I N A U G U R A L – D I S S E R T A T I O N

zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich-Mathematischen Gesamtfakultät
der
Ruprecht – Karls – Universität
Heidelberg

vorgelegt von

MSc Deyan Atanasov

aus Kubrat, Bulgarien

Tag der mündlichen Prüfung: 21. Dezember 2009

Design and Implementation of a System for Data Traffic Management in a Real-Time Processing Farm Operated at 1 MHz

Gutachter: Prof. Dr. Volker Lindenstruth

Prof. Dr. Ulrich Brüning

Abstract

Viele der heutigen Hochenergiephysik-Experimente erforschen seltene Phänomene und benötigen daher eine Echtzeitdatenverarbeitung mit hohen Durchsatzraten, um das Rohdatenaufkommen der Detektoren von einigen Terabytes pro Sekunde auf eine Rate zu senken, die für die Speicherung und detaillierte Auswertung geeignet ist. Anwendungsspezifische Trigger-Systeme wählen die für das physikalische Experiment relevanten Ereignisse aus. Häufig werden Datenfragmente, die zu einem Ereignis gehören, aber von mehreren verschiedenen Detektoren produziert werden, an einer Stelle gesammelt, bevor sie verarbeitet werden. Das sich ergebende Kommunikationsmodel kann bei ungesteuerter Datenübertragung zu Verzögerungen und ineffizienter Nutzung von Rechenzeiten führen, da eine Vielzahl von Quellen versuchen, um Netzwerkverbindungen und Rechenkapazitäten zu konkurrieren. Diese Arbeit behandelt die notwendigen Maßnahmen, um einen störungsfreien und lastverteilten Betrieb einer Echtzeittrigger-Farm sicherzustellen, die Paketgrößen von einigen Kilobytes bei einer Taktrate im Megahertz Bereich verarbeitet. Die über mehrere Quellen aufgeteilten Daten müssen dabei innerhalb einiger Millisekunden zusammengefasst und verarbeitet werden. Die Rechnerfarm besteht aus gewöhnlichen PCs, die ringförmig durch ein handelsübliches Hochgeschwindigkeitsnetzwerk mit niedriger Latenz verbunden sind. Es wird ein System zum Datenverkehrsmanagement vorgestellt, basierend auf einer globalen Steuerungseinheit und einem dedizierten Steuerungsnetzwerk. Erstere reserviert verteilte Rechenkapazitäten dynamisch, um Netzwerkstaus zu vermeiden und die Belastung der Systeme aufzuteilen. Letztere versorgt die Datenquellen mit den Steuerungsinformationen, um die überwachte Datenübertragung anzustoßen. Anhand eines Prototypen-Rechnerverbundes mit einer hardwarebasierten Steuerung des Netzwerkverkehrs wird der störungsfreie Betrieb gezeigt. Basierend auf den gemessenen Parametern werden Simulationsergebnisse für große Computerfarmen präsentiert. Prototyp und Simulation zeigen die Fähigkeit des Systems, 128 Byte Blöcke, die über mehreren PCI-Eingangsquellen mit einer Rate von mehr als 2 MHz zur Verfügung gestellt werden, auf einen fernzugreifbaren Speicher (Remote Shared Memory) zu transportieren. Sowohl die Mess- als auch die Simulationsergebnisse demonstrieren, dass es möglich ist, ein hochverfügbares Mehrcomputer-Trigger-System mit geringer Latenz auf handelsüblichen Komponenten aufzubauen. Dafür muss nur der tatsächliche Datentransfer sorgfältig gesteuert werden, um, bei sinnvollem Einsatz der Rechenkapazitäten, effizient Daten zusammen zu führen.

Abstract

The majority of contemporary high-energy physics experiments study rare phenomena, which necessitates real-time high-throughput data processing to reduce the raw detector data rate of several Tbyte/s to a rate which is feasible for storage and detailed analysis. Unique trigger systems select the physical events relevant to the experiment. Typically, data fragments corresponding to the same event and originating from multiple detector data sources need to be assembled in a specific location before being processed further. The resulting communication model can lead to congestions and to inefficient system utilization if data are transferred without supervision since numerous sources are attempting to use common interconnect and computing recourses concurrently. This thesis deals with the measures taken to ensure a congestion-free, load-balanced operation of a real-time trigger farm processing data packets as small as several kbytes at a megahertz rate. The input data are initially split among multiple data feeds and need to be assembled and processed within a few milliseconds. The processing farm is built around commodity PCs which are interconnected with a commercial high-speed low-latency network implementing a torus topology. The thesis presents a system for data traffic management based on a global traffic supervisor and a dedicated control network. The former allocates distributed computing resources dynamically in order to avoid network congestions as well as to balance the load of the system. The latter communicates supervising information to all data feeds in order to initiate a controlled data transfer. A congestion-free system operation is demonstrated in a farm prototype with an integrated hardware-based implementation of the traffic shaping system. Based on parameters measured in the prototype, simulation results of a large-scale processing farm are presented. Both the prototype and the simulation results demonstrate that the system is capable of transferring input data initially split among multiple PCI-based feeding nodes, each one transmitting sub-fragments of 128 bytes, to a specific remote shared memory location at a rate beyond 2 MHz. The obtained results demonstrate the applicability of multicomputer systems based on commodity components for high-rate, low-latency trigger processing if certain care is taken in organizing the actual data transfers. This organization has to ensure efficient event building and appropriate allocation of the available processing resources.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. Introduction

Contemporary experiments, which are related to fundamental research on matter, study reactions that are characterized with very low cross-sections. Considering the Large Hadron Collider (LHC) [1], for instance, and given the cross-section for inelastic proton-proton interactions at 7 TeV $\sigma_{in} = 60\ mb$ and the luminosity $L = 10^{34}\, cm^{-2} s^{-1}$, the expected event rate $L \cdot \sigma$ equals approximately $6 \cdot 10^{8}\, Hz$ [2]. The Higgs Boson, however, for whose existence evidence is essential for certain current and future experiments [3, 4], is expected to be produced at the LHC at rates between 10^{-2} and 10^{-1} per second, depending on its mass.

In order to select particular rare events from the total number of detectable interactions, and thus to extract the physical contents that are interesting and feasible for final storage and off-line analysis, most of the experiments use unique trigger systems. The triggers are typically designed together with the detector systems in order to conform with those requirements particular to the experiment (Figure I-1).

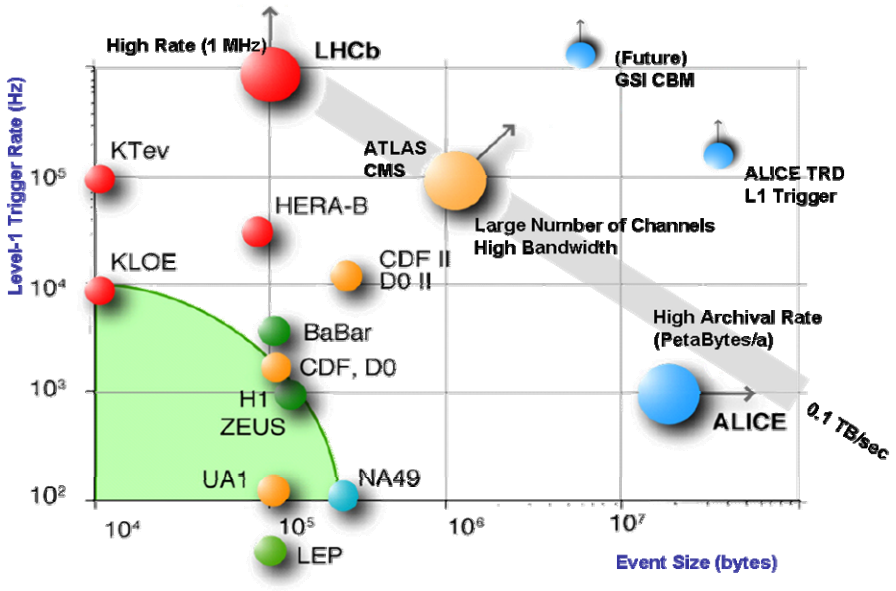


Figure I-1: A summary of some current and future high-energy physics experiments. Upcoming LHC experiments are depicted as larger balls. Different experiments set diverse requirements to the trigger and data acquisition systems. A picture from [5, 6]

Contemporary large-scale detector systems require multi-level trigger systems with a strictly defined hierarchy. First-level triggers operate on relatively small-size high-rate sub-events and are therefore characterized by maximum available trigger latency which is typically a few microseconds. Consequently, such triggers are implemented completely in hardware (Figure I-2), and the trigger decision is based on evident and relatively simple criteria.

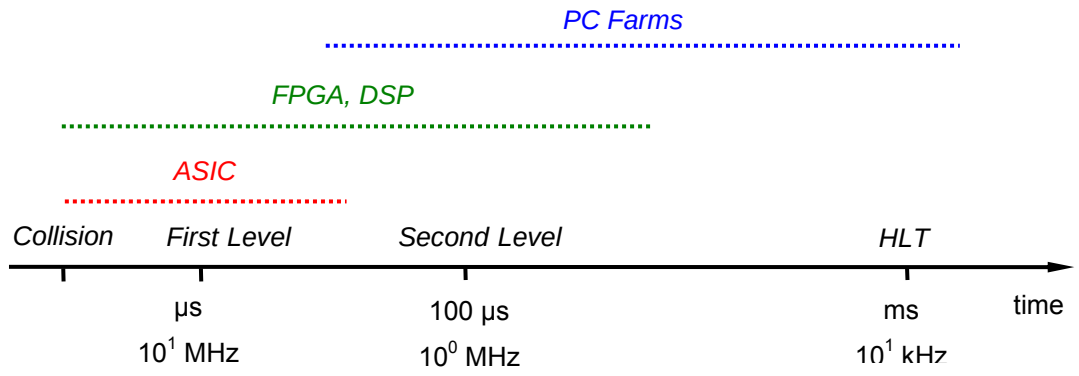


Figure I-2: The trigger-layer latency and the processing rate orders set different requirements to the technology choices for the various trigger levels.

Higher level triggers operate only on a subset of events that are not rejected by lower hierarchical levels. In general, the trigger decision complexity and the available processing time increase with the trigger layer number. Therefore, the realization of higher trigger levels in software is feasible. Clusters of computing nodes are being built to realize real-time or quasi-real-time data processing as well as off-line data analyses [7]. One of the main advantages of such triggers is the flexibility to introduce trigger algorithm modifications later on. Furthermore, compared with custom hardware implementations, the development of software trigger algorithms does not require a special infrastructure. Commodity PCs are being deployed as these are the computing instruments that offer the best price-performance ratio today.

However, different requirements apply to high-level and to lower level trigger processing farms. The LHCb experiment [8], for instance, foresees a three-level trigger system, where the Level-0 is realized by means of custom electronics, while the Level-1 as well as the High Level Trigger (HLT) are implemented as computing farms. The Level-1 trigger processor has to operate on relatively small event data the size of about 4 kbytes at a megahertz input rate. The HLT, on the other hand, uses larger fragments sent by many more data sources, which results in events being one order of magnitude larger than in the respective case. However, these full event data are coming at a much reduced Level-1 trigger acceptance rate of 40 kHz, which results in a significantly lower aggregate data rate than that at the previous stage. Furthermore, a tight latency requirement is set over the Level-1 since the event data need to be buffered in front-end electronics until a trigger decision has been taken. This imposes that some care has to be taken to allow for keeping the latency for the whole process of data transport and event building as short as possible and thus to maximize the time available for the execution of the algorithm. In contrast to the Level-1 trigger, latency constraints are not such a concern in the case of the HLT since event data which have passed through the previous filter stages fit well in the memory available in the computing nodes and even in their processors.

In the next chapter, the basic architecture and the prototype of a high-rate trigger processing farm built around commercially available components are presented [9]. The system has initially been designed to meet the requirements of the LHCb Level-1 trigger [10], where every microsecond new event data the size of about 4 kbyte, that are split among multiple data feeds, need to be processed with a maximum latency of a few

milliseconds. Trigger decision is taken by a software algorithm which, in the case of the LHCb application, reconstructs tracks in the vertex detector.

However, before the trigger algorithm is launched, a full event has to be assembled in a given computing node. This requires all sub-event data fragments originating from approximately twenty data sources to be transferred to the specific node. It should be noted that the time it takes to assemble an event is a portion of the total available trigger level latency.

Simultaneous transfer of data from multiple feeds to a single destination may lead to an overload of the destination's input stage. Furthermore, the operation of a non-load-balanced system at a high rate could lead to an overload of certain network paths beside the target node. In consequence, we observe inefficient system utilization and poor system performance, which is inconsistent with our prime requirements. Hence, the applicability of such a processing farm as a high-rate low-latency trigger processor farm could only be demonstrated with a traffic shaping mechanism which is an integral part of the system.

The current thesis presents the architecture and the prototype of a data flow control system built to realize a congestion-free operation of a computing farm. The latter is loaded at a rate exceeding one megahertz, with the requirement of about 4 Gbyte/s aggregate data rate. The processing farm is based on commodity off-the-shelf PCs that are interconnected with a ring-based network implementing a torus topology. The prime motivation for the development of the processing farm, the boundary conditions, the general system architecture, and the prototype are presented in chapter II. At the end of the chapter, the motivation of the current thesis is presented. The major sources of congestions within the system, and the typical hot-spots are discussed.

The measures taken to prevent congestions as well as the considerations taken into account when developing the prototype of the flow control system are discussed in chapter III. Two hardware devices that were developed to implement certain devices in this work as well as in other high-energy physics applications are presented.
The chapter introduces the concept of the flow control system, and gives more details about the architecture of its main components, which are:

- a scheduling unit which continuously supervises the way the next target node is chosen and thus assigns a global traffic model
- a custom control network which transfers messages produced by the scheduling unit so as to communicate traffic control information to all data feeds
- data feeds which perform data transmission under the control of the supervisor and the control network

Performance results obtained in the prototype with integrated traffic management system are presented and discussed in chapter IV. Based on parameter values measured in the prototype setup, the results of a simulation of a large-scale processing farm are presented in chapter V.

It should be noted that the application of the data flow control system presented in this thesis is not limited to the LHCb experiment. Traffic shaping is prerequisite for any data processing system involving high-throughput data assembling. The setup for shaping and traffic scheduling presented in this work is particularly relevant to event builders operated at rates going beyond the speed limitations of current software-based solutions, which is the case in the LHCb Level-1 trigger processor farm. Another possible application is the planned Compressed Baryonic Matter (CBM) experiment [11] which is one of the major scientific activities at the future international Facility for

Antiproton and Ion Research (FAIR) [12] in Darmstadt, Germany. The CBM experiment aims at investigating the properties of nuclear matter at highest densities and moderate temperatures, which is only little explored so far. In the experiment, event data fragments, the size of about 50 kbyte each, distributed over approximately thousand buffers have to be dispatched to a specific event buffer at a rate in the order of several tens of microseconds (i.e. approximately 1 Tbyte/s total data traffic) before being further processed [13].

II. Architecture of a High-rate Low-latency Scalable Trigger

The following chapter introduces the application which primarily motivated the development of the high-rate low-latency trigger processor designed in [9]. It then briefly presents the computing farm built to meet the requirements imposed by the operating conditions. Finally, the chapter gives reasons for the need of network traffic shaping. It also outlines two modern applications with respect to the measures taken to organize the data transfers through the network.

A. Initial Motivation

Violation of the CP symmetry – a phenomenon which has first been observed in the neutral kaon system, where kaons decay at a fraction of the time into two instead of three pions – is the prime objective of a number of the present experiments [14, 15, 16, 17]. CP violation enables us to explain the dominance of matter over antimatter in our universe today. However, the level of CP asymmetry that can be generated by the Standard Model is insufficient to explain the matter-antimatter ratio that can be observed.

While CP violation in the kaon system is very small and implies theoretical uncertainties, the Standard Model makes precise predictions for CP violation in a number of decay modes in the B-meson system, where larger effects are expected. This makes the B-meson system suitable for studying CP violation and searching physics beyond the Standard Model.

The LHC Collider is a high-luminosity proton-proton collider. Compared to other present or planned elementary particle physics facilities, it will be the richest source of B-mesons. About 10^{12} $b\bar{b}$ pairs are expected to be produced at the interaction point of the Large Hadron Collider Beauty (LHCb) experiment in one year [10].

The LHCb experiment is dedicated to precise measurements of CP violation and to the study of rare decays in hadronic systems which originate from *b*-quarks. In the experiment a spectrometer (Figure II-1) is used to detect particle tracks and to reconstruct B-decay vertices with resolution in the micrometer range.

Although the number of *b*-hadrons produced is relatively big, the amount of interesting events is only a small fraction of the total number of decays. This is due to the small branching ratios in the order of 10^{-5} or less and to the limited detector acceptance. Therefore, a trigger system, which efficiently selects events with B-mesons, is required.

The LHCb experiment foresees a three-layer trigger system [8] which successively reduces the detector data rate from 40 MHz down to 200 Hz for the final data storage and offline analyses. The trigger decision is based on particles with a high transverse momentum and displaced secondary vertices since these characterize events with B-mesons.

The system presented in this chapter has been designed with the requirements of the LHCb Level-1 trigger in mind[1] [9]. A new 4 kbyte event enters the Level-1 trigger every microsecond, which results in an aggregate data rate of about 4 Gbyte/s. A time frame of a millisecond is set for an event to be processed. Within this time frame, the event data originating from about 20 input data feeds have to be assembled in a specific processing node; subsequently, a trigger decision whether or not to discard the event has to be made; and finally, a result message has to be transferred to a result node.

In this chapter, the requirements to the trigger system, the basic architecture, and the utilized data transfer modes are described. Furthermore, the prototype [9] that has been developed is presented. At the end of the chapter, the need for traffic shaping, which motivated this thesis, is discussed.

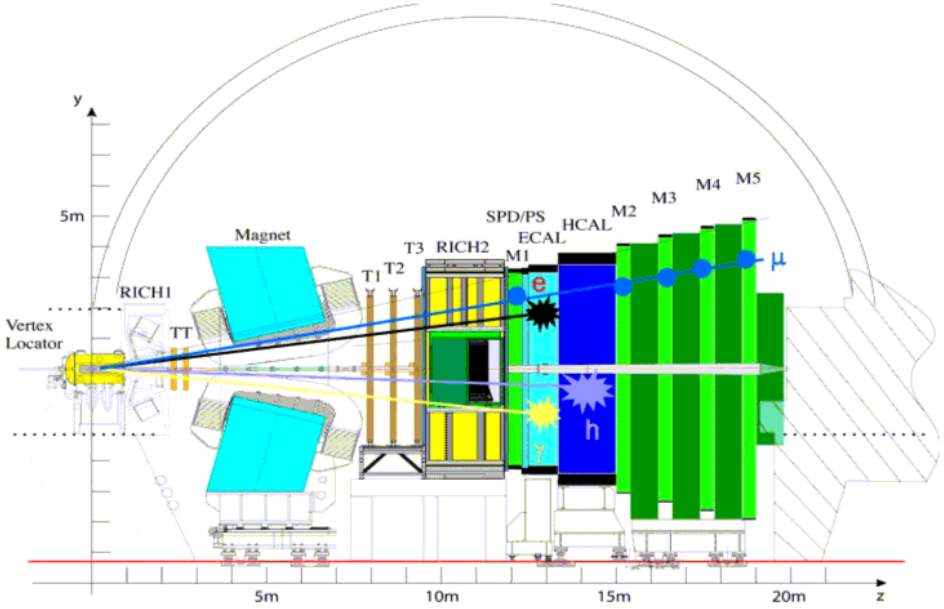


Figure II-1: Since at high energies both the b - and the $\bar{b}$ -hadrons are predominantly produced in the same forward cone [10], the detector covers low polar angles up to 300 mrad in order to achieve a high physics performance. The LHCb detector comprises the following sub-detectors: a vertex detector for a high-precision reconstruction of charged particles tracks near the interaction point; a tracking system comprising four stations (TT and T1-T3) for track reconstruction and a precise momentum measurement of charged particles; Ring Imaging Cherenkov Counters RICH1 and RICH2 for charged particle identification; a calorimeter system (a Scintillator Pad Detector (SPD), a Preshower detector (PS), an electromagnetic calorimeter (ECAL), and a hadron calorimeter (HCAL)) dedicated to the identification of electrons and hadrons with measurements of position and energy for trigger and offline analysis; five muon stations (M1-M5) for muon identification and Level-0 information.

[1] until 2002

1. The LHCb Trigger System

The LHCb experiment uses a three-level trigger system (Figure II-2) to reduce the raw detector data rate of 40 Tbyte/s to a few dozens of Mbyte/s for final storage.

Level-0 operates at the LHC bunch-crossing frequency of 40 MHz. It achieves a suppression factor of 40 by means of three high transverse momentum triggers, which operate on electrons, muons and hadrons, and a pile-up veto, which suppresses events with more than one proton-proton (pp) interaction. The maximum time available for collecting trigger input data, executing the trigger algorithm, and delivering the Level-0 decision to the front-end electronics, is set to 4 microseconds.

Level-1 operates at the average Level-0 accept rate of 1 MHz and is designed to achieve a suppression factor of 25. It selects events which contain secondary vertices. The maximum available Level-1 latency is set to a millisecond[1].

The high-level trigger, initially split into Level-2 and Level-3, uses all sub-detectors' information, and reduces the event rate gradually before the data are written on a permanent storage device for offline analysis. The trigger stage rejects events that are not associated with specific b-hadron decay modes.

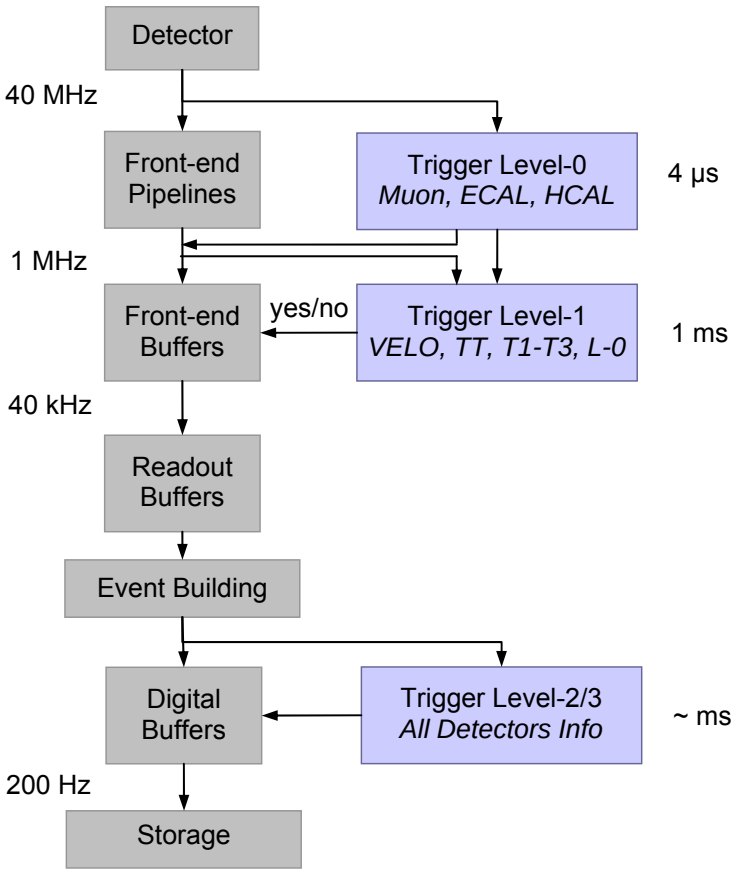


Figure II-2: The LHCb Trigger System in Conjunction with some Data AcQuisition (DAQ) System Elements

[1] As in 2001

2. The LHCb Level-1 and the Requirements for the Level-1 Trigger

The Level-1 trigger processor operates on data produced by the Vertex Locator (VELO) and the tracking stations TT and T1-T3, as well as on summary information of the Level-0. Its aim is to select events with secondary vertices since they are a significant sign for the presence of B mesons.

The main data source for the Level-1 trigger is the VELO detector (Figure II-3). It is a high-precision solid-state track detector that provides information for the precise reconstruction of charged particles' tracks near the interaction point. The vertex detector system also contains a pile-up veto counter, which is used by the Level-0 trigger to suppress events containing multiple pp interactions in a single bunch crossing.

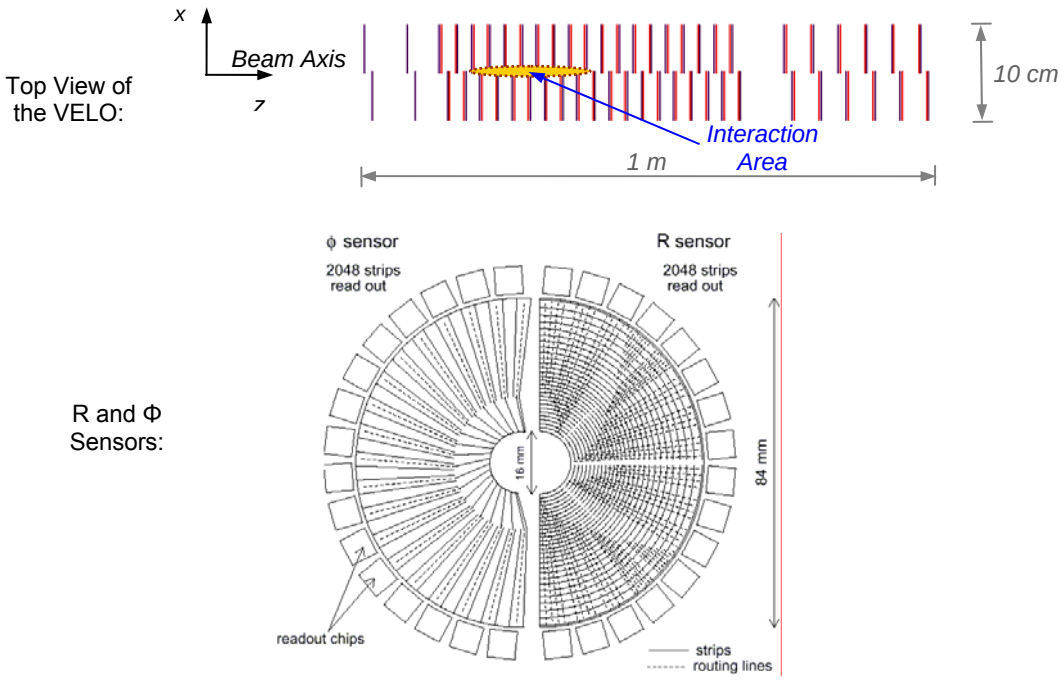


Figure II-3: The vertex locator [18] comprises a series of silicon stations placed along the beam direction. Each station consists of a left and a right detector module. Each module, except those in the first two stations, is composed of two half-circular sensors measuring R and Φ respectively. The first two stations are solely R-measuring stations and are part of the Level-0 pile-up veto counter. Both sensor types feature fine strips which allow for measurements with resolution in the micrometer range.

Data produced by the VELO are read out at the LHC bunch crossing frequency of 40 MHz and buffered in on-detector analog pipelines during the Level-0 processing time of 4 μs. Upon Level-0 acceptance, the data are transferred to Off-Detector Electronics (ODE) for digitizing and Level-1 buffering (Figure II-4). During the time the data are buffered in the ODE units, the latter have to preprocess and transfer these data to the Level-1 trigger, the Level-1 trigger algorithm has to be executed, and the

Level-1 trigger decision has to be sent to a result node. The total time needed for performing all these steps is about a millisecond.

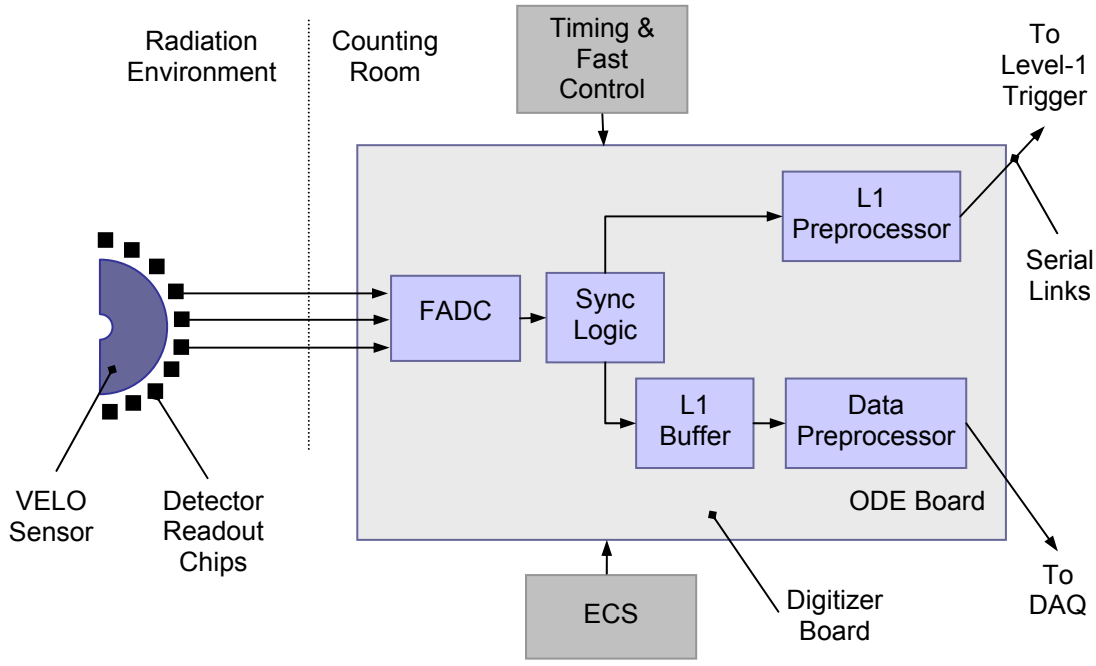


Figure II-4: Front-End Electronics Architecture

Level-1 data preprocessing involves a number of tasks since neither the digitized detector data nor the data format are appropriate to the Level-1 trigger processor. Pedestal subtraction is done since each detector channel has an offset (pedestal) which is measured in a special run and can be subtracted first. Masking of faulty channels prevents wrong detector hit information in case of channel oscillations. Hit detection is performed, whereas a hit is defined as a channel that has an amplitude above a certain threshold value. The latter value, as well as the pedestal data and the faulty channel masking data, can be downloaded through the Experiment Control System (ECS). In case the detector channels are read out in an order that does not reflect the geometrical sequence, a topological re-ordering has to be performed by the Level-1 pre-processor too. Since the Level-1 trigger algorithm requires clusterized data, clusters are encoded as follows: the cluster position is the detector strip position in case of a cluster of one hit; in case of a cluster of two hits, the cluster position is the first position, while an extra bit signals a two strip cluster. Since only two adjacent strips are taken to form a cluster, four hits, for instance, are treated as two clusters of two. Finally, the hit data are encapsulated with a header and a trailer. Every data packet is identified by an event ID.

After preprocessing has been performed in the ODE boards, the event data are transferred to the Level-1 trigger by means of serial links [19]. The Level-1 input event rate equals the Level-0 accept rate, which is 1 MHz on average, with peak values up to 1.1 MHz. Event data are fed into the Level-1 trigger through about 20 links, each

transferring sub-event data of 200 byte on average, which results in a total event size of approximately 4 kbyte and an aggregate Level-1 input data rate of 4 Gbyte/s.

Detector data shipped by the front-end electronics are collected in the Level-1 input stage which comprises about 20 [9] readout units [20] that build and buffer the sub-events before the latter are forwarded for Level-1 processing in a computing farm (Figure II-5). All detector data for a specific event are dispatched to a single processing node due to their small size and the relatively small number of tracks that have to be reconstructed, which makes classical parallel processing inefficient.

The track finding algorithm, in conjunction with the VELO detector (Figure II-3), aims to reconstruct the position of the primary, also called production vertex, to detect those tracks which do not originate from the production vertex, and to reconstruct the secondary, b-hadron decay vertices. B-hadrons which have all their products within the spectrometer's acceptance are typically produced within a polar angle below 200 mrad. Thus, the projection of the impact parameter of the b-decay products to the production vertex in the RZ-plane is large. The algorithm exploits this feature as it initially searches for tracks in the RZ-projection by using only a fraction of the measurements provided by the detector. This accelerates the track finding and increases processing speed. The accepted two-dimensional track candidates are then used to reconstruct the 3-D position of the primary vertex. In order to find the b-hadron decay vertices, the track finding algorithm only reconstructs those tracks in 3-D space that are characterized with a significant RZ impact parameter. Finally, the algorithm searches for two 3D-tracks to form a secondary vertex.

The main motive for the implementation of this solution is the fact that such a system could be built from low-cost commodity components without an early technology freeze, and that a later introduction of trigger algorithm modifications would be possible. Furthermore, compared with hardware, the development of trigger algorithms in software does not require special infrastructure and allows the realization of more complex trigger algorithm calculations.

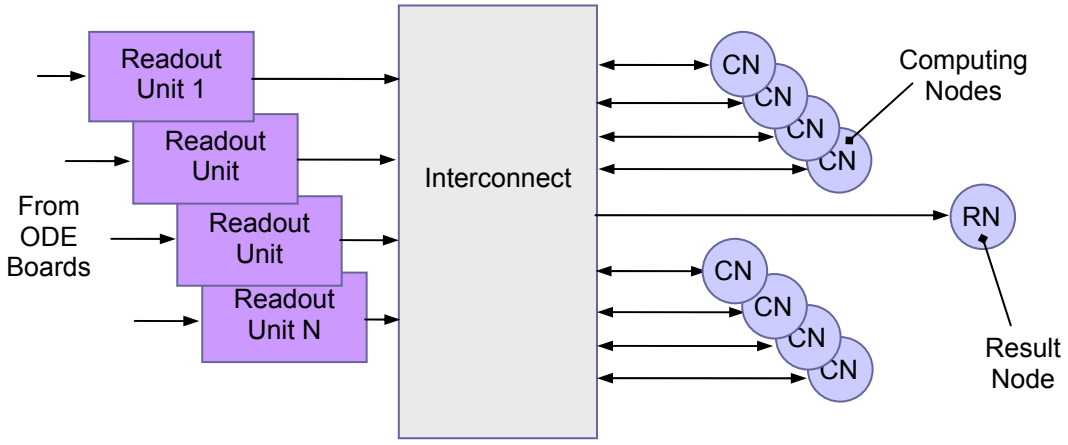


Figure II-5: An Abstract View of the Trigger System Architecture

The trigger algorithm is invoked in a Computing Node (CN) when all readout units' contributions associated with a specific event are written into the CN memory. The trigger algorithm has to be invoked every microsecond according to the Level-1 input rate. Hence, the number of computing nodes is the product of the average processing time and the input rate.

Upon trigger algorithm completion a result message is produced and sent to a Result Node (RN). Since the processing time depends on the event size and is therefore prone to fluctuation, the result messages do not arrive at the RN node in the order of the input. Therefore, the result node reorders the messages and produces the final Level-1 trigger decision. The trigger signal is distributed to the front-end electronics through the experiment Timing and Fast Control (TFC) system [21].

The Level-1 trigger operating conditions impose a number of requirements. Since small data packets have to be sent from the readout units to the computing nodes at high rate, the data have to be transferred with a minimum overhead because any additional traffic could influence the data transport performance considerably and thus become an obstacle to satisfying the input rate requirement. The deployed interconnect has to be characterized by high speed and low latency since the system has to provide a total data rate as high as several Gbyte/s and still induce the minimum latency to meet the strict latency requirements. A reliable data transport has to be completely implemented in hardware since any software associated with data transport handling would be too slow. The readout units have to be capable of forwarding a complete sub-event within less than a microsecond since a new transaction needs to be initiated every microsecond, on average. Since the detector architecture, the event size, and the trigger algorithms vary, the system has to scale in respect to the aggregate data rate and computing power. The architecture has to allow the addition of computing power on demand without considerably affecting the throughput and latency requirements, which means without causing effects of saturation. A robust system has to be built out of commodity components wherever possible to reduce the price.

B. General System Architecture

The system is based on commodity PCs interconnected with a high-speed low-latency network that is commercially available, in order to meet the trigger system requirements presented in section II.A.2. The network topology chosen is a two-dimensional (2-D) torus (Figure II-6) due to its node number scalability and avoidance of single points of failure. The alternative switch-based solution would increase the cost in the case of large systems. In addition, large switches imply undesirable centralized hot spots.

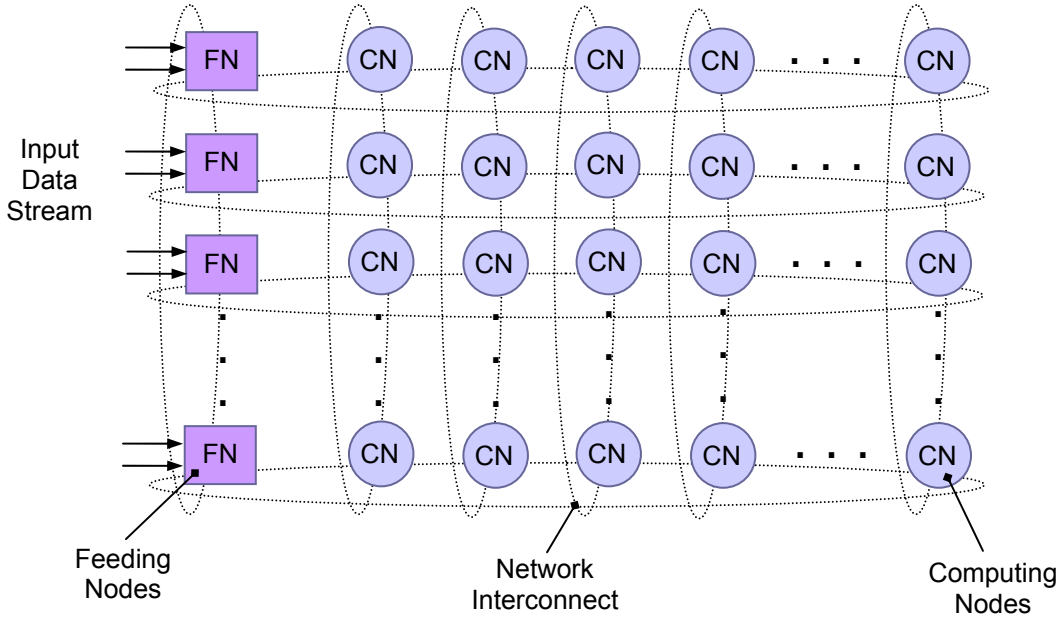


Figure II-6: The basic system architecture is a 2-D torus comprising Feeding Nodes (FN) and computing nodes that are interconnected by a high-speed low-latency network.

Transfer of data is realized using the shared memory concept since this does not involve any software overhead. Access to remote locations in a shared memory system is transparent since a global address space is set. The resulting programming model is simple. Remote locations are mapped to the local address space. Thus read and write access to a specific local address is translated into access to a specific remote location.

The network technology used is the Scalable Coherent Interface (SCI) [22]. However, the system architecture 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 commercial implementation of the SCI is deployed [23, 24]. The shared memory paradigm is implemented in hardware. In SCI, read and write transactions are standardized to carry data of either of the following size: 0, 16, 64, 256 byte. However, the deployed implementation does not support packets of 256 byte of data. A 128 byte packet has been defined instead. The most efficient transfer of data in that case can be achieved with 128 byte packets.

In this case, utilization of SCI is considered feasible mostly due to the following reasons:

- There is no software overhead involved in data transport, which means that the CPU power can be used for calculations and does not have to be spent for data transfer;
- The delivery of data packets is guaranteed on a hardware level, which means no CPU load is caused by handling a loss of data in software;
- Transfer of a data packet from the feeding node's buffer memory to the local SCI Network Interface Card (NIC) (Figure II-7) within less than a microsecond is possible.

1. The Scalable Coherent Interface

The Scalable Coherent Interface (SCI) has been developed specifically for high-performance cluster computing and defines a set of packet protocols for both the shared memory and the message passing programming models. The primary objective of SCI is to provide:

- high throughput (in the Gbps range)
- low latency (in the low microsecond range)
- low CPU overhead for communication operations

The SCI provides bus-like services. However, while a bus does not scale, one SCI key feature is its scalability in various aspects:

- Adding nodes leads to aggregate bandwidth growth (if the system is properly load-balanced).
- The distance between SCI nodes may vary widely, depending on the physical layer implementation.
- The maximum number of nodes supported is 64k, which accounts for the massive cluster realizations.

An SCI node can either be a workstation, a server, a single processor with its cache, a memory module, an I/O controller or device, or a bridge to another interconnect or bus.

Hardware-based distributed shared memory is utilized by SCI. The remote memory is transparent to software and CPUs. Access to remote memory is handled by the SCI hardware. Memory operations can be executed from user-level without involving the operating system. Thus, latency in the low microsecond range is achieved. Reliability of data transfer is guaranteed in hardware.

An SCI node operates split into request phase and response phase transactions. That way multiple outstanding transactions per node are possible. Every node provides its own elasticity buffers. Thus, transactions with a very high rate can be initiated using the interconnect in a pipelined fashion. As a result the link utilization increases.

In-order delivery of the packets is not guaranteed in the SCI. If a heavy-loaded responder is unable to accept a packet (due to its input queue being full) it generates a so-called echo packet that notifies the requester of a necessary packet re-transmission. In the mean time a later packet en route can overtake the rejected one if an empty buffer slot can be found.

The SCI standard foresees two allocation protocols. The bandwidth allocation protocol aims to assign bandwidth to a sending node so that it can transmit its packet in case of heavy bypass traffic. The queue allocation protocol aims to make traffic to heavily-loaded nodes possible: it insures that the input queue of a receiver reserves some space for re-transmitted packets.

Various network topologies are allowed by the SCI. The most popular of them are the ring- and the switch-based topologies. The basic SCI topology is a ringlet. The combination of multiple rings results in two- or multi-dimensional torus topologies.

C. Data Transfer Mode

Data transfer performance tests using Programmed I/O (PIO) and software-initiated Direct Memory Access (DMA) have been made and presented in [9]. PIO implies a CPU overhead since the CPU itself performs data transfer to and from a device. In case of DMA, a device equipped with a DMA controller is capable of direct access to memory (either the main system memory or memory available on another device), thus transferring data without CPU intervention. The DMA mode is usually inefficient in the case of small data sizes since the DMA controller has to be configured prior to transfer execution.

The results show poor performance in the case of PIO [9]. Software-initiated DMA except for small data packets shows a much better performance.

The boundary conditions presented in section II.A.2 require that each feeding node transfers its sub-event contribution within less than a microsecond. This does not refer to the time needed to deliver the sub-event data to the computing node but rather to the forwarding of these data to the input stage of the local network adapter, which afterwards transmits the data to the remote location (Figure II-7). The high data rate and the small data packet size imply that this transfer is characterized by high performance and involves as little overhead as possible. Furthermore, the time it takes to transfer the event data to the computing nodes is part of the limited trigger layer latency. Hence, the transfer of data from the Level-1 input stage to the computing nodes has to be fast. Data transport must not involve software overhead since CPU cycles in the computing nodes have to be used exclusively for the execution of the trigger algorithm software. Therefore, a third method of data transfer has been utilized: the so-called hardware-initiated DMA. Data are transferred by means of a DMA engine located in a dedicated hardware agent on the local bus (Figure II-7). The DMA engine is triggered by hardware means, without CPU intervention, and writes data directly into the memory-mapped input buffer of the network adapter[1]. The latter transfers those data to the remote node.

A descriptor characterizes every single action performed by the DMA engine. It consists of parameters such as the type of access, the target address[2], the block size to be transferred, etc. Therefore, this method requires the values of those parameters to be specified and passed to the device that accommodates the DMA engine. This happens during the initialization phase, which sets up the shared memory. During this phase the computing node exports a memory region, whereas the feeding node imports it. Once the desired shared memory model is established, initialization software configures the DMA engine and the data transfer can start.

The hardware-initiated DMA does not involve software. When the DMA engine and the network adapter are located on the same bus (Figure II-7), no host bridge is involved except for bus arbitration. Thus the transfer mode provides very high data rates even for small data packets. A 128 byte packet can be transferred in a single burst.

