RECONFIGURABLE OPTICAL GIGABIT NETWORK ADAPTER

T. MEIER[†], D. FEY[‡], W. ERHARD[†]

[†]Friedrich-Schiller-University Jena, Department of Computer Science,
Ernst-Abbe-Platz 1-4, 07743 Jena, Germany
[‡]University-GH Siegen, Institute of Computer Structures,
Hölderlinstr. 3, 57068 Siegen, Germany

Abstract: *We have developed a reconfigurable optical gigabit network adapter which offers a physical bandwidth of more than 1 Gbit per second in each direction. The distance between two adapters can be up to one kilometer. This combination makes our adapter suitable for use in distributed computing environments as well as in typical LAN environments. The media access protocol is implemented in a programmable FPGA processor. Due to the hardware reconfigurability of this processor we can utilize different media access protocols depending on application and performance requirements. This paper presents the hardware architecture of the network adapter and the implementation of media access protocols into the FPGA processor. Furthermore we discuss the protocols we will use with the adapter with regard to their achievable bandwidth, latency and network utilization.*

keywords: network adapter, gigabit networking, network protocol , cluster computing, optical link

1. Introduction

Besides the permanent increase in efficiency of modern microprocessors, the improvements in network technology have a great effect on the performance of applications in local area networks and distributed computing applications (Cluster Computing). The development of network standards like Gigabit-Ethernet, Asynchronous Transfer Mode (ATM) [1] and Fiber Channel [2] and special networks like Myrinet [3] or Scalable Coherent Interface (SCI) [4] reflects this trend.

The rapid improvements in the availability of high performance networks have enabled the evolution of cluster computers, which in essence means linking together a collection of workstations or PC's to jointly solve a computational problem. The bandwidth and latency of the interconnection network must be balanced compared to the performance of the workstation or PC to prevent the network from becoming the bottleneck of the computer cluster.

Our primary objective was the development of a high-speed network adapter, which is well suited for high-performance distributed computing (cluster computing) as well as for typical LAN applications. Thus, users can boost their processing power using off-the-shelf computers, connected by a high-speed network, at a relatively low cost.

The network adapter provides transmission bandwidth of more than 1 Gbit/s in each direction. The distance between two adapters can be up to one kilometer which is a requirement for the use of the adapter in LAN environments. The high transmission performance in combination with long distances is achieved by using newest optical transmitter and receiver devices, which are connected by optical fibers. Optical interconnection systems offer a higher bandwidth than their electrical counterpart, especially over long distances. Electrical interconnection systems are much more affected by attenuation, crosstalk and ringing than optical interconnection systems [5].

Another important feature of our network adapter is its reconfigurability. Typical LAN applications or cluster computing applications may be distinct with regard to their demand on bandwidth, latency or interarrival time. The adapter can be reconfigured to satisfy these individual requirements. This is achieved by implementing the appropriate hardware protocol[1)] into a programmable FPGA processor, which can be reconfigured anytime.

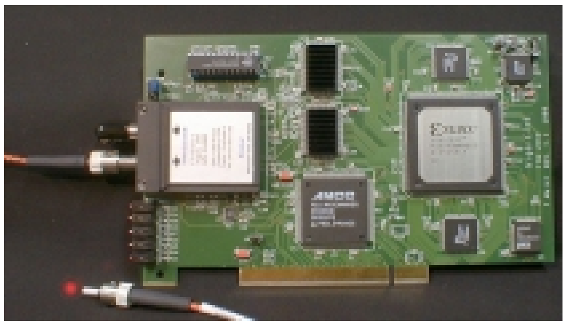
Figure 1: Prototype of the reconfigurable optical network adapter

Correspondence: T. Meier, mtm@uni-jena.de. Other authors: D. Fey: fey@rs.uni-siegen.de, W. Erhard: werner.erhard@uni-jena.de

[1)] During the text, the term network protocol, hardware protocol and media access protocol are used interchangeably and indicate the algorithms implemented into the FPGA processor for media access control.

Presently we have the first prototype of the network adapter in practical use (Figure 1). The first hardware protocol we have implemented into the FPGA processor is a simple token-protocol. We evaluate various hardware protocols concerning to their achievable bandwidth and latency. Some of them will be implemented to use them with our network adapter. From the results of this exploration we want to get a deeper knowledge about the relationship between network hardware, network protocol, network topology and applications.

