Enabling High Bandwidth Applications by High-Performance Multicast Transfer Protocol Processing

G. Carle, J. Schiller
University of Karlsruhe, Institute of Telematics,
Zirkel 2, 76128 Karlsruhe, Germany
Phone: +49 721 608-[4027,4003], Fax: +49 721 388097
e-mail: [carle,schiller]@telematik.informatik.uni-karlsruhe.de

Abstract

A large range of applications exists with demand for high-performance point-to-point and point-to-multipoint communication. Existing communication subsystems frequently represent a major performance bottleneck. To overcome this bottleneck, a framework for high-performance multicast transfer protocol processing is presented, based on hardware support for multicast error control in transmitters and in dedicated intermediate systems called Group Communication Servers. The design of a protocol processing coprocessor for selective retransmissions by end systems and servers in multicast scenarios is presented. High scalability for a large number of receivers can be ensured by the deployment of a VLSI component for list management of acknowledgement processing. The integration of the VLSI component into a generic coprocessor (the Generic ATM Protocol Processing Unit, GAPPU) is shown. Details of processing delays and implementation costs of the proposed hardware implementation are given, and compared with measurements of a typical software implementation.

Keywords

ATM, group communication, VLSI, coprocessor, error control

1 INTRODUCTION

Emerging applications, mostly, require both high performance as well as support of a wide variety of real-time and non-real-time communication services. For example, audio, video, and message passing of distributed systems may require different services. Networks, (e.g., ATM-based networks) are able to fulfil the basic requirements by providing data rates exceeding a gigabit per second and by supporting different kinds of services. However, current communication subsystems (including higher layer protocols) are not able to deliver the available network performance to the applications.
In the evolution of high speed networking, various multipoint communication services will be of increasing importance. Examples of applications that require point-to-multipoint (Multicast, 1:N) as

well as multipoint-to-multipoint (Multipeer, M:N) communication can be found in the areas of computer-supported cooperative work (CSCW), distributed control, and distributed computing as, e.g., in workstation clusters (Heinrichs, 1993). For a growing number of applications such as multimedia collaboration systems, the provision of a multicast service with a specific quality of service (QoS) in terms of throughput, delay, and reliability is crucial.

If multipoint communication is not supported by the network or by the end-to-end protocols, multiple point-to-point connections must be used for distribution of identical information to the members of a group. The support of multicasting is beneficial in various ways: It saves bandwidth, reduces processing effort for the end systems, reduces the mean delay for the receivers, and simplifies addressing and connection management.

Various issues need to be addressed in order to provide group communication services in high-speed networks (Waters, 1992), (Bubenik, 1992). Intermediate systems need to incorporate a copy function for support of 1:N connections. Communication protocols must be capable of managing multipoint connections, and group management functions need to be provided for administration of members joining and leaving a group. A key problem that must be solved to provide a reliable multipoint service is the recovery from packet losses due to congestion in the network nodes and end systems.

Different approaches (Ito, 1992), (Strayer, 1992), (Feldmaier, 1994), (Sterbenz, 1991), (Braun, 1993a) on implementing high performance communication subsystems have been undertaken during the last few years: software optimisation, parallel processing, hardware support, and dedicated VLSI components. Some of the approaches deal with efficient implementations of standard protocols such as OSI TP4 or TCP. Others developed protocols especially suited for advanced implementation environments.

The use of dedicated VLSI components is mostly limited to very simple communication protocols, only (e.g., (Balraj, 1992), and (Krishnakumar, 1993)). In this paper, we present VLSI support that is especially targeted towards more complex multicast protocols. As an example for the provision of specific support for processing intensive functions, the implementation of a dedicated coprocessor for selective retransmissions in a multicast environment is shown. It is also shown how this coprocessor can be integrated into a protocol processing unit featuring parallel processing and direct ATM access.

This paper is organised as follows: Section 2 gives an overview of multipoint communication in high-speed networks and presents the conceptual framework for integrating VLSI components for multicast support into end systems and Group Communication Servers. Section 3 discusses the functionality of a dedicated coprocessor for managing retransmission, and presents performance results as well as implementation complexity of the discussed component. Section 4 summarises the paper and points out some future directions.

2 MULTIPOINT COMMUNICATION IN HIGH-SPEED NETWORKS

2.1 Error Control

The dominant factor which causes high speed networks to discard packets is buffer overflow due to congestion. The probability for packet loss may vary over a wide range, depending on the applied strategy for congestion control. For multicast connections, the problem of packet losses is even more crucial than for unicast connections. It is more difficult to ensure a low packet loss rate. Losses occur more frequently, and every loss causes costly processing for a multicast transmitter.

