Reprint of Chapter in Book: "State-of-the Art in Performance Modeling and Simulation", G. Zobrist, J. Walrand, K. Bagchi (Eds.), Gordon and Breach Publishers, Dec. 1998, ISBN: 90-5699-635-5

CHAPTER 10

MODELING, SIMULATION, AND SYNTHESIS OF HIGH-PERFORMANCE ATM PROTOCOLS AND MULTIMEDIA SYSTEMS

Georg Carle and Jochen Schiller

10.1. INTRODUCTION

Emerging applications mostly require both high performance as well as support for a wide variety of communication services. For example, audio, video, and data transmission may require highly different services, e.g., guaranteed delay, jitter, or bandwidth. An additional challenge arises through the growing demand for multipoint communication services. ATM networks are capable of satisfying the basic application requirements by providing multipoint bearer services[6] with data rates exceeding a gigabit per second. However, current communication subsystems (including higher layer protocols) that provide reliable services are not able to deliver the available network performance to the applications.[14,35] In particular in multipoint communication scenarios, severe degradations of service quality can be observed. Additional problems need to be addressed in scenarios where quality of service (QoS) requirements and processing capabilities of individual receivers differ.

In order to provide the required high performance services to the applications, new protocols[29,25] as well as high-performance implementation architectures for the communication subsystems need to be designed.[28,16,5] Dedicated VLSI components should be used in flexible implementation platforms for time-critical processing tasks, such as retransmission support or memory management, in order to provide high performance communiation services.

This chapter presents a new approach for the flexible design of hardware-supported high-performance communication subsystems together with a framework for the provision of multipoint multimedia services in ATM networks and heterogeneous internetworks. The design process allows

mapping of a formal protocol specification onto a parallel, hardware-based implementation architecture. The highly modular VLSI implementation architecture designed with parameterizable and programmable components allows for service flexibility. The architecture is not limited to a certain protocol, but allows the implementation of a variety of high-speed protocols. We validated our approach with a design example using a formal specification of the protocol RMC-AAL (Reliable Multicast ATM Adaptation Layer, RMC-AAL[9]). This protocol provides error control functions for ATM end and intermediate systems in order to enhance the reliability of an ATM service. The concept of integrating error control functions into Group Communication Servers (GCSs) allows for the efficient provision of reliable multipoint services in large, widespread groups. The multicast error control capabilities of GCSs allow for increased throughput and reduced delay. GCSs provide protocol processing support for multicast transmitters and reduce the acknowledgement implosion problem. [10] They also support groups consisting of end systems with direct ATM access, as well as end systems connected over heterogeneous internetworks. [8]
This chapter is structured as follows. Section 2 presents the developed ATM protocol for multipoint error control. Section 3 describes the design flow for implementation in detail and gives some examples for each design step. Section 4 presents conclusions and potential further work.

10.2. ATM PROTOCOL FOR MULTIPOINT ERROR CONTROL

The Reliable Multicast ATM Adaptation Layer features the options of frame-based automatic repeat request (ARQ), cell-based ARQ and forward error correction (FEC) for an efficient provision of reliable multicast services under varying cell loss rates. RMC-AAL offers a fully reliable service and a service that assures delivery to a subset of K receivers. It can be used in ATM end systems and also in dedicated servers within the network (see Figure 10.1).
Lost retransmissions contribute significantly to the QoS of a reliable group communication service.[10] It is therefore of high importance to decrease the probability of lost retransmissions. This goal can be achieved by FEC.[3] The mean number of retransmissions required for the successful delivery of a frame can also be decreased by establishing virtual channels (VCs) for retransmissions which have a lower cell loss rate than the VCs used for the first transmission of a frame. This capability of ATM is in contrast to

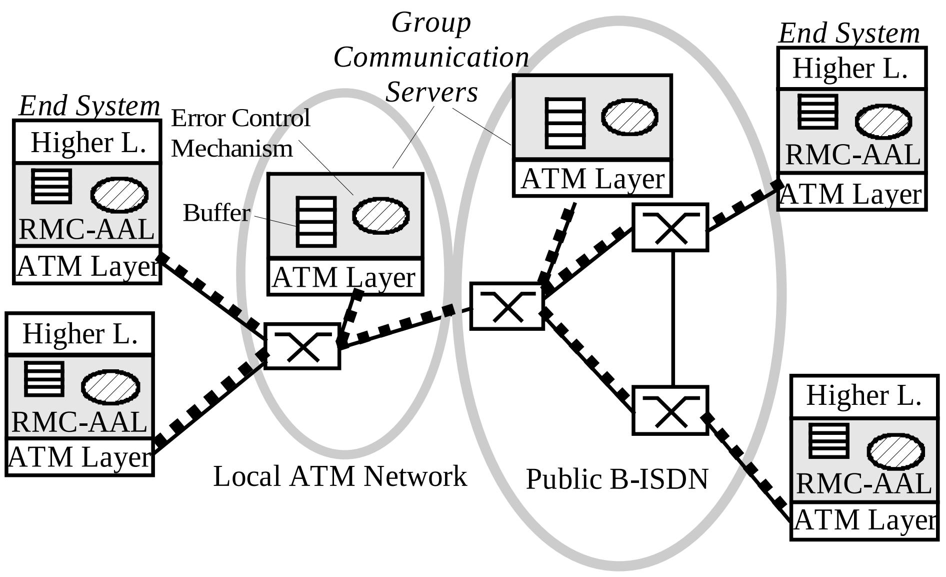

Figure 10.1: Multicast error control in servers and end system

single service networks, where initial transmissions and retransmissions will generally observe identical loss rates.

