Reprint: 31st Annual IEEE Hawaii International Conference on System Sciences, HICCS, 1998

Semi-automated Design of High-Performance Communication Subsystems

Jochen H. Schiller
Institute of Telematics, University of Karlsruhe,
Karlsruhe, Germany
j.schiller@ieee.org

Georg J. Carle
Institute Eurécom,
Sophia-Antipolis, France
g.carle@ieee.org

Abstract

Implementations of communication protocols are typically based on highly specialized components, e.g., VLSI chips for ATM Adaptation Layer processing. The development of these specialized implementations is very time-consuming and, thus, justified only for high volumes. For a large number of multimedia applications, it would be desirable to be able to provide quickly high-performance implementations of protocols with new functionality. Using a unifying design method and suitable design tools, this goal can be achieved today. Our solution is based on a standard language for protocol specification, a generic target architecture for protocol implementation, and customized/commercial tools for mapping of the protocol specification onto the target platform. Using this approach for a powerful ATM Adaptation Layer protocol featuring forward error correction and retransmission we demonstrate the applicability for complex protocols with stringent timing requirements. Additionally, our approach results in fairly small hardware solutions compared to common processors.

1. Introduction

Today's communication end-systems comprise a large variety of components with different properties. For example for mobile equipment often low power ASICs (Application Specific Integrated Circuits) are needed, whereas in high-end workstations the main CPU can perform most of the networking tasks. All these systems have to be inter-operable if it comes to the communication itself, i.e., using the same communication protocol and protocol mechanisms.

In order to provide high performance communication services to multimedia applications, new protocols as well as high-performance implementation architectures for the communication subsystems need to be designed. It is widely accepted (see e.g. [13], [10]) that there exists no single protocol which is able to meet these varying requirements. Therefore it can be expected that in addition to the large number of new protocols presented in recent years, even more protocols will be developed.

For new protocols or protocol mechanisms (e.g., forward error correction) to be successful, it must be possible to implement them under these varying conditions and evaluate their performance. The traditional approach is that a systems engineer implements the same mechanisms "by hand" as software, specialized ASIC, programmable hardware etc. What is needed are tools and strategies for a semi-automated derivation of implementations based on a common description. Furthermore, one should be able to simulate and evaluate the target mechanisms and protocols in advance on different platforms, which is especially important as soon as specialized hardware is involved.

This paper presents a new approach for the flexible design of hardware-supported high-performance communication subsystems. The design process allows for mapping of a formal protocol specification based on the standardized Specification and Description Language (SDL, [18]) onto a parallel, hardware-based implementation architecture. The highly modular VLSI implementation architecture designed with parametrizable 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 and protocol mechanisms. Depending on the performance requirements, the architecture comprises RISC-processors, programmable hardware, or application specific integrated circuits. If powerful general purpose CPUs can be used, the specification can also be mapped onto pure software (C-Code). To support the design process, several tools and methods have been developed and combined with commercial synthesis and design tools for hardware and software. In addition, certain properties of protocols implemented on the proposed hardware architecture can be formally verified, which is especially important for safety-critical applications. We will demonstrate the approach on protocols featuring forward error correction in the ATM environment implemented on a generic multiprocessor architecture.

The paper is organized as follows. Section 2 describes the design flow used in our approach, in the 3rd section we will demonstrate the approach on a complex example out of the high-performance communication area. After that follows a comparison with related work.

2. Design flow

The overall design flow of our approach works as follows (Figure 1). Basis for our work is the communication protocol or set of protocols and descriptions of implementation architectures and components as explained in section 2.1 in more detail. The protocol specification is done in SDL [18] with a commercial SDL-tool [30]. This allows for early simulation of the behavior and results in a standardized description of the protocol. The SDL specification of the protocol is developed using the top-down method as described in [33]. Several SDL processes are derived as result of step-wise partitioning of the protocol functionality. The status information, states and procedures of the processes are specified in SDL. Performance-critical parts at this level of description (e.g., complex data structures) can be described using C or C++ integrated into the SDL specification. The simulation is based directly on SDL, timing diagrams can be generated and simple validation checks can be applied. For safety-critical parts of the implementation these tests may not be satisfactory, since they are not a formal proof of correctness of the protocol and the implementation. Therefore, in addition to the SDL-based simulations we can formally proof the correctness of smaller parts of the architecture.

