Researching methods for efficient hardware specification, design and implementation of a next generation communication architecture

Inauguraldissertation

zur Erlangung des akademischen Grades

eines Doktors der Naturwissenschaften

der Universität Mannheim

vorgelegt von

Dipl.-Inf. Patrick R. Haspel

aus Mannheim

Mannheim, 2006

Dekan:	Professor Dr. M. Krause, Universität Mannheim
Referent:	Professor Dr. U. Brüning, Universität Mannheim
Korreferent:	Professor Dr. V. Lindenstruth, Universität Heidelberg

Tag der mündlichen Prüfung: 4. Mai 2007

Abstract

The objective of this work is to create and implement a System Area Network (SAN) architecture called EXTOLL embedded in the current world of systems, software and standards based on the experiences obtained during the ATOLL project development and test.

The topics of this work also cover system design methodology and educational issues in order to provide appropriate human resources and work premises.

The scope of this work in the EXTOLL SAN project was:

- the Xbar architecture and routing (multi-layer routing, virtual channels and their arbitration, routing formats, dead lock aviodance, debug features, automation of reuse)
- the on-chip module communication architecture and parts of the host communication
- the network processor architecture and integration
- the development of the design methodology and the creation of the design flow
- the team education and work structure.

In order to successfully leverage student know-how and work flow methodology for this research project the SEED curricula changes has been governed by the Hochschul Didaktik Zentrum resulting in a certificate for "Hochschuldidaktik" and excellence in university education.

The complexity of the target system required new approaches in concurrent Hardware/Software codesign. The concept of virtual hardware prototypes has been established and excessively used during design space exploration and software interface design.

Zusammenfassung

Das Ziel dieser Arbeit ist der Entwurf sowie die Implementierung einer System Area Network (SAN) Architektur namens EXTOLL, die in die heutige Welt der Rechensysteme, Software und Standards eingebettet ist.

Diese Arbeit beinhaltet auch die Aspekte der Systementwurfsmethodik und Ausbildungsfragen um eine geeignete Arbeitsumgebung sowei ein kompetentes Team zu gewährleisten.

Die Beiträge zu der EXTOLL SAN Architektur sind:

- Die XBar Architektur sowie Routing (multi-layer routing, Virtuelle Kanäle sowie deren Arbitrierung, Routingformate, Deadlockvermeidung, Testeigenschaften, Automatisierung der Wiederverwendung)
- Die on-chip Modulkommunikationsarchitektur und Teile der Hostkommunikation
- Netzwerkprozessorarchitektur und Integration
- Die Entwicklung einer Entwurfsmethodik und die Erstellung eines Designflows

Um den erforderlichen Ausbildungsstand der Studenten sowie eine geeignete Arbeitsabläufe für dieses Forschungsprojekt sicherzustellen wurden die Änderungen des SEED Lehrplanes von dem Hochschuldidaktikzentrum begleitet und dabei als Ergebnis das Zertifikat für Hochschuldidaktik und Exzellenz in der Lehre verliehen.

Die Komplexität des Zielsystems erforderte neue Ansätze im simultanen Hardware/Software Codesign. Das Konzept virtueller Hardwareprototypen wurde geschaffen und intensiv während der Entwurfraumanalyse und dem Entwurf der Softwareschnittstellen genutzt.

1 Introduction

System Design is a highly complex challenge requiring the breath and depth of more than one science. A team facing this challenge needs manifold know-how resources as well as several levels of expertise. Reseach/Teaching associates focusing on different areas lead groups of students with different levels of expertise guided and mentored by the head of the group, the professor. One very critical key to success is the ability to perceive and identify the different abilities and furthermore the leadership to encourage and challenge them.

The involved students experience a dedication in the team's success in a way that mostly all of them do not simply work for grades but sacrify their spare free time willingly and consciously to contribute their part to the system design challenge.

My role in this team was to lead a group of students (10 diploma students, 11 students at their studies-related project work, 11 seminar students and a bunch of research assistants over the period of 5 years) focusing on routing architectures for the scientific part and hardware design methodology for the engineering part. In order to educate, guide and stear the diverse characters I worked closely with the university didactics group (german: Zentrum für Hochschuldidaktik, Baden-Württemberg). The work of every student contributed a piece of a puzzle of the system design challenge formed in this work to a comprehensive section of the entire system design ATOLL/EXTOLL.

1.1 Objectives

The objective of this work is to create and implement a System Area Network (SAN) architecture called EXTOLL embedded in the current world of systems, software and standards based on the experiences obtained during the ATOLL project development and test.

The ATOLL project was established around 1996 at the Computer Architecture Group of the University of Mannheim. My contribution as research assistant, diploma student and later on as research associate and Ph.D. student built the history that influenced the methodology applied and architecture decisions made during the evolution of this work.

The influences with the greatest impact on this work came from the experiences during the final RTL implementation, functional verification, physical implementation (back-end) and system burn in phases. It has to be mentioned that the ATOLL project was not

only chip design and implementation but also software development (like API, daemons, low level OS drivers, MPI port), printed circuit board (PCB) design, cable design, discrete component selection and qualification (like qualification of LVDS signaling, double stacked connectors, ...) as well as chip package design.

1.2 Topics of this work

As the objective of this work is not only architecture definition but also implementation with a system focus, the topics of this work also cover system design methodology and educational issues in order to provide appropriate human resources and work premises.

1.2.1 Creating the work/design environment

For leading edge design environment we engaged with the major EDA (Electronic Design Automation) vendors like Cadence and Mentor. Together with the VCAD services group of Cadence Design Systems we deployed a leading edge design flow covering the entire ASIC design methodology from front to back and also include major enhancements students's education curriculum. This project is called SEED (Support for Education in Electronic Design) and is still operational due to it's success and tremendous feedback from industry.

For design and verification IP we engaged with major IP companies like Denali, CoWare (acquired LisaTek) or Virage Logic. Denali and CoWare provided most of the IP used in [47].

For cell libraries we engaged with Virtual Silicon Technologies and Artisan (now part of ARM), for I/O we created a close collaboration with Prof. Tielerts Microelectronics Group of the University of Kaiserslautern being able to design LVDS IO's before any library vendors were on market.

For foundry access we talked to TSMC and UMC as well as later on IBM.

In order to successfully leverage student know-how and work flow methodology for this research project the SEED curricula changes has been governed by the Hochschul Didaktik Zentrum resulting in a certificate for "Hochschuldidaktik" and excellence in university education.

1.2.2 Architecture definition

The scope of this work in the EXTOLL SAN project was:

- the Xbar architecture and routing (multi-layer routing, virtual channels and their arbitration, routing formats, dead lock aviodance, debug features, automation of reuse)
- the on-chip module communication architecture and parts of the host communication
- the network processor architecture and integration
- the development of the design methodology and the creation of the design flow
- the team education and work structure.

1.2.3 Hardware/Software design methodology

The complexity of the target system required new approaches in concurrent Hardware/Software codesign. The concept of virtual hardware prototypes has been established and excessively used during design space exploration and software interface design. Therefore, the ESL (Electronic System Level) language SystemC has been introduced. The definition of the EXTOLL API strongly relies on this concept.

Furthermore, a design methodology has been developed and a corresponding design flow has been created in order to target leading edge deep submicron designs.

1.3 Thesis organization

Because of the different scopes covered in this work the thesis is organized in three main parts:

1. The general metrics and requirements of SANs (chapter 3 "SANs in general" on page 30)
2. The applied methods and the approach (chapter 4 "Methods - Approach" on page 53)
3. The architecture definition and discussion (chapter 5 "Architecture and Function Scope of Building Blocks" on page 78)

Before this main parts start application driven communication requirements and state of the art SAN solutions are presented in chapter 2 "Cluster Computing and Implementa-

tions" on page 5.

The work is concluded and provides the reader with an outlook in chapter 6: ’Conclusion and Outlook’ on page 135.

Enjoy reading!

2 Cluster Computing and Implementations

As already depicted in chapter 1: 'Introduction', cluster computing is an emerging trend in High Performance Computing (HPC). In order to build efficient cluster interconnect or SANs the demands in HPC need to be analyzed. HPC is clearly dominated by distributed and parallel computing. This work focuses on the communication part in HPC omitting the compute part. The architectures of compute parts are well researched and usually known as processor architecture. The reader might wonder, why we then analyzed possible processor architectures in chapter 5.2: 'NPU (Network Processing Unit)' on page 80. Here, the processor architecture exploration was done regarding the needs of communication and the application on a network interface controller, not regarding general purpose computation power.

2.1 Communication demands of distributed and parallel computing

Consider a task, that has to be accelerated beyond the point, that is possible to compute on a single CPU. Consider further, that this task can be parallelized in a way that multiple CPUs can work on it. Now, it depends on the requirements of communication between these parallel threads/CPUs. Depending on the kind of task or algorithm these requirements can range from:

No communication required during computation

This means the only communication is required to scatter the working set (the data to be worked on) at the very beginning and to gather the results at the very end. A

perfect example for this category of algorithms is the computation of Mandelbrot Set fractal images like Figure 1.

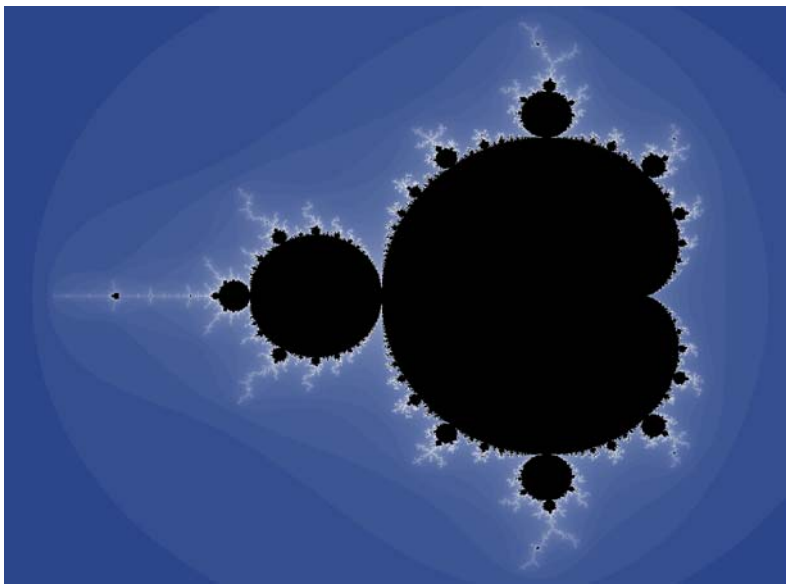


Figure 1: Mandelbrot Set fractal image

The parallel Mandelbrot algorithm does not need any communication during computation. Every image point can be calculated without interaction with his neighbours. So, considering 4 CPUs every CPU works on his dedicated image part without any communication needed at the boundaries. The only communication required is to assign the image part (working set) to the CPU at the beginning of the computation (scather) and to gather the results at the very end.

The example mentioned before refers to the problem of raytracing an entire movie. Here, the single frames are computed separately and therefore, there is no need for communication during the computation of a single frame. However, if one decides to further parallelize the computation of a single frame, the problem shifts classes, as rays in a frame are far from independent.

Massive communication demands during computation

This means communication is required to exchange iteration results between interacting CPUs. Here it is from highest importance to differentiate on the communication pattern of interacting CPUs. One example for all2all communication, defined as every CPU needs to interact with every other CPU, is the N-body simulation of fluid hydrodynamics [2]. The other extreme, called nearest neighbour communica-

tion, exchanges interation data only with their next (in terms of interconnection topology) partners. An example for nearest neighbour communication algorithms is the computation of heat distribution or QCD [39].

Now, applications and algorithms can be classified by their communication demand.

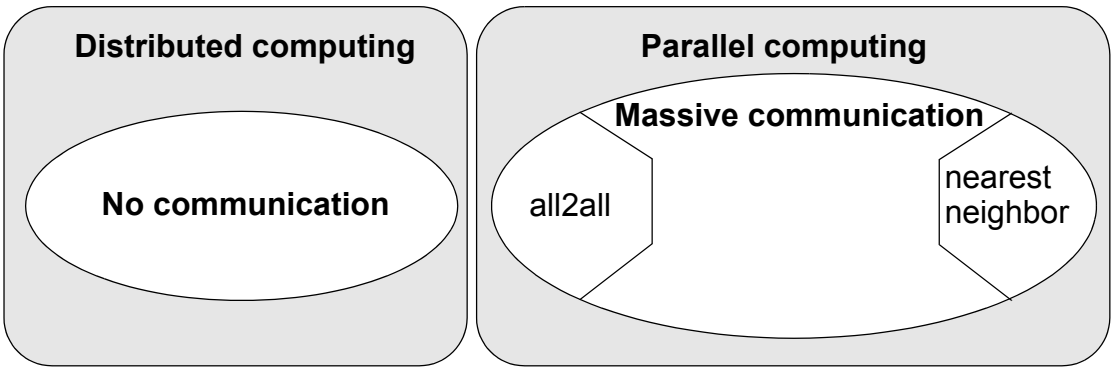


Figure 2: Classification of communication demands

Algorithms with 'No communication required during computation' are in the class solved by distributed computing, where algorithms with 'Massive communication demands during computation' are in the class solved by parallel computing. Figure 2: 'Classification of communication demands' gives an overview.

Some interconnection networks can take advantage of the algorithm specific communication pattern depending on its architecture and implemented topology.

In the following sections a couple of SANs are described and analysed, starting with the most sophisticated architecture of QsNet^{II}. During the description of the SAN features their use model and functionality are analyzed.

2.2 State of the art hardware support

There are several SANs available to build up clusters, which are discussed in the following sections. However, regulars LANs like Gigabit Ethernet are also used because of the low cost and the wide acceptance and big user community. In the TOP500 list of November 2004 there are 176 (which is 35.2%) systems using Gigabit Ethernet. The fastest Gigabit Ethernet cluster achieved rank 44. So, all faster systems do not use Gigabit Ethernet. It is a matter of fact, that from the performance point of view Gigabit Ethernet is not the optimal interconnection network to be used in cluster systems. There is no hardware support at all for common parallel computation constructs like barriers, broad-

casts and also small message sizes. Performance always comes at a cost, so relying on Gigabit Ethernet for low cost low communication performance is still an option. Especially clusters used for distributed computing (i.e. raytracing a movie, where the frames are computed in parallel or distributed web servers or databases), that does not require special SAN features are candidates for feasible Gigabit Ethernet communication. Here it has to be pointed out the difference in communication demands between parallel computation and distributed communication:

2.2.1 QsNet^{II} by Quadrics

With a market share of only 4% in the TOP500 list (November 2004) Quadrics is not one of the big players despite of its dedicated architecture.

The current interconnection network by Quadrics is called QsNet^{II} [27]. It consists of two hardware building blocks: a PCI-X based network interface controller called Elan4 and a connecting switch called Elite4.

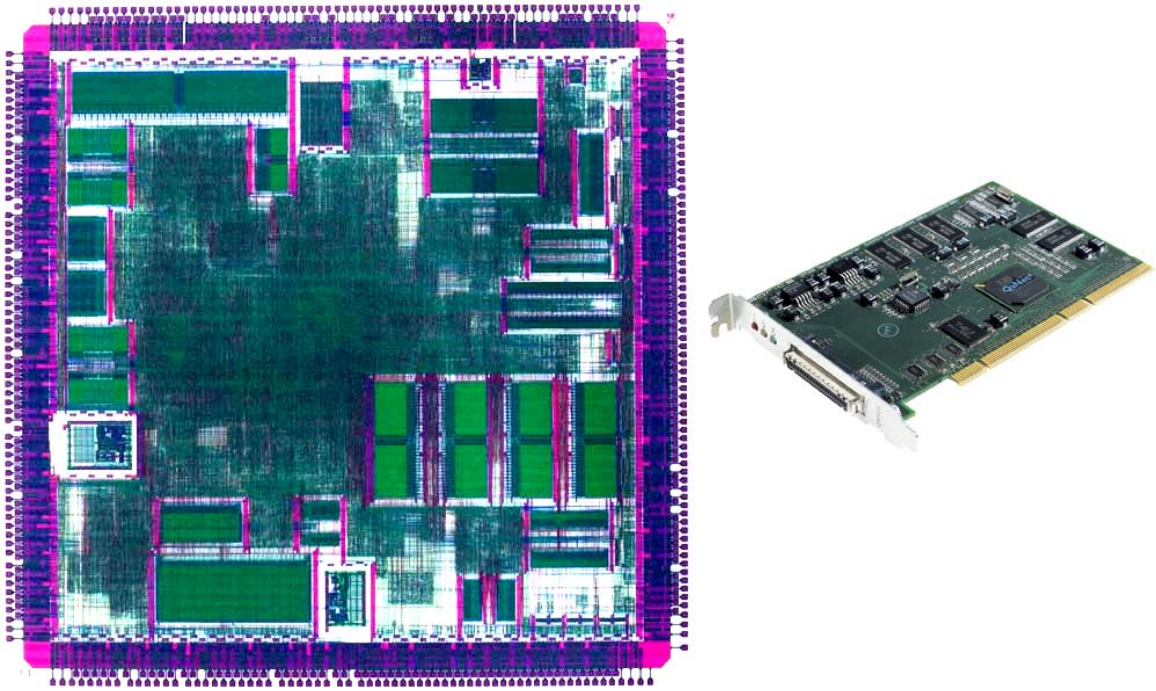


Figure 4: Layout of the Elan4 and the Elan4 network interface card [28]

Elan4

The network interface controller is responsible to insert messages from the host into the

network and vice versa. The Elan4 chip measures 7.5mm by 7.5mm, has approx. 6 million transistors consuming 4 Watts. It is fabricated using the LSI 0.18um process. The Elan4 contains a thread processor and an event engine responsible for CPU offloading of protocol tasks like MPI tag matching or global operations. Also some parts of the MPI progress engine are considered to be transferred from the main CPU to the NIC.

CPU protocol offloading are considered in [43].

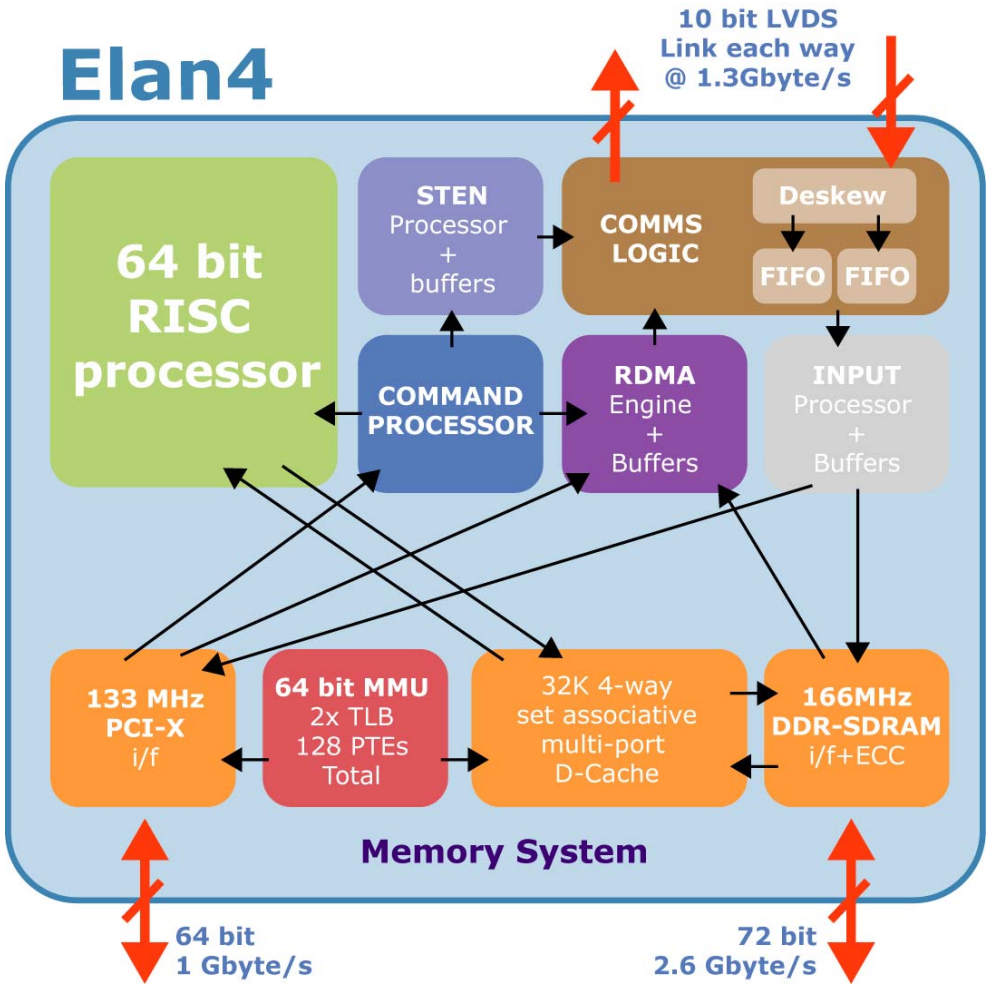


Figure 5: Elan4 top level block diagram [28]

Elite4

The Elite4 ASIC is the connecting switch of the QsNet^{II}. It is responsible for routing the messages through the switch hierarchy from and to the Elan4. The Elite4 chip measures 8.67mm by 8.67mm, has approx. 1 million gates consuming 6.5 Watts. It is fabricated

using the LSI 0.18um process and contains a 4 by 4 bidirectional crossbar supporting 2 virtual channels at each input port [29].

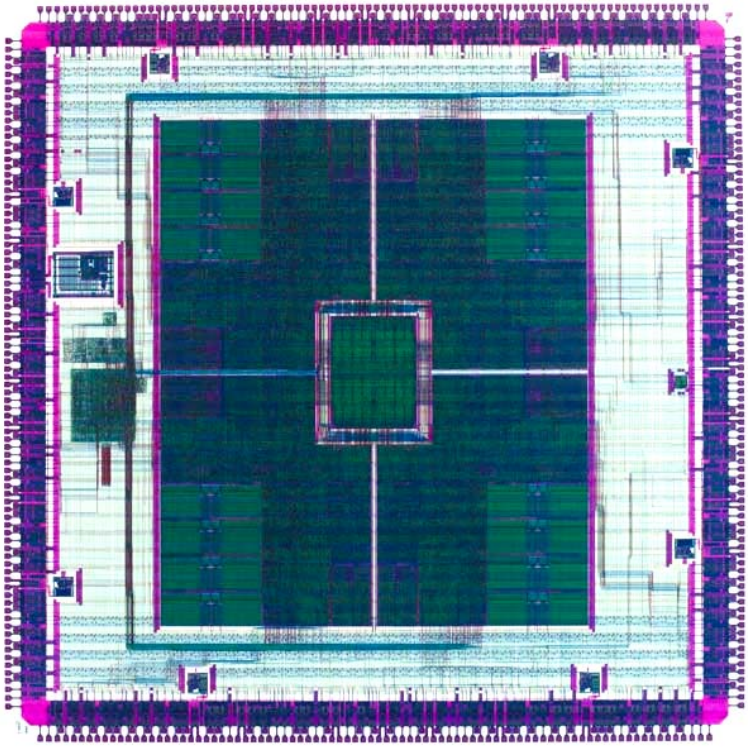


Figure 6: Layout of the Elite4 ASIC [28]

Some remarkable aspects for QsNet^{II} are following, that are described in detail later on:

- Pageable Virtual Memory support including address translation on the NIC and TLB coherence
- Fixed topology: quaternary fat tree
- Reliable transmission
- Virtual network interfaces
- Special short message processing for ultra-low latency
- Support for global operations

Virtual Memory Support

Quadrics with QsNet has been the first SAN delivering native support for the use of virtual addresses in communication. However, the concept has been initially developed in the SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor) project

at Princeton University using the virtual memory-mapped communication (VMMC) model.

> *Virtual memory-mapped communication (VMMC) was developed out of the need for a basic multicomputer communication mechanism with extremely low latency and high bandwidth. This is achieved by allowing applications to transfer data directly between two virtual memory address spaces over the network. The basic mechanism is designed to efficiently support applications and common communication models such as message passing, shared memory, RPC, and client-server.*
>
> *Experience with implementing connection-oriented communication using VMMC exposed some deficiencies in the VMMC model. VMMC-2 was designed to overcome those deficiencies. VMMC-2 extends VMMC with three mechanisms: a user-managed TLB mechanism for address translation which enables user libraries to dynamically manage the amount of pinned space and requires only driver support from many operating systems, a transfer redirection mechanism which allows to avoid a copy on the receiver's side, and a reliable communication protocol at the data link layer which allows to avoid a copy on the sender's side. [34]*

An efficient implementation of the programming paradigm of one-sided communication demands the initiator, regardless whether it is a "PUT" or "GET", to handle remote addresses in contrast to the paradigm of two-sided communication, where the remote address is delivered by the remote "RECEIVE" call.

Using one-sided communication, also called remote DMA, the initiator issues a communication instruction including localID, local_address, amount_of_data as well as the remodeID and the remote_address. The localID identifies the initiating node and process. The local_address is a virtual address in the context of the process (identified by localID). Respectively, the remoteID identifies the target node and process and the remote_address is the target virtual address within the process context.

As user-level communication is necessary regarding the demand for low-latency, the OS may not be involved recurrently in the address translation. To give an example the "PUT" communication pattern is described in the following:

1. The NIC is informed about the intended communication, by a "PUT" instruction containing the above described parameters.
2. In order to fetch the data the NIC needs to know the physical address of the data values to be processed.
   2.1 To avoid recurrent OS calls asking for an address translation for every message, a TLB (Transaction Look Aside) like structure is meaningful.
   2.2 In a system that uses paging, the NIC needs to be informed about paging activity as ongoing transfers to/from memory need to be cancelled, if the OS decides to swap a page to memory. More details about possible options can be found in chapter 3.1 "Functions, Features and important quality metrics of a SAN" on page 31.
3. Data will be fetched page by page and transmitted
4. On the target side the data arrives together with the remodeID (specifying also the remote process context) and the remote virtual address.
5. Address translation needs to be perfomed on the target side also. The same restrictions as in 2.1 and 2.2 apply.

The Elan4 contains a TLB structure and delivers a LINUX patch in order to get the NIC TLB involved in OS triggered CPU TLB flushes. This enables the QsNet to use virtual addresses in communication without pinning down related pages. Again, all possible options and design decisions are described in detail in chapter 3.1 "Functions, Features and important quality metrics of a SAN" on page 31. To get further information about LINUX patches and TLB coherence please refer to [14] ’Thomas Schlichter, "Exploration of Hard- and Software Requirements for one-sided, zero copy user level Communication and its Implementation", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2003’

Fixed topology: quaternary fat tree

> *QsNet connects Elite switches in a quaternary fat-tree topology, which belongs to the more general class of k-ary n-trees. A quaternary fat tree of dimension n is composed of* 4^n *processing nodes and* $n \times 4^{n-1}$ *switches interconnected as a delta network; it can be recursively built by connecting*

four quaternary fat trees of dimension n - 1. Figure 7 shows quaternary fat trees of dimensions 1, 2, and 3.[30]

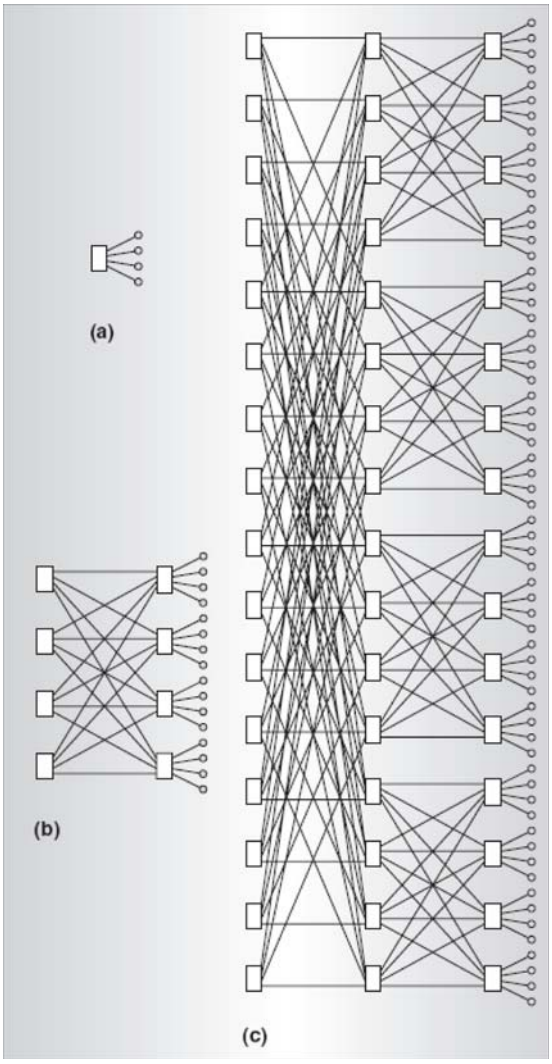


Figure 7: Quaternary n-trees of dimensions 1 (a), 2 (b), and 3 (c).

The fat tree topology was selected due the large number of alternate routes between nodes, the linear scaling in bisectional bandwidth with network growth and the ease of implementing global network operations such as broadcast. The broadcast mechanism is the same as implemented in previous generations of Elite allowing broadcasts to arbitrary ranges of nodes between an upper and lower limit.[28]

A further advantage of this topology is to make use of nearest-neighbor communication

patterns. Adjacent nodes have lower communication latencies.

The worst disadvantage can be seen in the scalability issue. Adding nodes to a complete topology means adding a dimension *n*. In the case of an incomplete node configuration parts of the topology remain unused and therefore, the hardware utilization is not efficient.

As the wiring between the Elite4 ASICs is hard mounted in the switch boxes custom changes are not possible. The topology is fixed on the PCBs of the switches.

Reliable transmission

The term "reliable transmission" has a defined meaning in the area of SANs. It states, that no security layer in software is needed to ensure a correct transmission. Instead, the hardware is responsible to retransmit packets in the case of bit errors on the link.

SAN architects have observed the distribution of latency between software and hardware and have found a heavy impact of copy operations due to the security layer in safe protocols. The security layers have been responsible to preserve a copy of every transmitted message in order to retransmit it in the case of a failure. As the hardware is now responsible for correct (reliable) transmission, there is no need to duplicate every message for security reasons. This reduces the over-all latency dramatically.

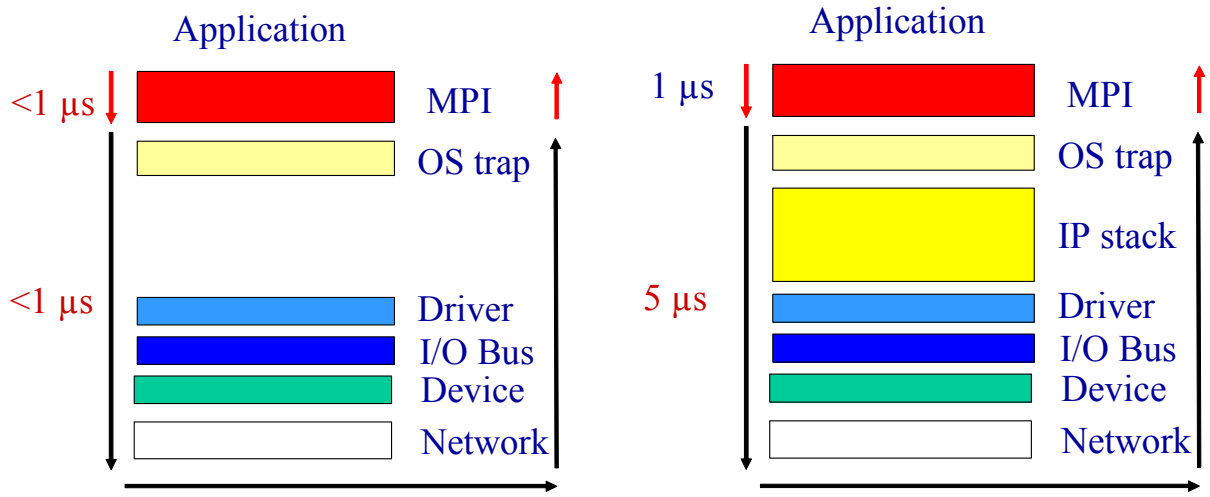


Figure 8: Latency distribution using 10GEthernet (right) and a typical SAN (left)

Of course, there are more operation principles to be applied in order to further reduce latency, that will be discussed in chapter 3.1 "Functions, Features and important quality metrics of a SAN" on page 31.

Virtual network interfaces

Virtualizing devices is a quite mature concept, that has been applied on most available devices from CPUs to disc controllers and also common LAN devices like Ethernet controllers in order to deliver the functionality of the device to arbitrary processes. However, this virtualization is currently managed by the operating system as the control instance. A perfect example are TCP ports, that are virtual endpoints of LAN communication. There is only one physical LAN device, that can be operated by arbitrary processes using the port concept.

As SANs always try to avoid OS calls due to latency aspects, the real challenge is virtualizing devices without OS interaction.

So called user-level virtual devices deliver full functionality of a regular device to arbitrary processes without OS intervention. The virtualization is meant to be 100% transparent to the software. However, OS interaction is still needed to create/manage/delete virtual device contexts, but regular communication happens on the user level only.

The entire concept of virtualizing a device, including the management of device contexts is described in [42].

Special short message processing for ultra-low latency

As the PCI-X bus is the main contributor of latency to communication, it is from highest priority to limit the number of PCI-X cycles, that are necessary to start a communication. So, it is an optimization for short messages to provide a way to start a communication with fewer PCI-X cycles than regular messages. A common way is to include the data payload in the message descriptor in order to save the data fetch step from main memory, that usually requires one or more PCI-X cycles.

Another possibility is to avoid main memory accesses by the NIC at all. The performance of the PCI-X bus delivering data by programmed I/O (PIO) regarding latency (approx. 350ns depending on the chip set) is much better, than a DMA read access to main memory (approx. 750ns) by the NIC [27]. Depending of the ability of the main CPU to combine stores directed to the PCI-X bus, the bandwidth increases also. Table 1 shows

the influence of burst length on the resulting bandwidth.

Processor	Maximum burst length by PIO	Resulting bandwidth
Itanium Tiger	128bytes	600MByte/s
Opteron	64bytes	450MByte/s

Table 1: Ability to combine program stores to bursts[27]

QsNet^{II} supports different handling of messages depending on their size. It is important to deliver high bandwidth and low latency for all message sizes, so there is a threshold where messages are delivered in one way or the other.

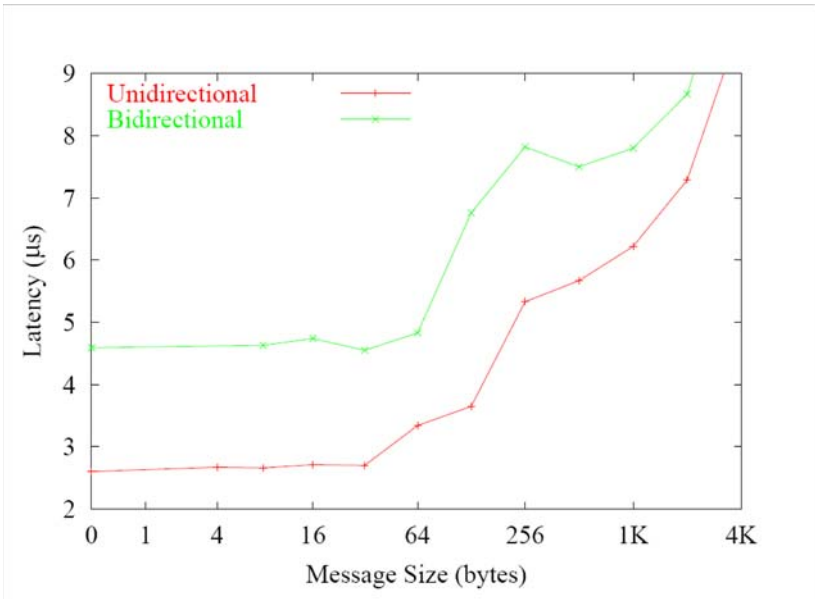


Figure 9: Latency depending on the message size

In Figure 9 the latency for variable message sizes is drawn. It can be seen, that the latency is quite stable for messages smaller than 32 byte. These messages are delivered using the "small message mode". For messages bigger than 32 byte the latency raises suddenly; the message transfer mode has been switched.

Support for global operations

> *The Elan4 has a synchronization engine, called the Event Processor, used to control the action to be performed when an operation completes. This controls the signaling of the completion of an operation. When an Event fires it*

> *causes a copy of data of up to 2KB in size to be written to a user defined virtual address. This copy can be directed into a command queue and provides a very flexible mechanism for a very low latency response to stimulus from the network without the need to start a thread running on the thread processor. For example this can be used in network scatter/gather operations. Incoming data, from the network, is gathered together and then copied in separate packets routed to a number of different network destinations. Using this method in excess of 4 million packets per second have been constructed and injected back into the network.[27]*

There is a rudimentary support by Elan4 for NIC controlled and processed global operations, but is not clear which MPI2 commands are hardware accelerated nor how big the performance impact is.

However, there are ways to offload the main CPU from simple arithmetic transformations if there is some kind of programmable processing unit running concurrently on the NIC. Methods and requirements are described in chapter 3.1 "Functions, Features and important quality metrics of a SAN" on page 31.

2.2.2 Myrinet by Myricom

With a share of 38,6% in the TOP500 list (Nov. 2004) Myricom has the widest acceptance and surpasses even Gigabit Ethernet with a share of 35,2%. Please find below the list of interesting features and Figure 10: ’Block diagram of a Myrinet F-Card [38]’.

Features:

- central crossbar (CLOS network)
- build from 16 x 16 integrated switch
- PCI-X Interface
- LANai on chip processor for communication function support (333MHz)
- does hashing for the TLB table lookup
- Cameleon port is 10GEthernet compatible
- 1067MB/s for 64-bit, 133MHz PCI-X

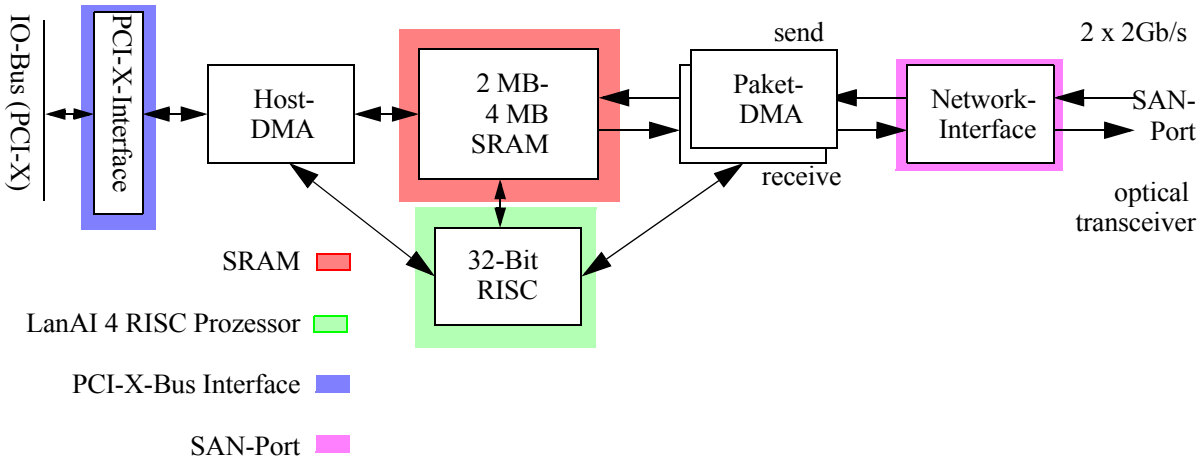


Figure 10: Block diagram of a Myrinet F-Card [38]

Network Topology

All Myrinet flavors implement a CLOS network. The tree topology is build in a manner, that all computation nodes (tree leafs) has the same distance with regard to the number of hops through switches. So, there is no positive impact of nearest neighbour communication patterns on communication performance.