RMC-AAL allows to send retransmissions by multicast or by unicast in selective repeat or go-back-N mode. It can be selected if retransmissions are frame-based (by retransmission of the original data frames) or cell-based (by retransmission of frame fragments). When FEC is used, the information cells of the frame are protected by additional redundancy cells. Encoding and decoding can be based on Reed-Solomon-Codes,[23] or on simple XOR-operations and matrix interleaving.[27]

In the following, the data format used by RMC-AAL is briefly explained. RMC-AAL consists of a service specific convergence sublayer (SSCS) with ARQ and FEC functions, based on the common part convergence sublayer (CPCS) of AAL5.[17] AAL5 uses a trailer of 8 bytes and protects the payload of an AAL frame by the cyclic redundancy check CRC-32. In addition to this 8 byte trailer, RMC-AAL data frames use a 10 byte frame header. Frames are identified by a frame sequence number (FSN, 24 bit) in the frame header. When using cell-based ARQ, each cell has an additional protocol overhead of one byte: 2 bits for specifying the cell type, and a 6 bit cell sequence number (CSN). Even for high speed VCs in WANs, no large cell numbering space is required, because a hierarchical sequence numbering scheme is used, where each cell is identified by both FSN and CSN. The alternative solution of identifying cells entirely by their cell sequence numbers leads to a significantly higher overhead per cell. For example, the protocol BLINKBLT,[13] which also offers cell-based retransmissions, has a per-cell overhead of 4 bytes. The RMC-AAL frame header contains a transmitter identifier and the length of the SSCS PDU payload. The frame header also contains a discriminator field with an identifier for the frame type, a flag to request an immediate acknowledgement, a flag for identifying the last frame of a burst, and a field for the number of redundancy cells that follow the data frame. Frame fragments consist of a Fragment Header Cell, followed by a selection of original data cells of this frame. The fragment header cell contains the header information of a regular frame, as well as a bitmap for identification of the data cells which follow. Further details of the RMC-AAL protocol are presented in Carle and Zitterbart.[9]

10.2.1. Group Communication Server

The application of the presented error control mechanisms is not limited to ATM end systems. The deployment of so-called Group Communication Servers with multicast error control mechanisms provides reliable high-performance multipoint services for a wide range of parameters. Further improvements of performance and efficiency can be achieved by using GCSs hierarchically.

GCSs support an efficient use of network resources by performing multicast error control within the network. Allowing retransmissions originating from the server avoids unnecessary retransmissions over common branches of a multicast tree. The integration of FEC mechanisms into the GCS allows for the regeneration of lost cells and for the reinsertion of additional redundancy for adjusting the FEC coding scheme according to the needs of subsequent hops.

By providing protocol processing support for multicast transmitters, GCSs also improve scalability. They also support heterogeneous multicasting by allowing different protocol parameters for different branches of a multicast tree, and by converting between different error schemes. For example, the simple frame-based go-back-N retransmission scheme could be used between a GCS and local receivers, while the more complex cell-based ARQ scheme with additional FEC could be used for a wide area connection between transmitter and GCS.

For groups with multiple transmitters, the GCS provides support for multiplexing of frames onto a single point-to-multipoint connection. This reduces the number of required VCs significantly for large groups with many transmitters.[36] Virtual LANs frequently require this multiplexing functionality. However, an additional queuing delay may be introduced by this multiplexing function. If LAN Emulation[1] is used in a local ATM network, a GCS might be incorporated into the service elements ‘LAN Emulation Server’ (LES) and ‘Broadcast and Unknown Server’ (BUS), thus making it possible for applications to ensure the reliable delivery of multicast and broadcast messages to all peers.

10.2.2. Formal Protocol Specification

RMC-AAL for end systems as well as for GCSs has been formally specified in SDL (Specification and Description Language).[18] Figure 10.2 shows the structure of a simplified GCS. Each octagon represents a process in SDL.

A data frame arriving from the sender enters the diagram at the upper right corner and is lead to the process Frame Manager Receive (FM_Receive) by the receiver process Filter_snd, which forwards frames depending on the frame type. FM_Receive allocates memory and stores the frame. The frame is then scheduled for transmission by the process Send_Manager. The process Frame Manager Send (FM_Send) assembles the head of the frame and passes the frame to the transmitter process (Switch_rcv). This process ensures that cells of different frames are not interleaved. The Pool_Manager manages the buffer of the GCS and the lower window edges (LWEs) of all receiving entities, which can be either end systems, or GCSs. Acknowledgements arrive at the process Filter_rcv. FM_Repeat interprets the acknowledgements and passes the results on to the Pool_Manager. The core of the frame-based error handling is formed by the process Frame_Control. Acknowledgement frames are created by the process FAck_Creator.

10.2.3. Protocol Simulation

It is important to know which error control scheme of RMC-AAL is best suited for a given situation. The achievable performance of the proposed error control schemes was evaluated by applying discrete-event simulation with the simulation tool BONeS/Designer.[31] For modeling the correlation properties of lost cells, a two state Markov chain (Gilbert Model) was applied. Based on the worst case observations of Ohta and Kitami,[24] a probability of 0.3 was used for a cell discard following a cell discard. This is equivalent to cell losses with a mean burst length of 1.428 cells. A multicast VC was simulated, where cell losses may occur on the common link or on individual links. The same error model was applied to all links. A data rate of 100 Mbit/s, a distance of 100 km, and a frame length of 50 cells was used. Figure 10.3 shows the efficiency (relation of successfully

