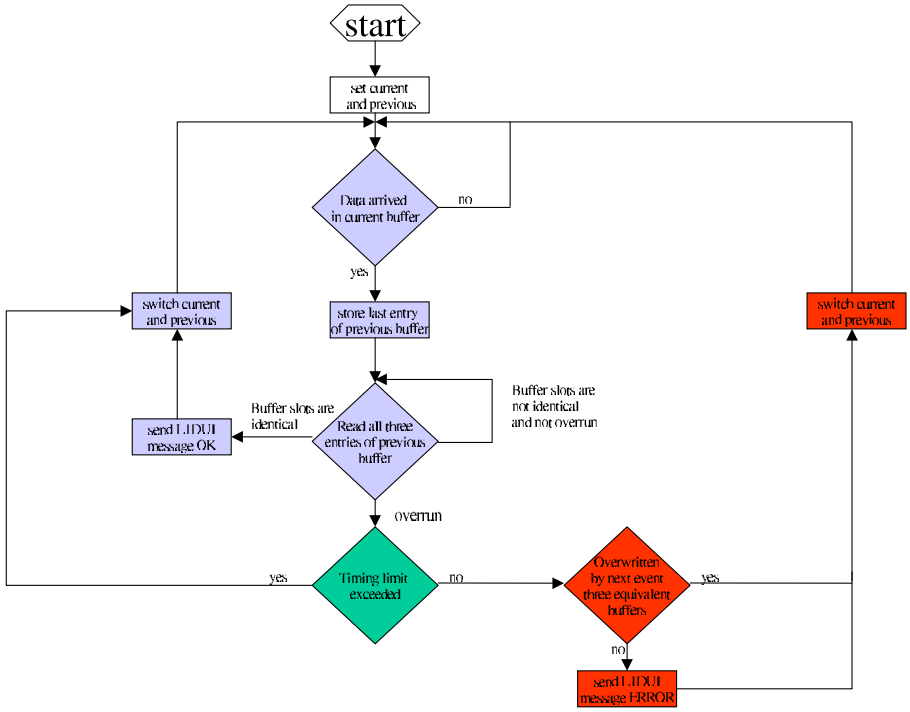

Figure 7.14: Flow diagram of the algorithm checking the incoming data packets.

The first data packet residing at offset zero in one of the buffers is analyzed by reading its first 32-bit word. As depicted in figure 7.13 the less significant 32-bit of a 64-bit word hold the tag ID and the index. Thus, the data transferred by the other two data packets can be compared easily by comparing the tag ID of all three data packets. If the buffer has been checked the process continues to check the other buffer.

If all three buffer slots show identical contents a 128-bit message is sent to the Level-1 Decision Unit Interface and the other buffer is analyzed. This is depicted by the blue (left) part in the data flow diagram. Since the tag IDs are temporarily stored until the next iteration is processed the algorithm can check if the new data received is the next that is expected. Under normal conditions the next data set analyzed has always been the expected one.

The current implementation waits for data to arrive in one buffer but checks the other one. The other data buffer must have new data by that time. However, if the buffers

are checked using this scheme, this has to happen within the $21\mu s$. There are two events that could detain the algorithm from its job and therefore induce wrong interpretation of results. One is process scheduling. It has been determined that the Dolphin device driver accesses the SCI card every second and thus causes a context switch. However, features like that can be turned off by manipulating the device driver. The second reason of process suspension are interrupts. Since standard Linux distributions have been used on the CNs the system clock which is based on periodic hardware interrupts is updated every 10 ms. Thus, the operating system has to process 100 interrupts every second. Every interrupt means suspension of the running process, interrupt handling, and eventually running the process again. This accounts to a total estimated minimum suspension time of $15\mu s$ for every interrupt processed. The number is based on the findings outlined in 4.3.2. Additionally, the system is equipped with an Ethernet card which creates an interrupt whenever a packet is received.

The algorithm reports mismatches in buffer slot contents by writing to a log file. However, a second method of debugging has been used. Whenever a situation occurs which is not understood a specific rare pattern is written to one of the sending nodes. Both nodes are equipped with PCI analyzers and trigger on the defined pattern. Therefore, the PCI trace on RU and CN can be analyzed to check for lost packets. However, in all cases that have been analyzed so far the reason of error messages has been the CN itself. Caused by a context switch or an interrupt data can not be checked and is eventually already overwritten when the process is running again.

Thus, situations that are understood but deliver buffer mismatches are ignored and no message is sent to the Level-1 Decision Unit Interface. However, if situations occur which seem to need further investigation the process branches into the right part of the flow diagram depicted in figure 7.14. Being in that part of the flow diagram will trigger PCI analyzers and produce an entry to a log file.

Figure 7.15 shows the summarized results after the system has been running for 5 minutes.

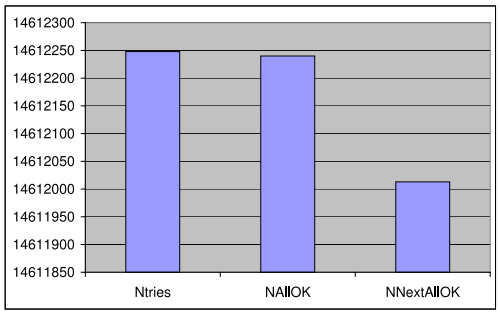

Figure 7.15: Result of a 5 min test run. The algorithm tested the buffers *Ntries* times, reported 8 times that the buffer shows data mismatches, and missed a total of 237 events.

The histogram resembles the log file written on one various picked CN. The algorithm looped 14612248 times (Ntries), reported that all buffer slots contain valid data for all events besides eight — NAllOK is 14612240. NAllOK is incremented whenever the buffer slots are identical which is depicted by the blue left part in figure 7.14. However, 237 events have been missed since NNextAllOK equals 14612013. NNextAllOK is incremented whenever the three buffer slots are identical and additionally contain the data expected next. Running for five minutes caused 30000 interrupts by the system clock. Therefore,

the process has been suspended for 450 ms total. Most of the times the process has been running to poll for new data to arrive. However, sometimes the process gets suspended in its critical phase. The critical phase is defined as the portion of the code which reads and compares the three buffer slots. During process suspension the data buffer might be overwritten by the following event.

Figure 7.16 shows the amount of time it takes until an event is checked. Especially the tails in the histogram account for suspension time caused by interrupts. The number of events that take more than $5\mu s$ to be checked accounts to less than 6500 for this specific example.

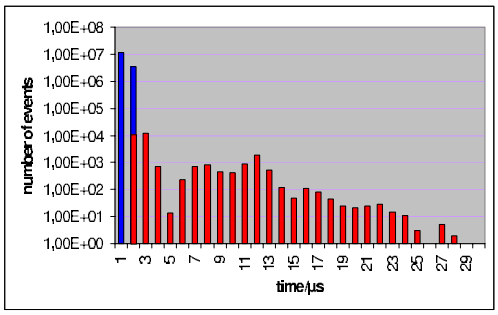

Figure 7.16: A time histogram shows that in 76% of all loops the event data is successfully checked within $1\mu s$. After $2\mu s$ 99.87% of all data is checked successfully. The part of the events being highlighted in red corresponds to 30000 events.

The result shows that all data packets are delivered and almost 100% of all events can be checked by the process. This is possible without using a real-time operating system. Given the overall Level-1 latency of 1.6 ms a real-time operating system is not necessarily required.

7.6 The Level-1 Decision Unit Interface

As described in the previous section a message is sent to the Level-1 Decision Unit Interface after the event buffer contents have been analyzed. Every event buffer in the system writes its message into a shared memory region located on the Level-1 Decision Unit Interface. The message sent by the CNs can be seen in table 7.3.

Table 7.3: 128-bit message sent to the Level-1 Decision Unit Interface .