[1] this mode of operation is also known as device-to-device copy
[2] physical address required

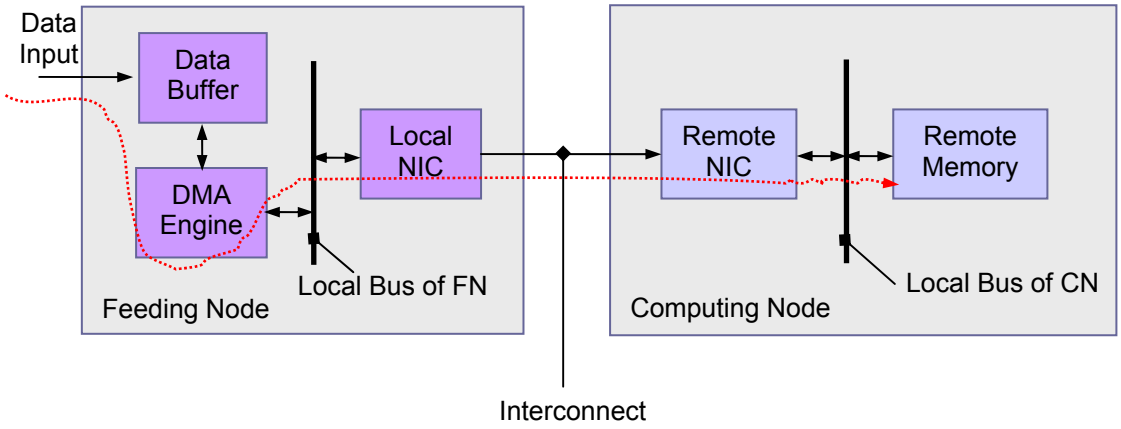


Figure II-7: Sub-Event Data Path

During the development of the trigger prototype, the Peripheral Component Interconnect PCI [25] was the bus commonly used in commodity PCs. Hence, the network adapter and the dedicated DMA agent are PCI devices. The transfer of a 128 byte packet is performed as follows. The data are located in the DMA agent's data buffer. Once the DMA agent has acquired bus mastership, it fetches the data from the buffer and pushes them through the local expansion bus directly into the network adapter's memory to an address that corresponds to the remote target. The transfer of 128 bytes over the local bus is performed in a burst with a minimum protocol overhead. Once the data are copied into the local NIC, the latter performs the transfer to the remote NIC without any software involved. The transfer of 128 bytes over SCI is made in a single network packet with practically no overhead. The remote NIC writes the data – again in a single burst – directly into the host's main memory, where they are accessible to the computing node's CPU. The implementation of the DMA engine is discussed in section III.E.4.

D. Data Routing

In a 2-D torus, data can reach their target in different ways. The routing strategies supported by the deployed implementation are presented in [26].

Dimensional routing is used in the application that is presented. A packet is first transferred along the horizontal X-ring until it reaches its target column. Thereafter, the packet is routed to the vertical Y-ring and forwarded along that ring until it reaches its target node. Figure II-8 shows several concurrent routes in a 3 by 4 torus.

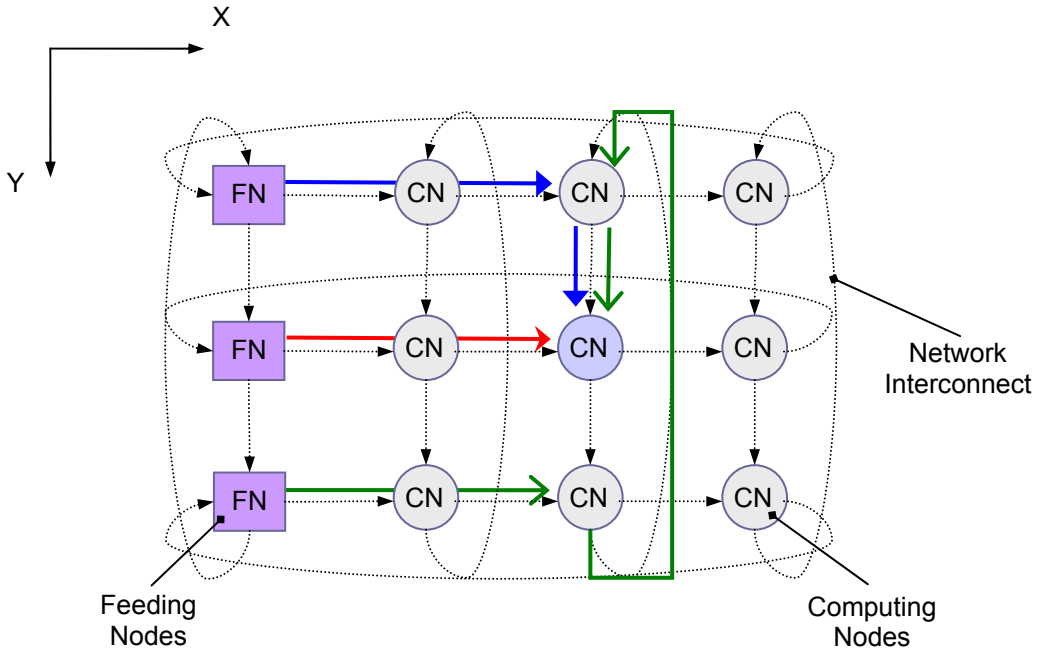


Figure II-8: Dimensional (X-Y) data routing is used in the torus. For every event, multiple data packets, each one originating from a different feeding node, are collected at a particular target computing node. The network paths of the different packets are shown by different-coloured arrows with different arrowheads.

E. The System Prototype

A 30 nodes Linux cluster has been built [9] for the purposes of evaluation and demonstration (Figure II-9). Different cabling schemes have been set up for different tests. They all implement a 2-D torus as shown in Figure II-6. Some of the nodes are dedicated feeding nodes, depending on the particular setup. Section III.E deals with the feeding nodes as well.

Twenty-four nodes are equipped with ServerWorks IIIHE-SL, the remaining 6 nodes with ServerWorks IIIHE chipsets [27]. Both chipsets support 64-bit/66 MHz PCI buses. Each node is equipped with two Pentium III processors: 24 run 800 MHz CPUs, 6 of them 733 MHz processors. For the experiment the nodes were equipped with 512 Mbyte RAM, a current SuSE Linux distribution, an up-to-date kernel version, and sufficient disk space. All compute nodes use Dolphin PCI 64/66 PCI SCI cards built around the SCI link controller LC3 [26]. The SCI adapter is located in a 64-bit/66 MHz PCI slot.

Figure II-9: The System Prototype at the Chair of Computer Sciences at KIP, University of Heidelberg

F. The Need for Traffic Shaping

In the system considered in this work, data corresponding to a particular physical event are initially distributed among a number of feeding nodes and need to be transferred to a specific remote memory location. Then the data can be efficiently accessed by a processor designated to execute the event selection algorithm. The process of event data assembling is known as event building. It has to be performed for every event that enters the system and accounts for a part of the total event processing latency. Hence, the time it takes to build an event in a computing node has an impact on the overall system throughput.

The way data are transferred from the feeding nodes to the computing nodes is discussed in section II.C and illustrated in Figure II-7. Independently of the specifics of any particular data transfer mode, the way to move data between two computers over a computer network consists in the following: data are first transferred to the NIC of the sender; the network adapter then transmits the corresponding packet(s) through the network to the NIC of the destination; finally, the remote network adaptor transfers the data to its host.

A network adaptor can be considered as a simple queuing system as shown in Figure II-10. Network packets arrive at the receiving NIC at an average rate λ. If the server part in the model is idle, a packet is served immediately, that is, the transfer of the packet to the host is undertaken immediately. In case the server is transferring a packet while a new one arrives, the latter is buffered in a queue until the server completes the ongoing transfer. The average time T_s it takes for the server to transfer a packet determines the theoretical maximum input rate that can be handled by the receiving system, which is $\lambda_{max} = 1/T_s$. The service or departure rate $1/T_s$ is limited by the throughput on the dataway between the NIC and its host, which is the throughput achieved on the expansion bus of the computer.

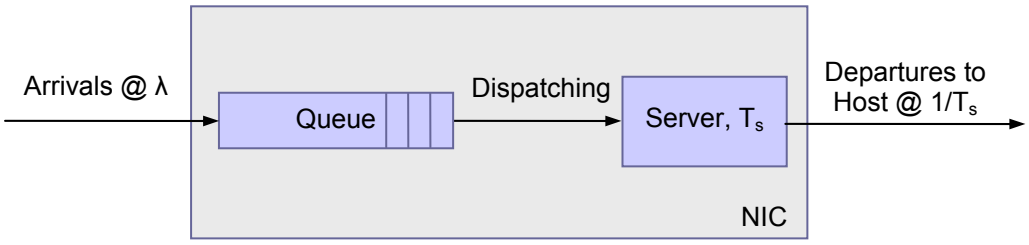


Figure II-10: A Network Adaptor Presented as a Queuing System

The event building requires N feeding nodes to send their data contributions to a single receiving node. This would require handling packets arriving at the receiver at a rate that is N times higher than the receiver's maximum theoretical input rate λ_{max}, as far as the feeding and the receiving nodes are based on an expansion bus of the same type and the senders are fully utilized. The receiver is thus prone to overloading since the communication pattern involved in an event builder implies that the arrival rate exceeds the departure rate of the receiving NIC. In fact, even a single node sending packets at a

rate λ_{max} leads to overloading the receiver since the server becomes saturated. When the server is 100% utilized, the queue length grows without bound. In a real system, where queue sizes are limited, this would inevitably result in packet loss or, if the network provides reliable data transport, in packet retransmission. The latter would, however, increase the transfer time and thus result in poor throughput. Even assuming an infinite queue size, operating the server near saturation would result in unacceptable packet latency since the corresponding queue would become very large. High network delays are inconsistent with the system requirements in the application considered in this work since maximum available latency is set for an event to be processed by the trigger level. Hence, the receiver must operate at average arrival rates below its theoretical maximum input rate λ_{max} so as to ensure that any potential contention does not turn into congestion (i.e., that multiple packets possibly contending for the incoming link of the receiver get into the waiting line until the contention subsides instead of being discarded due to buffer overflow).

Especially in the case of low-latency networks, it is of prime importance to keep packet waiting times as short as possible, which implies that queue lengths are typically small. Therefore, it is not reasonable for the respective hardware implementations to support extra buffering space. For example, the I/O stage of a network node as those used in the prototype presented in section II.E is shown as a network of queues in Figure II-11. The receiving queue of the network link chip can accommodate up to eight packets. However, one buffer slot is always reserved[1]. Hence, a receiving node can accept up to 7 packets in its incoming queue. So, it can be expected that such devices provide rather small queues, which makes the need for traffic shaping still more significant.

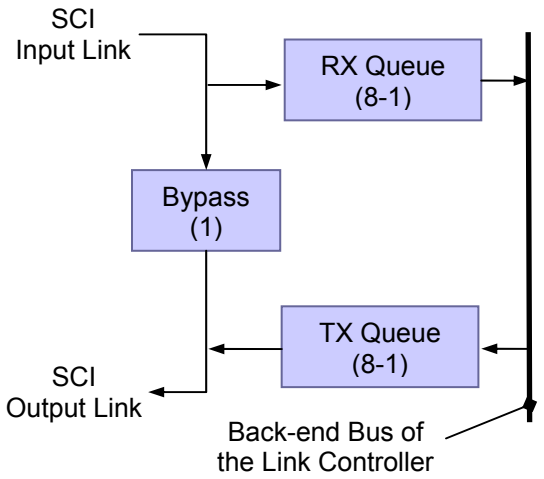


Figure II-11: The I/O stage of an SCI node presented as a network of queues: packets that have to take off the ring pass through the receiving queue; all packets sent by the node go through the transmitting queue; a packet which has to by-pass the node is stored in the bypass queue if the node is currently transmitting its own packet. The one-dimensional case is depicted. In case of a two-dimensional SCI NIC the picture has to be duplicated: two link controllers, one for each dimension, share a common back-end bus.

[1] for the opposite packet type, i.e. respond-send

Consequently, the trigger processing farm requires a mechanism that manages to reduce the arrival rate of approximately $20\lambda_{max}$ to less than λ_{max} so as to keep the feeding nodes from overrunning a given computing node. The arrival rate at the destination can be limited by delaying the transmission of each one of the event fragments. However, since there is a continuous flow of new event data into the feeding nodes, it is necessary to dispatch every subsequent event to an alternative destination node so that the processing farm can sustain the arrival rate of the events. This raises the question of how to choose the destination the next event has to be sent to. A factor which has to be taken into consideration when choosing the next destination node is the presence of network bottlenecks beside the end node. When multiple nodes transmit data to a specific receiver, all the transmitters potentially use a common network segment. A given transmitter may have enough capacity on its outgoing link to send its packets, but somewhere in the middle of the network, its packet flow encounters a segment that is being used by other senders, too. That intermediate segment can turn out to be a bottleneck that causes congestions to occur upstream from the end node. Thus, in order to ensure a congestion-free and efficient data transfer, there must be a mechanism which guaranties that the aggregate data transfer rate through a link segment does not exceed its bandwidth.

Figure II-12 shows an example of the processing farm, wherein two feeding nodes are located in a torus row for a better network link utilization[1]. Two feeding nodes labelled as 1 and 2 transmit data to two receiving nodes according to two different scenarios: 1) the receivers are nodes №14 and 15, which are located in the same ring together with the transmitters; thus, no re-routing is needed; 2) the receivers are nodes №23 and 33 so that an intermediate re-routing node, node №13, is involved in the data path. Re-routing requires that a packet is taken off the X-ring and passes the local bus of the link controller before entering the Y-ring (Figure II-13). It has been demonstrated [9] that the passage through the local bus of the link controller restricts the maximum data transfer rate: the maximum aggregate data rate measured in the second scenario is more than 5% lower than in the first scenario. Thus, the traffic directed to one vertical ring is potentially a cause for hot spots. Hence, the next computing node has to be allocated in a way that controls the load of a vertical ring and ensures that the required aggregate data transfer rate in a vertical ringlet is lower than the bandwidth provided by the link controller's local bus.

Another aspect of the way the next destination is chosen concerns the balancing of the load of the computing resources available in the end nodes. An event built in a given computing host with minimal delay would still encounter large total processing latency in case the computer has not processed previous events yet. In the meantime there may be idle computers, which may account for non-efficient system utilization although the traffic shaping mechanism has managed to prevent inefficiencies caused by network congestions. Hence, there is need for a mechanism which shapes the event traffic so as to appropriately and efficiently allocate network as well as computing recourses. At the same time, the traffic shaping policy needs to be based on simple rules so that the mechanism can be implemented in hardware since a new computing node has to be selected every microsecond, on average.

It should be noted that if a saturated node (either an end node or an intermediate network component) issues some kind of flow or congestion control, the non-balanced system is prone to odd behaviour. Issuing control could rapidly congest the entire

[1] The theoretical peak bandwidth provided by the SCI link is in excess of 600 Mbyte/s, while one feeding node sending 128 byte sub-event data packets with a megahertz rate accounts to approximately 130 Mbyte/s. Therefore, placing two feeding nodes in a torus row leads to a better link utilization.

network since a restriction of the data flow from another node could lead to an overflow of the buffers in that node. Hence, traffic control needs to be applied when packets enter rather than when they have already entered the network.

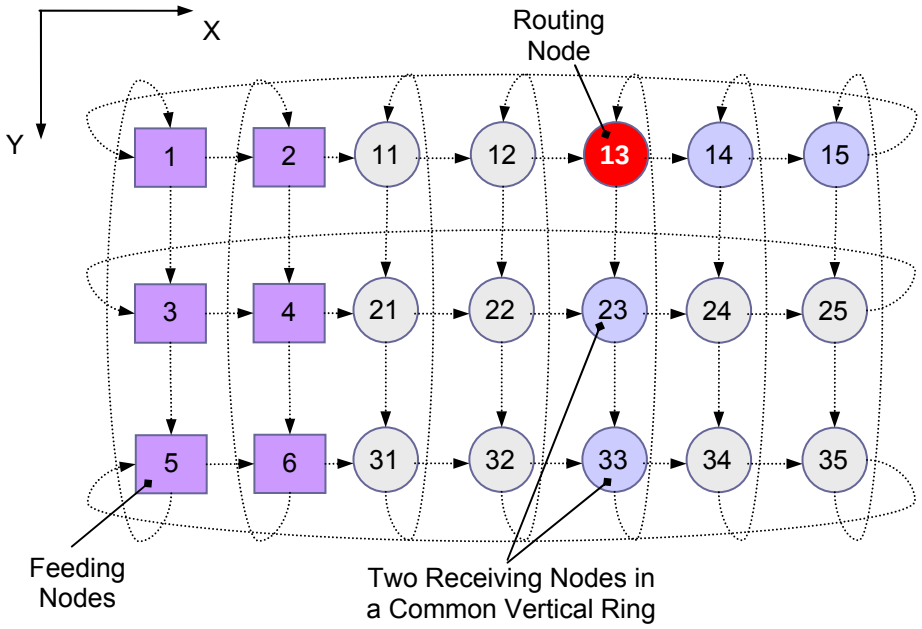


Figure II-12: Performance limitations can be observed when two nodes sharing a vertical ring receive data concurrently. The maximum data transfer rate is determined by the performance of the local bus of the link controller.

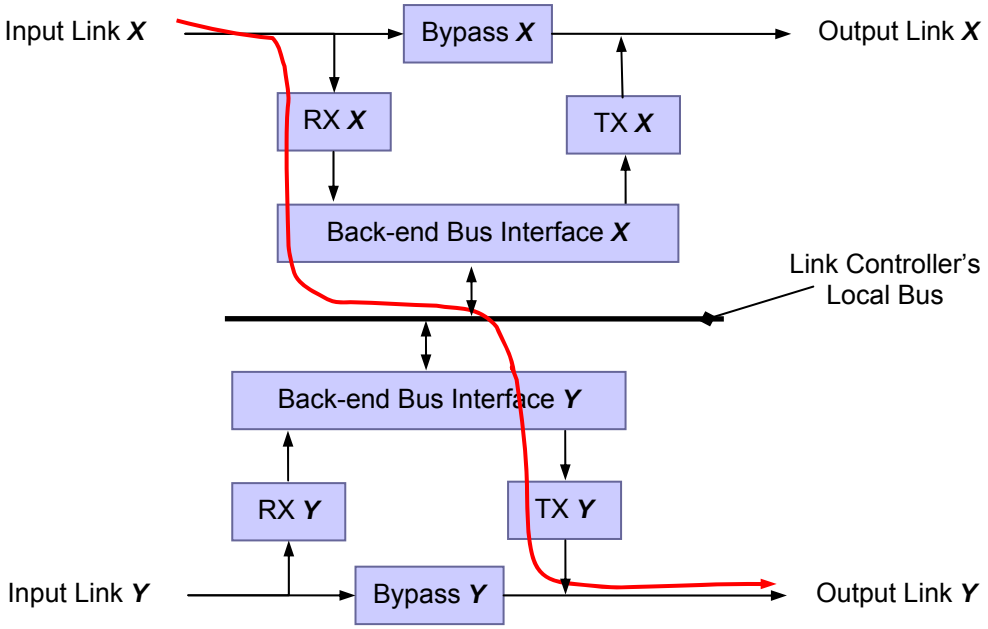


Figure II-13: Re-routing requires that a packet passes through the link controller's local bus, whose maximum transfer rate in a 2-D torus has been determined to be 450 Mbyte/s [9]

G. Traffic Shaping in the CMS Experiment

The Compact Muon Solenoid (CMS) [28], one of the upcoming LHC experiments, aims at achieving a broad physics program by employing a large-scale general-purpose detector with nearly complete solid-angle coverage. Driven by a number of considerations and design principles [29], the CMS collaboration has chosen to realize a two-stage trigger system. All detector data corresponding to an event selected by the first-level trigger have to be moved to a single location for further filtering performed by a high-level trigger. This implies that the DAQ system must provide the means to feed data, the size of about 1 Mbyte, from more than 500 front-end units to one out of about 1000 processors. Merging of new event data has to be initiated at the Level-1 output rate of 100 kHz. In order to realize the event building process, the CMS experiment builds a three-dimensional switch fabric (Figure II-14). However, in an event builder, all data sources and sinks need to operate continuously at rates as close as possible to 100% of the available network throughput. Furthermore, all sources must communicate data belonging to the same event to one and the same destination. Hence, in order to prevent inefficiencies due to congestions, the CMS Event Builder fabric is supplemented by the appropriate measures for traffic shaping.

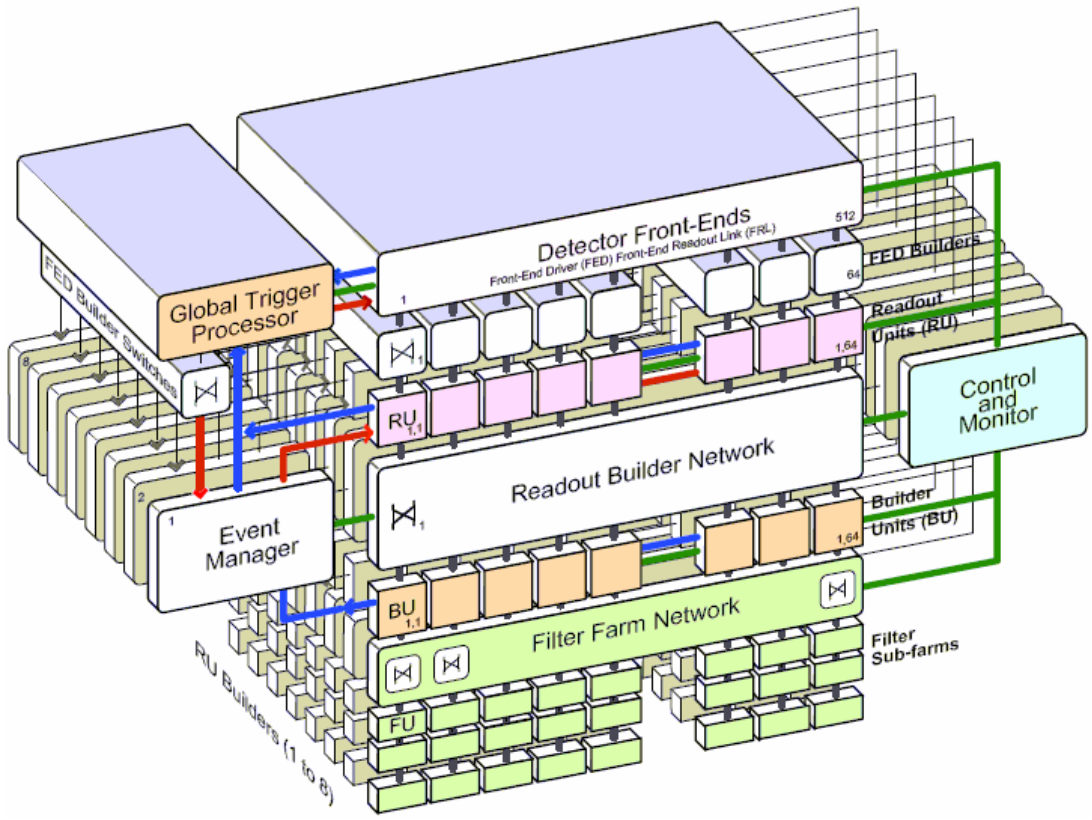


Figure II-14: A 3-D View of the CMS Event Builder [29]

The event building is performed in two stages (Figure II-15). First data merging is done in the so-called 'Front-End Driver Builders' (FED Builders) where the event data fragments from 8 front-end links are grouped into a larger data block, referred to as a 'super-fragment'. The latter is then transferred to a Readout Unit (RU). Thus, at the end of the first stage, given that the total number of front-end links is 512, all detector data for a specific event are contained in 64 RUs. In the second stage, referred to as the 'RU Builder', the 64 super-fragments for a given event are further merged into a full event in one out of 64 Builder Units (BUs). Note that the system can have one or more, up to eight, RU Builders. One of the advantages of this modular architecture is its scalability, which can be exploited already at the beginning of data taking when the accelerator will operate at a reduced luminosity and thus only a few RU Builders will be needed. In a system operating with one RU Builder, there will be only one output port in each FED Builder switch connected to its corresponding RU. In a bigger system, there will be up to 8 data sinks connected to each FED Builder, which results in the 3-D view of the Event Builder shown in Figure II-14. Hence, data from different events can be sent to different RUs, thus maximizing the bandwidth of the readout.

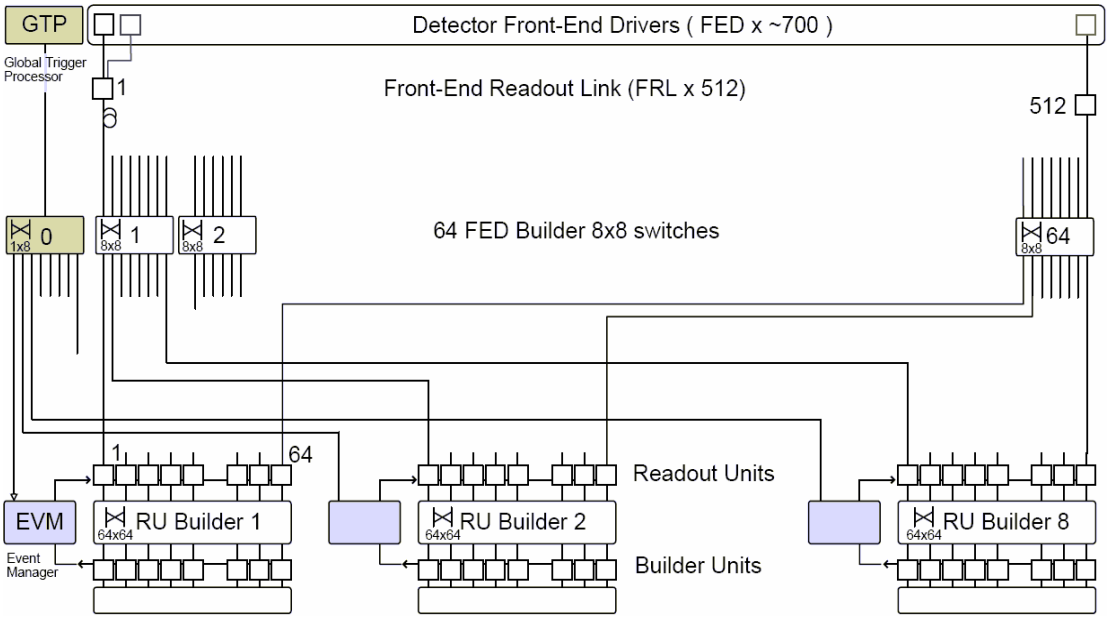


Figure II-15: Conceptual Design of the CMS Event Builder [29]

Both the FED Builder and the RU Builder have to deal with transferring data fragments from multiple sources to one destination. These are two unique examples of a many-to-one scenario which require appropriate measures for flow control and traffic shaping to allow efficient network utilization and successful event building. The CMS experiment realizes the two stages of the event building process in different ways.

Each one of the FED Builders has to merge event fragments from 8 front-end links into a super-fragment (s-fragment). Furthermore, it has to multiplex the s-fragments on an event-by-event basis between N ($1 \leq N \leq 8$) RUs. The interconnectivity between sources and destinations is realised by means of a switch (Figure II-16). Event

fragments the size of 2 kbyte, on average, are read out from front-end buffers and arrive at each one of the switch inputs at the Level-1 accept rate of 100 kHz. Hence, in order to maintain the data rate of the DAQ system, a switch port must be capable of sustaining the data rate of 200 Mbyte/s.

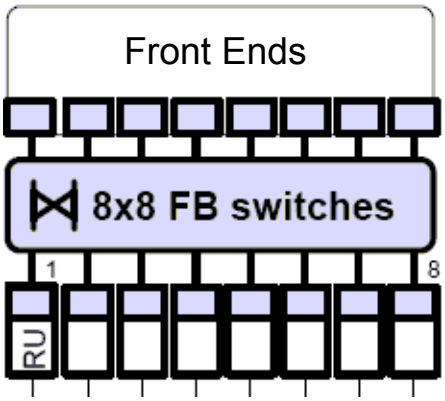


Figure II-16: The CMS FED Builder

In reality, the event fragment size will vary significantly across the inputs of an FED switch, depending on the group of detector channels the front-end link originates from. Each FED Builder switch will connect to front-end links with both small and large event fragment sizes grouped in the way that the inputs to the second stage, the RU Builder, are reasonably balanced.

In addition to differences in the average amount of data across the switch inputs, the actual amount of data for a particular input link will fluctuate statistically from event to event. For efficient switch utilization there must be some packet-level algorithm that provides an arbitration mechanism for a certain output link and ensures that all other switch links are used concurrently. Hence, one must either apply some traffic shaping policy or operate the FED Event Builder without traffic shaping at the expense of reduced switch utilization. The CMS collaboration has chosen the second option in order to avoid complicated traffic patterns at this early stage of the DAQ system. For a given trigger, the data fragments from the 8 inputs arrive at the output of the switch in an arbitrary order, depending on the switch load, the timing of the start of sending and the size of the fragments. Simulation studies and test-bench measurements have shown that the switch utilization is roughly 50% when transferring variable size fragments. This requires a doubling of the initially required switch throughput of 200 Mbyte/s per port.

The routing of an event through an FED Builder is a static function of its event number and the number of RU Builders available in the DAQ system. The algorithm used to determine the destination for every event, assuming more than one RU Builder installed, can vary from a round-robin across all switch outputs in use to cases designed to make efficient use of the system when the RU Builders are not identical in performance. The algorithm can also be used to redirect data traffic away from a faulty RU Builder. Additional requirement on the FED Builder with respect to data flow control is that it should provide backpressure to the system components which are on

upstream to the FED Builder. So, no data will get lost in case of congestion in the system downstream after passing the FED Builder.

All s-fragments for a given event are further merged into a full event, the size of 1 Mbyte, in one out of 64 BUs (Figure II-17). Assuming that all 8 RU Builders are installed, each one of them has to operate at an input rate of 12.5 kHz, on average. The event building process requires scheduling of the transfers of s-fragments to prevent simultaneous transmission from all RUs to a given BU and thus to avoid network inefficiencies or even packet losses[1] caused by congestions at the switch output ports.

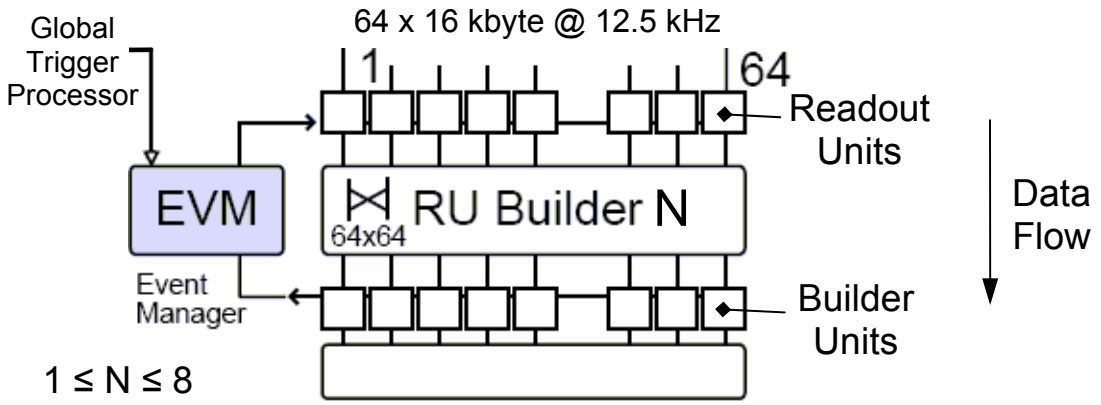


Figure II-17: The CMS RU Builder

Traffic in the RU Builder is shaped by the destinations, the BUs, aided by an Event Manager (EVM). The latter associates every event that enters the RU Builder with an event identifier and later distributes these associations to BUs requesting new events. A Builder Unit requests a new event from the Event Manager as soon as it has free resources, thus performing dynamic load balancing. Upon receiving a request, the EVM replies with an event identifier which is further used by the requesting BU to pull the s-fragment associated with that identifier from each RU. To prevent congestions at the switch output port, the BU requests s-fragments from each RU sequentially.

In reality, a BU does not request event identifiers on an event-by-event basis. Instead, it groups multiple requests as soon as a given fraction of its buffer space has been released. Thus, the load on the Event Manager, which has to communicate with each one of the BUs individually, is reduced. All the BUs are building events at the same time. Thus, event data are flowing from all switch inputs to all outputs and the switching fabric is operated at full capacity. Each BU is building multiple events concurrently in order to be able to overlap data input with processing performed by computing sub-farms in the system downstream.

[1] Data can get lost due to congestions if the switch does not implement endpoint-to-endpoint flow control.

H. Traffic Shaping in the ATLAS Experiment

ATLAS (A Toroidal LHC AparatuS) [30], another general-purpose detector at LHC, was designed to exploit the full discovery potential of the collider and thus to achieve a broad spectrum of physics goals related to known as well as to new physics processes. Hence, in order to maximize its physics coverage, the experiment aims at using inclusive triggers as much as possible in its event-selection strategy. The first level of event selection (Figure II-18) will be realized by the Level-1 trigger (LVL1), implemented as a system of purpose-built hardware processors [31], whose maximum available latency and maximum output rate are 2.5 μs and 100 kHz, correspondingly. Further event reduction to a rate in the order of 100 Hz, that is reasonable for final storage, will be performed by the High-Level Trigger (HLT) [32]. The HLT is based on commodity computing nodes and comprises two elements, the Level-2 trigger (LVL2) and the Event Filter (EF). The former will only operate on a few percent of the event data and will provide high rejection power using fast, limited precision algorithms. The latter, in turn, will refine the LVL2 selection, using the full event data in conjunction with more precise algorithms and having fewer constraints on the latency.

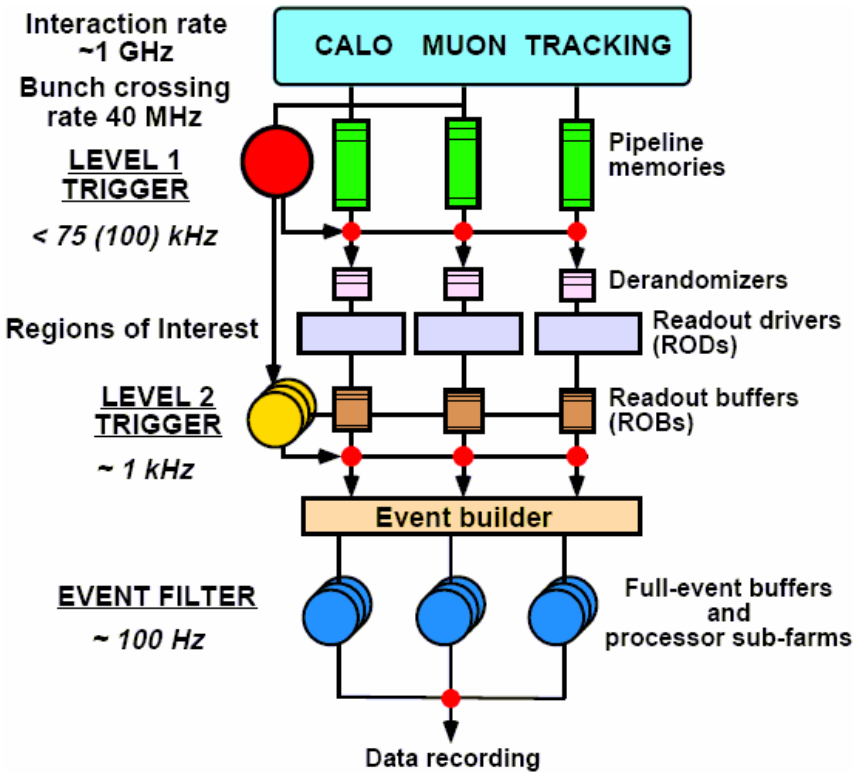


Figure II-18: Block Diagram of the ATLAS Trigger/DAQ System [31]

Events selected by LVL1 are read out from the front-end electronics into approximately 1600 ReadOut Buffers (ROBs). All of the detector data for the selected event, the size of about 1.6 Mbyte, are held in the ROBs either until the event is rejected

by LVL2 or until all data fragments have been transferred to a particular EF processing node, in case the event is selected by the Level-2.

The LVL2 trigger operates only on approximately 2% of the full event data located in so-called Regions-of-Interest (RoI) which are identified by the LVL1 trigger and pointed to positions in the detector where the trigger has found interesting features. The RoI information is transferred by the LVL1 to the LVL2 through a direct interface provided by the so-called RoI-Builder (RoIB). The main function of the RoIB is to collect the RoI information from the various LVL1 sub-systems for every LVL1 accept and to produce a single data record, which it then relays to the LVL2-SuperVisor (L2SV) (Figure II-19). To meet the input rate requirements in software, the L2SV is implemented by a processor farm of 10 nodes. Each one of the L2SV nodes, however, will communicate with only one sub-set of the LVL2 processing nodes. Hence, the choice of an L2SV node will affect the load balancing between the LVL2 processors (L2P). The RoIB would normally select a next L2SV on a round-robin basis; however, more complex algorithms, including use of the LVL1 trigger type, can be realized as well. The L2SVs exercise a load balancing function, too. The supervisor is designed to either select a processor from an available pool through a simple round-robin algorithm or to examine the number of queued events and to allocate the event to the least loaded processor.

Upon receiving a new assignment, an LVL2 processing node has to collect RoI data, the size of a few kbyte, stored in a few ROBs of interest (Figure II-19). This is achieved by the exchange of control messages and subsequent event data messages. Traffic is shaped by the receiver: an L2P requests the next event fragment after it has received the previous one.

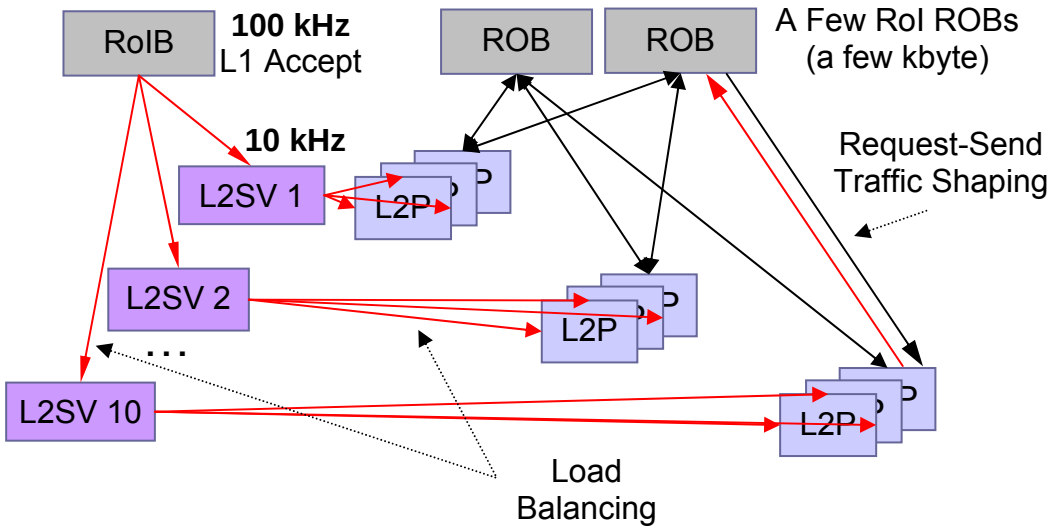


Figure II-19: RoI Building in the ATLAS Level-2 Trigger

Upon LVL2 algorithm completion, an L2P transfers its decision back to the L2SV. The latter in turn informs a Data Flow Manager (DFM) whether the given event data have to be transported further to the next trigger selection stage, the EF, or promptly cleared from the ROBs. Each event accepted by LVL2 has to be fully assembled in a particular EF input stage node referred to as a Sub-Farm Interface (SFI) node (Figure II-20). The event building is initiated by the DFM which allocates an SFI according to a load balancing algorithm. The exchange of control and data messages between the DFM, executing certain supervisor functions, the data sources (ROBs) and the data sinks (SFIs) is realized through a common switching network. Events are built concurrently into all SFIs at the overall rate of approximately 3 kHz which is the average LVL2 accept rate. The building of an event consists in collecting all the event fragments initially located in multiple distributed ROBs into a single destination. This process, however, can lead to hot spots and congestions even in a high bandwidth interconnect since multiple nodes attempt to use the same data path segment concurrently. This, in turn, results in performance or data loss and therefore requires certain traffic shaping measures. Although being receivers, these are the SFIs which actually control the transfer of event fragments through the ATLAS event builder. An SFI requests the fragment stored in the next ROB after it has received the data from the previous one, which prevents overloading the input stage of the destination as well as the switch I/Os. For efficiency reasons, the SFI can build more than one event in parallel. An event successfully built by the SFI is further assigned to one amongst a number of EF processing nodes (EF sub-farm) associated with that SFI. Transport of full events within an EF sub-farm is supervised by internal data flow control.

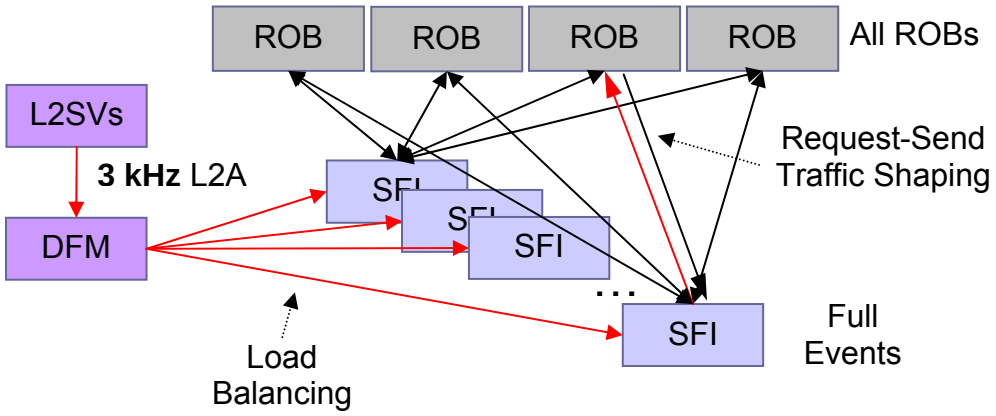


Figure II-20: Full Event Building in the ATLAS HLT Event Filters

III.Data Flow Control System Architecture

Chapter II outlines two modern applications that shape the network traffic by software means, which distinguishes them from the application considered in this work, where a solution is needed to manage with the megahertz transaction input rate. The following chapter deals with the measures taken to satisfy the needs for a controlled transfer of the data from the system front-end to the computing nodes at the required rate. It first exposes the concept of a hardware-based system for traffic shaping and then gives more details about each one of its components. Furthermore, the chapter discusses the considerations which guided the realization of the traffic shaping system. It also presents a versatile hardware platform which was developed for the purposes of the current work as well as for a number of additional applications, some of which are outlined, too.

A. The Concept

The trigger processing farm presented in chapter II requires a flow control mechanism that prevents congestions (as discussed in section II.F). Moreover, the data traffic has to be scheduled when data packets enter the system since applying flow control when the packets have already entered the system may congest the entire network.

The developed flow control system performs traffic shaping by means of a scheduling unit that controls the transmission of event data executed by the feeding nodes (Figure III-1). For every event that enters the system, the scheduler chooses a specific computing node so as to guarantee congestion-free network utilization. Every computing node sends a notification to the scheduler when the event processing is completed. Thus, the scheduler keeps track of all computing nodes available in the system.

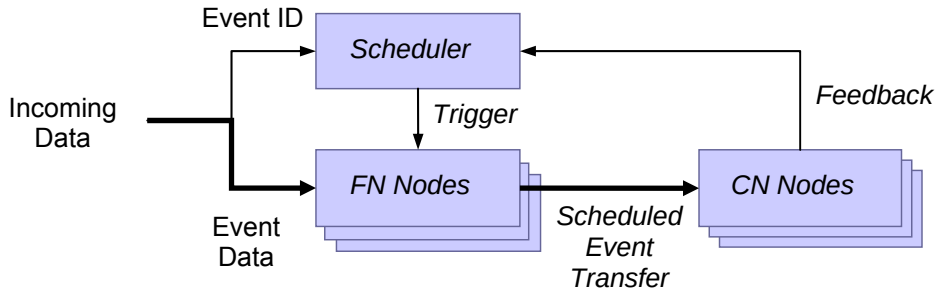


Figure III-1: Logical view of the flow-controlled system. The transfer of event data from the feeding nodes to the computing nodes is made under the supervision of a scheduling unit. The scheduler maintains a list of available computing nodes and chooses the next receiving CN in a way that prevents congestions in the system.

The scheduler triggers the feeding nodes by means of flow control messages that carry information about the address of the computing node chosen to process the next event. The flow control messages are transferred between the scheduler and the feeding nodes through a dedicated flow control system network which implements a ring with unidirectional data flow (Figure III-2). Upon receiving a message a feeding node decodes it, initiates the transfer of its sub-event to the specified computing node, and forwards the message to the next feeding node in the chain (Figure III-3). When all feeding nodes have transferred their sub-events, a full event is built in a given computing node and the event processing starts. After the completion of the trigger algorithm run on the computing node, the latter transmits a unique feedback message to the scheduler through the torus interconnect network in order to request a new event.