For applications that cannot tolerate packet losses of the network, error control mechanisms are required. Error control is a difficult task in networks that offer high bandwidth over long distances, where a large amount of data may be in transit. Two mechanisms are available for error correction: Automatic Repeat ReQuest (ARQ) and Forward Error Correction (FEC).
In contrast to the retransmission schemes, FEC promises a number of advantages (McAuley, 1990). The delay for error recovery is independent of the distance, and large bandwidth-delay products do not lead to high buffer requirements. Therefore, FEC is a promising approach in high-speed networks. In contrast to ARQ mechanisms, FEC is not affected by the number of receivers. However, FEC has three main disadvantages when applied for error correction in high speed networks. It is computationally demanding, leading to complex VLSI components. It requires constantly additional bandwidth, limiting the achievable efficiency and increasing packet loss during periods of congestion. The latter limits the usefulness of FEC in many cases. For an accurate assessment of FEC it must be considered that its best performance is achieved for random errors, while packet losses frequently occur in bursts (Biersack, 1993). The question when to apply FEC for real-time applications in high-speed networks requires extended assessments of various trade-offs. FEC has certain attractive properties in high-speed WANs and for multipoint connections. However, only retransmission schemes are able to provide fully reliable services. In many cases, retransmission schemes are superior to FEC in terms of the achievable throughput, the delay properties, or the implementation costs.
Protocols based on ARQ mechanisms are widely used in current data link and transport protocols. However, for high-performance multicast communication, there are still many open questions concerning acknowledgement and retransmission strategy, achievable performance and implementation. Retransmissions may be performed as go-back-N (e. g, in TCP) or as selective repeat (e. g., offered in XTP (XTP Forum, 1994a) and PATROCLOS (Braun, 1993b)). While go-back-N schemes are appropriate for point-to-point communication with low error rates and moderate path capacities, selective repeat schemes are essential for high-performance multicast communication in wide-area networks that may observe congestion (Carle, 1994). Large groups require that the transmitter stores and manages a large amount of status information of the receivers. The number of retransmissions is growing for larger group sizes, decreasing the achievable performance. Additionally, the transmitter must be capable of processing a large number of control information. If reliable communication to every multicast receiver is required, a substantial part of the transmitter complexity is growing proportionally with the group size. In addition, individual receivers may limit the service quality of the whole group. To overcome these problems, a scheme that provides reliable delivery of messages to K out of N receivers may be applied (K-reliable service, (Santoso, 1992)).

2.2 High-Performance Multicast Services

In order to meet the QoS requirements of many real-time applications, it is a common approach to meet the application reliability requirements without performing error control mechanisms. In situations where it is difficult to provide a network bearer service which meets the reliability requirements of the application directly, the following strategy may be applied: providing high protocol processing capability with a low latency and for ensuring that real-time requirements are met even after one or two retransmissions of a message. This strategy potentially offers a way for a better utilisation of network resources in particular for highly bursty source, as it allows to increase the load of intermediate systems up to a level in which losses relatively frequent.
In the past it was frequently debated whether real-time applications can be based on services with retransmissions. In (Dempsey, 1993), it was shown by simulation that a real-time retransmission

scheme is feasible within the end-to-end delay constraints of packet voice transmissions for overall one-way delays with an average of 12 ms and a maximum of 36 ms. While the authors of (Dempsey, 1993) used relatively high network access delays and protocol processing delays in transmitter and receiver in modelling the one-way delay, the propagation delay of 5 ms for a fibre-optic transmission over a distance of 1000 km shows that retransmission schemes for real-time applications may also be applied for relatively large distances.

A conceptual framework was described (Carle, 1994) for the use of error control mechanisms best suited for a specific multipoint communication scenario at locations that allow highest performance. The integration of specialised multicast components into the end systems represents an important step towards a high performance reliable multicast service. Further improvements of performance and efficiency may be achieved by the integration of dedicated servers in the network that provide support for group communication. In many cases of multicasting, the achievable throughput degrades fast for a growing group size. A significant advantage can be achieved if a hierarchical approach for multicast error control is chosen.

The support for protocol processing presented in this paper allows selective retransmissions to multiple receivers. This error control functionality may be part of a transport protocol, such as XTP Revision 4.0 (XTP Forum, 1994b) in combination with a connectionless network layer. This functionality may also be part of a transfer protocols combining layer 3 and layer 4 functions, such as XTP Revision 3.7 and PATROCLOS (Braun, 1993a) with multicast extensions. Such a transfer protocol may be used over a conventional LLC service, or over an adaptation layer service as for example offered by AAL5.

The framework on which this paper is based applies to protocols that use gaps of packet sequence numbers for positive and negative acknowledgements. Typical protocols use either sequence numbers identifying the first byte of the payload (as for example TCP, XTP and PATROCLOS), or they use packet sequence numbers (such as TP4, SNR (Sabnani, 1990), and SSCOP (ITU, 1994)). While byte sequence numbers usually have a length of 32 bit (or even 64 bit in XTP 4.0), packet sequence numbers with a length of 24 bits are sufficient even for high-speed WANs.

Figure 1 presents a network scenario with multicast mechanisms in the transport component of end systems and in dedicated servers. The term Group Communication Server describes an intermediate system with multicast error control capability which may be attached to conventional subnetworks or to an ATM network. It may be combined with routing functionality of, e.g., an XTP router.

The Group Communication Server (GCS) presented in this paper may integrate a number of mechanisms that can be grouped into three main tasks:

- Provision of a high-quality multipoint service with efficient use of network resources;
- Provision of processing support for multicast transmitters;
- Support of heterogeneous hierarchical multicasting.

For the first task, performing error control in the server permits to increase network efficiency and to reduce delays introduced by retransmissions. Allowing retransmissions originating from the server avoids unnecessary retransmissions over common branches of a multicast tree. In order to ensure low delay, the server does not guarantee an in-sequence forwarding of packets. Instead, it will forward every packet to the receivers as soon as possible. In combination with a network node with copy function, it is not required that the server processes a packet before forwarding to the receivers. Instead, copies may be forwarded in parallel to the server and the receivers. This guarantees minimal delay while allowing that the server detects losses prior to the receiver and initiates a retransmission by the sender.