Bit field	Name	Description
127:96	Nallok	Number of transfers
95:64	Ntries	Number of Algorithm loops
63:48	NodeID	Node ID
47:32	Status	Error Status
31:16	TagID	TagID
15:0	Rsv	Reserved

The TagID field holds the tag ID of the analyzed data, Status holds the error status of the event data currently checked, Ntries the total number of loops the process has executed

so far, and Nallok the number of loops with the analyzed event being the next expected one. The node ID of the node transmitting the message is also transferred. Currently three different status messages are distinguished. Under normal condition the status field signals that the data received has been correct. A different status message is sent if the three tag IDs compared are different but the situation is understood. The third status message produced labels a non understood condition. If the latter status message occurs the situation can be analyzed with a PCI tracer located in the node that detected the condition.

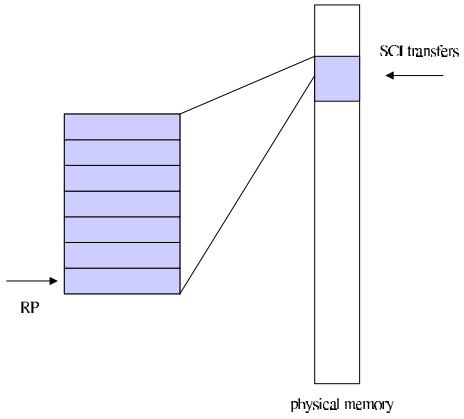

Figure 7.17: Level-1 Decision Unit Interface as implemented in software.

The Level-1 Decision Unit Interface has been implemented as a cyclic buffer. Every event buffer in the system gets assigned one buffer slot. Figure 7.17 shows a sketch of the buffer allowing memory mapped access from every CN. A read pointer (RP) is moved along the buffer entries. On reception of the next message an internal statistic is updated. If a message does not occur within a certain time a timeout watchdog increments an internal timeout counter and moves the read pointer to the next entry.

The Level-1 Decision Unit Interface software reports the system status. A summary can be seen in figure 7.18.

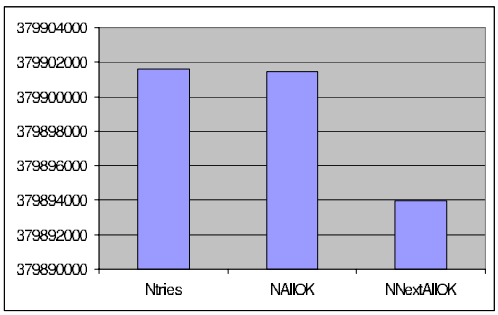

Figure 7.18: The Level-1 Decision Unit Interface collects data coming from 26 CNs and calculates some statistics. All nodes executed a total of 379901605 loops, verified almost 100% of all data as valid and the next in line.

The system has been running for 5 minutes. This can roughly be calculated by taking the number of tries and the system frequency of 1.24 MHz into account. 20 situations have occurred where the results reported by the processes running on the CNs do not match the buffer mismatches that have already been investigated closely. However, these mismatches reported do not show any wrong event data. The mismatches might be due to additional interrupts that have to be handled. An interrupt source is the Ethernet card

being present in every CN. However, the setup has focused to identify packet loss. The interrupt handling of the Linux OS is of minor interest at this point. Within the time period checked close to 100% of the events have equivalent buffers. Almost 100% of all events that have been checked successfully also had had the next expected tag ID.

Compared to the number of interrupts that is expected it can be stated that almost none of the interrupts occurring do influence the system. This measurement has been made on a compute cluster running standard Linux. Therefore, also a system wide analysis shows that a real-time operating system is not required necessarily for the LHCb Level-1 trigger.

7.7 System Performance

Since a global shared memory is used overhead is minimized. Both the sending and receiving nodes transmit the data in PCI bursts, which are characterized by the FRAME# signal asserted longer than one PCI clock cycle. A 128 Byte PCI data burst sequence on the sending node can be seen in figure 7.19. The burst frequency is measured to be 1.24 MHz, which corresponds to a period of $T_{burst} = 810ns$.

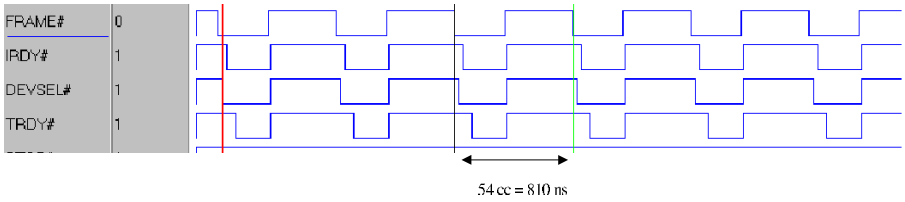

Figure 7.19: The figure shows a PCI trace of 128 Byte bursts in one of the sending nodes. The send frequency in the trace is 1.24 MHz.

A PCI burst, as depicted by the frame signal in figure 7.19, is characterized by one address cycle, seven wait cycles due to the target setup time of the Dolphin PCI card, and 16 data cycles to transfer 128 Byte. 30 idle cycles are in between bursts if the system is running with 1.24 MHz as expected according to section 5.2.2.

Figure 7.20 shows PCI transactions in one of the compute nodes. After reception of three data packets, each sent from a different RU the PCI bus is idle for a certain period of time. The idle time depends on the number of compute nodes N. Since 26 nodes export memory regions the idle time can be calculated to be $T_{idle} = N \times T_{burst}$. The zoomed trace shows the reception of three data packets coming from the RUs and the 128-bit result message. Differences in time in between packet reception is due to different network paths.

As explained earlier a 128-bit result message is sent to the Level-1 Decision Unit Interface. These messages are not synchronous compared to the data packets presented in figures 7.19 and 7.20. The messages received are not necessarily in the same order as an analysis shows. Reasons are differences in latency due to different network paths or simply delayed response of the analyzing process due to process scheduling. A PCI trace taken on the result node can be seen in figure 7.21.

The messages, which can be seen in the figure correspond to the message format shown in table 7.3. The system has always been monitored with PCI tracers. It has been running stable for several days and could be restarted without any problems. The traffic patterns measured and presented do agree with the patterns expected.

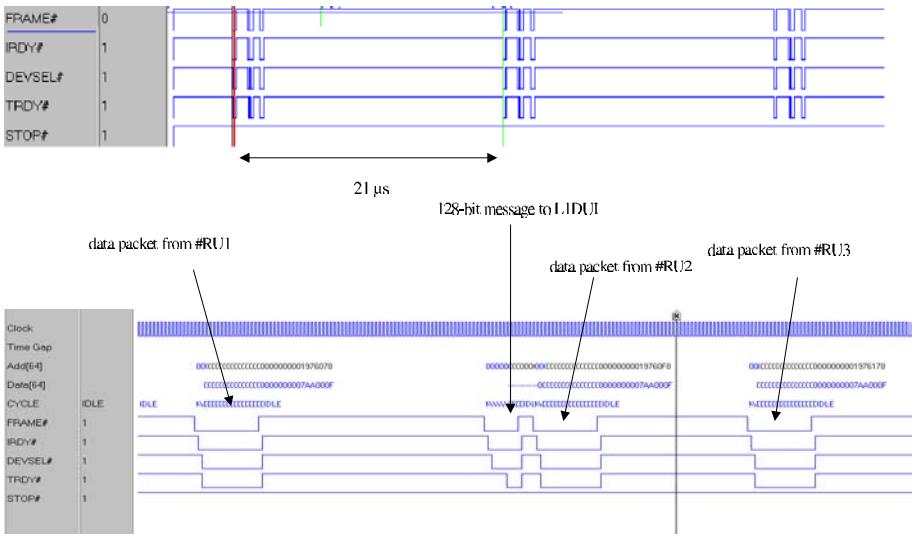

Figure 7.20: The figure shows a PCI trace in one of the receiving nodes. The trace shows a setup for N = 26 resulting in $T_{idle} = 21\mu s$ on the top PCI trace. The bottom measurement shows a zoomed PCI trace, which shows the data packets coming from the different RUs and the 128-bit result message sent to the Level-1 Decision Unit Interface .