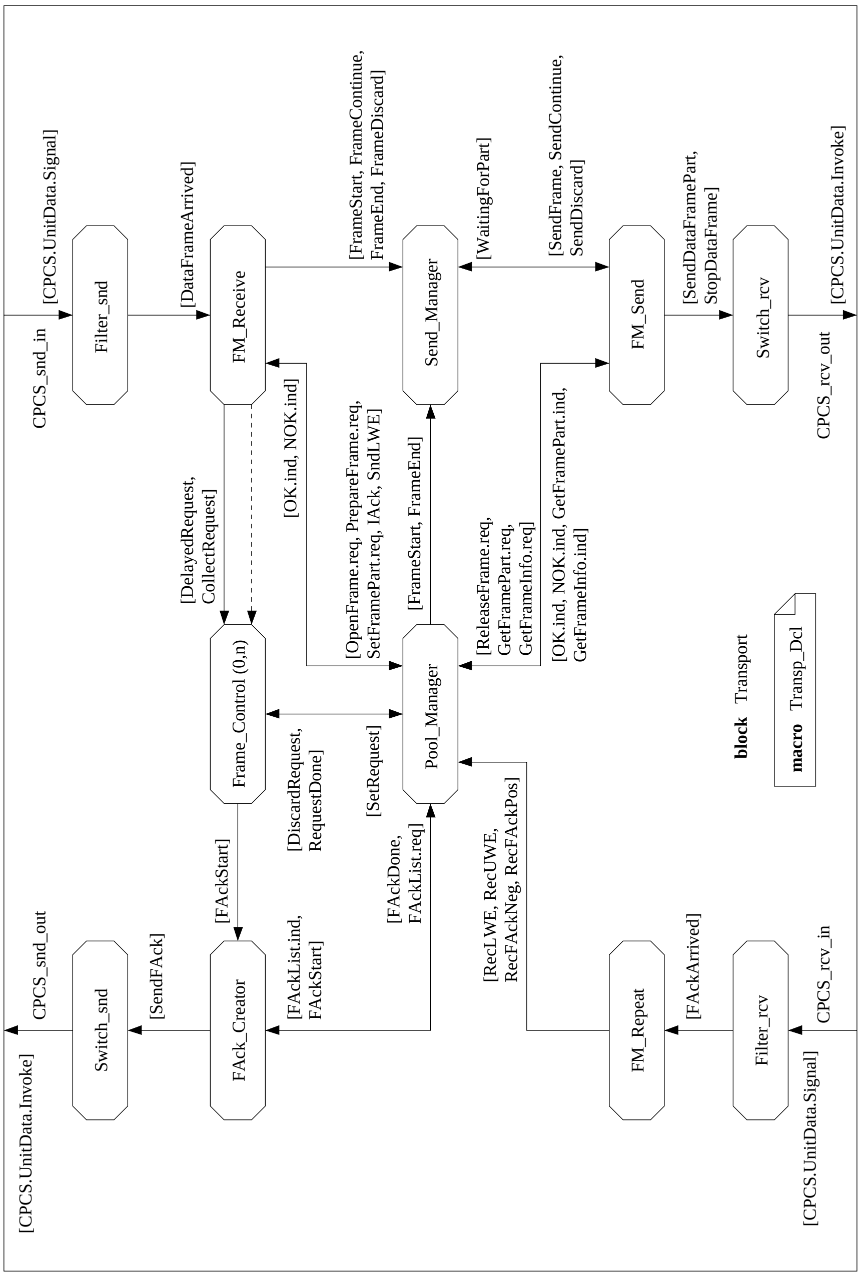

Figure 10.2: SDL specification of RMC-AAL for Group Communication Servers

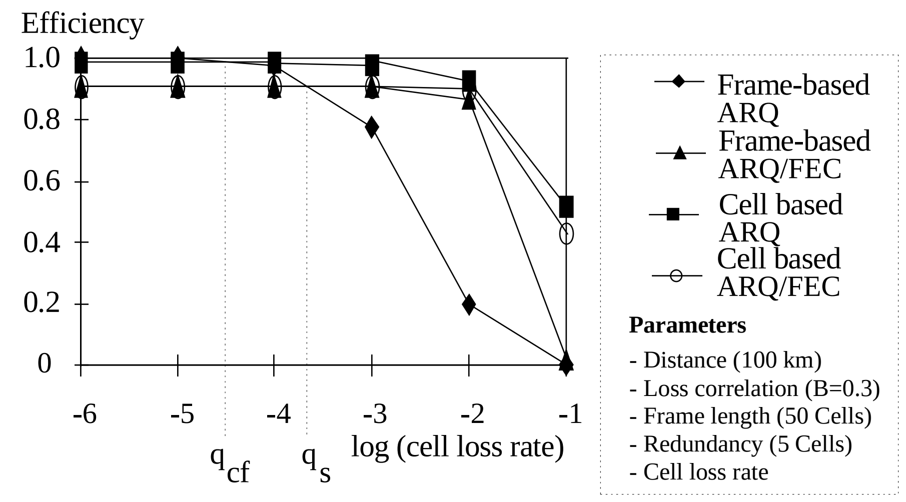

Figure 10.3: Simulation of efficiency for different error control schemes

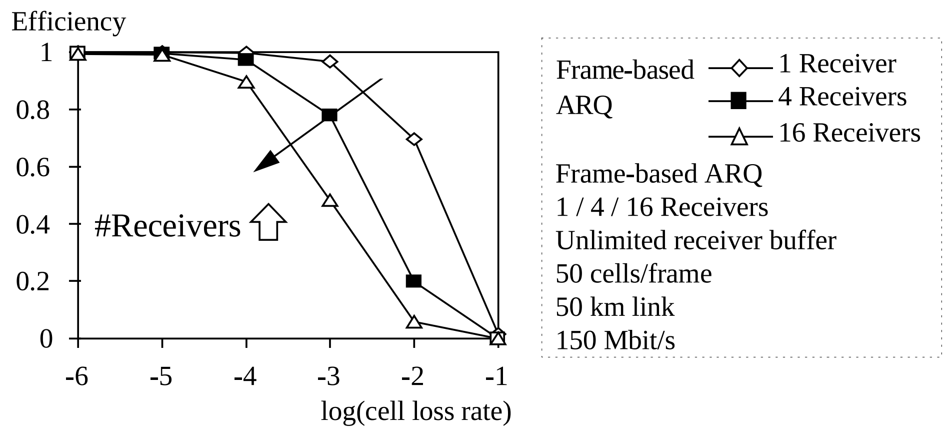

Figure 10.4: Influence of a growing number of receivers

transmitted useful frames to total number of transmitted frames) for the four schemes frame-based ARQ, frame-based ARQ/FEC, cell-based ARQ, and cell-based ARQ/FEC for varying cell loss probability.

Figure 10.4 shows how the achievable efficiency of frame-based ARQ decreases for an increasing number of receivers.[8]

In order to select an appropriate error control mechanism, the following question is of interest: up to which cell loss probability does a frame-based ARQ scheme result in higher efficiency than a framebased hybrid ARQ scheme? An interpolation of the simulation results shows a cell loss probability q_s of approximately $\log(q_s) = - 3.4$. A similar threshold q_{cf} exists for the efficiency equilibrium of the cell-based and the frame-based

ARQ. An analytical treatment of these questions can be found in G. Carle.[10]

10.3. IMPLEMENTATION

For the provision of high performance communication services, not only suitable protocols but also high performance implementations are required. Traditional implementation by hand is error prone due to the high concurrency of different processes forming the protocol. Therefore, it is much better to derive an implementation of a protocol at least semi-automatic. But this is only practicable, if the synthesis tools are powerful enough to produce implementations that fulfill the performance requirements. The following section shows some design steps performed with commercial tools in combination with tools we developed ourselves, resulting in high-performance implementations.