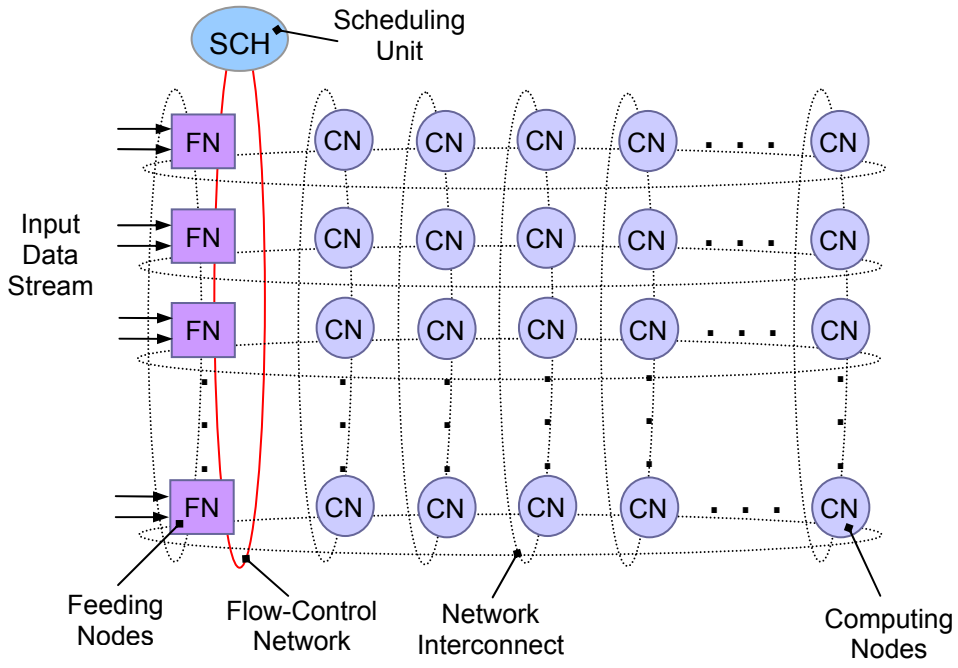


Figure III-2: The flow control system employs mainly two devices, i.e. a centralized scheduling unit and a custom scalable point-to-point serial network. The former realizes the dynamic allocation of free computing nodes. The latter interconnects the scheduler with the feeding nodes in order to initiate the transfer of data to a specific computing node at a specific time.

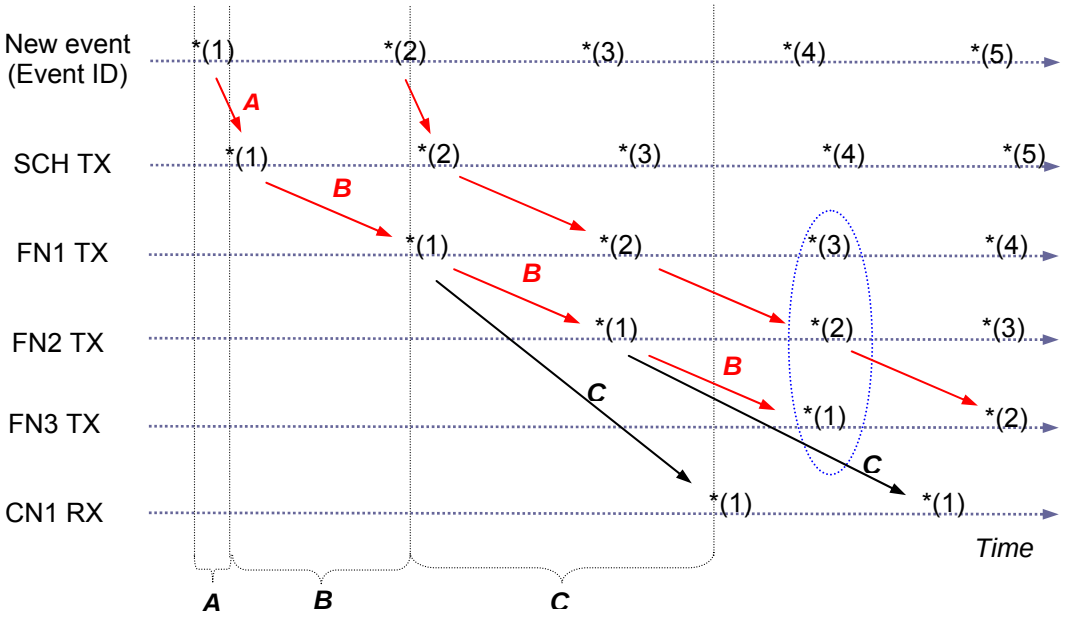


Figure III-3: Upon receiving a new event the scheduler transmits a flow control message to the first feeding node (FN1) in the chain (*A* denotes the latency of the scheduler). After a certain amount of time, *B*, the message arrives at FN1; FN1 sends its sub-event and forwards the control message to FN2 (*B* denotes the latency of a flow control message between two adjacent nodes in the flow control chain plus the time the sub-event transfer from the DMA engine to the local network adapter (Figure II-7) takes). When FN2 sends its first sub-event, FN1 transfers the next one to another node. Thus the feeding nodes never send data simultaneously to the same receiver. The time interval *C* denotes the latency of the packet through the network, which differs due to different network paths.

B. The Scheduler

The scheduler is the traffic supervisor in the system. It has two major functions which are closely coupled together. One of them concerns network congestion avoidance. The scheduler has to dispatch event data in a way which prevents congestions in the system. Therefore, it has to execute a scheduling algorithm which takes into consideration the communication pattern as well as the major bottlenecks in the system. The second function concerns the load balancing. The scheduler has to allocate computing resources so as to ensure efficient utilization of the available processing power. Advances in processor speed and improvements of event selection algorithms with respect to execution time can be of benefit only if the computing power available in the system is efficiently used, which requires a well balanced loading of the processing farm. For appropriate load balancing, the scheduling algorithm has to take into account the actual distribution of computing power in the (asymmetric) system in addition to the considerations concerning congestion avoidance.

The scheduler has to maintain a current list of available destinations in order to choose a specific receiver for every event that enters the system. Hence, it needs to interface all computing nodes and has to receive and to sort feedback messages that carry the addresses of the computers requesting new data. The messages are randomly

arriving since the algorithm processing time varies between successive events. The way the addresses are sorted and stored has to agree with the network topology and the scheduling discipline chosen in order to allow for a fast operation since a new destination has to be selected every microsecond, on average. Furthermore, the scheduler has to be as little intrusive as possible with respect to the latency introduced since the time it takes to initiate an event transfer accounts for the total trigger level latency.

Once having selected a receiver, the scheduling unit has to communicate its address to all feeding nodes in the system in order to initiate a controlled event data transfer. This is done by means of a control message sent through the flow control system network. The latter is a custom dedicated network which has to introduce completely predictable delays since the messages it transfers have control as well as timing function. Timing plays a crucial role in avoiding network congestions caused by multiple event data fragments sent simultaneously to the chosen receiver. It ensures that the fragments are being transmitted successively, i.e. that the transmission of each packet for a particular event is being reasonably delayed to shape the traffic at the receiver. However, this packet delay has to be optimal since it increases the time the event is being built, i.e. it postpones the invocation of the event selection algorithm while the event is occupying the limited-size front-end buffers (Figure II-4). The scheduler has to interface the flow control network in order to feed it with flow control messages. Furthermore, it has to receive and analyse all messages sent back from the feeding nodes in order to check the status of the control chain since the data integrity in this part of the system is of uttermost importance.

A fundamental requirement to the scheduler is also its flexibility. The scheduling unit is the global traffic controller of a scalable processing cluster. Consequently, there must be the ability to straightforwardly adopt the scheduler to various system sizes. System size may change, for instance, if more computing power or more input bandwidth is needed. Considering a 2-D torus topology, this would mean adding torus columns and rows, respectively. Such changes may be required in the initial phases of an experiment, when certain parameters are not fixed yet (e.g. detector architecture, event size, trigger algorithm), as well as later on if some of the requirements change. However, flexibility may also be needed in applications comprising multiple sub-farms of different size, each one having its local traffic supervisor. In that case, a configurable design of the scheduler will just be customized and deployed in each one of the sub-systems.

1. Structure and Operation

The structure of the scheduler is constructed in a way that the device is simple, fast and flexible. Simplicity is a property that, in this case, contributes to high-speed operation and low latency of the device. Both features are considered significant since every microsecond a new event enters the system, which has to be processed within a few milliseconds. Flexibility is required since the scheduler has to be easily adjustable to the actual size of the processing farm.

The operation of the scheduler is explained in the following steps (Figure III-4):

Initialization: In the initialization phase every computing node registers by sending a unique message to the scheduler. These messages are transferred through the torus interconnect and written into a buffer for destinations (1). Immediately after receiving such a message the scheduler processes it and stores the address of the computing node identified by the message in a buffer for queued destinations (2).

Configuration: After the initialization of the system, the scheduler can be configured and enabled through its Control and Status Registers (CSR) (3).

Computing Node Selection: After the user has enabled the scheduler, the core of the scheduler requests the address in the feeding nodes' buffers where the next event to be transferred is stored (4) and the address of the computing node that is chosen to process the event (5). The address of the computing node is obtained from the block with queued destinations, according to certain scheduling algorithm (6). As soon as the destination address is granted, the scheduler is ready to trigger event transmission.

Flow Control Message Transmission: When a new event enters the system (7) a trigger to the core is produced (8). However, if the previous step was not completed, for instance, due to temporarily unavailable computing nodes, the event ID is stored in a queue for pending events. When the core is ready to serve a pending event it triggers the flow control network interface unit (9). The latter then encodes the selected transmission buffer address (10) located in the feeding nodes, and the chosen receiving node address (11). Finally, it transmits the flow control message that is produced to the first feeding node (12).

After triggering the flow control network interface unit the core of the scheduler proceeds to steps (4), (5) and (6). So, when the next event enters the system, the scheduler is ready to dispatch it. When a computing node has completed event processing it sends a feedback message back to the scheduler to request new event data (1, 2).

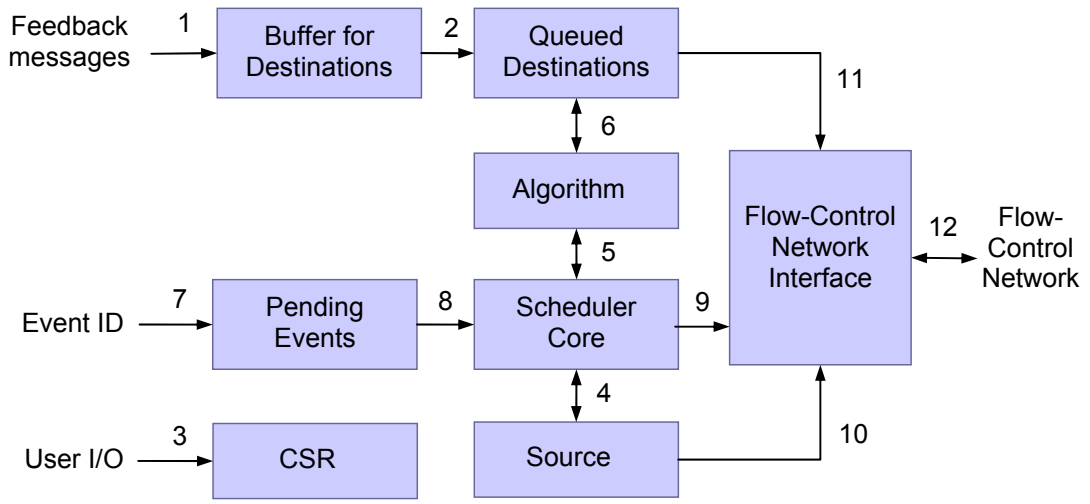


Figure III-4: Structure of the Scheduler

2. Scheduling Discipline

Scheduling discipline is the algorithm used to distribute event data among computing nodes which simultaneously and asynchronously do request them. The purpose of the scheduling algorithm is to select the next destination node so as to ensure a congestion-free system operation, given the many-to-one traffic pattern, and a well balanced loading of the distributed computing resources. There are two major causes for hot spots in the system as discussed in section II.F: one is the limited number of network packets that can be buffered in the input queue of a network node (Figure II-11); the second is the limited bandwidth provided by the back-end bus of the link controller in a network node (Figure II-13). Hence, the data flow control system has to guarantee that no computing node in the system has to receive data sent by more than one feeding node at a time. Additionally, it has to ensure that the required aggregate data transfer rate in a vertical ringlet is less than the maximum transfer rate provided by the link controller's local bus. This is achieved by applying the following flow control rules:

- A feeding node does not forward a flow control message to the next feeding node before it has completed the sub-event data transfer [1] initiated by that message. Thus, the arrivals of the sub-events at the receiver do not overlap (Figure III-3). This measure can only prevent congestions at the receiver if the scheduler dispatches subsequent events to different computing nodes.

- The scheduler does not send subsequent events to computing nodes residing in one and the same torus column. This measure controls the load of a vertical ring in order to ensure that the traffic directed to a vertical ring does not overload the link controller's local bus.

In the test set-up (section IV.A), the next column is chosen in a round-robin manner. The next node within a column is chosen in a first-come-first-served manner. This is done by means of a round-robin counter which points to the scheduled column (Figure III-5). The address of the computing node last used in the selected column is read out of the corresponding queue and encoded in the flow control message that is going to be sent to the feeding nodes. The Round-Robin (R-R) counter is controlled by the algorithm state machine (Figure III-6).

[1] i.e. the completion of data transfer from the local DMA engine to the local network adapter (Figure II-7)

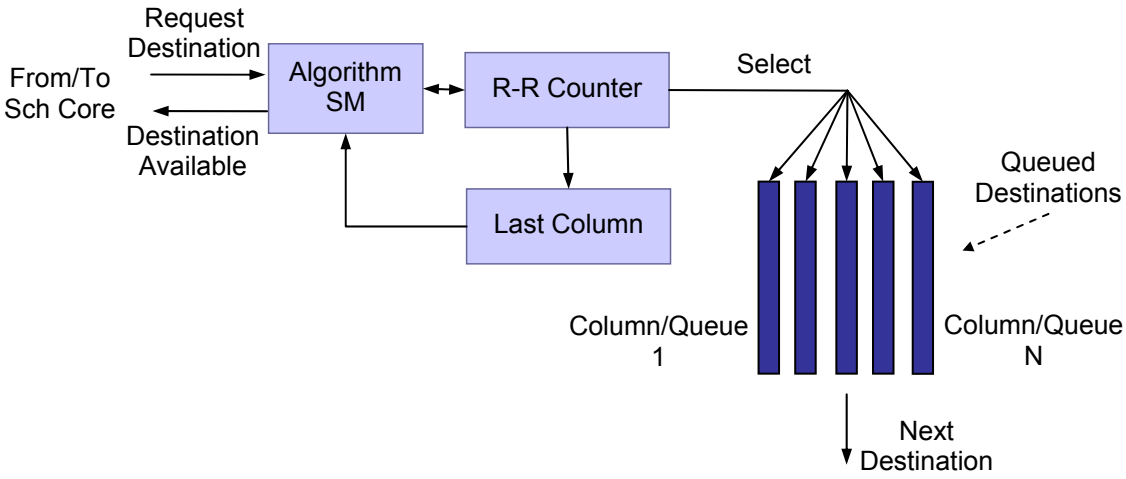


Figure III-5: The figure shows the architecture of the 'Queued Destinations' unit and the 'Algorithm' unit in Figure III-4.

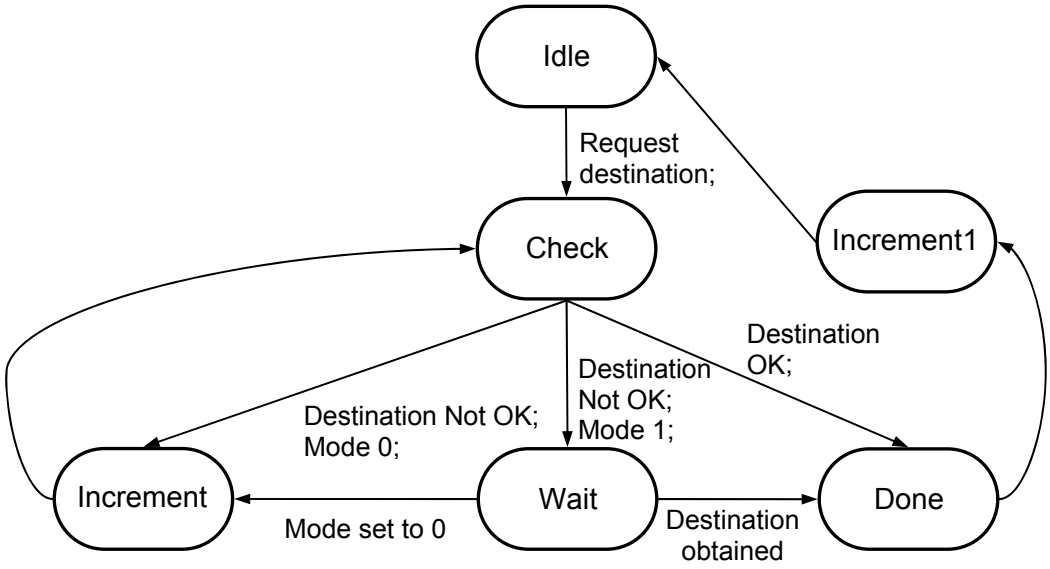


Figure III-6: Algorithm state-transition diagram. When the core of the scheduler requests a destination, the state machine checks whether the selected queue contains available destination addresses. If not, the R-R counter is incremented. The latter, however, only happens if the mode of operation set allows a torus column to be skipped (mode 0). The destination obtained is granted to the core of the scheduler only if it resides in a different column than the last one that was used.

The major function of the scheduling unit is to assign a global traffic model which prevents congestions in the system. However, it also exercises load balancing function in order to get efficient computing resource utilization. Selecting the next torus column on a rotating basis gives equal priority to all of the columns. However, in reality the system may comprise columns with widely differing processing power. This may happen, for instance, due to adding computing power (achievable through adding torus columns) some years after the initial processing farm has been built, or due to partial

upgrading. Scheduling on a round-robin basis across the columns in such a system would disregard the fact that computing power is not equally distributed. Such a system would require appropriate asymmetric load, which can be realized by assigning a fixed ratio to cause the columns with higher computing power to get a greater share of the workload than the others. A more sophisticated solution, however, would be to select the next column taking into account the demands of the torus columns (i.e. the occupancy of the queues in Figure III-5) without breaking the rules preventing congestions. In that case the scheduling unit does not need to be 'aware' of the actual distribution of computing power across the torus columns and can dynamically adjust itself to different processing power distributions.

a) Scheduling Discipline in a System with Two Feeding Nodes in a Row

It has been demonstrated [9] that a system with two feeding nodes in a torus row (Figure III-7) for better network link utilizations[1] is in fact characterized by a non-efficient link utilization. This is due to the fact that the whole traffic originating from the feeding node located upstream (the one far left in the figure) goes through the feeding node located further downstream, which results in very heavy by-pass traffic that prevents the node that is 'willing' to send data from actually transferring them.

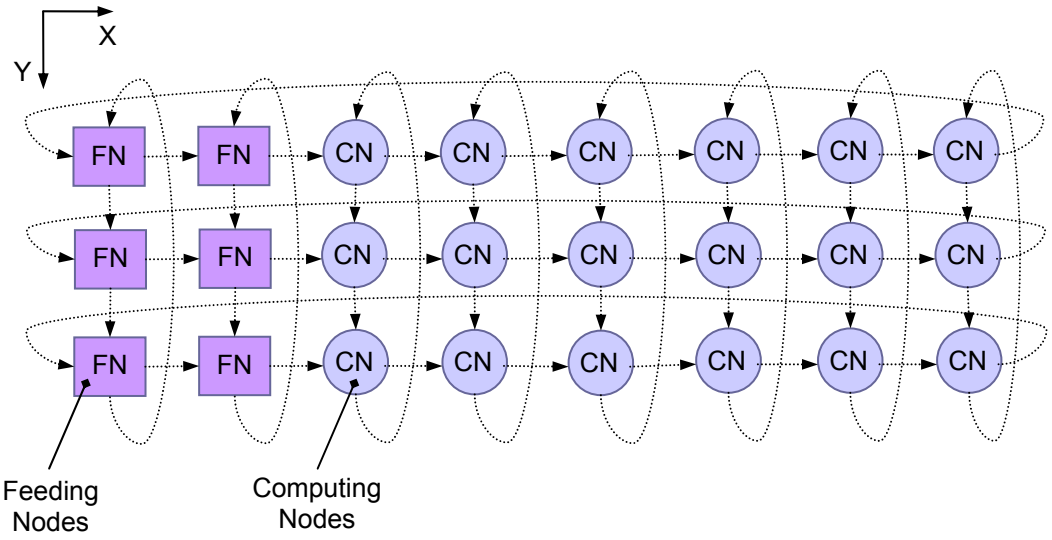


Figure III-7: By-pass traffic has priority over transmission. Therefore, one of the two feeding nodes per row is unable to send its packets freely due to heavy by-pass traffic originating from the other one. Thus the architecture shown in the figure does not demonstrate the expected data rate.

[1] The SCI link speed supported by the deployed network adapters is 667 Mbyte/s, whereas the PCI bandwidth sustained is about 300 Mbyte/s; Except when using two DMA engines in an interleaved mode as discussed in [9], the data transfer rate between the feeding node's DMA engine and the local network adapter card (Figure II-7) is less than 300 Mbyte/s (section IV.B)

The feeding nodes do not have to be located next to each other. They can be displaced (Figure III-8) to resolve the problem caused by heavy-loaded by-pass queues. However, this is only possible if the proper scheduling discipline is applied: alternating partitions *A* and *B* when assigning next destination node would result in reducing the by-pass buffer occupancy of a feeding node's network adapter since every feeding node has to by-pass half of the rows traffic.

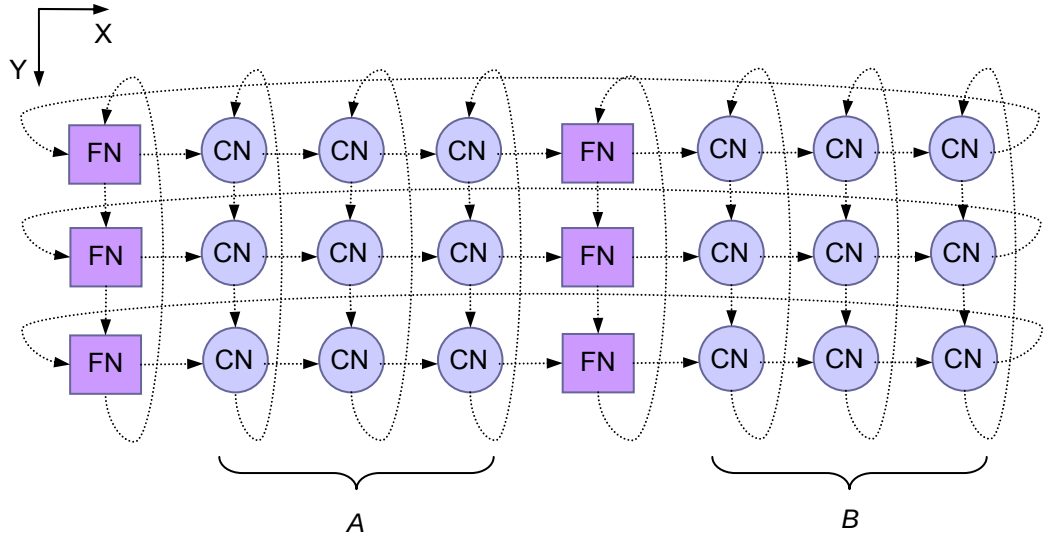


Figure III-8: The architecture with displaced feeding nodes allows overcoming the heavy by-pass problem that characterizes the architecture shown in Figure III-7 if the system is properly load-balanced.

3. Latency of the Scheduler

The core of the scheduler is a state machine which interacts with all other units (Figure III-4) and thus defines the latency of the scheduler. The latter accounts for part of the total trigger system latency that is limited to a few milliseconds. Consequently, the scheduler is required to be as little intrusive as possible with respect to the latency introduced.

The main state machine utilizes the time between two subsequent events to obtain the next computing node (Figure III-9). The core of the scheduler requests the algorithm unit to provide a computing node address immediately after the scheduler has been enabled or the previous event dispatched. When the next event enters the system the scheduler is already prepared to dispatch it. So, it immediately initiates the transmission of the corresponding flow control message.

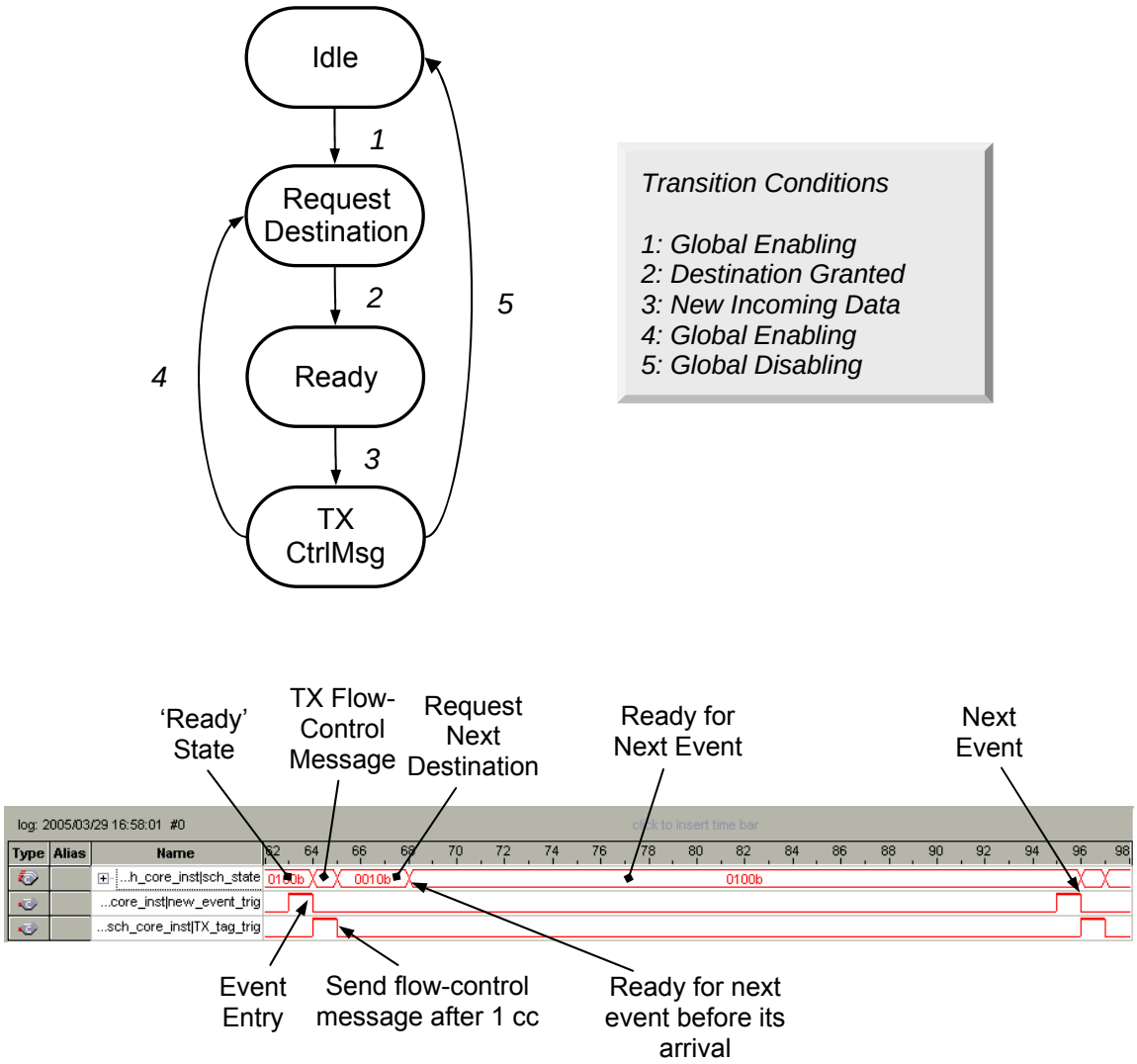


Figure III-9: Main state machine diagram. The scheduler accounts for a single clock cycle latency. The measurements in the figure are performed in a system run at an input data rate of about 2 MHz. The speed of the clock fed to the core of the scheduler and to the peripheral units relevant to the destination selection is 70 MHz. Under these circumstances there are about 30 clock cycles for the scheduler core to obtain the next destination without causing undesirable latency. The utilized algorithm and the queuing require 2 clock cycles. The design of the scheduler foresees running the scheduler core at a higher frequency in case of a more complex scheduling discipline.

4. Queuing of Destination Addresses

The transfer of event data from the feeding nodes to the computing nodes in the flow-controlled system is made under the supervision of the scheduling unit. The scheduler has to maintain a list of available computing nodes since it has to choose a receiving CN for every event that enters the system. Therefore, every computing node sends a unique feedback message to the scheduler in order to state its readiness to process new event data. The scheduler needs to handle the continuous flow of feedback messages. All incoming messages have to be promptly sorted, and the delivered CN addresses have to be appropriately stored in a block of queued destinations. The method has to allow a fast choice of the next destination since the latency of the scheduler is part of the total trigger latency.

The way this is realized is illustrated on Figure III-10. Feedback messages sent by the computing nodes are initially buffered in an input FIFO. Meanwhile the options of storing them in the block of queued destinations or of discarding them are evaluated. A feedback message is discarded if the associated computing node needs to be excluded during a system operation. The read of destination addresses out of the input buffer FIFO is controlled by a state machine. Its state diagram is shown on Figure III-11. Upon receiving a new destination address it is immediately read out and, if the user settings require so, the destination address is checked for validity. When the destination address is not qualified and should be disabled, it is discarded. The machine then continues to read out the next destination entry if one is pending. When a computing node is not excluded its address is stored in the block of queued destinations according to the column it originates from.

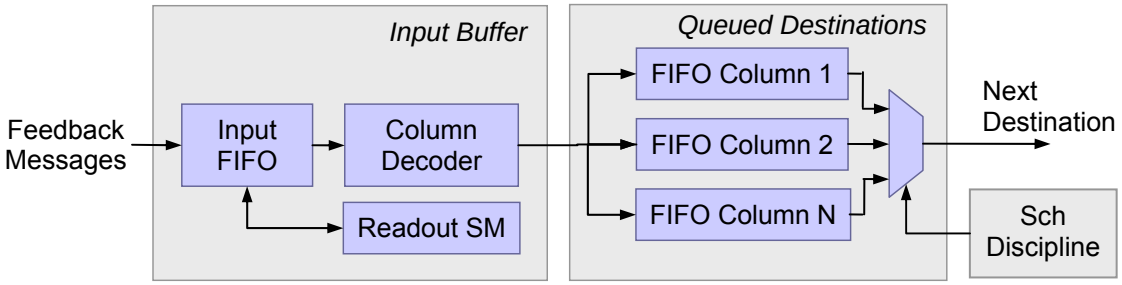


Figure III-10: Queuing of destination addresses. The architecture was designed to fit the chosen scheduling discipline and the system topology. Choosing another scheduling discipline or modifying the system topology may require another solution.

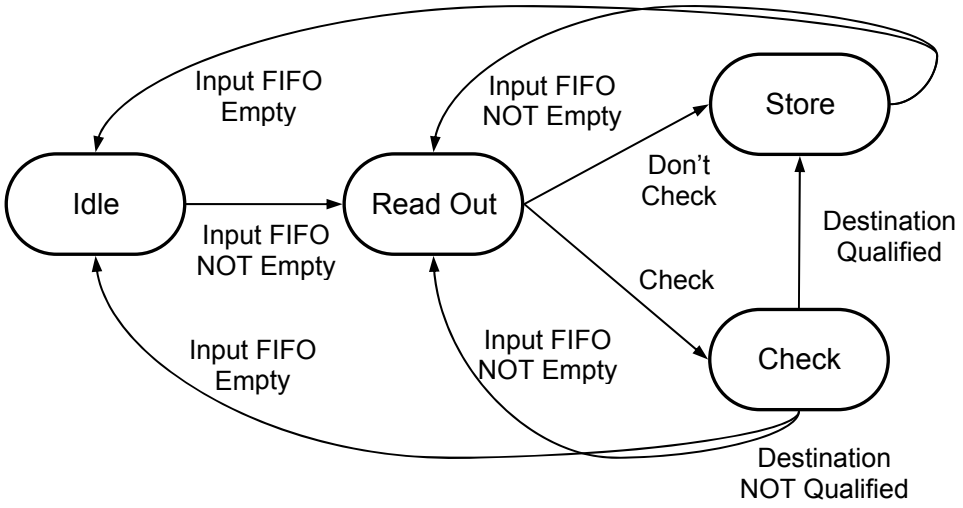


Figure III-11: State Diagram of the Input Buffer Readout State Machine

5. Flow Control Network Interface

The flow control network interface unit in the scheduler produces flow control messages, sends them to the flow control network, and ascertains the integrity of the control chain. A flow control message is produced by encoding the address of the TX buffer in the feeding node and the address of the computing node selected to process the event. The message is stored in a TX register until the core of the scheduler triggers the flow control network interface unit to send it (Figure III-12). The transmission of a flow control message is done in fragments under the control of the TX state machine.

After traversing all feeding nodes the message arrives back at the scheduler since the flow control network implements a ring topology (Figure III-2). The fragments of a received message are handled by the RX state machine, which builds a full message and stores it in an RX register. A copy of every flow control message sent is stored locally in the Check FIFO so that the scheduler can verify whether the message received is the one expected. Any violation detected here indicates system malfunction. Under normal operation, the Error flag returned by comparison must be permanently inactive. The Check FIFO must be empty after system halt. This signals that all flow control messages have arrived back and have been verified.

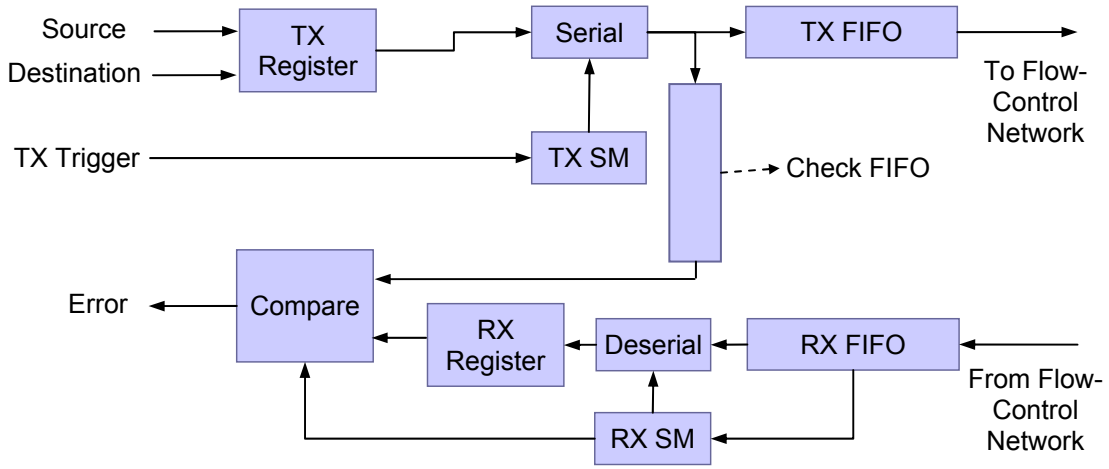


Figure III-12: Architecture of the Flow Control Network Interface in the Scheduler

6. Transfer of Feedback Messages

The scheduler can only operate if addresses of computing nodes are available in the buffer of queued destinations. Therefore, every computing node sends a unique feedback message to the scheduler in order to state its presence and its readiness to process new event data. Such a feedback message is considered a request from a computing node to the scheduler. A computing node sends its request first in the initialization phase and subsequently during system run upon every completion of the trigger algorithm. Thus, feedback messages arrive at the scheduler randomly at the rate of a megahertz, on average. The transfer of feedback messages does not require scheduling since the computing nodes do not send their messages simultaneously. The latency introduced in the path of a feedback message does not influence the maximum trigger level latency set over the system. Nevertheless, the transfer of feedback messages has to be fast since the latency introduced affects the efficiency of the system. Additionally, the transfer has to be reliable since any failure in delivering the feeding messages may lead to a system halt even though all computing nodes may be operational. The available main interconnect allows for a fast and reliable transfer of data. Consequently, it is used in conjunction with the global shared memory concept for the transfer of feedback messages (Figure III-13) the same way as in the case of the sub-event transfer between the feeding and the computing nodes, the only difference being that the transfer of feedback messages is software-initiated.

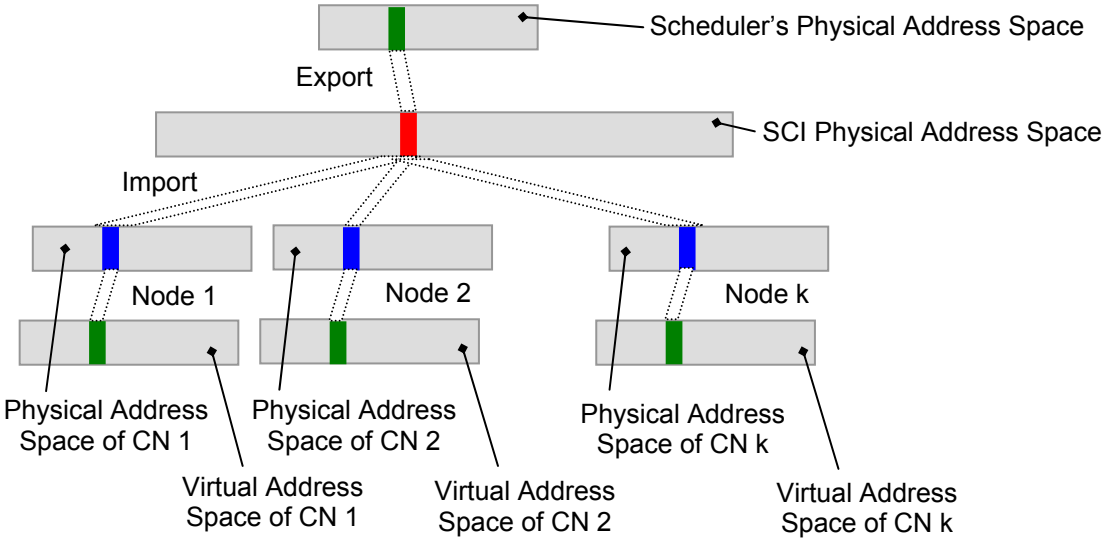


Figure III-13: Every computing node imports a dedicated memory region that is exported by the host of the scheduler to establish communication channels for the transfer of feedback messages. The shared memory paradigm allows this transfer to be performed by executing writes to remote memory, which is easily done in user software.

When a memory region is created and mapped (Figure III-13) its start address is returned to the user process. Since the available software infrastructure [33] does not allow the user to specify the address to the created memory segment, it was not possible to directly export a memory region available in the scheduling unit. A region in the memory of the PC which hosts the scheduling unit is exported instead. Hence, feedback messages sent by the computing nodes are written into the host memory. However, the subsequent write of a feedback message to the scheduler is not applicable as this would break the performance requirements and would increase the traffic on the expansion bus. Therefore, a module that analyses every transaction on the bus was embedded in the scheduler. It copies all feedback messages into the scheduler while the local network adapter transfers them to the host memory via the expansion bus (Figure III-14).

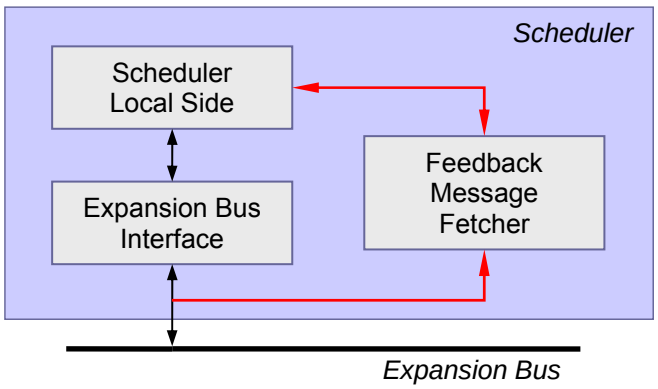


Figure III-14: The fetcher latches the data transferred via the expansion bus if the transaction is initiated against an address of interest that is specified in the initialization phase. The address of interest equals the physical address of the memory region exported by the host. Since a virtual address is returned to the user process when mapping the memory region, a function callable from user space [34] is used for translating the virtual address into the corresponding physical address.

Due to its small size a feedback message (Figure III-15) is always transferred in a single-cycle transaction on the local expansion bus (Figure III-16).

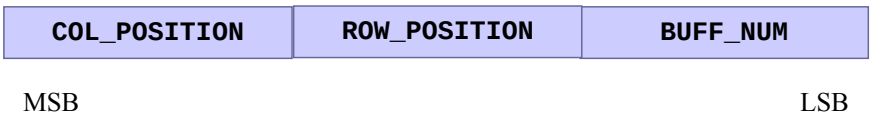


Figure III-15: Format of a feedback message. The `COL_POSITION` and the `ROW_POSITION` fields represent the number of the torus column and the row in which the node resides. Since there is more than one receiving buffer per computing node, the `BUFF_NUM` field identifies the receiving buffer within a node.

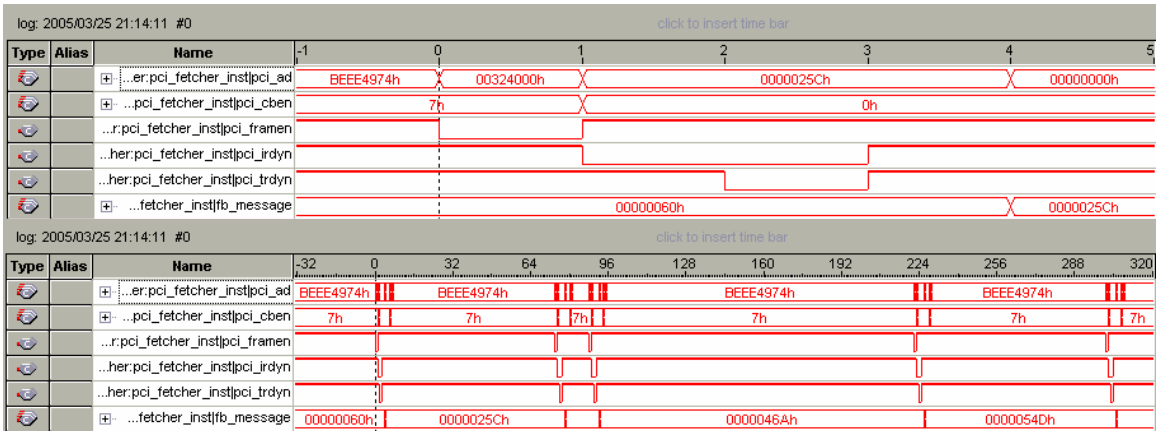


Figure III-16: Asynchronous reception of feedback messages at the scheduler. Five transactions can be distinguished in the lower trace. Each one represents the transfer of a feedback message originating from a different computing node.

7. Implementation

The data flow control system was developed to operate in a processing farm based on commercial off-the-shelf PCs. At the time of prototyping, the Peripheral Component Interconnect (PCI) was used as a local bus in commodity PCs. Therefore, the scheduling unit was implemented as a PCI expansion board to be easily integrated into the system.

The grounds for the technological choices for the implementation of the scheduler itself originate from the requirements that have to be met. A key characteristic of the scheduler is its scalability. By definition, the scheduling unit and the entire flow control system have to be easily adjustable to the actual size of the computing farm.

The system performance is one of requirements that influence the implementation considerations. The scheduler has to operate at a minimum of the PCI system clock rate, preferably at 66 MHz. Although such speed is achievable nowadays with CMOS technology, choosing the wrong techniques may lead to problems with timing.

Considering the requirements of speed and scalability, and the limited amount of identical devices in the system[1], the best solution is to implement the scheduler by means of configurable hardware components like FPGA[2]. Fundamentally, FPGAs provide great flexibility which satisfies the scalability needs. Contemporary FPGAs provide the features, density, and performance needed for the discussed application to be implemented. Choosing the FPGA technology for limited edition digital products is also essential to the respective price. In addition, reconfigurable components facilitate prototyping. Section III.C presents a PCI66/64 expansion card that was developed to accommodate the scheduler.

8. Setting up the Scheduler

The design of the scheduler is fully parameterized in order to meet the requirements for system scalability and to take advantage of reconfigurable hardware. Before running the system, the user must make sure that the scheduler is adjusted to the actual system size. This can be done by checking the values of certain design parameters. All parameters together with their values are located in a single file. In case some of the values do not correspond to the actual status of the system, the user should edit the corresponding entry in the file and produce a new programming file. A list of all design parameters, their description, and a range of valid values is available in section VII.B.