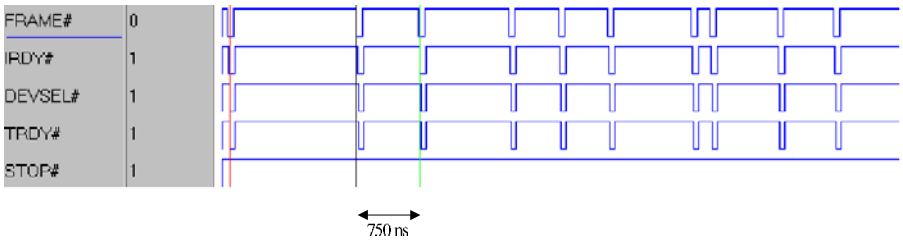

Figure 7.21: The figure shows a PCI trace in the Level-1 Decision Unit Interface. The PCI traffic shows asynchronous behavior.

7.8 System Frequency

The maximum transfer frequency of the system as implemented at the moment has been measured. The maximum frequency is reduced by the busy logic of the TagNet implementation. A timing analysis of the TagNet implementation has been made in 7.2.1. It has been measured that the TagNet accounts to an additional overhead of 14 clock cycles per transfer. By looking at the dependency $N_{cc} = 14 + N_{128} \times 34$, with N_{cc} being the number of clock cycles per transfer and N_{128} being the number of 128 Byte bursts, it becomes clear that the system is slowed down especially for small block sizes. Taking 128 Byte transfers as an example, the difference between the data presented in figure 5.11 and the result obtained with TagNet connections is 580 kHz. Therefore, an approach with an input queue and pipelined transfers are significant to the final system. However, the results achieved are within the requirements of the Level-1 trigger and can be considered sufficient for basic analysis of the system. Figure 7.22 shows the maximum frequency that the system can achieve in the current implementation. The measured results are less compared to figure 5.11 because of the overhead of the TagNet implementation.

As shown in the figure 7.22 the maximum frequency supported by the current implemen-

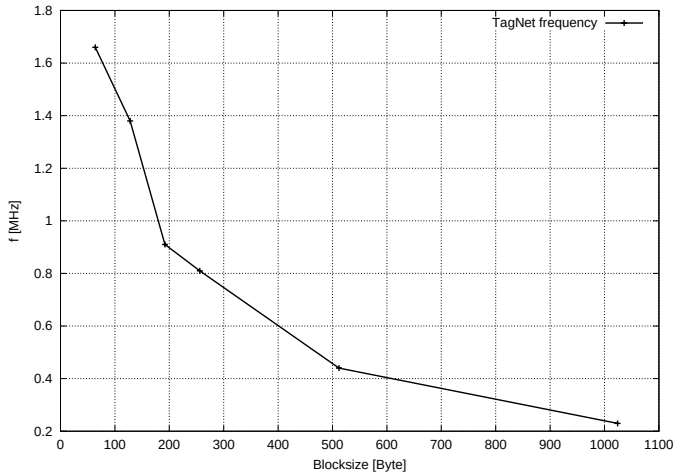

Figure 7.22: Maximum TagNet frequency in MHz as measured for block sizes up to 1024 Byte. The time periods measured differ by 14 clock cycles compared to the data presented in figure 5.11.

tation is 1.38 MHz for 128 Byte packets. This corresponds to an input bandwidth of 168 MByte/s. For our current tests usually the slightly smaller frequency of 1.24 MHz is chosen. This corresponds to 810 ns, which is slightly less than the readout time of the Level-0 derandomizer.

7.9 Scalability

A 2D-torus has been chosen as topology. Compute power can be added by adding a column, input bandwidth can be added by adding a torus row. The latter will also add additional compute power to the system. However, one limit could be set by the maximum number of memory regions that can be imported by one RU. Currently this number is set to 128. A RU in the final LHCb Level-1 trigger will import at least 1200 memory regions. However, the number has been fixed because of software implementation issues and is expected to be increased by the developers at Dolphin soon.

The system has been fed by maximum frequencies as shown in figure 7.22. No retry traffic caused by SCI network congestion has been observed. This statement holds true for setups utilizing different numbers of columns. However, since it is planned to use two RUs in the horizontal ringlets such scenarios have been investigated. Systems that do not scale can be identified by congestion. Congestion is commonly caused by independent devices, which substantially use the same part of the network. However, this can be identified by analyzing the RUs. If there is congestion in the network retry traffic on the RUs PCI bus can be measured since the buffer of the SCI card has not been cleared meaning that the node has not delivered its data yet. This is commonly caused by the following reasons:

- The receiving node can not accept all request send packets.
- An intermediate node can not cope with the bypassing traffic. This is caused by sharing the same link segments.

The first argument does not hold true for the system presented since traffic directed to a CN is orchestrated by the TagNet logic. However, if more than one RU is planned to be used on one horizontal ringlet both input feeds use parts of the horizontal ring concurrently. The bandwidth available through the SCI network should be sufficient for 2 RUs. However, the results presented in chapter 5 suggest that bypassing traffic can have a serious impact on a sending RU.

7.9.1 Non-Scheduled Transfer

Three RUs have been placed next to each other in a six node ringlet. Each RU has been configured to send 128 Byte packets to one specific receiver with maximum possible frequency. Figures 7.23, 7.24, and 7.25 show the result of one of the test runs.

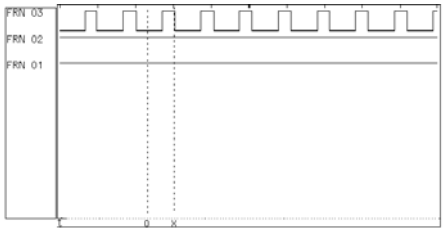

Figure 7.23: One RU sends 128 Byte of data with the maximum frequency of 1.96 MHz. The logic analyzer shows the FRAME# signal on the PCI bus of the sending node.

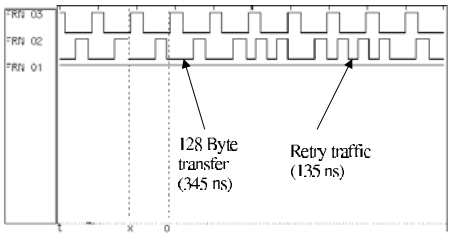

Figure 7.24: Two RU try to send 128 Byte of data with the maximum frequency of 1.96 Mhz each. However, retry traffic can be observed on one of the nodes.

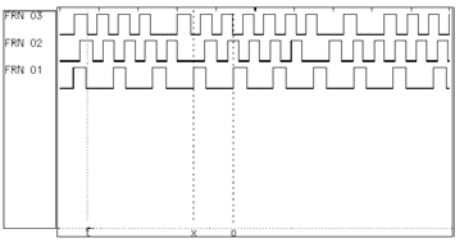

Figure 7.25: After the third RU starts sending the data rate on two RUs drops significantly.

The RUs have been initialized to send with the maximum frequency f_{max} of 1.96 MHz as shown in figure 5.14. As the traces show, one RU can send with maximum frequency. However, if a second and third RU starts sending data only one RU can keep the maximum data rate. Only the RU, which is located upstream at the most such that its bypass FIFO is never filled by the request send packets of the other RUs achieves full bandwidth. This test has been made with RUs, which have not been synchronized and an intended data rate of 239 MByte/s per RU. Figure 7.26 shows a similar setup in a 10 node ringlet. However, every RU has been sending data to four different receiving nodes each.

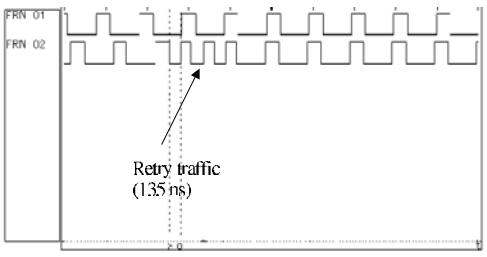

Figure 7.26: Two RUs share a horizontal ringlet and try to send with 128 Byte of data with a rate close to 2 MHz. The PCI FRAME# signal is analyzed and shows retry traffic for the RU presented by FRN 02. However, aggregate maximum transfer rates of up to 478 MByte/s payload can be observed.