Communication processor

Myrinet uses a 32-bit communication processor called LANai to enable message transfer. As every transfer task is controlled by the LANai this system is highly configurable with regard to the transfer mechanism. A dedicated message SRAM is used to buffer messages being sent and received. The LANai has control over the Host DMA engine, the SRAM and the Packet DMA. Configurability as an advantage stands against the disadvantage of complex programming of the LANai and the cost impact of the dedicated SRAM.

However, the presence of the dedicated SRAM and the communication processor enables Myrinet to implement a TLB-like functionality for virtual/physical address translation.

Network interface

The network interface called Cameleon is capable of driving the 10Gig Ethernet protocol. This feature is used to hook up to commodity 10Gig Ethernet LANs in order link clusters of high-performance communication with regular LANs.

Myrinets cost/performace ratio and the stability of the system as well as some historical issues led to the enormous market share. From a scientific view point the architectural features are not as interesting as for example Quadrics network.

2.2.3 Pathscale

The Pathscale's approach is a very pragmatic one creating a pretty simple but very effective SAN mostly build around de-facto and industry standards.

Being able to step up one hop nearer to the main CPU reduces latency fundamentally. Please refer to section chapter 3.1.6 "Communication to / Location of the Network Interface Controller" on page 50 for further discussion on other location related benefits.

Pathscape has no switching capability and relies on the Infiniband network infrastructure for routing messages through the network.

To sum up, Pathscale is the perfect example how to reduce latency by moving nearer to the main CPU. All other aspects of SANs (chapter 3 "SANs in general" on page 30) are not considered in this solution as Pathscape is more like a Hypertransport/Infiniband interface. However, the consequent use of standards and the avoidance of additional on NIC memory can make this solution very cost effective.

One drawback for current systems is, that only clusters of AMD CPU (because of the Hypertransport bus) can be used and a dedicated Hypertransport slot on the mainboard must be present (HTX connector).

Streamlined approach to significantly reduce latency

- Fully pipelined hardware
- No external memory necessary
- Low processor utilization

Built around industry standards:

- InfiniBand Physical/Link Layers and subnet management
- Linux
- MPI
- HyperTransport bus & HTX connector (Iwill motherboard)

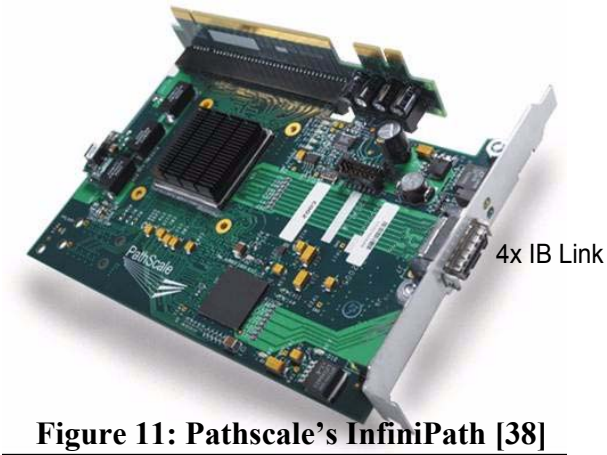


Figure 11: Pathscale's InfiniPath [38]

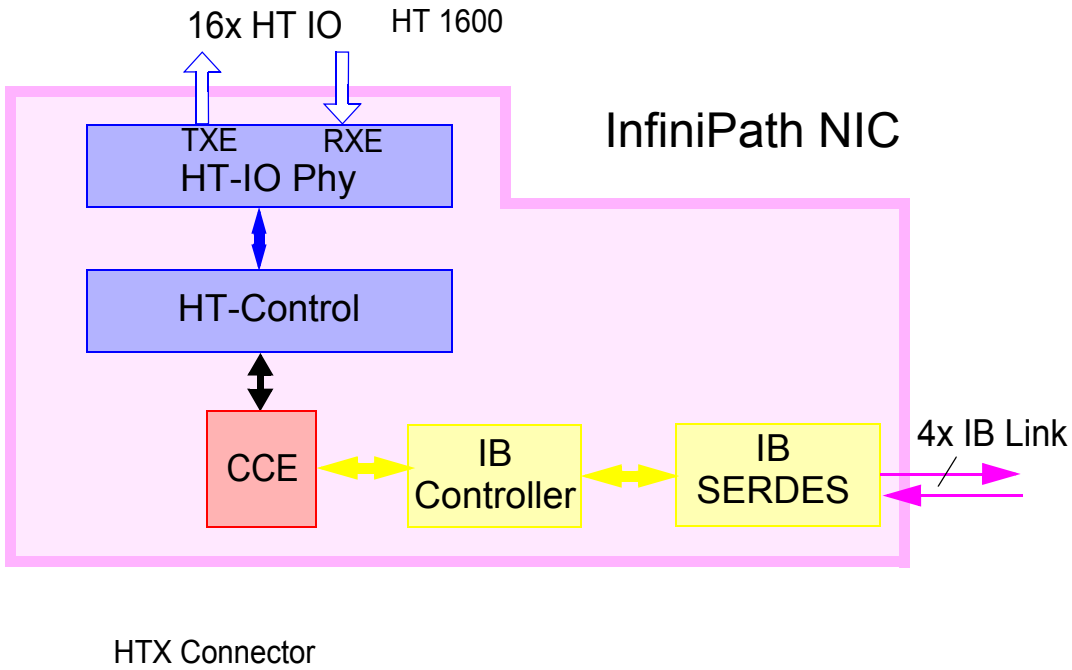


Figure 12: Block diagram of Pathscale's InfiniPath [38]

Performance:

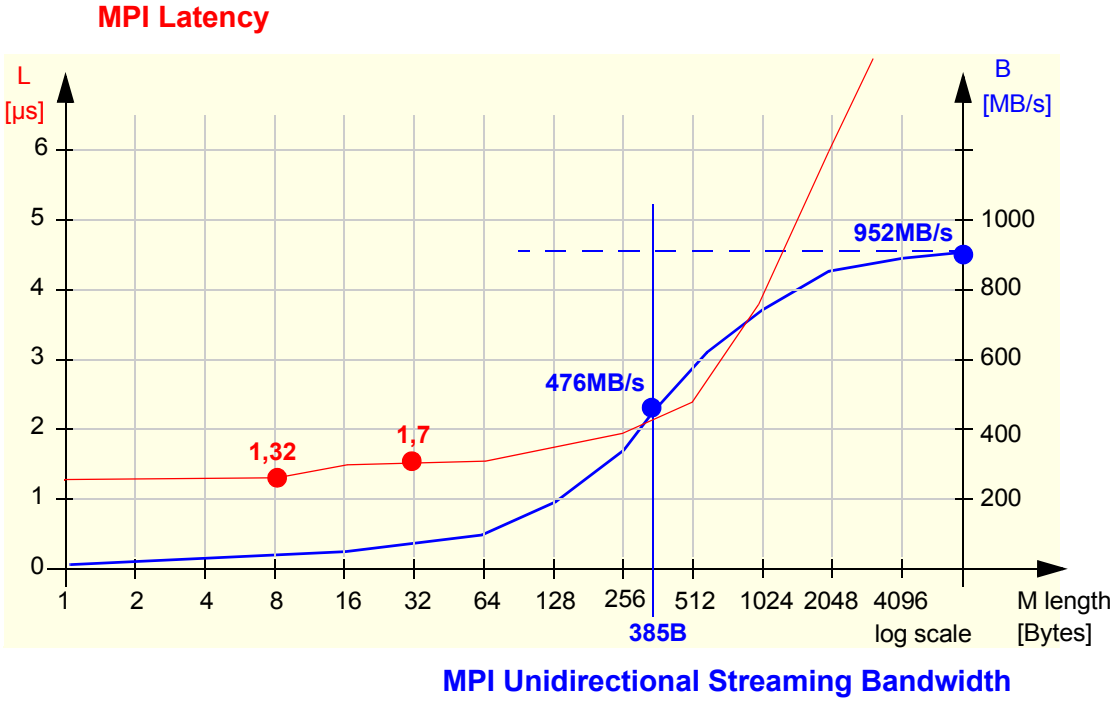
Peak bandwidth at HT IO (no overhead) = 1.6GHz x 16bit (2B) x bidir = 3.2GB/s x2 = 6.4GB/s

Peak bandwidth at IB = 4 x 2.5Gbit/s x 2 = 2.5GB/s, with coding overhead 2GB/s, one direction 1GB/s, protocol overhead ~5% = 950MB/s

Max. bandwidth = 952MB/s

$n_{1/2}$ message size is 385B, reduces further if more CPUs are in the node.

Bi-directional Streaming Bandwidth = 1875 MB/s



TCP/IP Latency of 6.7µs and a bandwidth of 583MB/s (Standard LINUX stack).

2.2.4 PCI-ASI (Advanced Switching Interconnect)

ASI is based on the layer 1 and 2 (physical and link layers) of the PCI express (PCIe) standard. The PCIe's transaction layer and above are exchanged in order to add features like protocol encapsulation and routing (in addition to the simple load/store interface of

PCIe) (Figure 12). It is important to note that despite of the other SAN examples given in this work this is the description of a standard not an implementation. Up to now there are no implementations of the ASI standard. However, there are concepts in this standard that has to be discussed as some of these ideas influenced this work.

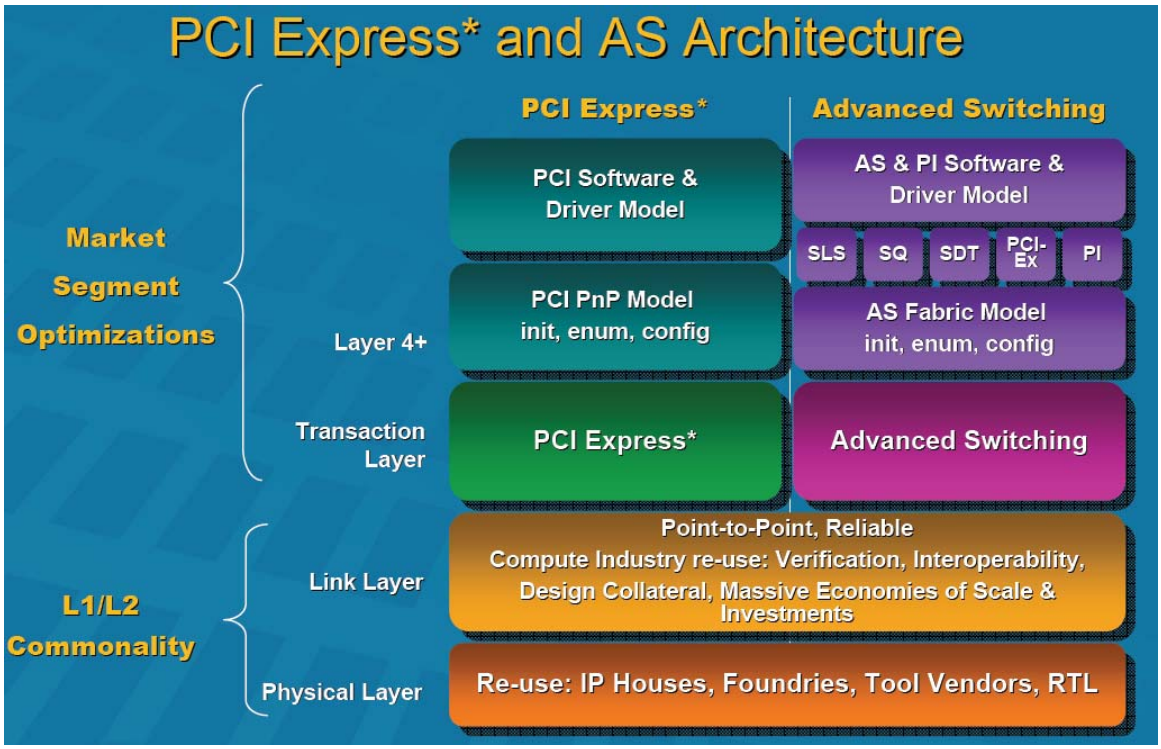


Figure 12: AS integration in PCIe (courtesy of ASI-SIG)

Especially interesting and needful to be considered is the routing function of ASI. It relies on a relative turn mechanism. The output port is calculated depending on the input port by adding the right turn value in the routing string. The right turn value is marked by the turn pool pointer (Figure 13). This routing function supports switches with different amount of ports as the turn pool pointer is interpreted bit-wise. Also interesting is the implicit return route. By processing the routing string backwards subtracting turn values,

the return route is implicitly given.

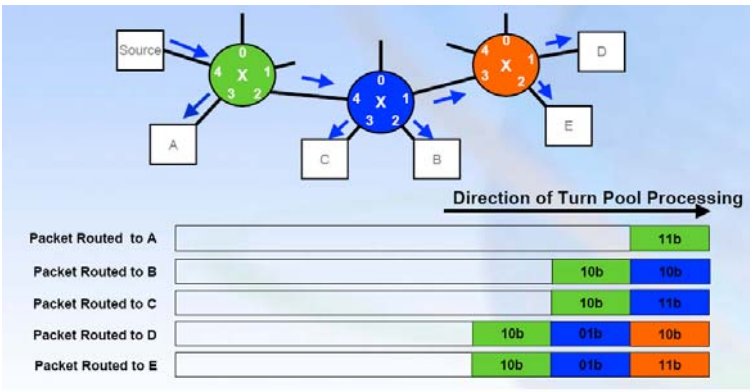


Figure 13: AS routing (courtesy of ASI-SIG)

However, none of the principles has been adopted due to several drawbacks:

- Using dimension ordered routes to minimize deadlock probability in tori topologies limits the use of return routes that are identical to the original route.
- As the routing string is not consumed, the crossbar arbiter needs to gather the entire string prior to a routing decision. This puts additional penalty on the latency introduced by the crossbar.

2.2.5 IBM Blue Gene BG/L

The ’IBM Blue Gene BG/L’ is kind of a brute force response to the Japanese Earth Simulator build by NEC. Using a large number of nodes with moderate clocked CPUs puts lots of pressure on the required parallelization of algorithms. However, using the cost saving principle of commodity CPUs (dual core PowerPC440 plus additional FPUs, Fig-

ure 14) the resulting price /performance ratio is impressive.

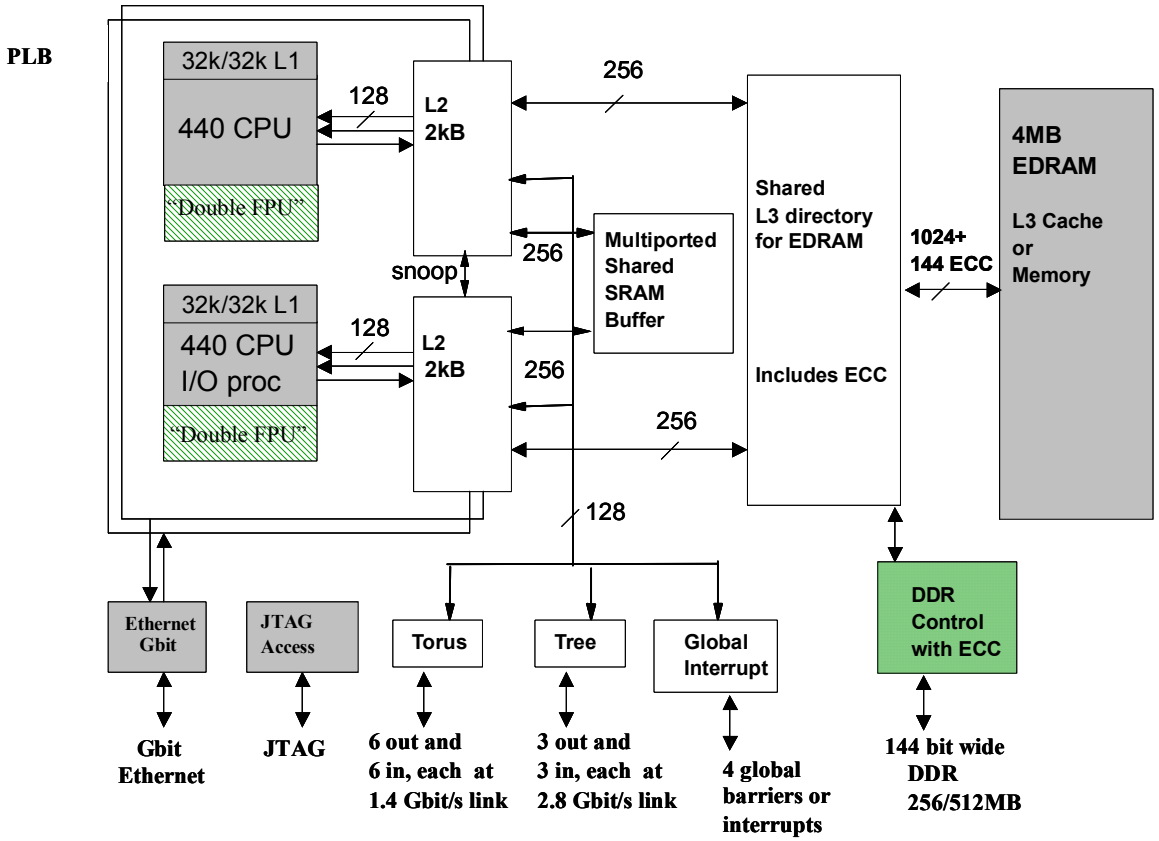


Figure 14: Block diagram of IBM's BlueGene [38]

Interesting for this work is the use of application dedicated concurrent networks (Figure 15). The 3D torus network has a bandwidth of 2,1 GBit/s per node coming from 6 bidirectional 350 MBit/s links. Due to the pin limitation IBM used serial links supporting four virtual channels. The route through latency is only 69ns per hop.

The tree network used for global operations and barrier communication has an ALU integrated to compute bitwise operations like XOR as well as integer operations like ADD or MAX.The latency for a global sum over 64k nodes is below 2,5 us. The tree network supports up to four independent barriers or interrupt channels with a latency of only 1,5

over 64k nodes.

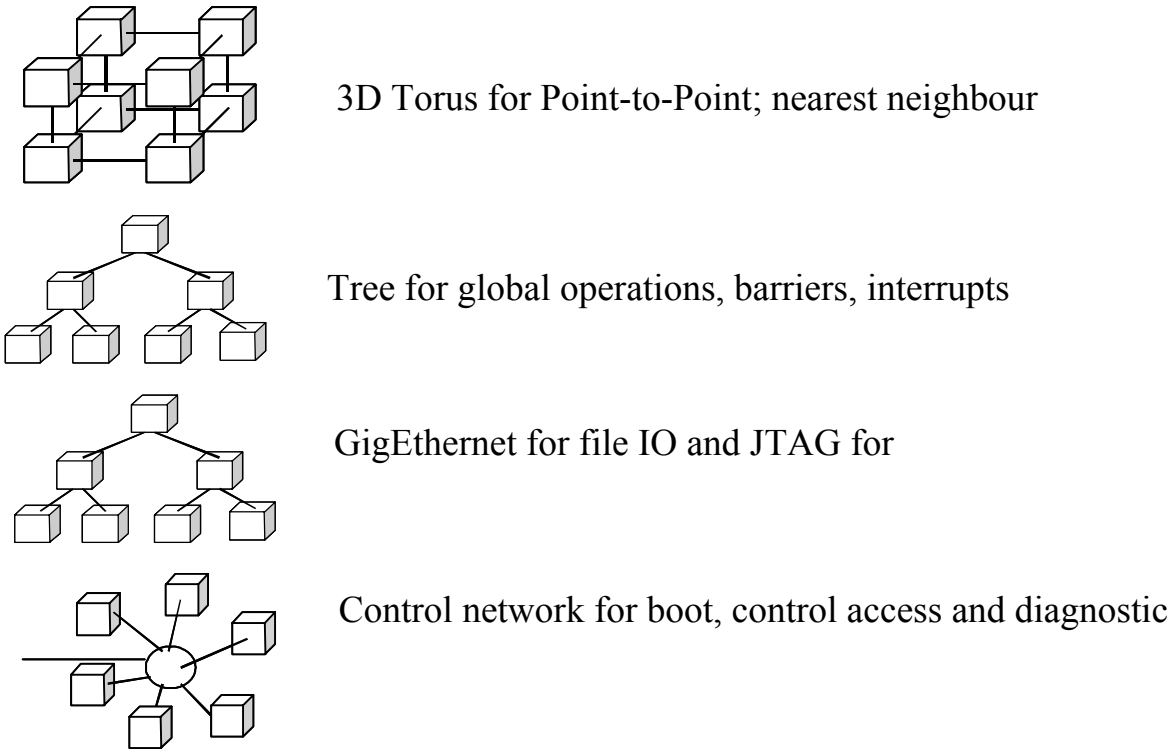


Figure 15: Different networks of the 'IBM Blue Gene BG/L'

The remaining two networks are commodity. A Gigabit Ethernet for file IO and a JTAG control network for the boot process, control access and diagnostics.

The integration of a compute resource in the EXTOLL network (chapter 5.2 "NPU (Network Processing Unit)" on page 80) has been influenced by BlueGene's ability to compute global operations by the network decreasing main CPU load as well as latency.

The barrier hardware support of the EXTOLL network (chapter 5.3 "Network Switch" on page 89) was solved avoiding additional physical networks by mapping a virtual tree topology on the torus physical connectivity.

2.2.6 Mellanox Infiniband

The Infiniband standard has been defined in the year 2000 in version 1.0 by the Infiniband Trade Association. Dell, Hewlett-Packard, IBM, Intel, Microsoft and Sun Microsystems founded the organization and sit on its steering committee. Mellanox was one of the first vendors supplying Infiniband standard conform hardware beginning of the year 2001. As the standard was originally defined to be the PCI successor the support for

parallel computation constructs like barriers and global operations is negligible. Also, the complexity of the standard is hampering fully compliant hardware implementations. However, the large consortium behind the standard drives industry acceptance and the success in the TOP500 list (rank 5) is proving the concept.

Nevertheless, from the research point of view only some principles like credit based flow control, virtual channels and the "Memory and Translation Mechanism (MTP)" are interesting.

Figure 16: MHEA28-XT Card (courtesy of Mellanox)

2.2.7 Cray XT3

Cray started to install the successor of the Cray T3E in September 2004 at the Sandia Lab called RedStorm. The system consists of 11.646 AMD Opteron processors with 10 TByte memory and 240 TByte hard disk space.

Figure 17: Installation in the Pittsburgh Supercomputing Center (courtesy of the PSC)

The building block consists of an AMD Opteron CPU with dedicated (local) memory and a HyperTransport link to the Cray proprietary SeaStar communication resource (Figure 18).

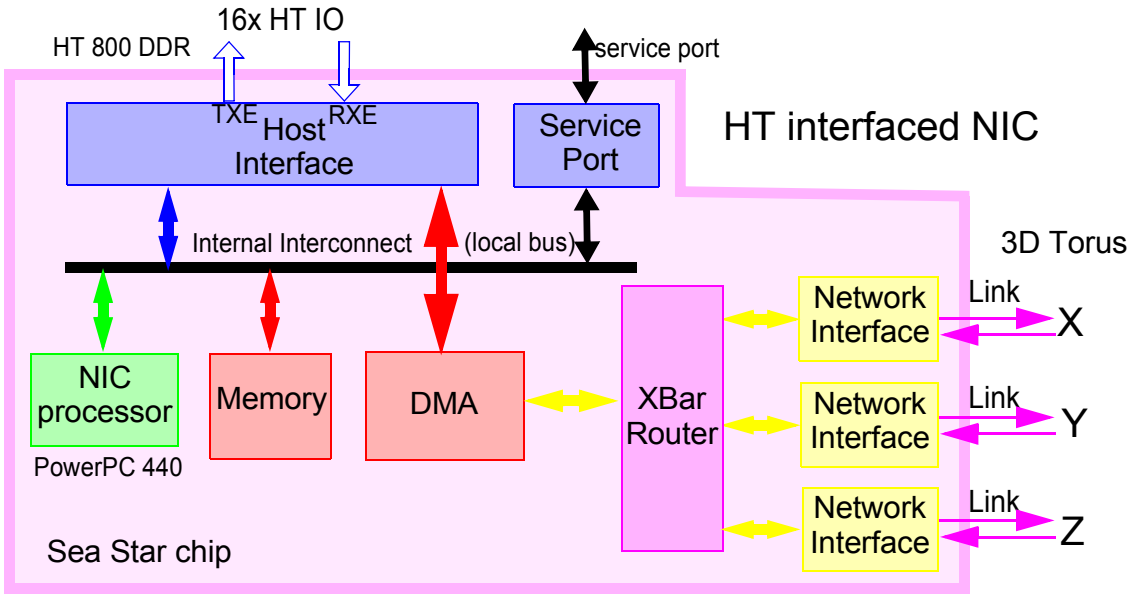


Figure 18: Cray SeaStar block diagramm [38]

The 16x HyperTransport interface provides up to 6,4 GBit/s bandwidth between the AMD Opteron and the SeaStar communication resource. This removes the bottleneck of the PCI interface used in most commodity SANs.

> *The Cray SeaStar chip combines communications processing and high speed routing on a single device. Each communications chip is composed of a HyperTransport link, a Direct Memory Access (DMA) engine, a communications and management processor, a high-speed interconnect router, and a service port. The chip features a DMA engine and an associated PowerPC™ 440 processor. These work together to off-load message preparation and demultiplexing tasks from the Opteron Processor [38]*

The decentralized crossbar architecture is comparable to the ATOLL network providing a peak bandwidth of 6 Gbit/s per bidirectional link, which sums up to 36 Gbit/s (6 links for the 3D torus topology) bandwidth to the network. The chip is manufactured in a 130nm IBM CMOS technology.

Here, Cray makes a promising move to use commodity computations resource combined with a proprietary communication resource linking these via the HyperTransport interface tightly. As mentioned throughout this work (chapter 3.1.6 "Communication to / Lo-

cation of the Network Interface Controller" on page 50) integrating the communication resource as tight as possible to the computation resource is reducing latency, enabling cache coherence and higher bandwidths.

2.2.8 10GigEthernet by Chelsio

> *Chelsio Communications is the market and technology leader for 10 Gigabit Ethernet adapters. Chelsio's board-level products employ the company's unique ICs that process compute-intensive communications protocols at 10Gbps rates. Unburdened of this communications-processing overhead, host servers and storage systems that use Chelsio adapters dramatically increase both applications performance, and communications bandwidth.*
>
> *Chelsio products accelerate network performance in enterprise data centers, high performance cluster computing (HPCC), enterprise-wide data storage systems, and post-production shops for digital film and video. Chelsio is a privately-held subsystems company in Sunnyvale, California. [48]*

The Chelsio approach relies on the ethernet standard, offloading TCP stack processing from the node's CPU. For compute intensive protocol stacks like the TCP stack and link bandwidths of 1Gbyte and more, TCP offload engines like the Chelsio's Terminator architecture reduce the node's CPU load dramatically. The Terminator architecture imple-

ments an entire TCP stack, providing a standard socket interface to the application.The

Figure 19: Chelsio's N210 10GigEthernet Adapter [48]

application to application latency of <10 us is quite in the range of typical SANs, but these are usually not measured though one or more switches. It further need to be considered, that the Chelsio hardware does only support a socket data transfer and no parallel computation paradigms. This requires the software to handle typical MPI operations, like barriers or global operations. Please refer to chapter 3.1: 'Functions, Features and important quality metrics of a SAN' on page 31 to read about why it is not sufficient comparing just latency and bandwidth of a communication device in order to predict or forecast the performance of parallel applications.

However, for certain parallel applications with a low degree of communication-to-computation ratio fast LANs like the Chelsio N210 might be used as cluster interconnect.

3 SANs in general

Different classes of communication has been discussed in chapter chapter 2.1 "Communication demands of distributed and parallel computing" on page 5. System Area Networks are considered to deliver the communication task in a parallel computing environment. Depending on the intended computational problem that has to be solved some requirements on SANs are from more or less importance to the overall performance of the parallel system.

Derived from the above, it is like in the processor evolution, where performance evaluation isn't just a collection of numbers combined in a performance expression. Like in the area of SANs evaluation of processor performace is done using a set of benchmarks, that have different characteristics. Again, depending on the computational problem to be solved a specific benchmark might be of more or less importance for the intended problem solution. The collection of benchmarks (Figure 20) range from microbenchmarks, that focus on a single or small set of metrics, over general benchmarks that consider a generic arithmetic problem like matrix multiplication, to application benchmarks that emulate exactly the intended algorithm.

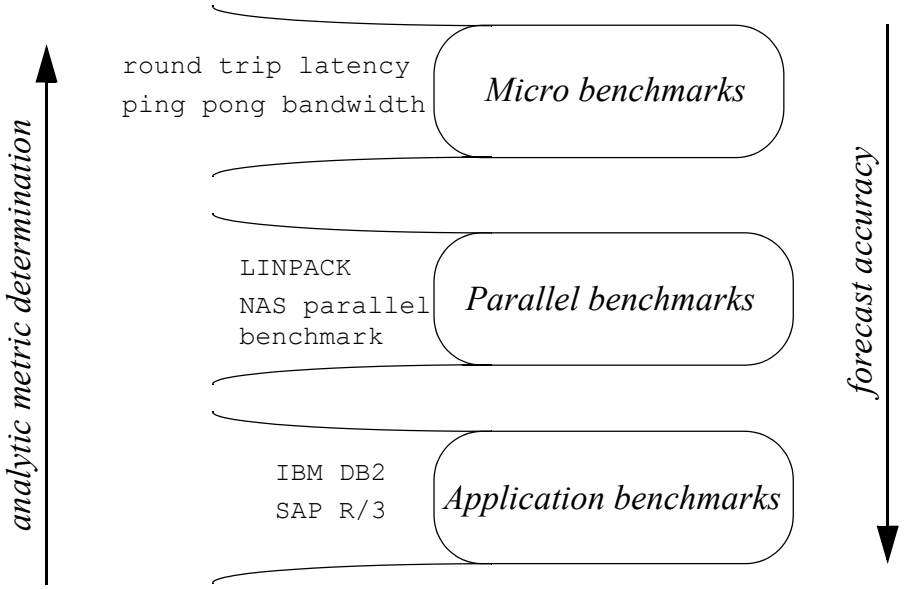


Figure 20: Collection of Benchmarks

There are some metrics enhancing communication in any way, like the communication latency. However, the problem in comparing different SANs still persists as SANs with

equal latency might behave completely different in different applications or even compute systems. This work focuses on methods, operation principles, architectures and programming models of SANs instead of numbers solely in order to highlight the background of the why and how.

The TOP500 list, a ranking of the 500 fastest compute systems worldwide, use the LINPACK benchmark to rank every system. Here, some comments of them regarding benchmarks:

> *The benchmark used in the LINPACK Benchmark is to solve a dense system of linear equations. For the TOP500, we used that version of the benchmark that allows the user to scale the size of the problem and to optimize the software in order to achieve the best performance for a given machine. This performance does not reflect the overall performance of a given system, as no single number ever can. It does, however, reflect the performance of a dedicated system for solving a dense system of linear equations. Since the problem is very regular, the performance achieved is quite high, and the performance numbers give a good correction of peak performance.*

3.1 Functions, Features and important quality metrics of a SAN

Obviously, SANs add some functionality to regular data transport networks or LANs like Ethernet or ATM. In the following sections SAN specific functions and features, important for performance and efficiency as well as scalability, will be categorized. Furthermore, operation principles and architectures are discussed in order to enhance the specific issue.

3.1.1 Latency

Latency is considered to have the most overall impact on parallel computation speedup. The higher the latency the coarser the granularity of the possible (and feasible) parallelization. So, it is from highest importance to decrease the overall latency. The first step is to analyze the contributing sources for latency starting with the MPI function call (Fig-

ure 21, left).

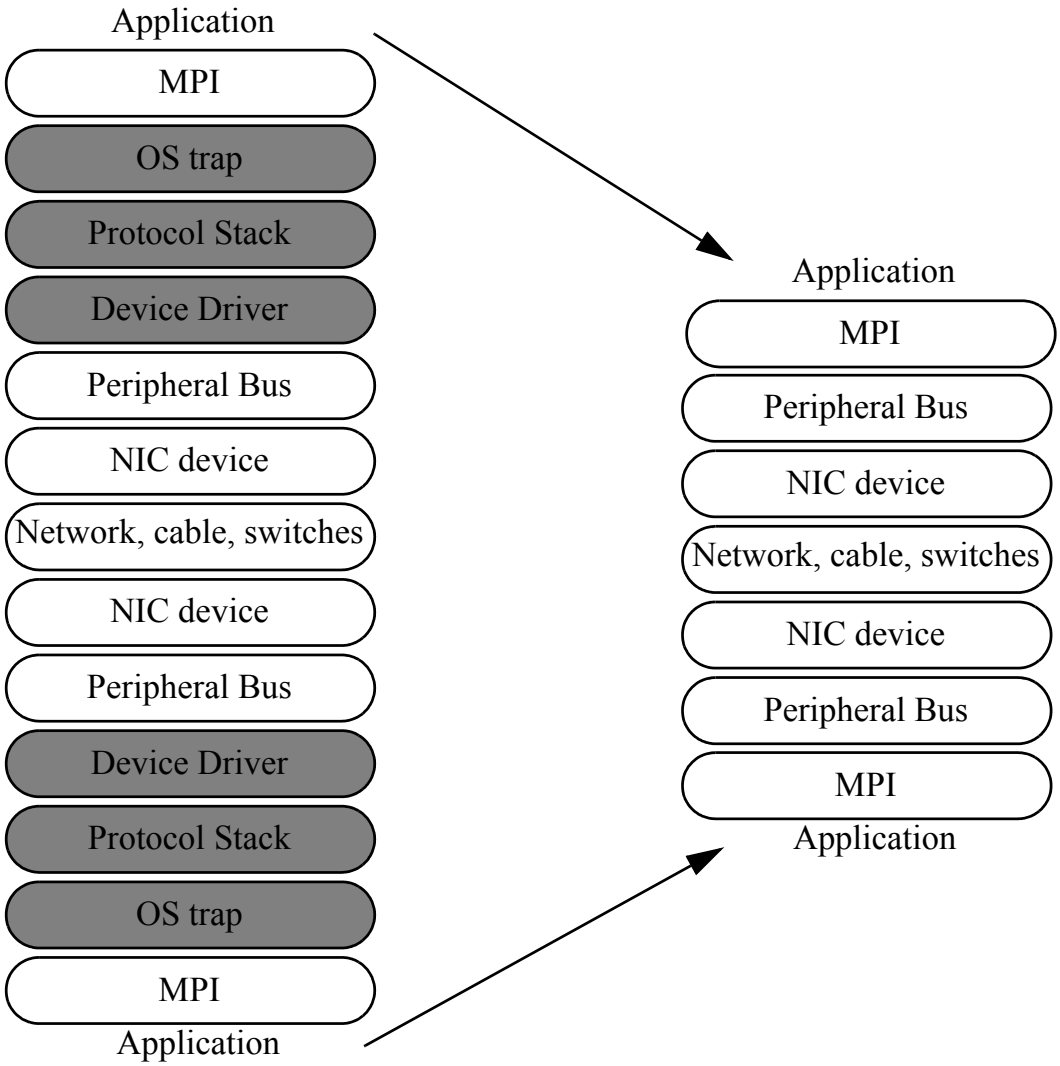


Figure 21: Latency contributing sources; no optimizations (left), regular SAN with user-level communication (right)

User-level communication

It is common understanding these times to avoid the OS with its protocol stack and the device drivers for regular communication. This is called user-level communication.

As the most time consuming task of the protocol stack is to provide reliable (and error free) communication the derived requirement for SAN hardware to get rid of the protocol stack is a hardware retransmission of faulty packets. All available SANs use hardware retransmission and so clear the need for a protocol stack.

Although, the OS and the device driver are involved during the setup and initialization phase, after the communication channels are setup no OS nor device driver intervention

are needed.

Regular SANs always use user-level communication. So, the sources of latency reduce to the communication library (MPI, i.e.), the bus the NIC is connected to, the NIC itself and the network with the cabelling and the switches (Figure 21, right).

User-level communication is enabled by mapping memory regions of the device to the user application's address space. Load and stores from these mapped pages by the user-level application (or the communication library) result in the corresponding bus read/writes to the device. OS and the device driver are needed to do the mapping [13], [14], but the system's bridge recognizes accesses to the mapped pages and forwards them to the device without OS interaction. All user-level communication works like this.

If a device can be accessed without OS interaction the device is attached to a single user-level process. In order to enable access to more processes, there are basically two ways:

1. Replicate the device
   Like in ATOLL [20], there are four replicated "Host Ports" to serve 4 different processes concurrently in order to satisfy compute nodes up to four way SMPs. The silicon area for one "Host Port" needs to be added for every process to be supported. This solution is inflexible in the number of concurrent processes currently allocating one "Host Port" but is easy to handle.
2. Virtualize the device
   Virtualizing a device means enabling a fast context switch mechanism. This mechanism needs to be so fast, that switching between contexts does not have a latency contributing impact. The advantage is that only one virtual "Host Port" is needed. Please refer to [42] for details on device context creation and handling and chapter chapter 5.4 "Hostinterface and EXTOLL block level communication" on page 114 for access mechanisms to virtual devices.

Optimizations to the communication library

The box titled "MPI" in Figure 21 refers to the entire communication software environment and includes the basic low-level Application Programming Interface (API) also.

Optimizations regarding latency can target the MPI implementation as well as the low-level API. Best results in optimization can be achieved if during the interface design of the hardware all requirements derived from the software are considered. Please find chapter chapter 4.2.2 "Seamless hardware/software interfacing" on page 70 and chapter

chapter 4.2.3 "Integrity and completeness of software required functionality" on page 71 for general methodology steps.

For opportunities offloading the MPI progress engine and the MPI tag matching task please refer to [43]. The MPI progress engine is responsible for the progress of non-blocking MPI commands. Currently most MPI implementations only progress if an MPI command is issued as no additional thread is spawn for concurrency. Also the tag matching task, that finds messages with a tag corresponding to the MPI command in the pool of received messages might be offloaded to the NIC processor. Offloading has the additional benefit of freeing the main CPU from communication tasks in order to accelerate computation.

Generally it can be stated, that all software levels need to be as lean as possible.

Location of the NIC (Box "Peripheral Bus")

Every bus bridge between the NIC and the main CPU adds latency. Therefore, regarding latency the NIC should be as close as possible to the main CPU.