The paper is structured as follows. In section 2 we describe the hardware architecture of our network adapter. We explain the main components and their functionality. Section 3 focuses on the FPGA processor and the implementation of hardware protocols in more detail. In section 4 we introduce the hardware protocols we want to use with the adapter. We will explain, how they differ with regard to their achievable bandwidth, latency and network utilization. Finally we conclude our results in section 5.

2. Hardware Architecture

In order to effectively work in a distributed environment, a fast network is of utmost importance. The network interface card[2)] (NIC) must provide high bandwidth and a low latency to meet the needs of today's distributed applications and to prevent the network from becoming the bottleneck along the data path between the sending and receiving application.

2.1. Data Transfer Functionality

This section describes the adapters data transfer functionality and its hardware realization. Data transfers between the network transmission media and the host memory can be grouped into two classes.

a) Transferring data between the host memory and the network adapter.
b) Transferring data between the network adapter and the transmission media.

The boundary between these two activities represents the FPGA processor, which controls most of the components on the network interface. This situation is demonstrated in Figure 2.

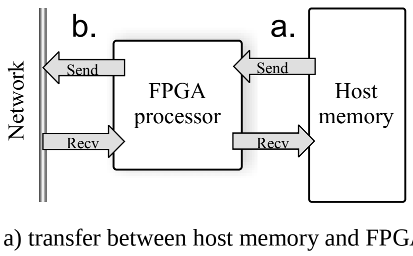

Figure 2: a) transfer between host memory and FPGA processor, b) transfer between FPGA processor and network media

2) In this paper, Network Interface Card (NIC) and network adapter are synonyms.

a) Data transfer between NIC and host memory

We use the PCI bus interface for data transfers between the NIC and the host memory. This standard interface is widely used in today's PC's and workstations and offers a transmission bandwidth of at least 1 Gbit/sec. Two main methods are possible to move the data in and out of host memory and to transfer the data over the PCI bus.

- direct memory access (DMA)
- memory mapped I/O

Some work has been done to find out, which one of the methods are better suited for high speed networking interfaces [6], [7], [8]. The right choice depends highly on the utilized software interface, i.e. the upper network layers, and the kind of data, i.e. the length of individual data frames, and the machines hardware architecture.

Our adapter is designed to support both mechanisms. This gives us the flexibility to change the transfer method if required. In the case of memory mapped I/O the adapter is mapped into the virtual memory of the host. The host CPU performs PCI transfers if load or store operations on this region occur. When using the DMA method the host CPU is only needed during the initialization phase of the DMA transfer. After the source address, the transfer length and the target address has been written to the DMA engine, which is located on the network adapter, the actual transfer can begin and the host CPU is not longer needed.
Our current network adapter design uses the 32 bit /33 MHz PCI bus version whereby we can attain a theoretical bandwidth of 1 Gbit/sec for a send or receive operation.

b) Data transfer between NIC and host memory

The network adapter achieves a throughput of 1 Gbit/sec for send and receive transfers between the FPGA processor and the network media. This high bandwidth is possible by using optical high-speed devices.

A fast laser diode module, which serves as the optical sender, transforms the electrical data signal into a optical light signal. We use a multimode laser at a wavelength of 850 nm which is equipped with a standard ST-connector to connect the optical fiber. Due to the optical power of the laser and the low attenuation of the optical fiber the distance between the optical sender and receiver can be as far as 1 kilometer. The distance can be increased by replacing the multimode laser module with a singelemode laser module (1300 nm wavelength). The laser modules have the same package and pin-out. Hence its easy to exchange them.

A fast photo diode, which must be sensitive to the wavelength of the sending laser diode, serves as the optical receiver. It receives the light signals from the optical fiber and performs its transformation into a electrical signal.

Optical sender and receiver are connected by a single fiber which means, that the connection is a serial point-to-point link. Our own experience have led us to the conclusion that nowadays this technique is to prefer over the parallel connection scheme in respect of PC network adapters [5].

Because of the serial character of the optical connection and the parallel data path on the remaining part of the adapter, a data conversion scheme between the optical sender/receiver and the FPGA processor must be applied. On the sender side of the source adapter we use a serializer to transform the 16 bit data words coming from the FPGA processor into a 16 times faster serial bitstream. The serializer also inserts some information for clock recovery and error detection into the bitstream. This information is intended for the deserializer which performs the reverse action on the receiving side of the target adapter. The deserializer, which performs the serial-to-parallel conversion, supplies 16 bit data words and a synchronization clock to the FPGA processor.