10.3.1. Design Flow

A significant amount of research analyzes the automatic derivation of a high-performance communication subsystem from a formal specification.[22,20] Figure 10.5 shows this goal embedded into several additional steps, representing the design flow of CHIMPSY (communication-oriented high-performance modular processing system).[26]
The *specification* mentioned here consists of the *protocol* itself and a set of *configuration parameters*. These parameters comprise the chosen integration alternative, technology, and interface depending on the desired performance, the existing software environment, and other non-formal values. From these parameters an *implementation framework* is derived.[4]
The configuration parameters describe the desired performance, the existing software environment, and the maximum costs of a system. Costs may be expressed in terms of processing costs or hardware complexity. We use simulation and measurement results of architectures synthesized in previous design cycles to determine the required number of processing units.
Using these parameters, an *implementation framework* is composed from a set of predefined functional units. This framework consists of the interfaces to an environment, static memory for protocol data, and a central crossbar to connect all components (see Figure 10.6). We describe all hardware components shown in Figure 10.6 using the standardized hardware description language VHDL (VHSIC Hardware Description Language[15]).

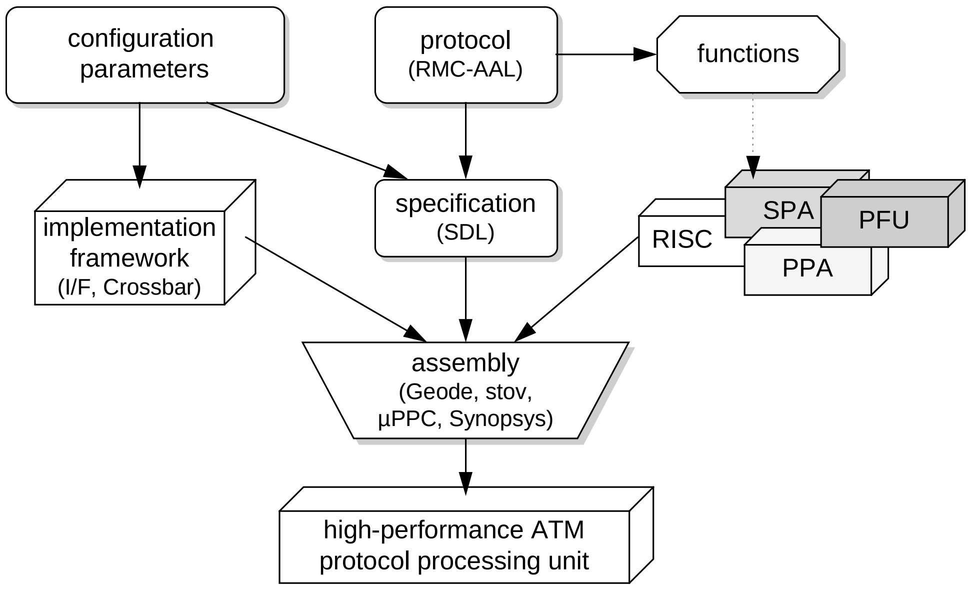
configuration parameters
protocol (RMC-AAL)
functions
implementation framework (I/F, Crossbar)
specification (SDL)
RISC
SPA
PFU
PPA
assembly (Geode, stov, µPPC, Synopsys)
high-performance ATM protocol processing unit

Figure 10.5: Customized design flow for high-performance ATM protocol processing units

10.3.2. Architectures for Implementation of Functional Units

From *protocols* like RMC-AAL we have extracted several *functions*, e.g., timer, CRC, FEC, transmit, and acknowledgement processing. These functions can be implemented on four different alternative architectures depending on the desired performance:

- *RISC-processors*: The tool GEODE[33] can be used to generate C-code from an SDL-specification. A RISC-processor can execute this code after compilation. Descriptions of different RISC-processors are available for example as VHDL-code or gate-level schematic.

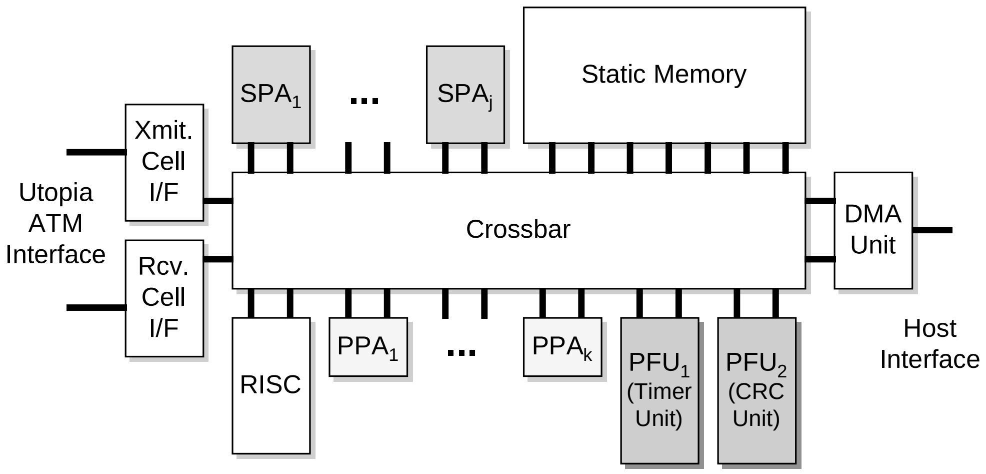

Figure 10.6: Flexible architecture for high-performance ATM protocol processing

- *Synthesizable Protocol Automata (SPA)*: With the help of a customized SDL-to-VHDL-compiler we can automatically map an SDL-description of a protocol automaton onto a VHDL-description of a hardware unit. After synthesis onto real hardware (e.g. with the tool Synopsys[30]) this unit acts as a protocol automaton. Due to the dedicated hardware for protocol processing, the performance of such a unit may be significantly higher compared to a general purpose RISC-processor. However, existing hardware synthesis tools do not achieve optimal performance when synthesizing netlists from high-level VHDL descriptions.
- *Programmable Protocol Automata (PPA)*: For even higher performance we have designed microprogrammable automata. These automata consist of only 2895 standard cells in CMOS-technology and can run with 100 MHz. Up to now, we program these units directly with microcode using a custom microcode compiler μPPC (microprogram protocol compiler).
- *Protocol Function Units (PFU)*: To achieve highest performance, we implement time-critical protocol functions as hardware macros. Examples are timer, CRC, and FEC units. Gate-level VHDL is used to implement PFUs.