Aggregate peak bandwidths of 478 MByte/s have been measured corresponding to 83% of the net SCI bandwidth. However, retry traffic has been observed on the node being located downstream at the most. In case of the LHCb Level-1 trigger this effect would lead to a non-predictable latency time. A node that throttles might influence all RUs since the tag can not be forwarded on time. However, the PCI interface of the RU could detect such a situation and forward the tag in advance.

7.9.2 Scheduled Transfer

The same measurements have been done in the LHCb prototype environment using the TagNet implementation. Figure 7.27 shows a sketch of a setup with two RUs being located next to each other.

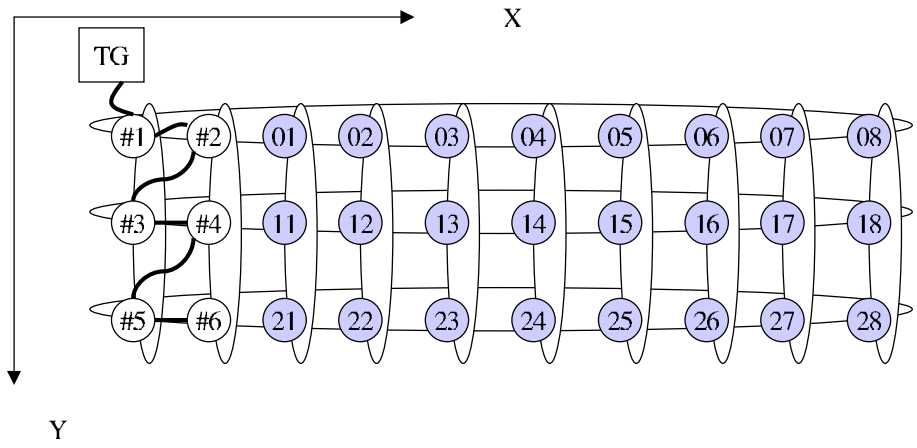

Figure 7.27: Two RUs share a horizontal ringlet and are connected by the TagNet. The SCI cabling is chosen such that packets sent by RU#2 have to traverse the bypass FIFO of RU#1.

Different block sizes have been used with the maximum frequency allowed by the TagNet implementation (see figure 7.22). One or more FRAME# signals asserted for 9 clock cycles in a row followed by a burst of data identifies retry traffic on the PCI bus. However, no impact on the sending frequency could be found for block sizes up to 256 Byte in size. However, when using 512 Byte blocks retry traffic can be observed. Figure 7.28 shows that the FRAME# signal of RU#1 (FRN01) signals retry traffic on the PCI bus.

By choosing 512 Byte block sizes the bypass FIFO of RU#1 always experiences the same traffic pattern; four 128 Byte packets separated by 150 ns and a larger gap caused by the TagNet overhead. However, this traffic scheme is sufficient to see the retry effect. The

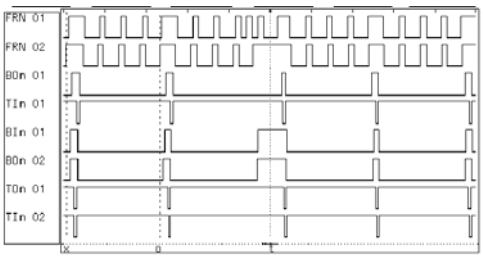

Figure 7.28: Retry traffic caused by bypass traffic. RU#1 has to issue retries which is due to bypass traffic originating from RU#2.

TagNet frequency has been adjusted such that no retry traffic can be observed anymore. The frequency can be adjusted in single clock cycles on the PCI bus. Figure 7.29 shows the traffic pattern when no retry traffic occurs anymore.

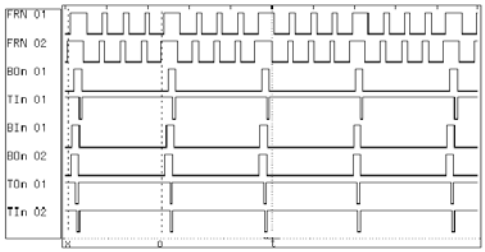

Figure 7.29: No retry traffic can be observed by adjusting the bypass traffic frequency. The sustained bandwidth accounts to 432 MByte/s.

The bandwidth measured is 432 MByte/s. This bandwidth is the maximum bandwidth feasible in a system of 2 RUs being located next to each other. This bandwidth, B_{Link}, has been introduced in 6.2. If two RUs want to utilize B_{Link}, they are allowed to send with a frequency of 1.79 MHz each, 128 Byte packets assumed.

However, RUs do not have to be located next to each other as depicted in chapter 6 figure 6.1. Figure 7.30 shows an architecture with displaced RUs.

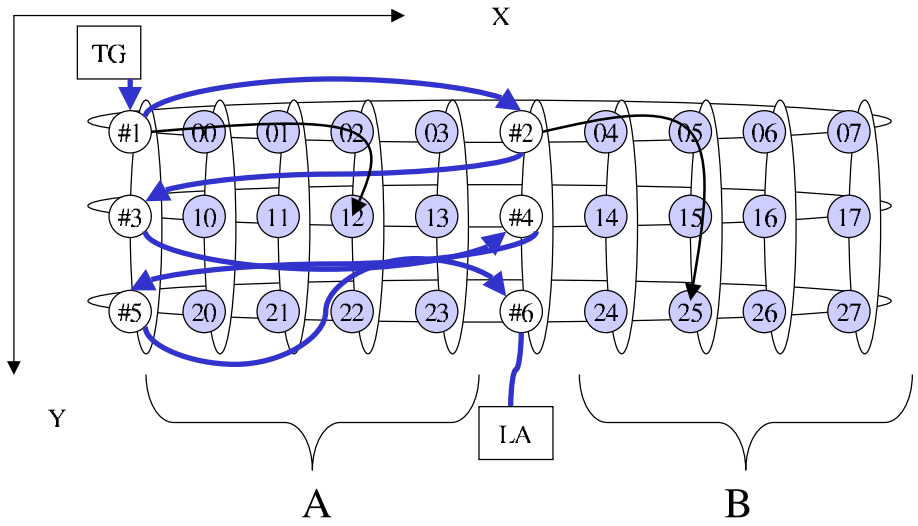

Figure 7.30: Displacing the RUs minimizes SCI bypass buffer occupancy on the node further downstream. The direction of request-send packets is depicted by two example transfers.

The figure shows six RUs connected by the TagNet and 24 CNs. As an example concurrent

data transfer of RU#1 and RU#2 is shown. If the RUs on the far left send data to partition A, no request-send packet has to traverse the bypass FIFOs of the RUs located in the center of the torus. Only targets which are located in partition B require bypassing the second RU column. The same situation is true for RUs being located in the center of the torus. Only when traffic is directed to a destination in A data packets will bypass a RU column. Assuming scheduling such that the load in the system is balanced every RU has to bypass 50% of the rows traffic. In the original setting the RU being located further downstream has to cope with 100% of the traffic originating from the RU upstream.

Measurements have been made with balanced traffic to investigate if B_{Link} can be increased by a setup like this. 512 Byte transfers have shown no impact on neither RU. The block size has been increased up to 4096 Byte with no retry traffic observed. Figure 7.31 demonstrates that no retry traffic can be observed anymore on neither RU.

Therefore, it can be stated that displacing the RUs increases B_{Link}. B_{Link} peak bandwidths of 478 MByte/s have been observed. This can be calculated to be 83% of the net SCI bandwidth. It corresponds to one RU sending 128 Byte packets with a frequency of 1.96 MHz as depicted in figure 5.11. However, the average bandwidth observed over a long time period has been 472 MByte/s.