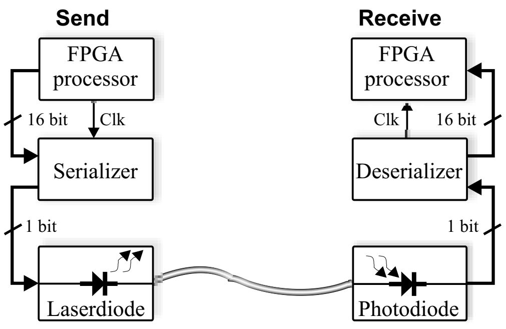

Figure 3: Data conversion on the sending and receiving side

Figure 3 depicts the conversion along the data path from the sending processor to the receiving processor. Besides the data and clock signals, there are some other control signal transferred between the serializer/deserializer and the FPFA processor for error detection and link control.

2.2. Hardware structure of the network adapter

This section describes the overall hardware structure of the network adapter. The block diagram of the network adapter is depicted in Figure 4.

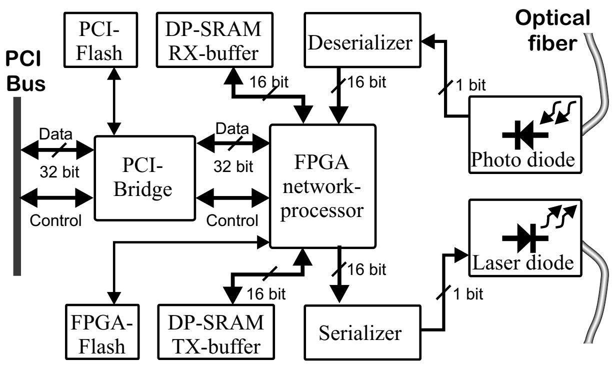

Figure 4: Block diagram of the network adapter

Besides the components for the transfer functions, which where subject to the last subsection, the network adapter utilizes the following components.

- *Dual-Port SRAM*

The SRAM components serve as buffers for transmission data (TX-buffer) and for data received from the network (RX-buffer). The high throughput of these buffers is achieved by using dual port memories, which can be written and read at the same time by the FPGA processor. The data rate of write and read operations between the SRAM and the FPGA processor is 1 Gbit/sec.

TX-buffer: Data packets arriving from the PCI Bus will be stored in the TX-buffer until the complete packet is transmitted over the PCI bus. Once the network processor has the right to send data onto the network, the data packet will be read out of the SRAM and transmitted to the optical sender module. This “store-and-forward” scheme increases latency but avoids fragmentation of data packets on the network.

RX-buffer: The SRAM module on the receiver side, the RX-buffer, can store data packets arriving from the network whose destination is the local host memory. This is necessary for synchronization purpose between the network-to-FPGA transfer, which offers a constant bandwidth of 1 Gbit/sec, and the FPGA-to-host-memory transfer, whose bandwidth averages less than 1 Gbit/sec.

The current version of our network adapter utilizes 64 Kbytes SRAM memory. The determination of the required SRAM capacity for the network adapter is subject to our current research and will affect the coming hardware development phases.

- *PCI Flash memory*

This non-volatile memory hold PCI configuration data. During the startup phase of the host-PC, the operating systems reads the PCI-configuration of every PCI device to reserve resources like interrupt lines and address regions. Furthermore, we can store device specific PCI information like device class, vendor ID and device ID in this flash memory. The flash memory can be written by software routines to change the PCI settings of the adapter.

- *FPGA Flash memory*

The hardware protocol is implemented into the FPGA processor which is subject to section 3. The on-board configuration flash gives the adapter the ability to automatically load a default power-up protocol into the FPGA processor. This way the adapter can be used like a standard network adapter without reconfiguration facility. The user doesn’t need to load a configuration into the FPGA as part of the network driver initialization.