9. Control and Status of the Scheduler

The scheduling unit features a series of registers which were implemented to allow for user configuration and control as well as to monitor the scheduler and to ease trouble shooting. Configuration includes, for instance, the choice of a system trigger out of three different trigger sources: an internal mock-up data generator, an external single bit stimulator which provides more flexibility when adjusting the system frequency during the test phases, and a PCI trigger which was used during trouble shooting to manually issue single triggers via the expansion bus of the host of the scheduler. Since the scheduler is the global supervisor of the system, controlling it impacts the whole system. Control includes, for instance, the global enabling of the scheduler and of the system trigger as well as changing the scheduler's state machine mode of operation (section III.B.2). The status registers were mostly used to facilitate trouble shooting and

[1] There is only one scheduler in the system. Even enhanced system architectures would require a small number of such devices.
[2] Field Programmable Gate Array

to monitor the behaviour of the system. One of the registers allows monitoring the current state of the main state machine in the scheduler. This feature was used to promptly detect an unexpected system halt and to appropriately search the reason for the stoppage. A series of registers allow monitoring the instantaneous number of words stored in all buffers in the scheduler. Checking the number of words used in each FIFO in the block of queued destinations (Figure III-10), for instance, can give an indication of each torus column's load. Implanted memory was used to monitor the overflow as well as the peak percentage of usage of certain buffers since the last system start. The result returned by the basic error checking realized in the flow control network interface (section III.B.5) can be read out as well via a status register. The flow control network interface in the scheduler ascertains the integrity of the flow control network by checking whether the message arrived back at the scheduler is the one expected.

Due to the implementation (section III.B.7) the control and status registers are accessible via PCI. A detailed description of all CSR can be found in section VII.C.

C. The CIA-RORC Board

1. Motivation

The CIA-RORC board[1] is a multipurpose universal 64-bit PCI expansion card based on an FPGA. The board is capable of working in a stand-alone mode, too. At the time of development, there were already similar commercial products available. However, notwithstanding their versatility, none of these products provided all the features considered significant for the purposes of the research activities at the Chair. Furthermore, all the alternatives examined were lacking particular functionalities. These are, for instance, the operability of the board in both 3.3V and 5V PCI signaling environment and the accessibility of FPGA device pins, supporting particular I/O standards, through expansion connectors.

These considerations motivated the development of a card which incorporates the features required by multiple active projects and thus overcomes the constraints imposed by the alternative cards. The major applications which guided the design of the CIA-RORC card are:

- a control unit (Read-out Control Unit RCU) in the ALICE Time Projection Chamber (TPC) read-out chain
- a receiver card (Read-Out Receiver Card RORC) interfacing the front-end processors of the ALICE High-Level Trigger (HLT)
- an independent hardware controller (Cluster Interface Agent CIA) for remote management of commodity PCs
- a reconfigurable PCI form factor co-processor

[1] The name CIA-RORC is built of the abbreviations of two of the applications of the board: CIA stands for Cluster Interface Agent; RORC stands for Read-Out Receiver Card.

The board was developed to provide the hardware means needed within this work, too. Consequently, it is the platform used to implement the scheduling unit and the logic in the feeding nodes (discussed further in section III.E) as well as a number of auxiliary devices used to facilitate trouble shooting and benchmarking.

In the following, the major applications of the card as well as its main features with respect to these applications are outlined.

a. Read-out Control Unit

The ALICE Time Projection Chamber (TPC) [35], the largest data source of the future-planned heavy-ion experiment ALICE at LHC, contains about 560 000 channels, whose data are to be read out through more than 4000 front-end cards. The latter perform detector data processing and buffer the data before their transport to the trigger and the DAQ system.

The Read-out Control Unit provides the interface between the front-end electronics cards and the RORC cards. The latter implement the input stage of the trigger and the DAQ system (Figure III-17).

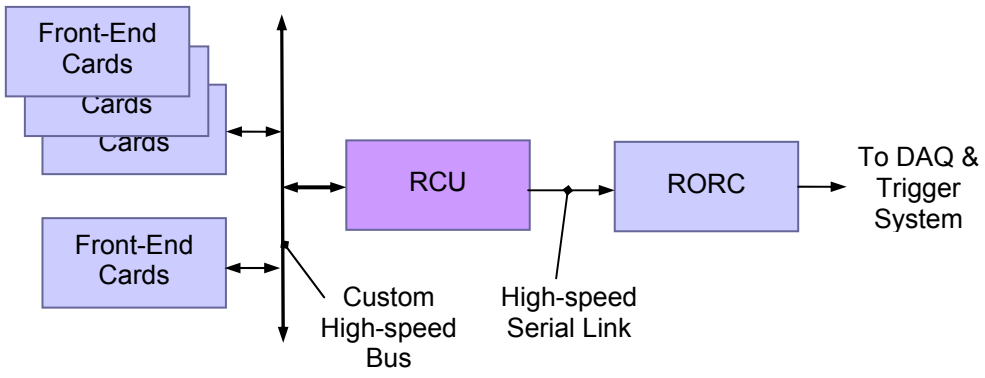


Figure III-17: A sketch of the ALICE TPC read-out chain. One RCU reads data out of 32 front-end cards [35], performs sub-event building, adds header information, and transmits the data to the inputs of the DAQ and the trigger system.

Basic RCU functionality tests have been done using a commercial FPGA-based PCI expansion card. However, the CIA-RORC card was substituted for the commercial one since the CIA-RORC was designed to provide features which allow a more detailed investigation of the various hardware options concerning the ALICE TPC RCU unit.

b. Read-Out Receiver Card

The main tracking detector of the ALICE experiment alone, the ALICE TPC, will produce a data volume of about 75 Mbytes per event in central Pb-Pb collisions

with the rate of 200 Hz [35]. This results in a rate of 15 Gbyte/s, which exceeds the mass-storage data rate available.

The High-Level Trigger (HLT) designed is a large-scale processing farm which has to perform real-time data processing in order to reduce the input data rate of up to 25 Gbyte/s to the mass storage rate of 1.2 Gbyte/s [7]. The input stage of the HLT is realized by means of Read-Out Receiver Cards. They implement the interface between the detector's front-end electronics and the HLT computing nodes. The RORC cards receive the digitized data coming from the on-detector front-end electronics, and transfer them by means of DMA to the main memory of the front-end processor nodes, thus feeding the HLT (Figure III-18).

Prior to data transfer, the RORC card performs hardware data pre-processing by means of an FPGA co-processor, thus reducing the total trigger algorithm processing time and the total number of CPU nodes. The FPGA co-processor implements cluster finding in case of low-multiplicity events. Local track finding based on the Hough Transformation of the raw data is realized in case of high-multiplicity events.

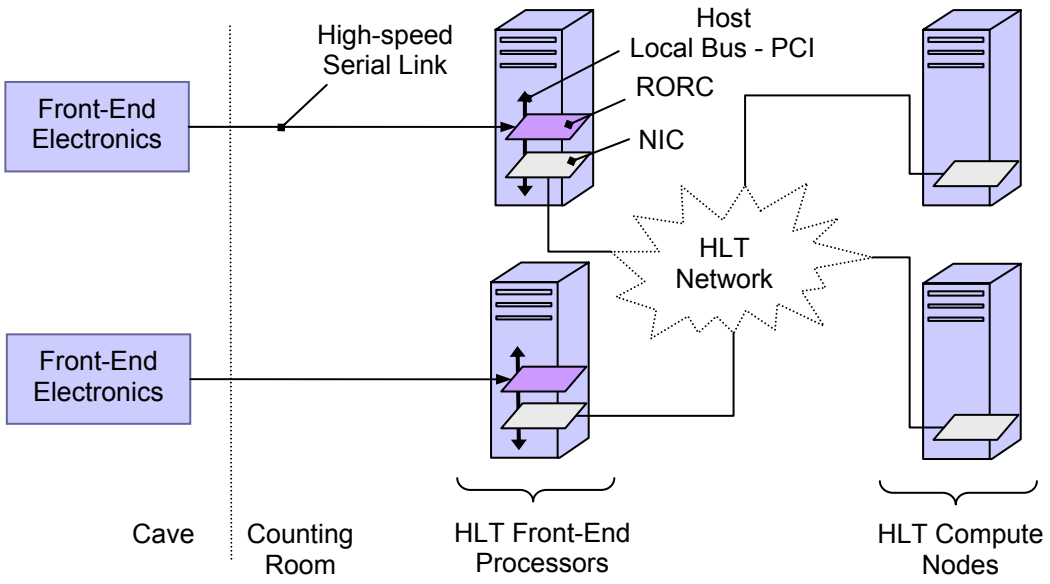


Figure III-18: Each HLT front-end processor node is complemented by a PCI expansion card based on reconfigurable hardware units. The card receives data coming from all major ALICE detector front-end electronics. Subsequently, it pushes the data through the local expansion bus to the host main memory, where they are accessible to the host CPU. Before transferring the data to the main memory, however, the RORC card performs data pre-processing in hardware to assist the CPU. This way it accelerates the trigger algorithm.

c. The Cluster Interface Agent

Contemporary computing clusters are large systems comprising hundreds of commodity PCs, which are, in general, not designed to run in a PC farm. Consequently,

configuring, monitoring, and controlling – tasks which are crucial to the operation of these systems – can only be performed reasonably if special measures are taken.

The software tools offered for control and monitoring of computing nodes lack portability and are not autonomously operational but rather operating system-dependently. Hardware tools available are typically system-specific, and, if not lacking a broad spectrum of supported features, they are expensive.

The CIA agent project aims to provide a low-cost solution that covers various aspects of remote cluster management. The concept is based on a low-cost autonomous hardware controller [36, 37] which supports a number of features allowing remote control and monitoring of commodity PCs. The latter are prone to failures. Monitoring of physical parameters like the temperature of hard drives or power supplies, the power supply voltage, or the noise generated by cooling fans or hard drives, could help to detect pending failures in advance and to avoid painful consequences like loss of data or system halt.

The modification of BIOS settings is a general administrative task; however, most of the systems do not support remote BIOS access. In order to give remote access to the BIOS, the CIA agent exports the host video, mouse and keyboard to a remote location.

Remote access to a typical system which has encountered a crash of the operating system is not possible. In such cases, a reboot is needed. By means of the CIA agent, hardware-initiated reset or power cycle of the affected node could be realized from remote. The installation of an operating system or a new boot of the operating system can be initiated remotely.

The CIA-RORC card presented was designed to support a number of features needed to implement the first prototype of the CIA agent. Subsequently, a commercial product based on this development has been released [38].

d. FPGA Co-processor

A considerable part of the processing time of CPUs running track reconstruction algorithms is combinatorial[1]. Methods for the reduction of that time have already been investigated [39]. Calculations of a certain type could be performed in parallel, and faster in hardware than in software. Considering the trigger system presented in chapter II, where every microsecond of processing time requires a CPU, reducing the total processing time by means of a FPGA co-processor may lead to considerable cost saving.

The CIA-RORC board provides a high-density FPGA and can easily complement a PC. These features make the card a suitable platform for building a custom reconfigurable co-processor for the investigation of such hardware acceleration.

[1] About 70% of the total processing time of the LHCb Level-1 trigger reconstruction algorithm is spent in search of two-dimensional tracks. This is mostly due to the high number of combinations and corresponding calculations needed [39].

2. Features

The features provided by the CIA-RORC board are summarized in the following:

FPGA: The CIA-RORC card was intended to be used in multiple projects. All of the applications were in the research and development phase, which required a versatile prototyping platform. For these reasons, the board is based on an FPGA device.

A device family [40] suited for system-on-a-chip solutions was chosen, since most of the applications require the integration of diverse types of memory units, serial transceivers circuitry for high-speed board-to-board data transfer, embedded product-term logic for the implementation of control logic functions such as address decoders and state machines, clock management, multiple I/O standards support, general purpose bus compliance, etc.

The board supports high-density FPGAs. Since designs on a smaller scale would show poor performance due to larger internal signal delays, the Printed Circuit Board (PCB) is designed to support a smaller device as well, which also contributes to a better price-performance ratio.

PCI and stand-alone operation: Most of the applications of the CIA-RORC board presented are PCI applications. The FPGA co-processor and the RORC card complement PCs and provide fast transfer of data through the host PCI bus to the host main memory, where these data become accessible to the host CPU.

The RCU unit is part of the on-detector electronics; its final version does not require PCI functionality. However, the interface to a PC was used during the test phases performed with the second prototype.

The CIA agent interfaces with the expansion bus (Figure III-19) in order to detect all present PCI devices and eventually access them to realize configuration, control and debugging. Furthermore, the remote user interface requires PCI functionality. The card exports the host display to a centralized remote control unit via an independent network. Video control is performed in two ways: firstly, by accessing the deployed video controller via PCI to capture video information; secondly, by replacing the default video controller with a CIA card which is then registered as a PCI video adapter and can appropriately handle the video information. Remote mouse and keyboard commands are appropriately transformed into PCI transactions against the host mouse and keyboard controller and thus passed to the host.

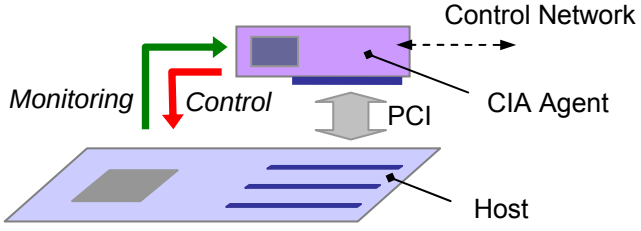


Figure III-19: The independent CIA agent can monitor the status of the installed PCI devices by accessing them through the host PCI bus. On the other hand, the card can also actively control the host by emulating certain I/O devices and passing control commands through the PCI bus to the host.

A commercial programmable logic PCI implementation [41] was utilized to interface the expansion bus. The board can be used in 64-bit or 32-bit PCI slots with a 3.3V or 5V signaling environment. It supports 66 MHz as well as 33 MHz PCI.

Beside PCI functionality, the CIA agent must be operable completely independently of the host status. Executing a host power cycle, for instance, requires that the CIA card is fed by a redundant power supply. Therefore, the CIA-RORC card is equipped with on-board power regulators. A power management unit switches between redundant and default PCI power supply. The on-board power management allows using the CIA-RORC card in desktop applications as well, even when a PC is not needed.

Ethernet*:* The CIA cluster management system provides full access to a cluster node remotely. Moreover, control and monitoring have to be realizable independent of the node condition. In addition, given the high data rates and the low latencies available, which characterize trigger processing farms, monitoring and control have to be implemented in a way that is as little intrusive as possible. For these reasons, all CIA agents are interconnected by a second redundant network, which is dedicated to cluster management (Figure III-20). Control and monitoring can then be performed independently without relying the host system resources.

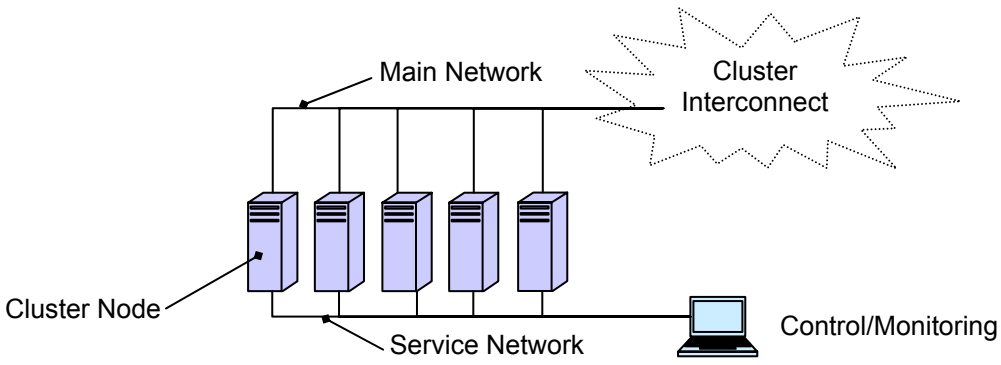


Figure III-20: Automated cluster control and monitoring are realized remotely via an independent service network which interconnects the dedicated autonomous CIA agents installed in every node.

The CIA-RORC Ethernet circuitry has also been used to perform various tests concerning the utilization of Ethernet to interconnect the RCU unit with the Detector Control System (DCS).

Memory: Although contemporary FPGA devices provide embedded memory blocks, which were sufficient for some of the applications, other CIA-RORC applications require additional memory to store larger data blocks temporarily or permanently. Therefore, the board is equipped with a sufficient amount of external memory of various kinds.

The architecture conforms with the requirements of the CIA agent (Figure III-21), where all operations are controlled by means of a general-purpose reconfigurable embedded processor. The processor's main memory is located in an external Random Access Memory (RAM), while the root file system and the kernel of the deployed embedded network-capable operating system are stored in a non-volatile Flash device.

The HLT RORC application of the CIA-RORC card requires temporary local buffering of detector data or storage of large look-up tables that are used to perform hardware data pre-processing.

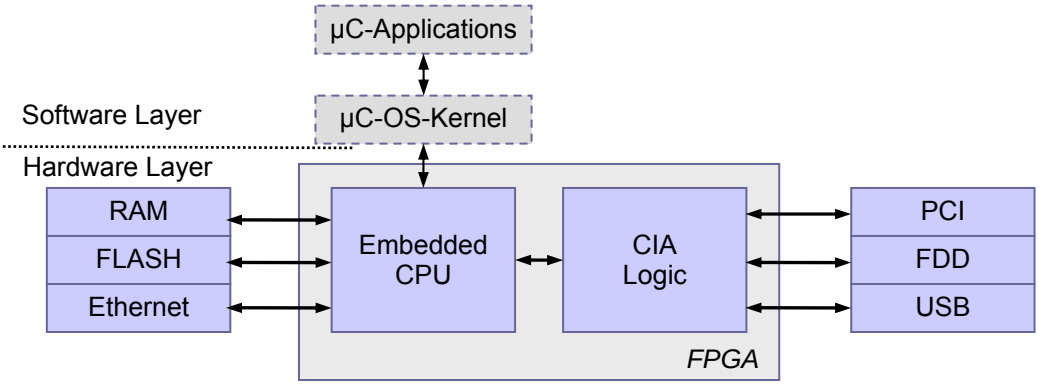


Figure III-21: The CIA agent requires a cost-effective, reliable, flexible, reduced-power solution that is easy to maintain, in order to implement special functionalities. Consequently, an embedded system running an open modular embedded operating system was chosen. The CIA-RORC card was designed to provide a hardware platform (Figure III-22, Figure III-23) that allows the embedded system to be ported to it.

Common Mezzanine Card (CMC): Most of the CIA-RORC applications involve data transport. Therefore, the CIA-RORC card has to provide appropriate I/O support. However, different applications put different requirements to the transceiver channels. Some applications require high-speed data transfer over a long distance. This case is shown in Figure III-17, where data have to be transferred from the on-detector RCU units to the RORC cards over approximately 300 meters at the rate of about 2 Gbps. Other applications have modest demands. Applications of this kind are discussed further in section III.D. The CMC family [42] was deployed to stay flexible. It provides modular I/O as well as modular local function expansion by means of small factor pluggable cards.

Optocouplers: A computing node which has encountered a crash of the operating system is no longer accessible remotely. In that case the node requires a hardware-initiated reset or a power cycle. Optocouplers available on the CIA-RORC card are used to commutate the power and the reset switches of a computing node.

ADC and DAC: Commodity nodes are prone to failures due to unreliable hardware components. In most cases, however, pending failures can be detected early enough. Analog quantities, such as the temperature of hardware units, the power supply voltage, and the noise generated by electro-mechanical devices, have to be continuously monitored in order to avoid unexpected hardware failures. The first prototype of the CIA agent provides generic resources that handle the signals produced by appropriate sensors.

FDD connection and USB Interface: Floppy Disk Drive (FDD) emulation provided by the CIA card makes the host node bootable in case the host operating system has been damaged. In that case, a minimal network-capable operating system can be booted through the CIA agent. Afterwards, the desired operating system can be installed via the network. Similar functionality can be achieved through the emulation of a USB mass storage device.

LVDS transceivers: The Low Voltage Differential Signaling (LVDS[1]) has become widely used in point-to-point interface applications, which is due mostly to its low power consumption, low noise and high transfer rates. General purpose LVDS drivers and receivers are provided for the implementation of generic board-to-board communication.

System EEPROM and Real-Time Clock: The CIA-RORC card has been designed to provide a hardware platform for building an autonomous system on a board. A low-cost low-power serial-interface Electrically-Erasable Programmable Read-Only Memory (EEPROM) device was therefore deployed for nonvolatile storage of system information like configuration, the version number and other card specific data. A real-time clock/calendar device can be used by the CIA agent, for instance, to register the exact time when an event occurs.

Battery Power Supply and Management: Battery power and a charger controller are provided for real-time clock power back-up as well as for applications with very low power consumption.

Configuration Controller: FPGA configuration data are stored in a low-cost upgradeable Flash memory device. A configuration controller located in an auxiliary CPLD device handles the configuring of the FPGA. Several configuration files can be stored in the Flash device. Loading of any of them into the FPGA can be done dynamically. A base configuration located at offset zero of the configuration Flash allows accessing all Flash devices via PCI. Thus, automated FPGA configuration management is possible remotely.

[1] Low Voltage Differential Signaling; ANSI/TIA/EIA-644-A Standard

Devices implementing various serial interfaces like **RS-232**, **IrDA**, and **CAN**, provide general-purpose communication support.

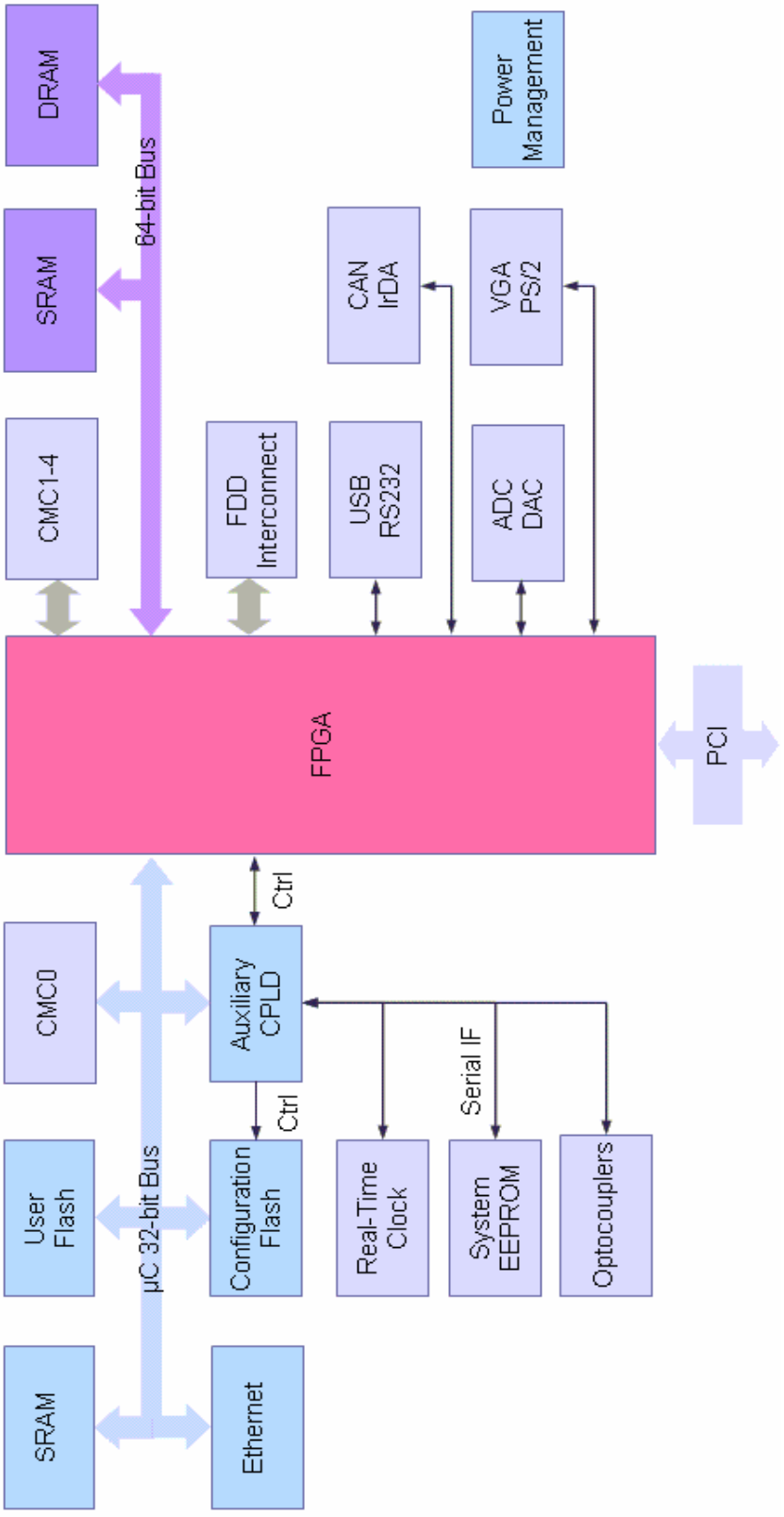


Figure III-22: The CIA-RORC Board: a Simplified Block Diagram

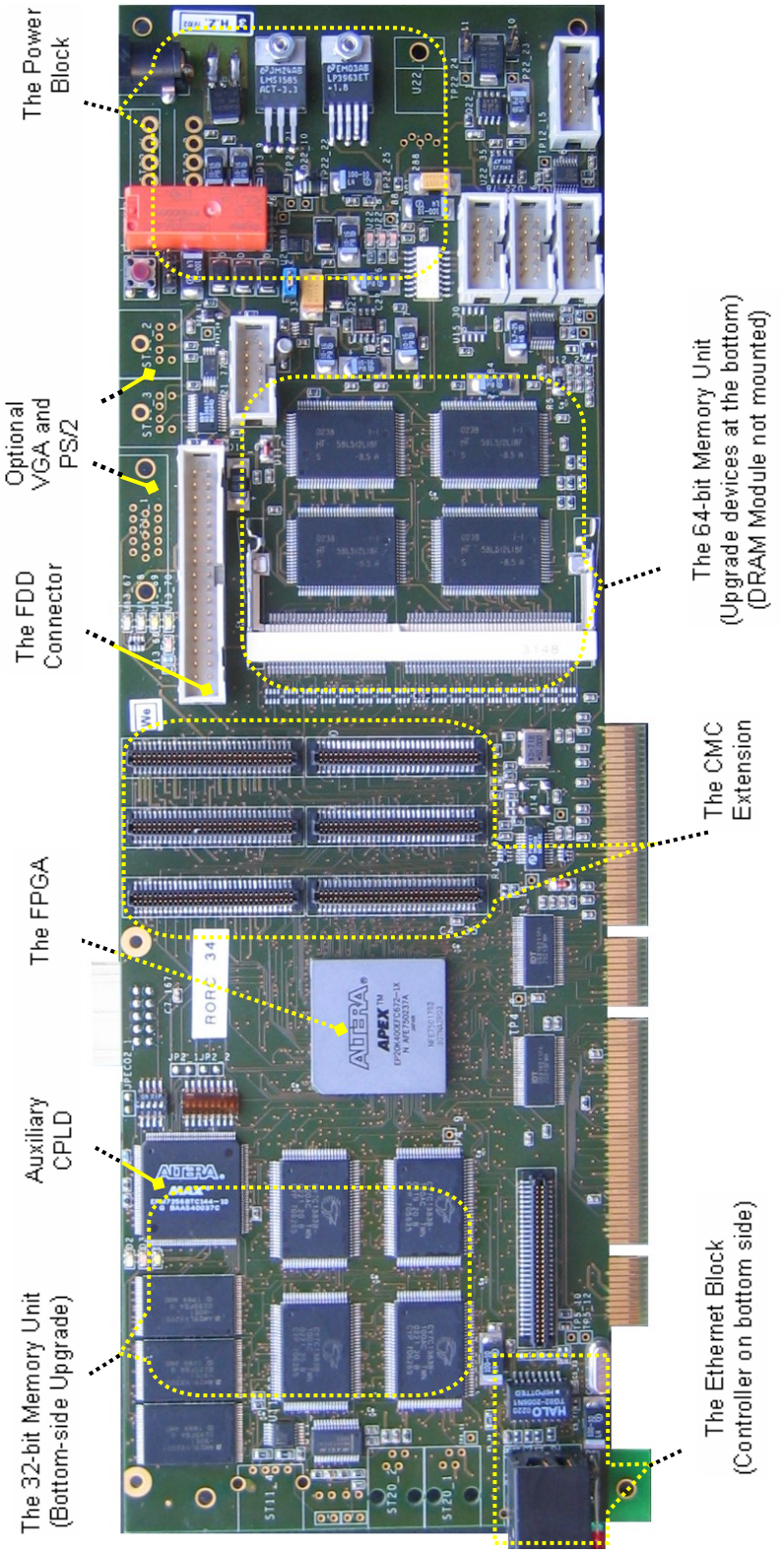


Figure III-23: The CIA-RORC Board. The data provided in [43 - 56] are used for the development of the board.

D. The Flow Control System Network

Since the feeding nodes are distributed throughout the processing farm, a communication channel between the scheduler and all feeding nodes was needed. It transfers flow control trigger messages according to the data flow control architectural concept presented in section III.A.

The time it takes to build a full event in the computing node is part of the total trigger latency available. Hence, the sub-event data packets split among the feeding nodes have to be delivered to the computing node within a very small period of time. This, however, is only possible if congestions at the receiver do not occur, which requires that the sub-event packets are sent successively with a certain time interval between two packets. Therefore, a fast flow control network is needed, where the latency of the messages is completely predictable. The existing network interconnect is not suitable for the transfer of this timing information since it is not possible to grant the highest priority of flow control messages that is needed to guarantee the delivery in time[1]. Moreover, the transfer of messages through the main interconnect would reduce the bandwidth available on the expansion bus in each one of the feeding nodes since all messages to and from the network adapter have to traverse the shared I/O bus of a sender.

Since data corruption at this stage could certainly lead to odd system behaviour (for instance, only partial event delivery to the chosen computing node with some sub-events sent to other nodes), reliability is a crucial demand on the flow control system network. On the other hand, a low-cost solution was needed.

The flow control network topology chosen is a ring since a node always sends its flow control message to the next one downstream (Figure III-24).

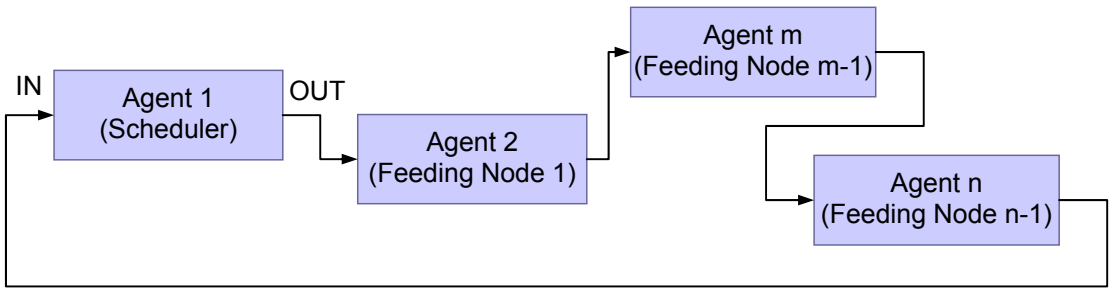


Figure III-24: The flow control system network implements a ring, where data flow only in one direction. The terms in brackets identify the agents in the actual set-up. The scheduler feeds the network with flow control messages. Upon reception of such a message the flow control interface unit performs a basic error check, decodes the message, triggers the DMA engine to initiate the data transfer, and forwards the message to the next agent. All messages arrive back at the scheduler after a certain amount of time, where a data integrity check is performed. Depending on the mode of operation, a feeding node may or may not modify the messages or transmit self-generated messages to the flow control network.

[1] The SCI implementation deployed does not support priority-based real-time capabilities.

Data are transferred over the physical media of the flow control network using a differential signaling technology – the Low Voltage Differential Signaling technology [57]. It is designed to support high-speed data transfer and is therefore characterized by a number of advantages, such as:

- common mode noise rejection (an equal induction in both conductors that is rejected by the receiver);
- lower spectral content (EMI) than other technologies (canceling of opposite magnetic fields; small switching spikes in current-mode output drivers);
- low power consumption and low-voltage power supply compatibility;
- robust transmission signals;
- cost-effective solutions;
- low signal swing[1] due to improved signal-to-noise ratio, which results in a short rise time and consequently a high data rate;

Basic tests of serial data transfer between two boards performed with the generic LVDS transceivers available on the CIA-RORC board were unsatisfactory. The high-speed serial transceivers available in the FPGA on the CIA-RORC board were not found suitable for box-to-box communication. Therefore, the transmission of data over the flow control system network is performed by means of a serializer/deserializer card, which is plugged into the CIA-RORC board. The transceiver card is based on a chipset dedicated to fast point-to-point applications and contains a limited number of external components. The interface chipset consists of a separate transmitter and a receiver. The transmitter converts the incoming parallel data into a multiplexed serialized data stream (Figure III-25), which allows the substitution of a wide medium for an economical medium such as a twisted pair cable.

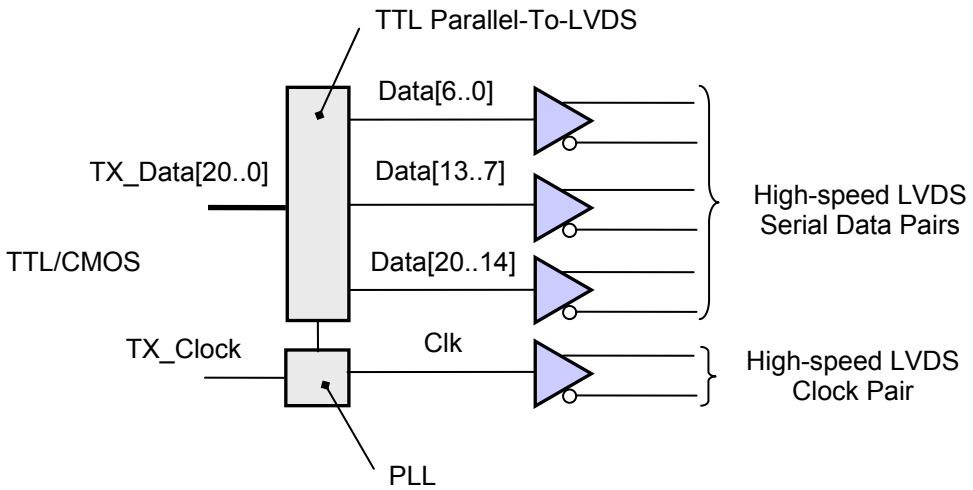


Figure III-25: The flow control network interface card deploys a 21-bit chipset with the following possible user configuration: two payload bytes, three control bits, and two parity bits. All bits are transferred over three differential data pairs, where each differential pair is used for transferring seven of the twenty-one parallel data bits. One extra differential pair is used for transferring the clock. This high-speed clock is managed by Phase-Locked Loops (PLL) in the transceivers.

[1] LVDS has a signal swing of 300 mV and is centered at + 1.20 V.

The use of fewer interconnect nodes leads to cost savings and improves reliability. Furthermore, it reduces the probability of transmission errors due to cable skew. The receiver converts the serial data stream back into parallel data. In addition to the serialization/de-serialization of the data, the transceivers also perform a conversion of the parallel TTL/CMOS data into LVDS serial data, and vice versa.

The interface card is equipped with a low-cost FPGA for intermediate data processing (Figure III-26). Two different volumes of FPGA are supported for a better price-performance ratio. The data provided in [58 - 67] are used for the development of the board.

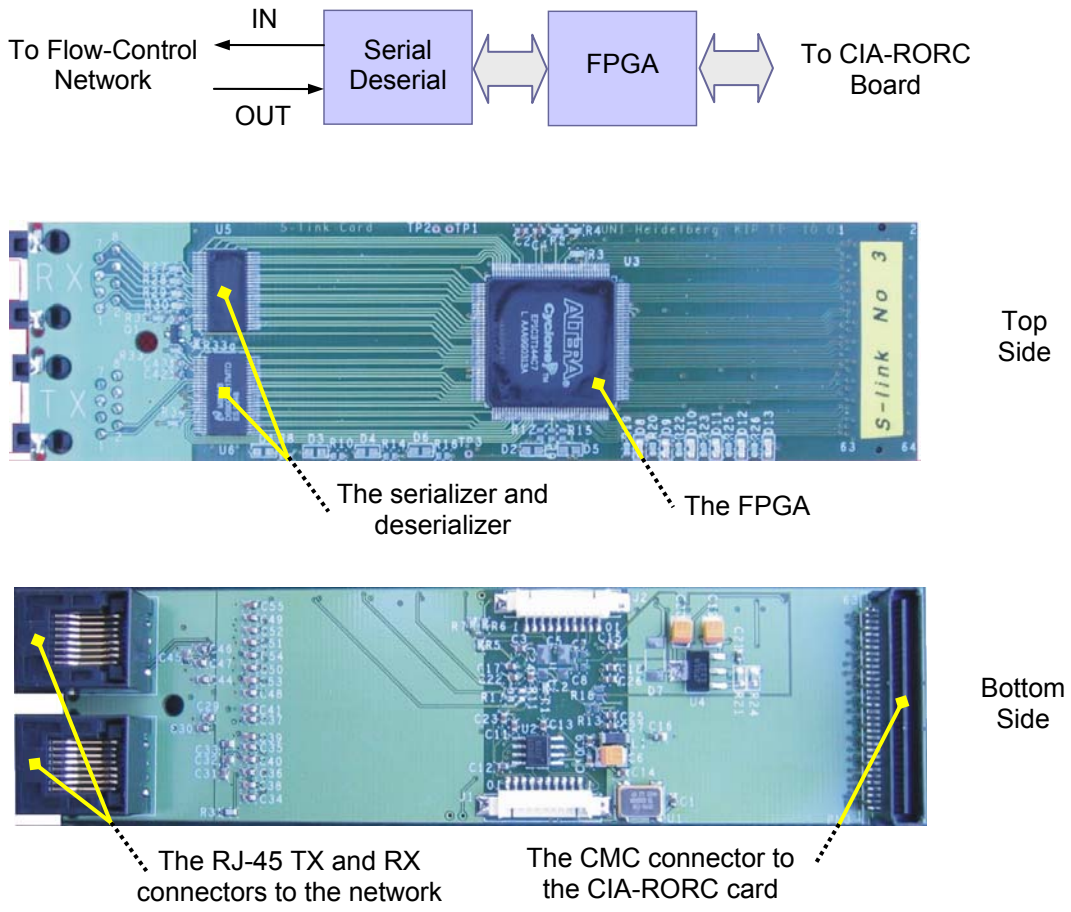


Figure III-26: The flow control network interface card was designed as a small-factor CMC card which can be plugged into the CIA-RORC board. The maximum input transmission clock supported by the chosen chipset is 85 MHz, which corresponds to a data rate of 1.785 Gbps. The maximum speed achievable depends also on other factors like utilized connectors, PCB material, and type and length of the interconnect media between the transmitter and the receiver. The developed card deploys low-cost CAT5E RJ-45 connectors. The feeding nodes are interconnected through standard CAT6 STP (Shielded Twisted Pair) cables which are 2 meters long. The link is operated at a data rate of 630 Mbps (480 Mbps data payload).

An advanced serial protocol has been developed and presented in [68]. However, a simple interface protocol was used for the purpose of this work (Figure III-27).

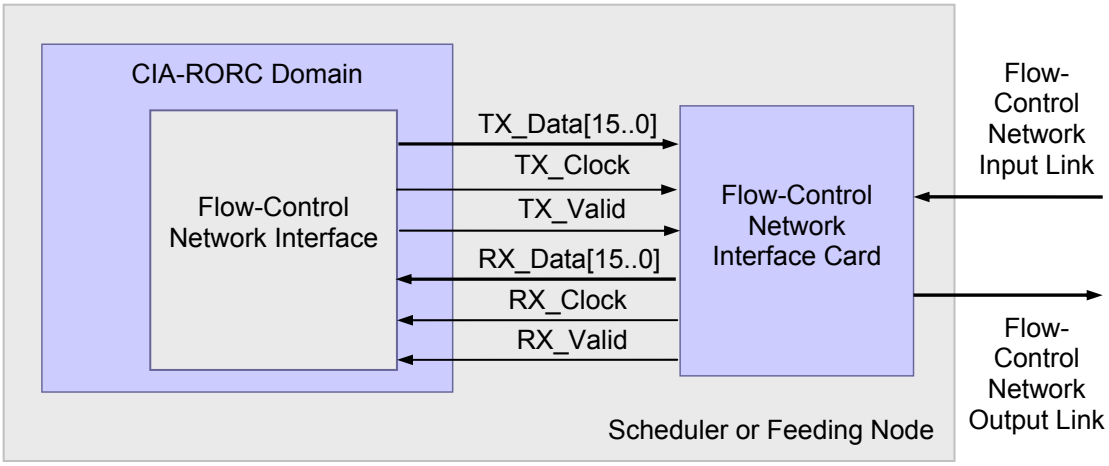


Figure III-27: The connection between the host CIA-RORC board and the associated interface card is full-duplex. Each direction comprises 16 data signals (TX_Data/RX_Data), a flag bit (TX_Valid/RX_Valid) used to mark each 16-bit word as valid or invalid, and a strobe signal (TX_Clock/RX_Clock) for latching the incoming data. Sixteen bits of data are transferred on every rising edge of the strobe signal. The rate of the strobe signal is adjustable[1]. It is set to 30 MHz in the set-up. The clock frequency of the high-speed LVDS link (Figure III-25) is 7 times higher.

The format of a flow control message is shown in Figure III-28. Flow control messages are transferred in fragments of 16 bits (Figure III-29). The number of fragments per message is adjustable (section VII.B).

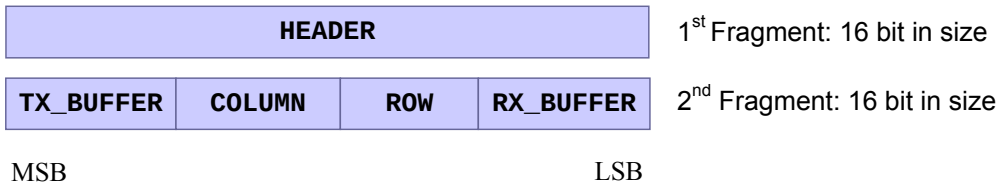


Figure III-28: Format of a flow control message transferred through the flow control network. The first fragment is a header that identifies the type of message. The header string is designed to be modifiable (section VII.B). In this work, only messages that initiate data transfer were used. The destination address is encoded by the fields COLUMN, ROW and RX_BUFFER. The fields COLUMN and ROW give the *x* and *y* coordinates of the destination node in the 2-D torus correspondingly. The field RX_BUFFER is the receiving buffer offset (see also section III.F). The field TX_BUFFER specifies the address of the sub-event in the feeding node since the latter can buffer several sub-events.

[1] statically, i.e. the adjustment requires new FPGA configuration files

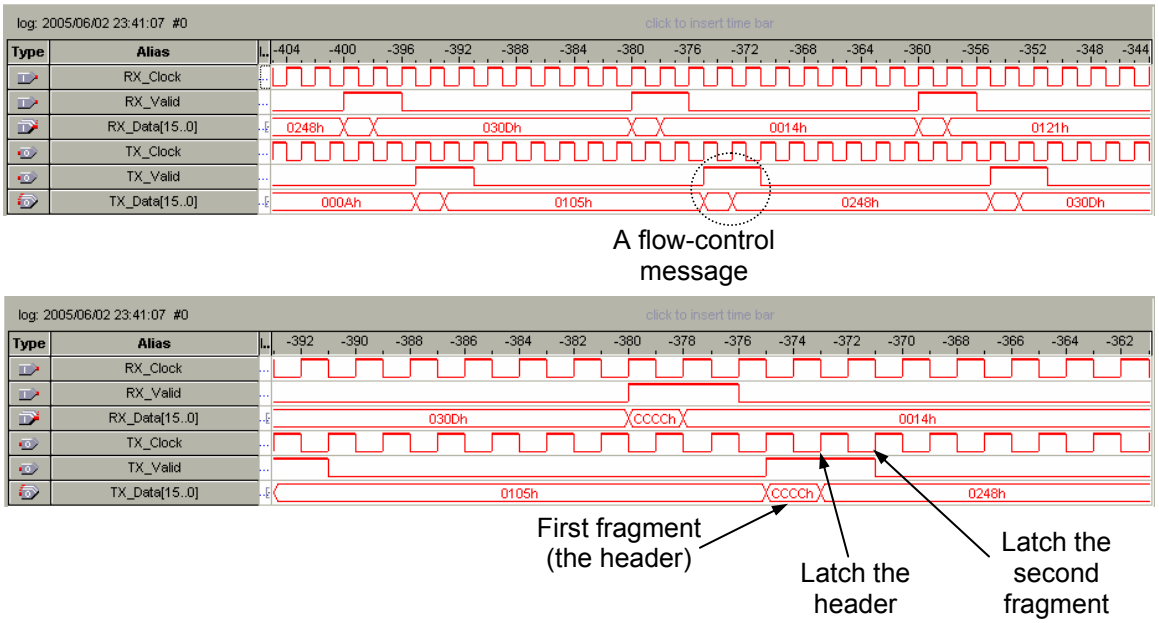


Figure III-29: Transfer of a Flow Control Message

Transmitting a statistically large sub-event by some of the feeding nodes increases the time needed for the event to be built in the computing node. Such an event may also delay the processing of the one that follows if the sender waits for the completion of the long sub-event transfer before forwarding the control message to the sender next to it. During such delay, the input buffers of all the feeding nodes are filled with new events. Therefore, the flow control system network requires mechanism in order to overcome problems arising from large variations in sub-event sizes. Simulations show (section V.F) that skipping a node that transmits a previous large sub-event and consequently sending that partially processed control message to the control chain for a second time leads to a shorter event building latency and makes the system more robust in large events processing.