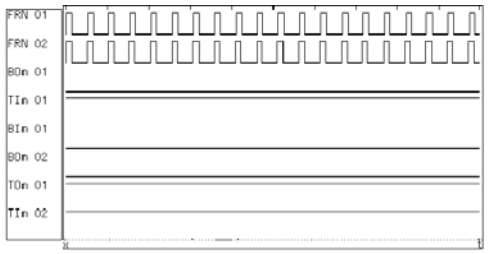

Figure 7.31: Two RUs being located as proposed in figure 7.30 increase B_{Link}. The peak bandwidth measured is 478 MByte/s whereas the average bandwidth over a long time interval is 472 MByte/s which is due to the overhead implied by the TagNet implementation. The block size has been set to 4096 Byte with the network load being balanced.

Unbalanced tests have been conducted as well. Figure 7.32 shows the result of a setup where both RUs send in partition B all the time. Since every request-send packet of RU#2 must traverse RU#1 the traffic pattern is similar to a setup with non-displaced RUs.

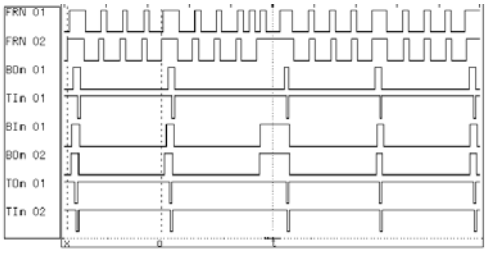

Figure 7.32: Two RUs being displaced send into the same partition. The trace shows retry traffic at a total bandwidth of 453 MByte/s and a block size of 512 Byte.

However, a different setup has been chosen to investigate the traffic pattern when RUs being located in the same row send constantly into the same column. Figure 7.33 reveals that retry traffic can be observed on both sending nodes. The aggregate bandwidth for this setup has been set to 453 MByte/s. Therefore, a filled bypass FIFO can not be the reason for performance loss.

The B-Link is the reason for decreased system performance since the aggregate bandwidth

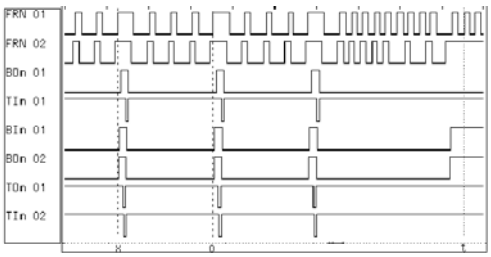

Figure 7.33: Two RUs being displaced send into the same column. Retry traffic on both RUs can be observed. The aggregate bandwidth is 453 MByte/s with the block size being 512 Byte.

in this setup accounts to 89% of the peak net bandwidth of the B-Link. This calculation assumes one bus turn-around cycle in between packets. However, this will be investigated in 7.10.

The SCI driver running on the RUs implements some heartbeat functionality which checks the local SCI card every 100 ms. Additionally remote nodes are probed every second. This causes additional traffic on the RUs PCI bus which is not due to network congestion. However, in the final system those features will be turned off.

It can be stated that a system with two RUs in a row is feasible. However, the results presented have impact on the way the TagNet scheduler must assign CNs. It must be avoided that RUs being located in the same row send to the same partition at any given time.

7.10 B-Link Performance

The RUs have to send the data to a specified CN. Therefore, the data have to be routed. B_{B-Link} is defined as the maximum bandwidth that can be handled by the B-Link. The aggregate peak bandwidth has been measured in a ringlet and in a torus. The sending nodes have been chosen to be CN 01 and CN 02 whereas the receiving nodes have been chosen to be CN 04 and CN 05. The nodes are depicted in figure 7.27. In case of the torus the data path has been chosen such that a route node is in between sending and receiving nodes. Thus, receiving nodes have been moved to be CN 13 and CN 23 with CN 03 being the route node.

Figure 7.34 shows the result of the measurement taken in the ringlet. The aggregate peak bandwidth observed at the receivers is 478 MByte/s. The aggregate data packet frequency is 3.92 MHz with the data size being 128 Byte.

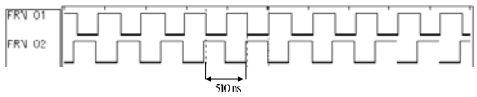

Figure 7.34: Aggregate bandwidth for two CNs in a ringlet. The maximum bandwidth achieved is 478 MByte/s. The measurement has been taking on the receiving sides. The aggregate data packet frequency is 3.92 MHz with the data size being 128 Byte. Sending nodes are CN 01 and CN 02 whereas CN 04 and CN 05 are receivers (see figure 7.27).

The very same setup has been transferred to a torus. Figure 7.35 shows the result of the measurement. The top screen shot shows the situation when only one CN is receiving data. However, when both CNs receive data the peak aggregate bandwidth drops to 450 MByte/s which is B_{B-Link}. The maximum B-Link net bandwidth can be calculated to be 512 MByte/s. Therefore, the measured value for B_{B-Link} is 88% of the theoretical one.

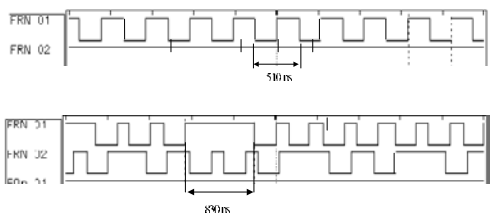

Figure 7.35: Aggregate bandwidth for two CNs in a torus. All traffic goes through the same route node. The maximum bandwidth observed has been 450 MByte/s. Sending nodes are CN 01 and CN 02 whereas CN 13 and CN 23 are receivers (see figure 7.27).

The only difference in the data path between ringlet and torus is the passage through the B-Link in the route node. However, in the final system data will not be routed by one single node at a rate coming close to this performance test. A scheduled system will pick the targets such that traffic originating from one torus row gets distributed into different columns. Assuming a system with two RUs in one row sending 128 Byte with 1.11 MHz each, the aggregate bandwidth on the horizontal accounts to 271 MByte/s. Assuming a system with 17 CN columns, every route node has to provide the fraction of 16 MByte/s on its B-Link to satisfy routing requirements.

7.11 System Latency

The time an event stays in the Level-1 prototype has been measured. Figure 7.36 shows the setup and the packet path, which has been used for measurements.

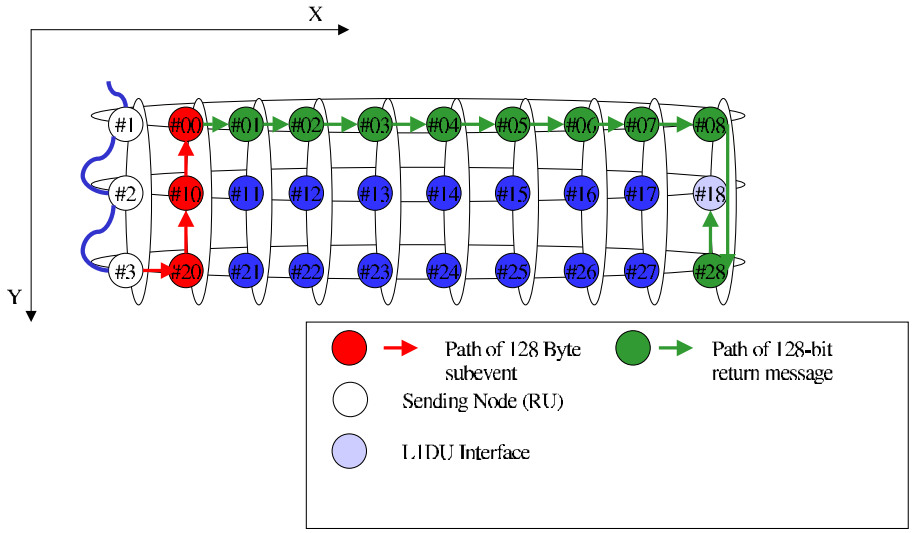

Figure 7.36: Packet path for system latency measurements.