- *PCI-Bridge*

The PCI bridge controls the data transfers over the PCI bus between the host memory and the FPGA processor as described in the last subsection. The PCI bridge has a 32 bit wide add-on data interface to the FPGA processor. Control signals give the FPGA processor access to PCI information to check the current transfer status. When a new data packet has been transferred to the PCI bridge by the FPGA processor, the bridge informs the operating system by activating the interrupt line or setting a control

register, whichever method is used. Then a PCI transfer is initiated by the network driver software, which belongs to the upper network layers and is not subject of this paper.
The other direction, from host memory to FPGA processor, works almost the same way. After the driver initiated a send transfer, the PCI bridge informs the FPGA processor of a new transfer. Then the processor reads the incoming data and stores them inside the TX-buffer. As soon as the complete packed has been arrived, the PCI bridge activates a interrupt line or sets a register to inform the operating system about a successful transfer.

3. Hardware protocol implementation

The facility to use different hardware protocols with the network adapter is one of its main features. Depending on changing requirements on latency, bandwidth and network utilization we can apply different hardware protocols with the network adapter by a reconfiguration of the FPGA processor, i.e. loading the accordingly configuration data into the FPGA processor. This process does not affect any other network protocol layers. This means, that the same driver software and upper protocol layer software, like TCP/IP protocol layers, can be used. Figure 5 depicts the ISO/OSI network layer model with the reconfigurable hardware as part of the data link layer. The gray shaded arrows represent the data flow through the protocol layers.

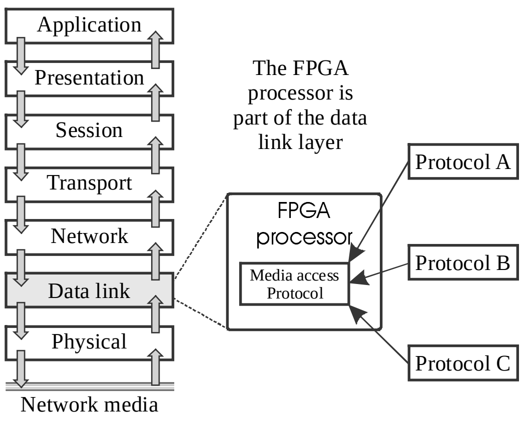

Figure 5: ISO/OSI network layer model: the media access protocol as part of the data link layer

In contrast to a software programmable processor, where the processor executes a previously loaded set of instructions, our network adapter utilizes a hardware programmable FPGA processor. Hence, the functional description of the hardware protocol needs to be mapped to the hardware resources of the FPGA processor. The advantage over a software programmable scheme is a faster execution because each function is represented by hardware, i.e. combinatorial logic and flip-flops. Hence, no time consuming load or store operations of data or instructions are necessary, which guarantees high speed along the data path inside the FPGA processor. The drawback is a higher implementation effort. The following steps are necessary to implement a hardware protocol into the FPAG processor.

1) Programming the hardware protocol
2) Synthesis of the protocol description
3) Verification of the hardware protocol
4) Downloading the protocol into the FPGA processor

3.1. Programming the hardware protocol

The hardware protocol can be designed by using a programming language or a schematic editor. We use VHDL ((VHSIC (Very High Speed Integrated Circuits) Hardware Description Language), which is a high level programming language, to describe the protocol functionality. Figure 6 depicts a VHDL code fragment, which performs the storage of a 16 bit data word in a register depending on the value of the *TRANS_START* signal.

```
Signal IN_DATA : std_logic_vector(15:0)
Signal OUT_DATA: std_logic_vector(15:0)
                      .
                      .
                      .
process()
Begin
  IF(CLK'event AND CLK = 1) THEN
     if (TRANS_START = 1) then
        OUT_DATA <= IN_DATA;
     end if;
  END IF;
End process;
```

Figure 6: VHDL code fragment used to describe a hardware protocol

The VHDL description of the hardware protocol program has been partitioned into a number of modules, which leads to a well structured program code. The modules interact with each other by using well defined interfaces, i.e. signals. Due to the modularization we can reuse some of the modules for the description of different hardware protocols which makes the design process faster and less error-prone. It also yields to easier modifications of the protocol description by simply replacing some modules or code fragments.
Figure 7 depicts a block diagram of a modularized hardware protocol, which has been implemented into the FPGA processor. It shows the following coarse modules partition.

- *PCI bus module*: This module controls the data transfer between the PCI-bridge and the FPGA processor.
- *Control unit*: This module is the main part of the hardware protocol description. It controls all the other modules depending on the applied media access protocol.
- *Transmitter module*: Data packets arriving from the PCI bus are stored to the TX-buffer by this module. As soon as this module gets the right to send data to the network it send data packets stored in the TX-buffer or packet previously received by the receiver module (forward packets) to the optical sender module.