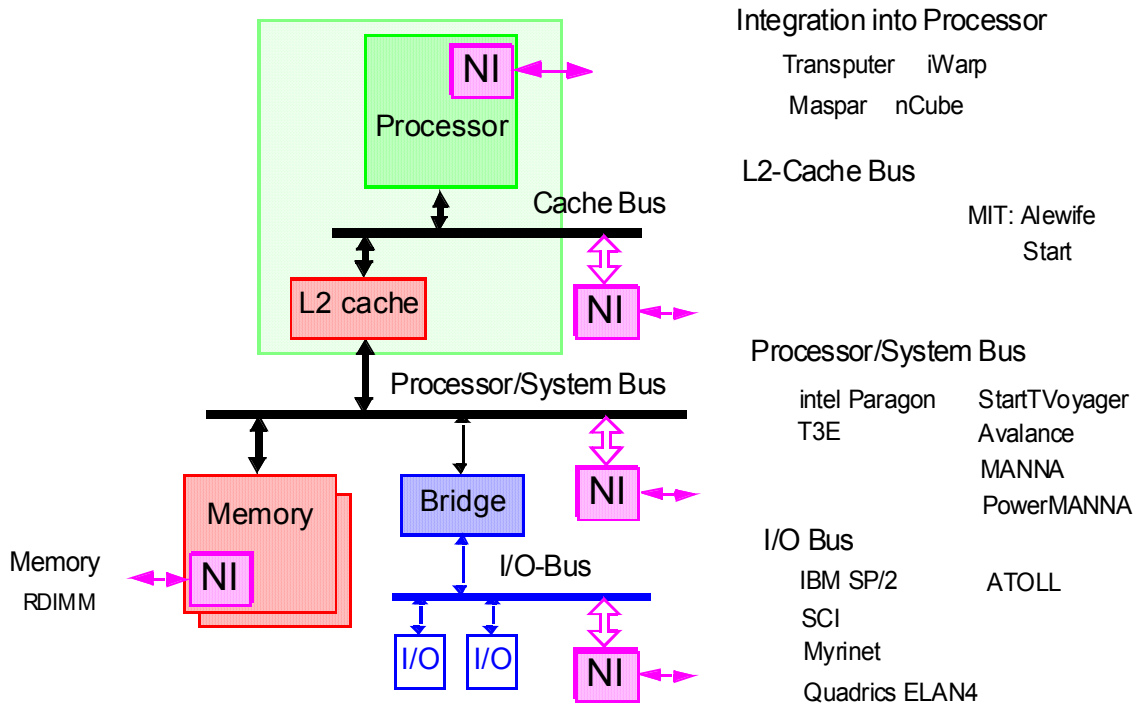


Figure 22: Network Interface Locations [31]

Figure 22 shows possible locations for the NIC in a commodity compute node. All commercially available SANs are located on the I/O bus. The reason herefore is the availability of open specifications like PCI, PCI-X or PCIexpress. The development of SANs

would target more closer interfaces to the main CPU, if there would be a processor independent bus specification of a system or cache bus available. Also a suitable possibility would be the AMD's Hypertransport processor interface, although it would limit the applicability of the SAN to AMD systems.

A very interesting aspect is the possible cache coherence of processor busses. Opportunities and enhancements are described in [44]. Cache coherence enables the SAN to deliver a virtual distributed shared memory system, where native processor loads and stores might be transparent remote loads and stores to arbitrary remote compute nodes.

NIC device

There are lots of opportunities reducing or hiding latency on NIC devices depending on the device's architecture. Here, all relevant operation principles of the computer architecture science can be applied. The more advanced the NIC devices become the more do they look like a superscalar processor: Concurrently working communication units, fast context switching, reorder unit, reservation stations, (communication-) instruction fetch and much more are on their way on to the NIC devices. One of the key elements of this work is the design space exploration and implementation of building blocks for a future interconnection network. Please refer to chapter chapter 5 "Architecture and Function Scope of Building Blocks" on page 78 for a detailed analyses of pros and cons of the selected architecture decisions.

However, in current systems the main factor for hardware communication latency is the amount of bus accesses to/from the NIC device to initiate a communication. Consider the PCI-X bus with an arbitration latency of between 350ns and 450ns. With this numbers it is from highest importance to reduce the amount of accesses needed for communication. In [32] we measured the resulting PCI-X bandwidth using different burst length. As accesses are not pipelined the high start-up latency reduces the available bandwidth dramatically depending on the bytes transferred with one access. Please refer

Figure 23 for a visualization.

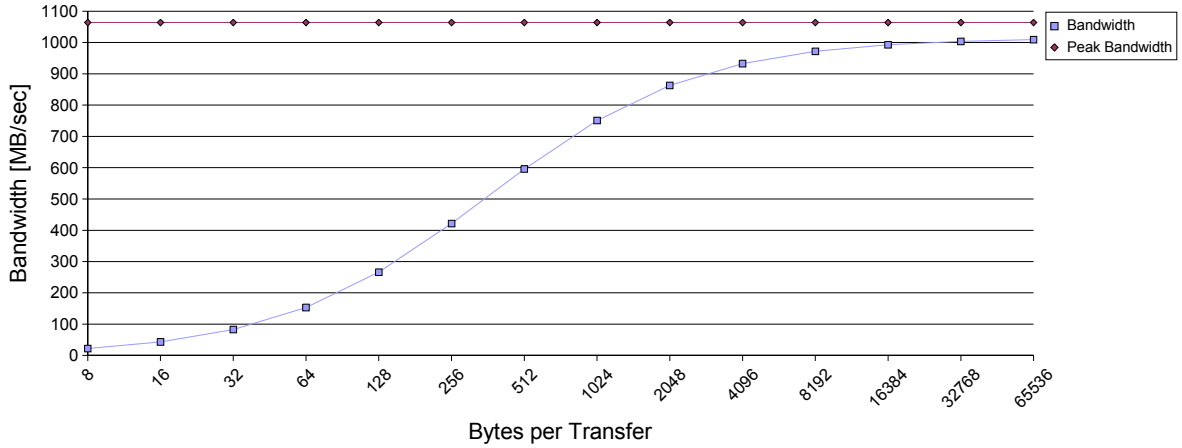


Figure 23: Bandwidth reduction due to high arbitration latency

Network, cable, switches

Similarly to the latency considerations for the NIC device, there are lots of ways to reduce latency on the network. The factor with the least potential to reduce latency is the cable. Typical cable delays are from 3-9ns per meter depending on the type of cable. Please refer Table 2 for details. As the cable length is very limited in a SAN, consider 30 meters, the cable delay is between 100 and 270ns:

Cable type	Delay per meter [ns]	Factor of c (speed of light)
Copper - twisted pair	9	0,37
Copper - Coaxial	5	0,66
Optical fibre	3,3	approx. 1

Table 2: Cable delay

Quite more potential can be found in the switch architecture. The evolution comes from the so called store-and-forward networks emerged to wormhole-routed networks and is basically now in the virtual-cut-through networks. These concepts have been thoroughly researched and can be found in [33]. The ATOLL SAN for example uses wormhole routing [22].

3.1.2 Bandwidth

The bandwidth, which is the most important quality metric for LANs, also must not be neglected, as data transmission needs to be completed as fast as possible in order to continue computation.

In fact, latency and bandwidth directly relate to the time needed to transmit a message. Derived from the below formula, latency is the offset and bandwidth the gradient of the straight line representing the needed time for a given amount of communication data:

$$Time(msglength) = latency + bandwidth \times msglength$$

For very small messages latency dominates, for very large ones the bandwidth. As in parallel computing small messages occur very frequently, because they are used to synchronize the parallel threads, the latency is especially important. Neglecting the bandwidth would let the network suffer during data set distribution of for example big matrixes. So, both metrics influence the network's performance drastically.

However, generating bandwidth is from the view of computer architects an easy task compared to lowering the latency. This stems from the fact that technology delivering bandwidth might be just multiplied in order to generate more bandwidth.

During benchmarking, when concrete numbers of bandwidth and latency are either unknown or should be verified, the definition bandwidth and latency somehow change. For latency, the time is measured to start-up a message of a given size and noted as latency of msglength. For bandwidth, the transferred bytes per second are measured for a given message size using the arithmetic average of a big number of messages and noted as bandwidth of msglength. In the following graphs exemplary benchmarks for start-up latency and send/receive bandwidth on MPI level are given. These measurements are taken during the hands-on tutorial of the International Supercomputer Conference 2005

(ISC05) by Holger Fröning.

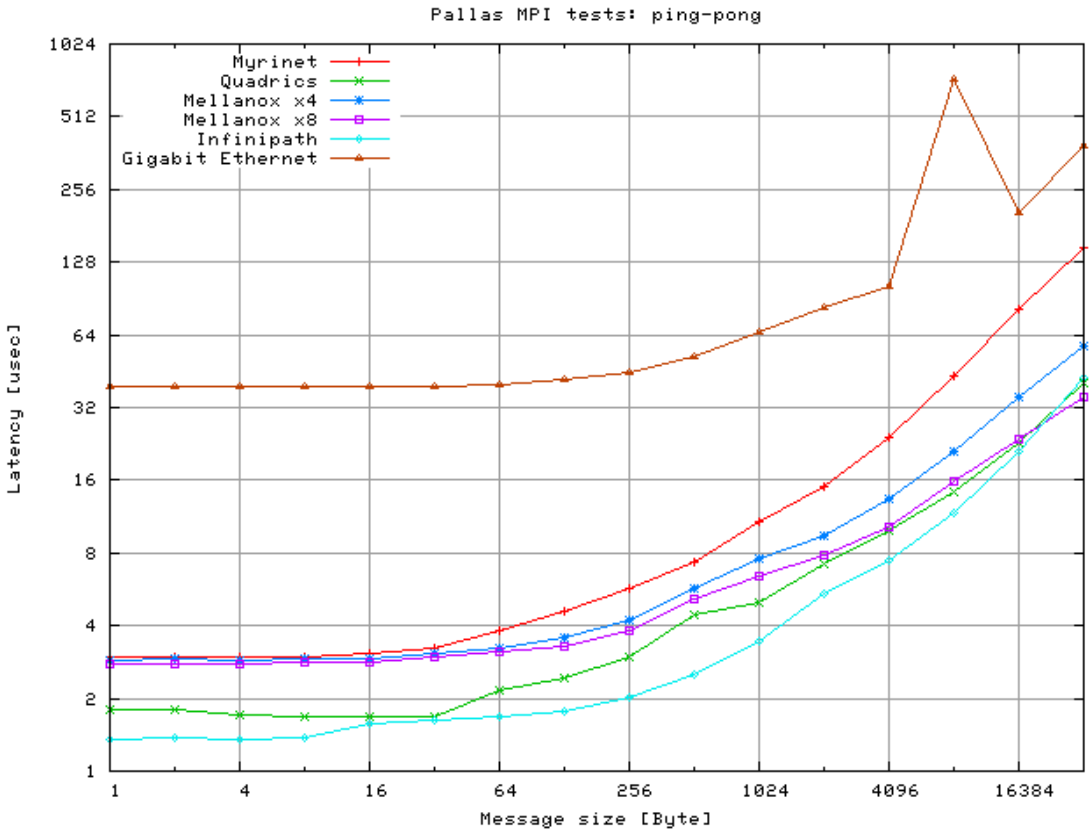


Figure 24: MPI Start-up latency for a given message size

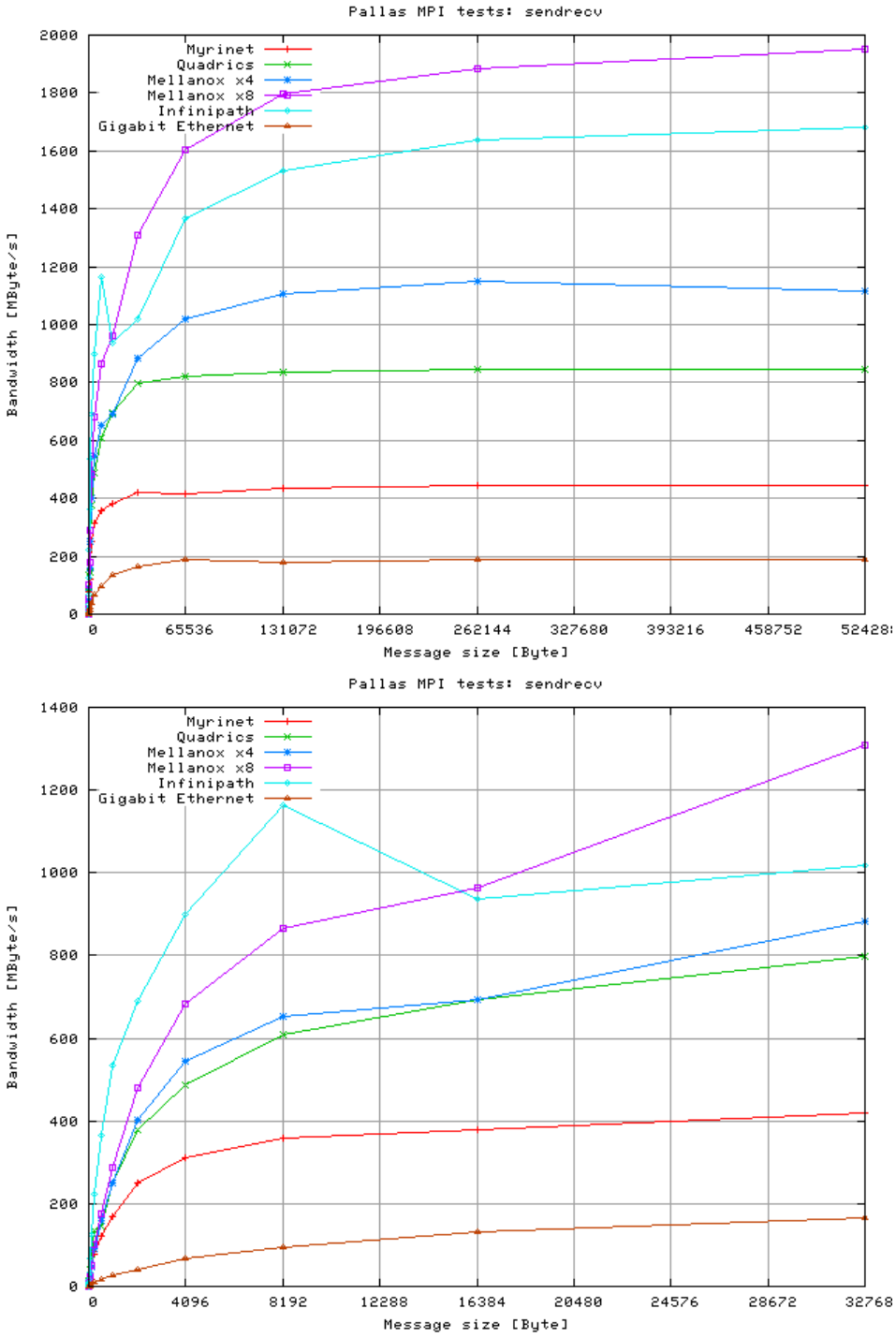


Figure 25: MPI Send/Receive bandwidth for a given message size (top) and in detail for smaller message sizes

The n/2 bandwidth metric

As stated above SANs rely on high bandwidths for small message sizes. Accounting this

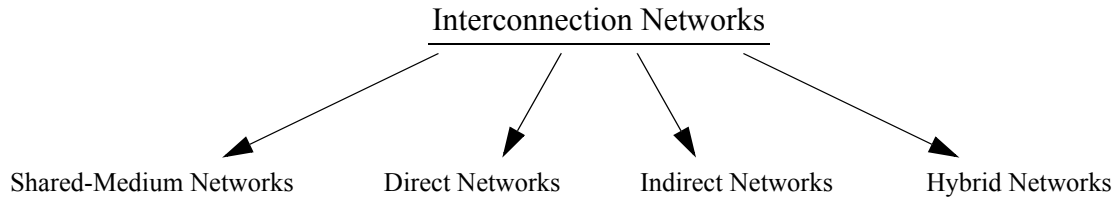
the n/2 bandwidth metric has been established. It notes the message size, where half of the peak bandwidth is reached. For the above measurements the following numbers calculate:

SAN configuration	n/2 bandwidth metric [Byte]
Myrinet	2048
Quadrics	4096
Mellanox x4	8192
Mellanox x8	32768
Infinipath	4096
Gigabit Ethernet	8192

Table 3: The n/2 bandwidth metric for several SAN configurations

3.1.3 Network Topology

The network topology defines how communicating partner are connected to each other. Most commonly used is the classification by Duato [35].]



Shared-Medium Networks

As the name impies all contributing communication partners share one physical medium, resulting in a blockage of the single communication channel during peer-to-peer communication. No further communication between the remaining communication part-

ners is possible, if two of them communicate peer-to-peer.

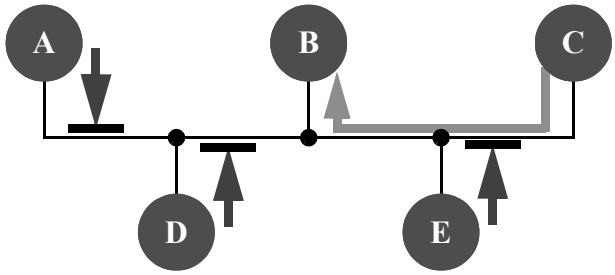


Figure 26: Communication of C->B blocks all other communication. A, D, E are blocked [17].

This problem aggravates the more communication partners are connected to the Shared-Medium and therefore, it prevents scalability.

However, communication patterns like multi- and broadcasts work perfect on this medium. Also, the "snooping" method, used frequently in cache coherence protocols, is only possible in Shared-Medium Networks.

In former times this network was also called bus; unfortunately misused by industry specifications like USB (Universal Serial Bus), which is indeed a pure point-to-point network in the class of direct networks.

Direct Networks

> *A direct network consists of a set of nodes, each one being directly connected to a (usually small) subset of other nodes in the network. Each node is a programmable computer with its own processor, local memory, and other supporting devices. A common component of these nodes is a router, which handles message passing among nodes. Each router has direct links to the router of its neighbors. Direct networks has been a popular interconnection architecture for constructing large-scale parallel computers. [36]*

Each communication partner brings in his own routing resource to the network. So each scaling step of communication partners also adds routing resources to the network. This is the reason why Direct Networks are supposed to scale very well. Usually SANs im-

plementing Direct Networks integrate the router into the NIC [11].

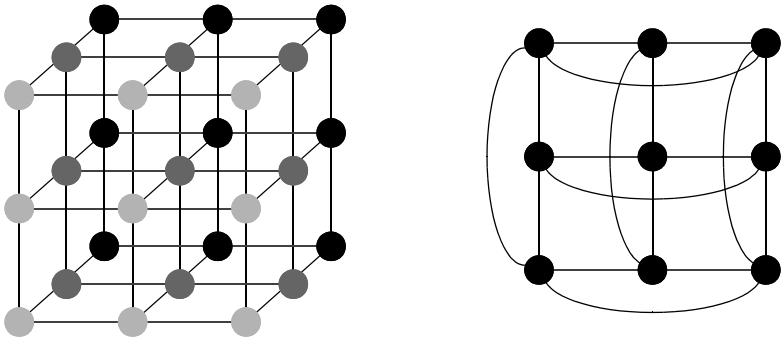


Figure 27: 3x3 cube (left), 3-ary 2-cube (right)[17]

In general, k-ary n-cubes are Direct Networks, where the n parameter is the dimension of the topology and the k parameter is the number of nodes along a dimension (also called radix). A 4x4 3d-torus would then be a 4-ary 3-cube. Binary n-cubes (2-ary n-cube) are also called hypercubes.

Indirect Networks

In contrast to the Direct Networks the Indirect Networks have switches or routers in between the communication partners. So, communication always is happening through a dedicated router or switch. Figure 28 shows a typical example.

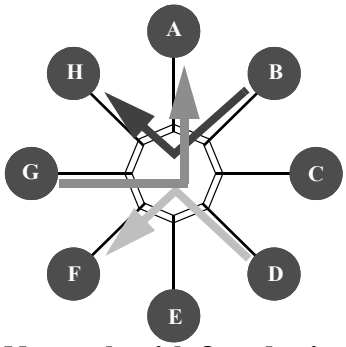


Figure 28: Indirect Network with 8 nodes in a star topology [17]

In order to keep the number of ports of the routers constant during scaling, additional stages of routers might be introduced (Figure 29). These subclass is then called multi-

stage networks.

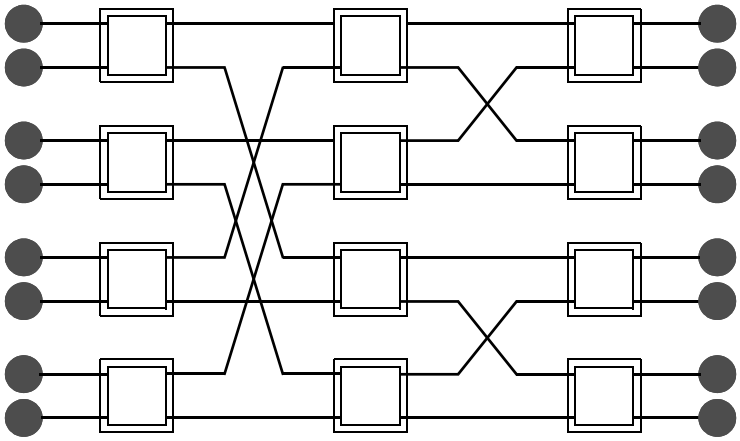


Figure 29: The butterfly topology as an example of multi-stage networks [17]

So, during scaling the number of ports per router remain constant, only the number of levels of the multi-stage network changes.

Hybrid Networks

In the class of Hybrid Networks all combinations of shared-medium, direct and indirect networks are concluded. Figure 30 shows an example of three shared-medium networks connected via an indirect network.

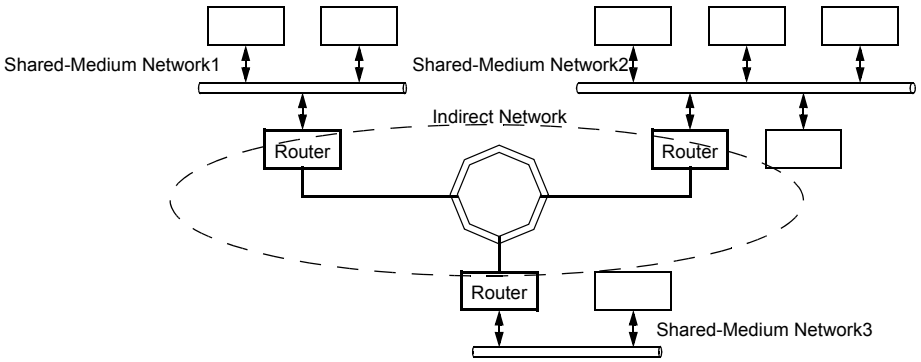


Figure 30: Hybrid Network consisting of a combination of Shared-Medium and Indirect Networks [17]

Important topology characteristics

There are lots of topology metrics available for most types of regular networks. Table 4 gives a rough overview.

Topology	Degree	Diameter	# of connections	Scalability	Symetrie
1D-grid	2	$N-1$	$N-1$	yes	no
1D-torus	2	$\frac{1}{2}\cdot(N-1)$	N	yes	yes
2D-grid	4	$2\cdot(\sqrt{N}-1)$	$2N-2\cdot\sqrt{N}$	yes	no
2D-torus	4	$\sqrt{N}-1$	$2N$	yes	yes
3D-grid	6	$3\cdot(\sqrt[3]{N}-1)$	$3N-3\cdot\sqrt[3]{N}$	yes	no
3D-torus	6	$\frac{3}{2}\cdot(\sqrt[3]{N}-1)$	$3N$	yes	yes
Hypercube	$\log_2 N$	$\log_2 N$	$N\cdot\log_2(N/2)$	no	yes
binary tree	3	$2\cdot(\log_2(N)-1)$	$N-1$	yes	yes
vollst. Vernetzung	$N-1$	1	$\frac{N\cdot(N-1)}{2}$	no	yes

Table 4: Some topology metrics [55]

However, only some of them have proven themselves as efficient for parallel computing. The reasons for this range from scalability issues over blocking behaviour to deadlock freedom or even manufacturability. All network/topology types for example that need to add ports to switches during scaling are totally omitted due to implementation economics.

From section chapter 2.2 "State of the art hardware support" on page 7, the viable topologies are clos networks, fat trees and tori/grids in various dimensions. Even for this limited amount of topologies some characteristics differ dramatically [54].

- Considering deadlock behavior and routing algorithms
- Considering blocking behavior
- Considering scalability
- Considering suitability for communication patterns (nearest neighbour, multicast, barriers)
- Considering reliability?

Considering deadlock behaviour and routing algorithms

A set containing a certain topology and a certain routing algorithm allow to make statements regarding deadlock freedom [50] [51] [52]. Considering the quaternery fat trees (like Quadrics's QsNet) all minimal routes are deadlock free. Quite more interesting are candidates in the class of direct networks, as these have proven to scale very well. Table 5 lists a set of combinations of topologies and routing algorithms that are deadlock free:

Topology	Routing algorithm	minimal routes?
Grid	Dimension-order	yes
Torus	Dimension-order	no
Torus	Dimension-order + Virtual Channels	yes
Torus	Up/Down	no
Grid	Up/Down	no

Table 5: Deadlock free combinations of topology and routing algorithm

There is a well understood relationship of routing algorithms on direct networks regarding deadlock freedom summarized in [18] 'Richard Sohnius, "Creating an Executable Specification using SystemC of a High-Performace Low-Latency Muli-Level Network Router", Diploma Thesis, Computer Architecture Group, 2005' and a description of routing algorithms in [37] 'Mondrian Nüssle, "Design and Implementation of a distributed management system for the ATOLL high-performance network", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005'.

Also described in [18] introducing virtual channels can improve the deadlock behavior by providing so called "escape channels".

Considering blocking behavior

In most network topologies, physical links between nodes are used for more than one route (In this context every pair of communicating nodes require a "way" through the network called route). In this case one link is used by two or more routes resulting in a reduction of the link or network bandwidth available for the communication partners. The metric that describes this fact is called "number of routes per link":

$$\#routes/link$$

As the generation of the routes is depending on the routing algorithm, only the combination of network topology and routing algorithm defines the #routes/link metric.

The switching behavior of the switching element in the interconnection network can increase the impact of the described effect for the communication performance. Using wormhole switching, message transfer A is blocked if one of the by the route defined links is already used by message B. This causes the message A being blocked. All links now used by message A are blocked also resulting in zero bandwidth for all links.

Reducing the maximum transfer unit (MTU) reduces the blocking penalty on the bandwidth until the switching behavior of the switching elements is "store-and-forward".

Another way to reduce the blocking penalty is to introduce virtual channels being able to reschedule the link after blocked by a non-transferring message [53].

However, the impact of the more-than-one-route-per-link circumstance on the link bandwidth still persists and must be observed during network scaling. Depending on the network topology the metric might vary if the number of nodes is scaled. Please refer the paragraph ’Considering scalability’ for further discussion.

Considering scalability

The terms scalability has been defined in a former published research paper [21]:

> *The term scalability can refer to many aspects of a network. Here, scalability regarding the size of the network, addressing and performance is considered. These three aspects are now treated in more detail.The major aspect regarding scalability in size is the required switching capability. Because every ATOLL card adds all the switching capability required for the integration into the network, the network is scalable without limitation. Regarding scalability of static networks, the node degree has to be invariant of the size of the network. The preferable topology for the ATOLL interconnect is a 2D-torus, and this topology has a fixed node degree of four. Thus, a typical ATOLL network with a 2D-torus topology has no limitations regarding scalability of network size. For a source-path-routed network the major issue regarding scalability is the maximal length of the routing string. In fact, for the ATOLL network the length of the routing string is practically unlimited (the length of the routing string is limited to 2^{27} hops)and consequently provides adequate support for a scalable network.*

> *Regarding the scalability of performance, the first issue is the end-to-end latency. Latency increases with each additional hop by 90 nsec, as mentioned before. For example, the network diameter of a 2D-torus with 256 nodes is 8. Thus, in worst case the start-up latency is increased by 7 hops or 0.63 usec, which is negligible compared to other latencies in the communication path. The bandwidth is not affected by the number of hops due to the pipelined crossbar as switching element. Unlike interconnects using bus structures as building blocks, here the full bandwidth is available to the four bidirectional channels in and out of the NIC. The only issue affecting bandwidth is blocking of messages. Long messages can block a lot of intermediate crossbar stages. This problem can be diminished by reducing the maximum size of the packet length. Per-channel FIFO input buffers in front of the crossbar reduce the effect of blocking and provide a smooth transition from wormhole switching to store-and-forward in case of a heavy loaded interconnect.[21]*

Considering suitability for communication patterns (nearest neighbour, multicast, barriers)

Another new metric must be defined depending on the network topology and the routing algorithm called "distance between nodes". This is the number of switching elements between two communicating nodes impacting the latency. A similar effect is known from wide area networks (WANs) where the latency is highly depending on the destination. If the distance between nodes is fix for all routes, which is the case in the ’Myrinet by Myricom’ the communication pattern especially nearest neighbour pattern has no impact at all in contrast to a grid or torus topology where the distance per node and the corresponding latency is highly varying. Here, nearest neighbour patterns can positively influence network performace.

Other communication patterns like barriers or broad/multicasts can be easily mapped on tree like network topologies as there search algorithms like depth-first or breadth-first search exist. These algorithms are required to implement a decentralized management of broadcasting patterns. Regarding a torus it is not easily possible to broadcast a message without sending and receiving it only once.

This is the reason why the highly sophisticated network architecture of chapter 2.2.5 "IBM Blue Gene BG/L" on page 23 describes several networks with tree and tori topologies for the different communication patterns.

Considering reliablility

Reliablility is of increasing importance the bigger the systems are. Every component in the system contributes with his chance of failure to the total system failure chance.

Therefore, failure models for each system entity has to be defined:

- What happens if a switch fails?
- What happens if a node fails (power-down)?
- What happens if the nodes OS crashes?
- What happens if the user application crashes (protection of remote memory available, message reception in wormhole switched networks)?
- What happens if a link fails (completely or increasing transmission errors)?

For each failure model the impact on the transmission capability of the network has to be considered. Especially for switch or link failures the availability of alternate routes is important. Again, tree like network topologies differ dramatically from grid like topologies as the availability of alternate routes becomes more and more difficult the nearer the failure to the tree's root is. Generally, it can be stated that topologies with some kind of centralized functionality, like a tree's root, are more sensitive to failures as a failure of exactly the root has to be managed in a special way. Grid or tori topologies are less sensitive in this area as they have no centralized functionality.

3.1.4 Supported Parallel Programming Model

In general, there are two parallel programming paradigms "message passing" and "shared memory". The applicability is defined by the node's and the network's architecture. We will see that the NIC location with regard to possibility of network wide cache coherence is the key issue in this discussion.

Please refer Figure 31: ’General parallel communication architectures [38]’.

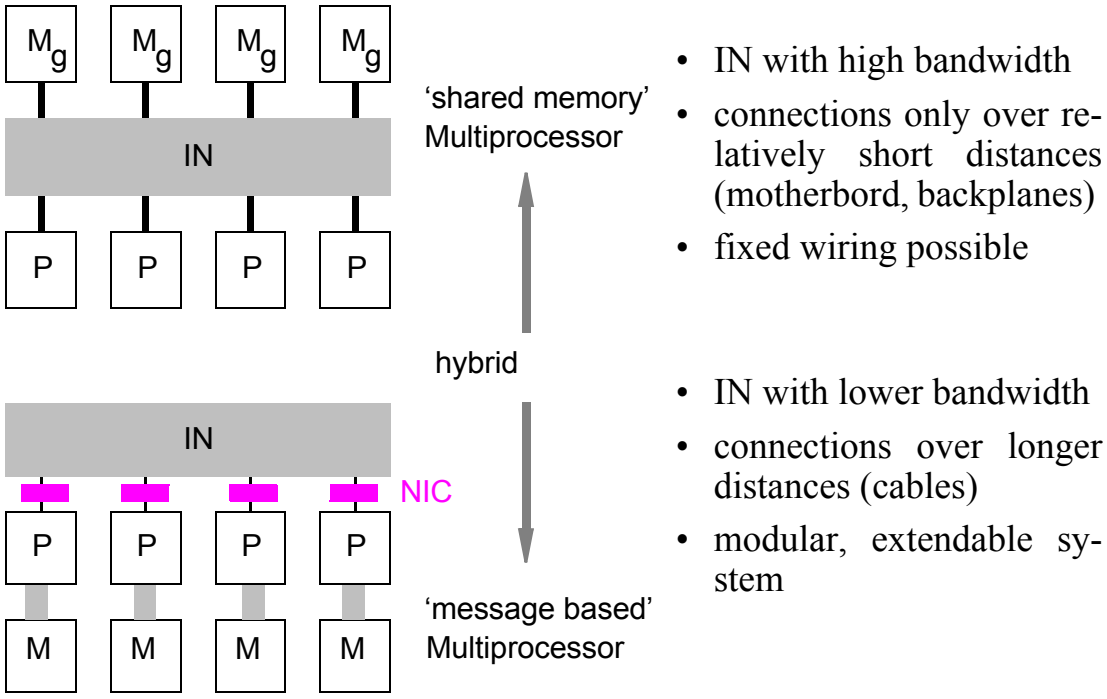


Figure 31: General parallel communication architectures [38]

Distributed Memory (Message Passing)

In distributed memory architectures remote memory can only be reached via explicit communication instructions. Message passing is used to issue a communication. Depending on whether only one or both communication partners need to issue a communication instruction we differentiate "one-sided communication" from "two-sided communication:

Two-sided communication uses so called "send" and "receive" instructions, most easy to implement in hardware, as the send call contains the memory address to send in the process context and the receive call the receive memory address respectively. User processes reside in their virtual memory context only and therefore, are not aware of the actual physical address. Memory paging, protection and segmentation features of the operating system can swap pages in physical memory without the knowledge of the process itself.

One sided communication uses so called "put" and "get" instructions being able to put or get memory contents of remote processes without the remote process’s knowledge.

The communication instruction therefore, must contain local and remote memory addresses.

Shared Memory (Threads using shared pages)

Using a shared memory architecture implicit communication via native processor load/store instructions is possible. Cache coherence can be obtained as the network is able to snoop at the processor's cache bus. Mixed architectures like clusters of SMP nodes might provide both communication paradigms; shared memory within the SMP node and message passing between the SMP nodes.

3.1.5 Cost

Especially for cluster systems where the cost/performance ratio was key to the commercial success the cost for the network must not be neglected. Therefore, the metric cost per network port which is cost of NIC plus router divided by the number of ports (NIC+router/#ports) has been introduced. During the development of the ATOLL network the cost factor has been taken into account leading to the decision to keep all message data in the main memory of the node avoiding costly SRAMs on the NIC [20]. Due to the recent SRAM cost reduction and the advances process technologies being able to integrate SRAMs on the same die this decision has to be reevaluated for future developments. The ATOLL network also was the only network integrating the routing capability into the NIC avoiding dedicated switching hardware further reducing cost in the above metric.

3.1.6 Communication to / Location of the Network Interface Controller

Figure chapter 22 "Network Interface Locations [31]" on page 34 shows possible locations of the NIC. There has been research as well as commercial solutions for every possible NIC location. Unfortunately all architectures with the best location with regard to the communication features - directly integrated into the node's CPU - died as they has been "before their time". They have been developed in a time where the communication ability of the system was not as demanding to performace as the compute power itself. Nowadays, CPU die area is growing to a point where additional on die cache size is not contributing to computation performace in a relevant factor as it has been in former days. Therefore, now might be the right time to dedicate any additional die area to implement

communication features.

There are two reasons why to put the NIC as near as possible to the node's CPU:

1. Reduction of latency
   Every bridge adds a significant portion to the latency.
2. Possibility of participating at the cache coherence protocol
   Placing the NIC at a position where it might participate at the cache coherence protocol enables true shared memory communication in distributed memory systems. For further discussion, please refer to [44].

The is another reason not placing the NIC (like ATOLL) on the CPU's bus: Intellectual property of bus protocols and processor interfaces hinders developments in this area. As well as quite frequently changing processor interfaces eliminating the possible commercial application due to small product life cycles.

Features	PCI-X	PCI-Express PCIe	Hypertransport HT
number of signal lines	64 + 39 = 101	4,8,16,32,64	26,36,57,105,199
multiplexed operation	yes Address/Data	yes	yes, message orient.
data width	32/64 bit	2,4,8,16,32 bit	Link width 2,4,8,16,32 bit
usage	I/O-Bus Peripheral-Extension	I/O-Bus Peripheral-Extension	I/O-Bus** Peripheral-Extension
operation mode	fully synchronous, clocked	source synchronous 8B/10B coded data	source synchronous 1 x clock pro Byte
clock frequency	0 - 33/64 MHz 100/133 (266*) MHz	2,5 GHz	200 - 800 MHz (1-1.6GHz)
signal transmission	CMOS-Level reflective wave signalling	CML-Level serial, differential	U-Level 600mV NRZ, serial, differential
data transmission	CMOS-Level clock synchronous	coded embedded clock	DDR double data rate packetized
termination	no	100-110 Ohm	100 Ohm on chip, overdamped
burst transfers	yes, many modes 4x burst, arbirary length	yes, message transfers	yes, comand + message transfers
Split Transactions	yes	yes	yes
max. Bandwidth	533MB@66MHz-64bit 1GB@133MHz-64bit	2x2,5Gbit/s@2bit 10GB@32bit	0,2GB@200MHz-2bit 12,8GB@1600MHz-32bit
max.no of devices	Bridge + 4,2,1 Devices 1 I/O-Device @133MHz	point to point	point to point, bidir.
max length of signal lines	aprox. 10cm	aprox. 3-10cm at FR4***	aprox. 3-10cm at FR4
Standard	Industrie (Intel) + IEEE	Industrie (Intel) + Konsortium	Industrie (AMD) + Konsortium
Spec. page no.	aprox. 220	aprox. 420	aprox. 330
Web Infos	www.pcisig.org	www.intel.com	www.hypertransport .org

*) DDR *double data rate transfer* **) extended version for CPU Interconnect with Cache Coherency-Protocoll ***) PCB material

Figure 32: Overview of commodity compute node interfaces [38]

4 Methods - Approach

The approach and the methods applied during the course of this work are key issue for efficient developments in the field of high performance architectures and their leading edge design (ASIC) implementations. Raising complexity in communication architectures as well as in deep submicron ASIC design require massive mutations in education, methods and Electronic Design Automation (EDA) tooling.

Coping with the complexity of this project absolutely required these issues and therefore, they are mentioned in this work. Starting with [4.1] ’Providing the basis for efficiency in hardware design’, where student education and EDA tooling are covered, I will continue with [4.2] ’Hardware/Software Codesign and Cosimulation’ describing innovative approaches for concurrent hardware and software development as well as the concept of "The Concept of Virtual Hardware Prototyping" on page 63. In chapter [4.3] ’FPGA based ASIC prototyping’ project related demands not suitable for simulation are introduced and finally, chapter [4.4] ’Physical design impact of UDSM designs’ addresses the influences on the hardware design methodology.

4.1 Providing the basis for efficiency in hardware design

The chapters [4.1.1] ’Education’ and [4.1.2] ’EDA tooling’ summarize the efforts in a abstract way, going into detail in section [4.1.3] ’Realization - SEED Project’.