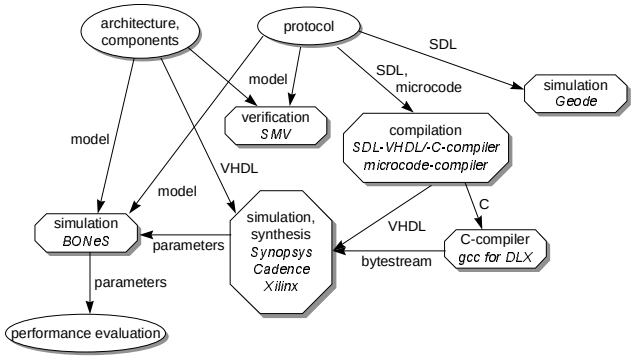

Figure 1. Overview of the design flow and tool support

For this it is very important not only to proof properties of the protocol standing alone, but to proof a certain behavior of the protocol implemented on a certain architecture. We have described several basic components of the architecture using SMV [2, 3] and are now able to integrate a protocol or parts of a protocol into this formal description. This gives us the ability to formally proof, e.g., if a certain protocol on a given architecture always performs a graceful release after an error. A very simple, but illustrative example is the following. We want to assure, that during the execution of a protocol none of the internal buffers overflow which can result in loss of data or a complete failure. Thus, we can formally proof the following condition for buffers (expressed in the syntax for the tool SMV):

```
AG !(empty & reading) &
AG !(full & writing)
```

The interpretation of this expression is the following: it should be valid on all possible execution paths that never the signal `empty` and `reading` are true at the same time, and it should be valid on all possible execution paths that never the signal `full` and `writing` are true at the same time. With other words: do not read from an empty buffer, do not write into a full buffer.

The next main step towards an implementation is the translation of the SDL specification into either VHDL [15, 16] or C. Additionally, we have developed a microcode compiler for microcode that is tailored to communication protocols. Such microcode can be directly executed on special protocol automata. The decision between translating to C or to VHDL is based on the performance requirements. We developed a generic target architecture for protocol processing which contains a single or several processing units, and which allows to execute one or several SDL processes on each processing unit. Different processing units are available for software-based or hardware-based execution of the SDL processes. While the same protocol-specific SDL-specification can be used for all generic implementation platforms, the appropriate SDL interface specifications must be selected and adapted.

Choosing C for process implementation requires the additional step of compiling the C-code into a bytestream for the DLX-RISC processor we use as generic RISC processor in our implementation [19, 25]. Different descriptions of the processor (e.g., using VHDL) and a GNU C-compiler are publicly available. Using other processors than the DLX requires rewriting of the interfaces between the processor and the internal interconnect as described in 2.1.

The following simulation and synthesis steps are now based on the standardized hardware description language VHDL. For the simulation all components are put together into the desired architecture, the protocol automata are configured via the VHDL-code generated by the SDL/VHDL-compiler or the microcode loaded, respectively. If RISC processors are used, we integrate also their VHDL model and configure them with the bytestream generated by the C-compiler. This allows for very

detailed hardware simulation of the whole implementation including a preliminary timing. Due to the standardized language chosen several powerful commercial tools are available for this step [5, 28]. Depending on the performance requirements these combined descriptions can now be synthesized onto programmable hardware (e.g., FPGA [32, 31]) or full-custom VLSI chips (e.g., 0.7 µm CMOS [12]). Certainly, already available standard components only have to be configured (e.g., programs for RISC processors). This final synthesis of the system provides us additionally with detailed timing information for every part of a communication protocol. Based on these parameters and more general models of the architecture, the components, and the protocol, very powerful simulation models can be built. For this purpose we use a discrete event simulator (BONeS, [29]) that allows for simulation of whole networks, systems, and generation of a large variety of different traffic patterns. Now finally, after putting all this together, the performance of the whole system under a simulated load can be evaluated.

2.1 Components

The following explains the main categories of components used in the approach in some more detail. All components are combined using a generic target architecture for protocol processing, called GAPPU (Generic ATM Protocol Processing Unit). Figure 2 shows an overview of GAPPU, already configured according to an example explained in section 3. This configuration shows an alternative using RISC processors and some special hardware support.