Depending on the specification, the different units are chosen and configured to assemble the *high-performance protocol processing unit*. Currently, we are using 0.7μm standard-cell technology for layout synthesis and, alternatively, FPGA-boards inserted into workstations for rapid prototyping.

10.3.3. Design Tools

Our design flow comprises several tools as shown in Figure 10.5. Up to now it is not possible to find a single tool for the whole design flow that is flexible enough for the different requirements and that produces communication components with the required performance. The following items give a short overview of the tools used in our approach, and summarizes their advantages and disadvantages.

- *SDL-to-C compiler (GEODE code generator)*: GEODE[33] is a commercial tool set for the design of event-driven real time systems, using the language SDL'88,[18] and Message Sequence Charts (MSC[19]) for formal protocol specification. The tool set provides support for graphical editing, simulation, debugging, and C-code generation. Both the graphical form of SDL called SDL/GR and the textual phrase form called SDL/PR are supported. SDL specifications are logically composed of a hierarchy of structural objects. It can be selected how the

GEODE code generator maps the SDL objects process, process instance, block, and system onto operating system processes. Specific functionalities which are specified as abstract data types can be mapped onto separately specified C functions. In our flexible design approach, we also map abstract data types onto specific hardware functions implemented in Protocol Function Units.

- *SDL-TO-VHDL compiler (stov)*: In order to facilitate the process of hardware implementation of SDL specifications we developed a dedicated SDL-TO-VHDL compiler called *stov*. The compiler generates VHDL code that is adapted for the flexible architecture shown in Figure 10.6. The generated code makes use of the existing VHDL libraries that describe the architecture. This allows for rapid prototyping of protocol processing units after successful simulation of the SDL specification. As there are some SDL constructs that cannot be translated into hardware descriptions, an appropriate subset of SDL is supported by the compiler.
- *VHDL compiler (Synopsys)*: Based on VHDL-descriptions of hardware on the register-transfer level, this commercial tool synthesizes netlists for different technologies.[30] These netlists can be used for further synthesis on ASICs or FPGAs. Compared to hand-coded netlists, this tool does not achieve the optimum speed and size of hardware due to the complexity of the synthesis. On the other hand, using such a powerful synthesis system is the only way to manage the complexity of large hardware systems. In addition to synthesis, this tool also allows for simulation and debugging of VHDL-descriptions.
- *Microcode compiler (μPPC)*: Our custom microcode compiler allows for easy programming of the PPAs using a simple assembly level language. The language comprises 19 operations, comments, labels, and macros.

  The compiler converts this microcode into a binary format which can be downloaded to the PPA. The disadvantage of this microcode is its low level language. Therefore, an additional SDL-to-Microcode-Compiler is under development.

10.3.4. Protocol Implementation

Several non time-critical finite state machines (FSM) of a protocol can be mapped onto a single PPA in the implementation architecture. Table 10.1 shows some lines from the specification of a retransmission FSM. The table comprises the actual *state*, incoming *events*, and *conditions* to check before *actions* and the transition into the *next state* are performed.

State	Event	Conditions	Actions	Next State
NULL	i.new_context		init_context(ARR)	ACTIVE
ACTIVE	i.rec_closing		signal(TI, i.rec_seq)	ACTIVE
GAPTEST		gap_no = 0	s.base = s.offset ...	ACTIVE

Table 10.1: Example lines from the retransmission FSM

The shaded line shows that the FSM issues the signal *i.rec_seq* to the FSM *TI* in the state *ACTIVE* as soon as the event *i.rec_closing* has been received. The next state is also *ACTIVE*. The translation of this line into microcode for the implemented FSM is shown in Table 10.2. The first three lines wait for receiving a signal via the input port and branch to the appropriate piece of microcode after receiving the signal. The last five lines send the signal *i.rec_seq* to the FSM *TI*, save the new state (*ACTIVE*), and jump back to the beginning to wait for the next signal.

10.3.5. Synthesis

As one example for a resulting component, Figure 10.7 shows the layout of a PPA synthesized using Synopsys[30] and Cadence[7] together with the 0.7 µm CMOS standard-cell library from ES2.[11] Due to a critical path of 9.6 ns of the control logic this chip can run with approximately 100 MHz. This means that every 10ns one row of microcode from Table 10.2 can be executed. We implemented together with the control logic 11 kbyte of SRAM, with 9.5 kbyte for microcode (approx. 1750 rows) and the remaining 1.5 kbyte for registers and stacks. The chip area is 52.5 mm² with a utilization of 78%. This size is mainly due to the SRAM technology used with the prototype design.

data	**selection**	**command**	**control**	**comment**
		`CNT`	`GETH`	wait for signal
`move in,base`		`CNT`	`MOVE S,A`	load context
		`BRV`	`MOVE H1,A`	jump to sub-routine
`1`		`CNT`	`MOVE S,C`	signal i_rec_seq to TI
`TI;` `i.rec_seq`		`CNT`	`SGNL`	
`OFFSET;` `move mem,out`		`CNT`	`MOVE A,Q`	
`ACTIVE`		`CNT`	`SAVE`	next state
`label`		`JMP`	`SLEEP`	return