4.1.1 Education

> *It’s not a secret that for efficient use of most EDA tools an amount of experience is needed, that students typically don’t have. Teaching these tools practical courses or exercises including the challenges imposed by the latest technological advances requires a thorough understanding of the underlying methodologies and up to date practices. Students have to deal with a lot of new skills at the same time and the complexity of the things to be considered can easily lead to frustration. Therefore, most universities try to avoid practical work in the field of backend (physical) design, work with simplified methodology and older technology, or even leave the ASIC domain to favor FPGAs [5].*

Due to the complexity and required EDA tooling most german universities skip the field of physical design needed for ASIC implementation. This tends to result in a lack of human resource capable of designing complex ASICs. Universities still doing ASICs make use of their research associates for backend tasks resulting in an improper and ineffective workload. In addition, university education in this field can not stand with the requirements of industry.

An EU integrated project proposal [7] targeted at the 6th EU framework program (FP6) has been initiated by IMEC during the course of this work (March 2005) and focuses exactly this problematic nature. Some relevant sections of [7]:

> *This proposal addresses the need for complementary measures, in particular: Stimulation actions aim at increasing the interest of students and improving the quality of education in SoC design. This will be done through IPs that emphasize research carried out by, and training of, students in SoC design.*
>
> *Implementation skills are slowly decreasing or even disappearing in many universities. This is shown by a decrease in the take-up of advanced SoC design tools by universities within the Europractice IC service and in a steady decrease in the number of circuits submitted for prototyping by these universities.*
>
> *Physical design in general, moving to nanoelectronics, has become very complex (SoC design, RF design, deep submicron technology, heterogeneous systems, etc.). As a result of the continuously increased complexity only a small number of 'top' universities can cope with the most modern nanoelectronics design techniques. These are the universities that perform research and education that is relevant to industry. Many of the 550 universities are lagging*

behind the technology curve and perform activities that are of less use to industry in this respect.

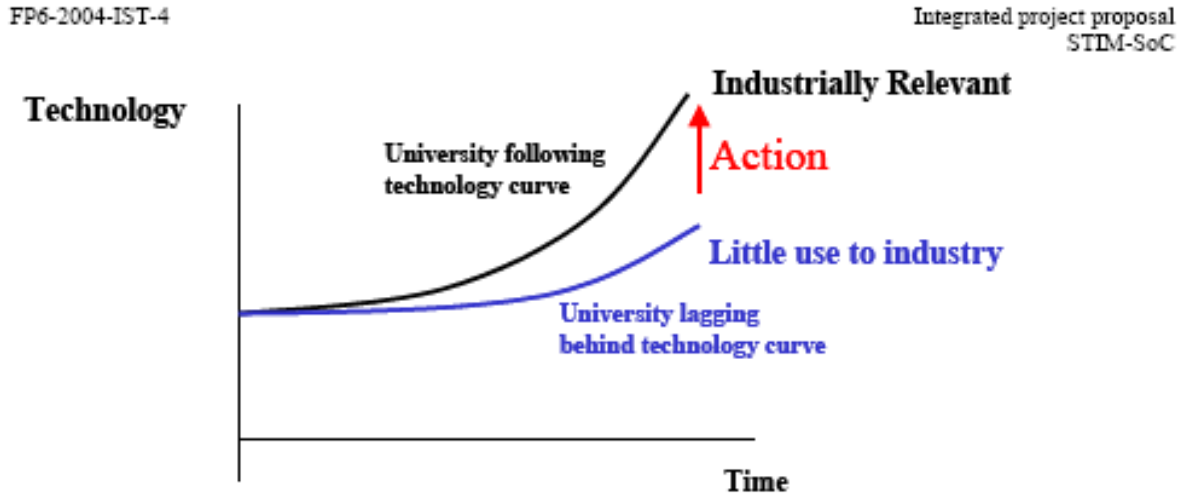


Figure 33: Univesities lagging behind technology [7]

It is the defined aim of the Computer Architecture Group (LSRA) to prove viability of a system architecture by designing a complete implementation in hardware.

During the last five years I worked on structural enhancements increasing the value of graduate students working on LSRA research. This covers several different activities:

- Improving Soft Skills (Teamwork, ability to communicate difficult contexts)
  Communicating with our industrie partners it turned out that industry requires more and more soft skills in addition to technical knowledge. This springs from the team aspect required for nowadays complex systems. Encouraging students presenting their work once a week in a common meeting together with research associates and assistants was one of the ways providing a platform for communication on different levels, please refer Figure 34: 'Students in the focus of educational endeavours'.
- Shift non-research work from research associates to graduate students of different experience levels (Diploma thesis, research assistants, project work...)
  Delegating non-research work from research associates to proper educated students frees resources that now can be used for deeper research. This requires proper education, please refer to [4.1.3] 'Realization - SEED Project'.
- Enable different ways of doing lessons to facilitate project oriented education (scandinavian way of teaching [8],[9].

- Force students to work independently, concurrently providing the permanent option of lecturer support (away from details to methods)[9].
  The students are supposed to learn how to cope with EDA tool complexity, especially avoiding tool focused education. The methodology is key, scripting is minor detail, to be learned on the job. During my lecture 'Semicustom Design Flow (SCDF)' the students learn what to do and why, providing them with some details of tool operation. Deeper tool details are discussed after they dig deeper in the manuals in a separate more practical lesson. This is the way to induct independency in working. They read manuals and consult faculty for the why and for what.
- Tightening relations to industry partners providing them with highly educated students for internships.

Other graduate students and graduates that passed our curriculum were rated by the Group Leader of 'Automotive Driver Information ASSP Design' at Toshiba Electronics Europe GmbH. He said:

> *"It is ideal for us, if students get an overview of the entire VLSI process chain (from front-end down to testing prototypes). This is outstanding for Mannheim's graduates, I didn't have the opportunity at my university."*

Figure 34 shows the different influences on a student's education

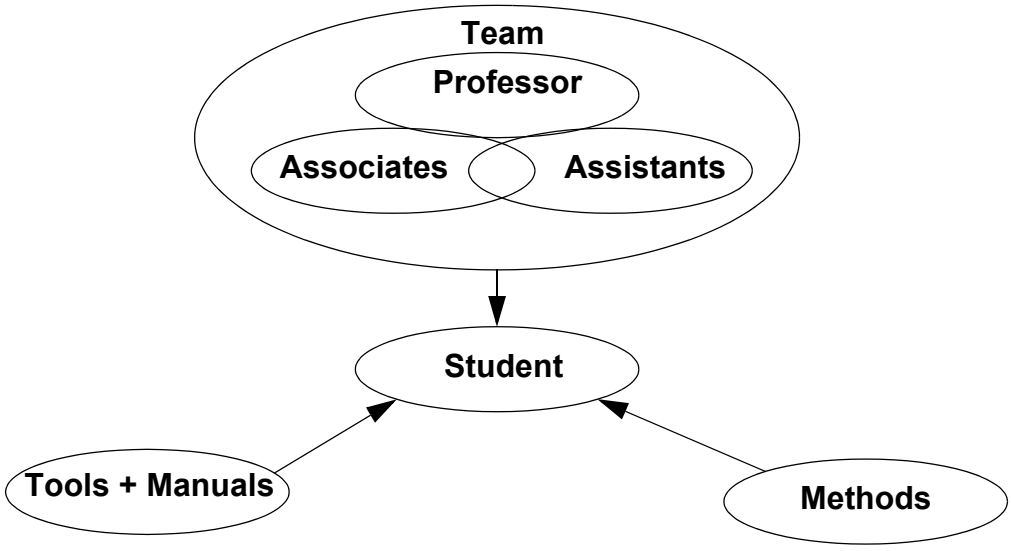


Figure 34: Students in the focus of educational endeavours

4.1.2 EDA tooling

> *We understand the importance of the design environment for our business and recognize Cadence's experience in electronic design, electronic design automation and design environments. We wish to improve the quality and efficiency of our design process by utilizing Cadence tools. Cadence will use its experience in design environments to help us by creating and maintaining design environments focused on our requirements. This will help us to reduce internal efforts and allow us to focus our efforts on education and research. By virtually integrating Cadence engineers into our Computer Aided Design ("CAD") activities, we will benefit not only from the experience of the specific engineers, but also from a direct channel to Cadence's know-how network. This will typically result in a more reliable design environment designed for flexibility and also in a faster and smoother integration of technology enhancements. For us, this will provide the possibility of designing higher quality designs, within shorter design cycles. Additionally we got access to Cadence's Sourcelink, a to Cadence users restricted documentation database that contains application/technical notes, known problems and workarounds, as well as customer focused solutions from Cadence VCAD services [5].*

A leading edge design environment supported by Cadence VCAD services [6] enables us to implement highest performance on newest technologies (below 130nm), not possible with standard university tooling like Europractice. Also supporting us with engineering knowhow we have been able to meet industry like deadlines for volume production runs using 180nm UMC technology. Cadence also lets us change the EDA tool environment depending on current research projects enabling a perfect starting situation for our architecture developments.

> *It is observed that providing access to CAD tools and ASIC prototyping is not sufficient any longer. [...] Installing and maintaining those tools requires a high investment in staff, teaching SoC design moreover also requires a high investment in expert staff members.[7]*

4.1.3 Realization - SEED Project

> *Together with Cadence Design Systems and SEED2002 we took a step forward. Beginning with this semester (mid of october to mid of february[2002]) we teach a lecture and practical course named 'Semi-Custom Design Flow' (SCDF), where our students learn the methodology of high-speed nanometer ASIC design by the use of the most innovative EDA tools. These include Physically Knowledgeable Synthesis (PKS), FirstEncounter, Nanoroute, CeltIC and Fire&Ice supported by Cadence Design Systems. This lecture and practical course affiliates perfectly to the lecture 'hardware design', that has been held for years now and covers (architectural) system, interface and digital circuit design using Verilog HDL [5].*

SEED is supposed to be understood as a precursor for further and tighter relations to leading industry enabling university researchers implementing leading edge designs. Covering education in the field of electronic design it 'seeds' the origins of the ASIC Competence Center (ASICCC), a collaboration of the Universities of Mannheim, Heidelberg and Kaiserslautern. All contributing universities undertake efforts to create or redesign curricula for lectures, practical courses and project work in the field of electronic design and electronic design automation (EDA).

SEED related Tooling

In Table 6 the EDA tools used in the context of SEED associated with the sites are listed.

Table 6: SEED EDA Tools

Tool/Package	University of Mannheim	University of Heidelberg	University of Kaiserslautern
SemiCustomWorkBench	X		
BuildGates(R) Synthesis	X	X	
First Encounter	X	X	
Nanoroute	X	X	
PKS	X	X	
Virtuoso(R) Schematic Composer		X	X

Table 6: SEED EDA Tools

Tool/Package	University of Mannheim	University of Heidelberg	University of Kaisers-lautern
Cadence(R) Analog Design Environment		X	X
Spectre(R) Circuit Simulator			X
Cadence(R) AMS Designer Environment	X		X
Cadence(R) AMS Designer Simulator	X		X
Cadence(R) Chip Assembly Router	X		
Diva(R) Physical Verification and Extraction Suite	X		X
Virtuoso(R) Layout Editor	X	X	X
Virtuoso(R)-XL Layout Editor	X	X	X
Assura(TM) Design Rule Checker		X	
Assura(TM) Layout Vs. Schematic Verifier		X	
Assura(TM) Parasitic Extractor		X	
Assura(TM) Graphical User Interface Option		X	
Assura(TM) RCX Field Solver Option		X	
Incisive(TM)	X	X	
Route Accelerator Multi-Threaded Route Option	X	X	
VoltageStorm (gate & transistor)	X	X	
SI Timing Convergence with VoltageStorm	X	X	
SI Timing Convergence	X	X	
Fire & Ice QX (gate & transistor)	X	X	
Cadence(R) SoC Encounter	X	X	
Advanced Package Engineer Expert	X		
Advanced package designer expert	X		
Conformal Ultra	X		
CeltIC NDC	X		

Table 6: SEED EDA Tools

Tool/Package	University of Mannheim	University of Heidelberg	University of Kaisers-lautern
Fire&Ice Nanometer Option	X		
SignalStorm Library Characterizer	X		X

SEED related lecture 'Semi Custom Design Flow' (SCDF)

As described in [5] one of the first actions after the start of the SEED project was the establishment of the completely new created lecture 'Semi Custom Design Flow (SCDF)'. Here students learn the methodology of physical design (also known as the backend task) of high-speed nanometer ASIC design by the use of the most innovative EDA tools. These include Physically Knowledgeable Synthesis (PKS), FirstEncounter, Nanoroute, CeltIC and Fire&Ice supported by Cadence Design Systems with a strong commitment to hands-on practical exercises focused on specific real-world projects. This lecture and practical course affiliate perfectly to the lecture 'hardware design', that has been held for years now and covers (architectural) system, interface and digital circuit design using Verilog HDL.

The enhancements in student education have already been reflected by industry, one quote of Dr. Gunter Strube, Senior Services Manager, Cadence Design Systems says:

> *„The SEED-2002 VCAD project with the ASIC Competence Center is a role-model for partnerships: open, result oriented, and efficient. This perfectly matches our VCAD culture. We are delighted about the level of expertise and commitment from the University staff. Students with such excellent education are highly valued at Cadence, during and after their studies."*

4.1.4 The System Realisation Bi-Cone

System realization is a very complex task where lots of project related constraints have to be considered. The idea to represent this task in a bi-cone springs from the famous "Platform Based Design" idea of Prof. Alberto Sangiovanni-Vincentelli. The content of the bi-cone drawn in Figure 35: 'ATOLL system realisation cone' is derived from the experiences gained during the ATOLL system development of the Computer Architecture Group.

The journey starts with the idea of what is beeing to be developed, nothing is fixed. Just the idea. This situation is extremely rare, usually there are constraints limiting the design space. However, for the task of imparting knowledge this idealization is acceptable.

The design space spans during the process of continuous refinement of the specification and further implementation. At some stage, when all considerable design spaces are explored and decision are made the diversity decreases and finally collapse to a single point, the final implementation. There are ways that lead to road blocks of which some of them might be detoured by a work around. There are also ways that lead to "complexity holes" where you might get lost in steadily increasing complexity. Sometimes one can not decide if he is on the "right" way, there might be some fog on its way with poor visibility. It is the challenge to find a path through this space that is possible, feasible,

viable and, with respect to the current business environment, efficient.

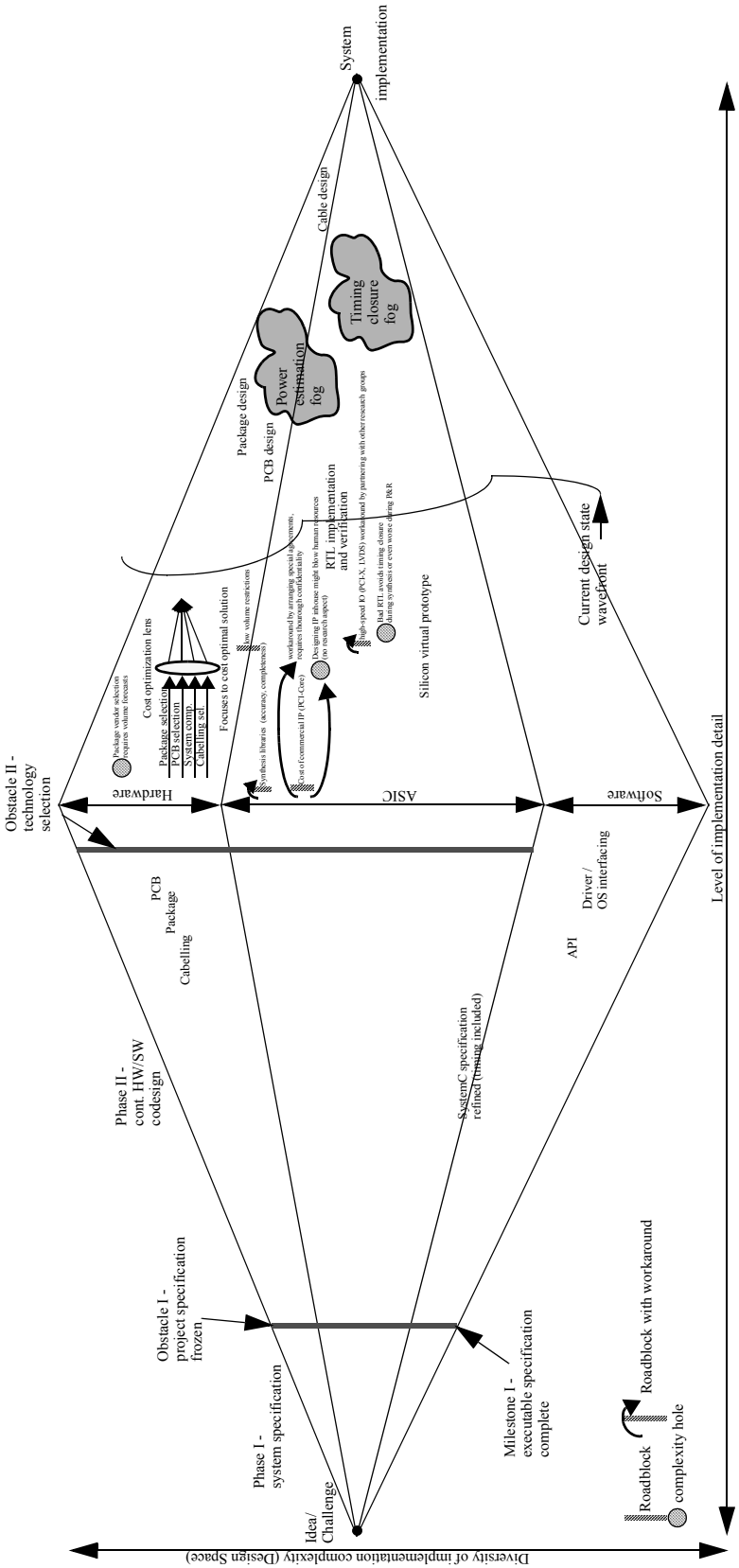


Figure 35: ATOLL system realisation cone

4.2 Hardware/Software Codesign and Cosimulation

Computer Architecture and System Design are not limited to hardware related issues, but also cover the entire system (system defined as a complex structure of software, firmware and hardware) driven by a challenge, verified by the correctness of the functionality and rated by the time required to solve the challenge as well as the related cost (knowhow, material, implementation, maintenance, operation). Estimations without an assessment factor or quality metric are inaccurate, so the goal is to have an exhaustive description of the entire system, executable at its best.

> *Each set of estimated quality metrics is compared to the given requirements and the implementation that satisfies the requirements optimally is selected.[1]*

As complex systems in most cases do not have a closed solution, simulation is the only way to gather reliable and significant characteristics of the system. The most profound issue here is to have models that reflect reality in a proper way. What proper is, has to be defined by the system architect. The accuracy with regard to the real system then defines the quality of work of the system architect.

Beside rating a system there are couples of other advantages HW/SW Codesign and Cosimulation has:

1. "Concurrent development of hardware and related low level software" on page 70
2. "Seamless hardware/software interfacing" on page 70
3. "Integrity and completeness of software required functionality" on page 71

The Concept of Virtual Hardware Prototyping

Virtual Hardware Prototyping is a special way of HW/SW Codesign focusing on software transparency and host-device interaction. The term 'Virtual Hardware Prototyping' refers to the fact that the software part is natively executed on the target host system as the hardware part -the device under test (DuT)- is simulated.

Ideally Virtual Hardware Prototyping is 100% software transparent. As there is always some software communicating with the HW directly (kernel-level device drivers) it is impossible to achieve 100% transparency. To keep the above advantages as effective as possible it is from highest importance to push transparency to a maximum. As more SW

needs to be changed during the transition from virtual to real HW as more loss of advantages can be observed. The reasons are the potential insertion of all kind of human errors as well as the degradation of closeness to reality, that obviously alleviates all advantages of HW/SW codesign and cosimulation.

The ATOLL test case is a perfect environment to explore and demonstrate the mightiness of this concept. Starting from a working hardware/software environment, we developed a SystemC representation of the complete ATOLL device functionality. Only minor changes in the kernel device driver has been made to get a fully working virtual hardware prototype of the ATOLL chip. All software layers above the kernel device driver remain unchanged [45].

This demonstrates the power of this concept, enabling the software development teams to influence the hardware functions and interfaces even during the design exploration phase. Also test software might be written prior to silicon delivery. As fewer changes has to made during the transition from the virtual hardware prototype to the real silicon device, the closer is the virtual prototype to reality.

As our research group is currently working on the development of the next-generation SAN called EXTOLL, we are using this concept now from scratch. The software development team starts writing the API concurrently to the design of the hardware pointing out obstacles and possible workarounds in interfacing and function of the hardware. Changes early in the design cycle are easy to handle and are more effective than costly software detours needed if silicon is already delivered.

The Verification Realm

Hardware Verification can be coarsely divided in two large areas (please refer also Figure 36: ’The Verification Realm’):

- Formal Verification,
  using formal mathematical methods

and

- Functional Verification,
  using user defined test patterns and user defined return values.

Furthermore, Formal Verification can be divided in Equivalence Checking and Model Checking.

Equivalence checking focuses on the boolean comparison of two netlists, RTL-netlist or netlist-netlist and has been introduced to double check the work of netlist transforming EDA tools. This can be Clock Tree Synthesis (CTS) or In-Place-Optimization (IPO) steps or even logic synthesis. So, Equivalence Checking can be omitted if EDA tools become more reliable and mature. There is no functional verification nor crosschecking against specifications.

"**Model checking** is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not. Extremely large state-spaces can often be traversed in minutes." [Carnegie Mellon, http://www-2.cs.cmu.edu/~modelcheck/]

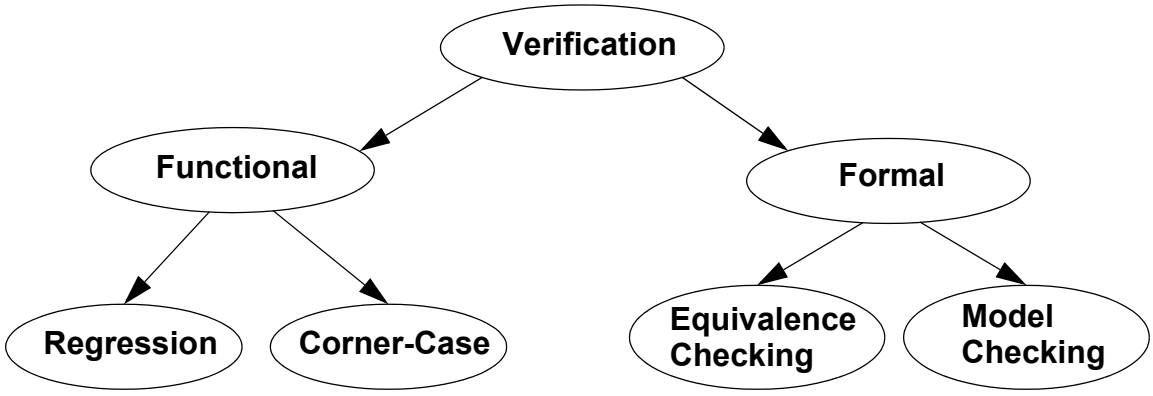


Figure 36: The Verification Realm

Functional verification on the other hand can be splitted in regression tests and corner-case tests.

Functional verification of the ATOLL system as an example

Regarding the ATOLL system development [11], [12] it was an implementation goal to integrate the ATOLL NIC in a commodity compute node using PCI-X. Here, Bus Functional Models (BFM) for PCI-X have been used during verification and Synopsys's Design Ware Implementation IP (DW_PCIX) for implementation.

Referring Figure 37: 'Verification Environment for the ATOLL NIC[4]' the PCI-X BFM consists of a PCI-X master acting as host CPU, a PCI-X slave acting as host main

memory and a PCI-X monitor checking the PCI-X protocol.

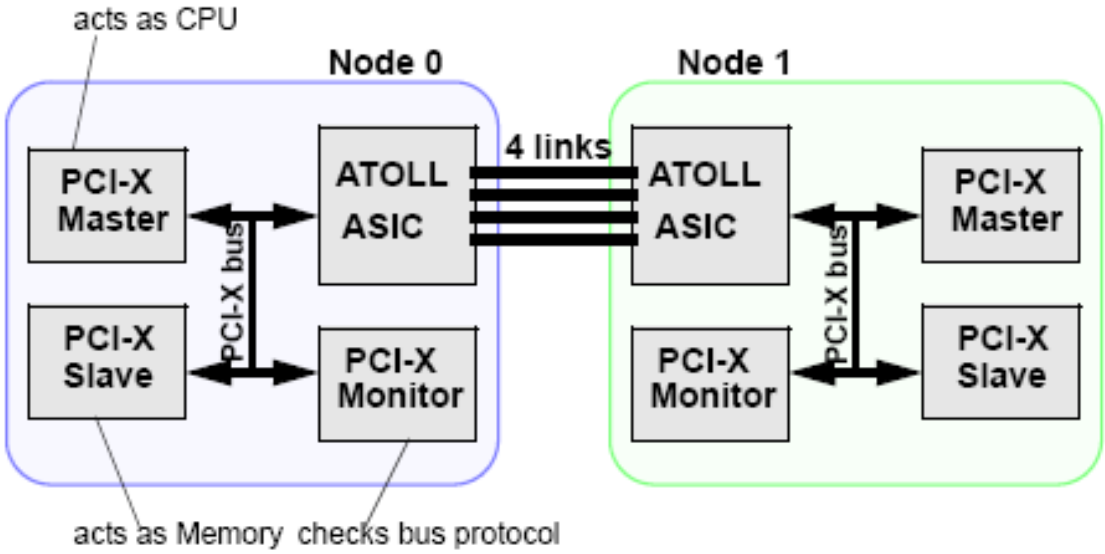


Figure 37: Verification Environment for the ATOLL NIC[4]

The DesignWare implementation IP has been integrated into the ATOLL ASIC.

The so called Command Streams provide the BFM with PCI commands (like pcimaster_write_cycle), which then generates the PCI signalling.

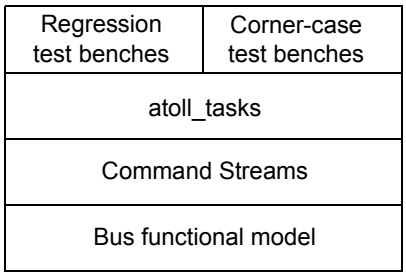


Figure 38: Test bench creation - levels of abstraction

All test benches are build using Verilog tasks (the library has been called atoll_tasks) like DMA_send using these BFM commands.

The atoll_tasks library

The atoll_tasks library (about 1900 lines-of-code) contains a set of Verilog tasks using the PCI-X BFM commands to communicate with the ATOLL NIC. This set ranges from configuring host ports to create and send a complete message. A list of relevant tasks is

listed in Table 7.

Task	Functional description
write_burst (handle, addr, data, length)	Sequence of BFM commands to emulate a burst of length to addr with ongoing word patterns based on data
atoll_conf (handle, base_addr, dma_threshold	Configure the ATOLL device (configure DW_pcix implementation IP: set BAR, command reg, lat timer, etc. and configure ATOLL global state: PLL freq, IRQs, Resets and the host ports) at PCI address base_addr and set the DMA receive threshold to dma_threshold
dma_send (handle, hp, data_len, header_len, route, tag, ...)	Send a message with the length of the header frame header_len, the length of the data frame data_len via route and host port hp using DMA (check for sufficient space in data region and descriptor region, write data and header, write descriptor to main memory, write pointer to the ATOLL device to signal a message ready to send)
dma_receive (handle, hp, tag, ret_val)	Poll on read/write pointers of the descriptor table of hp to recognize message reception. Get descriptor. Read and compare header and data with predefined word patterns. Return message tag and ok/nok signal in ret_val
pio_receive (handle, hp, tag, ret_val)	Poll on FIFO level of hp to recognize message reception. Read and compare header and data with predefined word patterns. Return message tag and ok/nok signal in ret_val
pio_send (handle, hp, data_len, header_len, route, tag	Send a message with the length of the header frame header_len, the length of the data frame data_len via route and host port hp using PIO (check for sufficient space in NIC FIFO, write header and data to NIC, on every last word of the respective frames write to special register to signal end of frame)
irq_handle (handle, irq_val)	Checks the ATOLL NIC's IRQ registers and returns the content in irq_val. Also reports the IRQ reasons in clear text to simulation log file.
disable_dma (handle, hp)	Disable DMA engine of hp
wait_dma_idle (handle, hp)	Polls on read/write pointers of descriptor region of hp and reads idle bit of HW status registers (used to safely switch contexts)

Table 7: Set of relevant Verilog tasks of the atoll_tasks library

The ATOLL test benches

The ATOLL testbenches take advantage of the Verilog tasks defined in the atoll_tasks library in order to simulate certain communication scenarios. There are testbenches focusing on a special communication function (like bench_DMA, testing DMA send and receive) as well as regression testbenches covering a randomly generated communication pattern using PIO and DMA with different routes and length of routes. Table 8 lists the available testbenches of the ATOLL simulation.

Testbench	Type	Lines-of-code
bench_bsd	corner-case; decrease size of descriptor table and data region in order to increase wrap-around frequency, switch contexts, enforce pressure on host ports	767
bench_clk_pci	corner-case; send and receive very big messages in order to generate high bandwidth on the PCI bus	639
bench_cntlregs	corner-case; global control register test, loop-back, general purpose IOs, global counter, hardware semaphore	1400
bench_dl_recover	corner-case; simulate link failure, test XBAR debug port for deadlock recovery	794
bench_dma	corner-case; send/receive DMA message in various configurations (header_length = 0, data_length = 0, massively big messages, send/receive with high pressure, switch contexts and continue)	767
bench_pingpong1	regression; send/receive PIO/DMA messages with random route, send host port, receive host port, header_length and data_length, vary pressure, automated response/verification of message reception	451
bench_pingpong2	regression; same as pingpong1 but with different pingpong cycle length	451

Table 8: ATOLL testbenches

Testbench	Type	Lines-of-code
bench_pio	corner-case; send/receive PIO message in various configurations (header_length = 0, data_length = 0, massively big messages, send/receive with high pressure, enabled/disabled PIO micro-pipelining, check extreme FIFO fill levels), check FIFO behaviour for unused addresses	453
bench_pio_np_fault	corner-case; send specially crafted PIO messages to track down missbehaviour of network port logic	340
bench_gatelevel_scan	corner-case; runs without PCI BFMs to check correct behaviour of internal and boundary scan chains at gatelevel with backannotated SDF, patterns are generated automatically extracting the BSDL file using JTAG compliance. Please refer [15] 'Thomas Schlichter, "Development of a Boundary Scan Pattern Generation Language", Project work, Computer Architecture Group, University of Mannheim, 2003'	31416

Table 8: ATOLL testbenches

It seams to be a reliable verification environment as all possible PCI cycles can be generated and ATOLL responses are monitored/checked. In order to reduce simulation time one has to make assumptions about the behaviour of the main board's host to PCI bridge.

Unfortunately, the behaviour of these PCI bridges was a big secret and far from any reasonable. This leads to the first golden rule in verification: Never make assumptions!

In addition, the verification environment does not reflect reality at 100% because the communication of the host processor to main memory uses the PCI-X bus, which in reality is done via the system bus. Also, performance measurements (forecasts) has been extremely inadequate because response times of communication between ATOLL and main memory has been neglected.

Obviously, simulation has its drawbacks here and they can not be removed due to missing models of the entire system and of course simulation complexity.

Further difficulties of this simulation environment:

- Verification of implementation IP
  Does it make sense to check the implementation IP (from Sysnopsys) with the verification IP (from Synopsys)?

- Performance forecast
  The latency of main memory responses influences architecture decisions and can not reliably derived from simulation.

However, it is a big advantage beeing able to simulate the entire NIC hardware in their cycle accurate PCI-X bus environment. During the time, where this verification happened, no main board supporting the PCI-X bus existed. The correctness of both the Verification IP and the Implementation IP are mandatory for a successful first implementation.

All verification techniques of ATOLL are described thoroughly in [4].

4.2.1 Concurrent development of hardware and related low level software

By parallelization of the hardware and software development phases time-to-market can be significantly decreased. By [45] it has been proved that even driver development can be started on a register accurate SystemC (hardware-)model. This work can be regarded as system level evaluation of this concept. A register accurate and fully functional model of the ATOLL [11] [12] ASIC using SystemC has been developed with the "The Concept of Virtual Hardware Prototyping" on page 63 in mind. This has been accomplished in all software levels beyond the Linux OS driver. Only the driver has to be changed in a few lines of code due to direct hardware system calls like ’pci_find_device’. A simple ’IFDEF’ is now included in the regular ATOLL driver to enable the same driver to be used with the SystemC simulation model. All other software from API (called PALMS) including the ATOLL MPI (MPICH based) implementation and all applications run without any changes. So software development can be started as soon as the SystemC hardware model is accurate with respect to the interface seen by the low level software.

4.2.2 Seamless hardware/software interfacing

During the development of hardware devices the question arises how to interface with the regarding software/OS levels. There might be situations, where software is already completely developed or is in a fixed state (OS, MPICH’s MPI implementation). In this case it is meaningful to test the interfacing by veridical simulation to ensure an elegant (in this meaning fast) implementation. The work of [49] ’Sven Stork, "Anpassungen für Virtual Hardware Prototyping des LINUX-Kerneltreibers des EXTOLL NICs", Project

work, Computer Architecture Group, University of Mannheim, 2003' describes this approach for the EXTOLL API intended to interface with the MPI2.0 specification.

4.2.3 Integrity and completeness of software required functionality

Hardware systems designed for complex software environments see the complexity of required functionality. It is a verification mechanism to co-simulate the software environment with the intended hardware ensuring completeness of the designed hardware functions. As an example, consider Ethernet as communication device. Ethernet is feasible for UDP/TCP like packet transmission, but regarding the MPI2.0 specification, as a message passing library, there are lots of requirements ethernet can not provide efficiently. An efficient hardware implementation must cover these requirements to avoid costly software detours. It has been found impossible to overview such complex software layers without a detailed implementation and coding.

4.3 FPGA based ASIC prototyping

The term 'FPGA based ASIC prototyping' refers to the use of reprogrammable hardware (i.e. FPGA) to emulate the function and behaviour of the intended hardware. This concept is already well known in industry for simulation acceleration [10].

ASIC-Prototyping Station using a 64Bit/66MHz PCI interfaced FPGA

In a diploma thesis [13] we developed the complete prototyping environment including LINUX kernel driver and FPGA design flow using a PCI interfaced FPGA board. Please find Figure 39: 'AVnet Virtex Development Board [13]' for a photo of the prototyping

board.

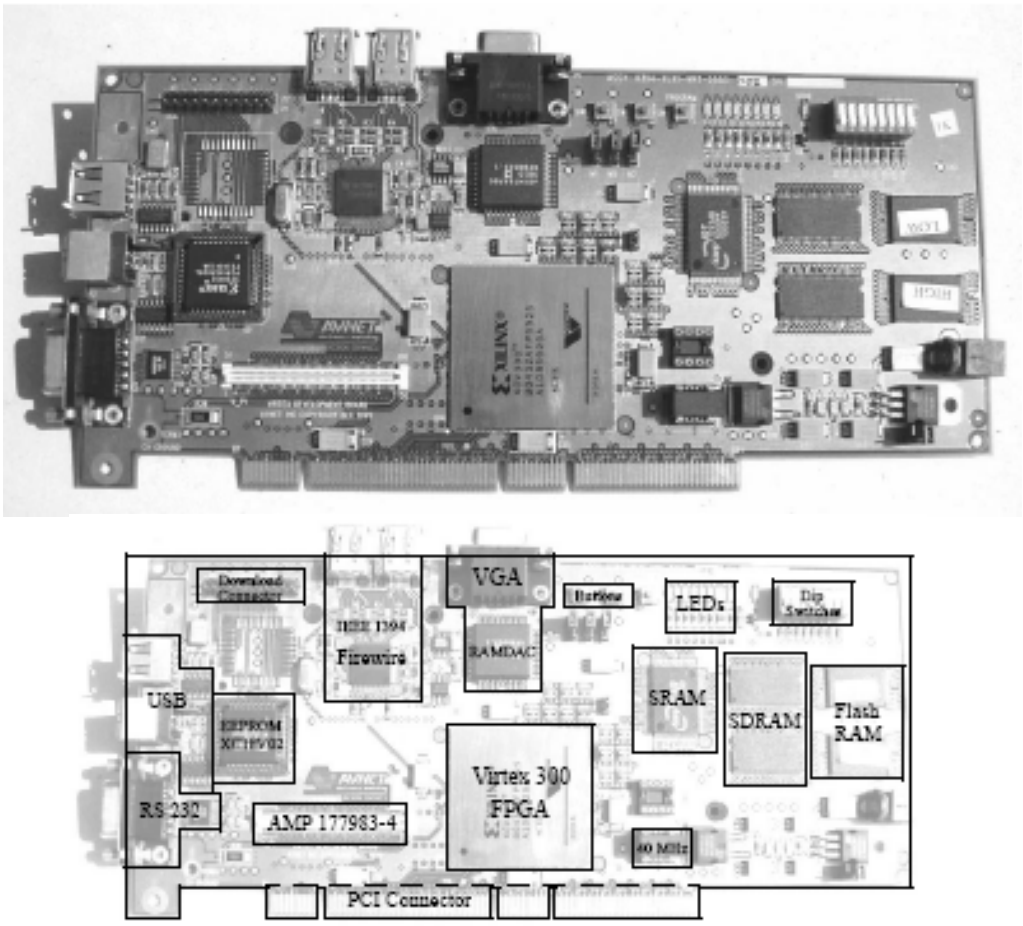


Figure 39: AVnet Virtex Development Board [13]

This thesis focuses on the definition and implementation of a hardware abstraction layer (HAL) which on the one hand provides ASIC engineers with easy-to-use interfaces to connect their designs to and on the other hand grants access to a 64 Bit / 66 MHz PCI bus to guarantee high bandwidth control and test possibilities. [13]

The work focuses on a generic Hardware Abstraction Layer (HAL) in order to get easy

access to all components on the prototyping board.