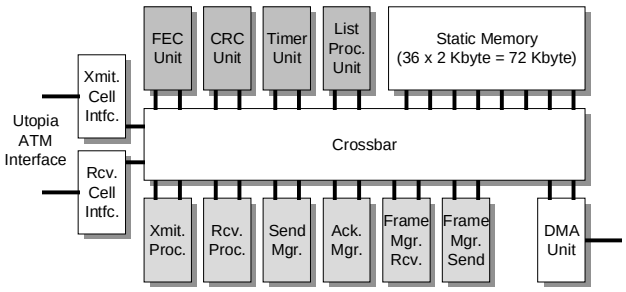

Figure 2. Architecture of GAPPU (Generic ATM Protocol Processing Unit)

- *Crossbar switch:* This component provides two separated unidirectional connections to every component inside the GAPPU and to the interfaces to the environment. The peak performance of the current implementation is 1.2 Gbit/s per port. Fair sharing mechanisms and a priority scheme for the access are integrated.
- *Protocol Function Units (PFU*): Figure 2 shows 4 examples of so called PFUs, a timer, a CRC unit, a FEC unit, and a unit for list processing. One of the main features of this approach is the ability to provide complex and time-critical functions by dedicated hardware. These components also have the same interfaces as all other components and are connected to the crossbar switch. Logically, they act as very specialized ALUs for the other components. In addition to the four components designed so far one can also think of e.g. encryption/decryption or compression/decompression units, i.e., very time-consuming functions that can up to now not be executed in pure software for real-time applications.
- *RISC processors:* The lower half of Figure 2 shows 6 RISC processors (shaded blocks) for the execution of different protocol state machines. These are exactly the RISC processors as described in [19, 25], the only change are the interfaces to the crossbar switch. These interfaces can include local memory and a command FIFO. If available, the local memory is used to store the program executed by the RISC-processor. Otherwise, this program can also be stored in the common memory. Again, this decision trades cost for performance. The FIFO command buffer is needed to decouple the execution from different components. In the SDL process model, different processes work independently and communicate only via signal exchange. All signals for a processor are now stored in this FIFO buffer and executed one after another. Thus, the semantics of a SDL description is directly reflected in the implementation architecture.
- *Synthesizable Protocol Automata (SPA):* For a complete hardware realization of a protocol automaton a SPA as shown in Figure 3 can be used. The execution and control unit is directly derived from the SDL description using StoV. In addition to the architecture shown, a FIFO buffer (input queue) and the interface to the crossbar switch is needed.
- *Programmable Protocol Automata (PPA):* If the use of a microprogrammable protocol automaton is the right choice in terms of speed and area requirements, the architecture shown in Figure 4 can be used. Providing identical interfaces as a SPA, a PPA downloads the microcode into internal RAM and starts autonomously the processing of incoming events.
- *Memory:* Depending on the performance needed the design flow allows for the integration of local memory (typ. SRAM) or the access of host memory via a DMA engine. Typically, all programs executed on the adapter and some context data of communication connections should be stored on the adapter, whereas most of the user data and connection information remains in the host memory.

- *Interfaces:* In our example shown in Figure 2 we have configured UTOPIA interfaces as standardized by the ATM-Forum towards the network. Generally, this could be any interface, only the access method has to be configured and adapted to the crossbar switch. The same is valid for the host interface, on this side typically a DMA engine adapted to e.g. a PCI bus or SCI interconnect is used.

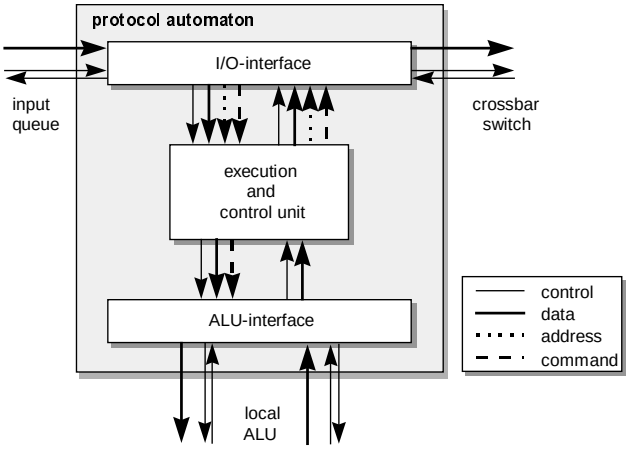

Figure 3. Implementation architecture of a protocol automaton for the use by StoV