The mechanism for bypassing busy senders can be realized by means of an additional bit field in the flow control message which is appropriately interpreted and modified by all nodes in the flow control chain. The bit field value is initialized by the scheduler and can be modified by a feeding node only if the latter has initiated the corresponding sub-event transfer. A feeding node processing an existing long sub-event cannot handle the current control message and therefore simply forwards it without modifications. Thus, the current event building continues without delay. As the control message arrives back at the scheduler, the latter analyses the bit field and detects that the event transfer has not been executed completely. It therefore sends the same control message back to the first feeding node in the control chain. The bit field value is also used by the feeding nodes to detect if the received control message either triggers a new transfer or it relates to a previous postponed sub-event transfer. This mode of operation imposes the requirement for a maximum number of loops allowed for a particular control message through the flow control network.

E. The Feeding Nodes

1. Structure

The logic in the feeding nodes consists of a flow control network interface unit that handles the trigger messages, a DMA engine that performs the data transfer, and a descriptor buffer for the storage of descriptors. The latter characterize the transfers (Figure III-30).

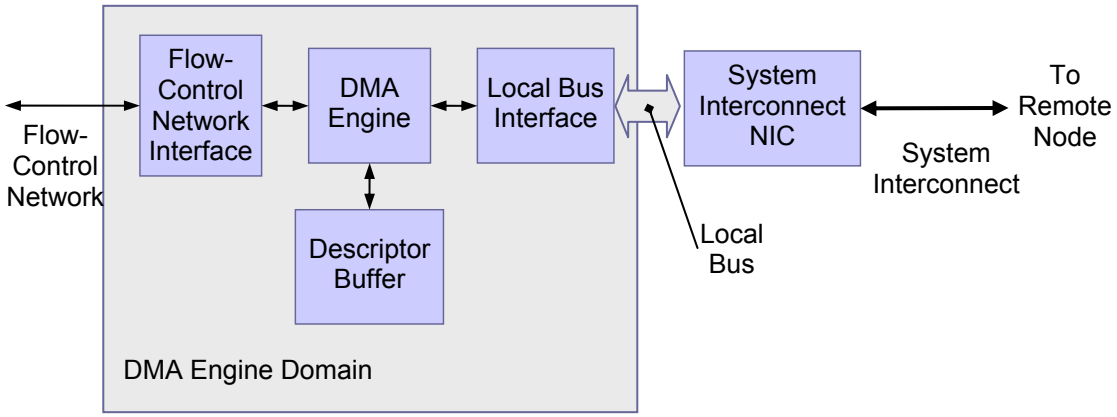


Figure III-30: Structure and Interface of the DMA Domain in the Feeding Node

Messages that come in through the flow control network are initially stored in a queue. A new flow control message is read out of the queue against the background of an ongoing data transfer. Thereafter it is processed by the flow control network interface unit, which returns an address to the descriptor buffer. This way a valid data transfer descriptor is already provided by the descriptor buffer at the end of the ongoing transfer. Thus two subsequent data transfers can be realized back to back without any latency induced at this stage. The flow control network interface unit triggers the DMA engine as soon as the last data transfer is completed.

The DMA engine is a finite state machine that interacts directly with the local bus interface to initiate master write transactions against the local network adapter. Target read/write transactions for accessing the control and status registers are also supported. Master read is not supported since it is not required by the application.

Upon incoming DMA triggers, the local bus interface requests bus mastership. When the bus is granted, the data are copied directly into the input buffers of the system interconnect interface controller. The latter then initiates a remote data transfer to a memory region which is being provided by the remote destination node. The data transfer is realized completely in hardware without any software or CPU overhead. A sub-event is transferred in a single burst transaction on the local expansion bus and subsequently in a single packet over the network.

The descriptor buffer is a memory-mapped region and can therefore be read and written through the local bus. The buffer is parameterized to be adjustable to the actual size of the processing farm.

a) Descriptor Buffer

Data transfers as explained in section II.C are only possible if the DMA engine is provided with valid local host physical addresses that correspond to remote destinations. In the initialization phase, the physical addresses corresponding to all remote computing nodes in the system are written to the so-called descriptor buffer. There, each computing node possesses its own slots. The partitioning of the descriptor buffer follows a defined format as illustrated in Figure III-31. Each entry contains a 32-bit local host physical address that corresponds to a particular remote receiving buffer.

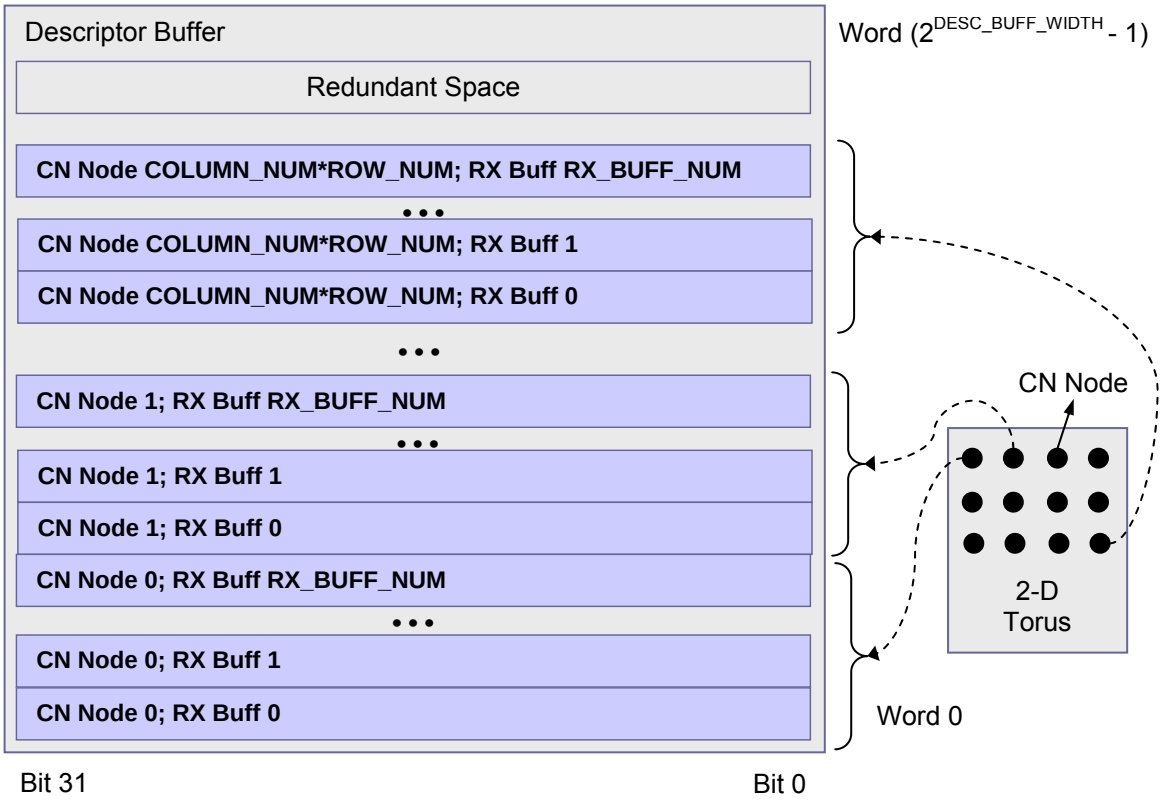


Figure III-31: The Descriptor Buffer Format

The transfer of data to a specific remote node requires addressing the corresponding descriptor buffer entry. The address to the descriptor buffer is selected according to the flow control message that triggers the data transfer. So, a flow control

message is an encoded descriptor buffer address. On the other hand, the flow control message is also an encoded feedback message (section III.B.6). Hence, there must be a complete correspondence between the way feedback messages are encoded and the order by which the descriptor buffer is initialized.

Every data transfer is characterized by a descriptor that, in addition to the physical address, also contains information about the size of the data block that has to be transferred. The block size in units of PCI data cycles is stored in a separate register.

2. Data Format

The event data that are transferred have to be available in a buffer linked with the DMA engine. However, mock-up data generation in the DMA domain is sufficient for the purpose of this work. The mock-up data word format applied is shown in Figure III-32. It allows a data word to be identified as unique within a certain time interval (more than 30 ms, given the 16-bit wide `CtrlMsgID` field and a 2 MHz input rate). This feature facilitates trouble shooting. The `CtrlMsgID` value is increased for each event while the `Index` value is increased for each PCI data cycle (Table III-1).

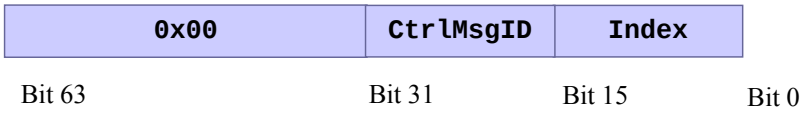


Figure III-32: Format of a Mock-up Data Word Transferred by the Feeding Nodes

Spare [63..32]	CtrlMsgID [31..16]	Index [15..0]	*64-bit PCI Data Cycle*	*128-byte burst number*
0	0	0	*1*	*1*
0	0	1	*2*	*1*
0	0	2	*3*	*1*
...				
0	0	15	*16*	*1*
0	1	16	*1*	*2*
0	1	17	*2*	*2*
0	1	18	*3*	*2*
...				

Table III-1: Mock-up Data Block

3. Flow Control Network Interface

The interface to the flow control network in the feeding nodes differs from the one presented in section III.B.5 and is therefore discussed separately.

Incoming flow control messages are first buffered in a receiving FIFO (Figure III-33). All fragments of a control message, each one 16-bit of size, are read out of the queue and deserialized to construct a full message, which is stored in an RX register.

The readout of flow control message fragments is controlled by an RX state machine (Figure III-34). The machine starts the readout as soon as the flow control network interface is enabled and a flow control message is pending in the RX FIFO. Upon completing the readout of the last fragment of a message, the machine terminates the RX FIFO readout and awaits the completion of the previous data transfer. As soon as the DMA machine has signaled 'idle state' (i.e. that it is ready to start a new transfer), the RX state machine produces a trigger to the DMA logic. At that stage the flow control message available in the RX register is already interpreted and the descriptor buffer feeds the DMA engine with a valid physical address. The DMA machine then undertakes the data transfer to the specified address.

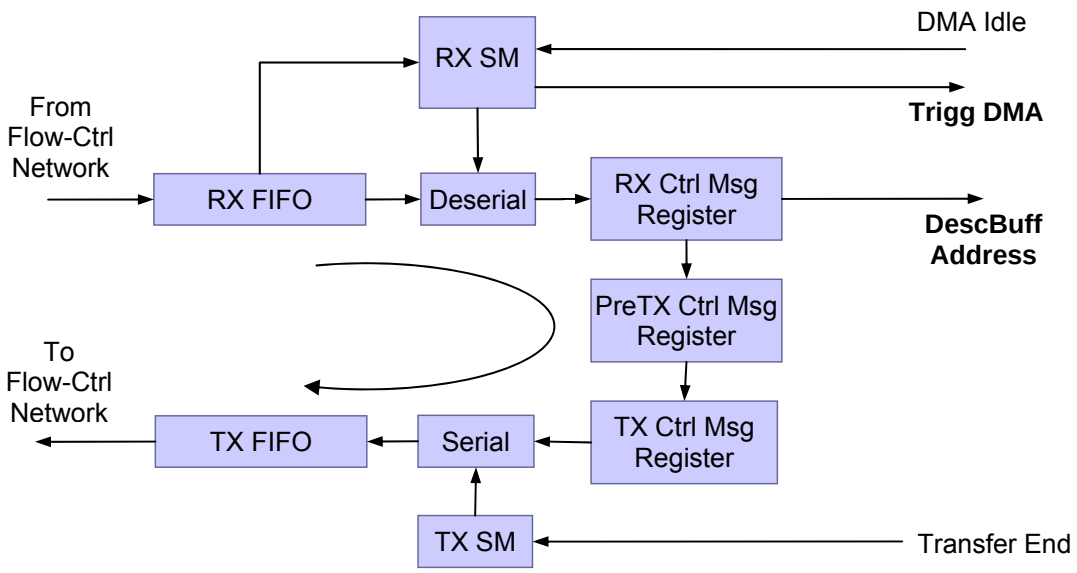


Figure III-33: Architecture of the Flow Control Network Interface in the Feeding Nodes

The RX state machine switches back to idle state to start a new iteration. In order to reduce the duration of the gaps between successive transfers to a minimum and so to improve the overall system performance, the readout of the next pending flow control message is started immediately against the background of the ongoing data transfer.

Following the rules for data flow control (section III.B.2), the control message associated with the ongoing data transfer must not be forwarded to the next feeding

node before transfer completion. Therefore this flow control message is temporarily kept in its place. Upon ending the transaction a TX state machine undertakes the forward procedure.

There might be a certain intermediate period of time, especially in case of a high number of flow control message fragments, when the readout of a new flow control message that is going to be stored in the RX register overlaps with the forwarding of the last message. Since the message associated with the current transfer must be protected, it is written to an intermediate PreTX register. A flow control message is written to that register immediately after the initiation of the corresponding transfer and stays there until completion. Then, the TX state machine (Figure III-34) copies it to the TX register. The latency caused by the intermediate 'CopyNew' state is negligible compared to the latency introduced by the RX FIFO and TX FIFO.

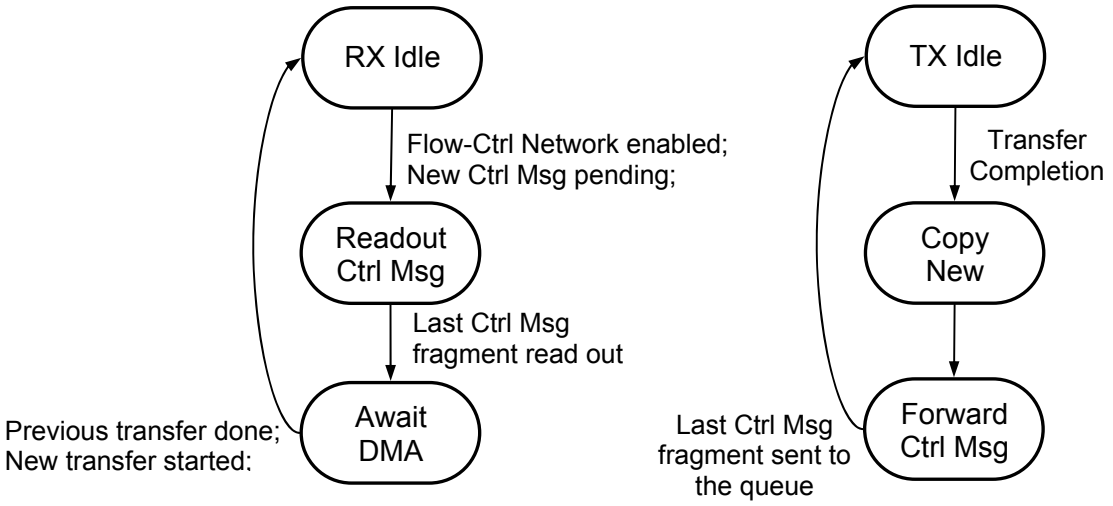


Figure III-34: State Diagrams of the State Machine of the Receiver Side (left) and the State Machine of the Transmitter Side (right) in the Flow Control Network Interface Unit of a Feeding Node

4. Implementation

The logic in the feeding nodes is implemented using FPGA devices since they provide the flexibility needed. Contemporary FPGAs support 66 MHz PCI functionality and allow the implementation of complex state machines and various kinds of embedded memory units. These are needed for the implementation of the DMA engine and the complemental logic. In addition, an FPGA-based implementation here contributes to price efficiency since there are a relatively few nodes which have to be equipped with DMA engines.

To access the PCI bus, the FPGA is located on a PCI expansion card. The initial tests of the transfer mode (section II.C) have been realized [9] using commercial 32-bit/33MHz PCI cards [69]. Commercially available 64-bit/66MHz PCI cards [70] have been substituted [9] for the 32-bit/33MHz cards to achieve higher performance. Subsequently the CIA-RORC card (Figure III-35) was developed (section III.C). The design of the DMA engine and the associated logic was adapted to the CIA-RORC board to overcome the lack of commercial cards. A commercially available general purpose PCI core was utilized to implement the PCI interface [41].

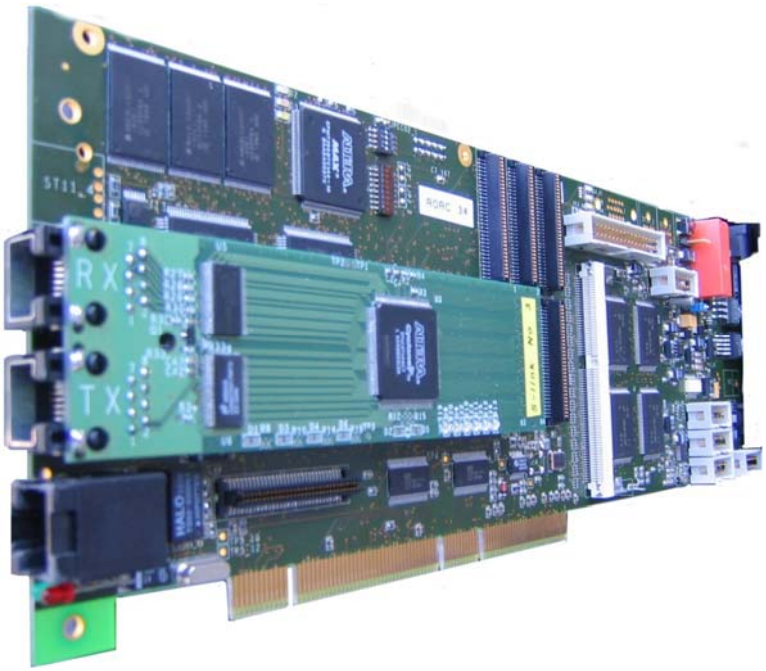


Figure III-35: The CIA-RORC board (section III.C) is a multi-purpose device. One of its major applications is the DMA engine in the feeding nodes. The photo shows the board with the flow control network interface card (Figure III-26) mounted on it.

5. Configuration, Control and Status of the DMA Engine Domain

The design of the DMA engine and the associated logic was parameterized to meet the scalability requirements. The parameters are presented in section VII.D.

Before the DMA engine can be used the addresses corresponding to all remote computing nodes in the system have to be written into the descriptor buffer as discussed in section III.E.1.a). Therefore, the buffer is a memory mapped region which can be accessed via the feeding node's local bus. The design of the DMA engine also implements a series of control and status registers which were used to configure the device, to get status information, and to ease trouble shooting. A detailed list of the registers is available in section VII.E, whilst a brief description of their purpose follows:

- The instantaneous data transfer rate, measured as the time between two successive data transfers in units of clock cycles, is stored in a separate register. Test software was used to periodically read out this register and thus to determine the system performance. The method of measurement is realized as shown on Figure VII-2 in section VII.E.
- Monitoring the expansion bus in the feeding nodes for target retries can provide an indication of network congestions. This is explained further in section IV.E. A separate status register was used to monitor the rate of retries measured and thus to monitor the rate of congestions.
- The number of successful data transactions executed by the DMA engine can be read out of a dedicated register. In the prototype, each feeding node executes a single data transfer per event. Hence, the values of these registers read out of all feeding nodes at a certain point of time must be equal. Thus, the feature has been used to ascertain the integrity of the flow control chain.
- Every data transfer is characterized by a descriptor that, in addition to the address of the remote location, also contains information about the size of the data block that has to be transferred. Initialization software writes the block size through a separate control register.
- Additional control and status registers were used to monitor the busy/idle status of the DMA engine, to enable and disable the data transfer, and to monitor buffer overflow flags of the flow control network interface in the feeding nodes.

F. Scheduler II

According to the concept of the data flow control system presented in section III.A every computing node sends a unique feedback message to the scheduler to state that it has completed event processing and that the receiving buffer was released. The way such messages are transferred is discussed in section III.B.6. Upon arriving at the scheduler, the feedback messages queue up as explained in section III.B.4. The scheduler assigns every event to a computing node, based on a rule described in section III.B.2. In order to transmit the event data to a specific computing node, the scheduler encodes the information regarding the location of the event in the feeding nodes as well as the address of the chosen computing node. It then sends the produced flow control message to the feeding nodes. The flow control messages are transferred via the custom serial network presented in section III.D. Upon receiving a flow control message each feeding node interprets the information encoded and finally transmits the specified event to the computing node chosen by the scheduler. The model summarized above is depicted in Figure III-36.

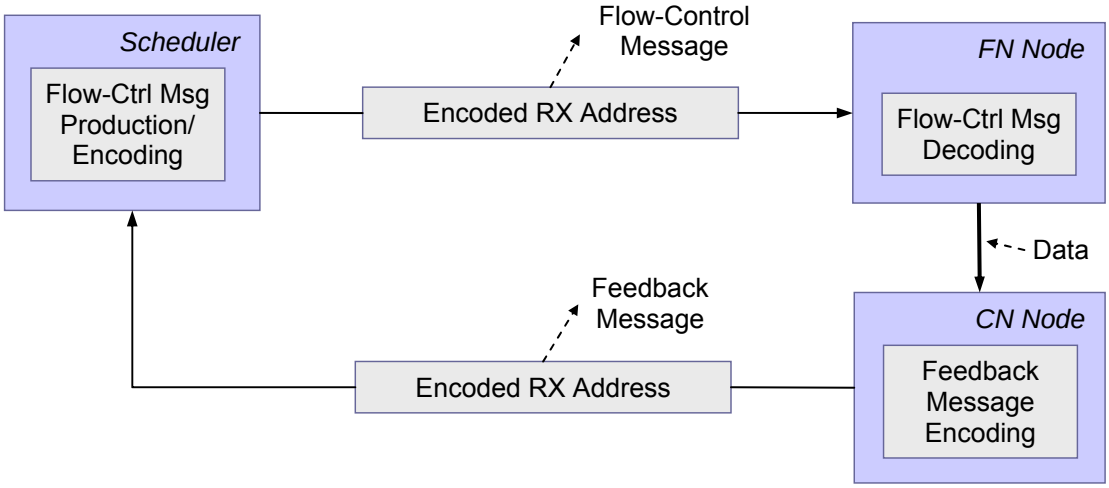


Figure III-36: Certain Data Paths within the System

The original version of the scheduler presented in section III.B utilizes flow control messages formatted as specified in Figure III-28. Computing node addresses are encoded by three coordinates: `COLUMN` and `ROW` indicate the x-y coordinates of a computing node within the 2-D torus; and the `RX_BUFFER` points to one of a minimum of two receiving buffers per computing node. The interpretation of such a flow control message involves processing and the corresponding processing time since the vector (`COLUMN`, `ROW`, `RX_BUFFER`) has to be translated into a descriptor buffer address (as described in section III.E.1.a). The elimination of the need for this processing is only possible if descriptor buffer addresses are supplied directly by flow control messages.

Since certain parameter values needed for calculating the descriptor buffer offset are known to each computing node, intermediate encoding and subsequent decoding of destination addresses as done so far is redundant. Therefore, the linear descriptor buffer address `DESC_BUFF_ADDRESS` has been substituted for the vector (`COLUMN`, `ROW`, `RX_BUFFER`) (Figure III-37). This is determined by the expression: $DESC\_BUFF\_ADDRESS = MY\_NODEID + i.nCN\_Nodes$.
MY_NODEID identifies each computing node. Its value is passed to the node during the initialization phase. The value of *i* is the number of receiving buffers per computing node; and *nCN_Nodes* is the number of computing nodes in the system. The way the value of *MY_NODEID* is specified correlates with the order in which the computing nodes are written into the descriptor buffer in the feeding nodes. The field `COL_POSITION` is still needed for proper queuing in the scheduler.

Figure III-37: Format of a Feedback Message in Scheduler II

The format of the flow control message was also changed (Figure III-38).

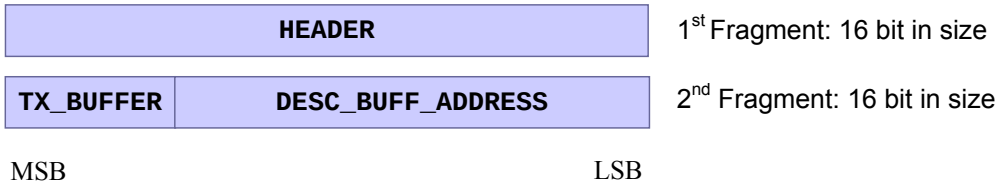


Figure III-38: Format of a Flow Control Message in Scheduler II

The new approach presented above is characterized by a number of advantages. As a result the system is simpler, faster, and scales better.

The system performance was improved since the removal of a stage reduces latency. Higher latency in the feeding node's logic means bigger idle gaps between data transfers, which reduces the system performance considering the highly intensive data transfer. A single transfer comprises one PCI clock cycle address phase, six clock cycles target setup time, and sixteen clock cycles data phase for transfer of 128 byte (Figure IV-3). Hence, the time t_{TRANSF} needed for a single data transfer equals $23T_{PCI\_CLK}$ (T_{PCI_CLK} is the PCI system clock cycle duration). Each data transfer is followed by a gap (Figure III-39). The time between two successive data transfers is

represented by $T_{EFF\_TRANSF} = t_{TRANSF} + t_{GAP} = 23T_{PCI\_CLK} + iT_{PCI\_CLK}$ (t_{GAP} is the idle time between two successive transfers measured in units of i numbers of PCI clock cycles). Induction of Scheduler II leads to a reduction of t_{GAP} since flow control message decoding is not needed. Hence, the time between two successive data transfers is $T^{NEW}_{EFF\_TRANSF} = 23T_{PCI\_CLK} + (i-1)T_{PCI\_CLK}$, assuming that one PCI clock cycle was saved. The ratio $\frac{f^{NEW}_{IN\_DATA\_RATE}}{f_{IN\_DATA\_RATE}} = \frac{T_{EFF\_TRANSF}}{T^{NEW}_{EFF\_TRANSF}} = \frac{(23+i)T_{PCI\_CLK}}{(23+i-1)T_{PCI\_CLK}} = \frac{23}{22} = 1.045$ shows an approximately 5% higher input data rate $f^{NEW}_{IN\_DATA\_RATE}$ compared to the input data rate $f_{IN\_DATA\_RATE}$ of the original version.

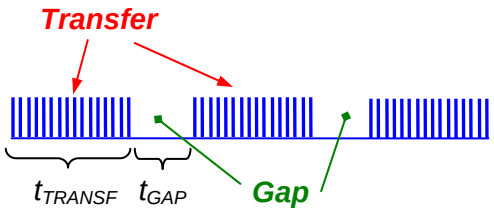


Figure III-39: Transfer of small packets at a high rate requires improvement of the t_{TRANSF}/t_{GAP} ratio.

The elimination of flow control message decoding in Scheduler II improves the system scalability, especially in case of big system sizes. A bigger system size requires a wider decoding unit. Considering the implementation of the logic in the feeding nodes (section III.E.4), it should be noted that both, product term and look-up table units in programmable logic devices, are characterized with a limited fan-in. Even though the realization of 'wide' decoders is possible, it is not recommended since it results in poor design performance. The latter characteristic is of great importance here, as discussed above. Scheduler II eliminates any design technique problems related to scalability at this stage.

The modifications introduced do not require any design modifications in the original version of the scheduler since the design is parameterized. The modifications of the logic in the feeding nodes have led to a simpler architectural solution. The other modifications concern feedback message encoding, which is realized in a single line software code.

IV. Flow-Controlled System Performance

The chapter presents tests performed in a farm prototype with an integrated traffic shaping system and analyses the measurement results obtained. In particular, the work focuses on:

- analysing the traffic on the expansion bus in both, the feeding and the receiving nodes
- determining the maximum system input data rate
- measuring the latency of the flow control network
- measuring the latency of the processing farm and thereby confirming certain system parameters determined in [9].

Based on detailed analyses of the traffic on the expansion bus in the feeding nodes, the chapter demonstrates the ability of the traffic shaping system to avoid network congestions.

A. Test Set-up

The results presented in this chapter are obtained by a 16 nodes 2-D torus system prototype (Figure IV-1). Each node is a standard Linux PC[1] equipped with two Pentium III processors[2] and either a ServerWorks IIIHE-SL or a ServerWorks IIIHE chipset[3] [27]. The nodes are interconnected by commercial 64/66 PCI SCI network adapters [23, 24] built around the SCI link controller LC3 [26].

[1] a current SuSE Linux distribution and an up-to-date kernel version
[2] either 800 MHz or 733 MHz
[3] both chipsets support 64-bit/66 MHz PCI

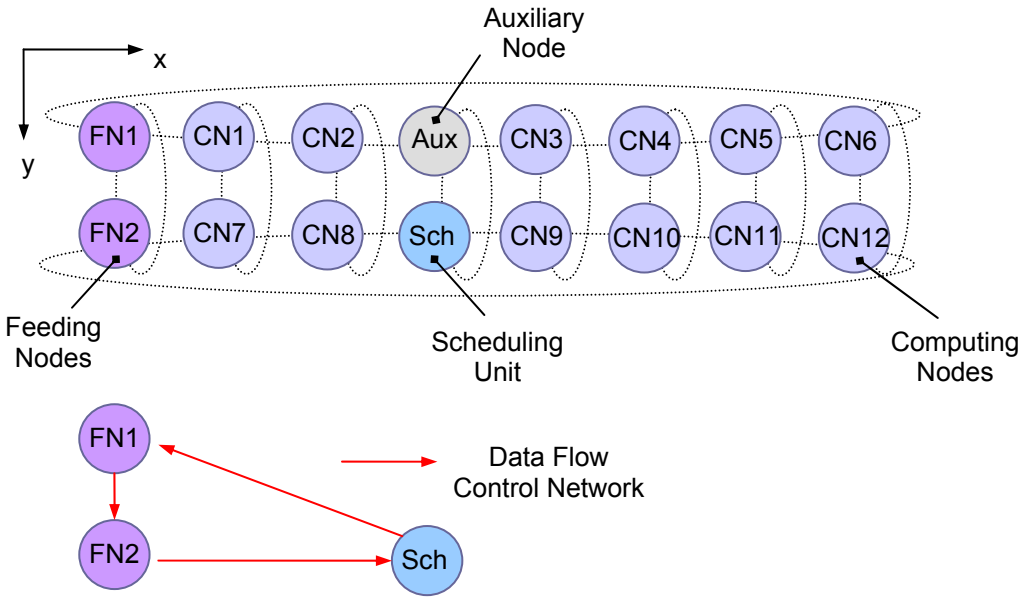


Figure IV-1: A Schematic View of the Test System Setup

The 16 nodes interconnected to form a 2 x 8 torus are distributed as follows:

Feeding Nodes: Two nodes serve as feeding nodes: these are the two nodes on the far left of Figure IV-1. Each one is equipped with a CIA-RORC card (section III.C) which is configured appropriately (section III.E). The card is equipped with a flow control network interface card (Figure III-26). The CIA-RORC board and the local interconnect adapter card reside in a mutual 64-bit/66 MHz PCI bus so that the feeding node's DMA engine writes data directly into the network adapter without traversing a PCI bridge.

Scheduling Unit: One node is dedicated to be the scheduler. It is equipped with a CIA-RORC card which is configured correspondingly (section III.B.7). The card is located on the 64-bit/66 MHz PCI bus together with the local network adapter for better performance. There is a flow control network interface card installed on the scheduler's CIA-RORC card, which serves to interface the flow control network.

Auxiliary Nodes: One node is used passively to realize measurements, like the one shown in Figure IV-11. That node is equipped with a CIA-RORC card configured according to the desired measurement set-up. Although not used as a receiver, the node is still important for data transport since it realizes network data by-pass and re-routing according to the 2-D ring topology presented (section II.D).

Computing Nodes: The remaining twelve nodes are used as receivers. Some of them are equipped with auxiliary CIA-RORC cards to realize measurements like the one shown in Figure IV-9. The local network adapter in each receiving node is located on the 64-bit/66 MHz PCI bus.

The system is stimulated by means of a factory-made waveform generator. All waveforms presented in the current chapter are realized with a logic analyzer embedded in the FPGA of the CIA-RORC card [71, 72]. The test set-up is shown in Figure IV-2.

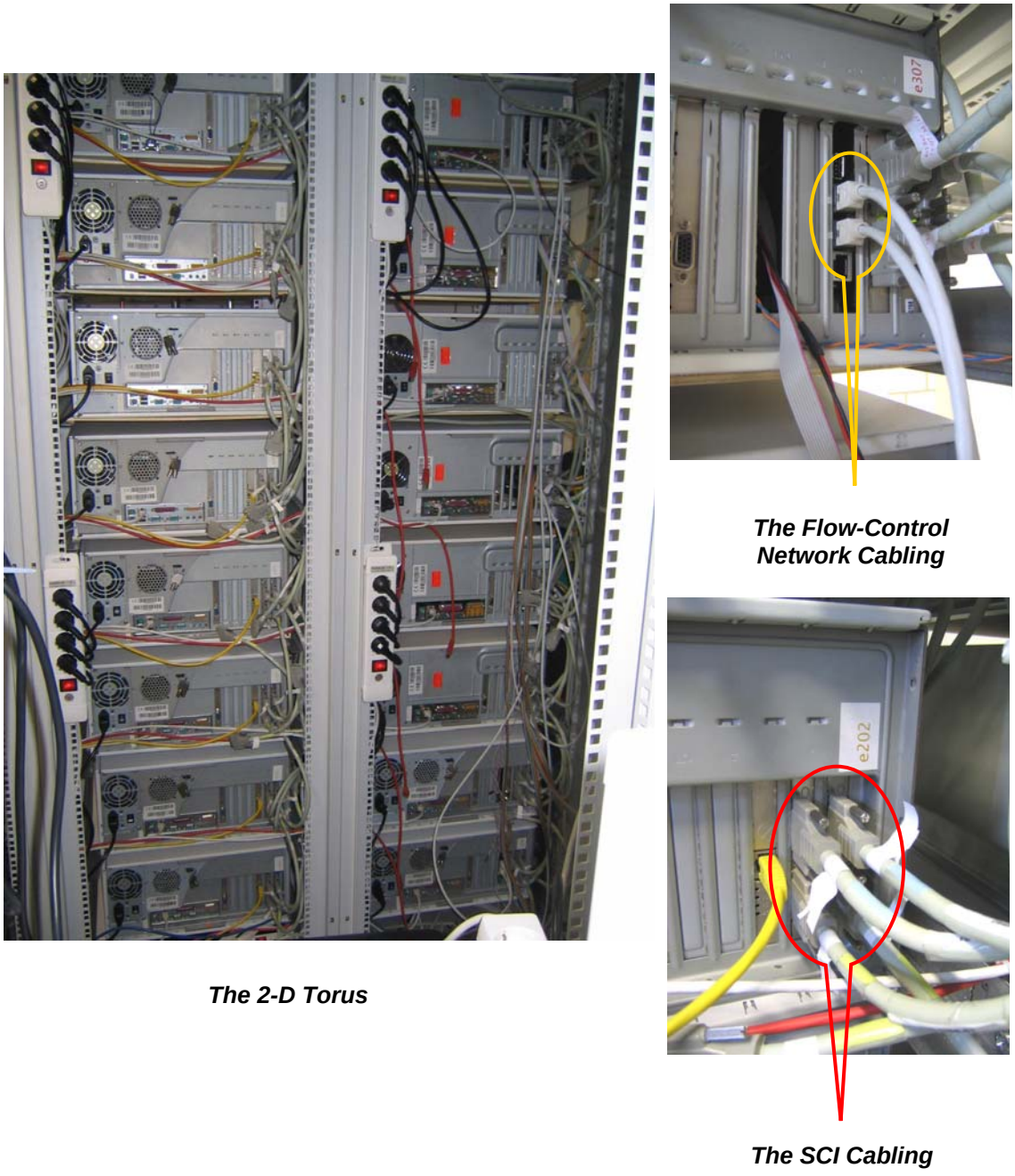


Figure IV-2: Test Setup

B. System Frequency

The maximum system frequency of a 3 x 10 torus with a basic implementation of a traffic-shaping mechanism has been determined and presented in [9]. The flow control mechanism used lacks a scheduling unit and does not implement input queues for flow control messages at the feeding nodes. It realizes the static allocation of computing nodes, where the address to the descriptor buffer (section III.E.1.a) increases by one after each data transfer. The latter measure prevents the sending of subsequent events to the same torus column and thus controls the load of a vertical ring for the reasons discussed in section II.F. A simultaneous transmission of data from multiple feeding nodes to one and the same computing node does not take place since a feeding node forwards a flow control message after it has finished its own data transfer. Moreover, a sender does only forward a flow control message to the next one if the latter does not signal 'busy'. Thus, control messages are never lost. A flow control message itself consists in a single bit pulse since it does not carry any information except the yes/no transmission trigger.

The maximum input frequency in the system presented above has been limited to a maximum of about 1.4 MHz for 128 byte packets [9]. The limitation is due to the busy logic which accounts for an additional overhead of multiple idle clock cycles per transfer. The maximum input frequency in a system with dynamic allocation of computing nodes was determined in this work. The peak input frequency for the transfer of 128 byte packets in the test system set-up presented in section IV.A, which is controlled by the traffic management system presented in chapter III, is 2.15 MHz (Figure IV-3). The rate is measured as the time between two successive data transfers on the feeding node's PCI bus. It is shown that the time the DMA machine remains idle between two successive transfers consists in a single clock cycle (this is the positive logic signal CNTLout[1] that signals the 'ready' status of the DMA engine), which has been achieved by adding pipeline functionality to the feeding node's logic and by additional optimization of the DMA logic used.

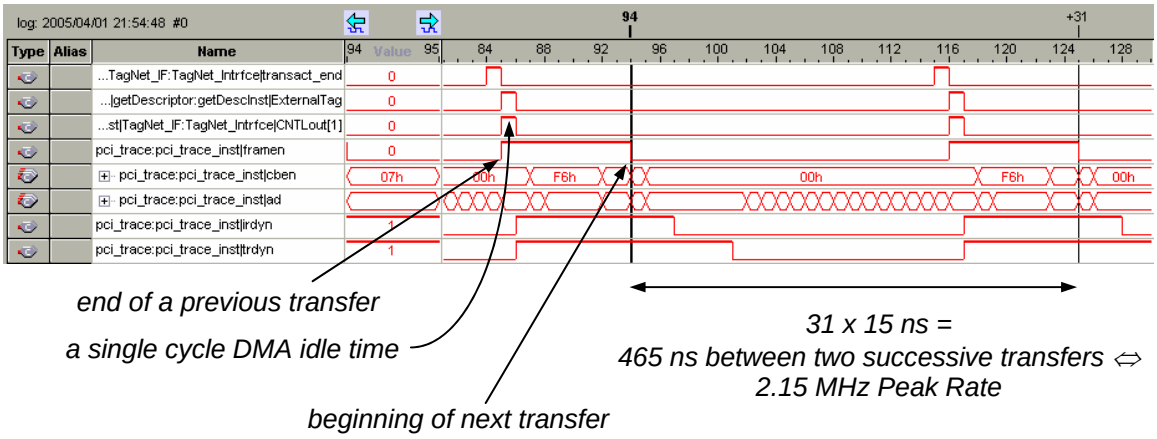


Figure IV-3: A PCI trace in a feeding node operated at the maximum input rate of 2.14 MHz. The transfer of one sub-event from the DMA engine to the local network adapter comprises one PCI clock cycle address phase, 6 clock cycles target setup time, and 16 clock cycles data phase for the transfer of 128 byte.

Figure IV-4 shows a series of data transfers on the PCI bus in the feeding node. The average input frequency in the system presented here was determined to be 2.14 MHz for 128 byte packets.

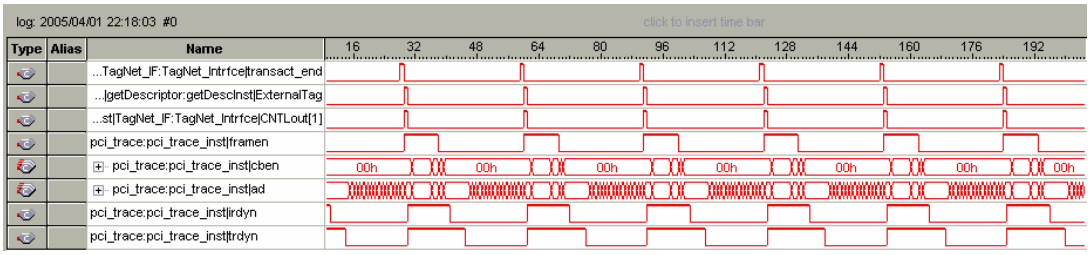


Figure IV-4: Series of 128 Byte Bursts on the PCI Bus in the Feeding Node

The difference between the peak and the average input system frequency is due to the fact that the SCI driver running on the feeding nodes regularly accesses the local SCI network adapter and the remote nodes. The activity of the driver causes additional traffic on the feeding nodes' PCI bus, which is observed as transactions targeted to the non-prefetchable PCI address space of the local SCI adapter. The rate of such transactions has been measured to be in the order of a few hundred hertz. During such an event the DMA engine located in the feeding node is unable to acquire PCI bus mastership. The first waveform in Figure IV-5 shows how flow control messages that trigger the DMA engine arrive at the feeding node at a rate of approximately one megahertz. The second signal indicates that the DMA engine is not ready to initiate data transfer for a certain time frame, while the input trigger messages still queue up. During that time no packets are sent, and no flow control messages are forwarded to the next feeding node, as it can be seen in the last waveform. The heartbeat functionality performed by the SCI driver can be turned off by manipulating the device driver.

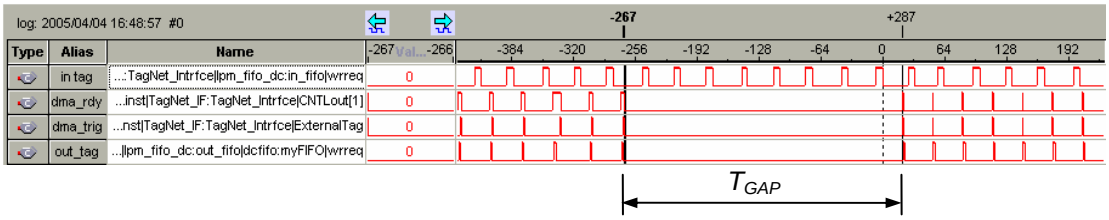


Figure IV-5: The figure demonstrates the inability of the DMA engine in the feeding node to process the continuously incoming trigger pulses. The gap, whose duration T_{GAP} varies from event to event (and in this particular case equals approximately 10 µs), is caused by the SCI driver running locally that regularly checks the local SCI adapter and the remote SCI nodes.

The data traffic on the PCI bus in a computing node is small and, on average, equals $\left(Event\_proc\_time^{-1} \cdot Event\_size\right)$, where *Event_proc_time* is the average time needed for the trigger algorithm to process an event, and *Event_size* is the average event size. Assuming an event processing time of 1 ms and an event size of 4 kbyte, the average data traffic on the PCI bus in a computing node is 4 Mbyte/s, which is rather smaller than the PCI bandwidth[1]. The event contributions from all feeding nodes arrive at a computing node in the order in which they were sent (Figure IV-6).