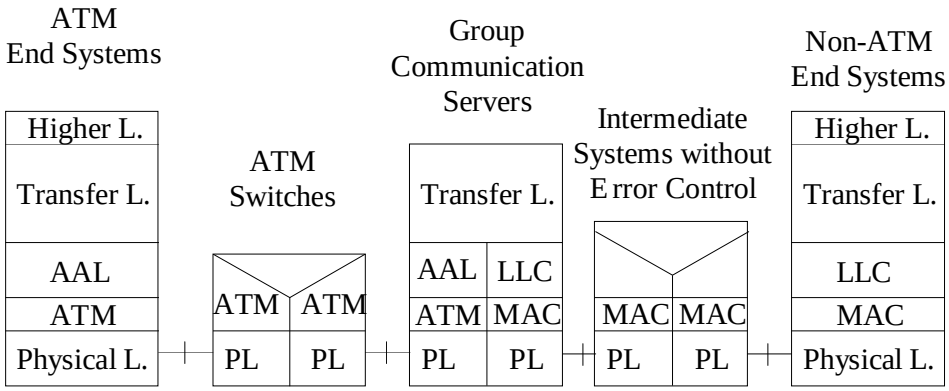


Figure 1 Support for reliable multicasting in servers and end systems.

For the second task, the GCS releases the burden of a transmitter that deals with a large number of receivers, providing scalability. Instead of communicating with all receivers of a group simultaneously, it is possible for a sender to communicate with a small number of GCSs, where each of them provides reliable delivery to a subset of the receivers. Integrating hardware support for reliable high performance multipoint communication in a server allows better use of dedicated resources such as coprocessors. For end systems, it is not required to have VLSI components for multicast error control. It will be sufficient to have access to a local GCS for participation in a high performance multipoint communication over long distances. Then, the error control mechanisms of individual end systems have only negligible influence on the overall performance, as simple error control mechanisms are sufficient for communication with a local GCS.

For the third task, a GCS may use the potential of diversifying outgoing data streams, allowing support of different qualities of service for individual servers or subgroups, may apply filtering functions for specific data streams. It may support different subnetworks, such as FDDI and ATM, where a different set of protocol parameters may be appropriate, and may also support different error control schemes, such as Go-back-N and selective repeat.

3 VLSI FOR RETRANSMISSION SUPPORT

The processing overhead associated with handling selective retransmissions and the required data structures may be extremely high compared to other protocol functions. As an example, the following protocol processing latencies may be observed for an XTP (XTP Forum, 1994a) implementation on Digital Alpha Workstations (150 MHz, 6.66 ns cycle time). The function to insert a new gap in a list needs 824 4-byte commands in the best case and takes, therefore, approximately 5.4 µs. The best case occurs if the new entry can be inserted at the beginning of the list. If the new entry has to be inserted after the first 10 entries, it needs 4054 commands or approximately 27.03 µs due to the search operations in the list. These calculations assume that the processor is not interrupted during execution of this function and all data is stored in the fast processor cache. In this example, XTP was implemented using C without special inline assembly code. Data sent at a rate of 1 Gbit/s results in more than 122,000 1024-byte packets per second. If the retransmission of each packet has to be controlled, this results in a new entry in the list in less than 10 µs.

Clearly, retransmission support is a time-critical task especially in a multicast environment. Therefore, we propose dedicated VLSI support for this task. The retransmission support presented in the following section can handle negative selective, positive selective, and positive cumulative ac-

knowledgements and can be used for gaps managed by the receiver to support the acknowledgement function, or for gaps managed by the transmitter to support the retransmission mechanism. The ALU has a set of commands to set, delete, insert, and read gaps for unicast or multicast connections. It can be used in Group Communication Servers as well as in other high performance end systems (Braun, 1994).

3.1 Logical Representation of Data

A dynamic linked list stores gaps of transmitted data in the following representation: *[seq_no_1, seq_no_2]* with *seq_no_1* and *seq_no_2* representing the beginning and end of a gap. These gaps are connected via linked lists (cf. Figure 2). For every multicast connection (MC), the pointers to the receivers participating in that connection are stored. For every receiver, the ALU stores a pointer to the appropriate list of gaps. Additionally, the ALU manages special lists for every multicast connection.

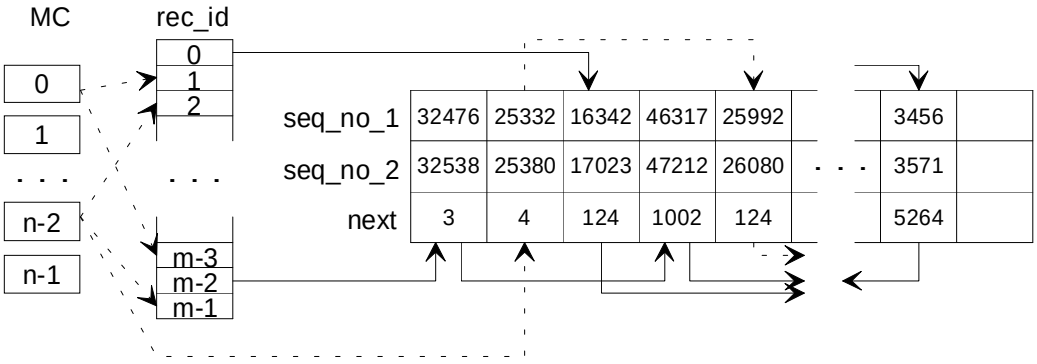


Figure 2 Logical structure of linked lists for retransmission support.

Depending on the implemented protocol, retransmission of data can be performed by multicast to all receivers, or by individual retransmissions to the appropriate receivers.

3.2 Operations of the Retransmission ALU

The component performs not only the insert and delete operations for the list, but also joins two adjoining gaps and updates the group list for a given reliability. Appendix A gives some examples of implemented operations of the retransmission ALU. Every operation sets the error flag if it failed due to memory overflow or violation of several conditions, such as $high\_ack \leq seq\_no \leq high\_seq$ and other range checking.