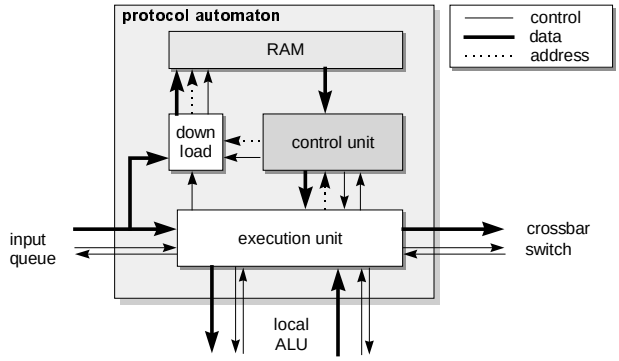

Figure 4. Implementation architecture of a micro-programmable protocol automaton

2.2 Design alternatives

Based on the more general design flow described above several alternatives can be realized. These alternatives typically trade cost against performance. The architecture allows, e.g., for different widths of data paths, different sizes and locations of memory, or multiplexing of certain units to be used by several other components. The choice of the width for data paths is directly reflected in the size of the internal interconnect. The size of the crossbar connect we use for performance reasons grows linearly in the width of data paths and in the number of connected components. If changing the size from the standard 32 bit, the interfaces between components and interconnect have to be adapted due to the fact that not all components can work with arbitrary data sizes. As most protocol specifications do not require communication of all components with all other components, a crossbar with limited connectivity can be used. For the example of the protocol RMC-AAL (c.f. section 3.1), only 30% of all possible communication paths need to be supported [9].

The memory used within the architecture consists of several local memories located at each component and one global memory shared by all components. Additionally, a DMA processor allows to use the memory of a host system. Obviously, this results in faster accesses or simpler implementation, respectively. The designer can chose between different configurations, i.e. using only one or two of the presented memory types. To reduce the gate count, components can be shared by several other units. The timer unit for example can be shared by several other components. This might result in access conflicts if two components want to access the timer at the same time. As the current design of the timer unit does not support prioritized requests, sharing of the timer unit is not a good approach if one component needs a high-precision timer. In such cases, a separate local timer should be used for this component.

In addition to the design of complete new systems, sub-components can be developed and integrated into off-the-shelf systems to enhance their performance. Figure 5 shows a schematic overview of an ATM-adapter integrated into a standard computer architecture. Focus of our considerations is the SAR/AAL (Segmentation and Reassembly/ATM Adaptation Layer) processor. In our design example, we have chosen the Fujitsu ALC processor [14] for AAL5 SAR/AAL processing, and an interconnection of the SAR/AAL processor with the host system via DMA.

We investigated 5 different alternatives for the integration of a component for Forward Error Correction (FEC), all based on the same protocol description (c.f. section 3). For this sub-component design we have chosen only the FEC-mechanism of the protocol for it is by far the most time-consuming part, and, thus, special hardware-support is required.

The alternatives differ in the interfaces and the way of accessing data. Alternative 1 enhances the SAR/AAL processor with capabilities for forward error correction. Therefore, the description of this component has to be synthesized together with the SAR/AAL chip to result in one component. The second alternative works as a pipeline, where the FEC component processes all data by changing, reading and including certain fields in a data

packet. For higher performance it is possible to connect the network adapter directly with a host CPU via dual ported RAM as suggested in [14]. The third alternative requires almost no hardware changes. Here, the FEC component just snoops the bus and inserts information when necessary. The fourth alternative acts like an additional processor inside the system. This is the alternative without hardware modification. This is also the alternative we use for functional and performance evaluation. The component consists in our case of a re-programmable FPGA with additional memory integrated in a standard workstation [31]. The fifth alternative acts similar to the MMX enhancements of Intel Pentium processors [17] or the VIS (Visual Instruction Set) of an UltraSPARC [27]. In both cases the functionality of a CPU was enhanced with several new instructions. In a similar way we can enhance the instruction set of a RISC processor by special commands supporting time-critical parts of the FEC algorithm.

In addition to these architectural alternatives, our approach allows also for splitting the protocol implementation between software and hardware. A minimal hardware intensive approach would realize only e.g. fast matrix operations and checksum generations in hardware, whereas all packet header generation, data transfer, and protocol processing is done in software by either the host CPU or a standard RISC CPU on the adapter. A full custom hardware approach would perform most of the activities in hardware, move data autonomously, and manage context data needed on-chip.