Two PCI tracers have been used to measure the time an event stays in the system. One

PCI tracer has been located in RU#3, which is the last RU in line that sends a packet to a specific CN. The PCI tracer triggers on a specific data set, which is sent to CN#00. The path of the 128 Byte SCI send packet is shown in red. On CN#00 a modified packet integrity algorithm is running. It polls on the event buffer where data is expected next and checks it immediately. After reception of all three data packets the 128-bit result message is sent to the Level-1 Decision Unit Interface. The path for this message is shown in green color. The Level-1 Decision Unit Interface is also equipped with a PCI tracer, which triggers on the message coming from CN#00. Since both tracers are capable of producing an external trigger signal both signals can be fed into an oscilloscope and the latency can be measured. Figure 7.37 shows the result of one measurement. Several measurements have been made. The result is shown in table 7.4.

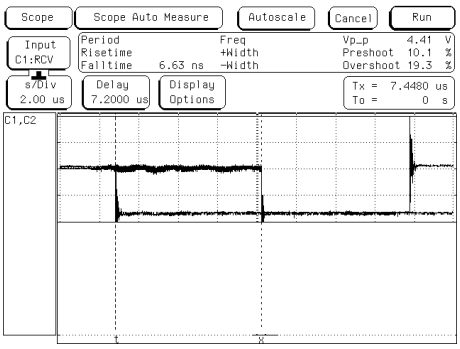

Figure 7.37: Latency as measured with two PCI tracers.

Table 7.4: System Latency for the setup shown in figure 7.36.

T_{SYSLAT}	ΔT_{SYSLAT}
7.38 μs	0.27 μs

The time measured can be split as follows:

- $T_{networkIN}$: Data is moved to the specified CN. The path is shown in red color.
- T_{CN}: The process compares the three received SCI packets and sends a message to the Level-1 Decision Unit Interface .
- $T_{networkOUT}$: Data is moved to the Level-1 Decision Unit Interface. The path is shown in green color.

If the results presented in section 5.1.4 are taken into account T_{SYSLAT} for the setup presented in figure 7.36 can be split up in the following way:

$$\begin{aligned} T_{SYSLAT} &= T_{networkIN} + T_{CN} + T_{networkOUT} \\ &= 2 \times T_{Lat\_PCI} + 2 \times T_{Lat\_Rt} + 8 \times T_{Lat\_By} + T_{CN} \\ &= 4.42\mu s + T_{CN} \end{aligned}$$

With the value measured for T_{SYSLAT}, T_{CN} roughly accounts to 3 μs. The measurements have been verified by choosing a CN which does not require routing of data packets. The measured value has been less according to the results presented in section 5.1.4. The latency does not depend on the system traffic. Setups have been chosen where only part of the network are utilized. No influence on the system latencies has been observed.

7.12 Fault Tolerance

Some simple tests with respect to fault tolerance have been made. The full system has been started and one CN, which has been randomly chosen has been stopped but not powered off. All measuring devices attached to the system have shown normal output. The same result has been achieved by stopping the Level-1 Decision Unit Interface. A CN node has been stopped and its PCI bus monitored. All messages are still written to the specified location in host memory. However, since the data integrity check process has been killed no result message is returned. These observations show that the system is not influenced by a node that crashed. However, in case of a power failure one node will affect two ringlets. To prevent that case, it has to be considered to implement a fault tolerance system in hardware, which can cope with power failures in a CN.

Chapter 8

Summary and Outlook

This thesis has been the first step towards the design and implementation of the Level-1 trigger processor of the LHCb experiment. Boundary conditions have been set upon the trigger which have been studied both theoretically and in experimental work. The boundary conditions which had to be met are as follows:

- A large scale COTS compute farm should be connected by high-speed interconnect. The number of network nodes is not final and depends heavily on simulations done by physicists at CERN. Therefore, a scalable solution had to be found which satisfies network latency and bandwidth needs.
- The Level-1 trigger receives data every microsecond with data being split into tiny fragments of 128 Byte on average. The interface to the NIC is PCI to minimize cost especially on the receiving side. Therefore, data has to be moved across the PCI bus every microsecond, avoiding additional overhead and making best use of the bus resource. Additionally, the 128 Byte data fragment size is an assumption of the author. Average data sizes change frequently depending on the latest simulation.

A shared memory based NIC has been investigated — the Scalable Coherent Interface. Analyzing the PCI bus with respect to additional overhead, revealed that moving data by SCI contains no overhead on the PCI bus which has seemed to be a possible solution for the LHCb project. However, using conventional methods like PIO and DMA make the system depend on the CPU load, the chipset, and on the response time of the SCI hardware in case of DMA. The latter is especially noteworthy when discussing backup solutions without system changes. PIO which is especially appropriate for small block sizes could not give the necessary yield on the sending side. However, data is not located in the host memory of a PC in case of LHCb and the data does not have to traverse the hostbridge of the sending node.

Therefore, a solution named Hardware Initiated DMA (HDMA) has been introduced. HDMA relies on a device-to-device copy with an additional logic device being located on the same PCI bus as the SCI NIC. Once an initialization phase has been finished, data can be moved directly to a remote node without intervention of the CPU or the chipset. The latter is only used for bus arbitration which happens reasonably fast in nowadays I/O solutions. The logic has been implemented in an FPGA with PCI interface. Initial studies

on a 32-bit/33 MHz architecture have been done with interleaved and non-interleaved solutions. Especially the interleaved solutions has made best possible use of the PCI resource. Using data bursts and only one idle cycle between different mockup sub-events, data rates of up to 1.51 MHz for 64 Byte data payloads could be achieved.

With 64-bit/66 MHz bus solutions becoming available the interleaved transfer scenario had to be dropped because of lacking slot resources on the motherboards. However, moving to faster FPGAs making use of the fast PCI bus has shown a promising result. Measurements have been presented in chapter 5.2, showing results of HDMA, packet latency, and behavior of large ringlets.

Packet latency is a crucial issue when designing a large scale system. The LHCb Level-1 trigger can be calculated to have roughly 300 farm nodes. The SCI cards have sufficient buffer space each as mentioned in 4.5. However, scenarios with multiple senders sending to one receiver and a data rate being as high as 4.5 GByte/s on the receiving side will cause congestion and retry traffic. A hardware based scheduling network has been designed which is only touched in this thesis. The basics of the TagNet are discussed and data traffic as foreseen has been sketched and measured. The network latency can also be predicted for large systems by extrapolating hands-on results presented in this thesis.

Taking the buffer space and the SCI protocol into account the maximum ringlet size which does not show retry traffic has been calculated. The calculation is based on the network path combined with the fact that SCI output buffers are not end-to-end based but rather ringlet oriented. The maximum ringlet size which can be run congestion free has been roughly determined to be around 100 which is far beyond the size the LHCb experiment will implement. The expected maximum ringlet size will be around 18 for the final system.

Taking the consideration above into account a 30 node compute farm has been set up. It comprises three sending nodes (RUs), 26 receiving nodes (CNs), and one dedicated node which receives the result messages of the CNs. A TagNet basic version has been implemented using the internal LVDS drivers of the FPGAs and flat ribbon cables. However, this implementation has been successful in many overnight runs guaranteeing data orchestration and congestion control. A software suite has been developed which initialized the compute farm in the beginning, starts processes on every CN, and a software based Level-1 Decision Unit Interface process on one node. Stimulated by an external tag generator the system has been moving data for many days in row, keeping a log file when a situation occurred which could mean an error condition. A data integrity process has been running on every CN. No packet loss has been detected. However, mainly caused by interrupts the process is sometimes suspended and can not check its event buffers.

Many obstacles had to be taken until the system had been up and running. Trouble shooting revealed that part of the cards had a design flaw. After their replacement the cluster ran out of the box. Two FPGA families have been used since 1999. However, since logic has been described in VHDL porting of functionality to the Altera solution has not been that time consuming. However, dealing with the PCI core of the Altera does require more logic compared to Lucent.