Table 10.2: Microcode example of a FSM suited for a PPA

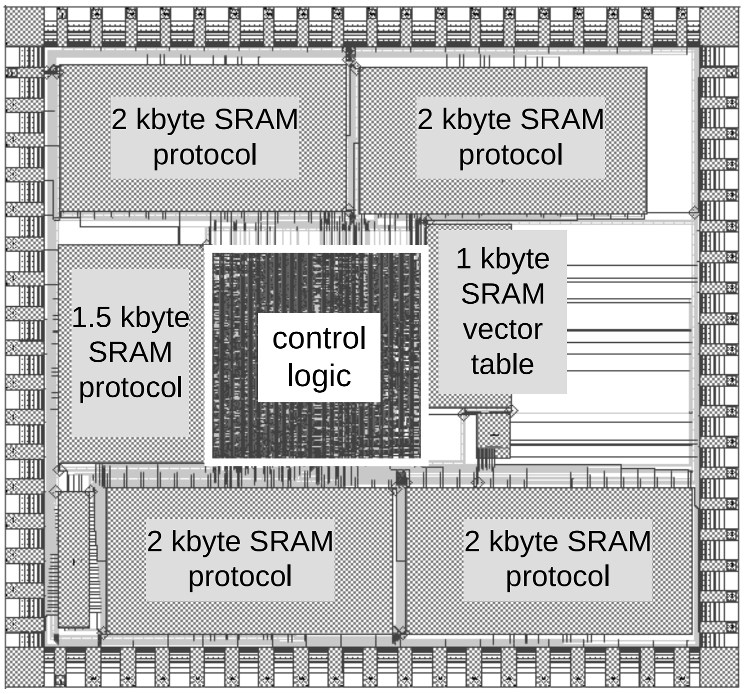

Figure 10.7: VLSI chip layout of the programmable protocol automata

10.3.6. Performance Evaluation

In order to study processing delay and implementation complexity, the prototype implementation of a GCS on a network adapter with the following properties was investigated: Protocol processing is performed on one or more 32 bit RISC processors with an average performance of 100 MIPS. In addition, hardware support for segmentation and reassembly, hardware for CRC32 calculation, and hardware for FEC processing were assumed. Based on a specification of the processes of a GCS in assembly-level language, the number of instruction cycles necessary to perform the required functionality was determined for each process of the GCS. Subsequently, the delay of a process that is executed on a processor with 100 MIPS was evaluated.

In Figure 10.8, the processing delays are compared to the cell interarrival time of 2.74 µs for an ATM link of 155 Mbit/s. The figure shows the processing delays for GCS configurations of increasing complexity from left to right. The configuration with the lowest processing delays is a GCS with frame-based ARQ. The figure also shows the additional processing delays of a GCS with cell-based ARQ, and with additional FEC. The right edges of the bars indicate the processing delays in each component if cell-based ARQ, FEC, and multiplexing are performed. The figure illustrates several properties. First, the receiver process and the transmitter process have a constant delay independent of any ARQ or FEC processing. Second, frame-level multiplexing does not take much time in any component other than the Send Manager which does the scheduling of the frames. Note that the GCS does not provide a copy function for multicasting. This copy function for multicasting is provided by an ATM switch. Third, and most important, it can be seen that the delay is dominated by the Frame Manager Receive whenever the first cell of a frame of a connection with cell-based ARQ and FEC is processed. In this module, the processing delay of the first cell of a frame is 2.71 µs when cell-based ARQ is selected in combination with FEC. The processing delay of a cell in the middle of a frame is 1.31 µs, while the processing delay of the last cell of a frame is 1.47 µs. Thus, this component is the first candidate for optimization, and for the deployment of hardware components for processing support.

A single processor of 100 MIPS leads to a processing bottleneck at high loads, as the overall delay for processing of a cell by the GCS (summarizing the processing times of all modules) is larger than the cell interarrival time. A set of three processors with support of dedicated hardware to perform table lookup, filtering, and the construction of outgoing cells allows for maximum load with an ATM link of 155 Mbit/s even for frames consisting of a single cell. Not shown in the diagram are queuing delays of cells that have to wait because frames of other senders in the same group have to be sent first. Furthermore, operations caused by the processing of acknowledgements in a GCS or in a sending host are not contained in the diagram. The latter operations heavily depend on the number of receivers that acknowledge the reception of frames or cells, and on the acknowledgement strategy (e.g., NAKs might be sent as soon as possible after detection of an error, as opposed to ACKs which are sent cumulatively).

This performance evaluation shows clearly that for higher performance dedicated VLSI has to be used. With the assumed RISC processors and only moderate hardware support only a bandwidth of 155 Mbit/s can be achieved. For implementation in intermediate systems one has to use the above described hardware components.

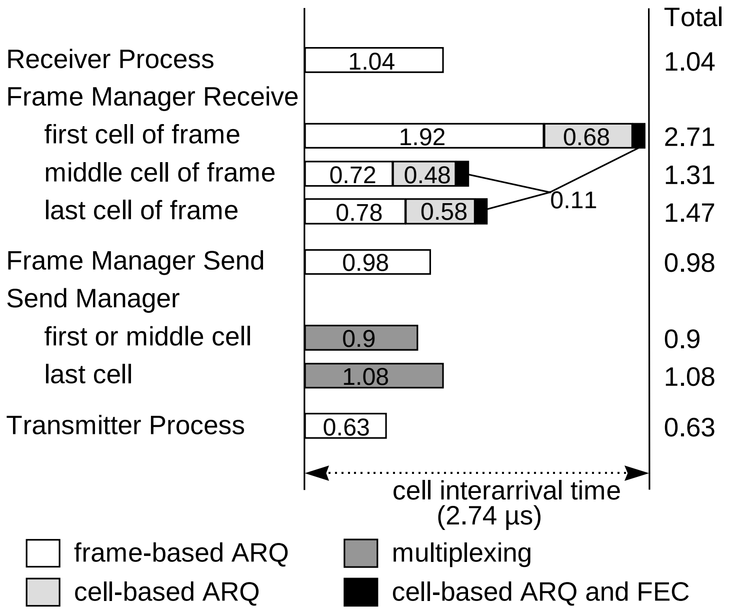

Figure 10.8: Processing delays for error control

10.3.7. Summary of the Design Flow for Protocol Implementation

Figure 10.9 shows a simplified diagram of the design flow used for the hardware implementation of high-performance communication protocols. Cycles in the design flow have been omitted for clarity. The two basic technologies used in our design are CMOS ASICs and FPGAs. Therefore, the figure shows the synthesis and derivation of parameters for these two technologies in more detail. Furthermore, we use FPGA-boards inserted in standard workstations for rapid prototyping.[34]

10.4. CONCLUSION & FURTHER WORK

A new framework for multipoint error control is presented which has the potential to fulfill many requirements. For small groups and low cell loss rates, a frame-based end-to-end error control is most appropriate. Cell-based retransmission as well as FEC allow high-performance reliable multicasting even for significant cell loss rates. For better scalability and support of heterogeneous scenarios, the deployment of a new network element called the Group Communication Server (GCS) is proposed. It allows an hierarchical approach for multicast error control and the combination of different error control schemes.