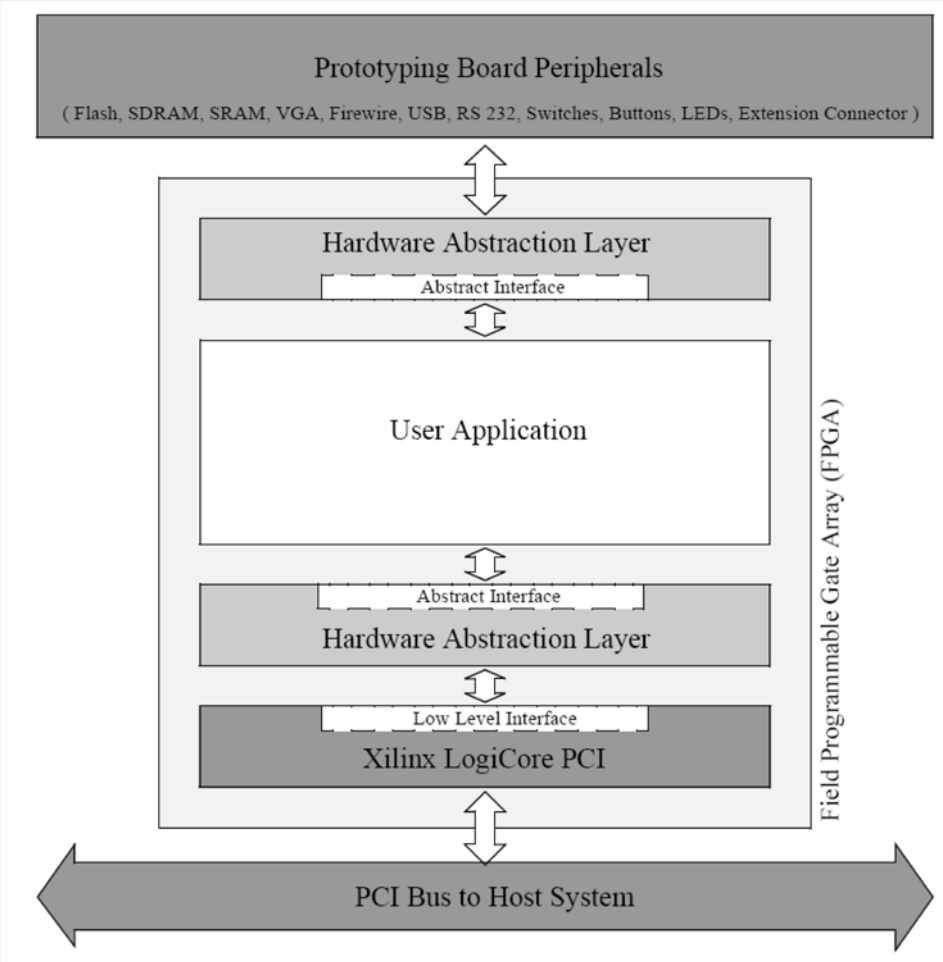


Figure 40: User application encapsulated by the HAL [13]

Using this prototyping station any interfacing from hardware components located on the PCI bus to the host system can be tested and benchmarked. We tested several systems on their main memory DMA bandwidth and latency as well as interrupt reaction latencies and used these information during design space exploration for the EXTOLL system.

Regarding EXTOLL several OS specific mechanisms like TLB (Transaction Lookaside Buffer) coherence (investigated in [14]) with external devices are perfect candidates for this prototyping station as simulation is not feasible.

4.4 Physical design impact of UDSM designs

> *Efficiency, methodology and complexity (integration) are heavily related and interdependent. The higher the degree of integration (the more transistors per area), the better the design methodology must be to result in a true increase in efficiency.[16]*

Motivation of a backend flow inhouse

Since the introduction of the top-down design methodology system architects need to estimate technical and physical limiting conditions. The design exploration phase regularly driven by the specification might also be driven by engineering feasibility. Considering the EXTOLL design exploration phase the number of dimensions for the future crossbar was evaluated by a technical exploration of the possibilities of IBM's 0.13um process node [17], while some other decisions like Virtual Channel capability was evaluated by high-level SystemC simulations in [18].

Additionally, there are some physical design constraints besides the regular floorplan hard to communicate to the back-end team. During the ATOLL design phase the back-end task was outsourced to IMEC's IC Design Service, Leuven, Belgium. Regarding the ATOLL chip layout there has been several special requirements:

- Placement of input and output sample registers
  Using parallel clocked transmission it is crucial to avoid skew between the parallel bits and the clock.
- Placement and routing of the Delay Locked Loop (DLL)
  The design of the DLL requires well defined signal delay elements. The routing impact must not be neglected.

4.4.1 Creating a leading-edge design flow

Starting with a diploma thesis [16], we got a thourough understanding of design methodology and tooling, as well as file interface formats and industry standards. The abstract of the work follows:

> *The present work deals with the methodology of today's Application-Specific Integrated Circuit (ASIC) construction process, their design, practical implementation and theoretical cornerstones. It analyzes the crucial points and*

challenges in the design of cell-based ASICs, identifies the key elements and procedures and differentiates between important principles and minor matters. It prepares the physical implementation design flow, elaborates solutions to practical obstacles and shortcomings and provides the knowledge and experience an engineer must have to bring an up-to-date high-performance, multi-million gate System on a Chip (SoC) from RTL to silicon. [15]

The work took as a basis, we build up a complete leading-edge nanometer aware design flow in order to get reliable physical implementation forecasts of the desired circuits. In chapter 4.4.2: 'UDSM characteristics' is explained why logic synthesis does not longer deliver reliable and accurate physical implementation forecasts.

4.4.2 UDSM characteristics

With nanometer process technologies of 150nm and below, the combination of fine-line geometries, high clock rates and lower supply voltages used in leading-edge designs gives rise to electrical and physical effects that can dramatically degrade design function and performance. [19]

Shrinking design geometries expose physical effects not supported nor handled by the traditional design flow. Second order effects not modeled by these tools are the driving factor behind the decreasing accuracy of the used models.

With shrinking geometries the signal delay related impact of the nets rises in comparison to the gate delay. Depending on the circuit and the technology the net delay may rise to 80% of the total delay (net delay and cell delay) [24].

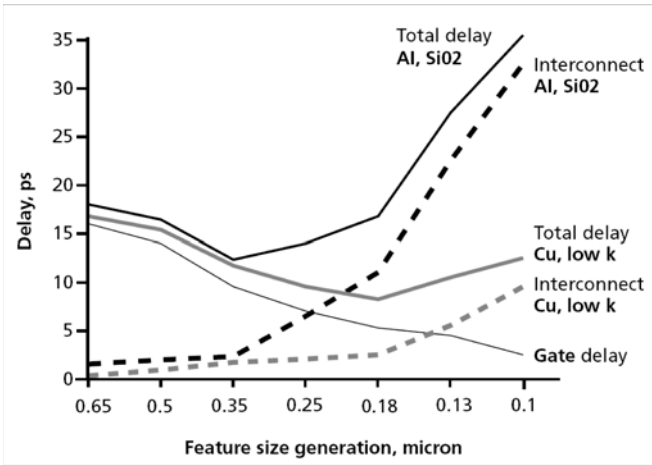


Figure 41: Wire and gate delay in Al and Cu [25]

In the traditional design flow, there is a symbolic wall between the frontend (logic) and the backend (physical) design. Frontend designers usually don't care about the backend design process. Just handing over a netlist/constraint set (so called ASIC sign off) often prevents timing closure. The only physical impact in logic synthesis was introduced by the wire load model (WLM) modelling net delays and net load depending on the net's fanout and the module size. WLMs are statistically generated and delivered together with the .lib technology file [16]. Alternatively, WLMs can be generated on the design itself (custom WLM) beeing some more accurate.

Taking into account that the net delay rises against the cell delay, the possible variation of the accuracy changes, also leading to a miss-prediction of the synthesis tools. The synthesis tool reports inaccurate timing and also treats capacitive related transformations like buffer insertion/deletion and cell up/down sizing in a wrong way.

The netlist now contains buffers or other driving cells that are inappropriate leading to a unpropitious starting point for the heuristic placement algorithms.

There are now two general possibilities how to address this problem:

- Physical Synthesis
- Physical Implementation

Both try to overcome the traditional burden of the separation of physical design from logic design and either bring physical information to synthesis or synthesis abilities to the placer.

Physical Synthesis

This way pushes physical information to the synthesis tool. Physical synthesis is geometry aware and also reads .lef files just like the placer[16]. Together with the technology file and the cell libraries some floorplan information is also needed to do a quick placement and a trial route. The information now available for timing analysis is derived from the initial placement and the trial route. The so called placed gate netlist or placed netlist feeds in to the placer back to the regular design flow.

As the synthesis tools is aware of the placement/routing information the generated netlist is much closer to the final netlist. WLMs are no longer needed.

Physical Implementation

This way integrates some synthesis features into the placer. The placer now knows how

to correct wrong driving strengths and is able to insert/delete buffers. The "footprint" information classifies the standard cells in categories with the same boolean function and so enables the placer to switch cells with different driving strengths. Some placers even know how to split complex gates in order to buffer up long paths.

Signal Integrity issues

In addition to the increased wire delay, other timing related effects like IR drop and crosstalk induced delay need to be considered. The EDA vendors react with appropriate analysis tools integrated in the regular design flow.

With all these effects the implementation complexity rises and turn around time prolongs to an inacceptable level. During design space exploration some corners require a technology focused feasibility study and also during RTL design timing closure is the key issue. For these requirements a prototyping way is provided to deliver a quick forecast. Of course this way is not as reliable as the sign-off flow. Cadence calls their prototyping flow the Silicon Virtual Prototyping (SVP) flow [25].

IBM proposes another interesting way to qualify designs for the backend process. So called Zero-Wire-Load-Models (ZWLM) assume "perfect" wires with zero resistance and zero capacitance [26]. There are several effects described below:

- The timing calculated by the synthesis tool is a hard bound. It can not be faster after P&R as wires only add delay and no net induced delay is assumed during synthesis. Technology specialists can set rules of thumb to define a certain slack for a specific design flow step in order to do risk assessment for the later steps. I.e. "To enter the floorplanning phase, the incoming netlist must meet the -30% of a cycle ZWL slack target."[26]
- The netlist is free of WLM introduced buffers. This is supposed to be a better starting point for placement tools capable of above mentioned (chapter 4.4.2 "UDSM characteristics" on page 75) physical implementation transformations. As WLMs have been introduced in times where the placer doesn’t have any netlist transformation capabilities it makes no sense to start the placement step with an "over-buffered" netlist.

5 Architecture and Function Scope of Building Blocks

In this section the building blocks of the EXTOLL network are described, namely:

1. Host Interface (and EXTOLL block level interfacing) in section 5.4 on page 114
2. Network interface device (for EXTOLL they are virtualized and called Virtual Devices)[42]
3. Network switch in section 5.3 on page 89
4. Network Processing Unit (NPU) in section 5.2 on page 80

Please refer Figure 42: ’EXTOLL Top-level schematic’ for an overview.

In this thesis the Host Interface, the Network Switch as well as the NPU are covered. For details on the Virtual Devices please find [42].

5.1 Top-Level Architecture Decisions

As in the ATOLL network a single chip solution for lower cost and better scalability is aimed. The routing resource therefore will be located together with the network interface device on a single die. Also the PHY’s for the physical transmission are integrated.

The superseeding idea of EXTOLL might be summarized as follows:

"Building Massively Scalable Building Block For A Monolithic Supercomputer Out Of Commodity Hardware"

The commodity idea is based on the cost advantages of mass production, which is the underlying principle of commodity cluster systems and system area networks

in general.

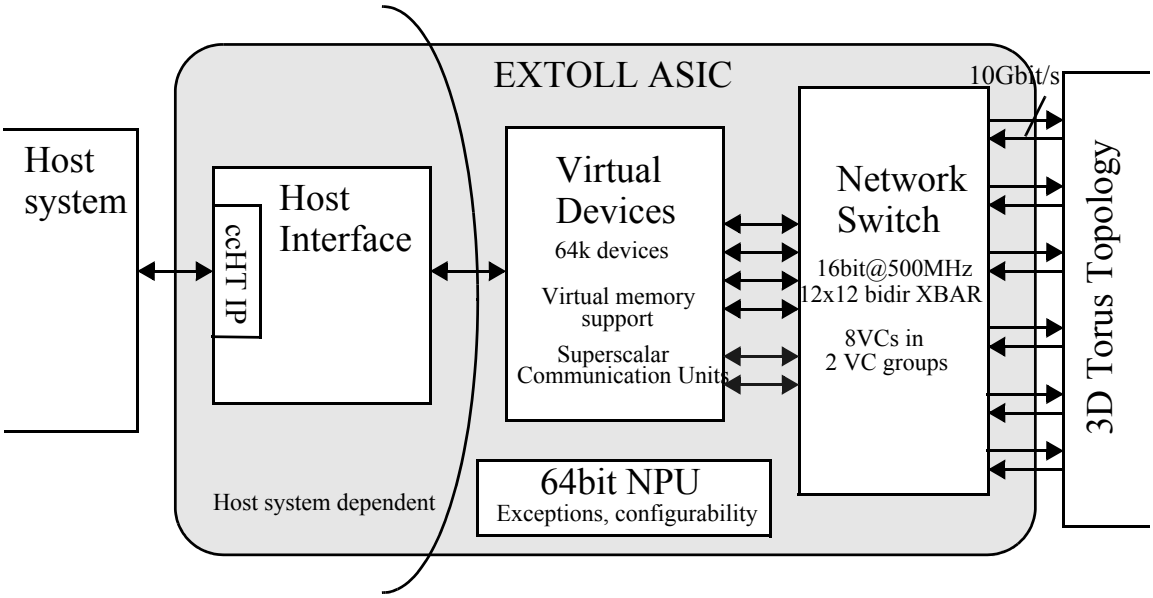


Figure 42: EXTOLL Top-level schematic

5.2 NPU (Network Processing Unit)

This section covers all considerations as well as implementation details of a dedicated processing resource on the network interface working concurrently to the node's main CPU.

There have been three diploma thesises influencing this work:

1. Holger Bellm, "Architectural Design and Prototype Implementation of an Embedded Network Processor Core using Language for Instruction Set Architectures (LISA)", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2003
   This work covered the complete instruction set elaboration and processor design space exploration. Also, the implementation using CoWare's Language for Instruction Set Architectures (LISA) and the technology synthesis step have been performed.
2. Ingo Feldner, "High Level Executable Specification Development of a high performance SAN chip", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005
   This work covered the EXTOLL integration issues like program loading, execution, interrupt servicing and exploration of the access architecture to the functional units as well as the cache design space exploration. Furthermore, a system level simulation environment has been established using the HW/SW Codesign methodology reported in section 4.2 on page 63.
3. Timo Sponer, "Development, Verification and Integration of a Processing Unit in the Communication Function of a SAN Device in SystemC", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005
   This work finalized the EXTOLL integration and system simulation and elaborated the MPI window locking functionality as an example to be offloaded to a concurrent NPU.

During the course of this work a complete pipelined 64-bit RISC processor has been developed. The discussion about the following design consideration can be found in [47]:

- RISC vs CISC Processors
- Superscalar vs VLIW Architecture

- Pipeline Architecture
- Dependencies Between Instructions and NPU Instruction Dependency Detection
- NPU Integer Arithmetic Implementation
- Control Flow Architecture
- Result State Concept using Condition Codes (Two-Instruction Implementation with Condition Registers versus Single-Instruction Implementation with Compare and Branch)
- NPU Control Flow Architecture
- Reducing Branch Penalty by Branch Prediction
- Addressing Modes
- External Events to the NPU Core (Signals and Interrupts)

This section is structured in four parts beginning with the ’NPU Features and Overview’ followed by the ’Reasons for a dedicated compute resource in a SAN’ and the NPU special ’Instruction Set enhancements’ and concluding with the ’Implementation details’.

5.2.1 NPU Features and Overview

The following list comprises the NPU features [47]:

- RISC microprocessor core
- 64-bit architecture
- five-stage load use interlocked pipeline
- one instruction issue per clock cycle
- single clock cycle execution for all except memory load instructions
- 32 x 64-bit general purpose registers
- two read, two write port register file
- high instruction throughput
- 64-bit integer arithmetic logical unit

- completely decoupled compare and branch
- non-interlocked pipeline with forwarding paths to eliminate data dependencies
- one delay slot for control flow instructions
- outstanding loads supported
- out of order load completion supported
- support for touch-loads
- hardware interrupt and software signal servicing

Figure 43 provides a block diagram of the APU processor core pipeline structure. It shows the fivestage RISC pipeline layout with the most important data and control paths. All pipeline stages are separated by pipeline registers. The Instruction Fetch (IFE) unit supplies the pipeline with operations from the instruction memory, which is accessed via a separate bus (Harvard Architecture). The Instruction Decode (ID) unit generates the control and activation signals depending on the processed instruction for the following pipeline stages. It also generates control signals used by the pipeline controller to flush or stall pipeline registers. The Register Read (RR) unit fetches the values of the register file needed for the calculations performed in the Execution Unit. The Execution (EX) unit selects which values to use for calculation, the ones provided by the RR Unit, or the values provided by the forwarding paths. Data memory load and store operations are also initiated in this unit.

The Register Write (RW) unit stores the calculated results to the register file.[47]

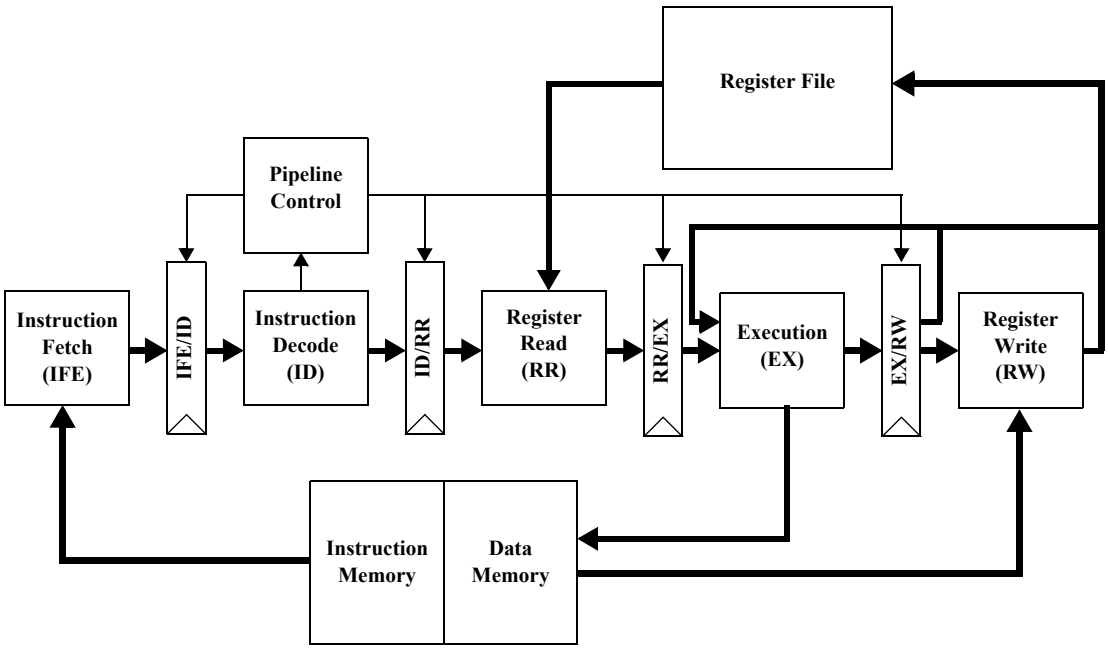


Figure 43: The NPU block diagram [47]

5.2.2 Reasons for a dedicated compute resource in a SAN

Embedded processing units in network interfaces have already been used in several SANs like 'Myrinet by Myricom' (please refer section 2.2.2 on page 17) or 'QsNetII by Quadrics' (please refer section 2.2.1 on page 8). However, they usually worked on regular message transfer tasks. That is not the case in the EXTOLL SAN, where dedicated hardware in the form of functional units work on all the regular duties of the SAN. In the EXTOLL SAN the NPU is dedicated for special tasks like:

- Protocol offloading
  The embedded network processor, which is user programmable, can be used to offload asynchronous protocol handling tasks. Current MPI implementations use threads to avoid the need to wait for an MPI call in order to do message reception work, if at all. Having a dedicated compute resource on the network interface doing MPI message reception tasks will further reduce MPI software latency. Protocol offloading is already standard for LAN implementations like '10GigEthernet by Chelsio' (please refer section 2.2.8 on page 28) but here protocol offloading is aimed to reduce the node's CPU load. For networks dedicated to TCP like data transport this makes sense

as the TCP stack is quite compute intensive and high data rates then result in a non-negligible CPU load. Therefore, protocol offloading in LANs increases the node's compute power. In the area of SANs, the goal is to increase the communication power as in parallel computation usually the communication power lags behind the computation power, which lowers the maximum degree of parallelization (please refer chapter 2.1 "Communication demands of distributed and parallel computing" on page 5). So, protocol offloading for SANs differs from LANs in the type of tasks being offloaded.

- Support for parallel computation global collective operations
  The embedded network processor can support communication library specific or user defined collective communication primitives like broadcasts, scatter, gather or even reduce operations. MPI supports the following arithmetic/logic functions for the reduce operation:
  - MPI_MAX maximum
  - MPI_MIN minimum
  - MPI_SUM sum
  - MPI_PROD product
  - MPI_LAND logical and
  - MPI_BAND bitwise and
  - MPI_LOR logical or
  - MPI_BOR bitwise or
  - MPI_LXOR logical exclusive or
  - MPI_BXOR bitwise exclusive or
  - MPI_MAXLOC max value and location
  - MPI_MINLOC min value and location
- Support for intelligent communication protocols
  The embedded network processor is a programmable processor in the network interface that allows the implementation of intelligent communication protocols. As in [45] specified, the NPU is able to modify messages on-the-fly. This makes user defined flexible protocols possible.

All these features add flexibility in the form of programmability and concurrency to the node's main CPU, which is especially interesting for the research purpose of the EX-

TOLL SAN.

5.2.3 Instruction Set enhancements

The NPU's instruction set was augmented with extra instructions to construct network packets, manipulate events, effectively schedule threads and block, save and restore a thread's state when scheduled.

The flexibility of adding or modifying instructions was supported through a dedicated language called LISA (property of LisaTek acquired by CoWare in 2003):

> *A unified language, LISA 2.0 (Language for Instruction Set Architectures), is used to model embedded processors. All required software tools, fast simulation models, as well as a synthesizable HDL model of the architecture can be automatically derived from this model. This allows the concurrent development of hardware and software. LISA can be used to describe the behavior, the instruction set coding and the syntax of processor architectures in a syntax similar to the C programming language. Instruction set architecture accurate and cycle accurate models of processor architectures including pipeline behavior and memory hierarchies can be modelled using LISA.[47]*

The LisaTek environment further generates necessary implementation software like a debugger and even a C compiler able to compile ANSI C to the defined instruction set.

Special Instructions

These instructions are introduced to perform byte mask and merge operations. Every instruction in this group provides one destination register index, one source register index and two 6 bit values to specify bit ranges.

This group contains the following instructions:

- invert selected bitrange (BITINV R[dst], R[src], #i, #j)
  Invert selected bit range copies the value of the source register to the destination register. Each bit in the range specified by i and j is inverted.
- clear selected bitrange (BITCLR R[dst], R[src], #i, #j)
  Clear selected bit range copies the value of the source register to the destination register with each bit in the range specified by i and j set to 0.
- set selected bitrange (BITSET R[dst], R[src], #i, #j)

Set selected bit range copies the value of the source register to the destination register with each bit in the range specified by i and j set to 1.

- extract selected bitrange (BITEXT R[dst], R[src], #i, #j)
  Extract selected bit range only copies the bits in the range specified by i and j to the lowest part of the destination register. The upper part of the destination register is set to 0.
- replace selected bitrange (BITINS R[dst], R[src], #i, #j)
  Overwrite selected bit range copies the correct amount of bits out of the lower part of the source register to the destination register. The position in the destination register is specified by i and j. All other bits of the destination register remain unchanged.

In Figure 44 some examples of the instruction format have been given. For a complete instruction set format description please refer [47].

SWAP instructions

The SWAP instructions swap the lowest two bytes (SWAP.B), words (SWAP.W) or double words (SWAP.D) of a source register into a destination register. The remaining bits are untouched. There instructions are especially interesting for routing manipulation.

Logic instructions

In the group of logic instruction in addition to the regular arithmetic and logic shift operation a special rotate instruction (ROTL for rotate left, ROTR for rotate right) has been introduced. The rotate operations are useful for field manipulations because no "content" in the sense of bits is lost during the operation in contrast to the shift operation.

These extra instructions ease the register manipulation in a very efficient way by reducing the amount of instructions needed dramatically.

5.2.4 Implementation details

First synthesis results from [47] show impressive results in area and timing for a 0,18um standard cell library. The pre-layout area is below one square millimeter, where approximately 55% of the area is consumed by the register file. Here, VLSI optimization might reduce the area effort even further. The pre-layout maximum frequency is slightly above

BITINV (Invert Selected Bitrange)

Operation:

R[dst] <= R[src1] with R[src1][i:j] inverted

Assembler Syntax:

BITINV R[dst], R[src1], #i, #j

Description:

In the region specified by the immediate values i and j all bits of the source register are inverted, the result is stored in R[dst]. All other bits of the source register are not affected. If i < j nothing is done. Both immediates i and j are 6 bit unsigned values, thus have to be specified between 0_{10} an 63_{10}.

Instruction Format:

31 29	28 26	25 22	21 16	15 10	9 5	4 0
mopc	*fid*	*reserved*	*immediate*	*immediate*	*register index*	*register index*
1 0 1	0 0 0	x x x x	i	j	src1	dst

BITCLR (Clear Selected Bitrange)

Operation:

R[dst] <= R[src1] with R[src1][i:j] set to 0

Assembler Syntax:

BITCLR R[dst], R[src1], #i, #j

Description:

In the region specified by the immediate values i and j all bits of the source register are set to 0, the result is stored in R[dst]. All other bits of the source register are not affected. If i < j nothing is done. Both immediates i and j are 6 bit unsigned values, thus have to be specified between 0_{10} an 63_{10}.

Instruction Format:

31 29	28 26	25 22	21 16	15 10	9 5	4 0
mopc	*fid*	*reserved*	*immediate*	*immediate*	*register index*	*register index*
1 0 1	0 0 1	x x x x	i	j	src1	dst

Figure 44: Instruction Set Format of BITINV and BITCLR [47]

300MHz and can be further increased by the targeted 0,13um library and a better arithmetic architecture selection, as the critical path goes through a 64-bit adder (Figure

45).A clean design style would require a register at the outputs of a pipeline stage, which

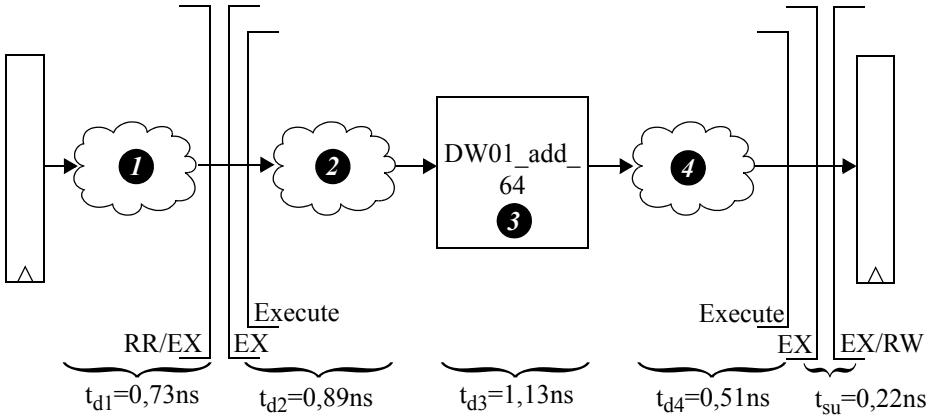


Figure 45: Timing Critical Path of the NPU [47]

is here not the case (see cloud 1). This is one of the drawbacks of the limited design freedom using automated generators like the LisaTek tools. However, newest synthesis technology is able to "retime" critical paths by moving registers through combinational logic relaxing critical paths.

EXTOLL integration

> *The design of an on-chip communication structure which connects the functional units with the NPU is a basic requirement. It has to provide access to all functional units that has been designed so far and it has to be extendable to allow an attachment of future modules. The communication structure allows the NPU to move data from and to the functional units. These data movements are initiated and controlled by the NPU whereas the functional units are passive and only respond to actions initiated by the NPU. The NPU can be considered as a master and the functional units as slaves. [32]*

With the NPU being a second master concurrently to the node's CPU the NPU is able to perform the same tasks as the main CPU. This brings in full flexibility for the range of possible tasks performed by the NPU. In addition the NPU's memory can be considered a slave module, which enables the main CPU to access the NPU's memory structure. For details of the on-chip communication structure please refer chapter 5.4: 'Hostinterface and EXTOLL block level communication' on page 114.

5.3 Network Switch

The Network Switch is the routing resource of the EXTOLL network and is implemented in a XBAR architecture like the ATOLL network. There has been several project works and diploma thesises to approach this complex building block on one hand side finding the physical constraints derived from semiconductor technology on the other hand exploring the architectural possibilities of multiple transmission levels and virtual channels avoiding head-of-queue blocking and enabling fast hardware supported parallel computation functions like barriers. The final implementation is a 12x12 bidirectional

XBAR. Please find Figure 46 and Figure 47 for top level diagrams.

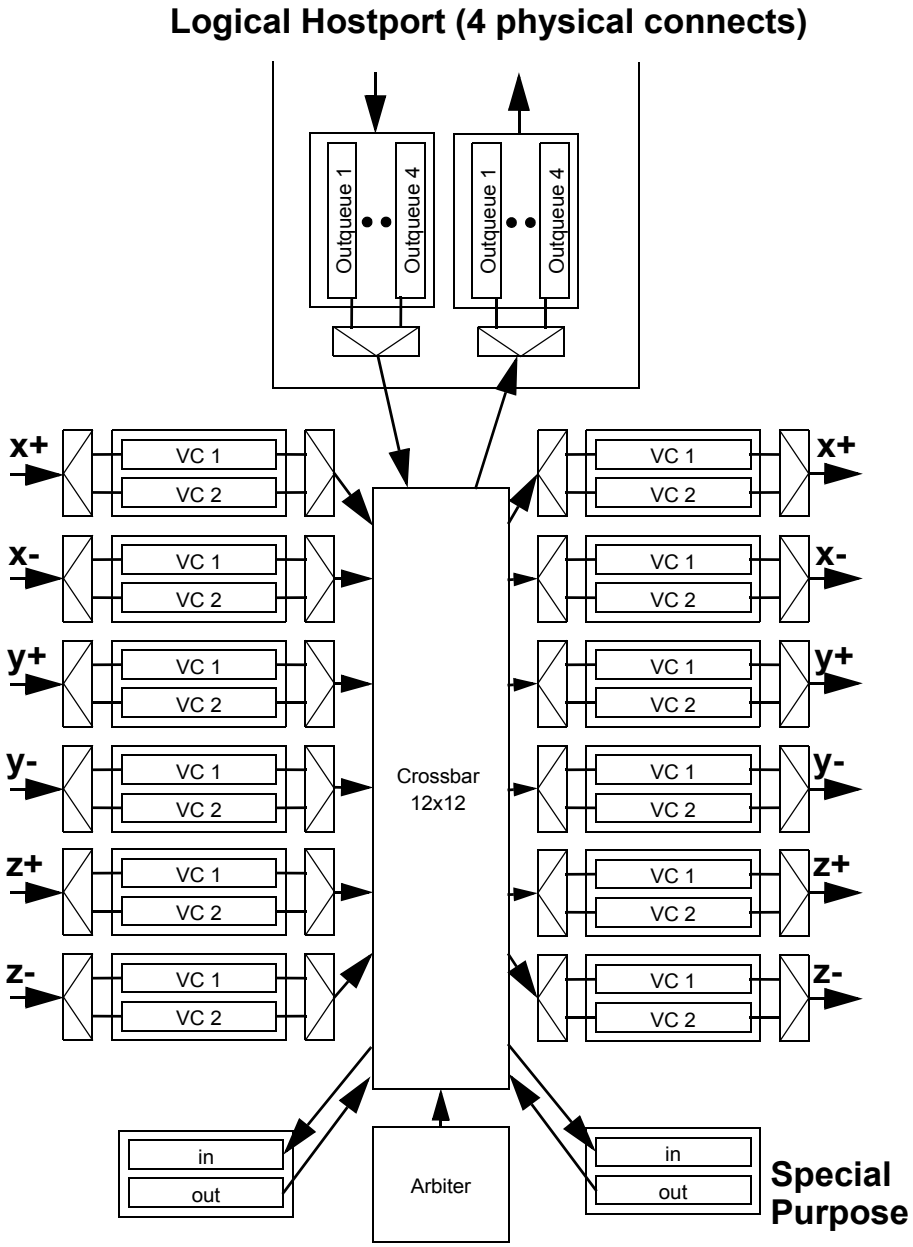


Figure 46: Top level diagramm (unidirectional view)

This subchapter is organized as follows; section 5.3.1 will describe the approach and

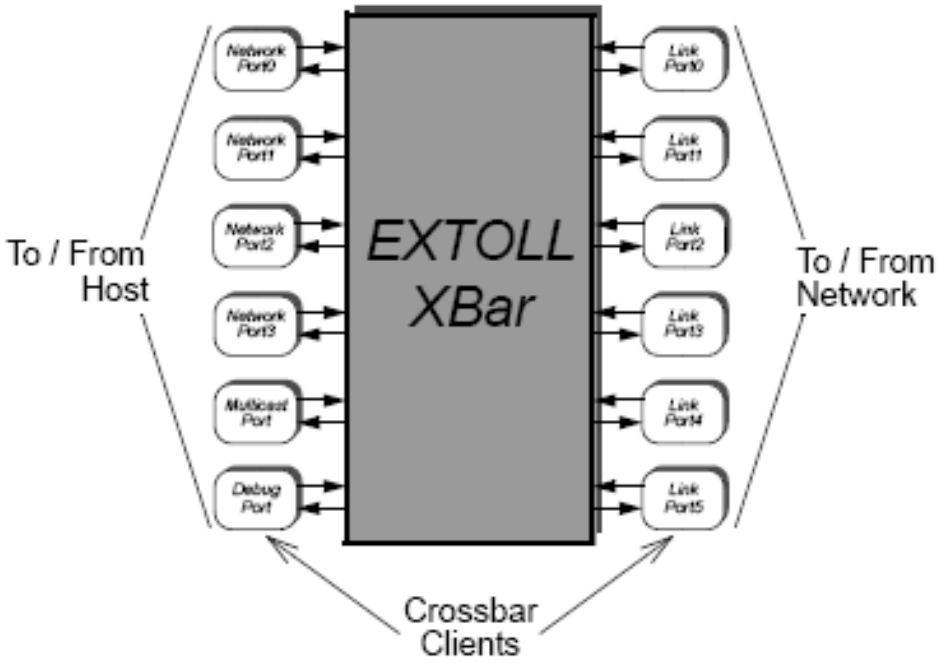


Figure 47: Top level diagramm (bidirectional view)

methods used to explore the complex design space, section 5.3.2 digs into details of the functional enhancements of the XBAR architecture.

5.3.1 Design Space Exploration - Approach and Methods

The work performed on this building block has been divided in 4 steps:

1. Analysis and generator creation for the ATOLL XBAR arbiter [40]
2. Complete parametrizable ATOLL XBAR IP generator including synthesis scripts [41]
3. Adding next generation features in an executable specifiation [18]
4. Hardware implementation of the EXTOLL XBAR [17]

The work on this building block has started with the analysis of the currently in the ATOLL network implemented arbitration functions. With the deep understanding of the boolean equations and state machines responsible for the XBAR arbitration it was possible to build a parametrized generator [40] for Verilog RTL. With this results we did first measurements on scalability with respect to the number of ports being arbitrated.

Furthermore, the results built the basis of the next student work targeting the entire XBAR implementation to be parametrizable and also beeing able to generate synthesis scripts for a first shot on timing [41]. These two student works has been mainly done to estimate the effort in introducing automation in the design space exploration step as silicon was ready as these works has been performed. We started with the current implementation of the ATOLL XBAR and ended up with a fully parametrizable and synthesizable soft IP of the XBAR building block, including synthesis scripts. Again, we also used this work to get first estimates on how potent the next generation silicon tech-

nology is. First pre-layout results can be found in Figure 48.Regarding methodology,

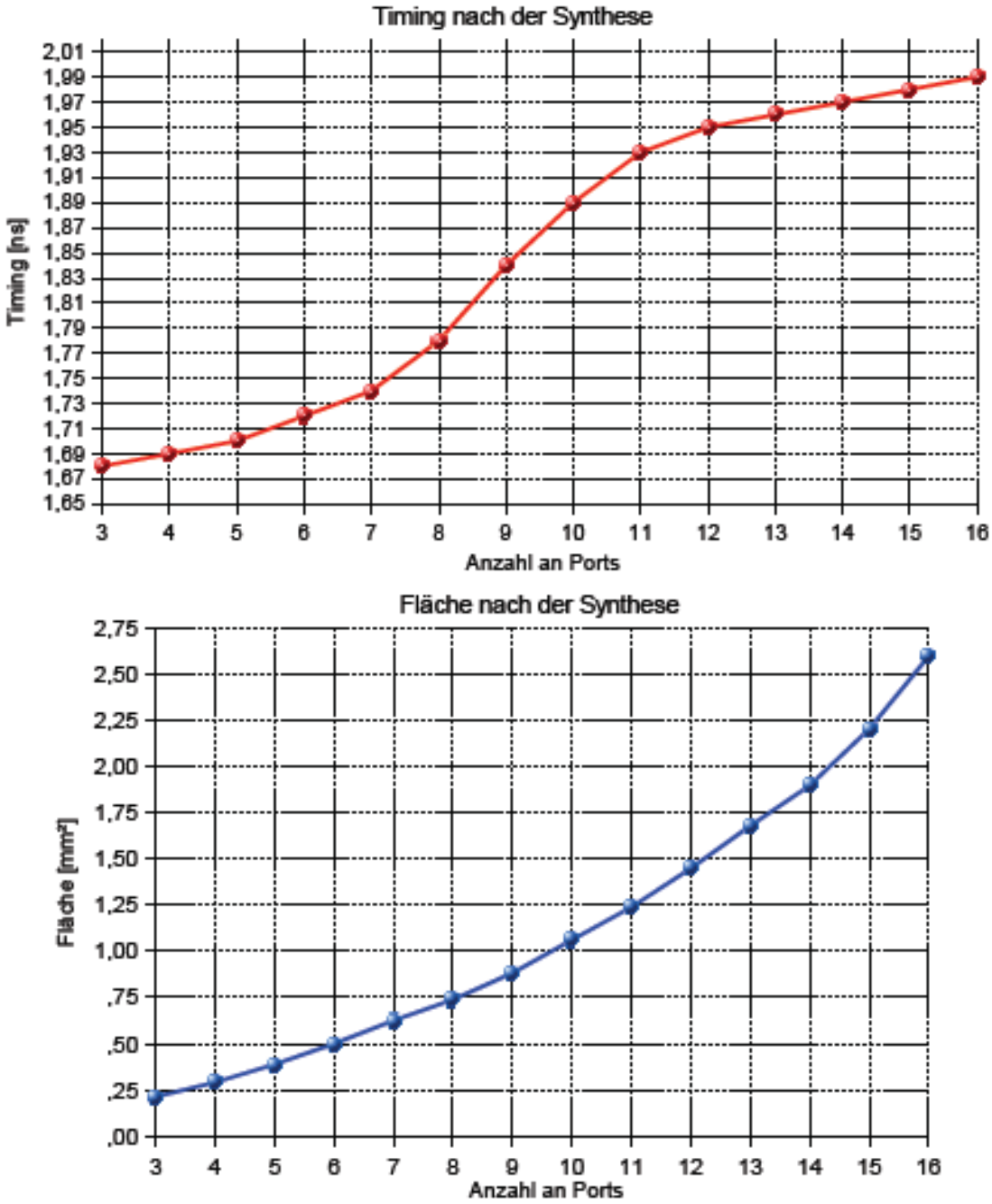


Figure 48: Pre-layout timing and area results from [41]

this work for the first time used and defined a group wide (Computer Architecture Group) Verilog RTL coding style automatically checked by a linter software. As well for the first time incorporated signal integrity EDA tooling using CeltIC and the SI-

aware features of the router NanoRoute. Both tools are Cadence tools available through the SEED project (chapter 4.1.3 "Realization - SEED Project" on page 58).