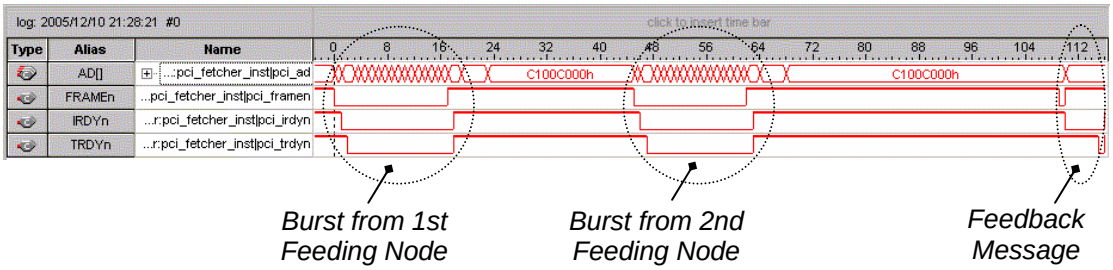


Figure IV-6: A PCI Trace in a Computing Node

The maximum input frequency was determined in a system where the process running on certain computing nodes is intentionally suspended for a certain period of time. This test setup emulates a longer event processing time due to statistically larger events and demonstrates the ability of the traffic shaping system to overcome such occurrences. In the test setup, there is always one computing node suspended; and immediately after its resumption a computing node in another torus column is suspended. This was verified by regularly probing the buffers holding queued free computing node addresses (Figure IV-7). The test showed stable system operation (Figure IV-8). The process suspension was realized using the `usleep()` function which suspends the process execution for at least the specified number of microseconds. The actual delay inserted by the `usleep()` function may differ from the suspension time specified and was therefore independently measured in hardware (Figure IV-9). The suspension is triggered using a modified mock-up data word (Figure IV-10).

[1] The theoretical peak bandwidth for 64-bit 66 MHz PCI is 528 Mbyte/s [25].

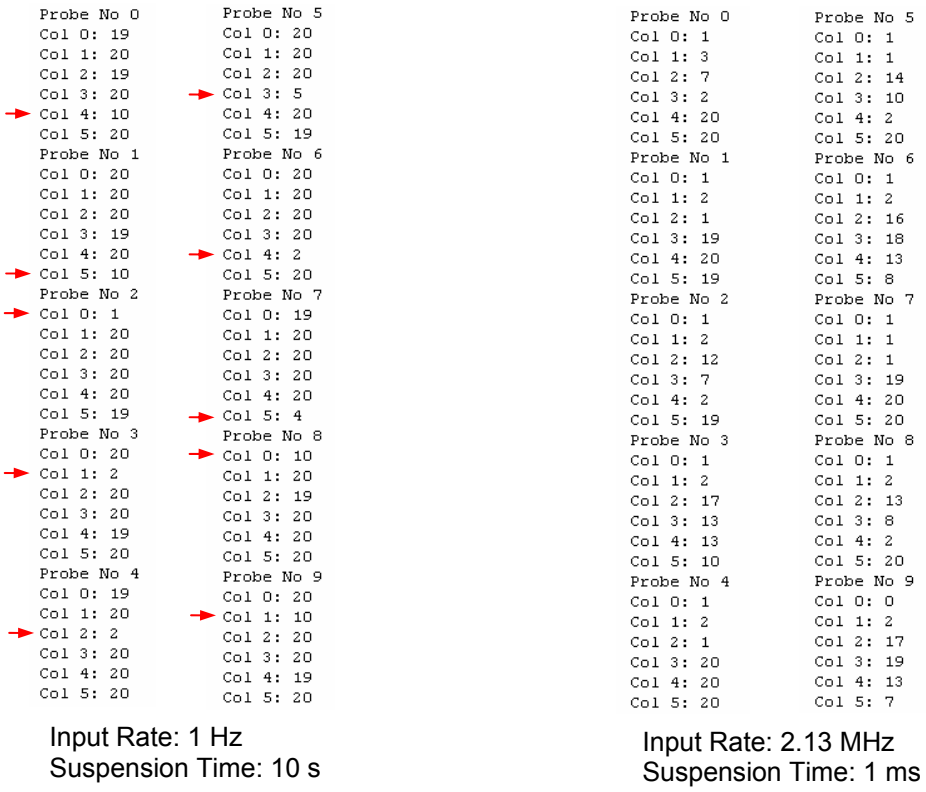


Figure IV-7: The number of receiving buffers per computing node is set to 10. Hence, the maximum number of receiving buffers in a torus column is 20. The log file on the left demonstrates how the suspended node moves. A new measurement is performed every 10 s in a system operated at the rate of 1 Hz with a suspension time of 10 s. The log file on the right shows the status in a system operated at 2.13 MHz with a suspension time of 1 ms.

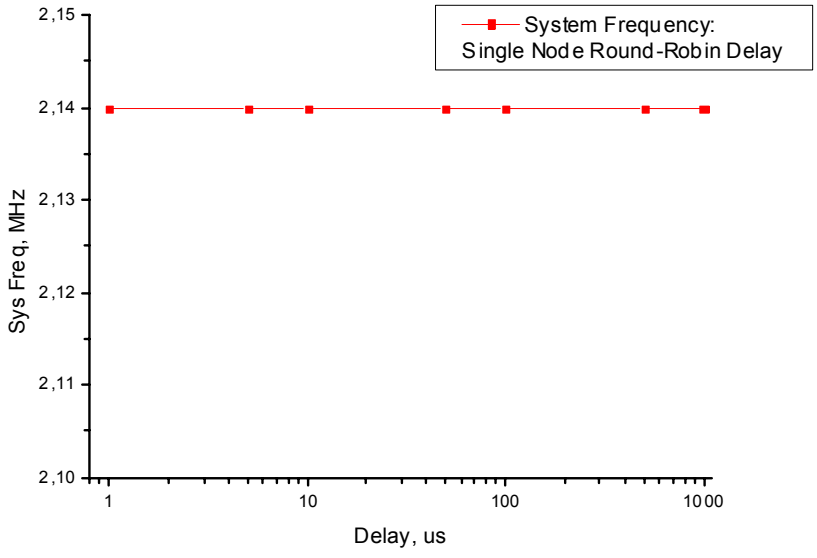


Figure IV-8: Maximum system frequency vs. computing node suspension time. One node at a time is suspended. Immediately after its resumption a node in the torus column next to the previous one is suspended.

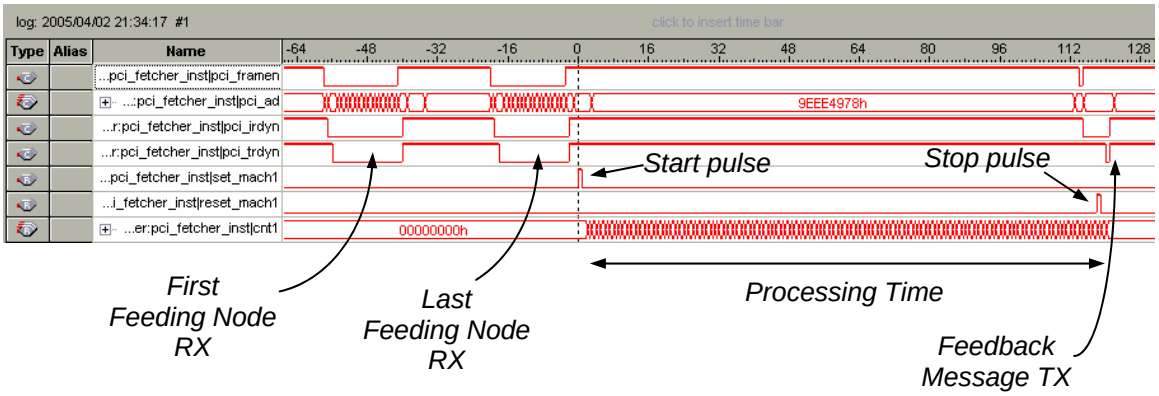


Figure IV-9: The suspension starts immediately after receiving the sub-event sent by the last feeding node. When the suspension expires, the computing node transmits a feedback message to the scheduler. The suspension time is measured in hardware: the PCI clock cycles between the start and the stop signal are counted and periodically read out.

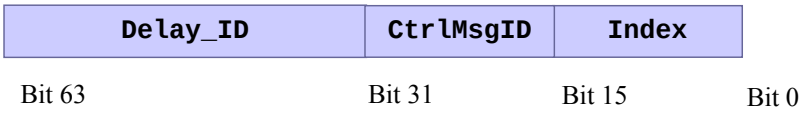


Figure IV-10: A modified mock-up data word, where the upper 32 bits that are not in use in the original version (Figure III-32) contain the ID of the node that has to be suspended. If a computing node receives data containing its own ID, the node suspends the process execution.

C. Flow Control Network Latency

The flow control message latency through a flow control network segment was measured. The latency was defined as the time between the transmission of a flow control message by a flow control network agent (Figure III-24) and the transmission of the same control message by the agent next to it. Here, the term ‘transmission’ means the writing of the flow control message into the TX FIFO of the flow control network interface unit (Figure III-12, Figure III-33).

Table IV-1 shows the time between the transmission of a flow control message by the scheduler and the transmission of the message by the first (position 1) and the second (position 2) feeding node. The measurements are performed in hardware (Figure IV-11).

The message latency in a flow control network segment of approximately 1.4 µs can be reduced by optimizing the way the clock synchronization in the flow control network interface is realized (Figure IV-12). However, although the latency in the test setup is relatively high, the transfer of a new event can start before the transfer of the

previous one is completed. This is possible since the flow control system network is used in a pipelined fashion, which allows achieving the input system frequency presented in section IV.B.

1	Scheduler to First Feeding Node Flow Control Message Latency, µs	1.38 ± 0.02
2	Scheduler to Second Feeding Node Flow Control Message Latency, µs	2.79 ± 0.06

Table IV-1: Average flow control message latency through the network. A flow control network agent adds approximately 1.4 µs to the total flow control message latency through the chain in the test set-up.

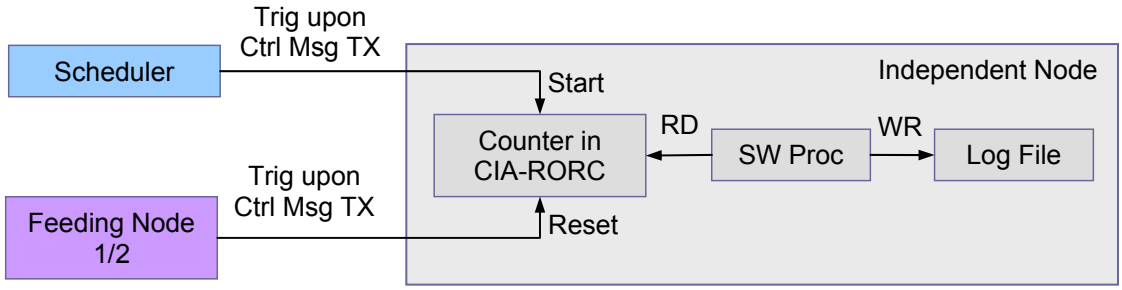


Figure IV-11: Flow control network latency measurement setup. A trigger pulse produced upon writing a specific flow control message into the output FIFO in the flow control message interface of the scheduler is fed to an independent auxiliary node to start a counter. A second trigger pulse produced upon writing the same message into the output FIFO in the feeding node copies the instantaneous counter contents to a result register and clears the counter for the next measurement. The result register is regularly read out by a software process running on the auxiliary node, and the measurements are stored in a log file.

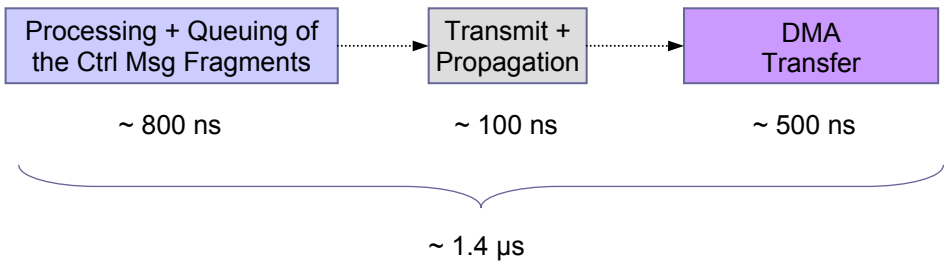


Figure IV-12: Approximate distribution of the flow control message latency in a network segment. A substantial fraction of the total latency is caused by the dual-clock FIFOs [73] in the flow control network interface unit (Figure III-12, Figure III-33), and especially by the TX FIFO, whose input clock is set to the local design system frequency of 70 MHz, while data are read out at the system frequency of 30 MHz of the flow control network interface card, which causes large latency for the proper clock synchronization to be performed.

The average flow control message latency in the network measured at different input rates demonstrates that the flow control system network latency does not depend on the input system frequency (Table IV-2). A large deviation from its average value is observed (Figure IV-13) due to the activity of the SCI device driver in the feeding nodes discussed in section IV.B. However, this can be avoided by modifying the driver.

Flow Control Message Latency between Scheduler and Second Feeding Node at Input System Frequency *100 kHz*, µs	2.79 ± 0.02
Flow Control Message Latency between Scheduler and Second Feeding Node at Input System Frequency *1.7 MHz*, µs	2.79 ± 0.06

Table IV-2: Average Flow Control Message Latency through the Network Determined at Different Input System Frequencies

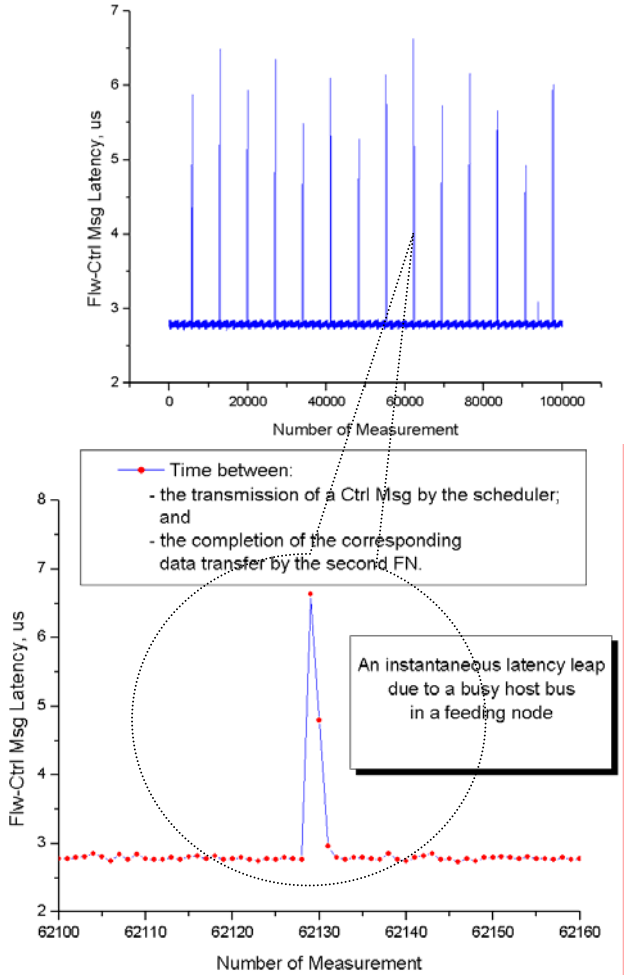


Figure IV-13: When the PCI bus in a feeding node is utilized by the local SCI device driver, whose heartbeat functionality is also seen here, the DMA engine is unable to send events, and the incoming flow control messages queue up (Figure IV-5) in the RX FIFO of the flow control message interface unit (Figure III-33). Consequently, the latency of the affected flow control messages increases.

D. System Latency

The system latency, defined as the time between the transmission of a flow control message by the scheduler and the arrival of the corresponding feedback message back at the scheduler, was measured. The measurement was performed by means of a counter in the scheduling unit. The counter is started when the scheduler core triggers the flow control interface unit to transmit the control message (point 9 in Figure III-4). The counter is stopped when the corresponding feedback message is written into the destinations' buffer (point 1 in Figure III-4). Assuming equal processing time in all computing nodes, the latency differs for every computing node due to the different packet network paths.

In this particular test set-up (Figure IV-14), the packet latency when CN7 is the receiver is $\tau_{CN7} = \tau_{SCH\_INTRNL} + \tau_{SCH-FN2} + \tau_{FN2-CN7} + \tau_{CN7\_PROC} + \tau_{CN7-SCH}$, where:

- τ_{SCH_INTRNL} is the latency caused by the scheduler, which is negligible;
- $\tau_{SCH-FN2} \approx 2.8\mu s$ is the latency through the flow control system network determined in section IV.C;
- $\tau_{CN7\_PROC} = T_{NO\_PROC} + T_{SUSPEND} \approx 1.5\mu s + T_{SUSPEND}$, where $T_{NO\_PROC} \approx 1.5\mu s$ is the mean time between the reception of a full event in the computing node and the transmission of the corresponding feedback message when another processing is not performed in-between. T_{NO_PROC} was measured using the method depicted in Figure IV-11: the trigger signals, produced by a CIA-RORC card analysing every transaction on the expansion bus of CN7, are fed to an independent node measuring the time in hardware. The 'Start' pulse is generated upon completion of the transfer of the last event fragment (i.e., the one sent by FN2) through the expansion bus of CN7. The 'Reset' pulse is generated upon transferring the corresponding feedback message through the expansion bus to the local network adapter in CN7. $T_{SUSPEND}$ is the duration of the process suspension inserted by the `usleep()` function to emulate event processing;
- $\tau_{FN2-CN7}$ and $\tau_{CN7-SCH}$ are the latencies of the second sub-event and correspondingly the feedback message through the SCI network. The packet latency through the SCI network for 128 byte packets has been determined in [9]. It can be calculated by the expression $T_{Lat\_By} \cdot x + T_{Lat\_PCI} = 0.06x + 1.40$ for non-routed data, where $T_{Lat\_By} = 60ns$ is the bypass latency of an SCI node, x is the number of nodes residing between the receiver and the sender, and $T_{Lat\_PCI} = 1.40\mu s$ is the so-called PCI-to-PCI latency [9]. The time $\tau_{FN2-CN7}$ can then be calculated as $\tau_{FN2-CN7} = 1.4\mu s$, taking into account that nodes are not present in the packet path between the second feeding node FN2 and the receiving node CN7 (Figure IV-14). The time $\tau_{CN7-SCH}$ equals $\tau_{CN7-SCH} = 0.06 + 1.40 = 1.46\mu s$ (the node CN8 bypasses the feedback message sent from CN7 to the scheduler).

The system latency in this particular case equals $\tau_{CN7} \approx T_{SUSPEND} + 7\mu s$. This agrees with the values measured in the test set-up (Table IV-3).

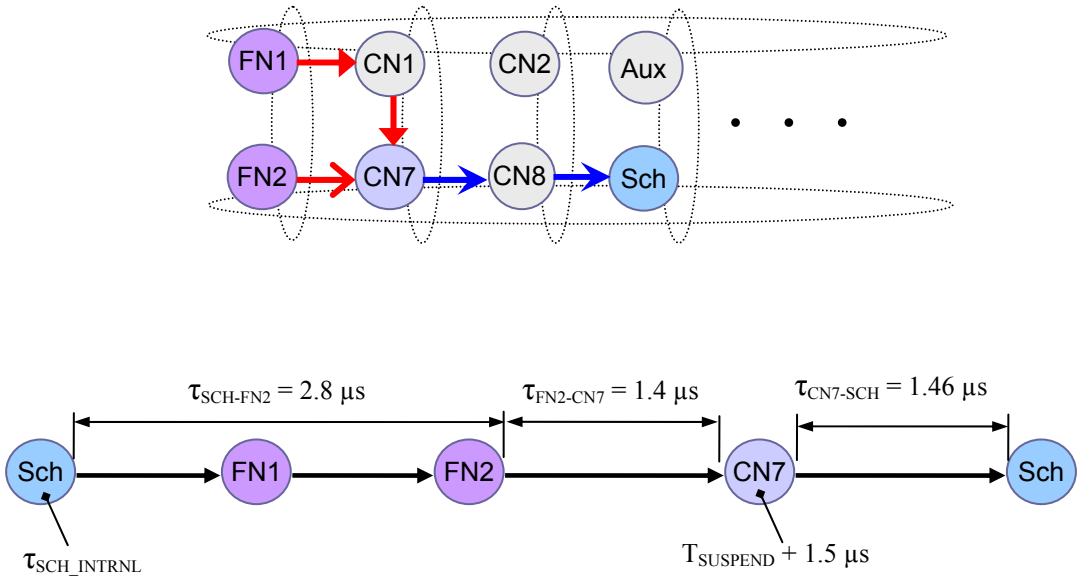


Figure IV-14: The packet latency through the system set-up. Taking into account the dimensional routing of network packets, the path of the sub-event sent by the first feeding node is FN1 – CN1 – CN7. The path of the second sub-event is FN2 – CN7. The feedback message path is CN7 – CN8 – Scheduler.

τ_{CN7}, µs	$T_{SUSPEND}$, µs	$\tau_{CN7} - T_{SUSPEND}$, µs
8.13 ± 0.86	1	7.13
8.95 ± 0.85	2	6.95
11.90 ± 1.02	5	6.90

Table IV-3: The Average System Latency Measured at Different Suspension Times $T_{SUSPEND}$ in the Computing Node CN7 (Figure IV-14)

E. Feeding Nodes' PCI Bus Analysis

When an SCI node is unable to receive a packet due to its input buffer being full (Figure II-11) it sends an echo to the transmitter to request packet retransmission. In that case the transmitter has to keep the packet buffered for retransmission. This leads to an overflow in the SCI node at the transmitting side if the node is heavily loaded, too. When the SCI adapter card in the feeding node gets overloaded, it throttles the speed of data feed by signaling PCI target retries to the DMA engine. Target retry is one of the transaction termination schemes defined by the PCI specification [25], where a PCI target device is capable of requesting the termination of a transaction, in which no data have been transferred. This can happen because the target device is unable to transfer data at this time or cannot meet the latency requirements. Therefore, the target requests the master that initiated the transaction to retry it later. Consequently, monitoring the PCI bus in the feeding nodes for PCI target retries can provide a sign of network congestions.

1. PCI Bus Idle Time Analysis

By definition, a PCI bus is idle when both the *framen* and *irdyn* signals are inactive [25]. Figure IV-15 demonstrates how the PCI bus idle time in a feeding node was measured in units of PCI clock cycles. The measuring counter is disabled for as many clock cycles as the PCI transactions take.

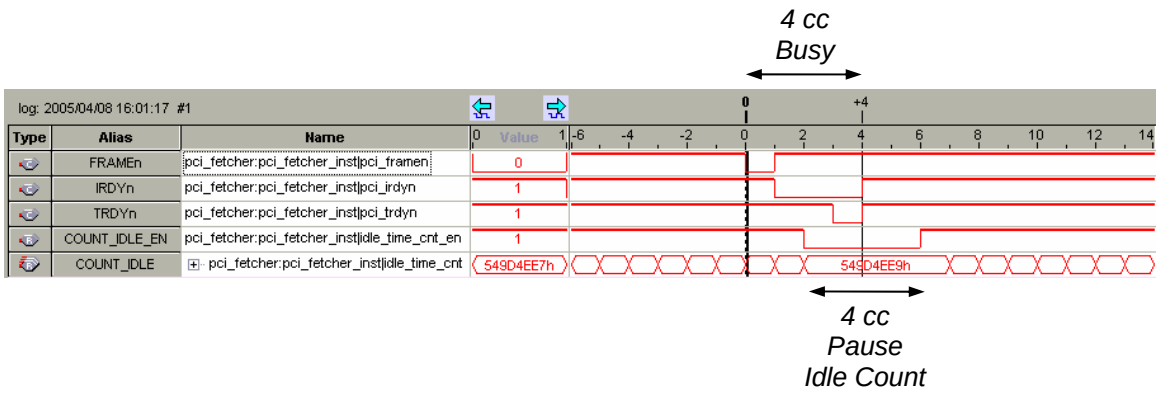


Figure IV-15: The PCI signals FRAMEn, IRDYn and TRDYn indicate a single data phase transaction with a duration of 4 clock cycles. Before and after the transaction, the PCI bus is idle since both the FRAMEn and IRDYn signals are inactive high. The counter COUNT_IDLE counts every clock cycle except for the four clock cycles when the PCI bus is non-idle and the counter enable signal COUNT_IDLE_EN is inactive low.

Table IV-4 contains the measured fraction of time when the feeding node's expansion bus is idle. Positions 2 and 3 refer to an idle system, where 'idle' means that

the scheduler is disabled and the feeding nodes are not sending data at all. The results show that even in an idle system the traffic on the PCI bus after SCI initialization is higher than that prior to the initialization phase. The traffic increases on account of the activity on the PCI bus discussed in section IV.B, which is caused by the SCI device driver.

1	Theoretical Idle Time in an Idle System	100 %
2	Idle Time in an Idle System *before* SCI Initialization	99.9998 %
3	Idle Time in an Idle System *after* SCI Initialization	99.9756 %
4	Idle Time in a Loaded System at 2.14 MHz Input Data Rate	25.8904 %

Table IV-4: PCI Bus Idle Time in a Feeding Node as Percentage of the Total Measurement Time

Position 4 in Table IV-4 shows the time the PCI bus in a feeding node is idle when the system is run at the maximum input data rate. Figure IV-16 shows that, although the flow control network interface in a feeding node triggers the DMA machine to transfer the next data packet immediately after the previous one and thus does not introduce latency, the new transfer is only initiated after a certain amount of time, which results in a substantial fraction of the PCI bus idle time. The time it takes for the DMA machine to actually start a new PCI transaction is over 25.8% of the period of the 128 byte burst data transfer in a feeding node run at the maximum input data rate of 2.14 MHz. The idle time introduced is needed for the deployed PCI core [74] to get the PCI bus granted by the PCI arbiter to the DMA engine's PCI master agent. The idle time can be reduced either by using another implementation of the local PCI interface or by setting two DMA engines in separate PCI agents working in interleaved mode [9]. However, the current consideration demonstrates that the transmitter's PCI bus is utilized completely under the given circumstances.

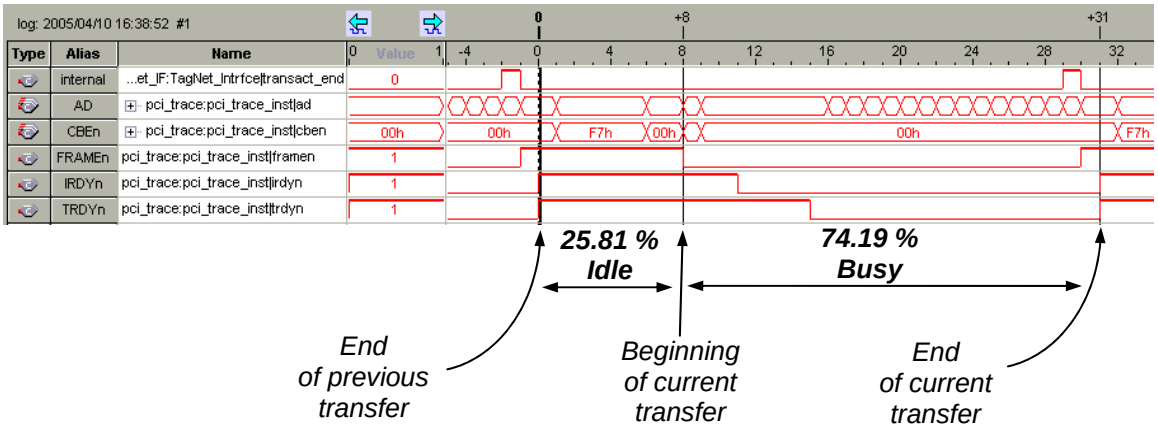


Figure IV-16: PCI Bus Idle/Busy Ratio in a Feeding Node Sending 128-Byte Bursts at the Rate of 2.14 MHz

2. Target Retry Analysis

A PCI target device is capable of requesting a transaction to be terminated[1] if the local side is not ready to receive or supply the requested data [25]. There are several termination schemes:

- Target retry: the target device does not accept any data phase because, for instance, it is not able to transfer data at this time or it cannot meet the latency requirements; and it requests the master that initiated the transaction to retry the same transaction at a later time.
- Target disconnect: the target device has already accepted at least one data phase but it needs to terminate the outstanding transaction. There is a target disconnect 'with data' when the target device requests a disconnect but the *trdyn* signal is still asserted since the local side is capable of receiving or supplying some more data before a disconnect finally takes place. There is a target disconnect 'without data' when the target device requests an outstanding transaction to be terminated immediately, ensuring that no more data phases take place. In the latter case, the target device requests disconnect while *trdyn* is already deasserted. A target disconnect does not force the initiator to retry the transaction.
- Target abort[2]: this is an abnormal transaction termination that occurs when a fatal error is detected or the target device cannot be expected to complete a requested transaction at all. The termination scheme allows graceful transaction termination in order to preserve the normal operation for other PCI agents.

A master device which has initiated a transaction considers terminations like those given above abnormal transaction terminations. The PCI core used detects such events and signals them to the local-side design [74]. Therefore, it is easy to count the number of abnormal transaction terminations.

The number of target retries and disconnects on the PCI bus in the last feeding node was measured. The system is run at the maximum input data rate of 2.14 MHz. Target disconnects have not been observed. The number of retries measured is shown in Table IV-5.

Mean number of PCI target retries in 2nd transmitter per 32 seconds	10359 ± 50
Target retries per second	324

Table IV-5: PCI target retries measured in the feeding node. The system is run at an input data rate of 2.14 MHz, which results in a data rate of about 274 Mbyte/s per feeding node.

The analyses presented in section IV.E.1 demonstrate that the transmitter's host PCI bus is utilized completely. Under these circumstances, the measured number of target retries in the transmitter is around 320 per second, which is commensurable to the number of PCI transactions initiated by the SCI driver against the non-prefetchable

[1] does not refer to configuration transactions
[2] Target abort is not of interest in the current consideration.

memory regions in the local SCI adapter as discussed in section IV.B. The SCI adapter requests a retry of the PCI write transaction initiated by the PCI master agent in the DMA engine whenever the SCI device driver performs its heartbeat functionality (Figure IV-17, Figure IV-18). Hence, the observed target retries are not due to network congestions.

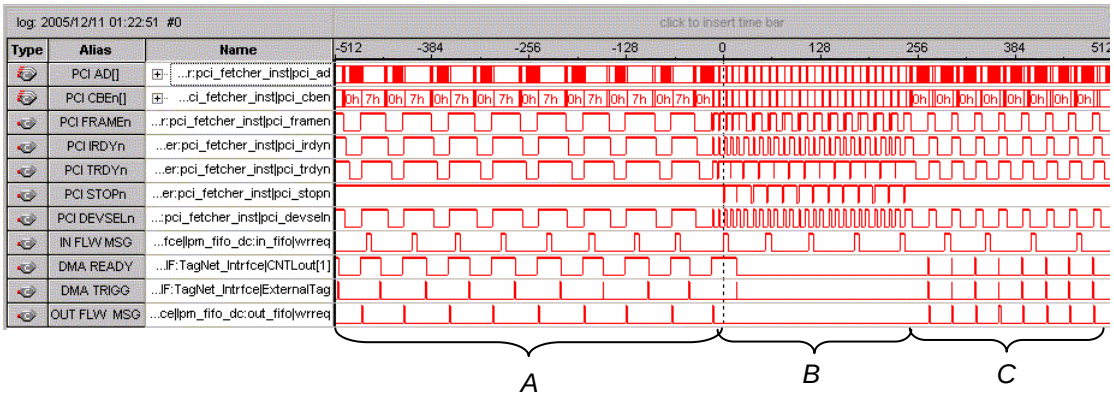


Figure IV-17: Section A in the figure illustrates the continuous transfer of data from the DMA engine to the SCI adapter in the feeding node. In section B, the SCI device driver performs its heartbeat functionality, which suspends the transmission of bursts. Data transfer is resumed in section C, where the rate is higher than the input rate since several input flow control messages have already queued up during the suspension. The events that occur in section B are shown in Figure IV-18.

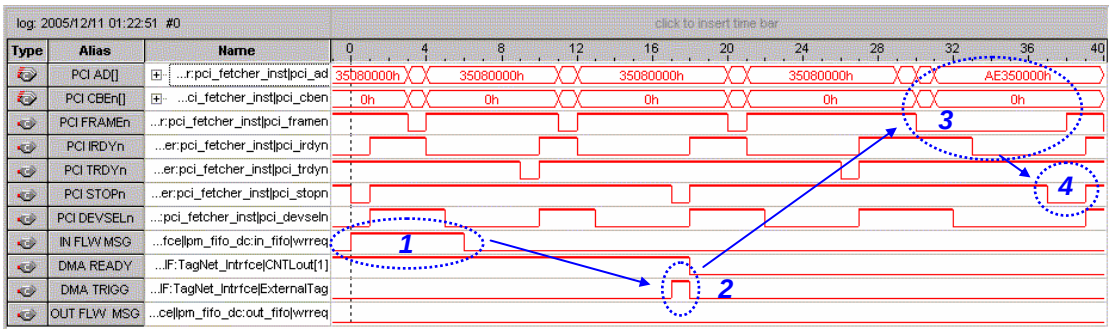


Figure IV-18: A flow control message enters the input queue in the DMA engine domain (1). After clock synchronization a trigger pulse to the DMA engine is produced (2). Although there is already active traffic on the PCI bus comprising single-cycle PCI transactions caused by the SCI device driver, the PCI bus is granted to the PCI master agent in the DMA engine domain. The PCI master agent then initiates a PCI write against the local SCI adapter by asserting the FRAMEn signal (3). However, instead of asserting the TRDYn signal to indicate readiness for the transaction, the local SCI adapter signals a target retry by asserting the STOPn signal (4). The DMA master agent then deasserts FRAMEn, and the transaction is postponed. The same scenario occurs several times in section B in Figure IV-17 until the SCI device driver has completed its activity.

Considering that the PCI bus at the transmitter side is utilized completely and that the number of PCI target retries by the local SCI adapter is understandably low, it is reasonable to conclude that the input data rate is limited by the transmitter but not by

the receiver. The measurements demonstrate that the data flow control system developed saves the receiver from congestions and subsequent retry traffic, which would dramatically reduce the system performance.

For demonstration purposes the system was intentionally congested. That was realized with a modified data flow control system, where the flow control rules are broken. The following modifications were implemented:

- A feeding node does not wait for the completion of an outstanding transfer before it forwards the associated flow control message to the next feeding node. Instead, the message is forwarded immediately after reception. This intentional modification creates conditions for congestions at the receiver, especially in case of a high number of feeding nodes as discussed in section II.F.
- The scheduler was modified with respect to the scheduling discipline. Transfer of multiple subsequent data packets to one and the same destination was allowed. In addition, since a congested system cannot operate at a high rate, the scheduler was internally fed with destinations to achieve a high input data rate.

The number of target retries was measured in a system run at 1.9 MHz. The scheduler was set to send eight subsequent data packets to one and the same column and then switch to the next one. The number of receiving buffers per node was set to 10. The measurement results are shown in Table IV-6. Compared to the frequency of target retries in a properly controlled system, which was measured to be practically zero (Table IV-5), the intentionally congested system shows a relatively intensive retry traffic (Table IV-6) and odd behaviour.

Mean number of PCI target retries in 2nd transmitter per 32 seconds	3536160 ± 20878
Target retries per second	110505

Table IV-6: PCI target retries measured in the feeding node. The system is intentionally congested as described above.

V. Simulation of the Trigger Processing Farm

This chapter deals with simulation studies which have been performed to investigate a large-scale processing farm. The simulation is based on parameters measured in the prototype. The chapter introduces the system with respect to architecture and size and outlines the simulation model and the simulation environment. Then it presents and discusses simulation results regarding latency, efficiency, and scalability of the system.

A. System Architecture

The simulation studies of the trigger processing farm investigate a three-dimensional (3-D) torus (Figure V-1). The available computing power can be augmented by adding computing nodes in both the X and Z directions. Compared to a 2-D system with an equal number of computing nodes, the 3-D architecture results in a more compact system with shorter network paths, i.e. lower network latencies and better scalability.

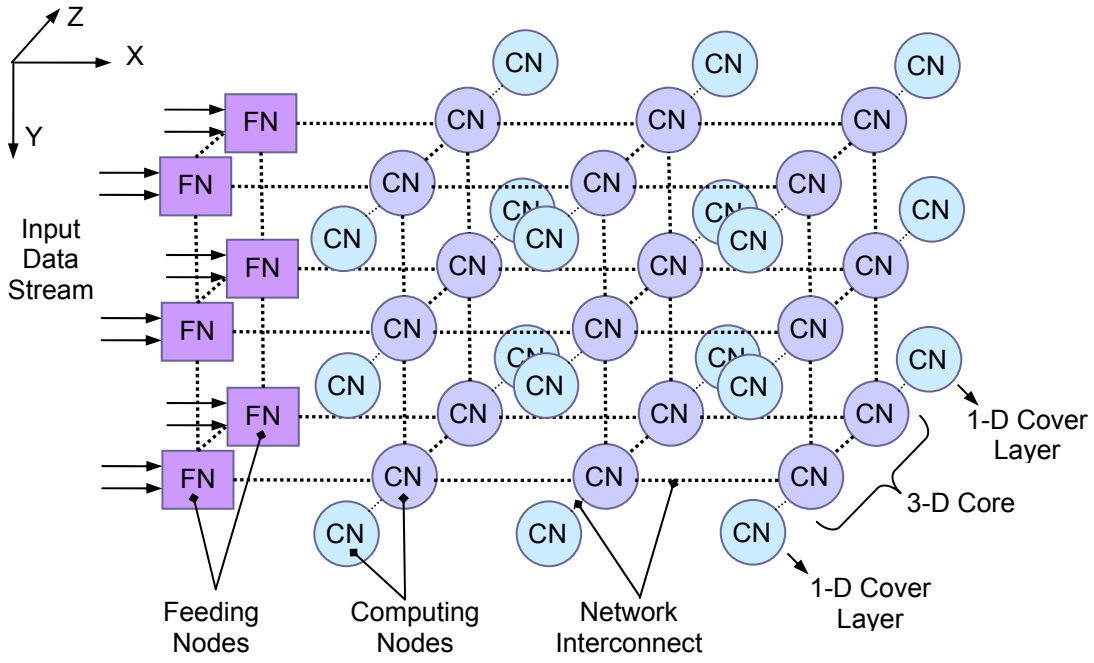


Figure V-1: Adding one-dimensional cover layers to the system considerably increases the computing power available without producing long network paths.

Network packets are first routed along an X-ring, thereafter along a vertical ring, and finally along a Z-ring. Not all nodes, however, are involved in the dimensional routing. Only a limited number of computing nodes, located in the so-called core, are equipped with 3-D network adapters in order to reroute the data packets fed by the feeding nodes. The rest of the nodes are located in the so-called cover layers and are equipped with one-dimensional network adapters, which results in lower cost. In the case of 2-D architecture the augmentation of computing power is achieved by adding whole torus columns, while in a 3-D system it can be done by adding as many computing nodes into the cover layers as needed.

B. Input Data

The trigger processing farm in the simulation is fed by data from the Level-0 Decision Unit (L0DU), the VELO, the TT, and the T1-3 detectors of the LHCb experiment (section II.A). The data file contains approximately 2000 Level-0-yes minimum bias events. The mean amount of data per event is approximately 6.6 kbyte with a tail up to 20 kbyte (Figure V-2).

According to the trigger strategy of the LHCb experiment, the data from the L0DU, the VELO, and the TT are processed for each event. These data have the size of approximately 3.5 kbyte per event. The remaining data that are produced by the T1-3 stations are only transferred for processing on computing node's demand, which occurs for up to 10% of the events.

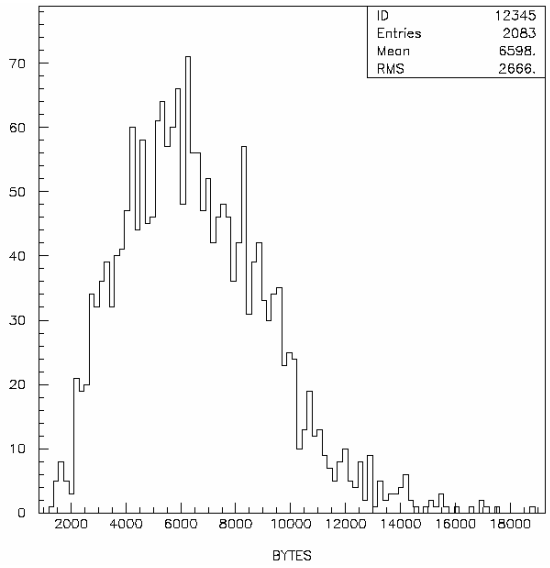


Figure V-2: Total Data Size per Event

C. System Size

Assuming the input data size given in section V.B and the mean input event rate of 1 MHz, and taking into account that the data transfer rate achieved in a feeding node is approximately 275 Mbyte/s (section IV.B), the number of feeding nodes needed is 24. The number of computing nodes required is 275, assuming that each node is equipped with two CPUs and that the average time it takes to execute the trigger algorithm is 500 μs (Figure V-3). Additional 50 μs are foreseen for monitoring and control, which need to be regularly performed on each node.

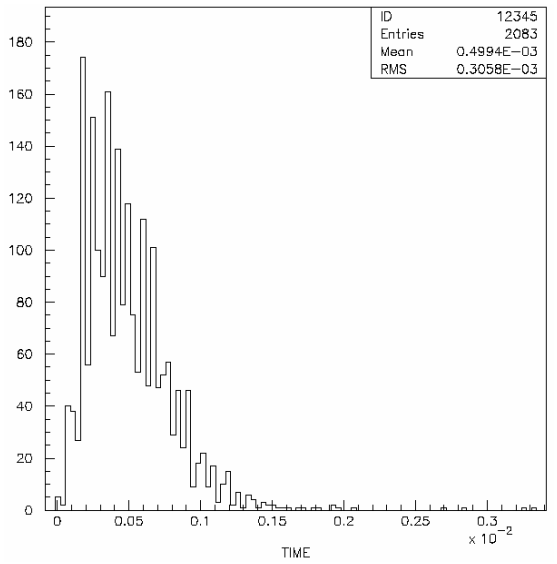


Figure V-3: Event Processing Time Scaled for a Farm of 500 CPUs

Two feeding nodes are placed in a torus row (Figure V-4) for a better network link utilization since the data rate provided by the network interconnect [26] is approximately twice higher than the transmission rate achieved by one data feed. However, if the two feeding nodes are located next to each other, the whole traffic originating from the feeding node located upstream goes through the feeding node located further downstream, which results in very heavy by-pass traffic that prevents the node that is 'willing' to send data from actually transferring them [9]. The two feeding nodes are displaced in order to overcome the heavy by-pass traffic.

The number of rows required is 6, considering that the number of data feeds per torus row is 2, that the architecture implements a core with two layers as discussed in section V.A, and that the total number of feeding nodes needed is 24. The number of columns has to be equal to or greater than the number of rows as discussed in [9]. Hence, the number of computing nodes located in the 6*x*6*x*2 core is 72. The remaining 203 computing nodes need to be located in six cover layers, considering that one cover layer consists of 36 nodes.

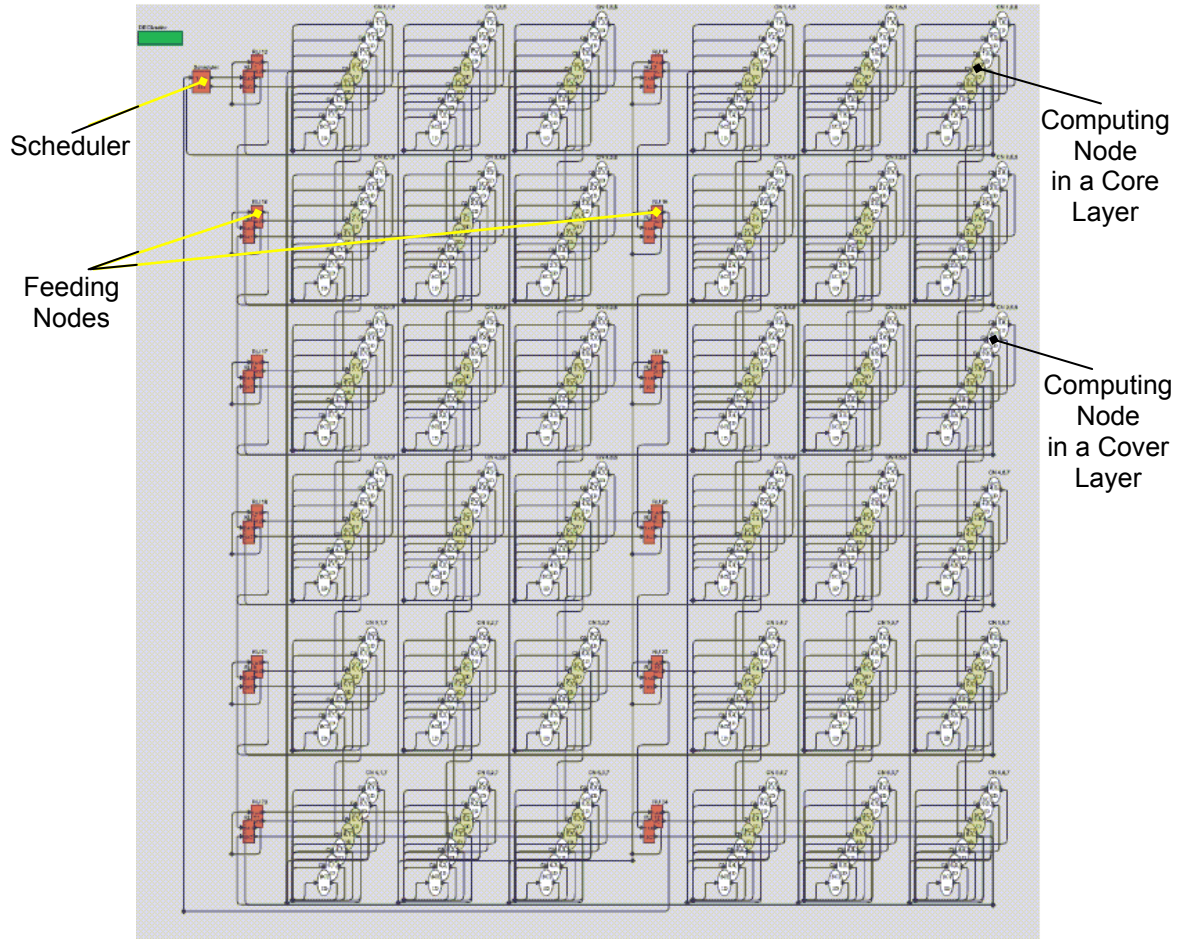


Figure V-4: The simulation model as displayed by the GUI of the Ptolemy II simulation framework. The model consists of 24 feeding nodes and 275 computing nodes located in 2 core layers and 6 cover layers according to the system architecture presented in section V.A.