Investigation of the processing delay demonstrated the feasibility of the proposed error control schemes even for very high speeds. It also revealed that cell-based error control schemes contribute little to the processing load for error-free transmissions.

A basic idea of our work is the semi-automatic implementation of high-performance protocols based on formal specifications. This design process allows for rapid prototyping, and gaining valuable insight into the tradeoff of protocol performance and protocol processing costs. With this design flow, even processing-intensive cell-based algorithms can be implemented at high speeds, using a high-performance parallel protocol architecture.

Based on communication requirements, e.g. group communication in the context of multimedia applications and ATM networks, we develop communication protocols and describe them using formal description techniques. One example for this is the standardized language SDL. Together with a generic implementation architecture we can now derive a specific implementation. The generic implementation architecture consists of several basic components with different flexibility and performance. Depending on the protocol and performance requirements these basic components are combined and configured. We have developed several tools to support the different design steps, e.g. the mapping of SDL specifications onto VHDL hardware description. In addition, for the last design steps towards real hardware implementation we use commercial design tools.

Future work will concentrate on the design of more complex units to support protocol processing and the implementation of these units in workstations as prototypes. Furthermore, additional design tools for high-performance communication subsystems are under development.

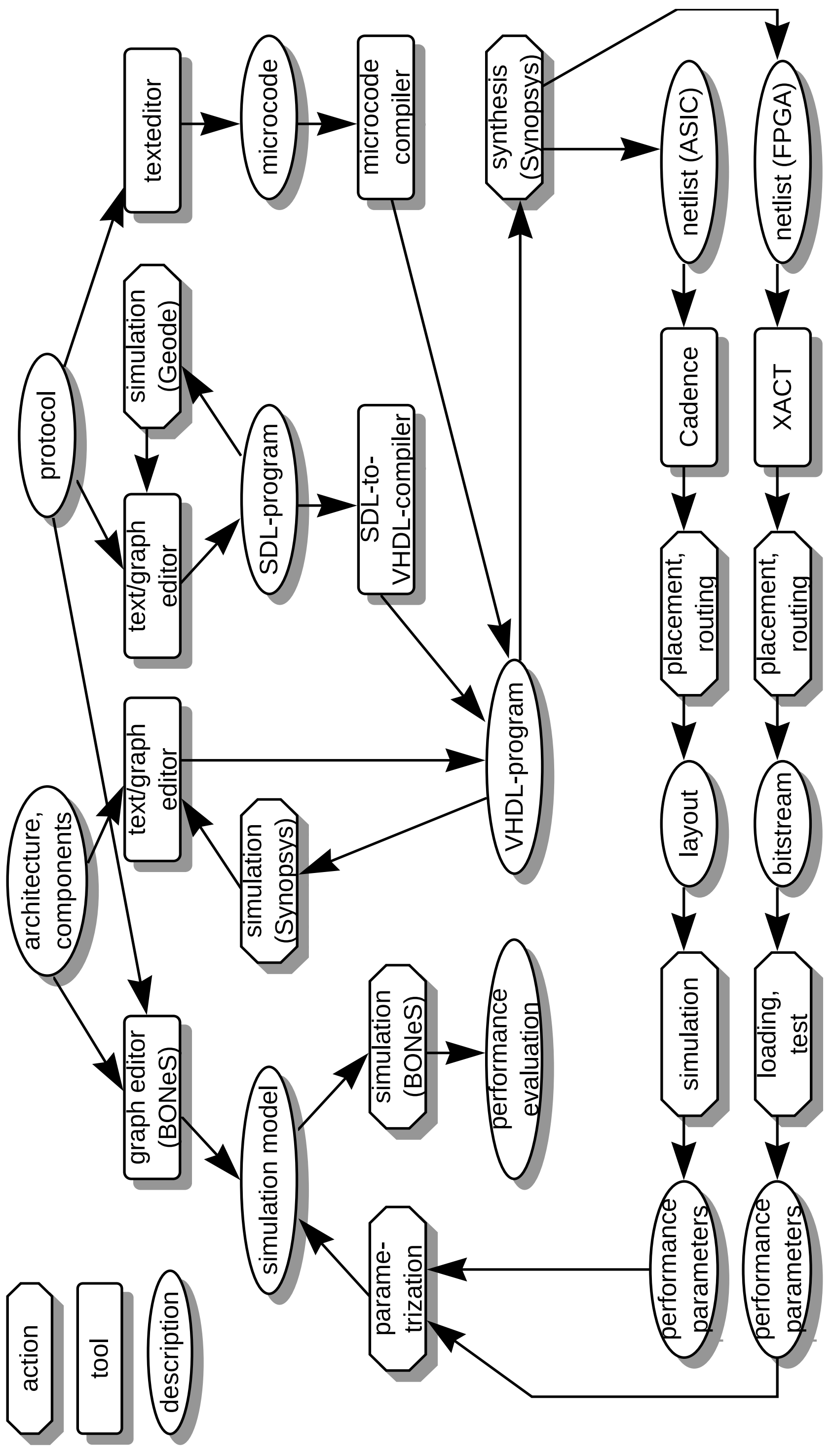

Figure 10.9: Simplified implementation design flow

REFERENCES

1. ATM FORUM, LAN Emulation Over ATM: Draft Specification, LAN Emulation Sub-working Group, ATM Forum Technical Committee, August 1995
2. BALRAJ, T.; YEMINI, Y., „Putting the Transport Layer on VLSI - the PROMPT Protocol Chip," in: Pehrson, B.; Gunningberg, P.; Pink, S. (eds.): Protocols for High-Speed Networks, III, 1992, North-Holland, pp. 19-34
3. BIERSACK, E. W., „Performance Evaluation of Forward Error Correction in an ATM Environment," IEEE Journal on Selected Areas in Communication, **11**, 4, pp. 631-640, (1993)
4. BRAUN, T.; SCHILLER, J.; ZITTERBART, M., „A Highly Modular VLSI Implementation Architecture for Parallel Transport Protocols," IFIP 4th International Workshop on Protocols for High-Speed Networks, Vancouver, Canada, August 1994
5. BRAUN, T.; ZITTERBART, M., „Parallel Transport System Design," in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 397-412
6. BUBENIK, R.; GADDIS, M.; DEHART, J., „Communicating with virtual paths and virtual channels," Proceedings of the 11th INFOCOM'92, pp. 1035 - 1042, Florence, Italy, May 1992
7. CADENCE DESIGN SYSTEMS, INC., Documentation for DFW II (Design Framework II), Cadence Design Systems, Inc., San Jose, California, 1994
8. CARLE, G., SCHILLER, J., „Enabling High-Bandwidth Applications by High-Performance Multicast Transfer Protocol Processing," 6th IFIP Conference on Performance of Computer Networks, Istanbul, Turkey, October 23-26, 1995, in S. Fdida, R. Onvural (Eds.): Data Communications and their Performance, Chapman&Hall 1996, pp. 82-96
9. CARLE, G., ZITTERBART, M., „ATM Adaptation Layer and Group Communication Servers for High-Performance Multipoint Services," 7th IEEE Workshop on Local and Metropolitan Area Networks, pp. 98-106, March 26-29, 1995, Duck Key, Marathon, Florida, USA
10. CARLE, G., „Towards Scalable Error Control for Reliable Multicast Services in ATM Networks," 12th International Conference on Computer Communication, ICCC'95, Seoul, Korea, August 20-25, 1995