3.3 Implementation Architecture for Retransmission Support

Figure 3 shows an overview of the internal structure of the ALU. The retransmission ALU consists of a data memory that stores sequence numbers representing gaps, pointers of linked lists, and state information, such as connection and multicast identification, register number of an anchor element, and other flags indicating the state of a connection. The I/O-bus connects the input/output-port (32 bit) of the retransmission ALU with the 5 register banks (Ai through Ei, $0 \leq i \leq 3$). From these registers data can be transferred to the memory via the move unit.

Two simple ALUs (*ALU A*, 32 bit and *ALU D*, 8 bit) perform the operations OR, XOR, AND, ADD, SUB, and NEG. Two specialised 32 bit modulo 2^{32} comparators (*COMP 1* and *COMP 2*)

perform fast comparisons needed for list operations. The ALUs and the comparators can work concurrently if no data dependencies exist.

For the command `set_gap_2`, for example, first of all the command itself and the connection identification are read from the I/O-bus into the registers E and D, respectively. In the next two cycles the central control unit reads the sequence numbers (`seq_no_1`, `seq_no_2`) into the registers A and B, respectively. After reading the complete command and performing several range checking operations, the loop for searching the right position to insert the new entry in the list starts. Therefore, the first entry of the appropriate list is loaded into the registers A and B, respectively, and compared with the new entry. The loop terminates if the new entry fits. Otherwise, the next entry is loaded and compared.

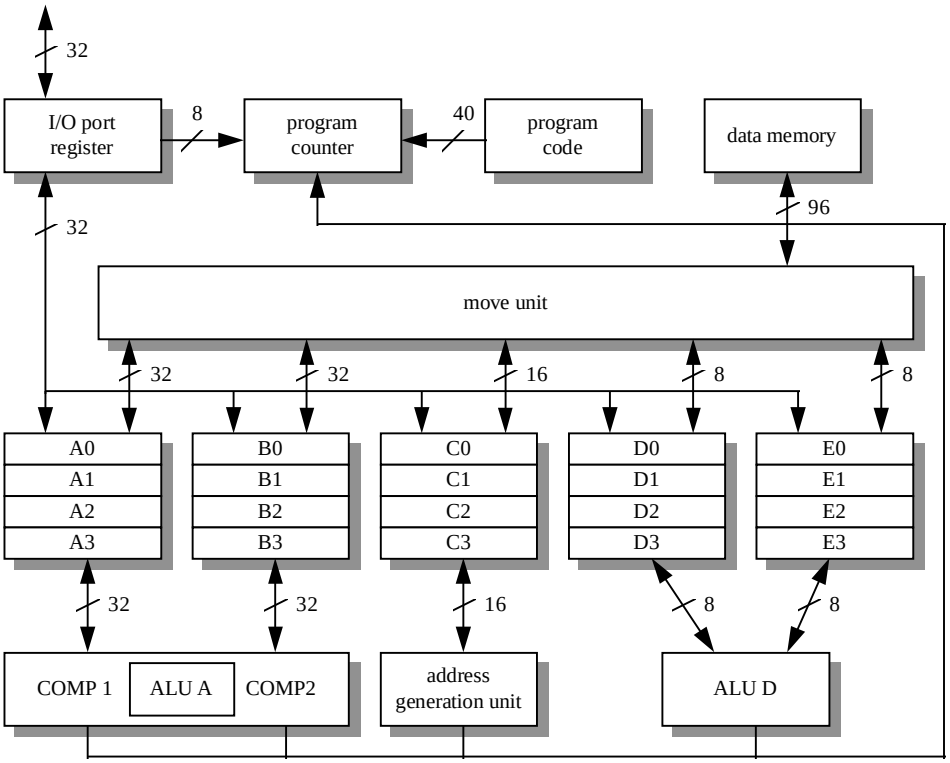


Figure 3 Functional architecture of the retransmission ALU.

3.4 Memory Management

The retransmission processor stores all list data in a single address space. Therefore, it is necessary to design a special memory management unit. Memory is divided into 8 separated areas (cf. Figure 4), each with a dedicated LIFO to manage the pointers to the memory.

The *connection list* stores information related to a certain unicast connection, such as the lowest and highest sequence numbers in use, a pointer to the appropriate unicast list, and additional status information (number of gaps, empty etc.). The gaps of an unicast connection are stored in the *unicast list* with starting and ending sequence numbers and a pointer to the next entry. Maximum memory for unicast applications can be calculated as follows:

*memory_depth = NR + NG*NR*

*memory_size = memory_depth * 96 Bit*

with *NR* denoting the maximum number of receivers and *NG* the maximum number of gaps stored per receiver.

65535	multicast auxiliary list 2
	multicast auxiliary list 1
	multicast list
	multicast start list
	group list
	group start list
	unicast list
0	connection list

Figure 4 Memory structure for the retransmission processor.

Additional memory areas are needed to support multicast. The *group start list* stores besides status information a pointer to the connection lists of its first member and a pointer to the appropriate group list. The *group list* itself stores the pointers to the other members of a group. This is done to provide a maximum of flexibility, weather there are 500 multicast connections with up to 10 receivers, 10 multicast connections with up to 300 receivers, or other combinations. The *multicast start list* and the *multicast list* store similar to the connection list and the unicast list the lowest and highest sequence number in use inside a multicast group and all gap information. The *multicast auxiliary list 1* and *2* are used to support list operations. Maximum memory for multicast application is calculated as follows:

$$memory\_depth = NR + NG*NR + MC + MC*(RM-1) + (RM-1) + RM*NG + NG + NG$$

$$memory\_size = memory\_depth * 96\ Bit$$

with *MC* denoting the maximum number of multicast connections, and *RM* denoting the maximum number of receivers per multicast connection. A multicast scenario with, e.g., NR = 1024 receivers with a maximum of NG = 30 gaps per receiver and 32 multicast connections with up to RM = 256 receivers each, results in a maximum memory need of 47931 * 96 Bit = 562 kByte.