D. The Simulation Framework

Ptolemy II [75] is a software infrastructure for modeling, simulating, and design of heterogeneous concurrent systems, such as mixed analog-digital electronic devices, hardware-software systems, and electro-mechanical devices, as well as systems that mix widely different operations, such as signal processing, sequential decision making, and user interface. Modeling aims at formally representing a system on the basis of appropriate models of computation. In the particular case of this work, where a model of a real-time processing farm is wanted, the models of computation have to handle concurrency and time since the system consists of multiple simultaneously operating nodes that have multiple concurrent sources of stimuli and work in timed environment. Design is the process of defining a system or subsystem, which involves refining the defined models of the system until the desired functionality is obtained.

In Ptolemy II, a model of a system or subsystem is constructed as a set of components, whose interaction is governed by a set of rules (semantics) imposed by a model of computation. Some of the mostly used Ptolemy II models of computation are implemented by the following domains:

Continuous Time (CT): Components in the CT domain interact via continuous-time signals. The job of the director in the domain is to find a set of continuous-time functions that satisfy all the algebraic or differential relations between inputs and outputs. The domain is useful for modeling physical systems with linear or nonlinear algebraic/differential equation descriptions, such as analog circuits and mechanical systems. In case a system incorporates both analog and digital components interacting with each other (for instance, a sensor connected to a digital electronic recipient), the CT domain interoperates with other Ptolemy domains, such as the discrete-events domain, to realize mixed signal modeling.

Discrete-Events (DE): The components in the DE domain communicate via sequences of events placed along a real time line. An event is composed of a value and time stamp. The model of computation is widely used for describing and simulating digital hardware (realized in languages like VHDL and Verilog).

Finite-State Machines (FSM): The entities in the FSM domain represent state, and the connections represent transitions between states. Execution is a strict sequence of state transitions. The domain is useful for expressing control logic. It can be also used in combination with concurrent domains. Combining the FSM and the CT domain, for instance, yields modal models (models with distinct modes of operation, where behaviour is different in each mode; modal models are used, for instance, to represent physical systems with different regimes of operation, where each regime is represented by a distinct simple model).

Communicating Sequential Processes (CSP): In the CSP domain, concurrently executing processes implemented as Java threads communicate by instantaneous actions called rendezvous (synchronous message passing), where the exchange between the processes is initiated and completed in a single uninterruptable step. Rendezvous models of computation are useful in case of applications characterized with resource sharing like, for example, multitasking or multiplexing of hardware resources.

Process Networks (PN): The PN domain implements asynchronous message passing, i.e. processes communicate by exchanging messages; however, the sender does not need to wait for the receiver to be ready to receive the message (like in the CSP domain case) since the communication channels can buffer the messages.

Some of the major capabilities in Ptolemy II include:

Domain and data polymorphism: Components in Ptolemy II can operate in multiple domains. The way they communicate with each other depends on the domain in which they are used. Furthermore, components can be designed to operate on multiple data types. These two forms of polymorphism maximize reusability.

Higher level concurrent design: Ptolemy II supports modeling and design in Java. However, it provides a number of domains that support concurrent design at a level of abstraction much higher than that supported by Java.

Modularization: The Ptolemy II architecture consists of a set of packages that can be distributed and used independently.

Separation of the abstract syntax from the semantics: Ptolemy II models are represented by means of clustered (hierarchical) graphs. The latter provide a general abstract syntax for component-based modeling, without imposing any semantics (interpretation) on the models. In Ptolemy II, the infrastructure supporting such graphs and the mechanisms that attach semantics to the graphs are separated into distinct packages.

E. The Simulation Model

The functionality of each of the major building blocks of the system (the scheduler, a feeding node, a one-dimensional destination node, and a three-dimensional destination node) is described in an individual file[1]. Another kind of file has been composed to determine the structure of the system together with the interconnections between the system's components. The simulation model has been developed so as to be convenient for system scalability evaluation: the number of rows, columns, and computing nodes is adjustable by input parameters' values.

The model of an SCI node consists of a host sub-block and a network interface sub-block. The host sub-block generates requests and responses at a user-specified rate. It also gathers performance statistics while the simulation is running. The network interface sub-block consists of a node interface part and a processor part. The node interface manages the interface to the SCI ringlet and consists of receiver and arbiter logic and a bypass queue. The processor part contains the feeding and receiving queues and the logic for routing packets and transferring them between the network interface and the host.

In order to achieve reasonable computing efficiency and keep the gains of using a higher-level simulator, an SCI packet is modeled on the symbol[2] field level rather than the bit level. Although all of the fields of a packet are represented, the simulation events concern full packets: a packet is generated and transferred as a whole.

When a packet is queued by the processor in a transmitting queue, an echo time-out is started for it. The packet stays in the queue until being explicitly deleted by one of several mechanisms (one of which is time-out expiration).

The node's access to the output link is controlled by the arbiter sub-block in the network interface block. Packet transmission is initiated if one of the following events occurs:

- there is a packet in the transmitting queue; and the appropriate go-bit conditions are met;

[1] The functionality of system components in the Ptolemy II framework is described in Java.
[2] a 16-bit data quantity; multiple (packet type dependent) symbols build a packet

- a previous transmission has been completed; and the by-pass queue is not empty;
- the node receives a packet that has to be forwarded.

Packet selection and transmission are pipelined, thus the selection of the next packet may start during an ongoing packet transmission. However, another packet is not selected until the transmission of the last packet that was selected begins. A packet is transmitted after a certain processing delay. The simulation includes the physical propagating delay.

Upon packet reception, the receiver sub-block in the network interface determines the packet type and decides whether the packet should be transferred to the host or forwarded. Bypass traffic takes precedence over the transmission of packets originating from the local node. If a packet has to be forwarded although the node is currently transmitting, the packet is sent to the by-pass queue. Packet forward and transfer to the by-pass queue begin after a receiver processing delay but without waiting for the end of the packet to arrive. The transfer of an echo packet begins after the corresponding incoming packet has been received completely.

The processing interface transfers data from the receiving queue to the host as long as such packets are present. A fixed processing delay and a delay dependent on the number of symbols in the packet are introduced.

F. Simulation Results

The simulation of the system shown in Figure V-4 has been performed in order to investigate latency, efficiency, and scalability of such a large-scale processing farm. The parameters and the dependencies that have been investigated are: the event building latency and its dependence on the input event rate, the event size and the system size; the load of the network interconnect; and the CPU utilization.

The event building latency for transfer of 128 byte packets per feeding node (Figure V-5) is characterized with a narrow distribution due to the compact structure of the 3-D system. The event building latency of less than 15 μs is mostly determined by the time it takes to consecutively transfer the 128 byte bursts via the local buses of the 24 feeding nodes. The transfer of a single 128 byte burst to the local network adapter via the PCI bus of a feeding node comprises 1 PCI clock cycle address phase, 6 clock cycles target setup time, and 16 clock cycles data phase (Figure IV-3). However, additional 8 clock cycles for getting the PCI bus granted have to be added, too. Hence, the time it takes for a DMA engine to transfer a 128 byte sub-event to the feeding network adapter via the 66 MHz PCI bus is 465 ns. It is assumed that the flow control network is optimized in order to achieve latencies considerably lower than those measured and presented in Table IV-1.

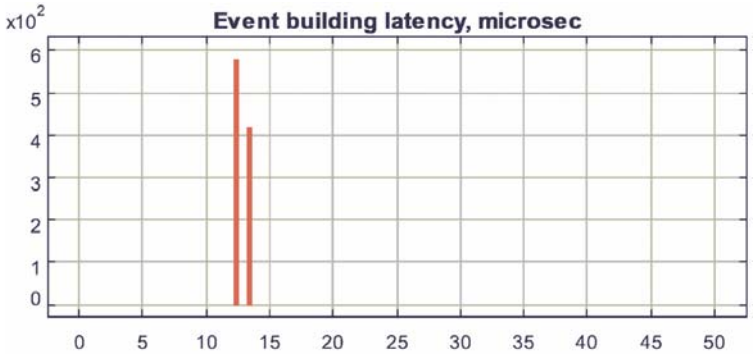


Figure V-5: Event Building Latency in the Case of 128 Byte Packets per Feeding Node

The saturation observed at rates above 2.1 MHz (Figure V-6) is in agreement with the results measured in the test prototype since it was demonstrated in sections IV.B and IV.E that the maximum frequency for transfer of 128 byte packets is limited by the transmitter to up to 2.14 MHz, on average. At that frequency, the data rate on the PCI bus of the feeding node approaches the maximum available PCI-SCI bandwidth of approximately 300 Mbyte/s.

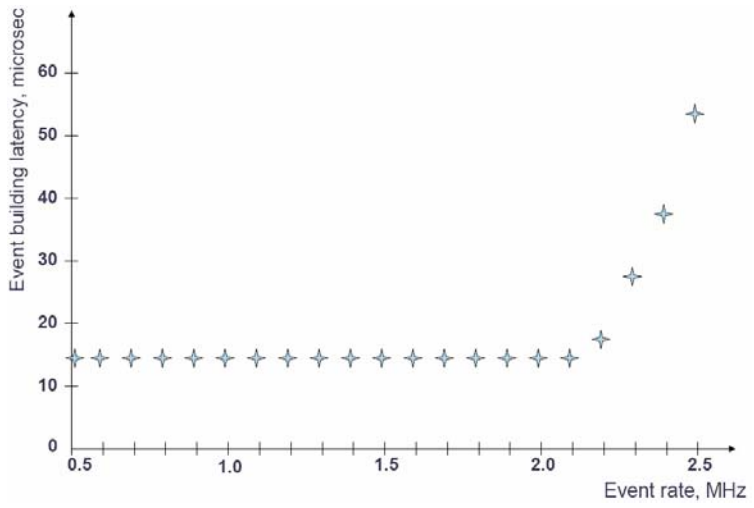


Figure V-6: Event Building Latency vs. Input Event Rate in the Case of 128 Byte Packets per Feeding Node

The most efficient way to transfer data in the test setup is using 128 byte packets per feeding node. In that case the transfer is completed using single transactions with a minimum overhead involved: the DMA engine in the feeding node transfers the data to the local network adapter using a single burst; thereafter, the local network adapter transmits the data to the remote network adapter using a single network packet (Figure II-7). If the DMA engine has to send a data packet larger than 128 byte, however, at least one additional burst has to be initiated since the local network adapter

issues a target disconnect upon reception of 128 bytes. The initiation of a new transaction involves bus arbitration and increases the latency. Furthermore, at least one additional network packet has to be sent to the remote network adapter since the biggest packet supported by the interconnect can carry up to 128 byte of data [26]. The average sub-event size for transfer of data originating from the L0DU, the VELO, and the TT is in excess of 128 byte. Hence, two PCI bursts and two SCI packets are needed to transfer a sub-event. Consequently, the average event building latency for data produced by the L0DU, the VELO, and the TT (Figure V-7) is twice larger than in the previous case of 128 bytes per sub-event. The latency distribution reflects the distribution of the input data.

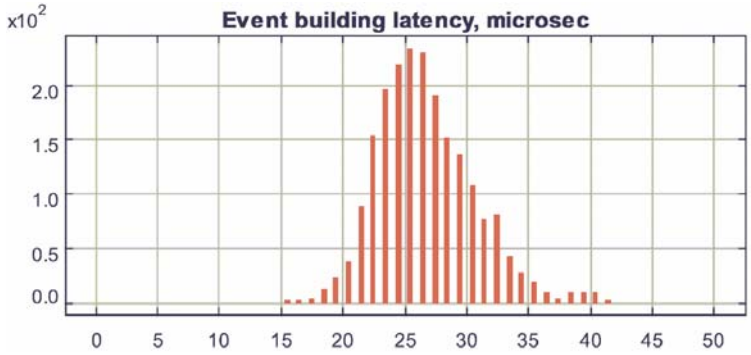


Figure V-7: Event Building Latency in the Case of L0DU, VELO, and TT Data at the Input Event Rate of 1 MHz

The trigger processing farm has also been simulated using linearly scaled data that have the same distribution like the data produced by the L0DU, the VELO, and the TT. The obtained results (Figure V-8) show that up to 30% more data can be transported to the computing nodes without breaking the latency requirements. Data from the T1-3 need to be transferred at the rate of 50-100 kHz [76], which corresponds to a scaling factor of 1.1 in Figure V-8.

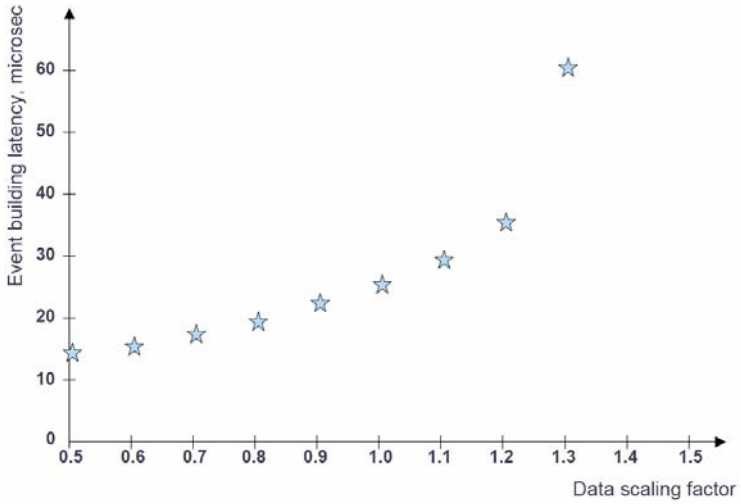


Figure V-8: Event Building Latency in the Case of Scaled L0DU, VELO, and TT Data at the Input Event Rate of 1 MHz

A second part appears in the distribution shown in Figure V-7 when the data originating from the T1-3 stations are transported on computing node's demand. That part can be seen as a tail in Figure V-9. The full event building latency is twice larger in case of long events since the amount of the T1-3 data is approximately equal to the amount of the data produced by the L0DU, the VELO, and the TT detector.

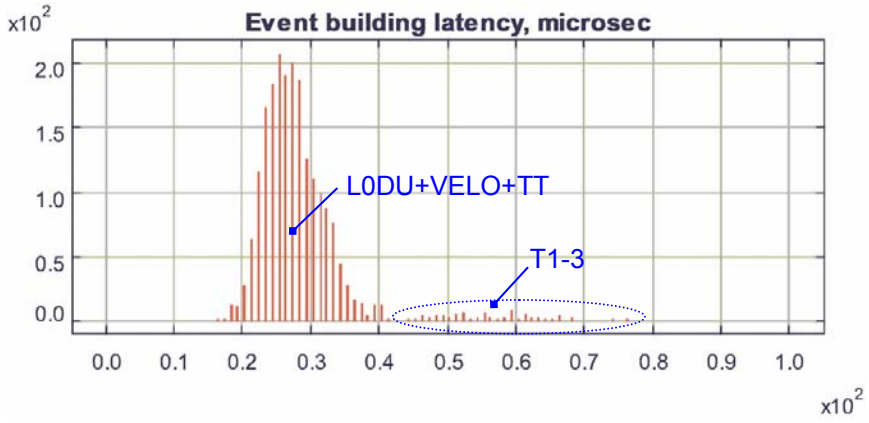
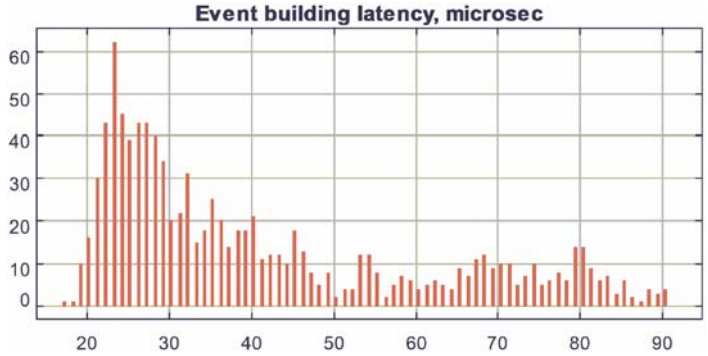


Figure V-9: Full event building latency at the input event rate of 1 MHz. The main part of the distribution corresponds to the transport of the data originating from the L0DU, the VELO, and the TT. The data produced by the T1-3 stations are transferred for 5% of all events. This secondary transfer results in twice larger full event building latency, which is seen in the tail of the distribution.

The flow control network developed (section III.D) is a ring with a unidirectional data flow. When a feeding node has finished its transfer, it forwards the flow control message to the feeding node located next after it. Thus, the full event building latency may considerably increase in case a feeding node is still busy sending a previous large sub-event while the flow control message associated with the new event

has already arrived. Therefore, the ability to bypass such a busy feeding node is foreseen. In that case, the flow control message has to be sent to the feeding nodes for a second time. The simulations show that the bypass mode results in a lower full event building latency (Figure V-10) and makes the system more robust against large events.

a)



b)

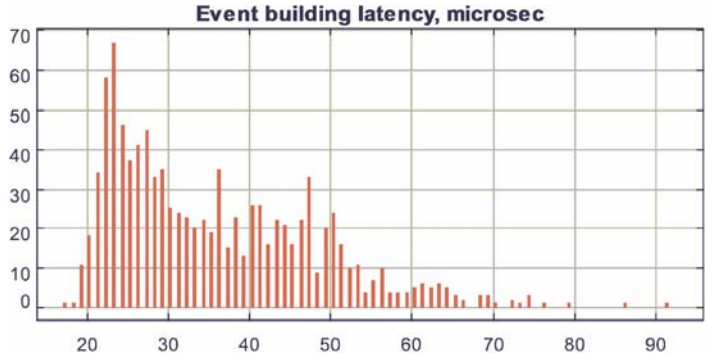


Figure V-10: Event building latency when the bypass of a feeding node is (a) disabled and (b) allowed. A few of the events in these simulations are intentionally increased in size by a factor of 5.

One of the advantages of the 3-D architecture is that the computing power can be considerably increased without producing long network packet paths. Therefore, the full event building latency depends slightly on the number of computing nodes in the system (Figure V-11). Adding two cover layers and one column to the system shown in Figure V-4, for instance, increases the available computing power by approximately 60%; however, the latency of a packet routed via the longest network path is increased by approximately 200 nanoseconds, which cannot be seen in Figure V-11. The number of processors used in a system built of 550 CPUs is shown in Figure V-12.

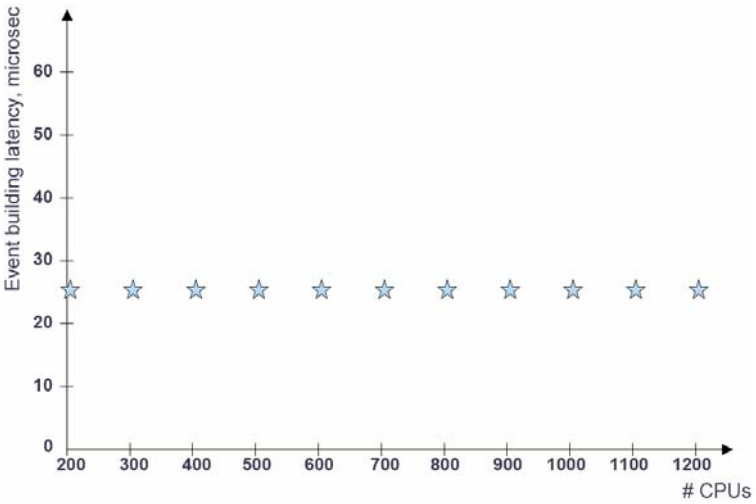


Figure V-11: Full Event Building Latency at the Input Event Rate of 1 MHz vs. Number of CPUs Available in the 3-D Processing Farm

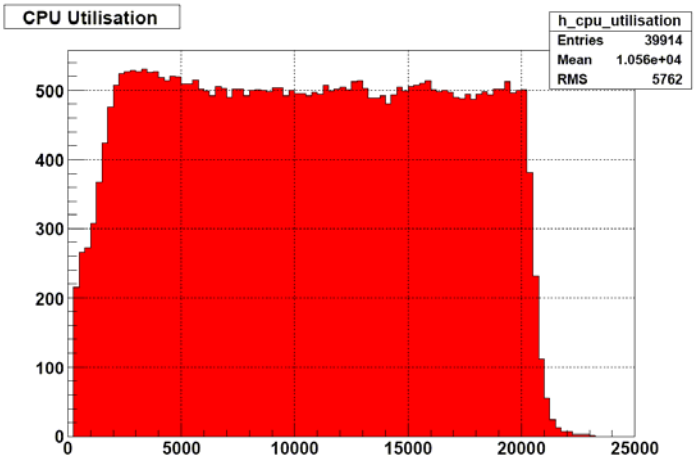


Figure V-12: CPU Utilization in the Case of a System Comprising 550 CPUs with a Mean Event Processing Time of 500 µs

VI. Conclusion

The processing farm developed in [9] and briefly presented in chapter II is applicable to high-rate real-time trigger processing only if provided with appropriate measures for traffic shaping so as to keep the senders from overrunning the receivers, to prevent congestions in overloaded intermediate network nodes and to balance the load of the computers. Flow control issued by the receiver and congestion control realized by routing and bypassing nodes limit the flow from the nodes causing the overload, which makes these approaches inapplicable in the current consideration since, if issued, such kinds of control can rapidly congest the entire network. Therefore, this thesis is concerned with a mechanism which schedules the traffic when packets enter the network so as to avoid congestions by appropriately allocating the available network resources.

The mechanism of resource allocation addresses the problem from the edge of the network by controlling the senders so as to orchestrate the actual data fragment transfers. [9] determines the maximum system frequency in a prototype with a basic scheduling network which transfers a single-bit message to successively initiate the transmission of each one of the fragments corresponding to an event, thereby controlling the arrival rate of the receiver. The handshaking mechanism between adjacent senders, implemented to prevent loss of messages, adds approximately 40% overhead to the sub-event transfer time and thus causes considerable inefficiency, which limits the maximum system frequency. Therefore, the current work extends the scheduling network by adding pipelined functionality, which allows each feeding node to send sub-events back to back and, together with additional optimization of the DMA logic in the sender, improves the maximum system frequency measured in the prototype by 55%.

Since event processing times vary, dispatching events to computers in a fixed order is inapplicable. The scheduling network is, therefore, further extended by adding a scheduling unit which allocates computing nodes dynamically. While introducing minimal decision latency, the scheduler selects the next target node so as to balance the load of the computing resources as well as to avoid network congestions during event building. The scheduling discipline rules out dispatching of subsequent events to computers residing in one and the same torus column. Simulations showed that this simple rule, in conjunction with the mode of operation of the scheduling network and the ability of the main network to buffer a certain amount of data, is sufficient to avoid overrunning the receiver and congesting the routing nodes.

In the prototype, the next target column is selected on a rotating basis, thus giving equal priority to all of the columns. In reality, however, the algorithm applied has to consider the fact that processing power may be unequally distributed and has to allow the columns with higher processing power to get a greater share of the workload than the other ones in order to achieve efficient computing resource utilization.

Considering the megahertz system input rate and the scalability requirements, it can be argued that configurable hardware is most suitable for the realization of the traffic shaping system components. Consequently, a significant part of the current work was the development of a multipurpose hardware unit based on FPGA. Moreover, the board developed is a versatile prototyping platform which was designed to provide features required during the research and development phase of a number of additional high-energy physics applications, among which are: the second prototype of the ALICE

TPC RCU Unit, the first prototype of the ALICE HLT RORC Card and the first prototype of an independent hardware agent for automated remote management of PCs.

The processing farm considered in this work is characterized by highly intensive data transfers performed by a number of feeding nodes. The analysis of the traffic on the expansion bus in any of these nodes in the prototype shows that a single data transfer comprises 23 bus clock cycles. Thus, a single clock cycle more already adds approximately 5% overhead to the transaction. It was, therefore, crucial for the high-transaction-rate processor to design the logic in the DMA modules as well as in the scheduler so as to induce minimal latency and thus to achieve higher bandwidth utilization. Nevertheless, it is determined that the maximum network throughput achieved in the prototype is limited to approximately 52% of the theoretical maximum input bandwidth (assuming that a DMA module feeds the data into the network through a 64-bit 66 MHz I/O address-data bus in a burst comprising 1 cycle address phase and 16 cycles data phase).

Network congestions can be the reason for inefficient system utilization since they result in packet retransmission which increases the transfer time and thereby reduces the network throughput. It can be shown that the number of transaction retries signaled by the network adapter on the expansion bus in a feeding node reflects the rate of network congestions. This work determines the number of target retries on the bus of the sender and demonstrates that the network throughput in the prototype with an integrated traffic shaping system is already limited by the transmitting side but not by the receiving or intermediate network nodes.

Detailed analyses of the bus traffic in the senders show that approximately 22% of the bus bandwidth are used for transaction overhead, mostly caused by the target setup time in the deployed network adapters. The remaining 26% of the total available input bandwidth are, however, unutilized as the measurement of the bus idle time shows. It is, therefore, crucial for obtaining high system throughput to achieve higher bus utilization in the senders.

The scheduling unit and the DMA logic are capable of initiating new transfers practically back to back. It is shown that the DMA logic remains idle for a single clock cycle in-between two transfers. A substantial part of the bus idle time is caused by the local bus interface deployed in the DMA modules which induces additional latency before requesting the bus. Thus, the bandwidth utilization can be improved by modifying the way the expansion bus of the sender is interfaced. Another approach, the feasibility of which has been investigated in [9], consists in setting two DMA modules in separate bus agents working in interleaved mode.

The viability of the computing farm for high-rate real-time trigger processing is demonstrated by means of simulation, too. The simulation focuses on investigating latency, efficiency, and scalability of a large-scale processing farm and is based on parameters measured in the prototype.

The simulation results show that the event building latency is mostly determined by the time it takes to consecutively transmit all sub-events by the feeding nodes, which demonstrates the congestion-free operation of a system comprising multiple feeding nodes. The simulation model, however, assumes that the scheduling network is optimized with respect to latency so that the transfers of the sub-events, corresponding to one and the same event, are initiated almost back to back by each one of the feeding nodes and reach the receiver within a minimum time without overlapping.

The simulation studies examine a three-dimensional torus. The 3-D architecture results in a more compact system compared to a 2-D system with an equal number of nodes. This results in lower network latencies and better scalability, which is

demonstrated by the narrow distribution of the event building latency and its negligible dependence on the number of processing nodes.

VII. Appendices

A. Setting up the System

In order to run the system a strict procedure has to be followed. This paragraph provides information about what should be observed and what has to be performed in order to set up and run the system. Basic knowledge of Linux and familiarity with Altera devices [77] and Dolphin's configuration and diagnostic tools [78, 79] is assumed. The procedure has hardware and software aspects.

Dealing with the hardware aspect, one has to make sure that:

- The SCI set-up is proper. This includes:

1) The SCI adapters are properly installed [80];
2) The SCI adapters are properly configured [78];
3) The SCI cabling implements the desired topology. The easiest way to ascertain the SCI set-up is proper is through the diagnostic tool described in [79];

- The flow control system set-up is proper. This includes:

1) The CIA-RORC cards in the feeding nodes are properly installed and configured[1]. See also section III.B.8 and section III.E.5. The nodes need to be rebooted after a programmable logic device in the DMA engine domain has been reconfigured/ reprogrammed;
2) The flow control network cabling implements the desired topology.

Dealing with the software aspect, the user has to run certain software processes in a certain sequence as follows:

1. Run the process `sch` (as a root) on the node where the scheduler is installed. This will export a local memory region needed for receiving feedback messages as explained in section III.B.6. The process can be run through the following command line:

   ```
   sch –clients <number of clients>
   ```

   - The flag `clients` specifies the number of receiving nodes, which is 12 in the test set-up as shown in Figure IV-1.

2. Run the process `one2many` as a root on each receiving node. This will: a) import the memory region exported by the scheduler in the previous step to establish a connection for the transfer of feedback messages between the receiving node and the scheduler; b) export a local memory region which is needed to receive data from the feeding nodes. The process can be run through the following command line:

[1] Note that there are three programmable logic devices in the feeding nodes: there are the main FPGA and the auxiliary CPLD on the CIA-RORC board, and the FPGA on the flow-control network interface card. They all have to be configured/programmed appropriately. The programming files for these devices are available on the attached CD-ROM. The programming file for the main FPGA has to be reproduced to fit the actual system size.

```
one2many -c <SERVER_NODEID> -e
<EVALUATIONSERVER_NODEID> -h <SCHEDULER_NODEID> -a
<COLUMN_WIDTH> -d <ROW_WIDTH> -f <RX_BUFF_WIDTH> -x
<COL_POSITION> -y <ROW_POSITION> -i <MY_NODEID>
```

- The program `one2many` is designed in a way that a node can be run either as a receiving node or as a feeding node. One of the feeding nodes has to be labelled as a server node, whose SCI logical node ID `SERVER_NODEID` is specified here by the `c` flag.
- The SCI logical node ID `SCHEDULER_NODEID` of the scheduler's node is specified by the `h` flag.
- The value of `EVALUATIONSERVER_NODEID` should be equal to the value of `SCHEDULER_NODEID` (an evaluation server is not used in the current set-up; it was used in the work presented in [9]).
- The values of `COLUMN_WIDTH`, `ROW_WIDTH`, and `RX_BUFF_WIDTH` have to be equal to the parameters of the same name presented in section VII.B.
- The *x* and *y* coordinates of the receiving node in the 2-D torus have to be specified by `COL_POSITION` and `ROW_POSITION` correspondingly. See Figure VII-1 and Table VII-1, for instance.
- The value of `MY_NODEID` passed through the `i` flag specifies the slot(s) in the descriptor buffer occupied by the receiving node (refer to section III.E.1.a) and Table VII-1).

The script `clients_sch_root_2by8`, which is available on the attached CD-ROM, has been used to execute step 2.

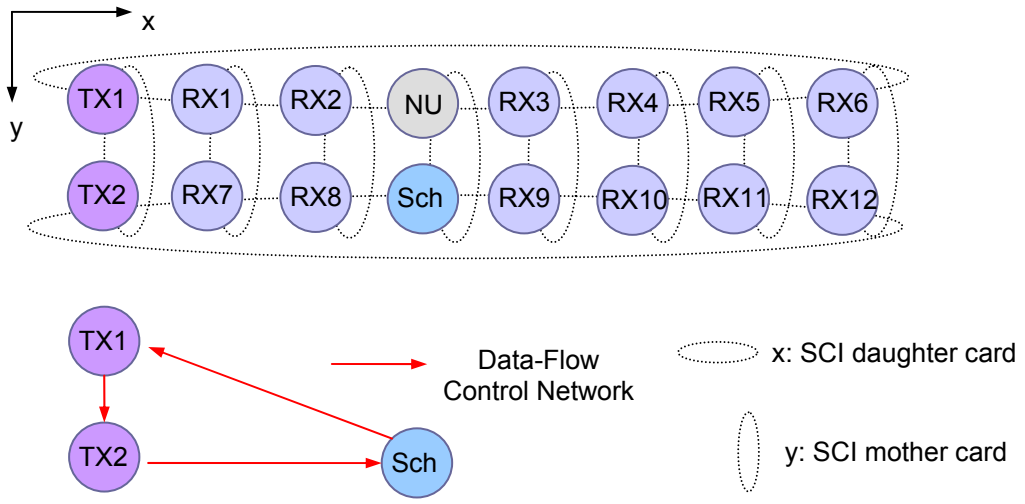


Figure VII-1: The Test Setup; A Figure to Table VII-1

Node Alias in Figure VII-1	SCI logical ID	COL_POSITION	ROW_POSITION	MY_NODEID
TX1 (server node)	4	-	-	-
TX2	8	-	-	-
RX1	68	1	1	0
RX2	132	2	1	1
RX3	260	3	1	2
RX4	324	4	1	3
RX5	388	5	1	4
RX6	452	6	1	5
RX7	72	1	2	6
RX8	136	2	2	7
RX9	264	3	2	8
RX10	328	4	2	9
RX11	392	5	2	10
RX12	456	6	2	11
Sch (scheduler)	200	-	-	-
NU (passive)	196	-	-	-

Table VII-1: Actual Settings in the Test Setup

3. Run the process one2many on each feeding node in order to import all memory regions exported by the receiving nodes in the previous step to establish the connection for the transfer of data between the feeding nodes and the receiving nodes.

   Use the following command line to run the process on each feeding node except the server node:

   ```
   one2many –c <SERVER_NODEID>
   ```

- The value passed through the `c` flag must be equal to `SERVER_NODEID` in step 2.

  Use the following command line to run the process on the server node:

  ```
  one2many –r <RX NodeIDs> -p <TX NodeIDs> -s <RX buffer size> -b <burst size> -n <number of RX buffers per node>
  ```

- The SCI logical IDs of all receiving nodes following the `r` flag are separated by space. The settings used in the test set-up are available in Table VII-1.
- The SCI logical IDs of all feeding nodes except the server node have to be specified through the `p` flag, exactly like the node IDs of the receiving nodes.
- The `s` flag is used to specify the size of an RX buffer that should be allocated by a receiving node. The value is in byte; however, specifying 4k instead of 4096 or 1M instead of 1048576, for instance, is valid too. The size of the receiving buffer actually used in the test set-up is 4096 byte.
- The `b` flag specifies the size of data that are transferred by a feeding node in a PCI burst. Setting the value of this flag to 128 means a burst size of 128 byte, which was used in this work.
- The `n` flag specifies the number of receiving buffers per receiving node. In fact, all nodes are configured by the server node remotely in this last step of the initialization phase.

After the successful execution of steps 1 to 3, the system is initialized and can be run in the following way:

- Set system trigger through scheduler **CSR 2**. A description of all supported CSR in the scheduler is available in section VII.C.
- Enable the scheduler through **CSR 1**.
- Check the state of the scheduler's main state machine through **CSR 4** in order to ascertain that the scheduler is ready.
- Enable the system trigger through **CSR 3**.
- The instantaneous number of processed input requests (events) can be obtained through **CSR 9**.

Information regarding user access to the scheduler and the DMA engine domain in the feeding nodes is available in sections III.B.9 and III.E.5.

A software configuration and monitoring tool dedicated to the scheduler is not available at the time of writing. Access to CSR was gained by using the tool `set_mem`[1] available on the attached CD-ROM. The tool allows the initiation of single cycle PCI memory transactions.

[1] Property of the Chair of Computer Science at KIP, University of Heidelberg

B. Scheduler Design Parameters

The following list presents all scheduler design parameters, their description, and the range of valid values. The list is divided into two parts: firstly, parameters that are typically edited by the user are listed; secondly, one can find all parameters that are modified, mostly by the designer. To modify a parameter value, edit the corresponding entry in sch_pack.vhd.

COLUMN_NUM

This is the number of columns in the system, considering a 2-D torus topology, excluding the feeding nodes. COLUMN_NUM is assigned integer values. Modify COLUMN_WIDTH whenever you modify COLUMN_NUM.

COLUMN_WIDTH

COLUMN_WIDTH must fulfill the following requirement:

$$\text{COLUMN\_WIDTH} \geq \log_2 \text{COLUMN\_NUM}$$

COLUMN_WIDTH is assigned an integer value.

ROW_WIDTH

ROW_WIDTH must fulfill the following requirement:

$$\text{ROW\_WIDTH} \geq \log_2 \text{ROW\_NUM},$$

where ROW_NUM is the number of rows in the system, considering a 2-D torus topology.

RX_BUFF_WIDTH

This is an integer that must fulfill the following requirement:

$$\text{RX\_BUFF\_WIDTH} \geq \log_2 \text{RX\_BUFF\_NUM},$$

where **RX_BUFF_NUM** is the number of receiving buffers in a computing node. The default number of receiving buffers per computing node is 2.

DESC_BUFF_WIDTH

This is an integer that specifies the width of the address bus of the descriptor buffer in the DMA engine domain (section III.E.1.a)). DESC_BUFF_WIDTH must meet the following requirement:

$$\text{DESC\_BUFF\_WIDTH} \geq \log_2(\text{COLUMN\_NUM}.\text{ROW\_NUM}.\text{RX\_BUFF\_NUM}),$$

where COLUMN_NUM is the number of columns in the system considering a 2-D torus topology excluding the feeding nodes; ROW_NUM is the number of rows in the system considering a 2-D torus topology; and RX_BUFF_NUM is the number of receiving buffers per computing node.

The value of this parameter must coincide with the value in the design of the DMA engine (section VII.D).

The user is not supposed to edit the following parameters. The designer should modify their values when needed.

UD_USE_DUAL_CLK_FIFO

UD_USE_DUAL_CLK_FIFO must be assigned a string that is either 'YES' or 'NO'. The assignment 'YES' implements dual clock FIFOs for the storage of queued destinations (see section III.B.4). The default value is 'NO' since the current architecture implements another buffer in front of the FIFOs for queued destinations. That buffer is the Input Buffer, where clock synchronization is done (see UIB_USE_DUAL_CLK_IN_BUFF_FIFO).

The ability to choose between dual-clock and single-clock FIFOs for the storage of queued destinations may be considered redundant. However, it is designed to allow using the unit in other applications where dual-clock FIFOs may be required, as well. It also predicts the architecture of the scheduler where the Input Buffer is not used.

UD_RESERVED

This is an integer parameter that allows the extension of UD_DATA_WIDTH with UD_RESERVED bits. UD_RESERVED adds redundancy that may be useful in future.

UD_DATA_WIDTH

This is the width of a word in a FIFO for the storage of queued destinations. It is defined as the sum of RX_BUFF_WIDTH, ROW_WIDTH, COLUMN_WIDTH, and UD_RESERVED. The value of UD_DATA_WIDTH is not intended to be modified directly.

UIB_USE_DUAL_CLK_IN_BUFF_FIFO

UIB_USE_DUAL_CLK_IN_BUFF_FIFO must be assigned a string that is either 'YES' or 'NO'. Assigning 'YES' implements the Input Buffer (section III.B.4) as a dual-clock FIFO. This option was used due to the necessity of clock synchronization between two clock domains: the PCI system clock domain and the scheduler system clock domain. The string 'NO' should not be assigned to UIB_USE_DUAL_CLK_IN_BUFF_FIFO unless another method that ensures clock synchronization between the PCI clock domain and the scheduler clock domain is realized.

UIB_IN_BUFF_FIFO_WIDTH

This is the width of a word in the Input Buffer FIFO (section III.B.4). UIB_IN_BUFF_FIFO_WIDTH is assigned 32, which corresponds to the data width of a 32-bit PCI transaction. UIB_IN_BUFF_FIFO_WIDTH must fulfill the following requirement: UD_DATA_WIDTH ≤ UIB_IN_BUFF_FIFO_WIDTH ≤ 64.

UIB_IN_BUFF_FIFO_WIDTHU

The following equation is true:

$$2^{\texttt{UIB\_IN\_BUFF\_FIFO\_WIDTHU}} = \texttt{UIB\_IN\_BUFF\_FIFO\_WORD\_NUM},$$

where UIB_IN_BUFF_FIFO_WORD_NUM is the number of words which can be stored in the Input Buffer.
The following requirement must be met:

$$\texttt{UIB\_IN\_BUFF\_FIFO\_WIDTHU} \geq$$
$$\geq \log_2(\texttt{COLUMN\_NUM}.\texttt{ROW\_NUM}.\texttt{RX\_BUFF\_NUM})'$$

where COLUMN_NUM is the number of columns in the system considering a 2-D torus topology excluding the feeding nodes. ROW_NUM is the number of rows in the system considering the 2-D torus topology; and RX_BUFF_NUM is the number of receiving buffers per computing node.

UIB_IN_BUFF_FIFO_SLACK

Overflow in the Input Buffer FIFO (section III.B.4) is signaled when:

$$\text{in\_buff\_fifo\_usedw} \geq 2^{\texttt{UIB\_IN\_BUFF\_FIFO\_WIDTHU}} - \texttt{UIB\_IN\_BUFF\_FIFO\_SLACK},$$

where **in_buff_fifo_usedw** is the instantaneous number of words used in the Input Buffer FIFO (see **CSR 12** in section VII.C); and $2^{\texttt{UIB\_IN\_BUFF\_FIFO\_WIDTHU}}$ is the maximum number of words that can be stored in the FIFO. UIB_IN_BUFF_FIFO_SLACK is the number of words that should always be reserved in the Input Buffer FIFO to avoid overflow and loss of data.

UE_USE_DUAL_CLK_EV_BUFF_FIFO

UE_USE_DUAL_CLK_EV_BUFF_FIFO must be assigned a string that is either 'YES' or 'NO'. The assignment 'YES' implements the buffer of event IDs as a dual clock FIFO. This option has been implemented in the past and is obsolete since the current architecture implements custom synchronization cells in front of the buffer of event IDs to synchronize all incoming signals with the scheduler's system clock. Assign 'NO' to UE_USE_DUAL_CLK_EV_BUFF_FIFO for proper system operation.

UE_EV_ID_WIDTH

This is an integer that specifies the data width of the buffer of event IDs. The value of UE_EV_ID_WIDTH corresponds to the width of the Event ID number; the latter identifies the incoming event data packets. UE_EV_ID_WIDTH is set to 32 to match the 32-bit width of Event IDs entered via PCI.

UE_EV_BUFF_FIFO_WIDTHU

The following equation is true:

$$2^{\texttt{UE\_EV\_BUFF\_FIFO\_WIDTHU}} = \texttt{UE\_EV\_BUFF\_FIFO\_WORD\_NUM},$$

where UE_EV_BUFF_FIFO_WORD_NUM is the number of words which can be stored in the buffer of event IDs.

UE_EV_BUFF_FIFO_SLACK

Overflow in the buffer of event IDs is signaled when:

$$\text{ev\_buff\_fifo\_usedw} \geq 2^{\text{UE\_EV\_BUFF\_FIFO\_WIDTHU}} - \text{UE\_EV\_BUFF\_FIFO\_SLACK},$$

where **ev_buff_fifo_usedw** is the instantaneous number of words used in the buffer of event IDs (see **CSR 16** in section VII.C), and $2^{\text{UE\_EV\_BUFF\_FIFO\_WIDTHU}}$ is the maximum number of words that can be stored in the buffer. UE_EV_BUFF_FIFO_SLACK is the number of words that should always be reserved in the buffer of event IDs to avoid overflow and loss of data.

US_SRC_NUM

US_SRC_NUM equals the number of TX buffers per feeding node.

US_SRC_ADDR_WIDTH

This is an integer that must fulfill the following requirement:

$$\text{US\_SRC\_ADDR\_WIDTH} \geq \log_2 \text{US\_SRC\_NUM},$$

where US_SRC_NUM is the number of data buffers per feeding node.

UA_INCRMNT_CNT_WIDTH

This is an integer value that specifies the width of a monitoring counter that is incremented every time the scheduler algorithm state machine enters the 'Increment' state as shown in the state diagram in Figure III-6. The value of the counter can be read out via **CSR 10** (section VII.C). The value of UA_INCRMNT_CNT_WIDTH must be less than or equal to 32.

TAG_FRAGM_NUM

Every flow control message is transferred in 16-bit wide fragments. The value of TAG_FRAGM_NUM is an integer that specifies the number of fragments per flow control message. In the current design, TAG_FRAGM_NUM must be set to 2. The format of a flow control message is shown in Figure III-38. The value of TAG_FRAGM_NUM must coincide with the value in the design of the DMA engine (section VII.D).