Regarding parametrization it was a challenge to extract "repeating" terms in the boolean logic of XBAR arbiter. This was solved by the use of a sophisticated visualization method.

Please find below (Figure 49) an example of how we visualized the boolean functions

Figure 49: Visualisation of the arbiter boolean functions[40]

of the arbiter in order to analyze the repeating structures for parametrization. Each row represents an "and" term and each column an input variable, where a green marked cell stands for the positive appearance of an input variable in the term and red for the negative respectively. A white cell means that the input variable does not appear at all.

The original Verilog RTL code below (Figure 50) exemplifies how hard to understand

```
A1_o0 <= #(Tco) (
(~A1_m1 & ~A1_m2 & ~A1_m3 & ~A1_m4 & ~A1_m5 & ~A1_m6 & ~A1_m7 & req0 & ~stop
& ~h_full) |
(A1_m0 & req0 & ~stop) |
(req0 & ~req1 & ~req2 & ~req3 & ~req4 & ~req5 & ~req6 & ~req7 & ~stop &
~h_full) |
(~A1_m1 & req0 & ~req2 & ~req3 & ~req4 & ~req5 & ~req6 & ~req7 & ~stop &
~h_full) |
(A1_m3 & req0 & ~req3 & ~req4 & ~req5 & ~req6 & ~req7 & ~stop & ~h_full) |
(A1_m4 & req0 & ~req4 & ~req5 & ~req6 & ~req7 & ~stop & ~h_full) |
(A1_m5 & req0 & ~req5 & ~req6 & ~req7 & ~stop & ~h_full) |
(A1_m6 & req0 & ~req6 & ~req7 & ~stop & ~h_full) |
(A1_m7 & req0 & ~req7 & ~stop & ~h_full) |
(A1_o0 & req1 & ~stop & h_full) |
(A1_o0 & req2 & ~stop & h_full) |
(A1_o0 & req3 & ~stop & h_full) |
(A1_o0 & req4 & ~stop & h_full) |
(A1_o0 & req5 & ~stop & h_full) |
(A1_o0 & req6 & ~stop & h_full) |
(A1_o0 & req7 & ~stop & h_full) );
```

Figure 50: Verilog RTL code of the example function of Figure 49.

the arbiter functions without visualisation are. At the end a perl script was able to generate the Verilog RTL arbiter functions with the different parameters:

- number of requests/grants
- a selector to choose if the disjunctive normal form (DNF) consists of 0-terms or 1-terms
- several switches to enable testbench generation, debug features and delay for Verilog assignments

The next two diploma thesises worked on the functional design space exploration of new features like hardware barriers and virtual channels resulting in an executable specification in SystemC [18] and their hardware implementation in synthesizable Verilog RTL [17]. Please refer to chapter 4.2: ’Hardware/Software Codesign and Cosimulation’ on page 63 for the value added by executable specifications to the entire system development process.

The scope of the last two steps is covered in the next section ([5.3.2] ’Functional Enhancements’).

5.3.2 Functional Enhancements

There has been introduced major changes to the ATOLL XBAR mostly related to flow control, which is now credit based, and multiple transfer levels, used for the barrier mechanism as well as the virtual channel methods used for head-of-queue blocking reduction and deadlock avoidance. This work focuses on the following important enhancements, other enhancements like credit-based flow control was just an implementation issue, well known in the scientific society, and therefore omitted.

The important functional enhancements can be coarsely divided in:

1. "Virtual Channels" on page 95
2. "Barriers" on page 105
3. "Routing string format" on page 110

Virtual Channels

One of the lessons learned from the ATOLL network design and its implementation is the significant impact of so called "Head-of-queue" (HOQ) or "Head-of-line" blocking.

It is a known fact that wormhole switched networks are sensible to this effect. [17] ’Frank Ueltzhöffer, "Design, Verification and Physical Implementation of a High-Performance Low-Latency Multi-Level Network Router", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005’ describes this effect in detail. HOQ blocking has an impact on latency and bandwidth of physical channels, if a physical channel is assigned to a route (a message is currently transmitted) and transmission progress is stalled because of some blocking. The blocking can have several reasons from blocked XBAR ports to reception delays at the compute nodes.

Introducing virtual channels ensures that the bandwidth reducing effect of HOQ blocking can be eliminated by re-assigning the physical channels to different routes if the originally assigned route stalls. Possible hardware implementations of virtual channels are well known and also described in [17]. The amount of virtual channels is a design space exploration decision based on the area impact of the additional buffers and the network’s probability of HOQ blocking. The network’s probability of HOQ blocking is strongly depending on its #routes/physical link metric and the message size.

Before digging deeper in the design space exploration and hardware implementation of the different virtual channel arbitration algorithms, please note the difference to ’Virtual Channel Groups’ which are pre-assigned during the route creation algorithm and therefore, do not be arbitrated in this sense. The next section covers ’Virtual Channel Groups’ in detail.

The crucial question is how virtual channels are arbitrated/assigned. Running out of virtual channels or missing to assign a virtual channel where it is needed eliminates the positive effect on the average bandwidth at all.

> *So how would a perfect virtual channel arbitration algorithm behave? When receiving a virtual channel request, the arbitration algorithm would grant this request only if the assignment of the virtual channel increases the throughput of the physical channel. If the physical channel is already utilized 100%, there is no point in assigning an additional virtual channel (at least from a throughput point of view). For example, a perfect arbitration algorithm should not assign a virtual channel to a message that will get blocked in a congested area. Assigning a virtual channel to such a message would only increase link throughput temporarily until the head of the message becomes blocked and all buffers on the way are full. Then, the utilization of the*

> *virtual channel would drop to 0 while preventing the virtual channel from being reused by a message that could otherwise utilize it. Of course, such a virtual channel arbitration algorithm would require complete knowledge of the network and its future behavior, which is not realistic.[17]*

In the context of the EXTOLL network architecture this issue would reduce to the knowledge of the final message destination as routes are not adjusted adaptively. So, if two messages have the same destination node, in the EXTOLL network, they would use the same route. Now, if there is a block somewhere in this route, assigning a new virtual channel to the second message would not help in any way as the blocking factor exists for both. However, as stated before omitting virtual channels at all would hinder messages to different destinations to proceed along the "worm".

An intelligent virtual channel arbitration algorithm is needed, that prevents running out of virtual channels in the case that several messages to the same destination node would use up all available virtual channels but block all at the same blocking cause.

Please refer to Figure 51 below to understand the following illustration. Consider scenario (b) where both messages have the same destination node in a congested or even blocked area. Further consider that message 1 uses virtual channel 1 and message 2 virtual channel 2 in a case where the arbitration algorithm assigns virtual channels as soon as it determines some blocking at the selected OutPort of the XBAR. More messages with the same destination might follow using up all available virtual channels with no positive impact on the link usage. When all available virtual channels are used up another message arrives with a different destination now beeing unable to proceed, although, the physical channel is currently not used as all messages before are stalled.

Now, imagine that the arbitration algorithm will only assign new virtual channels to messages with different destinations. A message like in scenario (c) might use a virtual channel to bypass the blocked message 1.

As the EXTOLL network is source path routed it is quite difficult to determine in hard-

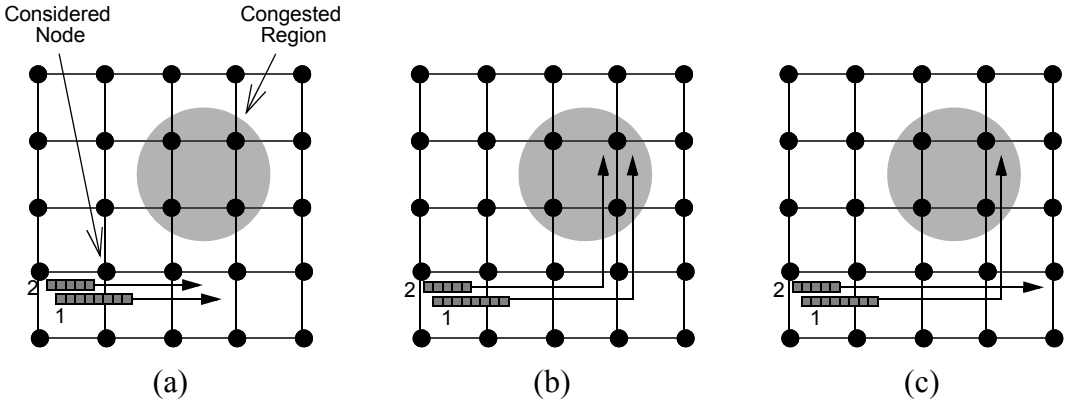


Figure 51: Virtual channel arbitration scenario [17]

ware what messages do have the same destination. This would require a comparison and storage of the entire routing string for each virtual channel. The algorithm to come around the described problems is called the "Smart Arbitration Algorithm".

> *A problem of the smart arbitration scheme is that only the outport knows which of its outgoing virtual channels is bound to which destination node in the network. Therefore, it is necessary to transfer the information about the destination node of a message from crossbar inport to crossbar outport when a virtual channel is being requested.*
>
> *One way to avoid this overhead of transferring the information from inport to outport is to leave part of the arbitration decision up to the inport. In such a simplified smart arbitration scheme, each inport independently checks whether it is already holding an outgoing virtual channel destined to a certain node before it sends a request for a virtual channel to the outport. Therefore, each inport can at hold no more than one outgoing virtual channel destined to a certain node.*
>
> *As the inports are not allowed to share their information, it is possible that there are several outgoing virtual channels destined to the same node in the network. However, those virtual channels are bound to different inports.*
>
> *The hope is that this "imperfect smart arbitration" provides similar performance results at lower implementation costs. This hope is justified as mes-*

sages within a dimension-order routed network have the tendency to stay within a single dimension for a relatively long period of time.

Therefore, most messages going to the same node should also arrive through the same inport, and this case can be handled perfectly by the simplified smart arbitration scheme.[17]

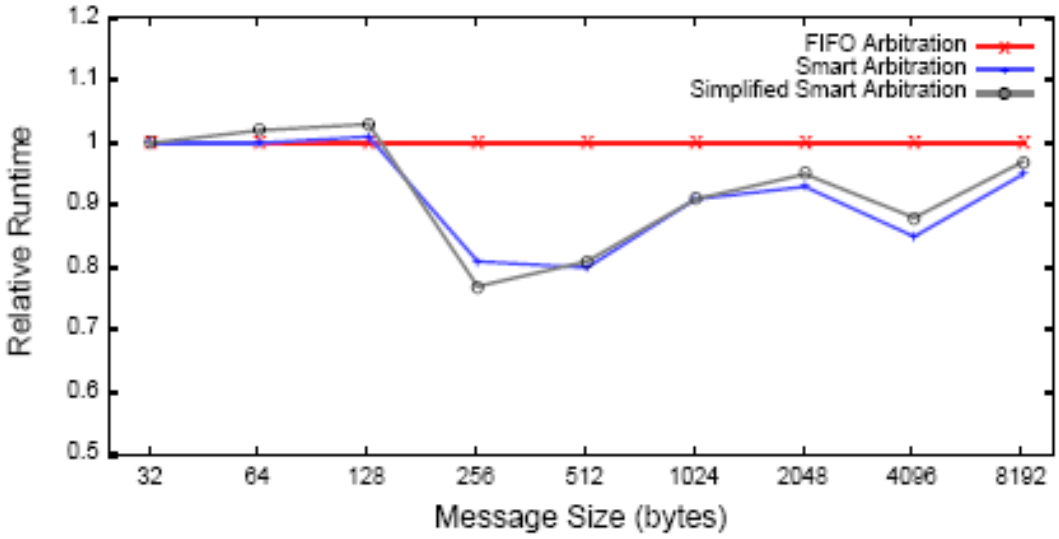


Figure 52: Comparision of Smart Arbitration with the Simplified Smart Arbitration Algorithm [17]

For very small packets, none of the smart arbitration schemes provides any performance advantage over the original first-come-first-serve algorithm. This is not surprising as short messages fit entirely into the inport buffers of the crossbar. Thus, if a message becomes blocked in a node, only the incoming virtual channel remains blocked. All previously used virtual channels are already deallocated and available to other messages.

With increasing message sizes, however, the link packets of a blocked message spread over several inport buffers and block all virtual channels in between them. In this situation, the number of available virtual channel becomes the bottleneck and the smart arbitration schemes can prove their effectiveness.[17]

Again, implementing this algorithm would require a comparision of the complete routing strings and as this is not feasible the following solution has been implemented:

Introducing the Destination Host Tag

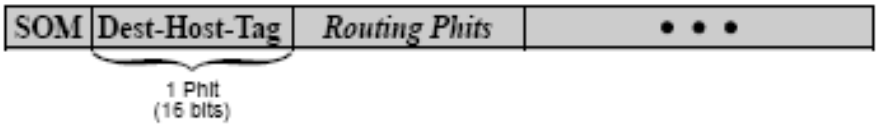


Figure 53: Destination host tag introduced before the actual routing string[17]

The destination host tag consists of 16 bits being able to differentiate 2^{16} different destinations. Although, this amount of hosts is quite large for a cluster system even larger systems are supported with the possibility of double assigned destination host tags. In this case the system will face a decreased performace due to unassigned virtual channels, due to the misinterpretation of the fact that same destination host tags might target different destinations. The arbitration algorithm denies the assignment of additional virtual channels, thus preventing the positive impact of virtual channels to head-of-queue blocking.

For an overview please refer Figure 54: 'Illustration of an EXTOLL message switching virtual channels[17]'. The figure is simplified with regard to the number of XBAR ports.

Channel A, B and C are physical channels connecting LinkPorts of different nodes.

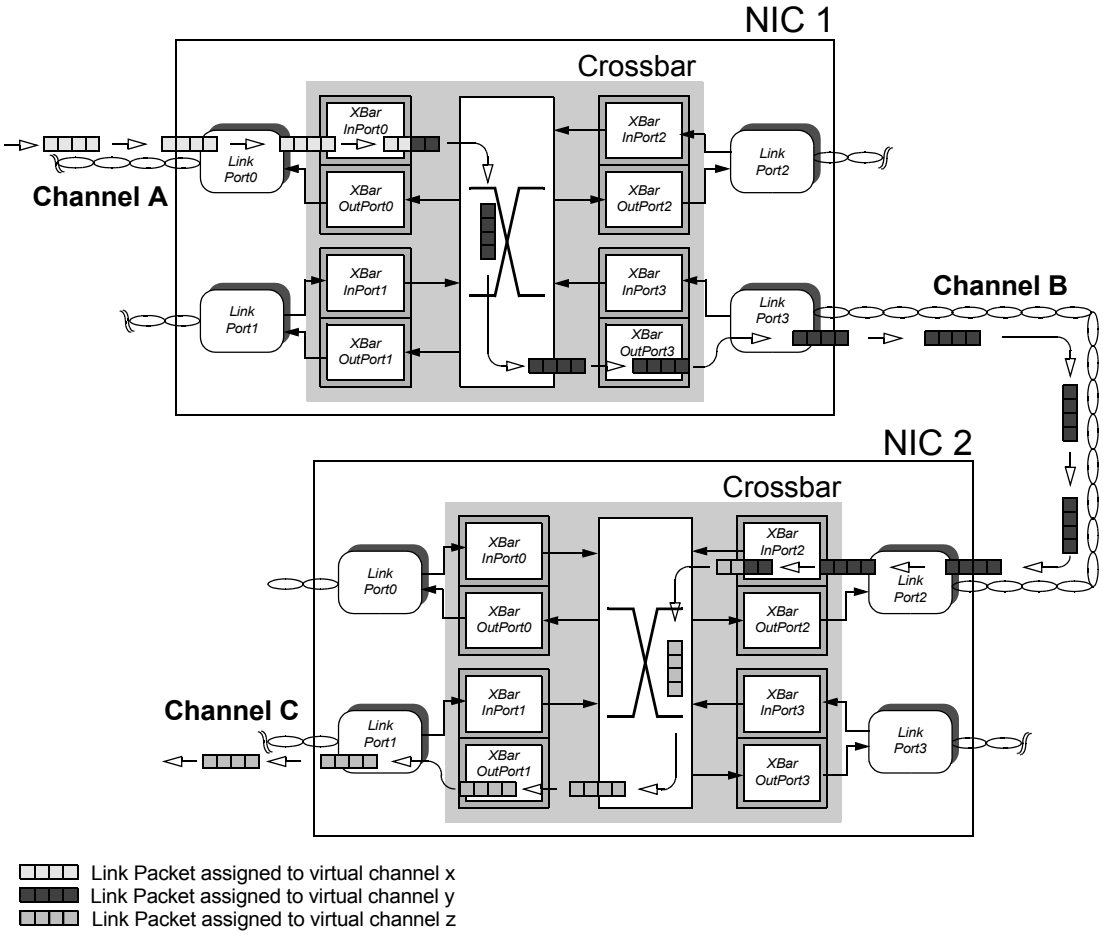
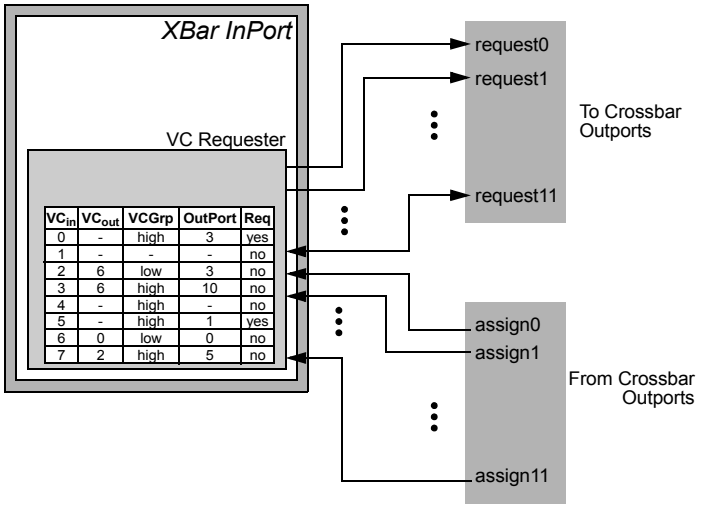


Figure 54: Illustration of an EXTOLL message switching virtual channels[17]

Please note that the virtual channels are switched at each Inport of the XBAR. Here the arbitration decision is made. The unit responsible for issuing request for incoming messages is called Virtual Channel Requester. This unit also stores successful arbitration requests for succeeding message parts in a data structure showed in Figure 55. VC_{in} denotes the virtual channel number where the message enters the InPort and VC_{out} the

virtual channel number leaving the Inport.



VC_{in}	VC_{out}	VCGrp	OutPort	Req
0	-	high	3	yes
1	-	-	-	no
2	6	low	3	no
3	6	high	10	no
4	-	high	-	no
5	-	high	1	yes
6	0	low	0	no
7	2	high	5	no

Figure 55: Generic virtual channel requester for eight incoming virtual channels [17]

The next figure (Figure 56) depicts the simplified OutPort implementation.

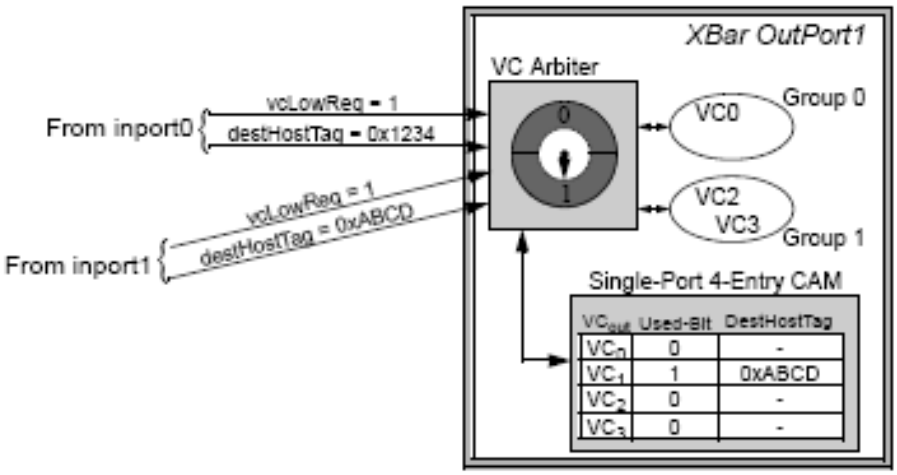


Figure 56: Destination Host Tag aware OutPort [17]

And to understand the necessary connectivity between InPorts and OutPorts please find Figure 57.

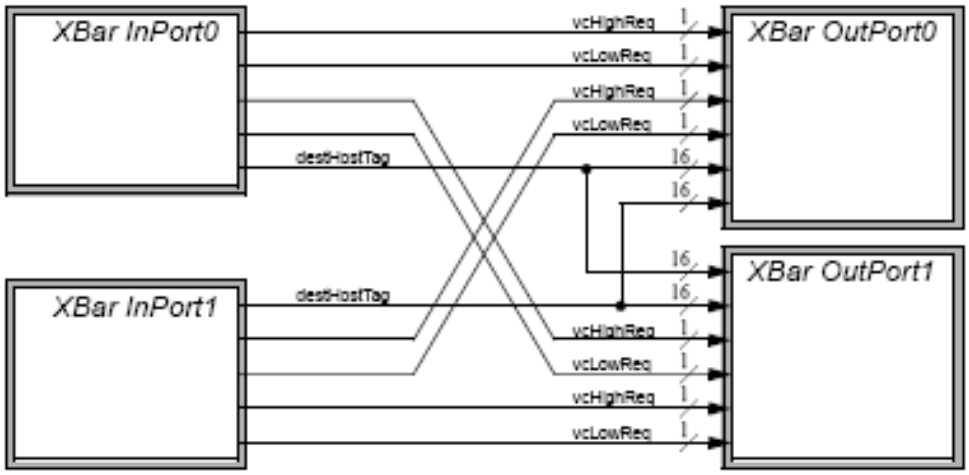
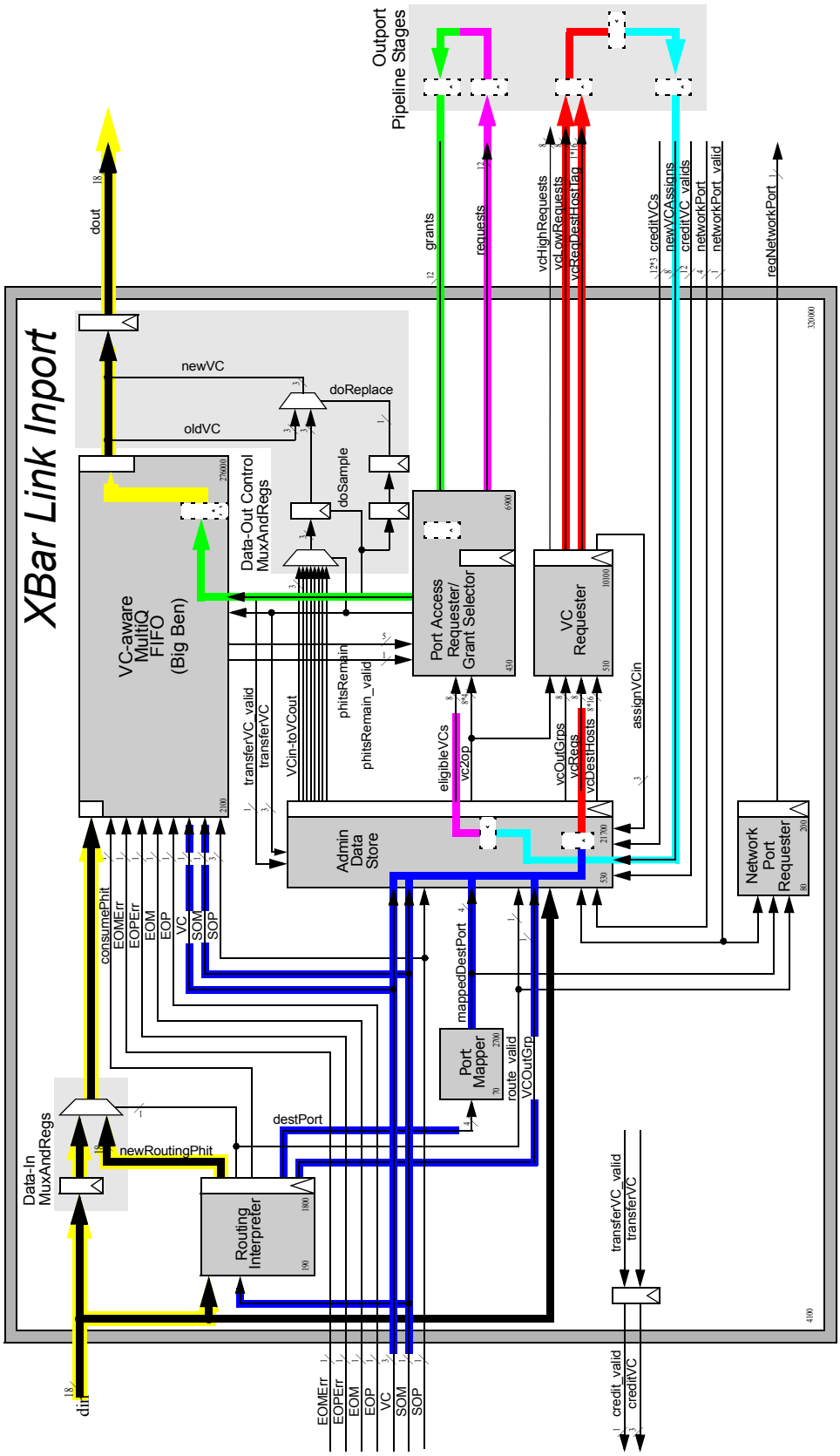


Figure 57: InPort - OutPort connectivity [17]

Just as an example to demonstrate the complexity of the entire XBAR InPort solution please find the next figure. The work [17] contains the entire specification and imple-

mentation details, including timing diagrams and complete block level schematics.



Virtual Channel Groups

Another case where virtual channels improve a network's behaviour is deadlock avoidance. Here, virtual channels define additional routes used to break circles in the channel dependency graph. Topology/routing algorithm combinations that usually are not deadlock free can profit from so called escape channels that break circles in the dependency graph. Please refer to Table 5: 'Deadlock free combinations of topology and routing algorithm' on page 45. Especially interesting for EXTOLL and this work is the combination of 3d-tori and dimension order routing, which is usually not deadlock free. However, introducing virtual channels to add escape routes to the dependency graph solves this problem. A in-depth discussion on how virtual channels impact the channel dependency graph can be found in [18] 'Richard Sohnius, "Creating an Executable Specification using SystemC of a High-Performace Low-Latency Muli-Level Network Router", Diploma Thesis, Computer Architecture Group, 2005'.

Using virtual channels to solve the deadlock issue is a well known method. Here, virtual channels are assigned through the routing algorithm and coded in the routing string (in source path routed networks). The pre-assignment is the fundamental difference to the use of virtual channels avoiding HOQ blocking where from routing hop to routing hop the arbitration decision must be made adaptively.

Therefore, we group the existing virtual channels used to avoid HOQ blocking and name them Virtual Channel Groups. The virtual channel groups itself form another virtual channel used to solve the deadlock issue.

Barriers

Barriers are a fundamental paradigm in parallel computing and have a long history in proprietary parallel computers. Although, there are no existing hardware solutions in commodity SANs like Myrinet or Quadrics. Where no hardware solution is present, barriers are handled in software resulting in performance drawbacks. Therefore, modern supercomputers like the chapter 2.2.5 "IBM Blue Gene BG/L" on page 23 have dedicated networks to support hardware barriers. Due to cost reasons dedicated barrier networks are not feasible for SAN solutions.

The challenge in the context of this work therefore is to use the existing torus network cabelling topology to implement a hardware solution for barriers.

Usually tree topologies are used to scather and gather barrier information. It is important

to broadcast barrier information to every participating node/process without duplicating the information. For tree topologies there exist search algorithms like depth-first search or breadth-first search that ensure visiting all nodes without visiting them twice.

The solution presented in this work is based on the approach to map a tree topology on a torus topology and use this virtual network for barrier distribution.

Figure 58 shows the design space explorated during this work.

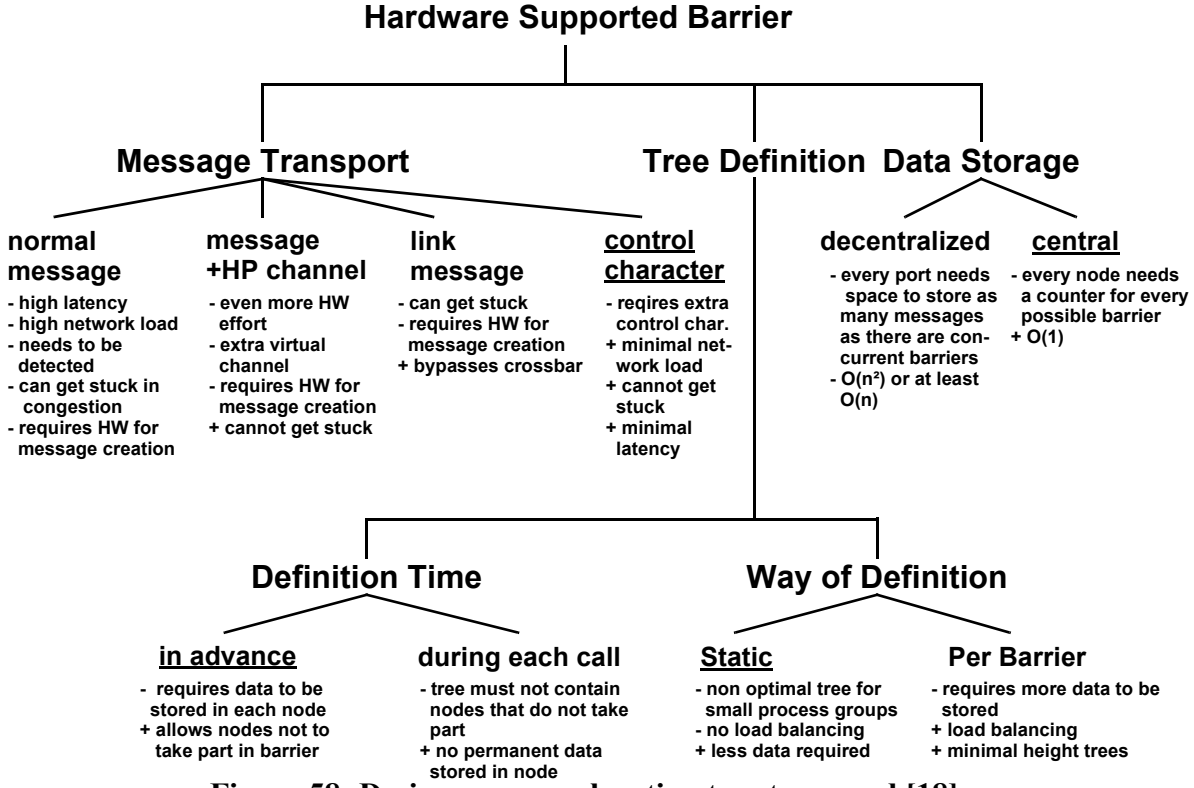


Figure 58: Design space exploration tree traversed [18]

There are well known mathematical algorithms that are able to map trees on tori easily. From the hardware implementation view the virtual tree network is no more that an "up" or "down" tag available at each XBAR port. Exactly one port per XBAR has the up tag asserted (except the root node, which has none) and one or more ports have the down tags asserted (except the leaf nodes, which have none).

During the collect phase all barrier notifications head to the root, during the release phase

to the leaves respectively.

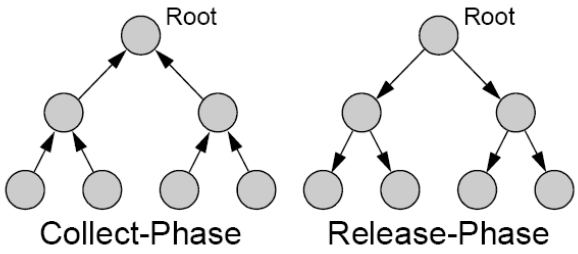


Figure 59: The two phases of the barrier communication [17]

This functionality is considered to be an additional layer on top of the regular message routing capability of the EXTOLL XBAR.

Barrier message flow - Collect-Phase

The start of the barrier operation for the barrier module in the collect phase is either triggered by a HostPort through a process entering a barrier (1) or a barrier message travelling up the tree (2).

(1) A process enters a barrier using the MPI_barrier function. The MPI implementation needs to check if this is the only process in the affected MPI communicator on the current node (because of the possibility of multiple MPI processes running on an SMP node). If this is the only process in the current communicator on this node or all processes of the current communicator entered the barrier, the MPI implementation notifies the NIC. The notification works through the regular communication possibilities.

(2) An up message entered the barrier module. Only if from all available down links an up message has arrived and the current node entered the barrier also (see (1)), an up message is send to the up link. In the hardware implementation the possibility of nodes not taking part in the barrier mechanism is realized also.

Now, if the root receives up messages from all of his down links and the current node, the collect phase has ended and the release phase starts.

Barrier message flow - Release-Phase

The release phase always begins at the root, as described before. Now, the root notifies the MPI implementation of the current node to release the barrier for its processes and sends down messages to all down ports. Every barrier module receiving a down message from an up port notifies the node's MPI implementation and further sends down messages to all of its down ports. If no down ports are available the release phase for this node

and barrier has ended.

The hardware implementation supports 16 different barriers by tagging the barrier messages with a barrier ID.

Depending on the host to NIC communication the barrier module needs to store a virtual port ID per barrier in order know how and who to notify if the barrier releases.

The overall hardware impact of the barrier logic is considered to be very low (in the range of <500 D Flip-Flops for 16 supported barriers. Please refer [18] for the in depth analysis of the hardware implementation and their area impact.

We considered to also virtualize the barrier module in order to support an almost infinite amount of barriers, but due to performance and complexity issues this has been discarded. The barrier logic has to be extremely low latency and therefore, any accesses to the node's main memory needs to be avoided.

In the case an MPI program needs more than 16 different barriers or there are running more than 16 different applications on the network, there is still the possibility to support the additional barrier requirements in software.

Barrier XBAR interfacing

The interfacing to the barrier module heading in the direction to links is pretty simple (Figure 60).

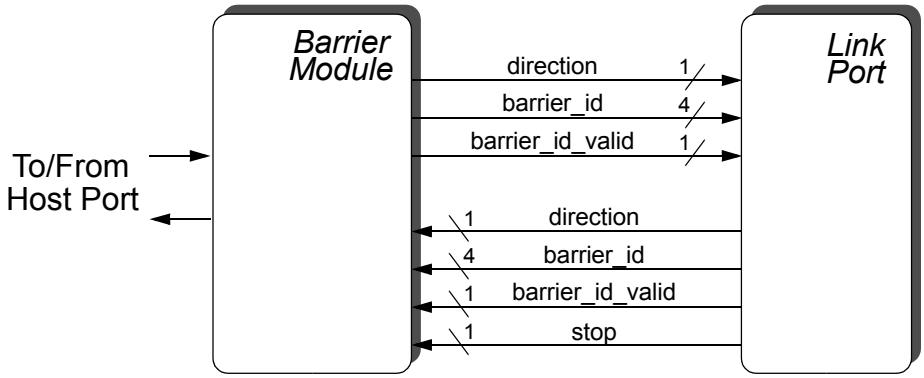


Figure 60: Interfacing to the LinkPort [17]

- direction: Up/Down direction for incoming or outgoing barriers
- barried_id: ID of the transmitted barrier
- barrier_id_valid: Asserted if direction and barrier_id are valid

- stop: Asserted if the link is not able to transmit data

The interface to the HostPort (Figure 61) is nearly as simple.

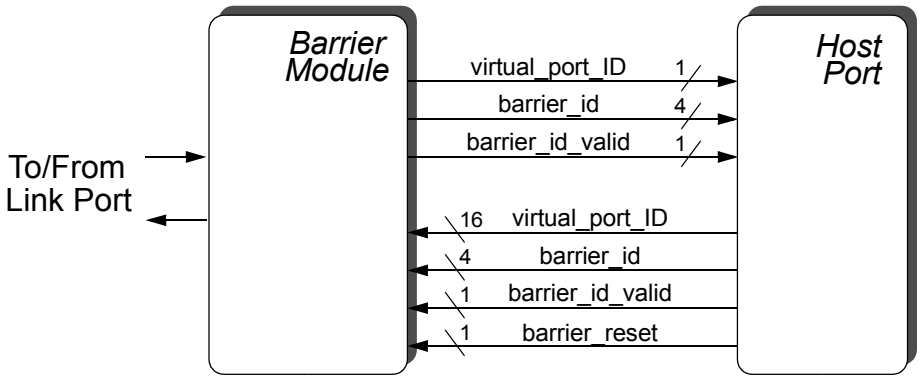


Figure 61: Interfacing to the HostPort

- virtual_port_ID: The virtual port ID of the process that issues a barrier operation or that needs to be notified in the case of a barrier release
- barrier_id: The ID of the barrier to be used (The software needs to arbitrate the available hardware barriers).
- barrier_id_valid: The flow control signal
- barrier_reset: In the case of a software abort due to any reason, the hardware needs to be able to reset a barrier operation (release barrier with error)

Again, for a complete discussion of the design space exploration and the hardware effort analysis please refer to [18].

Debug and Fault Tolerance Features

The diploma thesis working on the complete XBAR IP generator [41] included the entire ATOLL XBAR implementation with parametrizable FIFO structures. Here, also some improvements of the ATOLL XBAR regarding debug functions has been implemented as lessons learned from the first ATOLL silicon start-up tests.

The XBAR now has functionality to:

1. Mask individual ports to be unavailable in order to address faulty nodes
2. Signal the software if a masked port is addressed (IRQ)

3. Enter default detour ports in case of addressing a masked port (automatic route detouring without software interaction)

This functionality enables software controlled routing changes for messages in transfer. By these add-ons now new fault models are available to address node/link breakdowns, that can be further controlled in software.

Furthermore, performance counters has been introduced in order to set the basis for adaptive routing controlled by software. The software is able to read the link utilization and specifically and locally change the eligible routes.

Routing string format

The evolution of the proposed routing string format has under gone several changes due to massive constraints from the architecture and the implementation side.

> *The crossbar and links of the EXTOLL architecture will use 18-bit phits in contrast to the 9-bit phits used by ATOLL.*
>
> *The routing string preceding each ATOLL message is determined by the sender (source path routing). On the way to the recipient of the message, each hop removes the first routing phit of the routing frame. Although this routing scheme is simple, it suffers from the disadvantage that the length of the routing string is directly proportional to the number of hops on the path to the destination node. In large networks and/or in the case of very short messages, this overhead is no longer negligible.*
>
> *The EXTOLL network is designed to achieve high throughput even with very short messages as they are typical for remote direct memory accesses (RDMA). As the EXTOLL phit is twice as large as that of ATOLL, using the same routing scheme as ATOLL would double the routing overhead. This is unac-*

ceptable, and therefore a new routing scheme for EXTOLL has been developed in diploma thesis [18].