One essential feature of this implementation is its inherent flexibility. Protocols like SSCOP (ITU, 1994) or RMC-AAL (Carle, 1994) use sequence numbers of 24 bits instead of instead of 32 bit sequence numbers of XTP. The ALU can be easily adapted to smaller sequence numbers. An ATM WAN with 5000 km maximum distance and a link capacity of 600 Mbit/s has a round trip capacity of approximately 70000 cells. The component will be dimensioned as follows. If the cell loss rate is known to be less than 10^{-4} for the time interval of one RTT, at most 7 cells will be lost during one RTT. If independent cell losses are assumed, at most 7 packets are corrupted during this time interval. Therefore, assuming an average of 8 gaps will be sufficient. With a maximum of 1024 connections in parallel, approximately 1024 + 8*1024 = 9216 entries will be needed in the list for unicast connections only. Now the memory width is only 24+24+16+8+8 = 80 bit. For this scenario an additional 9216*80 = 90kbyte RAM is needed. For the multicast scenario shown above, memory size can be calculated as follows.

$$memory\_size = [1024 + 8*1024 + 32 + 32*255 + 255 + 256*8 + 8 + 8]*80\ bit = 2.4kbyte$$

The complete VLSI coprocessor for this scenario is described in section 3.6.

3.5 Microcode Examples of the Retransmission ALU

To provide a maximum of flexibility all functions of the *Retransmission ALU* are translated into a sequence of microcode operations. These operations are adapted to the implementation architecture shown in Figure 3. The *program counter* controls the microprogram via a special micro sequencer.

Essential microcode operations of the *Retransmission ALU* are listed in the following table. The operations of the ALUs, the central control unit and the comparators are always executed in parallel in one clock cycle. A representative example for the use of the microcode operations can be found in Appendix B, where range-checking for the operation `delete_gap` is listed.

Table 1 Microcode examples of the retransmission ALU

operations	***comment***
`RMOVE S, D`	move a complete row of entries from the registers or RAM into the registers or RAM. S, D ∈ {Ri, RAM; 0 ≤ i ≤ 3}, Rn = (An, Bn, Cn, Dn, En), S ≠ D
`ANOP`	no operation, ALU A
`MOVE I/O, D`	move data from the I/O-bus into the register D; D ∈ {Ai, Bi; 0 ≤ i ≤ 3}
`TBBC Ri.n, ra`	test bit n of register Ri and branch to relative address ra if clear; R ∈ {Di, Ei; 0 ≤ i ≤ 3}, 0 ≤ n ≤ 7

`CBMOD Ri, Rj, Rk, Rl, ra1, ra2, ra3`	compare Ri ≤ Rj ≤ Rk and Rj ≤ Rk ≤ Rl modulo 2^{32} and branch to:
	result = 00 then PC := PC + 1; result = 01 then PC := PC + ra1;
	result = 10 then PC := PC + ra2; result = 11 then PC := PC + ra3;
	PC: program counter; Ri, Rj, Rk, Rl ∈ {Ai, Bi; 0 ≤ i ≤ 3}

`AADD S, D`	S + D -> D; S, D ∈ {Ai, Bi; 0 ≤ i ≤ 3}

3.6 Implementation Results

The retransmission processor was designed, simulated, and synthesised using the hardware description language VHDL (IEEE, 1987) in combination with commercial design tools. The control logic of the processor needs 28800 gates, the critical path is 45 ns using a 0.7 µm CMOS standard cell library. The two comparators COMP 1 and COMP 2 together with the ALU A need 8000 gates. The comparators are implemented as four 8 bit carry-select adders forming a ripple carry adder. The address generation unit consists of 2000 gates plus 7300 gates for memory management. 1600 gates are used in the 8 bit ALU D. The data registers Ai through Ei need 8400 altogether. For the program counter including its micro-sequencer 1500 gates are needed. The move unit is distributed over the registers and, therefore, included in their gate count. It needs 10 ns to decode a microcode operation, 7 ns to fetch the appropriate data from a register, a maximum of 23 ns to execute the operation, and 5 ns to store the results in the registers. Performing only one half of the CBMOD (c.f. Table 1) operation on an Alpha processor needs 11 operations which results in a duration of more than 72 ns (6.6 ns cycle time, incl. load/store). The coprocessor needs only 45 ns for the complete CBMOD operation and, therefore, this is the point of further optimisations. This implementation also allows to perform up to four operations in parallel. The die size of this chip for the control logic is 49 mm² if the chip is adapted to XTP (c.f. Appendix C). Assuming RMC-AAL as protocol, not only the data paths can be smaller, but also operations like CBMOD will be faster. The reason for this is the trade-of for speed and chip-area. For the comparators now only three 8 bit carry select adders will be needed and, therefore, one does not have to wait for the forth carry select adder until it gets the carry-bit from the third unit and finally present the result at the output. For further increase in speed the whole adder could be built as carry select adder or other fast implementation variants. Table 2

shows the influence of four different protocols on the chip size and speed if this component is used to support list processing. The differences are mainly due to different sizes of the sequence numbers.