TAG_FRAGM_WIDTH

This is an integer that must meet the following requirement:

$$\text{TAG\_FRAGM\_WIDTH} \geq \log_2 \text{TAG\_FRAGM\_NUM},$$

where TAG_FRAGM_NUM is the number of fragments per flow control message. The value of TAG_FRAGM_WIDTH must coincide with the value in the design of the DMA engine (section VII.D).

UTN_IN_REG_USED

UTN_IN_REG_USED must be assigned a string that is either 'YES' or 'NO'. The assignment 'YES' implements input registers in the receiving lines of the flow control network interface unit between the FPGA on the transceiver card (Figure III-26) and the host FPGA on the CIA-RORC board (Figure III-22). The registers are placed in the I/O cells and improve the performance of data transfer between the plug-in card and the CIA-RORC card. UTN_IN_REG_USED is set to 'YES' in the test setup. The value of this parameter must coincide with the value in the design of the DMA engine (section VII.D).

UTN_OUT_REG_USED

This parameter has the same meaning as UTN_IN_REG_USED. However, it refers to the output lines of the flow control network interface. The value of UTN_OUT_REG_USED must coincide with the value in the design of the DMA engine (section VII.D).

UTN_RX_FIFO_WIDTHU

The following equation is true:

$$2^{\mathrm{UTN\_RX\_FIFO\_WIDTHU}} = \mathrm{UTN\_FIFO\_WORD\_NUM},$$

where UTN_FIFO_WORD_NUM is the number of words which can be stored in the TX FIFO or RX FIFO of the flow control network interface (Figure III-12).

In the prototype, the value of UTN_RX_FIFO_WIDTHU is set in a way that all control messages flowing through the system can be stored in the output or input FIFO. Note that the storage of a single flow control message in the output and input buffers of the flow control network interface unit requires TAG_FRAGM_NUM words, with TAG_FRAGM_NUM being the number of flow control message fragments. The current implementation requires two words per message. Therefore, UTN_RX_FIFO_WIDTHU meets the following requirement:

$$\mathrm{UTN\_RX\_FIFO\_WIDTHU} \geq \log_2\left(2.\mathrm{COLUMN\_NUM}.\mathrm{ROW\_NUM}.\mathrm{RX\_BUFF\_NUM}\right),$$

where COLUMN_NUM is the number of columns in the system, considering a 2-D torus topology and excluding the feeding nodes; ROW_NUM is the number of rows in the system considering a 2-D torus topology; and RX_BUFF_NUM is the number of receiving buffers per computing node. The value of UTN_RX_FIFO_WIDTHU must coincide with the value in the design of the DMA engine (section VII.D).

TAG_HEADER_TYPE1

Every flow control message consists of TAG_FRAGM_NUM fragments, with the first fragment being the header of the message (Figure III-38). The current implementation supports one header type that is set to 0xCCCC through the parameter TAG_HEADER_TYPE1.

UTN_BUFF_FIFO_SLACK

Overflow in any of the FIFOs[1] in the flow control network interface unit (Figure III-12) is signalled when:

$$\text{utn\_fifo\_usedw} \geq 2^{\text{UTN\_BUFF\_FIFO\_WIDTHU}} - \text{UTN\_BUFF\_FIFO\_SLACK},$$

where utn_fifo_usedw and UTN_BUFF_FIFO_WIDTHU have the following meanings:

- For the output buffer: utn_fifo_usedw is the instantaneous number of words used in the output buffer of the flow control network interface unit (see **CSR 13** in section VII.C); and UTN_BUFF_FIFO_WIDTHU represents UTN_RX_FIFO_WIDTHU so that the term $2^{\text{UTN\_BUFF\_FIFO\_WIDTHU}}$ is the maximum number of words that can be stored in the output buffer of the flow control network interface unit;
- For the input buffer: utn_fifo_usedw is the instantaneous number of words used in the input buffer of the flow control network interface unit (see **CSR 15** in section VII.C), and UTN_BUFF_FIFO_WIDTHU represents UTN_RX_FIFO_WIDTHU so that the term $2^{\text{UTN\_BUFF\_FIFO\_WIDTHU}}$ is the maximum number of words that can be stored in the input buffer of the flow control network interface unit;
- For the check buffer: utn_fifo_usedw is the instantaneous number of words used in the check buffer of the flow control network interface unit (see **CSR 14** in section VII.C); UTN_BUFF_FIFO_WIDTHU represents UTN_CHECK_BUFF_WIDTHU so that the term $2^{\text{UTN\_BUFF\_FIFO\_WIDTHU}}$ is the maximum number of words that can be stored in the check buffer of the flow control network interface unit.

UTN_ERR_CNT_WIDTH

This is an integer that specifies the width of a counter that counts the number of wrong flow control messages received back at the scheduler. The value of this error counter can be read out via **CSR 6** (section VII.C). UTN_ERR_CNT_WIDTH must be assigned values that are less than or equal to 32.

UTN_CHECK_BUFF_WIDTHU

The following equation is true:

$$2^{\text{UTN\_CHECK\_BUFF\_WIDTHU}} = \text{UTN\_CHECK\_BUFF\_WORD\_NUM},$$

where UTN_CHECK_BUFF_WORD_NUM is the number of words which can be stored in the check buffer of the flow control network interface unit (Figure III-12).

In the prototype, the value of UTN_CHECK_BUFF_WIDTHU is chosen so that the total number of flow control messages can be stored in the check buffer. Storage of a single message in the check buffer requires a single word. Therefore, UTN_RX_FIFO_WIDTHU meets the following requirement:

$$\text{UTN\_CHECK\_BUFF\_WIDTHU} \geq \log_2(\text{COLUMN\_NUM}.\text{ROW\_NUM}.\text{RX\_BUFF\_NUM}),$$

where COLUMN_NUM is the number of columns in the system considering a 2-D torus topology excluding the feeding nodes, ROW_NUM is the number of rows in the system

[1] There are the TX FIFO, the RX FIFO, and the Check FIFO.

considering a 2-D torus topology, and `RX_BUFF_NUM` is the number of receiving buffers per computing node.

C. Control and Status Registers of the Scheduler

The scheduler[1] implements two 32-bit base address registers: BAR0 and BAR1, each one set to reserve 1 Mbyte of memory space. Access to the control and status registers can be gained with 32-bit single-cycle target memory write and/or 32-bit single-cycle target memory read transactions.

The scheduler is identified on the host PCI bus with the following parameters:

- Device ID 0x0003;
- Vendor ID 0x1172.

The following contains a list of all control and status registers in address incremental order.

CSR 1: **sch_en**

Access Type: Write/Read
Width: 1
Address Offset: 0x00, BAR0
Description:

Scheduler Enable; Write 0x01 to enable the scheduler; Write 0x00 to disable the scheduler; Read returns the actual value.

CSR 2: **trig_sel**

Access Type: Write/Read
Width: 2
Address Offset: 0x08, BAR0
Description:

Selects system trigger; Write 0x00 or 0x01 to select an internal mock-up data generator; Write 0x02 to select an external stimulator; Write 0x03 to select PCI trigger (see also **PCI_event_ID** for PCI trigger); Read returns the actual value. The active signal of the external stimulator has to be connected to pin № 2 of the FDD connector on the CIA-RORC card.

CSR 3: **ev_gen_en**

Access Type: Write/Read
Width: 1
Address Offset: 0x10, BAR0
Description:

Global system trigger enable; Write 0x01 to enable system trigger; Write 0x00 to disable system trigger; System trigger can be configured through **trig_sel**; Read returns the actual value.

[1] PCI 66 MHz capable

CSR 4: **sch_state**

Access Type: Read
Width: 4
Address Offset: 0x20, BAR0
Description:

Scheduler Main State Machine Current State (Figure III-9). Returns: 0x00 when 'Idle' state, 0x01 when 'RequestDestination' state, 0x02 when 'Ready' state, 0x03 when 'TXData' state.

CSR 5: **mode**

Access Type: Write/Read
Width: 1
Address Offset: 0x28, BAR0
Description:

The scheduling discipline (section III.B.2) allows two modes of operation of the scheduler. In the first mode, the scheduler is not allowed to skip columns when choosing the next destination even though the scheduled column might be completely busy at the time of destination request. Write 0x01 to select this mode of operation.

The other mode of operation allows a busy column to be skipped. If a column is completely busy at the time of destination request, the scheduler immediately attempts to obtain a destination located in another column. Thereby it avoids sending two subsequent data packets to destinations residing in one and the same torus column. Write 0x00 to select the second mode of operation. Read returns the actual value.

CSR 6: **utn_tagnet_err_num**

Access Type: Read
Width: UTN_ERR_CNT_WIDTH (section VII.B)
Address Offset: 0x30, BAR0
Description:

Number of errors over the flow control system network (Figure III-12).

CSR 7: **PCI_event_ID**

Access Type: Write
Width: UE_EV_ID_WIDTH (section VII.B)
Address Offset: 0x40, BAR0
Description:

System Trigger via PCI. Initiate single cycle target write transactions against this address to trigger the system, i.e. to emulate an incoming data packet. Requires **trig_sel** set to 0x03.

CSR 8: **fifos_ovrflw**

Access Type: Read
Width: 4
Address Offset: 0x50, BAR0
Description:

Implanted memory to log FIFOs overflow in Unit_TN (Figure III-12) and Unit_E. The register has the following binary encoding:

utn_rx_fifo_ovrflw | *utn_check_buff_ovrflw* | *utn_tx_fifo_ovrflw* | *ev_buff_fifo_ovrflw*,

where *utn_rx_fifo_ovrflw* is the most significant bit, which signals overflow in the input FIFO of the flow control network interface unit; *utn_check_buff_ovrflw* signals overflow in the check buffer of the flow control network interface unit; *utn_tx_fifo_ovrflw* signals overflow in the output FIFO of the flow control network interface unit; *ev_buff_fifo_ovrflw* is the least significant bit, which signals overflow in the buffer of event IDs.

Under normal circumstances, **fifos_ovrflw** returns zero. Any value of **fifos_ovrflw** different than zero shall be considered as system failure.

CSR 9: **event_ID**

Access Type: Read
Width: UE_EV_ID_WIDTH (section VII.B)
Address Offset: 0x60, BAR0
Description:

Input Data Packet ID

CSR 10: **ua_incrmnt_state_cnt**

Access Type: Read
Width: UA_INCRMNT_CNT_WIDTH (section VII.B)
Address Offset: 0x70, BAR0
Description:

Read returns the value of a monitoring counter that is incremented every time the scheduling unit algorithm state machine enters the 'Increment' state (section III.B.2). If the value of **CSR 5** is set to 0x00 and the algorithm state machine does not find any free destinations in the next scheduled 2-D torus column, the scheduler skips the busy column and attempts to get a destination in another column. This act is registered by the monitor counter.

CSR 11: **ev_buff_fifo_usedw_log**

Access Type: Read
Width: 4
Address Offset: 0x90, BAR0
Description:

Implanted memory to log the level of usage of the buffer of event IDs.
Returns the following values:
0x00: up to 25% of the buffer have been used since last system start;
0x01: 25%-50% of the buffer have been used since last system start;
0x03: 50%-75% of the buffer have been used since last system start;
0x07: 75%-87.5 of the buffer have been used since last system start;
0x0F: over 87.5% of the buffer have been used since last system start.
See also **fifos_ovrflw** and **ev_buff_fifo_usedw**.

CSR 12: **in_buff_fifo_usedw**

Access Type: Read
Width: UIB_IN_BUFF_FIFO_WIDTHU (section VII.B)
Address Offset: 0xA0, BAR0
Description:

Read returns the instantaneous number of words used in the input buffer FIFO, which buffers new free destination entries prior to their decoding and queuing (section III.B.4). The number of words used equals the number of addresses temporarily

stored in the buffer destination. Under normal circumstances, the value read out of this register has to be zero since the destination addresses are immediately read out of the input buffer for decoding and queuing (section III.B.4).

CSR 13: **utn_tx_fifo_usedw**

Access Type: Read
Width: UTN_RX_FIFO_WIDTHU (section VII.B)
Address Offset: 0xB0, BAR0
Description:

Read returns the instantaneous number of words used in the output buffer of the flow control network interface unit (Figure III-12). Note that storage of a single flow control message in the output buffer requires **TAG_FRAGM_NUM** words, where **TAG_FRAGM_NUM** is the number of message fragments (section VII.B). The current implementation requires two words per message.

The number of words stored in the output buffer of the flow control network interface unit depends on the system size and the input data rate. The value read out of **utn_tx_fifo_usedw** is typically greater than zero. See also **fifos_ovrflw**.

CSR 14: **utn_check_buff_usedw**

Access Type: Read
Width: UTN_CHECK_BUFF_WIDTHU (section VII.B)
Address Offset: 0xC0, BAR0
Description:

Read returns the instantaneous number of words used in the check buffer of the flow control network interface unit (Figure III-12). Storage of a single flow control message in the check buffer requires one word.

The number of words stored in the check buffer of the flow control network interface unit depends on the system size, the number of feeding nodes, and the input data rate. During system run, the value read out of **utn_check_buff_usedw** is greater than zero. It represents the instantaneous number of flow control messages present in the flow control system network. See also **fifos_ovrflw**.

CSR 15: **utn_rx_fifo_usedw**

Access Type: Read
Width: UTN_RX_FIFO_WIDTHU (section VII.B)
Address Offset: 0xD0, BAR0
Description:

Read returns the instantaneous number of words used in the input buffer of the flow control network interface unit (Figure III-12). Note that storage of a single flow control message in the input buffer requires **TAG_FRAGM_NUM** words, where **TAG_FRAGM_NUM** is the number of message fragments (section VII.B). The current implementation requires two words per message.

The number of words stored in the input buffer of the flow control network interface unit depends on the system size and the input data rate. The value read out of **utn_rx_fifo_usedw** is typically greater than zero. See also **fifos_ovrflw**.

CSR 16: **ev_buff_fifo_usedw**

Access Type: Read
Width: UE_EV_BUFF_FIFO_WIDTHU (section VII.B)
Address Offset: 0xE0, BAR0
Description:

Read returns the instantaneous number of words used in the buffer of event IDs. The value represents the number of input data packets pending for processing. See also **fifos_ovrflw** and **ev_buff_fifo_usedw_log**.

CSR 17: **ud_fifos_usedw**

Access Type: Read
Width: ROW_WIDTH (section VII.B)
Address Offset: 0x80000 – 0x800F8, BAR0
Description:

Read returns the instantaneous number of words used in a selected FIFO in Unit_D (section III.B.4), which corresponds to the instantaneous number of destination addresses available in the selected torus column. Note that the number of destination addresses does not equal the number of receiving nodes, since the number of receiving buffers per receiving node in the system setup is **RX_BUFF_NUM** ≥ 2.

Offset 0x80000 selects the 1st column, offset 0x80008 selects the 2nd column, and offset 0x800F8 selects the 32nd torus column. The current implementation of the scheduler supports up to 32 torus columns.

CSR 18: **pci_fetch_addr_of_intrst**

Access Type: Write
Width: 32
Address Offset: 0x00, BAR1
Description:

Initialization software writes at this address offset to specify the address of interest of the PCI Fetcher (section III.B.6).

D. DMA Engine Design Parameters

Certain design parameters, which are located in the scheduler's design package sch_pack.vhd, are also used in the design of the DMA engine. The following contains a list of these parameters, their description in the context of the DMA engine, and the ranges of valid values. The values of these parameters are typically set while adjusting the scheduler to the actual system size (section III.B.8). So the user does not need to modify the values explicitly when recompiling the design of the DMA engine. However, the user should make sure that there is only one version of the design package sch_pack.vhd.

DESC_BUFF_WIDTH

This is an integer that specifies the width of the address bus of the descriptor buffer (section III.E.1.a)). DESC_BUFF_WIDTH must meet the following requirement:

$$\text{DESC\_BUFF\_WIDTH} \geq \log_2(\text{COLUMN\_NUM} \cdot \text{ROW\_NUM} \cdot \text{RX\_BUFF\_NUM}),$$

where COLUMN_NUM is the number of columns in the system, considering a 2-D torus topology excluding the feeding nodes; ROW_NUM is the number of rows in the system, considering a 2-D torus topology; and RX_BUFF_NUM is the number of receiving buffers per computing node (section VII.B).

TAG_FRAGM_NUM

Every flow control message is transferred in 16-bit wide fragments. The value of TAG_FRAGM_NUM is an integer that specifies the number of fragments per message. In the current design, TAG_FRAGM_NUM must be set to 2. The format of a flow control message is shown in Figure III-38.

TAG_FRAGM_WIDTH

This is an integer that must meet the following requirement:

$$\text{TAG\_FRAGM\_WIDTH} \geq \log_2 \text{TAG\_FRAGM\_NUM},$$

where TAG_FRAGM_NUM is the number of fragments per flow control message.

UTN_IN_REG_USED

UTN_IN_REG_USED must be assigned a string that is either 'YES' or 'NO'. The assignment 'YES' implements input registers in the receiving lines of the flow control network interface unit between the FPGA on the transceiver card (Figure III-26) and the host FPGA on the CIA-RORC board (Figure III-22). The registers are placed in the I/O cells; they improve the performance of the data transfer between the plug-in card and the CIA-RORC card. UTN_IN_REG_USED is set to 'YES' in the test setup.

UTN_OUT_REG_USED

This parameter signifies the same as UTN_IN_REG_USED, but it refers to the output lines of the flow control network interface.

UTN_RX_FIFO_WIDTHU

The following equation is true:

$$2^{\text{UTN\_RX\_FIFO\_WIDTHU}} = \text{UTN\_FIFO\_WORD\_NUM},$$

where UTN_FIFO_WORD_NUM is the number of words which can be stored in the TX FIFO or RX FIFO of the flow control network interface unit (section III.E.3).

In the prototype, the value of UTN_RX_FIFO_WIDTHU is chosen so that all flow control messages in the system can be stored in the receiving or transmitting FIFO. Note that storage of a single message in the receiving and transmitting buffers requires TAG_FRAGM_NUM words, with TAG_FRAGM_NUM being the number of fragments per message. The current implementation requires two words per message. Therefore, UTN_RX_FIFO_WIDTHU meets the following requirement:

$$\text{UTN\_RX\_FIFO\_WIDTHU} \geq \log_2\left(2.\text{COLUMN\_NUM}.\text{ROW\_NUM}.\text{RX\_BUFF\_NUM}\right),$$

where COLUMN_NUM is the number of columns in the system, considering a 2-D torus topology excluding the feeding nodes; ROW_NUM is the number of rows in the system, considering a 2-D torus topology; and RX_BUFF_NUM is the number of receiving buffers per computing node (section VII.B).

E. DMA Engine Control and Status Registers

The DMA engine domain[1] implements two 32-bit base address registers, BAR0 and BAR1, each one set to reserve 1 Mbyte of memory space. Access to the control and status registers can be gained with 32-bit single-cycle target memory write and/or target memory read transactions to BAR0, according to the address distribution shown below. The descriptor buffer (section III.E.1.a)) is mapped to BAR1 and can be accessed by 32-bit target memory write and/or read transactions.

A DMA engine domain on the host PCI bus is identified by the following parameters:

- Device ID 0x0004;
- Vendor ID 0x1172.

Name:	**DMACNTRL**
Access Type:	Write/Read
Width:	32
Address Offset:	0x100, BAR0
Description:	

Bit 0: **Start**

The start bit is set by a software process and is redundant in the current version.

Bit 1: **Status**

This bit returns 0x0 when the DMA engine is busy executing data transfer. The bit returns 0x0 immediately after initialization and before the start of the very first transaction. The status bit is set to active high level on completion of a data transaction.

Bit 2: **Resv3**

The bit is reserved.

Bit 3: **Clear**

Initialization software writes 0x1 to initialize certain registers.

Bit 4: **Resv2**

The bit is reserved.

Bit 5: **FlwCtrlIntfcEn**

Active high flow control network interface enable. Initialization software enables the flow control network interface by default.

Bit 7-6: **Resv4**

This bit field is reserved.

[1] PCI 66 MHz capable

Bit 10-8: **Resv1**
Reserved
The bit field is reserved.

Bit 31-11: **Resv5**
Reserved
This bit field is reserved.

Name: **DESCS**
Access Type: Write/Read
Width: 32
Address Offset: 0x300, BAR0
Description:

The value of the DESCS register shows the number of descriptors actually stored in the descriptor buffer (section III.E.1.a)) minus one. The value (DESCS+1) equals the product of `COLUMN_NUM`, `ROW_NUM` and `RX_BUFF_NUM`, where `COLUMN_NUM` (section VII.B) is the number of columns in the system, considering a 2-D torus topology, excluding the feeding nodes. `ROW_NUM` is the number of rows in the system, considering a 2-D torus topology; `RX_BUFF_NUM` is the number of receiving buffers per receiving node.

Name: **TAGS**
Access Type: Read
Width: 32
Address Offset: 0x400, BAR0
Description:

Read the value of this register to get the number of successful data transactions executed by the DMA engine. The value corresponds to the number of data packets transferred from the feeding nodes to the receiving nodes. The values of the TAGS registers read out of all feeding nodes at a certain point of time must be equal.

Name: **INST_RATE**
Access Type: Read
Width: 32
Address Offset: 0x500, BAR0
Description:

Read the contents of this register to obtain the instantaneous data transfer rate, measured as the time between two successive data transfers in units of PCI clock cycles. The method of measurement is realized as shown in Figure VII-2.

The register comprises two fields as follows:

Bit 31 – 16: **unit_m_result**
This is the last measured value of the instantaneous data transfer rate in units of PCI clock cycles.

Bit 15 – 0: **result_id**
This is a measurement identification number that allows the user or user software to discard an old invalid **unit_m_result** value. In case of two successive identical **unit_m_result** values read out of the INST_RATE register, the second **unit_m_result** value should only be considered valid if the two corresponding **result_id** values differ.

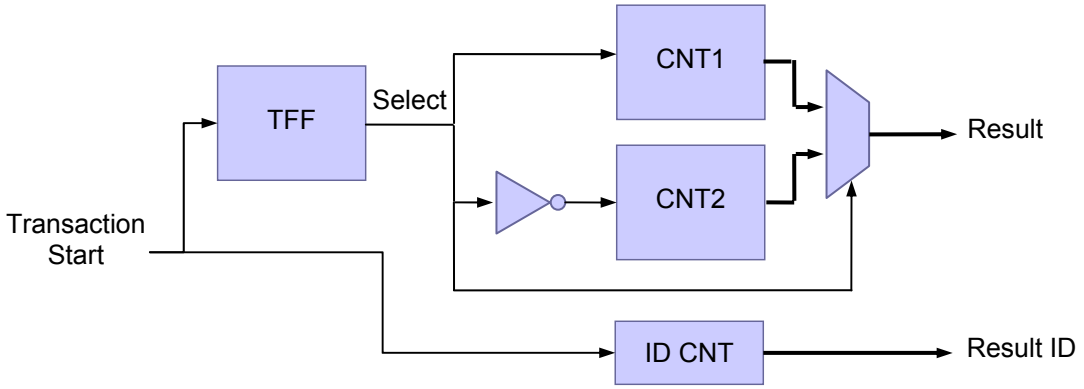


Figure VII-2: A Simple, Efficient, and Precise Method of Measuring the Instantaneous System Performance Rate

Name: **FLW_CTRL_INTFC_OVFLW**
Access Type: Read
Width: 2
Address Offset: 0x600, BAR0
Description:

This is a 2-bit flag that signals overflow in the TX FIFO (bit 1) and RX FIFO (bit 0) of the flow control network interface unit. The flag is implemented as implanted memory.

Name: **BLOCK_SIZE**
Access Type: Write/Read
Width: 32
Address Offset: 0x700, BAR0
Description:

Initialization software writes the data block size that is to be transferred in units of PCI data cycles. A burst length of 128 byte requires 16 PCI data cycles. Read returns the actual value specified.

Name: **DESCBUF**
Access Type: Write/Read
Width: 32
Address Offset: begins at 0x00, BAR1
Description:

The memory-mapped descriptor buffer (section III.E.1.a)).

The number of words in the descriptor buffer is $2^{\texttt{DESC\_BUFF\_WIDTH}}$ (section VII.D).

F. CD-ROM Contents

- Hardware descriptions:

a. Scheduler
b. Feeding nodes' Logic
c. Flow Control Network Interface Card
d. CIA-RORC Configuration Controller

- Schematics:

a. CIA-RORC Board
b. Flow Control Network Interface Card

- Software

The CD contains a README file which should be consulted first.

G. Acronyms and Abbreviations

ADC	Analog-to-Digital Converter
ALICE	A Large Ion Collider Experiment, a future experiment at LHC, CERN
ALICE TPC	the ALICE Time Projection Chamber
ALICE TRD	the ALICE Transition Radiation Detector
ANSI	American National Standards Institute
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ATLAS	A Toroidal LHC AparatuS, a future experiment at LHC, CERN
BaBar	a SLAC experiment, studying CP violations in B meson decay
BAR	Base Address Register
BIOS	Basic Input/Output System
BU	Builder Unit
CAN	Controller Area Network, a serial communication protocol
CBM	Compressed-Baryonic-Matter, a future experiment at GSI, Germany
cc	clock cycle
CDF	Collider Detector at the Fermi National Accelerator Laboratory FNAL, Illinois, USA
CERN	European organization for nuclear research in Geneva, Switzerland
CIA	Cluster Interface Agent, an autonomous hardware unit for remote control of commodity computing nodes
CIA-RORC	Cluster Interface Agent- Read-Out Receiver Card, a multipurpose PCI/stand-alone card
CMC	Common Mezzanine Card
CMOS	Complementary Metal-Oxide-Semiconductor, a technology for production of semiconductor chips, a class of integrated circuits
CMS	Compact Muon Solenoid, future experiment at LHC, CERN
CN	Computing Node
CNT	counter

CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CSR	Control and Status Register(s)
DAC	Digital-to-Analog Converter
DAQ	Data Acquisition System
DCS	Detector Control System
DESY	Deutsches Elektronen-Synchrotron in Hamburg, Germany
DFM	Data Flow Manager
DIMM	Dual In-line Memory Module, memory form factor
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
D0	an experiment at the Fermi National Accelerator Laboratory FNAL, Illinois, USA
ECAL	Electromagnetic Calorimeter
EEPROM	Electrically-Erasable Programmable Read-Only Memory
EF	Event Filter
EIA	Electronic Industries Alliance
EMI	Electromagnetic Interference
EVM	Event Manager
FAIR	Facility for Antiproton and Ion Research in Darmstadt, Germany
FDD	Floppy Disk Drive
FED	Front-End Driver
FIFO	First-In First-Out
FN	Feeding Node
FPGA	Field Programmable Gate Array
GSI	Gesellschaft für Schwerionenforschung in Darmstadt, Germany
GUI	Graphical User Interface
HCAL	Hadron Calorimeter

HERA-B	an experiment at DESY in Hamburg, Germany, dedicated to CP-violation in decays of B mesons
HLT	High Level Trigger
H1	an experiment at DESY in Hamburg, Germany
IEEE	Institute of Electrical and Electronics Engineers
ID	Identification
INFN	the Italian National Institute for Nuclear Physics
I/O	Input/Output
IrDA	Infrared Data Association
KIP	Kirchhoff Institute of Physics, University of Heidelberg
KLOE	an experiment at INFN, Italy
KTev	Kaons at the Tevatron, an experiment at the Fermi National Accelerator Laboratory FNAL, Illinois, USA; dedicated to study of CP-violation and rare decays of the neutral Kaon
LC3	Link Controller 3 (refer to [26])
LEP	Large Electron and Positron collider
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty, a future experiment at LHC, CERN
LSB	Least Significant Bit
LVDS	Low Voltage Differential Signaling
LVL1	Level-1 Trigger in ATLAS
LVL2	Level-2 Trigger in ATLAS
L1	Level-1 Trigger
L2P	LVL2 Processor
L2SV	LVL2 SuperVisor
MSB	Most Significant Bit
NA49	an experiment at the Super Proton Synchrotron (SPS) at CERN
NIC	Network Interface Card
NU	Not Used

ODE	Off-Detector Electronics
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PLL	Phase-locked Loop
PS	Preshower detector
RAM	Random Access Memory
RCU	Read-out Control Unit
RD	read
RICH	Ring Imaging Cherenkov Counter
RN	Result Node
ROB	ReadOut Buffer
RoI	Region-of-Interest
RoIB	RoI-Builder
RORC	Read-Out Receiver Card
RU	Readout Unit
RX	receiver or receive or receiving
Sch	Scheduler
SCI	Scalable Coherent Interface
SDRAM	Synchronous Dynamic Random Access Memory
SFI	Sub-Farm Interface
SISCI	Software Infrastructure for SCI
SLAC	Stanford Linear Accelerator Center
SM	State Machine
SRAM	Static Random Access Memory
SPD	Scintillator Pad Detector
STP	Shielded Twisted Pair
SW	software

TFF	T-Flip-Flop
TIA	Telecommunications Industry Association
TTL	Transistor-Transistor Logic
TX	transmitter or transmitting
UA1	Underground Area 1, an experiment at the Super Proton Synchrotron (SPS) at CERN (W and Z Bosons discovery)
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WR	write
ZEUS	an experiment at DESY in Hamburg, Germany
µC	microcontroller
2-D	two-dimensional
3-D	three-dimensional

Bibliography

[1] The Large Hadron Collider, Conceptual Design, CERN/AC/95-05(LHC), CERN, 1995.

[2] R. Frühwirth, M. Regler, R. K. Bock, H. Grote and D. Notz, Data Analysis Techniques for High-Energy Physics, Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology 11, Cambridge University Press, 2000, Pages 7–9.

[3] G. Abbiendi, et al. (the ALEPH Collaboration, the DELPHI Collaboration, the L3 Collaboration and the OPAL Collaboration, the LEP Working Group for Higgs Boson Searches), Search for the Standard Model Higgs Boson at LEP, Phys.Lett.B565:61–75, 2003, arXiv:hep-ex/0306033v1, http://arxiv.org/abs/hep-ex/0306033 (December, 2008).

[4] V. Büscher and K. Jakobs, Higgs Boson Searches at Hadron Colliders, Int.J.Mod.Phys.A20:2523-2602, 2005, arXiv:hep-ph/0504099v1, http://arxiv.org/abs/hep-ph/0504099 (December, 2008).

[5] P. Sphicas, Data Acquisition Systems, CERN Summer Student Lectures, July 2005, http://webcast.cern.ch/home/pages/lecser_cds.php?prog_year=sslp_2005 (November, 2005).

[6] V. Lindenstruth and I. Kisel, Overview of Trigger Systems, Nuclear Instruments and Methods in Physics Research A 535 (2004) 48–56, February 2004.

[7] ALICE High-level Trigger Conceptual Design, December 2002, http://www.kip.uni-heidelberg.de/ti/L3/concept.html (November, 2008).

[8] LHCb Trigger System, Technical Design Report, CERN/LHCC 2003-031, September 2003.

[9] A. Walsch, Architecture and prototype of a real-time processor farm running at 1 MHz, PhD Thesis, University of Mannheim, 2002, http://www.kip.uni-heidelberg.de/ti/publications/phd/2002AlexanderWalsch.pdf (November, 2008).

[10] LHCb Technical Proposal, CERN/LHCC 98-4, February 1998, http://lhcb-tp.web.cern.ch/lhcb-tp/ (November, 2008).

[11] The CBM Collaboration, Letter of Intent for the Compressed Baryonic Matter Experiment at the Future Accelerator Facility in Darmstadt, 2004, http://www.gsi.de/documents/DOC-2004-Jan-116-2.pdf (December, 2008).

[12] An International Accelerator Facility for Beams of Ions and Antiprotons, Conceptual Design Report, GSI, 2001, http://www.gsi.de/GSI-Future/cdr/ (December, 2008).

[13] CBM Collaboration, Compressed Baryonic Matter Experiment, Technical Status Report, January 2005.

[14] M. J. Wilking, Recent Results from the KTeV Experiment, 2006, arXiv:hep-ex/0606044v1, http://arxiv.org/abs/hep-ex/0606044 (December, 2008).

[15] Gautier Hamel De Monchenault, CP Violation: Recent Results from BABAR, 2003, arXiv:hep-ex/0305055v2, http://arxiv.org/abs/hep-ex/0305055 (December, 2008).

[16] HERA-B Collaboration, Peter Krizan, Rainer Mankel, Dominik Ressing, Sergey Shuvalov, Martin Spahn, HERA-B, an experiment to study CP violation at the HERA proton ring using an internal target, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 351, Issue 1, 15 November 1994, Pages 111–131, http://www.sciencedirect.com/science/article/B6TJM-473FJCM-87/2/8294a37e88a2a81393d7075faaba918b (December, 2008).

[17] Stephen L. Olsen, for the Belle Collaboration, Recent Belle results on CP violation, Int.J.Mod.Phys.A23:3277-3281, 2008, arXiv:0712.2353v1, http://arxiv.org/abs/0712.2353 (December, 2008).

[18] LHCb Vertex Locator Technical Design Report, CERN/LHCC 2001-0011, May 2001.

[19] CERN S-LINK Homepage, http://hsi.web.cern.ch/HSI/s-link/ (November, 2008).

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

[21] The LHCb Timing and Fast Control (TFC) Home Page, http://lhcb-online.web.cern.ch/lhcb-online/TFC/default.html (November, 2008).

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

[23] Dolphin Interconnect Solutions Homepage, http://www.dolphinics.no/ (November, 2008).

[24] M. C. Liaaen and H. Kohmann, SCI: Scalable Coherent Interface, Pages 71–87, Springer Berlin/Heidelberg, 1999.

[25] PCI Local Bus Specification, PCI Special Interest Group, Rev. 2.1, June 1995, http://www.pcisig.com/specifications/conventional/ (November, 2008).

[26] Link Controller 3 Specification D666 – LC-3, Dolphin Interconnect Solutions, Version 1.9, June 2002, http://www.dolphinics.no/support/techweb.html (November, 2005).

[27] ServerWorks, Inc. http://www.serverworks.com (2001).

[28] The CMS Collaboration, CMS Physics Technical Design Report, Volume II: Physics Performance, J. Phys. G: Nucl. Part. Phys. 34 995–1579, 2007, http://stacks.iop.org/JPhysG/34/995 (December, 2008).

[29] CMS Collaboration, Data Acquisition and High-Level Trigger Technical Design Report, CERN/LHCC 02-26, December 2002.

[30] The ATLAS Collaboration, Expected Performance of the ATLAS Experiment – Detector, Trigger and Physics, 2008, arXiv:0901.0512v2, http://arxiv.org/abs/0901.0512 (December, 2008).

[31] ATLAS Collaboration, ATLAS Level-1 Trigger Technical Design Report, CERN/LHCC 98-14, 1998.

[32] ATLAS Collaboration, ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report, CERN/LHCC 03-22, 2003.

[33] F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B. D. Johnsen, H. Kohmann, R. Nordstrom and P. Werner, Low-level SCI Software Functional Specification, version 2.1.1, March 1999.

[34] ALICE High-level Trigger Software and Documentation, http://www.kip.uni-heidelberg.de/ti/L3/ (November, 2008).

[35] Alice Technical Design Report of the Time Projection Chamber, CERN/LHCC 2000-001, ALICE TDR 7, January 2000.

[36] T. Alt, Entwicklung eines autonomen FPGA-basierten Steuerrechners, Diploma Thesis, University of Heidelberg, 2002, http://www.kip.uni-heidelberg.de/ti/publications/diploma/2002ThorstenAlt.ps (November, 2008).

[37] R. Panse, V. Lindenstruth, T. Alt, H. Tilsner, L. Hess, A Hardware Based Cluster Control and Management System, Computing in High Energy Physics (CHEP) Conference, Interlaken, September 2004.

[38] The certonCHARM Homepage, http://www.certonsystems.com/pci.php (March, 2008).

[39] K. Giapoutzis, LHCb Vertex-Trigger Algorithmus, Diploma Thesis, University of Heidelberg, 2002, http://www.kip.uni-heidelberg.de/ti/publications/diploma/2002KonstantinGiapoutzis.pdf (November, 2008).

[40] APEX20K Programmable Logic Device Family Data Sheet, Altera Corporation, http://www.altera.com/literature/ds/apex.pdf (November, 2008).

[41] PCI Compiler Data Sheet, Altera Corporation, version 1.0, August 2001, http://www.altera.com/literature/ds/pcicompiler_ds.pdf (2001)

[42] IEEE, Draft Standard for a Common Mezzanine Card Family: CMC, P1386 Draft 2.4a, March 2001.

[43] APEX20K Programmable Logic Device Family Data Sheet, Altera Corporation, version 2.06, March 2000, http://www.altera.com/literature/ds/apex.pdf (November, 2008).

[44] APEX20K400E Embedded Programmable Logic Device Errata Sheet, Altera Corporation, version 1.0, November 2001, http://www.altera.com/literature/ds/es_ep20k400e.pdf (2001).

[45] Designing with Fineline BGA Packages for APEX, FLEX, ACEX, MAX 7000 & MAX 3000 Devices Application Note 114, Altera Corporation, version 1.1, August 2001, http://www.altera.com/literature/an/an114.pdf (November, 2008).

[46] Using LVDS in APEX20KE Devices Application Note 120, Altera Corporation, http://www.altera.com/literature/an/an120.pdf (November, 2008).

[47] LVDS Signaling Using APEX Device I/O Pins Application Note 138, Altera Corporation, version 1.0, May 2001, http://www.altera.com/literature/an/an138.pdf (November, 2008).

[48] Using the ClockLock & ClockBoost PLL Features in APEX Devices Application Note 115, Altera Corporation, http://www.altera.com/literature/an/an115.pdf (November, 2008).

[49] Using Altera Devices in Multiple-Voltage Systems Application Note 107, Altera Corporation, version 2.0, August 2001, http://www.altera.com/literature/an/an107.pdf (November, 2008).

[50] 5.0V Tolerance in APEX20KE Devices White Paper, Altera Corporation, version 1.02, July 2000, http://www.altera.com/literature/wp/wp5Vapex1.02.pdf (2001).

[51] Board Design Guidelines for LVDS Systems White Paper, Altera Corporation, version 1.0, July 2000, http://www.altera.com/literature/wp/wp_lvdsboard.pdf (2002).

[52] SameFrame Pin-out Design for FineLine BGA Packages Application Note 90, Altera Corporation, version 1.01, September 2000, http://www.altera.com/literature/an/an090.pdf (November, 2008).

[53] MAX 7000B Programmable Logic Device Family Data Sheet, Altera Corporation, http://www.altera.com/literature/ds/m7000.pdf (November, 2008).

[54] Operating Requirements for Altera Devices Data Sheet, Altera Corporation, version 9.02, December 1999, http://www.altera.com/literature/ds/dsoprq.pdf (November, 2008).

[55] Evaluating Power for Altera Devices Application Note 74, Altera Corporation, version 3.1, July 2001, http://www.altera.com/literature/an/an074.pdf (November, 2008).

[56] Metastability in Altera Devices Application Note 42, Altera Corporation, version 4, May 1999, http://www.altera.com/literature/an/an042.pdf (November, 2008).

[57] ANSI/TIA/EIA-644-A (LVDS) Standard, Telecommunications Industry Association/Electronic Industries Association, February 2001.

[58] Cyclone Device Handbook, Altera Corporation, Volume 1, http://www.altera.com/literature/hb/cyc/cyc_c5v1.pdf (November, 2008).

[59] Channel Link Moving and Shaping Information in Point-to-Point Applications, National Corporation Semiconductor Application Note 1041, May 1996, http://www.national.com/an/AN/AN-1041.pdf (November, 2008).

[60] Channel-Link PCB and Interconnect Design-In Guidelines, National Semiconductor Corporation Application Note 1041, August 1998, http://www.national.com/an/AN/AN-1108.pdf (November, 2008).

[61] A Practical Guide to Cable Selection, National Semiconductor Corporation Application Note 916, October 1993, http://www.national.com/an/AN/AN-916.pdf (November, 2008).

[62] Data Transmission Lines and Their Characteristics, National Semiconductor Corporation Application Note 916, October 1992, http://www.national.com/an/AN/AN-806.pdf (November, 2008).

[63] DS90CR215/DS90CR216 +3.3V Rising Edge Data Strobe LVDS 21-Bit Channel Link-66 MHz, National Semiconductor Corporation Data Sheet DS012909, March 1999, http://cache.national.com/ds/DS/DS90CR215.pdf (November, 2008).

[64] DS90CR217/DS90CR218A +3.3V Rising Edge Data Strobe LVDS 21-Bit Channel Link- 85 MHz, National Semiconductor Corporation Data Sheet DS101080, May 2002, http://cache.national.com/ds/DS/DS90CR217.pdf (November, 2008).

[65] MAX9209/MAX9211/MAX9213/MAX9215 Programmable DC-Balance 21-Bit Serializers, Maxim Integrated Products Data Sheet 19-2828, April 2003, http://pdfserv.maxim-ic.com/en/ds/MAX9209-MAX9215.pdf (November, 2008).

[66] MAX9210/MAX9212/MAX9214/MAX9216 Programmable DC-Balance 21-Bit Serializers, Maxim Integrated Products Data Sheet 19-2864, April 2003, http://pdfserv.maxim-ic.com/en/ds/MAX9210-MAX9222.pdf (November, 2008).

[67] LVDS Owner's Manual, National Semiconductor Corporation, Second Edition, 2000, http://www.national.com/appinfo/lvds/files/ownersmanual.pdf (November, 2008).

[68] H. Mueller, et al., TAGnet, a twisted pair protocol for event-coherent DMA transfers in trigger farms, 8th WS on Electronics for LHC, Colmar, 2002.

[69] ORCA OR3TP12 Field-Programmable System Chip (FPSC) Embedded Master/Target PCI Interface Data Sheet, Lucent Technologies Microelectronics Group, March 2000.

[70] APEX20KE PCI Development Board Data Sheet, Altera Corporation, version 1, December 2000, http://www.altera.com/literature/ds/dspcibd20ke.pdf (2001).

[71] SignalTap Embedded Logic Analyzer Megafunction Data Sheet, Altera Corporation, version 2.0, April 2001, http://www.altera.com/literature/ds/dssignal.pdf (November, 2008).

[72] Design Verification Using the SignalTap II Embedded Logic Analyzer Application Note 280, Altera Corporation, version 1.1, June 2003, http://www.altera.com/literature/an/an280.pdf (2003).

[73] Single- & Dual-Clock FIFO Megafunctions User Guide, Altera Corporation, version 1.0, June 2003, http://www.altera.com/literature/ug/ug_fifo.pdf (November, 2008).

[74] PCI Compiler User Guide, Altera Corporation, version 2.0, August 2001, http://www.altera.com/literature/ug/ug_pci.pdf (November, 2008).

[75] Johan Eker, Chamberlain Fong, Jorn Janneck, Jie Liu, Design and simulation of heterogeneous control systems using Ptolemy II, Proceedings IFAC Conference on New Technologies for Computer Control, November, 2001, http://www.gigascale.org/pubs/293.html (December, 2008).

[76] F. Teubert, Status of the Level1 Trigger Simulation, 27th LHCb Week, December 2002.

[77] The Altera Homepage, http://www.altera.com (November, 2008).

[78] SCICONFIG: Configuration Tool for Dolphin Adapter Cards, User's Guide, DI950-10294, Dolphin Interconnect Solutions AS, version 2.2, February 2003, http://www.dolphinics.com/support/documentation.html (November, 2005).

[79] SCIDIAG: SCI Diagnostic Tool, User's Guide, DI950-10295, Dolphin Interconnect Solutions AS, version 0.1, April 1999, http://www.dolphinics.com/support/documentation.html (November, 2005).

[80] Installation Guide for the Dolphin SCI Adapter Card, DI950-10353, Dolphin Interconnect Solutions AS, version 1.1, May 2004, http://www.dolphinics.com/support/documentation.html (November, 2005).

Acknowledgements

I would like to express my gratitude to Prof. Dr. Volker Lindenstruth for the opportunity to work on this project, for the support through the years as well as for his patience.

I also thank Prof. Dr. Ulrich Brüning, the second referee of this thesis, for his willingness to assess my work.

Ivan Kisel is gratefully acknowledged for providing the simulation results used in this dissertation.

I also owe a debt of gratitude to Venelin Angelov and Ivan Kisel for all the comprehensive discussions on various crucial topics relating to my work.

Many thanks to Ivan Rusanov for a number of fruitful discussions and advice concerning digital electronics issues.

I have to thank the members of the Electronics Department at KIP, especially Holger Hoebel, Volker Kiworra, and Klaus Schmitt, for the good cooperation during the implementation of the printed circuit boards developed within my work.

Timm Steinbeck and Arne Wiebalck have continuously provided me with support related to the Cyclops cluster as well as to Linux and software-related problems.

It is not possible to mention all the people who contributed to my work. Therefore, I would like to finally thank all the colleagues at the Chair as well as all the external collaborators who enjoyed working with me so as I did.

Most importantly, none of this would have been possible without the love and encouragement of my family throughout all my time in Germany. I deeply appreciate their belief in me.