Two bottlenecks have been investigated which have not been obvious in the very beginning of the project. Bypassing traffic is influencing the RU being further downstream and thus limiting B_{Link} to 432 MByte/s. However, by displacing the RU first measurements have shown that B_{Link} can be increased up to 472 MByte/s of sustained bandwidth.

The maximum bandwidth that the B-Link can handle has been measured as well. The result has no significance for LHCb since the TagNet scheduler will choose the CNs such that B_{B-Link} will never reach the crucial bandwidth of 450 MByte/s. However, the aggregate bandwidth in a vertical ringlet must never exceed B_{Link}, therefore requiring that $n_{rows} \leq n_{col}$.

SCI has been shown to be a reasonable solution for the LHCb Level-1 trigger. However, a backup solution can be investigated. The concept of HDMA could also be applied to message passing solutions which support PIO. This has never been prototyped but could work with solutions like Atoll [49], QNIX [50], and the Infinibridge[51] based on Infiniband [52] just emerging.

The PCI bus utilization on the CNs is small due to the TagNet. Assuming an event size of 4.5 kByte and an event being sent to a specific CN every $250\mu s$ the overall bandwidth roughly accounts to 18 MByte/s. Even for larger events the PCI bus on the receiving nodes is far from being utilized significantly. Future investigations might include the combination of one SCI receiver and a multiprocessor board having more than two CPUs.

The idea mentioned above are not of uttermost importance. Important steps that have to be done in the near future include:

- Take the results of this thesis into account and start a simulation of the system with a size equal to the final trigger.
- Implementation of the TagNet in a more sophisticated way which includes the scheduler and makes data distribution dynamic.
- Implementation of the Level-1 Decision Unit Interface in hardware.
- Reduction of the 10 idle cycles in between PCI burst. This might require a pipelined PCI core.

Especially the last item mentioned could push the performance of the system. However, a faster input might also require to increase the SCI link frequency. Future investigations might also concentrate on a PCI-X based SCI solution. However, at the time of writing no implementation has been released.

Appendix A

Implementation

A.1 Design Flow

Both TagNet and the DMA logic have been implemented using VHDL descriptions. The source files have been synthesized using Synopsys FPGA Express [53], which creates an EDIF netlist [54] as output. The netlist serves as input for the Altera Quartus II software [47], which creates a configuration file. The device has been configured using JTAG configuration. For more details on the PCI board that has been used and its configuration refer to [55].

A.2 TagNet Implementation

The tag forwarding mechanism is pipelined and allows to store one tag whilst a new tag is already processed. Since the current implementation does not foresee any pipelining in the input stage the store and forward state machine in tagnetFSM waits for three clock cycles after the tag has left the FPGA. This has been implemented to account for the latency of the BIn signal, which has to be set after the adjacent board receives the tag. If no wait cycles had been implemented two tags could be sent back-to-back, which results in loss of the second tag.

The physical layer of the TagNet links has been designed using the internal LVDS drivers of the Altera Apex20K400E. This simple implementation does not synchronize the signals on reception. The signal TIn is asserted for four clock cycles, which makes sure that the receiving node can receive the signal. The actual cabling, which has been used for all the results presented in 7 can be seen in figure A.1.

Despite the crude setup signalling has always been reliable. However, the total cabel length is less than 50 cm since problems with longer cables have been observed. Table A.1 shows the TagNet signals and their assignments to the FPGA's TX and RX lines.

Aux0 is used to transmit the output tag of the tag generator.

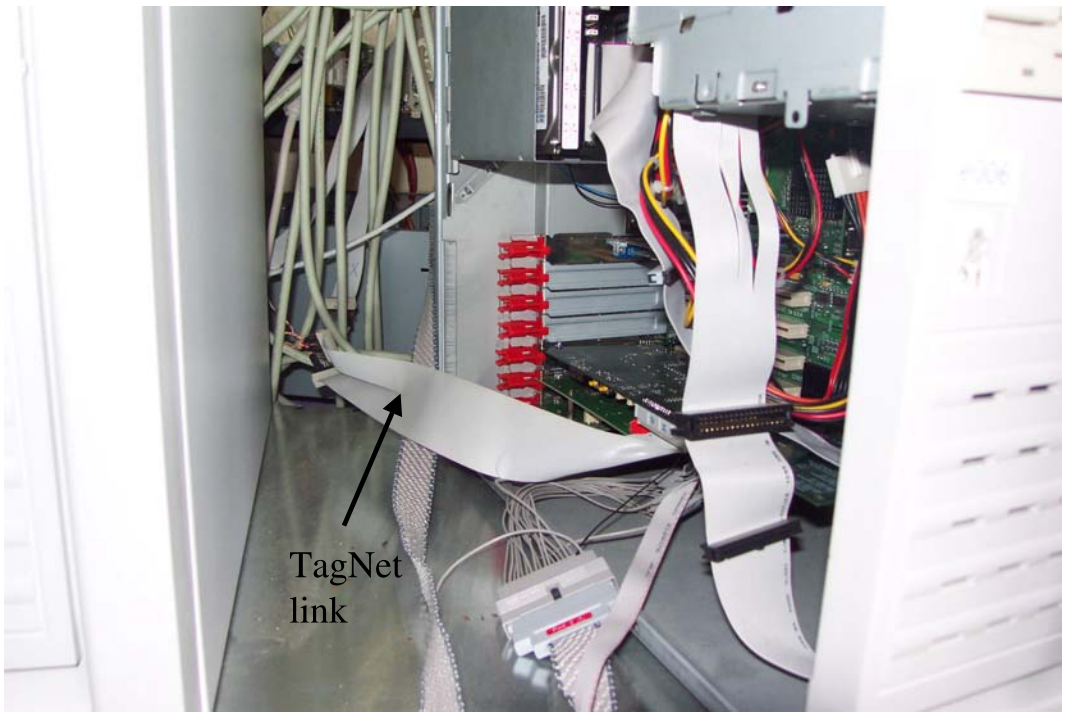

Figure A.1: Flat ribbon cables have been used to connect the boards.

Table A.1: Assignment of the RX and TX channels.

TagNet line	top unit ouput (VHDL)	channel
TIn	tagIn	RX0
TOn	tagOut	TX3
BIn	bsyin	RX1
BOn	bsyout	TX0
Aux0	LVDSexponeout	TX1
Aux1	LVDSexptwoout	TX2

A.3 Virtual to Physical Address Translation

When a program looks up a virtual address, the address is converted to a physical address in order to access physical memory. The step is usually performed by splitting the address into bitfields. In the Linux operating system each bitfield is used as an index into an array, called a page table , to retrieve either the address of the next table or the address of the physical page that holds the virtual address.

The function

```
PSI_getPhysAddress( const void* virtAddr, void** physAddr, void**busAddr );
```

which is part of the psi device driver developed in Heidelberg [48], walks along those page tables and returns the physical and the bus address. On Intel x386 bus addresses and

physical addresses are identical. For more information on these topics refer to [56].

Appendix B

Contents of the CD

- The complete source of this thesis including all figures.
- All Hardware descriptions and software that has been used for the results presented. The CD contains a README file which should be consulted first.
- All documentation mentioned in the bibliography.

Appendix C

Glossary

Compute Node (CN) — A node of the Level-1 farm that processes event data. In the prototype presented, a CN runs an algorithm which checks for packet loss.

Direct Memory Access (DMA) — A facility of some architectures which allows a peripheral to read and write memory without intervention by the CPU.

ECS — The Experimental Control System handles configuration, monitoring, and operation of all experimental equipment.

Event data — VELO data originating from the same interaction.

Field Programmable Gate Array (FPGA) — A gate array where the logic network can be programmed into the device after its manufacture. An FPGA consists of an array of logic elements, either gates or lookup table RAMs, flip-flops and programmable interconnect wiring.

LHC — LHC stands for Large Hadron Collider.

LHCb — One of the four LHC experiments.