Table 2 Influence of different protocols on synthesis results

	XTP 3.6	*XTP 4.0*	*SSCOP, RMC-AAL*
number of gates			
COMP1, COMP2, ALU A	8 000	16 000	6 000
address generation	2 000	2 000	2 000
memory management	7 300	7 300	7 300
ALU D	1 600	1 600	1 600
registers A through E	8 400	16 000	7 000
program counter, µsequencer	1 500	1 500	1 500
Σ	28 800	44 400	25 400
critical path (ns)			
decoding	10	10	10
fetch data	7	7	7
execution	23	35	20
store data	5	5	5
Σ	45	57	42

3.7 System integration

Distribution of protocol processing tasks onto a number of units operating in parallel plays a key role in the provision of high performance services (Zitterbart, 1993). The functionality of transfer protocols to be executed in end systems and in the GCS can be distributed onto several general purpose processors. Additional performance improvements can be achieved by additional support with dedicated coprocessors. Advances in VLSI technology allow to integrate multiple RISC processors, memory, a switching unit, and additional components for special processing and I/O onto a single chip.

Figure 5 shows an integration of the list processing unit into a generic coprocessor with direct ATM interfaces that provides multiple processing units coupled by a switching unit. This architecture is similar to the TMS320C80 (Texas Instruments, 1994). The unit called GAPPU (Generic ATM Protocol Processing Unit) is unique in combining parallel processing with direct ATM access and components for retransmission support. In Figure 5, the 6 microprocessors provided by GAPPU are used for a software implementation of basic GCS modules (Receiver Processor, Transmit Processor, Send Manger, Ack Manager, Frame Manager Receive, Frame Manager Transmit).

Special coprocessors provide more complex time-critical functions. One example is the list coprocessor discussed above. UTOPIA is used as interface to ATM, a DMA unit performs all data transfer operations between host and network interface. For use in interworking units further network interfaces could be added. All components can communicate via a simple crossbar switch which supports two priority levels. To guarantee certain quality of service a fair round-robin scheduling strategy is used with the crossbar. Up to now, the list coprocessor, the timer unit and the FEC unit together with the crossbar and memory have been implemented using VHDL and powerful synthesis tools.

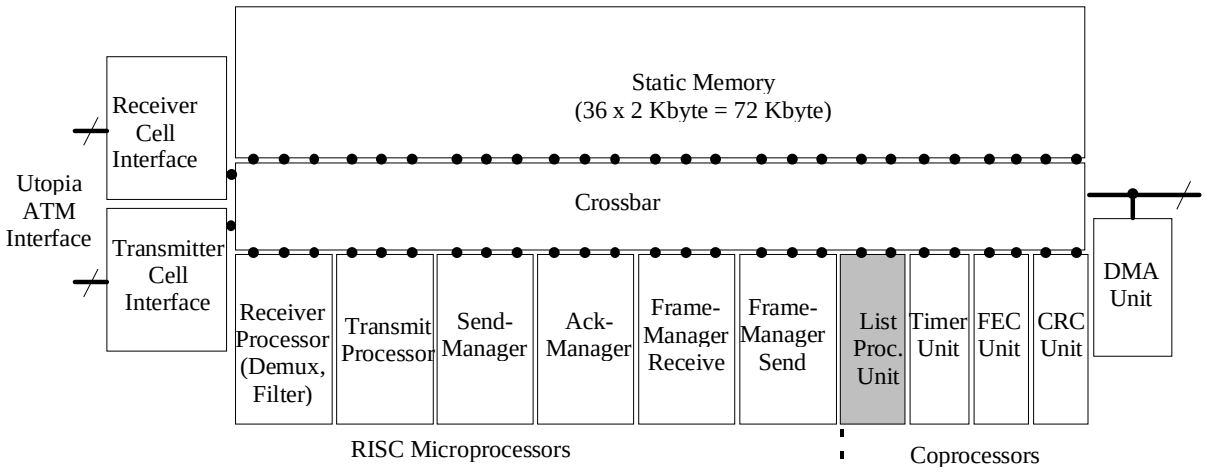


Figure 5 Architecture of GAPPU, the Generic ATM Protocol Processing Unit.

4 SUMMARY AND FUTURE WORK

Within this paper, a framework for the provision of high performance multicast services has been presented which has the potential to fulfil the requirements of upcoming distributed applications. It is based on VLSI components dedicated to specific processing tasks that are to be integrated into end systems and Group Communication Servers. Details of retransmission support have been discussed. The architecture of the generic protocol processing unit shows how multiple microprocessors and specialised VLSI components can be combined on a single chip.

However, not only high protocol processing performance and efficient use of network resources, but specifically service integration are required for forthcoming communication subsystems. Components may be parametrized based on the requested application service, providing a high degree of flexibility. Currently, the implementation of additional components for CRC and memory management are under development..

Acknowledgement

The support by the Graduiertenkolleg „Controllability of Complex Systems“ (DFG Vo287/5-2) is gratefully acknowledged.