11. EUROPEAN SILICON STRUCTURES, Documentation for 0.7µm-Library, European Silicon Structures, Rousset, France
12. FELDMEIER, D.C., „An Overview of the TP++ Transport Protocol," in: Tantawy A.N. (ed.): High Performance Communication, (Kluwer Academic Publishers, 1994)
13. GOLDSTEIN, F., „Compatibility of BLINKBLT with the ATM Adaptation Layer," ANSI Technical Subcommitee T1S1.5/90-009, Raleigh, NC, USA, Feb. 1990
14. HEINRICHS, B.; JAKOBS, K.; CARONE, A., „High performance transfer services to support multimedia group communications," Computer Communications, **16**, 9, (1993)
15. IEEE, Standard VHDL Language Reference Manual, IEEE Std 1076-1987
16. ITO, M.; TAKEUCHI, L.; NEUFELD, G., „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 1992
17. ITU-T, Recommendation I.363, BISDN ATM Adaptation Layer (AAL) Specification, Geneva, 1993
18. ITU-T, Recommendation Z.100: Functional Specification and Description Language (SDL), Telecommunication Standardization Sector of ITU, Geneva, 1988
19. ITU-T, Recommendation Z.120: Message Sequence Chart (MSC), Telecommunication Standardization Sector of ITU, Geneva, 1993
20. KRISHNAKUMAR, A.S., „A Synthesis System for Communication Protocols," Proceedings of the 5th Annual IEEE International ASIC Conference and Exhibit, Rochester, New York, September 1992
21. KRISHNAKUMAR, A.S.; KNEUER, J.G.; SHAW, A.J., HIPOD, „An Architecture for High-Speed Protocol Implementations," in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, (North-Holland, 1993), pp. 383-396
22. KRISHNAKUMAR, A.S.; KRISHNAMURTHY B.; SABNANI, K., „Translation of Formal Protocol Specifications to VLSI Designs," Protocol Specification, Testing and Verification, VII, Elsevier Science Publishers B.V., (North-Holland, 1987), pp. 375-390
23. MCAULEY, A., „Reliable Broadband Communication Using a Burst Erasure Correcting Code," ACM SIGCOMM '90, Philadelphia, PA, USA., Sep. 1990

24. OHTA, H.; KITAMI, T., „A Cell Loss Recovery Method Using FEC in ATM Networks,“ IEEE Journal on Selected Areas in Communications, **9**, 9, (1991), pp.1471-1483
25. SANTOSO, H.; FDIDA, S., „Transport Layer Multicast: An Enhancement for XTP Bucket Error Control,“ in: Danthine, A.; Spaniol, O. (eds.): High Performance Networking, IV, IFIP, (North-Holland, 1993)
26. SCHILLER, J., „CHIMPSY - a Modular Processor-System for High-Performance Communication,“ 1. GI/SI Jahrestagung, Zurich, September 1995
27. SHACHAM, N.; MCKENNY, P., „Packet recovery in high-speed networks using coding,“ in Proceedings of IEEE INFOCOM '90, San Francisco, CA, pp. 124-131, June 1990
28. STERBENZ, J.P.G.; PARULKAR, G.M., „AXON Host-Network Interface Architecture for Gigabit Communications,“ in: Johnson, M. J. (ed.): Protocols for High-Speed Networks, II, (North-Holland, 1991), pp. 211-236
29. STRAYER, W.T.; DEMPSEY, B.J.; WEAVER, A.C., XTP: The Xpress Transfer Protocol, (Addison-Wesley, 1992
30. SYNOPSYS INC., Documentation of Simulator, Design Compiler, and Design Analyzer, Version 3.2a, Synopsys, Inc., Mountain View, California, USA, 1995
31. THE ALTA GROUP, Block Oriented Network Simulator^{TM} (BONeS^{TM}) User’s Guide, The Alta Group of Cadence Design Systems, Inc., USA, 1994
32. THE XTP FORUM, XTP Protocol Definition Proposed Revision 4.0, 1994
33. VERILOG SA, Technical documentation of the GEODE toolset, Verilog SA, Toulouse, France
34. VIRTUAL COMPUTER CORPORATION, EVC1s Technical Reference, Virtual Computer Corporation, Reseda, Kalifornien, USA, May 1995
35. WATERS, A. G., „Multicast Provision for High Speed Networks,“ 4th IFIP Conference on High Performance Networking HPN'92, Liège, Belgium, December 1992
36. WEI, L.; LIAW, F.; ESTRIN, D.; ROMANOW, A., LYON, T.: „Analysis of a Resequencer Model for Multicast over ATM Networks,“ 3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video, San Diego, CA, USA., 1992