- *Receiver module*: Data packets arriving from the network are handled by this module. It organizes the transfer of data packets to the PCI module and transfers forward packets to the transmitter module.

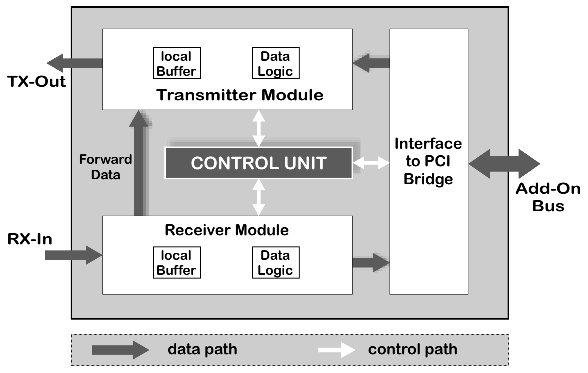

Figure 7: Block diagram of a modularized hardware protocol description

Supplementary link layer functionality like error detection or data encryption can be implemented in additional modules.

3.2. Synthesis of the protocol program

After the hardware protocol has been described using VHDL, the program code needs to be synthesized. This step performs the mapping between the functional description, specified by the VHDL program, and the architecture of the FPGA processor, i.e. their hardware resources. Figure 8 depicts the resulting hardware representation of the synthesized VHDL code fragment presented in Figure 6.

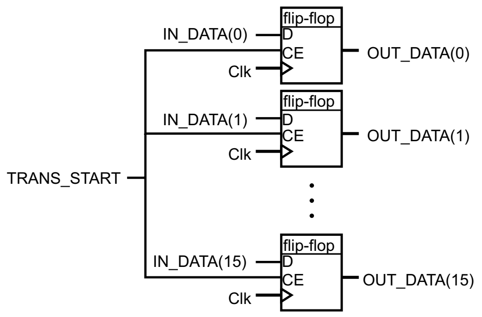

Figure 8: Hardware representation of the synthesized VHDL code

The resulting hardware representation can be considered as a netlist of FPGA components, like flip-flops or combinatorial gates, which are interconnected by the signal lines. During the place-and-route phase the individual components of the netlist will be placed to the appropriate locations of the FPGA processor and the interconnection network between these components is routed. The resulting information, a image of the configured FPGA contents, is finally written to a FPGA specific configuration file, which can be downloaded to the FPGA processor.

3.3. Verification of the hardware protocol

The Verification process of a implemented hardware protocol performs two steps. First, the synthesized hardware representation of the protocol is simulated inside a VHDL testbench. During this simulation step each signal transition inside the FPGA processor is simulated and the resulting signal pattern is compared to their expected values. Due to the great number of signal transitions this simulation step takes some time to execute, even on high-performance workstations. Hence, a complete network consisting of several network adapters can not be simulated efficiently this way. Here starts the second simulation step, which uses a discrete event driven simulation library [9] to simulate the hardware protocol in the context of a complete network. A representation of the FPGA processors functionality has been programmed for this purpose using a C++ class library. Various conditions like packet inter-arrival time, fiber length and number of network adapters can be supplied as parameters at run time. As a result we can retrieve detailed information on latency, bandwidth, network utilization, starvation of individual network nodes etc.

3.4. Downloading the FPGA processor

The downloading of the FPGA processor can be achieved either by using the flash memory or as part of the device driver initialization step. The first case has been already described in section 2.2. For the latter case a simple downloading routine, executed during the driver initialization, reads the appropriate configuration file from disk and writes this data into the FPGA processor. Once the configuration data is stored inside the FPGA processor, the network adapter is ready-to-operate.

4. Hardware Protocols

Three different protocols have been chosen for a implementation on the network adapter, which differ with respect to their achievable bandwidth, latency and network utilization. In the following section we give a brief description of the chosen protocols. A detailed explanation goes beyond the scope of this paper.
Due to the link character of optical connections we use a ring topology to build up a network, in which each station is directly connected to two neighbors: its immediate predecessor and its immediate successor on the ring. A medium access protocol for such a network should establish a collection of rules for the following two elements of the stations behavior:

- Transmission rights, i.e., when a station is allowed to transmit data onto the medium
- Cleaning rules, i.e., how data packets are removed from the ring