5 REFERENCES

Balraj, T.; Yemini, Y. (1992) *Putting the Transport Layer on VLSI - the PROMPT Protocol Chip;* in: Pehrson, B.; Gunningberg, P.; Pink, S. (eds.): Protocols for High-Speed Networks, III, North-Holland, pp. 19-34

Biersack, E. W. (1993) *Performance Evaluation of Forward Error Correction in an ATM Environment*; IEEE Journal on Selected Areas in Communication, Volume 11, Number 4, pp. 631-640, May 1993

Braun, T.; Zitterbart, M. (1993a) *Parallel Transport System Design;* in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, pp. 397-412

Braun, T. (1993b) *A Parallel Transport Subsystem for Cell-Based High-Speed Networks;* Ph.D. Thesis (in German), University of Karlsruhe, Germany, VDI-Verlag, Düsseldorf, Germany

Braun, T.; Schiller, J.; Zitterbart, M. (1994) *A Highly Modular VLSI Implementation Architecture for Parallel Transport Protocols;* IFIP 4th International Workshop on Protocols for High-Speed Networks, Vancouver, Canada, August 10-12, 1994

Bubenik, R.; Gaddis, M.; DeHart, J. (1992) *Communicating with virtual paths and virtual channels*; Proceedings of the Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM'92, pp. 1035 - 1042, Florence, Italy, May 1992

Carle, G. (1994) *Adaptation Layer and Group Communication Server for Reliable Multipoint Services in ATM Networks;* in: Steinmetz, R. (ed.): Multimedia: Advanced Teleservices and High-Speed Communication Architectures, Springer, 1994, pp. 124-138

Dempsey, B.; Liebherr, J.; Weaver, A. (1993) *A New Error Control Scheme for Packetized Voice over High-Speed Local Area Networks*, Proceedings of 18th Conference on Local Computer Networks, September 19-22, 1993, Minneapolis, Minnesota, U.S.A, pp. 91-100

Feldmeier, D.C. (1994) *An Overview of the TP++ Transport Protocol; in:* Tantawy A.N. (ed.): High Performance Communication, Kluwer Academic Publishers

Heinrichs, B.; Jakobs, K.; Carone, A. (1993) *High performance transfer services to support multimedia group communications*; Computer Communications, Volume 16, Number 9, September 1993

IEEE (1987) *Standard VHDL Language Reference Manual;* IEEE Std 1076-1987

Ito, M.; Takeuchi, L.; Neufeld, G. (1992) *Evaluation of a Multiprocessing Approach for OSI Protocol Processing*; Proceedings of the First International Conference on Computer Communications and Networks, San Diego, CA, USA, June 8-10, 1992

ITU-T (1994) Draft Recommendation Q.2110: *B-ISDN Adaptation Layer - Service Specific Connection Oriented Protocol (SSCOP)*, Geneva

Krishnakumar, A.S.; Kneuer, J.G.; Shaw, A.J. (1993) *HIPOD: An Architecture for High-Speed Protocol Implementations*; in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, pp. 383-396

McAuley, A. (1990) *Reliable Broadband Communication Using a Burst Erasure Correcting Code*; Presented at ACM SIGCOMM '90, Philadelphia, PA, U.S.A., September 1990

Sabnani, K.; Netravali, A.; Roome, R. (1990) *Design and Implementation of a High Speed Transport Protocol*, IEEE Transactions on Communications, Vol. 38, No. 11, pp. 2010-2024, November 1990

Santoso, H.; Fdida, S. (1993) *Transport Layer Multicast: An Enhancement for XTP Bucket Error Control*, in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland

Sterbenz, J.P.G.; Parulkar, G.M. (1991) *AXON Host-Network Interface Architecture for Gigabit Communications;* in: Johnson, M. J. (ed.): Protocols for High-Speed Networks, II, North-Holland, pp. 211-236

Strayer, W.T.; Dempsey, B.J.; Weaver, A.C. (1992) *XTP: The Xpress Transfer Protocol;* Addison-Wesley Publishing Company

Texas Instruments (1994) *TMS320C80 Multimedia Video Processor (MVP): Technical Brief*, Texas Instruments, Houston, Texas

The XTP Forum (1994a) *XTP Protocol Definition Proposed Revision 3.7*

The XTP Forum (1994b) *XTP Protocol Definition Proposed Revision 4.0*

Waters, A. G. (1992) *Multicast Provision for High Speed Networks*; 4th IFIP Conference on High Performance Networking HPN'92, Liège, Belgium, December 1992

Zitterbart, M.; Tantawy, A.N.; Stiller, B.; Braun, T. (1993) *On Transport Systems For ATM Networks*, Proceedings of IEEE Tricomm, Raleigh, North Carolina, U.S.A., April 1993

Appendix A: Example Operations of Retransmission ALU

operation	*input parameters*	*output parameters*	*comment*
init_list	rec_id, seq_no		initializes a new list for the connection rec_id with the initial sequence number seq_no, sets the error flag if rec_id is already in use
close_list	rec_id		closes the list for connection rec_id, sets the error flag if the list does not exist
init_mcg	mc_con_id, rel		initializes a new multicast group with the identification mc_con_id and the reliability rel (rel ≥ number of connections denotes full reliability)
close_mcg	mc_con_id		closes a mulitcast group and deletes all linked lists
add_mcg	mc_con_id, rec_id		adds a new connection rec_id to an existing multicast group mc_con_id
set_rel	mc_con_id, k		sets the value k for the reliability of the multicast group mc_con_id
set_high_ack	rec_id, seq_no		sets the high_ack register to the value of seq_no; sequence numbers less than high_ack have been already acknowledged
shift_high_seq	rec_id, length		shifts the high_seq register to high_seq + length
set_gap_1	rec_id, seq_no, length		inserts new entry (seq_no, seq_no + length); overlapping entries are automatically joined or deleted, respectively
del_gap_1	rec_id, seq_no, length		deletes an existing entry, a part of an existing entry, or several existing entries, the deleted part is of the form (seq_no, seq_no + length); if necessary an entry is automatically divided into two new entries
read_reg	rec_id, reg_id	cont	reads the contents cont of the register reg_id (e.g. high_ack, high_seq, number_of_gaps)
read_mc_reg	mc_con_id, reg_id	cont	reads the contents cont of the multicast register reg_id (e.g. number_of_gaps)
get_gap_1	rec_id, ptr	seq_no, length, next	reads the entry ptr points to; if ptr = 0, the first gap is read out, if next = 0 the entry represented by (seq_no, length) is the last one, otherwise next point always to the next entry of the list
get_mc_gap_1	mc_con_id, ptr	seq_no, length, next	analogous to get_gap_1, but now the entries of the multicast group mc_is are read out