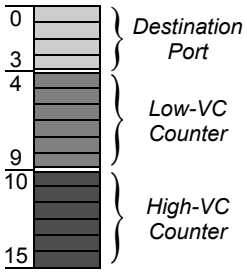


Figure 62: 16-Bit EXTOLL routing phit

Each routing phit will now contain a destination port field and two counters (Figure 62). The destination port field specifies through which outport the message must leave from the current node, whereas the counters specify for how many hops this port needs to be taken. Each time a hop forwards the message to the specified node, the counter is decremented. As Figure 62 depicts, each phit contains two counters: one for the low virtual channel group and one for the high virtual channel group. As long as the low-VC counter is not zero, the message is forwarded on a low virtual channel. As soon as this counter reaches zero while the high-VC counter is still greater than zero, the message remains in the same dimension but is sent through high virtual channels. When both counters have reached zero, the routing phit will be consumed.

Assuming a 3-D torus network and minimal routing, more than 16,000 nodes can be addressed with only 4 routing phits.

During the design phase of the crossbar, slight modifications became necessary to the originally suggested routing scheme.[17]

The routing string is build out of routing phits in the following fashion (Figure 63).

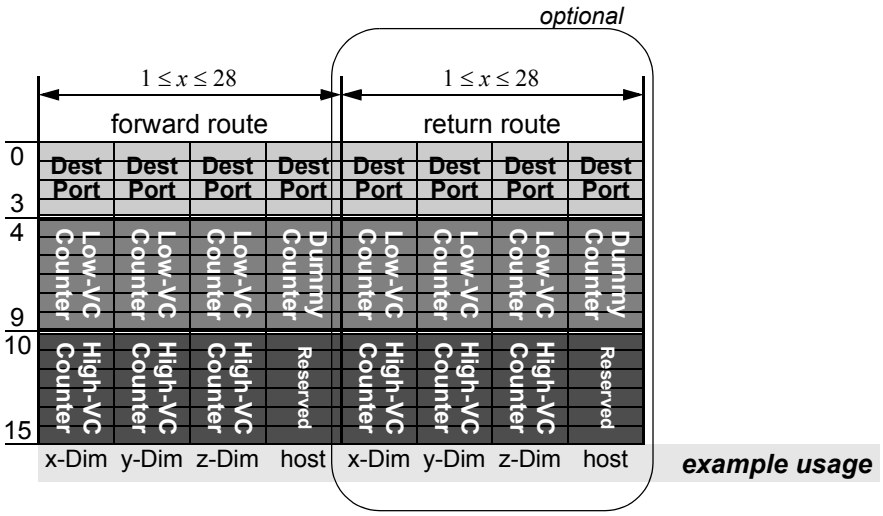


Figure 63: EXTOLL routing string [17]

The return route is optional and will be important for source-destination acknowledgements [42].

The possible values for the DestPort (destination port) field can be found in Table 9. During regular operation messages to be routed to a network port (going out to the host) need to be routed to destination port 15. Here, the crossbar itself determines which physical network port is to be used depending on the availability. For debug purposes the network ports might be addressed directly (ports 8 to 11).

Destination-Port ID	Crossbar Port	Comment
0	Link-Port 0	
1	Link-Port 1	
2	Link-Port 2	
3	Link-Port 3	
4	Link-Port 4	
5	Link-Port 5	
6	Special Purpose Port 0	
7	Special Purpose Port 1	
8	Network Port 0	Only used internally by the crossbar. Must not be used as destination port directly.
9	Network Port 1	
10	Network Port 2	
11	Network Port 3	

Destination-Port ID	Crossbar Port	Comment
12	reserved	
13		
14		
15	Logical Network Port	Crossbar determines which of the physical network ports is available and redirects the message.

Table 9: Routing destination ports

The fields "Low-VC Counter" and "High-VC Counter" represent the virtual channel groups mentioned earlier. Every hop taken reduces the respective counter by one. The message stays in the high virtual channel group as long as the High-VC Counter counter does not equal zero. If both counters are zero the destination dimension is reached and the routing phit is consumed. As soon as the phits for each of the three dimensions are consumed, the host phit denotes if the message goes to one of the four networks ports (to the node) or to one of the two special purpose ports. At this step the networks ports are never addressed directly, instead the "Logical Network Port" (Destination Port ID = 15) is addressed to let the XBAR itself choose the actual network port depending on availability. The function of the special purpose ports is described in the following section.

The Reserved field contains information for the HostPort that includes the possible existence of a return route and other information for the special purpose ports. Please refer to [42] for more details on possible usage of this field.

Broadcasts / Multicast - Special Purpose Ports

There are two special purpose ports considered in the EXTOLL XBAR. They have special functions available not related to the common user processes taking part in the parallel application.

The so called "Debug Port" might be used to remove misrouted or erant messages from the network. The debug port is implemented like the former ATOLL DMA function of a host port spooling messages to a dedicated pinned supervisor page in main memory. The daemon software is responsible to screen this area and decide how to process the incoming messages. For example, the XBAR outport to a faulty link might be masked and messages to this port would then be redirected to the debug port to be processed by the software.

The so called "Multicast Port" is considered to spool incoming multicast messages to

main memory and automatically resend them to the different ports without software interaction. This function is not implemented yet.

5.4 Hostinterface and EXTOLL block level communication

This section covers the communication architecture of the building blocks of EXTOLL as well as the communication interface architecture to the host. Depending on the actual location of the NIC and the protocol used to connect to the host, the host interface is required to support different features. The PCI Bus for example relies during burst cycles on a continuous data stream and also requires the "forecast" of the length of the burst cycle. During the course of this work the requirements from several node bus protocols has been taken into account to design a most generic and performant architecture.

Mainly two diploma thesises contributed to this section:

1. ’Thomas Schlichter, "Exploration of Hard- and Software Requirements for one-sided, zero copy user level Communication and its Implementation", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2003’[14]
2. ’Timo Sponer, "Development, Verification and Integration of a Processing Unit in the Communication Function of a SAN Device in SystemC", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005’[32]

Atomic virtual command queue modification

One of the most challenging issues was the atomic trigger functionality for virtual devices via a bus protocol not supporting atomic operations. As EXTOLL implements virtual devices [42], the question was how to enter commands in the virtual command queue responsible for triggering operations of virtual devices. Task switching by the operating system introduces the requirement for atomic accesses. For a detailed discussion of the design space please refer [14].

The solution is to use read accesses to virtual channel dedicated mapped pages of the NIC and refer to the read value as an acknowledge or negative acknowledge signal. The address of the referred page indicates the virtual device ID and the operation being triggered. The response on this request signals if the request has been entered in the virtual command queue successfully or not. This mechanism is atomic as it uses a single access to the device, communicates the virtual device ID as well as the command type using the

referred address. Also, lowest latency is ensured to issue the communication command.

TLB coherence

Another challenge was to keep the TLB (Transaction Look aside Buffer) of the NIC consistent to the CPU's TLB. It was necessary to have a TLB located on the NIC to implement zero copy remote memory accesses (RMA) for virtual to physical address translation. Also in [14] a hook mechanism was developed for the LINUX operating system to signal a TLB flush or modification to the NIC concurrently to the CPU. This was submitted as a change request to the LINUX kernel community and accepted as soon as the hardware is available. The TLB hook mechanism was tested in an FPGA environment developed by 'Matthias Scheerer, "Definition and Implementation of a Hardware Abstraction Layer (HAL) for an ASIC-Prototyping Station using a 64Bit/66MHz PCI interfaced FPGA", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2002'. Please refer to chapter 4.3: 'FPGA based ASIC prototyping' on page 71 for a detailed discussion of the hardware platform.

5.4.1 Intermodule Communication Architecture

The requirements to be put on the on chip communication architecture has been derived from the following [32]:

1. A huge amount of functional units that can even increase if future modules will be developed. So the design has to be extendable.
2. An arbitrary distance between the functional units and the communication structures on the die. This is necessary as a floorplaning step has not been performed. When the floorplaning reveals the need for a pipeline stage the design has to cope with such a situation.
3. A generic interface towards the peripheral bus. As the support for a specific peripheral bus has not been decided a generic interface allows a connection to a wide range of peripheral busses.

There are three modules involved in the communication with the host (Figure 64):

1. Host Interface: Responsible for the communication to the connected host bus. It allows the remaining EXTOLL system to be as independent as possible from a specific peripheral bus

2. Master Interface: Responsible for accesses initiated through the EXTOLL device
3. Slave Interface: Responsible for accesses initiated through the host

Depending on the location of the NIC the South Bridge might be omitted.

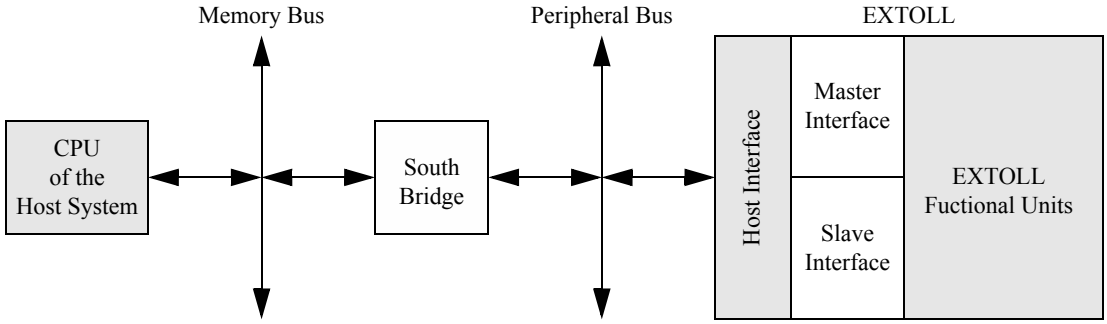


Figure 64: EXTOLL host communication [32]

In order to derive the communication constraints from the several functional units, we grouped them in bunches depending on their functional task (Figure 65):

- The modules present in the Initiator Path group start network transmissions initiated by applications running on the local node.
- Messages that are received from the network are forwarded to the addressed application by a module of the Completer Path group.
- The Cache group contains the context cache, the window descriptor cache and the routing cache. The caches buffer the EXTOLL data structures that reside in the main memory. A functional unit always requests a data structure from the appropriate cache. In case of a cache miss the cache fetches the data structure from the main memory, buffers it and hands it over to the functional unit.

- The Response Path group contains all modules that injects a message into the network as a reaction to a received message. So the functional units respond to received messages without the interaction of an application. [32]

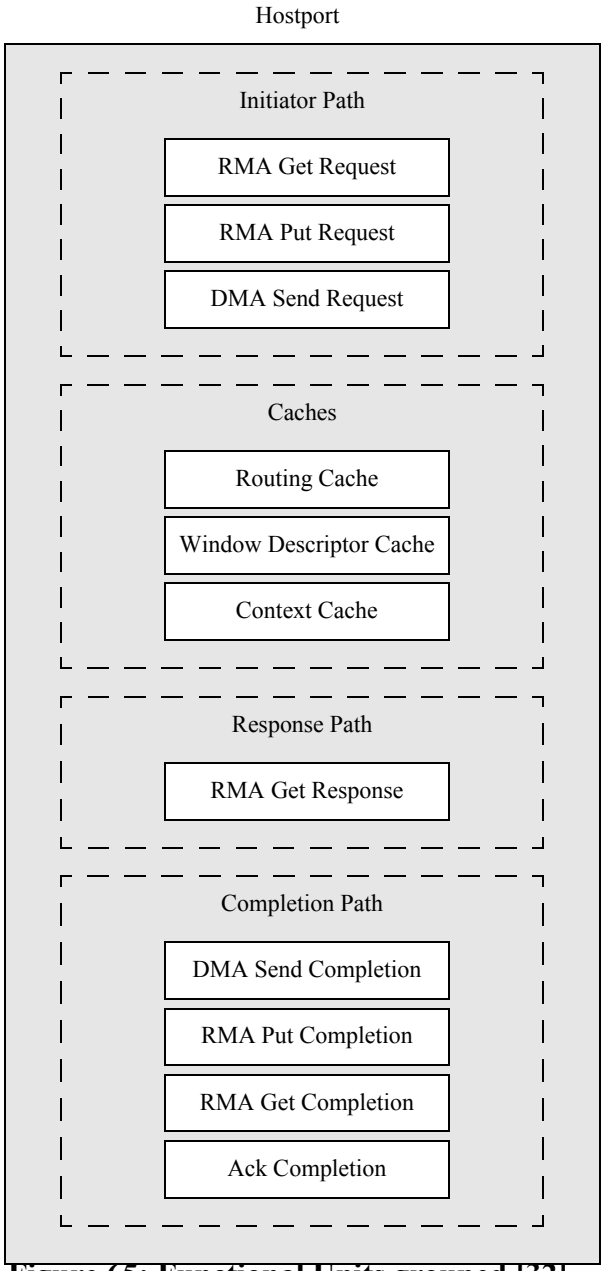


Figure 65: Functional Units grouped [32]

From these groups we derived the communication pattern required for each functional unit. Figure 66 depicts the initiator path; Figure 67 the completion and response path. The light grey boxes denote the amount of data and transferred for each step as well as the communication direction (master/slave). Please refer [32] for a detailed description

of each step.

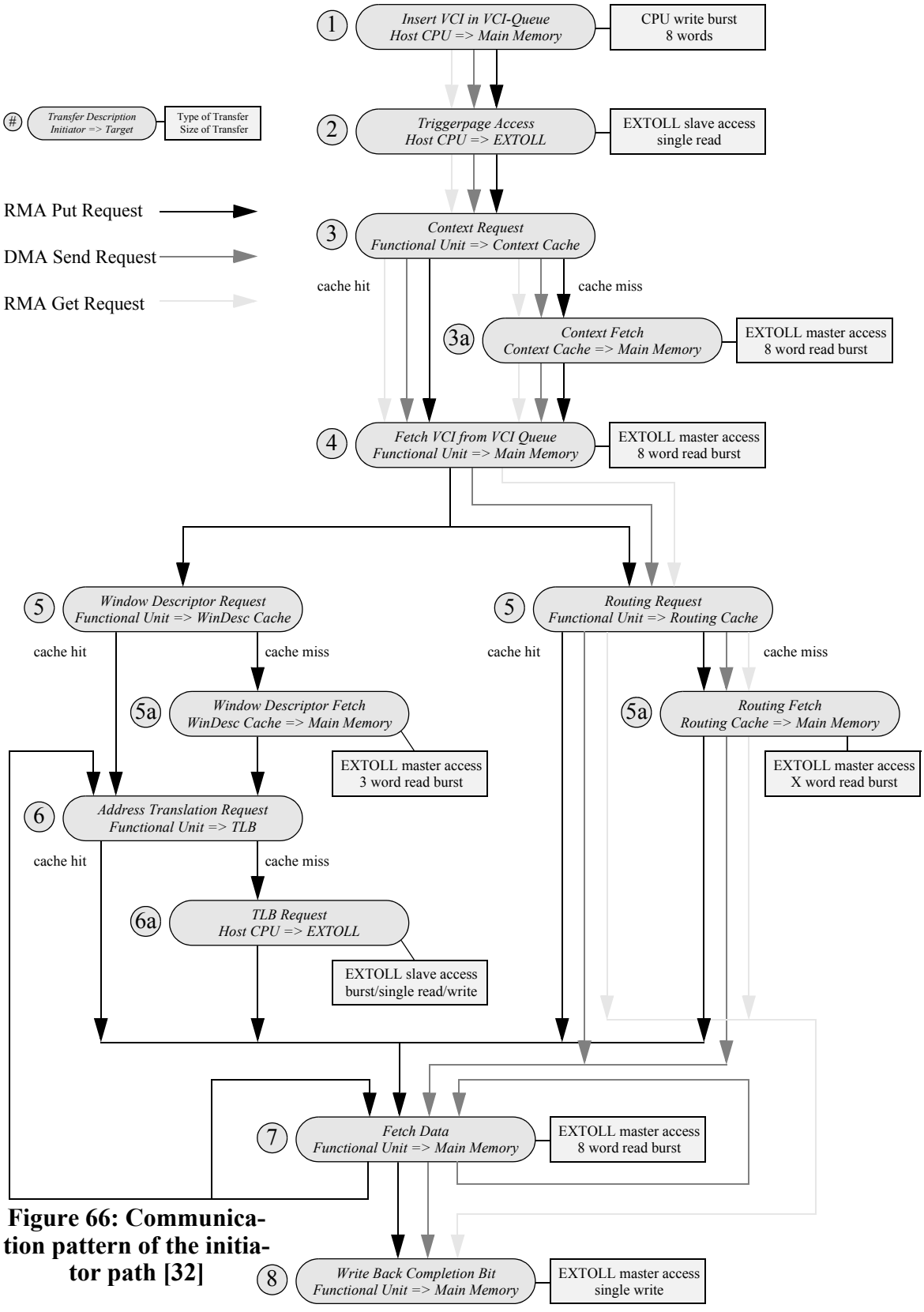


Figure 66: Communication pattern of the initiator path [32]

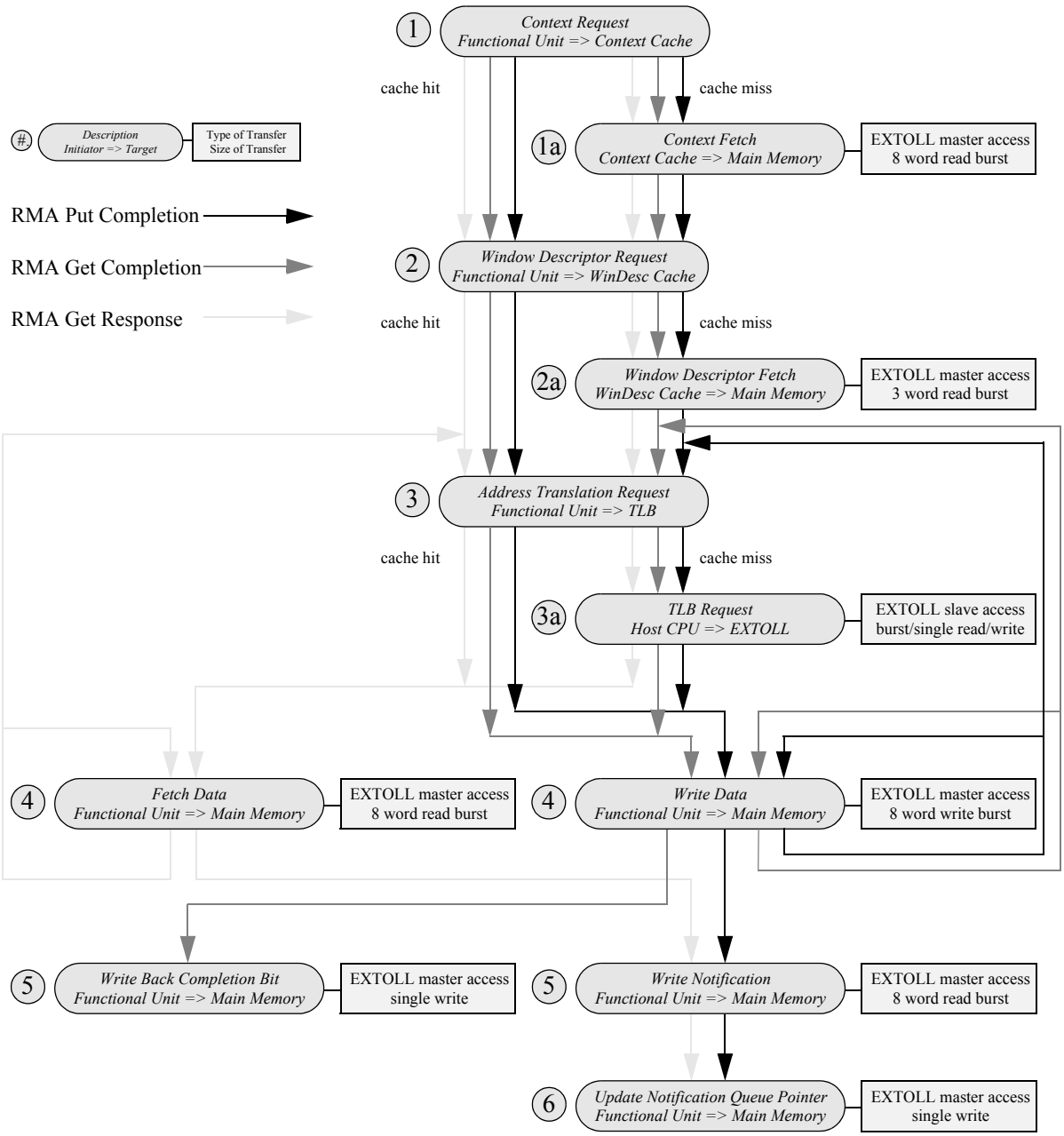


Figure 67: Communication pattern of the completion and response paths [32]

5.4.2 The Slave interface Unit

The following figure depicts the entire design space exploration tree for the slave inter-

face observed:

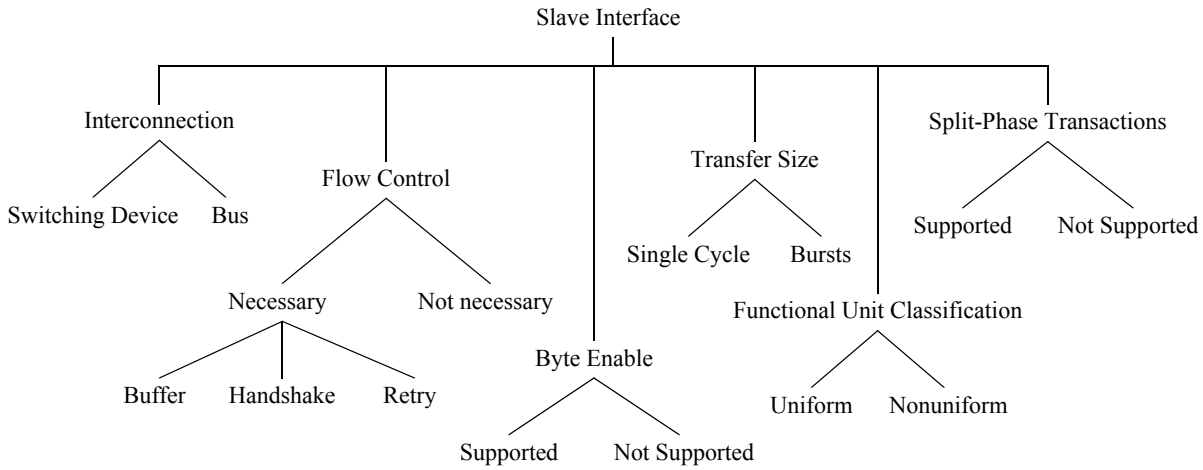


Figure 68: Design Space Exploration tree of the Slave Interface [32]

As the result the Slave Interface Unit is divided in three main modules (Figure 69):

- The Request Path accepts transaction requests from the two masters. It converts the requests into an internal format and passes the requests one after the other to the Socket Tray. If the request is a write request the Request Path also accepts the write data, buffers it and passes the data along with the request to the Socket Tray.
- The Socket Tray provides an interface for each Slave Module. In case of an incoming request it forwards the transaction to the appropriate Slave Module. In case of a read request the Socket Tray delivers the read data to the Response Path.
- The Response Path collects the read data from the Slave Modules and delivers it to the master that has requested the data.[32]

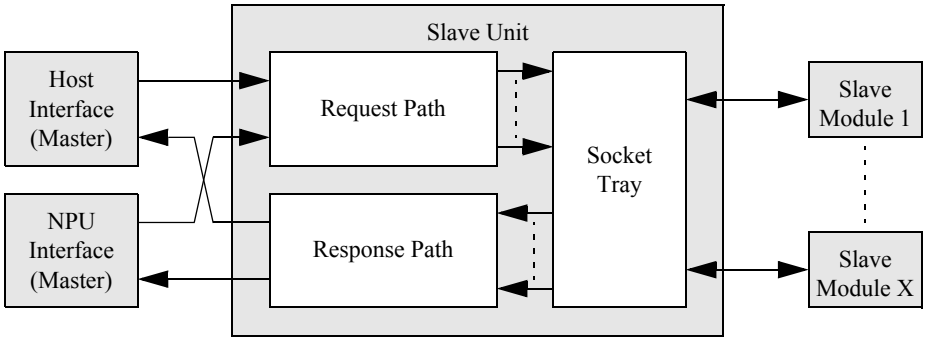


Figure 69: The Slave Interface Unit [32]

Usually there is only one master connected to the Slave Interface Unit, which is the host's CPU. However, in our case, where a Network Processing Unit works concurrently

to the node's CPU, we need two interfaces. As both have identical features and constraints the interface is just duplicated.

In order to increase performance and concurrency the Slave Interface Unit supports split-phase transactions from both masters.

The Request Path

The main elements of the Request Path are the Job Generator Unit, the Job Queue, the Write Data Buffer and the Issue Unit all described below in more detail.

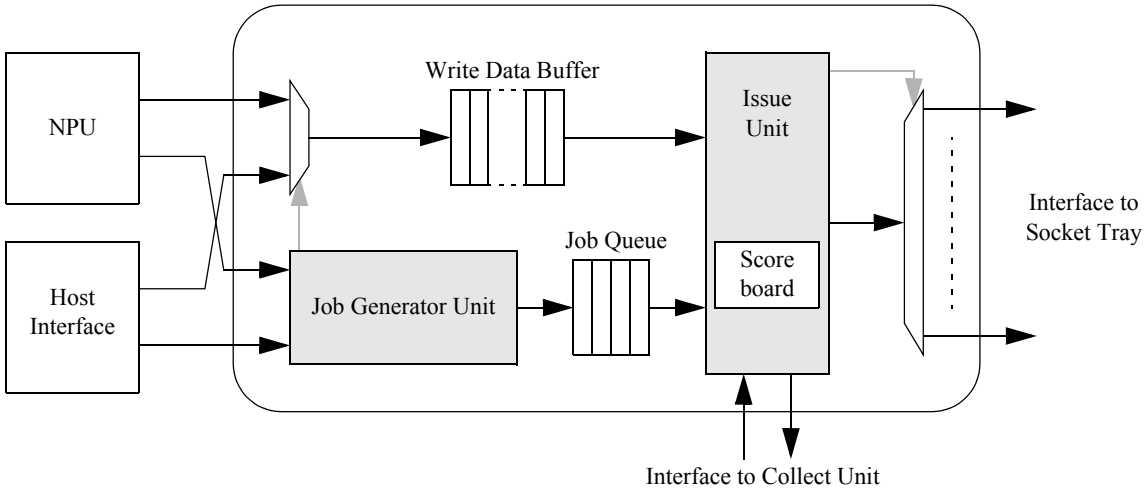


Figure 70: The Request Path [32]

The Job Queue used to enable concurrency of consecutive requests contains all issued requests. Each element of the Job Queue represents one request and is defined in Table 10:

Field Name	Bitwidth	Description
Command	2	Specifies whether job is read or write transaction
Master ID	1	Number of the Master that has initiated the request.
Slave ID	5	Number of the Slave Module that represents the destination of a transaction
Address	64	Start address of transaction
Size	5	Amount of 64-bit word involved in the transfer
Sequence ID	5	Split-phase transaction tag associated with the transaction

Table 10: Element of the Job Queue [32]

If the request is a write request the corresponding data is stored in the Write Data Buffer.

The SlaveID is derived from the requested address by the Job Generator Unit.

The Job Generator Unit (Figure 72) contains an Address Decoder responsible for assign-

ing address spaces to the slave modules. Depending on the actual address space distribution the address translation is perfomed resulting in a SlaveID and a local slave address.

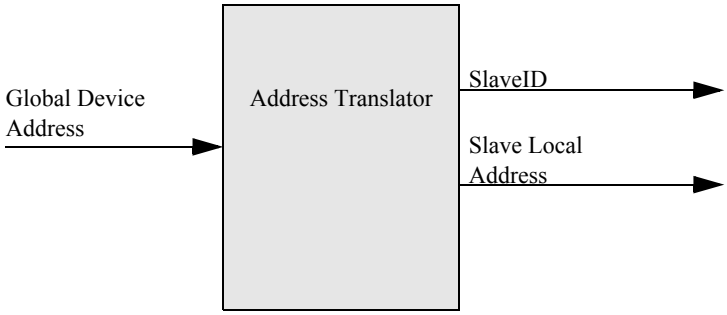


Figure 71: The Address Translator

Usually the address space is divided in chunks of 2^n. If the address spaces for each Slave Module would be equal, the resulting hardware of the Address Translator just would be a bit split. Consider for example that each Slave Module gets an address space of 2^9 (512) Bytes. Then, in Verilog notation the SlaveID would be:

```
assign SlaveID = Global Device Address [63:9]
```

and the Slave Local Address:

```
assign Slave Local Address = Global Device Address [8:0]
```

However, the hardware effort strongly depends on how regular the address space distribution is. Also important to consider is the support from the peripheral bus side, where the available address space for the entire device might be very limited.

In general, the Slave Local Address is the Global Device Address minus the Slave Start Address. Only if the entire Slave Modules are well defined the creation of the address

space distribution makes sense.

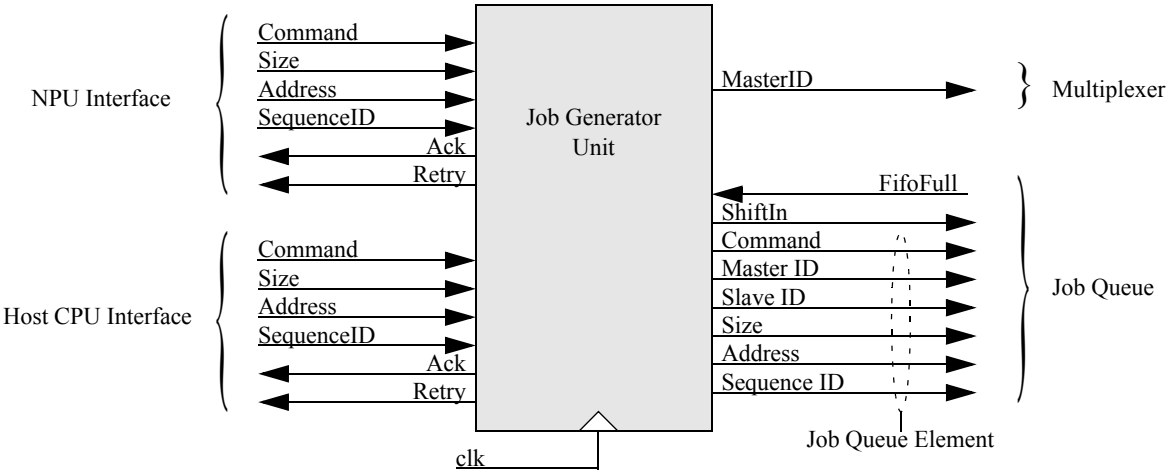


Figure 72: The Job Generator Unit [32]

The Job Generator Unit further is responsible for the arbitration of the two masters. The acknowledge (Ack) and retry (Retry) signals are used to control the data flow. The requesting signals need to be asserted as long as the Job Generator Unit asserts either the Ack signal or the Retry signal. A request will be acknowledged as long as there is space in the Job Queue and if it is a write request there is enough space in the Write Buffer.

A detailed timing diagram for both read and write requests can be found in [32].

The Issue Unit shown in Figure 73 is pulling requests from the Job Queue dispatching them to the available Slave Modules:

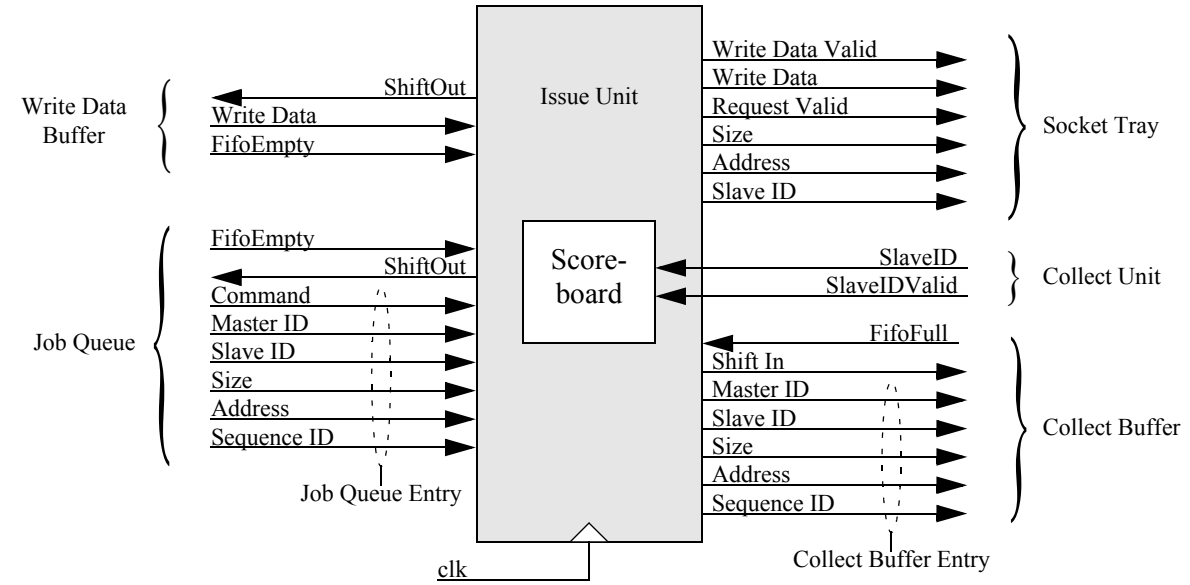


Figure 73: The Issue Unit [32]

The availability of the Slave Modules can be derived from the Score Board entries consisting of the SlaveID and the pending read data values. Furthermore, the Issue Unit interfaces to the Collect Buffer (part of the Response Path) to enable the correct assignment of split-phase read requests to the requesting master.

The Socket Tray

The Socket Tray contains an array of Slave Sockets (Figure 74) providing a point-to-point connection to each Slave Module. The Slave Sockets run an OCP (On Chip Protocol) protocol [46].

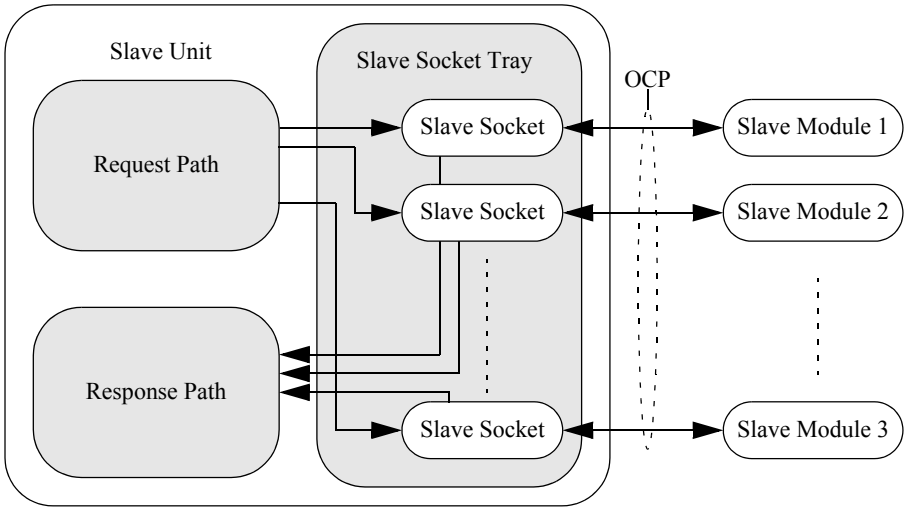


Figure 74: The Socket Tray [32]

Each Slave Socket (Figure 75) has three interfaces, one to the Request Path (Issue Unit) accepting requests and forwarding them to the Slave Modules, one to the Response Path providing read responses and one to the Slave Modules themselves

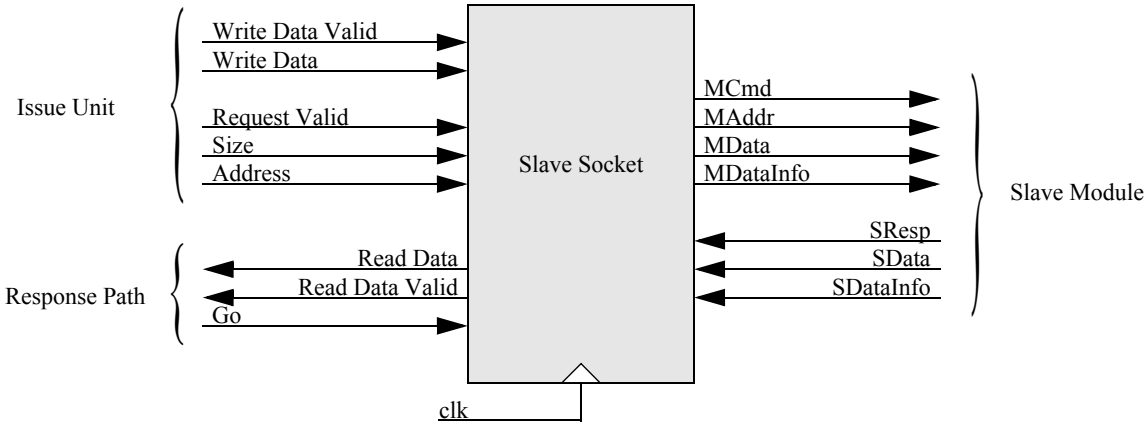


Figure 75: The Slave Socket [32]

In order to decrease the complexity of the Slave Module's interface, we decided to translate burst requests into single requests by the Slave Socket. This also abolishes the need for dedicated flow control signals. So, the Slave Socket only issues commands to the Slave Modules if it has been signalled by the Go signal from the Response Path. Details of the OCP signalling and the Slave Socket implementation can be found in [46] and [32] respectively.

The Response Path

The Response Path uses the Collect Unit fed by the Collect Buffer to collect response data from the Socket Trays (Figure 76).

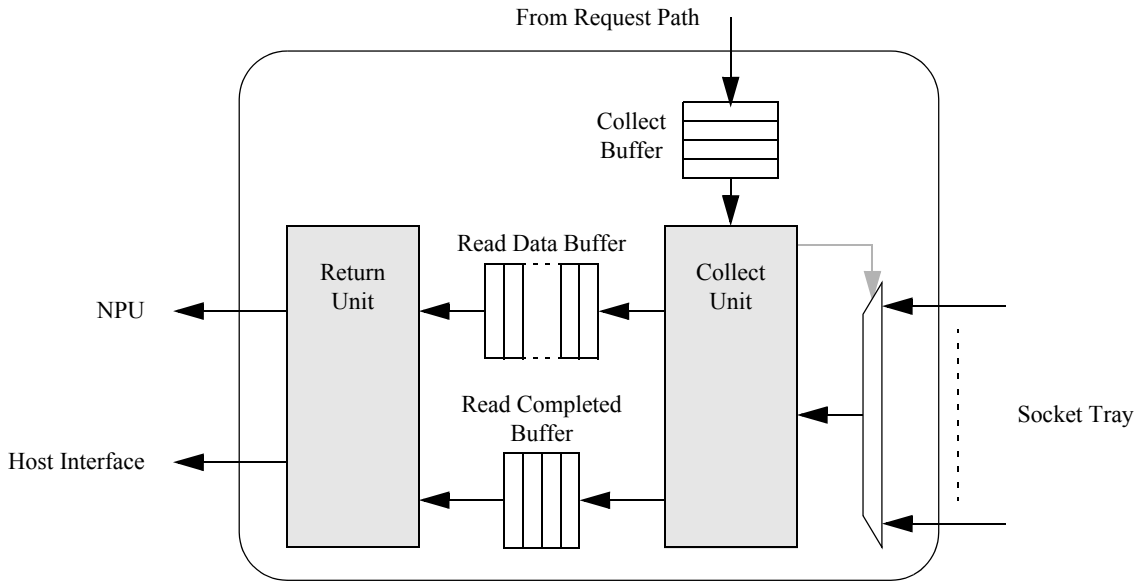


Figure 76: The Response Path [32]

Each entry of the Collect Buffer (Table 11) represents the response to one request:

Field Name	Bitwidth	Description
Master ID	1	The number of the Master that has initiated the request
Slave ID	5	The number of the Slave Module that has pending read data values
Size	5	The amount of 64-bit words that has to be retrieved from the Slave Module
Sequence ID	5	The split-phase transaction tag associated with the transaction
Address	64	The address of the first read data value

Table 11: Content of a Collect Buffer entry [32]

It identifies the requesting master, the target slave as well as the SequenceID for tagged split-phase transactions. The Collect Unit graps an entry and controls the multiplexer to target the correct Slave Socket in the Socket Tray. For each data word transferred it is-

sues the SlaveID to the Scoreboard in the Issue Unit of the Request Path in order to keep track of the outstanding read data values and stores the data in the Read Data Buffer. As soon as the transfer is complete it feeds the Read Completed Buffer with the transaction information (same as the content of a Collect Buffer Entry without the SlaveID, which is no longer needed).