4.1. Token Protocol

The token ring protocol [10] is the most widely used protocol for ring networks. A token ring network uses a special frame, the token, that rotates around the ring when no station is actively sending data frames. When a station wants to transmit data on the ring, it must capture this token frame. Since there exists only one token frame in the network, the owner of the token frame is the only station that can transmit data on the ring. Depending on the applied token release method the token frame is released after the station has send a data frame or after a assigned period has elapsed. The next station captures the token and can send data on the ring. The advantage of this synchronization scheme is its fairness since every station has access to the network media during a predictable time interval. Thus, we can make predictions on the upper bound of the latency times. The drawbacks of this protocol are its weak network utilization due to the exclusive sending rights and high access delays even under light load.

4.2. Buffered Insertion Ring Protocol

The insertion ring protocol tries to eliminate the weak spots of the token protocol, high access times and small network utilization, by allowing each station in a ring to transmit their data frames spontaneously. This token-less scheme makes use of a special memory, called insertion buffer, to prevent conflicts between the transmission of forward frames and new frames. The basic operation of the insertion ring protocol can be described as follows. Forward frames received by the station are loaded into the insertion buffer. If the stations output port is free, the forward frame can be transmitted on the ring immediately (cut through operation). If the station is transmitting a new frame, the receiving forward frame is feed into the insertion buffer until the output port becomes free which virtually enlarges the ring size by the size of used portion of the insertion buffer (store and forward operation). With this scheme it seems to be clear that the minimum size of the insertion buffer must be at least equal to the maximum frame size.

Since our network adapter has no explicit insertion buffer, we use the TX-buffer (section 2.2) for this purpose. This buffer has enough capacity to hold both, new data frames arriving from the PCI-bus and forward frames waiting for their transmission on the ring. Thus, forward frames arriving at times the adapter sends a new data frame to the ring, the forward frame will be stored to the TX-buffer and waits there until the transmission of the new data frame has been finished (store and forward operation). Switching between these two frame sources is accomplished by changing one address bit of the TX-buffer. If a forward frame arrives while the output port is free, the frame will be forwarded to the output port immediately without writing the data frame to the TX-buffer first (cut through operation).

The only drawback of this method is the fact, that no data frames arriving from the PCI bus can be written to the TX-buffer, while the buffer input is in use by forward frames. We will examine this situation by upcoming simulations of this network protocol as described in section 3.3.

Since a continuous flow of forward frames prevents a station from inserting new frames, the insertion ring protocol can't be considered to be fair. Hence, the possibility of long latency times and the starvation potential are the most painful disadvantages of the pure insertion ring protocol. The advantages of the insertion ring protocol are its high network utilization and zero access delays under light load. There exist a number of variations and strategies with this protocol which are explained in [11] and [12].

4.3. DPMA Protocol

Although the DPMA (Distributed Pretzel Multiple Access) protocol achieves best performance on a pretzel topology [12], which is obtained from a dual ring by cutting both rings in the same spot and splicing each to the other ring, it can be adapted to a simple ring topology as well. The basic operation of this form of the DPMA protocol can be described as follows:
This protocol uses a weak-token scheme, i.e. a station doesn't have to acquire the token to start a frame transmission. A general rule is that the token holding station can always transmit new data frames on the ring. In contrast to the pure token protocol, the station holds the token for a constant amount of time (Token Holding Time *THT*). To avoid collisions between frames send by the token holding station and frame send by other stations the following two rules must be applied to all other stations:

1. Other stations can transmit a new frame on the ring if the station is not receiving a forward frames. This first condition can be satisfied by applying the insertion ring scheme presented in the last section.
2. Other station must ensure that a transmitted data frame doesn't catch up with the token on its way to destination. In this case the data frame would be destroyed because of the general rule that the token holding station can always send new data on the ring. This condition can be meet by comparing an internal timer value, the Token Rotating Timer (*TRT*), with the value of the Total Token Rotating Time (*TTRT*). The *TRT* counts the time elapsed since the moment the station last acquired the token. The *TTRT* equals the sum of all stations *TRT*'s plus the propagation length of the ring (*L*) and is given by the formula:

   $$TTRT = L + \sum_{i=1}^{n} TRT_i$$

   A station willing to transmit a frame is allowed to do so, provided that the following condition holds:

   $$TRT \geq TTRT - L$$

   The application of this rule ensures that the new data frame, which moves faster on the ring then the token, doesn't reach the token station on their way to the destination and thus no collision with the token holding station can occur.