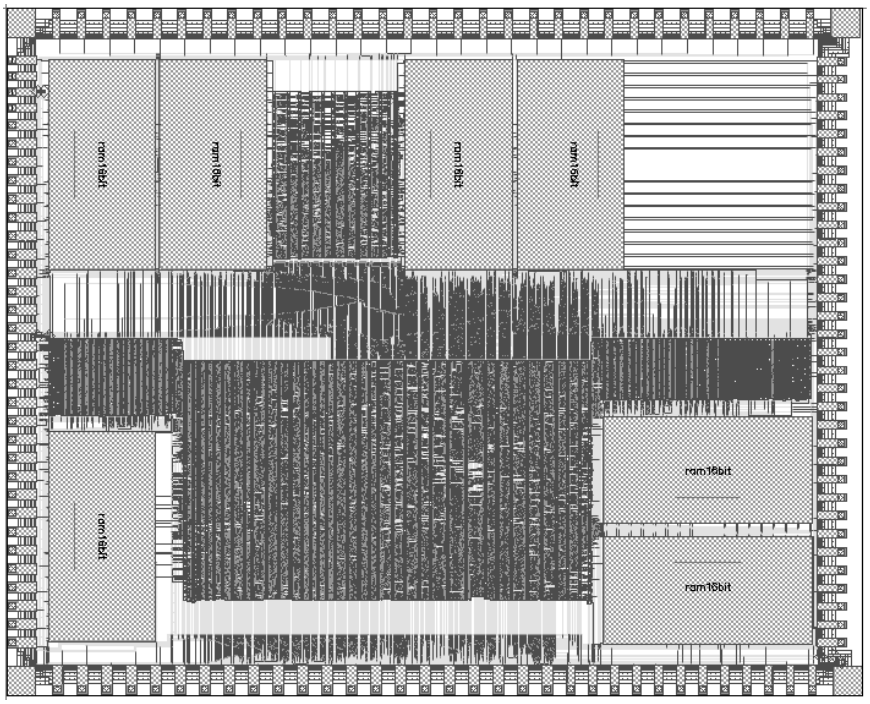
Dimensioning of the component: *rec_id, k* ∈ [0, 255]; *mc_con_id* ∈ [0, 63]; *seq_no, seq_no_1, seq_no_2, length, cont* ∈ [0, 2^{32}-1]; *reg_id* ∈ [0, 15]; *ptr, next* ∈ [0, 2^{16}-1]

Appendix B: Microcode

```
-- DEL_GAP_1
-- delete an existing entry, a part of an existing entry, or several existing
-- entries, the deleted part is of the format (seq_no_start, seq_no_end)
-- if necessary an entry is automatically divided into two new entries
-- start address: 002C (preprocessing for subroutine delete gap)

002C: MOVE I/O, C0      -- load connection ID
      RMOVE[L] C0,R1; MOVE I/O,A0   -- load context and seq_no_start
                  MOVE I/O,B0 -- load seq_no_end
      TBBC D1.7,D1_ERR        -- exists list?
      INC A1,A2               -- high_ack+1
      INC B1,B2               -- high_seq+1
      MOVE A7,A3              -- load first gap L0
      CBMOD A3,A0,A2,B2,5,4,4 -- compare: high_ack-L0, seq_no_start, high_ack+1
                              -- high_ack+1, seq_no_start, high_seq+1
      SETBIT ACK              -- error
      SETBIT BUSY
      JMP WAIT
      MOVE A2,A0              -- seq_no_start:=high_ack+1
      CBMOD A3,B0,A2,B2,2,3,3 -- compare: high_ack-L0, seq_no_end, high_ack+1
                              -- high_ack+1, seq_no_end, high_seq+1
      MOVE B0,B1              -- high_seq:=seq_no_end
      JMP DEL_GAP             -- delete gap subroutine
      SETBIT BUSY
      SETBIT ACK
      JMP WAIT
D1_ERR:     SETBIT ACK
      SETBIT BUSY
      JMP WAIT
```

Appendix C: Chip Layout



Biographies

Georg Carle

Since Sept. 1992, Georg Carle is preparing a PhD in Computer Science at the University of Karlsruhe, Institute of Telematics. He obtained a degree in Electrical Engineering from the University of Stuttgart, where he performed his diploma project at the Institute for Communication Switching and Data Techniques. In 1990, he performed a seven month research project at the Département Communications, Télécom Paris. In 1989, he obtained the degree Master of Science in Digital Systems at Brunel University, London, U.K. His main areas of interest are protocol engineering for ATM networks, high performance protocol implementation, and performance evaluation.

Jochen Schiller

In 1993 Jochen Schiller received his diploma degree in Computer Science from the University of Karlsruhe, Germany. Currently he finishes his PhD thesis at the Institute of Telematics. He participates in the fellowship program ‘Controllability of Complex Systems’ of the faculty of Computer Science. Key aspects of his work are the efficient implementation of protocol functions in hardware, hardware/software codesign and the support of automatic synthesis of high-performance communication systems from formal specifications. He is member of the IEEE Computer and Communication Society since 1993.