It is necessary to store the complete transaction in the Read Data Buffer and wait for the end of the transaction due to possible constraints from the peripheral bus not to interrupt burst transfers. The PCI-X protocol for example requires this behaviour.

The Return Unit (Figure 77) waits for entries in the Read Completed Buffer and issues the response to the appropriate master:

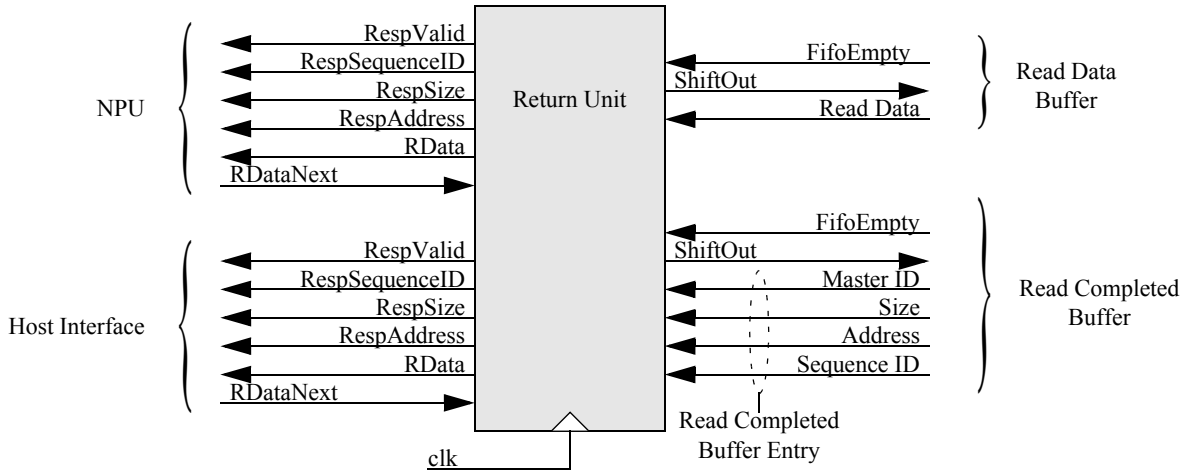


Figure 77: The Return Unit [32]

Detailed timing diagrams can be found in [32].

5.4.3 The Master Interface Unit

The master interface unit contains of three submodules:

- Request Path: Buffers all requests from a Master Module and forwards them to the host interface.
- Response Path: Handles all incoming read data values that are delivered by the host interface. It determines the Master Module that has requested the data and forwards the data to the Socket Tray.
- Socket Tray: Provides an array of uniform interfaces or sockets for the Master Modules. If a Master Module requests a data packet transfer the Socket Tray forwards this

request to the Request Path. In case of a read request the Socket Tray forwards the incoming read data to the Master Module.[32]

The "Master Modules" refer to each functional unit with master functionality, i.e. RMA put request, RMA get request, DMA send request or even the NPU.

This keeps the architecture extendable and generic. In order to support concurrency of the different Master Modules the Master Interface Unit is capable of split-phase transactions. The Socket Tray is responsible to handle these transactions and forwards split-phase replies to the dedicated Master Module. Figure 78 shows the submodules of the

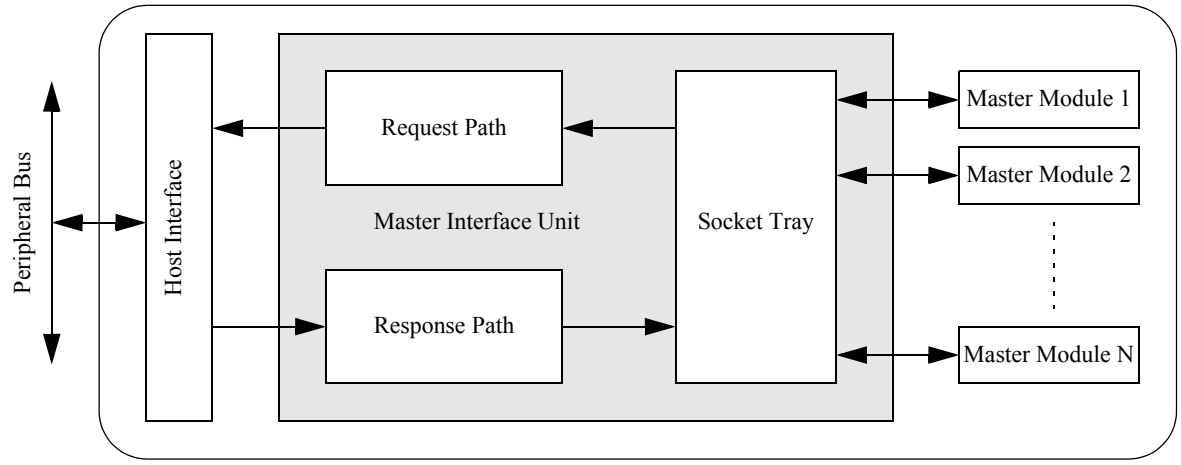


Figure 78: The Master Interface Unit [32]

Master Interface Unit.

The Socket Tray

The Socket Tray (Figure 79) contains an array of Master Sockets, where each Master Socket can be regarded as an adaptor to a Master Module. Also, the Socket Tray is re-

sponsible for the data handling to and from the Request and Response Paths.

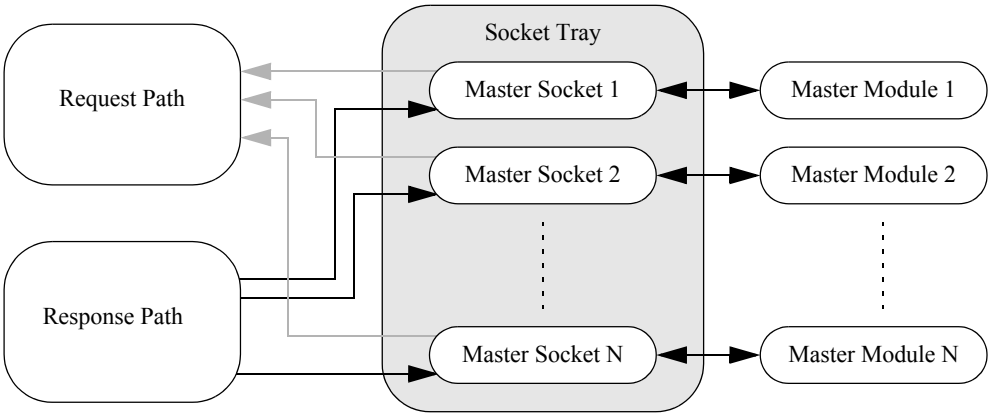


Figure 79: The Socket Tray [32]

The interface of a Master Socket is shown in Figure 80:

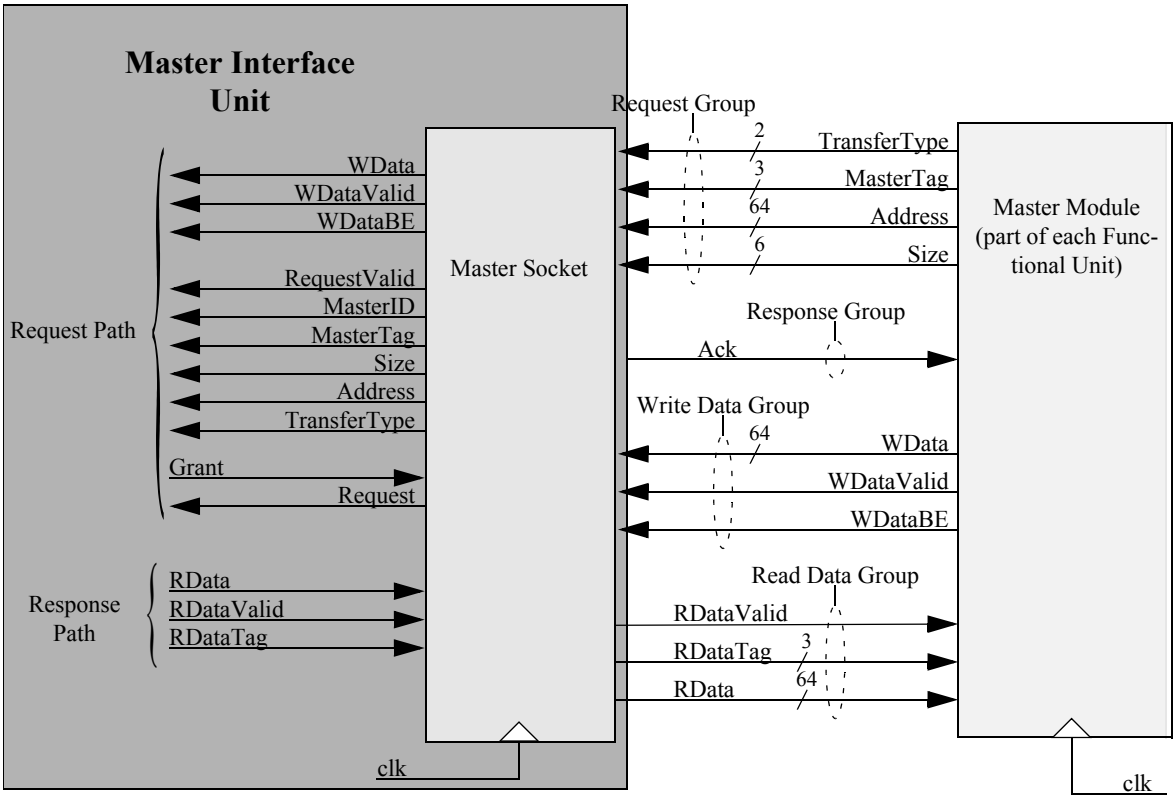


Figure 80: Interfacing of a Master Socket

The protocol to the Master Modules is designed to accept pipeline stages introduced to allow signals with a net delay more than the cycle time. With this feature the global routing can be relaxed. Further architecture decision can now be made independently from the floorplan.

The split-phase transaction protocol supports up to eight outstanding read requests, tagged by the signal MasterTag (Figure 80) handled by a Master Module.

Each Master Module needs to accept incoming data as requested, so there is no buffer space available in the Master Interface Unit.

The signals of the Request Group are marked valid throughout the TransferType signal encoded as follows:

Command	Description	Bit Encoding
00	No Request	IDLE
01	Master indicates read request	READ
10	Master indicates write request	WRITE
11	-	reserved

All signals of the request group are invalid if the coding of the TransferType signal is IDLE.

Write Transactions

A Master Module starts a write transaction by asserting the WRITE command on the TransferType signal bus together with the address and the size of the transfer. The Master Tag signal is ignored as it is only used for tagging split-phase read transactions. After the Master Socked signals the acknowledge of the request by a one cycle assertion of the Ack signal, the Master Module writes the data on the WData bus. The WDataValid signal marks the data as valid; the WDataBE signals are the corresponding byte enable signals. As soon as the transaction is completed the signals of the request group must change to IDLE again.

Read Transactions

A Master Module starts a read transaction by asserting the READ command on the TransferType signal bus together with the address and the size of the transfer. Furthermore, it has to tag the request with a MasterTag in order to assign the later data transfer to the corresponding request. After the Master Socked signals the acknowledge of the request by a one cycle assertion of the Ack signal, the Master Module might issue further requests. When the data arrives valid data is marked by the RDataValid signal and the RDataTag. Using the RDataTag the incoming data can be assigned to the corresponding read request. The Master Modules are responsible to handle the split-phase transaction tags appropriately.

Detailed timing diagrams of write and read transactions can be found in [32].

The Request Path

The Request Path arbitrates and collects the requests from the Socket Tray (Figure 79). Please find Figure 81 for a block diagram of the Request Path.

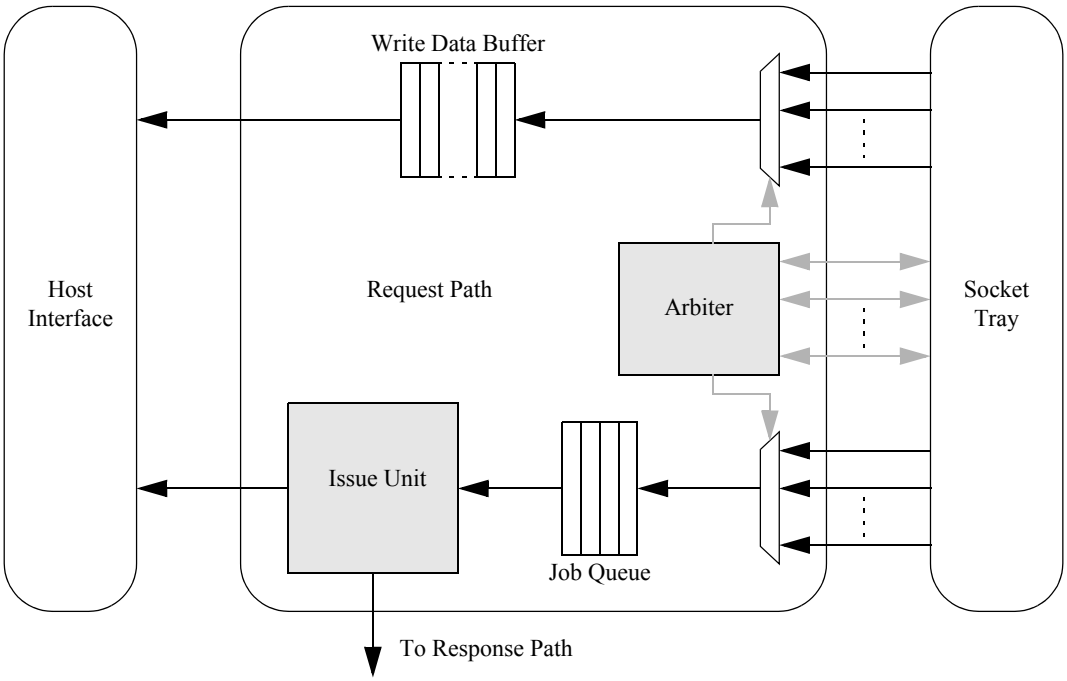


Figure 81: The Request Path [32]

The Job Queue stores the requests from the Master Modules. Each Job Queue Element has the following structure (Table 12):

Field	Bitwidth	Description
Master ID	5	The number of the Master Module that initiates the transaction
Master Tag	3	A tag associated with request
Size	6	The amount of bytes involved in the transaction
IsRead	1	The type of the transfer, i.e. read or write
Address	64	The start address of the transfer

Table 12: Format of a Job Queue Element

The MasterID field was introduced by the Socket Tray, where the rest of the information directly comes from the Master Modules. By introducing a FIFO for these requests concurrency between the Master Modules is possible.

The Write Data Buffer stores (in the case of a write request) the data corresponding to

the request represented by the Job Queue Entry. This FIFO is needed because of the constraint from several bus protocols i.e. PCI-X to deliver burst data continuously with minimum delay/latency.

The Arbiter issues grants for requesting Master Sockets if there is a free entry in the Job Queue and space for at least one maximum transfer unit in the Write Data Buffer. Currently, the maximum transfer unit equals the size of a link packet (64bytes) but this is free to define as soon as the bus connected to the Host Interface is fixed. Furthermore, the Arbiter notifies which Master Socket has been granted to add the MasterID to the request and to control the multiplexers (Figure 82 and Figure 81).

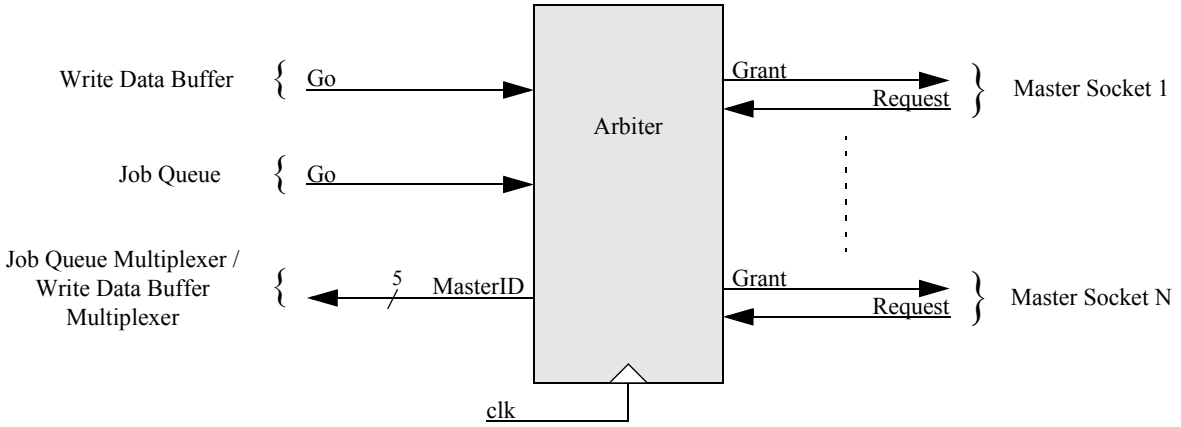


Figure 82: The Arbiter [32]

The Issue Unit compiles the necessary interfacing signals from a Job Queue Entry (Fig-

ure 83):

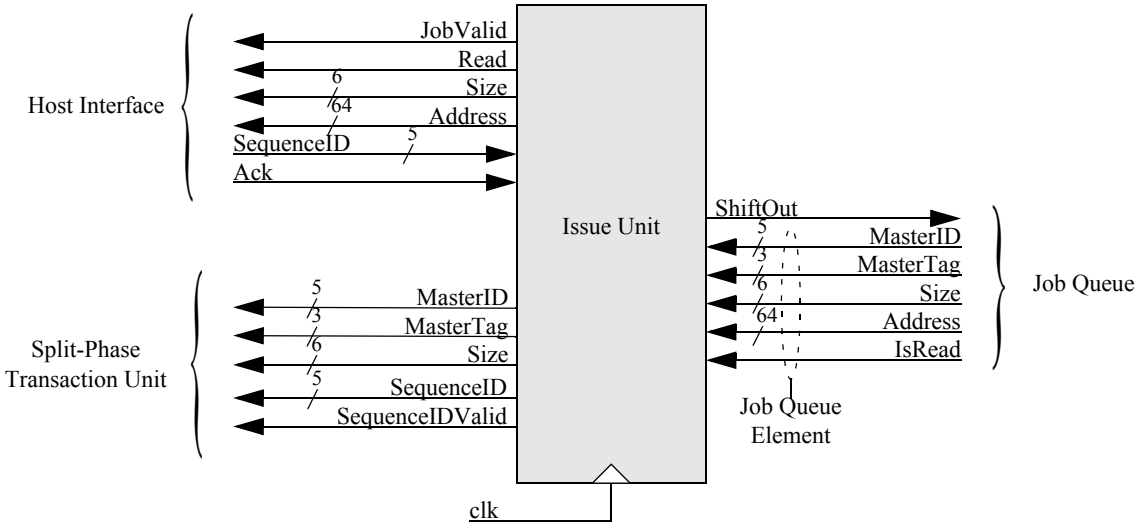


Figure 83: The Issue Unit

It combines the Master Tag, the Size and the Master ID fields of the Job Queue Element with the Sequence ID coming from the Host Interface and forwards it to the Split-Phase Transaction Unit (described later in the Response Path section).

The complete interface signalling to the Host Interface then looks like this:

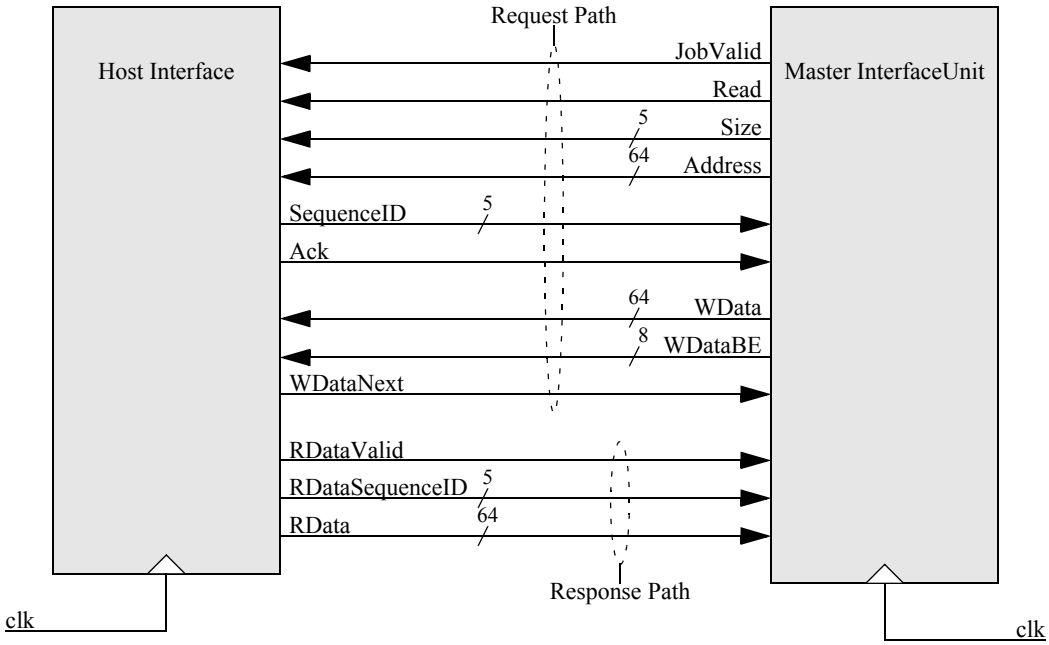


Figure 84: Signalling to the Host Interface [32]

Again, for timing diagrams please refer [32].

The Response Path

The Response Path mainly consists of the Split-Phase Transaction Unit and a multiplexer structure to connect the Master Sockets (Figure 85):

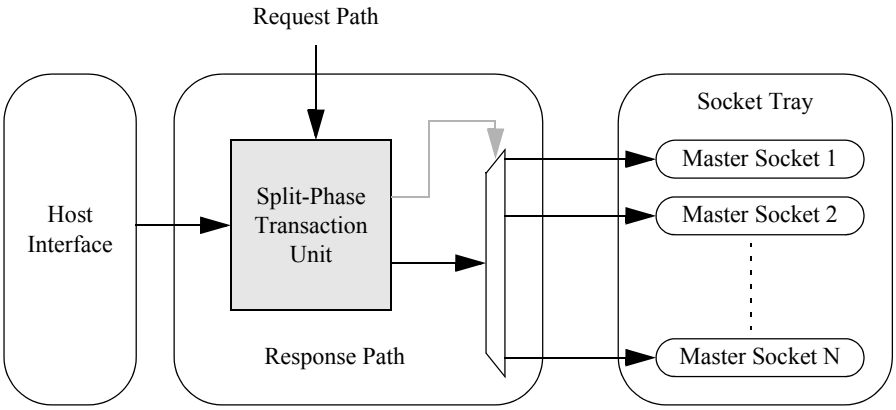


Figure 85: The Response Path [32]

The Split-Phase Transaction Unit contains all necessary information (Table 13) to assign an incoming split response to the correct Master Socket. This Unit is able to manage interrupted split responses as long as the byte/word order within each spit responses is kept, which is the case for all common peripheral bus protocols. The Size field, initiated by the transfer word count is decreased for each word transmitted.

Field	Bitwidth	Description
Master ID	5	Number of master to which a read data value has to be delivered
Master Tag	3	This value tags each data value forwarded to a Master Module
Size	6	The amount of pending read data values
Sequence ID	5	Split-phase transaction tag. Allows the assignment of incoming read data to a Master Module
Valid	1	Indicates whether entry contains valid data

Table 13: A Split-Phase Transaction entry [32]

Only if the Size equals zero the Valid bit is deasserted, which marks the entry as invalid. A new entry is generated if the Issue Unit (from the Response Path) signals a new request. The amount of entries depends on the amount of by the peripheral bus supported pending split-phase transactions. There is no need for a content addressable memory (CAM) as the array may be ordered by the SequenceID assigned by the peripheral bus. With an incoming split response the Split-Phase Transaction Unit just needs to read the MasterID and MasterTag and decrease the Size until it reaches zero. It must be considered an error if a split response arrives with an invalid SequenceID. The MasterID con-

trols the multiplexer structure so that the incoming data can be forwarded together with the MasterTag to the appropriate Master Socket in the Socket Tray.

6 Conclusion and Outlook

During the course of this work the greatest experience with the strongest impact on my personality was to sense the effect or factor of education and staff leadership. The students has been engaged, open-minded and thirsty for knowledge. And together with the outstanding technical requisites this combination exponentiates the efficiency in order to cope with the complexity of a system designs like ATOLL/EXTOLL. It was challenging to explore the very different disciplines from software engineering to ASIC design, from parallel computing paradigms to system area network architectures and from system exploration and implementation to EDA methodology and design flow development; many thanks for this opportunity.

The main results are:

- a next-generation credit based flow controlled XBAR architecture with a deadlock free tori based routing and a virtual tree based hard coded barrier.
  - as transaction accurate ESL model in C++ integrated in a network simulator (called SWORDFISH)
  - as block-level simulated synthesizable RTL code in Verilog including synthesis scripts for Cadence's RTL Compiler
  - as timing clean gate-level netlist synthesized with a IBM 0.13u CMOS standard cell library and IBM RAM hardmacros
- a pipelined 64-bit network processing unit with a special instruction set
  - as instruction set architecture written in LISA
  - as ESL model in System C integrated as virtual hardware prototype in a real system with Linux kernel driver and parallel programming API
  - as timing clean gate-level netlist synthesized with a UMC 0.13u CMOS standard cell library
- a SOC interconnect architecture
  - as ESL model in System C integrated as virtual hardware prototype in a real system with Linux kernel driver and parallel programming API

The results of this work are part of the system design EXTOLL which will be completed by the results of the work of Holger Fröning, David Slogsnat and Mondrian Nüssle. EXTOLL then is a system area network architecture implemented in a submicron ASIC technology on a hyper transport based HTX-board with software support including Linux kernel driver, network daemon, low-level message passing API and MPI2.0 support. As first steps hardware would be implemented using FPGA prototype HTX boards

connected via Hyper Transport to the host system.

A second way of implementing the EXTOLL system area network architecture is just evolving due to the great opportunity of beeing on the radar of AMD R&D research. The Computer Architecture Group just got access to the cache coherent Hyper Transport protocol specification and the corresponding soft IP including test benches. This enables the Computer Architecture Group to design for a AMD processor core. EXTOLL would then be the communication unit of AMD's high-performance server CPUs implementing a revolutionary communication architecture.

The contract in order to continue the SEED project (chapter 4.1.3 "Realization - SEED Project" on page 58) for another 3 years has been signed end of 2004. My former diploma student David Slogsnat, who is since 2003 research associate, has taken over my duties in this project. The upcoming Cadence Technology Day will take place at the University of Mannheim bringing together more than six universities. This is unique in Europe and shows Cadence's confidence in the SEED "idea".

7 References

[1] Daniel D. Gajski, et.al., "Specification and Design of Embedded Systems", PTR Prectice Hall, Englewood Cliffs, New Jersey, USA, 1994

[2] Sellwood, J.A., "The art of N-body building", Annual Review of Astronomy and Astrophysics, 25, p.151-186,1987

[3] Fabrizio Petrini, Wu-chun Feng, Adolfy Hoisie, Salvador Coll, Eitan Frachtenberg, "The Quadrics Network (QsNet): High-Performance Clustering Technology", The Ninth Symposium on High Performance Interconnects (HOTI '01), p. 125ff, 2001

[4] Lars Rzymianowicz, Patrick Schulz, Ulrich Brüning, "Designing the ATOLL ASIC with the EUROPRACTICE EDA tools packages", DATE Year 2001 Design Contest Category: Conceptual Designs, 3rd rank, DATE 2002: Design, Automation and Test in Europe

[5] Patrick R. Schulz, Ulrich Brüning, Gunter Strube. SEED2002: Support of Educational course for Electronic Design. IEEE International Conference on Microelectronic Systems Education (MSE), June 1-2, 2003, Anaheim CA, USA

[6] W.-E. Matzke, G. Strube, H. Schmidt-Habich, L. Drenan. VCAD - A Virtual Enterprise Collaboration Model Impacting the Semiconductor Industry. IASTED International Conference on Knowledge Sharing and Collaborative Engineering KSCE 2004

[7] Integrated project proposal for IST Call 4 (FP6-2004-IST-4), "Stimulating of SOC-implementation through academia (STIM-SoC)", March 2005

[8] Patrick R. Haspel, "Didaktische Reflexion einer experimentellen Lehrveranstaltung in Modul III „Vorlesung und Übung - SemiCustom DesignFlow (SCDF)“", submitted to and accepted by the Hochschuldidaktikzentrum der Universitäten des Landes Baden-Württemberg, April 2005

[9] Bernhard Christmann, "Berufliche Handlungskompetenz von Ingenieurinnen und Ingenieuren - Eine Herausforderung für die Studienreform.", H. Orth, M. Fickenscher (Hg.): Schlüsselqualifikationen praktisch, Neuwied 2001

[10] "Accelerated Hardware/software Co-verification", Cadence Design Systems, White Paper, 2005

[11] Jörg Kluge, Ulrich Brüning, Markus Fischer, Lars Rzymianowicz, Patrick Schulz and Mathias Waack, "The ATOLL approach for a fast and reliable System Area Network", Third Intl. Workshop on Advanced Parallel Processing Technologies (APPT'99) conference, October 19-21 1999, in Changsha, P.R. China

[12] Lars Rzymianowicz, Ulrich Brüning, Jörg Kluge, Patrick Schulz and Mathias Waack, "ATOLL: A Network on a Chip", Cluster Computing Technical Session (CC-TEA) of the PDPTA'99 conference, June 28 - July 1 1999, in Las Vegas, NV

[13] Matthias Scheerer, "Definition and Implementation of a Hardware Abstraction Layer (HAL) for an ASIC-Prototyping Station using a 64Bit/66MHz PCI interfaced FPGA", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2002

[14] Thomas Schlichter, "Exploration of Hard- and Software Requirements for one-sided, zero copy user level Communication and its Implementation", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2003

[15] Thomas Schlichter, "Development of a Boundary Scan Pattern Generation Language", Project work, Computer Architecture Group, University of Mannheim, 2003

[16] Matthias Harter, "Quality Analysis of Back-end Tools in a Cell-based Design Flow of a High-performance Multi-million Gate ASIC", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2002

[17] Frank Ueltzhöffer, "Design, Verification and Physical Implementation of a High-Performance Low-Latency Multi-Level Network Router", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005

[18] Richard Sohnius, "Creating an Executable Specification using SystemC of a High-Performace Low-Latency Muli-Level Network Router", Diploma Thesis, Computer Architecture Group, 2005

[19] Simon Young, "Post-Layout Analysis Is Critical For High-Speed Nanometer Designs", EDAVision Magazine, March 2002

[20] Ulrich Brüning, Holger Fröning, Patrick R. Schulz, Lars Rzymianowicz, "ATOLL: Performance and Cost Optimization of a SAN Interconnect", IASTED Conference: Parallel and Distributed Computing and Systems (PDCS), Nov. 4 - 6, 2002, Cambridge, USA

[21] Holger Fröning, Mondrian Nüssle, David Slogsnat, Patrick R. Haspel, Ulrich Brüning, "Performance Evaluation of the ATOLL Interconnect", IASTED Conference: Parallel and Distributed Computing and Networks (PDCN), Feb. 15 - 17, 2005, Innsbruck, Austria

[22] David Slogsnat, Patrick R. Haspel, Holger Froening and Ulrich Bruening, "The ATOLL System Area Network (SAN)", IEEE Task Force Cluster Computing Newsletter, September 2003

[23] Markus Fischer, Ulrich Brüning, Jörg Kluge, Lars Rzymianowicz, Patrick Schulz and Mathias Waack, "ATOLL, a new switched, high speed Interconnect in Comparison to Myrinet and SCI", IPDPS 2000, PC NOW Workshop, May 1-5 2000, Cancun Mexico

[24] Dennis Sylvester, Kurt Keutzer, "Rethinking Deep-Submicron Circuit Design", IEEE Computer, Vol. 32 No. 11; November 1999, pp. 25-33

[25] Lavi Lev, Ping Chao, "Down To The Wire - Requirements For Nanometer Design Implementation", White Paper, Cadence Design Systems, 2002

[26] Brad Marshall, Jürgen Köhl, Tilman Wagner, "A New ASIC Timing Signoff Methodology", IBM MicroNews, Second Quarter 2002

[27] Jon Beecroft, David Addison, David Hewson, Moray McLaren, Fabrizio Petrini, Duncan Roweth, "Quadrics QsNet^{II}: Pushing the Limit of the Design of High-Performance Networks for Supercomputers", IEEE Micro, to appear 2005

[28] Jon Beecroft, David Addison, Fabrizio Petrini, Moray McLaren, "Quadrics QsNet^{II}: An Interconnect for Supercomputing Applications", Quadrics White Paper, 2003

[29] David Addison, Jon Beecroft, David Hewson, Moray McLaren, Fabrizio Petrini, "Quadrics QsNet^{II}: A network for Supercomputing Applications", Presentation at IEEE HotChips'03, Stanford USA, August 2003

[30] Fabrizio Petrini, Wu-chun Feng, Adolfy Hoisie, Salvador Coll, Eitan Frachtenberg, "The Quadrics Network: High-Perfomance Clustering Technology", IEEE Micro, Volume 22, Issue 1, Jan.-Feb. 2002

[31] Ulrich Brüning, Wolfgang K. Giloi, "Future Building Blocks for Parallel Architectures", Keynote talk at of the 2004 International Conference on Parallel Processing (ICPP.04), Montreal, CA, 2004

[32] Timo Sponer, "Development, Verification and Integration of a Processing Unit in the Communication Function of a SAN Device in SystemC", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005

[33] Ho Won Kim, Hyun Suk Lee, Sunguu Lee, Jong Kim, "Adaptive Virtual Cut-Through as a Viable Routing Method", Journal of Parallel and Distributed Computing, Vol. 52, Academic Press, 1998

[34] SHRIMP project, http://www.cs.princeton.edu/shrimp/html/communication.html, Princeton University

[35] J. Duato, S. Yalmanchili, "Interconnection Networks - An Engineering Approach", IEEE Computer Society Press, Los Alamos, California, 1997

[36] S. Peng, "Design of Interconnection Networks for MPP of Next Generation", Faculty of Computer and Information Sciences, Hosei University, Tokyo, Japan

[37] Mondrian Nüssle, "Design and Implementation of a distributed management system for the ATOLL high-performance network", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005

[38] Ulrich Brüning, "Network Interfaces and their Features", Talk at the International Supercomputer Conference (ISC2005), Heidelberg, Germany, 2005

[39] Tilo Wettig, "QCDOC: a massively parallel supercomputer", Talk at the GSI (Gesellschaft für Schwerionenforschung), May 19, 2004, Darmstadt, Germany

[40] Holger Bellm, "Entwurf und Implementierung eines parametrisierbaren Arbitergenerators in Verilog", Project Work, Computer Architecture Group, University of Mannheim, 2002

[41] Alexandra Bernardt, "Design, Implementation and Synthesis of a Parameterizable Soft IP Cell for a high performance crossbar", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2003

[42] Holger Fröning, "Methods and Mechanisms for efficient Multithreading Device Structures", Inaugural Dissertation, Computer Architecture Group, University of Mannheim, 2006 (to be submitted)

[43] Mondrian Nüssle, Inaugural Dissertation, Computer Architecture Group, University of Mannheim, 2006 (to be submitted, currently internal technical report)

[44] David Slogsnat, Inaugural Dissertation, Computer Architecture Group, University of Mannheim, 2006 (to be submitted, currently internal technical report)

[45] Ingo Feldner, "High Level Executable Specification Development of a high performance SAN chip", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2005

[46] OCP International Partnership, "Open Core Protocol Specification", Release 2.0, 2003

[47] Holger Bellm, "Architectural Design and Prototype Implementation of an Embedded Network Processor Core using Language for Instruction Set Architectures (LISA)", Diploma Thesis, Computer Architecture Group, University of Mannheim, 2003

[48] Chelsio web page, http://www.chelsio.com/about/index.php, Chelsio Communications, 2005

[49] Sven Stork, "Anpassungen für Virtual Hardware Prototyping des LINUX-Kerneltreibers des EXTOLL NICs", Project work, Computer Architecture Group, University of Mannheim, 2003

[50] J. Duato, "A necessary and sufficient condition for deadlock-free routing in cutthrough and store-and-forward networks", IEEE Transactions on Parallel and Distributed Systems, vol.7, no.8, pp.841-854, August 1996

[51] William J. Dally, Charles L. Seitz, "Deadlock Free Message Routing in Multiprozessor Interconnection Networks", IEEE Transactions on Computers, vol. C-36, no.5, pp.547-553, 1987

[52] R. Cypher, L. Gravano, "Requirements for deadlock-free, adaptive packet routing", 11th Symposium on Principles of Distributed Computing, pp. 25-34, August 1992

[53] William J. Dally, "Virtual Channel Flow Control", IEEE Transactions on Parallel and Distributed Systems, vol.3, no.2, pp.194-205, March 1992

[54] Andrew S. Tanenbaum, “Computernetzwerke”, 3rd revised edition, Prentice Hall, München, Germany, 1998

[55] Ulrich Brüning, "Computer Architecture II", lecture notes, Computer Architecture Group, University of Mannheim, 2004

8 Acknowledgements

The years at Prof. Brüning's group have been an extremely exciting part of my life.

I would like to thank all contributors starting with Prof. Brüning. We have been worked together for more than 9 years. During this time he mentored me not only in science but also in social skills and something like just being human. I thank my collegues Holger Fröning, David Slogsnat and Mondrian Nüssle. They are a great team and I really appreciate the time in that we have worked together. Boris Strohmeier was not only a colleague but is also a good friend. We met first at the very first day at university in the first semester both being late to our first course. Thanks, Boris.

Beyond university my best friend for a very long time Thomas Kinfe decided to share a flat with me. We had just a tremendous time. Thank you, Thomas!

But the best year at university was the last one. I met my wife, Stefanie Haspel. She changed my life. Her support gave me the necessary power to complete my work. We are now facing the exciting challenge of parenting our daughter Nemira.

I'm looking forward to experience our future. Thank you!