The DPMA protocol eliminates the most important drawback of the insertion ring protocol, namely its starvation potential, by using a weak-token scheme which provides a fairness even under heavy load and thus an upper bound for the latency. Due to the weak-token scheme, DPMA incurs small medium access delay when the network traffic is low. The multiple access facility of this protocol yields a better network utilization then the pure token protocol does.

Detailed simulations of these protocols are subject to our upcoming work. Dependencies between performance criteria, i.e. latency times, bandwidth and network utilization, and network parameters, i.e. ring length, number of stations and media access protocol, are of most essential interest for our future work.

5. Conclusion

In this paper we presented the design of a reconfigurable optical network adapter which is capable of receiving and transmitting with a data rate of 1 Gbit/sec at the same time. The distance between two neighboring stations in a ring network can be as far as 1 kilometer. This high performance in combination with long distances has been achieved by using optical transmitter and receiver devices.

Due to the application of a programmable network processor into which the network protocol is implemented, we can employ different media access protocols depending on required bandwidth and latency. The desired protocol can be specified in form of a hardware description language program like VHDL, which is synthesized and simulated by appropriate design tools.

We have chosen three different ring network protocols for a implementation with the network adapter. The protocols differ with regard to their expected bandwidth and latency, which we will verify by means of upcoming simulations and practical employment of these protocols.

An early version of the token protocol has been already implemented. We measured an application bandwidth of about 100 Mbit/sec between any two stations on a three station ring network of 133 MHz Pentium PC’s using the TCP/IP protocol. Improvements of the driver software and the use of present-days PC’s will contribute to better results.

Another approach is the implementation of a user-level interface to circumvent the software overhead of the TCP/IP protocol by providing a direct user-level access to the network interface hardware. For this reason we are currently working on a VIA interface (Virtual Interface Architecture) for our reconfigurable optical network adapter [13].

References:

[1] J.-Y. Le Boudec, The Asynchronous Transfer Mode: A Tutorial, *Computer Networks and ISDN Systems*, 24(4), 1992, pp. 279-310

[2] The Fibre Channel Association, *Fibre Channel: Connection to the Future*, Fibre Channel Association, 1994, ISBN: 1-878707-19-1

[3] Nanette J. Boden et al., Myrinet: A Gigabit-per-second Local Area Network, *IEEE Micro,* 15(1), February 1995, pp. 29-36

[4] D. Gustavson, The scalable coherent interface and related standard projects, *IEEE Micro*, 12, February 1992, pp.10-22

[5] T.Meier et al., Optische Verbindungstechniken für schnellen Datentranfer zwischen integrierten Schaltkreisen, *Berichte zur Rechnerarchitektur,* Bd 4(4), Friedrich-Schiller-Universität Jena, 1998

[6] Raikumar Buyya, *High Performance Cluster Computing: Architectures and Systems, Volume 1*, New Jersey, Prentice Hall, 1999

[7] P.Druschel et al.,Network Subsystem design: A Case for an Intergrated Data Path, *IEEE Network* (Special Issue on End-System Support for High-Speed Networks), July 1993, 7(4), pp. 8-17

[8] P.Druschel, L.L.Peterson, B.S.Davie, Experiences with a High-Speed Network Adaptor: A Software Perspective, *Proceedings of the ACM SIG-COMM’94 Symposium*, London, September 1994, pp. 2-13

[9] A. Varga, *OMNeT++ User Manual*, Dept. of Telecommunications, Technical University of Budapest, 1997, http://www.hit.bme.hu/phd/vargaa/omnetpp.htm

[10] IEEE Std. 802.5-1989. Local Area Networks 802.5 Token Ring Access Method, Institute of Electrical and Electronic Engineers, 1989, ISBN 1-55937012-2

[11] Joseph.L.Hammond, Peter J.P. O’Reilly, *Performance Analysis of Local Computer Networks,* Addison-Wesley, 1986, ISBN 0-201-11530-1

[12] Pawel Gburzynski, *Protocol Design for Local and Metropolitan Area networks,* New Jersey, Prentice Hall, 1996, ISBN 0-13-554270-7

[13] Thorsten von Eicken, Werner Vogels, Evolution of the Virtual Interface Architecture, *IEEE Computer*, 11, Nov.1998, pp.61-68