NIC — Network Interface Card.

Off Detector Electronic (ODE) — The ODE digitizes the data coming from the VELO front-end chips, buffers the data during Level-1 latency, and preprocesses the data for the Level-1 algorithm.

Peripheral Component Interconnect (PCI) — A standard for connecting peripherals to a personal computer, designed by Intel.

Programmed I/O (PIO) — Read and write operations are initiated by the CPU.

Readout Unit (RU) — Feeds the Level-1 network.

Scalable Coherent Interface (SCI) — IEEE Std 1596-1992 is a high-performance, low latency interconnect that supports distributed shared memory.

Sub-event data — Part of the event data. A sub-event is assembled by the RUs and has to be sent to a CN within a microsecond.

Trigger — Part of a readout chain which looks for valuable data in real-time. If data is assumed to be worth keeping a trigger signal is issued. Otherwise the data are discarded and therefore reduced.

Acknowledgments

Many people contributed and helped on the way to complete this thesis. My adviser Prof. Dr. Volker Lindenstruth contributed with many ideas and discussions. Without his help and the working environment he provided this thesis would not have been possible.

I want to thank both present and former members of the Chair for the various contributions and discussions. Especially I want to mention Konstantinos Giapoutzis, Dr. Ivan Kisel, Falk Lesser, Rolf Schneider, Dr. Markus W. Schulz, and Timm M. Steinbeck. I want to thank Béatrice Bähr for her kind support especially when I prepared my trips to meetings and conferences.

It was the first time that I worked in a large collaboration like LHCb with people coming from all around the world. There is one person I want to mention especially because of the discussions we had on the Readout Unit, the meaning of life, and because working with you has always been a lot of fun — thank you Hans.

Bibliography

[1] *LHCb Technical Proposal.*
CERN LHCC 98-4 .

[2] T. Berners-Lee. WWW proposal.
http://www.w3.org/History/1989/proposal .

[3] *CERN document server.*
http://weblib.cern.ch .

[4] *HERA-B Home Page.*
http://www-hera-b.desy.de/ .

[5] *BaBar Home Page.*
http://www.slac.stanford.edu/BFROOT/ .

[6] *Lucent Technologies.*
http://www.lucent.com .

[7] *BELLE Home Page.*
http://bsunsrv1.kek.jp/ .

[8] *LHCb Vertex Locator Technical Design Report.*
CERN LHCC 2001-011 .

[9] *TTC System Information.*
http://ttc.web.cern.ch/TTC/intro.html .

[10] T. Nakada. *Meeting with LHCC referees.*
http://documents.cern.ch/AGE/current/fullAgenda.php?ida=a02917 .

[11] F. Anghinolfi et al. *SCTA - a Rad-Hard BiCMOS Analogue Readout ASIC for the ATLAS Semiconductor Tracker.* IEEE Trans. Nucl. Science Vol. 44, No. 3, June 1997.

[12] *ATLAS Homepage.*
http://atlasinfo.cern.ch/Atlas/Welcome.html .

[13] D. Baumeister et al. *The Beetle Reference Manual.*
CERN LHCb 2001-046 .

[14] *Kirchhoff-Institut für Physik.*
http://www.kip.uni-heidelberg.de .

[15] Aurelio Bay et al. *LHCb VELO Off Detector Electronics Preprocessor and Interface to the Level-1 Trigger.*
CERN LHCb 2001-043 .

[16] *CERN S-Link Homepage.*
http://hsi.web.cern.ch/HSI/s-link/ .

[17] Jose Francisco Toledo. *Study and Design of the Readout Unit Module for the LHCb Experiment.* PhD thesis, University of Valencia, Gandia, 2001.

[18] *Hans Muller's Home Page.*
http://hmuller.home.cern.ch/hmuller .

[19] *Hans Müller private communication.*
Hans.Muller@cern.ch .

[20] H. Dijkstra. *The LHCb Vertex Locator and Level-1 Trigger.*
CERN LHCb 2000-001, TRIG .

[21] *PCI SIG PCI-X Specification.*
http://www.pcisig.com/specifications/pci_x .

[22] *Tyan Computer Corporation.*
http://www.tyan.com .

[23] *SuperMicro Computer, Inc.*
http://www.supermicro.com/ .

[24] V. Lindenstruth. *Informatik II Vorlesung.*
http://www.kip.uni-heidelberg.de/ti/Lehre/index.shtml .

[25] Tom Shanley and Don Anderson. *PCI System Architecture.* Addison-Wesley Longman, Inc., 1995.

[26] *IEEE Standard for Scalable Coherent Interface (SCI) 1596-1992.* The Institute of Electrical and Electronics Engineers, Inc. 1993.

[27] *Myricom Homepage.*
http://www.myri.com .

[28] *IEEE Standard for Low Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI) 1596.3-1996.* The Institute of Electrical and Electronics Engineers, Inc. 1996.

[29] *IEEE Standard Control and Status Register (CSR) Architecture for Microcomputer Buses 1212-1991.* The Institute of Electrical and Electronics Engineers, Inc. 1991.

[30] *Dolphin Interconnect.*
http://www.dolphinics.no .

[31] *LC3 Specification.*
http://www.dolphin.no .

[32] PCI Special Interest Group. *PCI Local Bus Specification, Revision 2.1.* 5200 Elam Young Parkway, Hillsboro, Oregon 97214-6497, USA.

[33] *ServerWorks, Inc.*
`http://www.serverworks.com` .

[34] *IEEE 1076 VHDL Reference Manual.*
`http://www.ieee.org` .

[35] *IEEE 1364-1995 Verilog Language Reference Manual.*
`http://www.ieee.org` .

[36] *Altera Corporation.*
`http://www.altera.com` .

[37] J. Christiansen. *Requirements to the L0 front-end electronics.*
`CERN LHCb 2001-014` .

[38] J. Christiansen. *Requirements to the L1 front-end electronics.*
`CERN LHCb 2001-127` .

[39] *LHCb Electronics Key Parameters.*
`http://lhcb-elec.web.cern.ch/lhcb-elec/html/key_parameters.htm` .

[40] Y. Ermoline. *Vertex Detector Electronics: L1 electronics systems issues.*
`CERN LHCb 2001-124` .

[41] *Hans Dijkstra, private communication.*
`hans.dijkstra@cern.ch` .

[42] *Niels Tuning, private communication.*
`niels.tuning@cern.ch` .

[43] Roland Richter. Implementierung eines Algorithmus für den LHCb Level-1 Vertex Trigger. Master's thesis, Univerität Heidelberg, 2000.

[44] *LHCb Trigger Meetings.*
`http://documents.cern.ch/AGE/current/displayLevel.php?fid=2l63` .

[45] *Ivan Kisel, private communication.*
`ikisel@kip.uni-heidelberg.de` .

[46] *Moore's Law.*
`http://www.intel.com/research/silicon/mooreslaw.htm` .

[47] *Altera Quartus II Software.*
`http://www.altera.com/products/software/quartus2/qts-index.html`.

[48] *Alice High Level Trigger Software and Documentation.*
`http://www.kip.uni-heidelberg.de/ti/L3/`.

[49] L. Rzymianowicz et al. *Atoll: A Network on a Chip.*
PDPTA 99. Las Vegas, June 1999 —.

[50] A. De Vivo et al. *QNIX: A Flexible Solution for a High Performance Network.* SCI 2002. Orlando, July 2002 —.

[51] *Mellanox Technologies.* http://www.mellanox.com —.

[52] *Infiniband Trade Association.* http://www.infinibandta.org/home —.

[53] *Synopsys.* `http://www.synopsys.com` .

[54] *Edif.* `http://www.edif.org` .

[55] *Altera Apex 20KE PCI Development Board.* `http://www.altera.com/literature/ds/dspcibd20ke.pdf`.

[56] Alessandro Rubini and Jonathan Corbet. *Linux Device Drivers.* O'Reilly & Associates, Inc., 1998.

Index