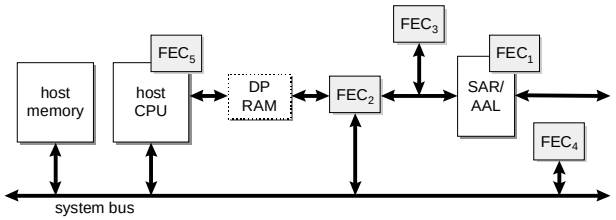

Figure 5. Alternatives of integration of a FEC component into an off-the-shelf system

2.3 Design tools

This section describes only the compilers used in our approach, we refer for the other tools (simulators, hardware synthesis, verification) that are used in our design flow to their documentation [3, 5, 28, 29, 30, 32].

- *SDL-to-C compiler (GEODE code generator):* GEODE [30] is a commercial tool set for the design of event-driven real time systems, using the language SDL, and Message Sequence Charts for formal specification [18]. The tool set provides support for graphical editing, simulation, debugging, and C code generation. Both the graphical form 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 performs mapping of 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 perform mapping of abstract data types onto specific hardware functions (e.g., list handling, priority queues).
- *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 specially suited to the flexible architecture shown in Figure 3. The execution and control unit is generated automatically. The generated code makes use of the existing VHDL libraries that describe the interfaces and the structure of the automaton. 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 (e.g., infinite buffers, dynamic process creation), 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 [28]. 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 the 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 programmable protocol automata (PPA) as shown in Figure 4 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. The following gives a small microcode example for an

automatic repeat request (ARQ) automaton that inserts information about a gap in transmitted data (`i.gap`) into a queue holding information over all gaps in transmitted data and increments the number of gaps by one. Comments in the microcode are marked with an `@`, label start with `*/`. A hardware macro available for the automaton is in this example `PutQueue`, which inserts an element into a queue. Arithmetic and logic operations and the handling of local memory is available via an ALU, therefore, the protocol automaton issues ALU commands like `inc a_r` to increment a certain register. This is a low level microprogram, thus, also the program counter has to be controlled. This is done via commands like `CNT` to continue linearly with program execution or `JMP` to jump to labels. After handling of an event like `i.gap` in this example the new state for this connection is stored (`ARQ_ACTIVE`) and the automaton waits for the next event.

```
@event i.gap
*/ARQ_ACTIVE_i.gap
@insert i.gap into the queue
/:PutQueue(ARQQueue, [inp,inp])
@store gap_no + 1
arq_gap_no mv mem a_r CNT MOVE S,A
inc a_r CNT MOVE S,A
arq_gap_no mv a_r mem CNT MOVE S,A
ARQ_ACTIVE CNT SAVE
/*LabelARQ JMP SLEEP
/*END
```

3. Design example

In the following, the application of our design method for implementing a complex and powerful ATM-specific protocol is presented.

3.1 ATM-specific protocol for reliable multicast services

When applications require a higher reliability than offered by the network, protocols with error control mechanisms must be used. For reliable data communication, protocols with Automatic Repeat Request (ARQ) mechanisms are widely used. ARQ mechanisms have several drawbacks when used in high speed wide-area networks, such as high delay in case of errors, and bad scalability for multipoint connections. An alternative approach is Forward Error Control (FEC), which has a number of advantages such as reducing delay and improving scalability for multipoint connections when used in high-speed wide-area networks. Appropriately dimensioned FEC allows to achieve an overall gain, as shown, e.g., by Biersack [Bier93]. However, FEC is not yet widely used over high speed networks, as there still remains a number of important questions such as how to dimension the amount of redundancy and the size of protocol data units. Another significant problem with FEC, which prevented its deployment in many cases, is the high consumption of processing power. Therefore, the general use of FEC for high-speed communication by processing FEC functions on a host CPU in software would frequently result in high load share of the CPU for FEC processing. As a consequence, the CPU is occupied and not able to execute applications with high speed at the same time.

There exist a number of protocols with ARQ or FEC mechanisms which are suitable for ATM networks, and which all have individual strengths and weaknesses. The adaptation layer protocol RMC-AAL (Reliable Multicast ATM Adaptation Layer) provides frame-based ARQ, cell-based ARQ and FEC mechanisms for reliable point-to-point and point-to-multipoint services [6, 7]. It is an extension of AAL5 and is suitable for a hardware-based implementation. It can be integrated into end systems and into AAL-level servers. Figure 6 shows how RMC-AAL is positioned within the B-ISDN Protocol Reference Model used for ATM networks. RMC-AAL is a protocol of the user plane. Fast execution of its protocol mechanisms is therefore crucial if high performance communication services are to be provided to the applications. As RMC-AAL is a connection-oriented protocol, less time-critical functionality is required for connection management.

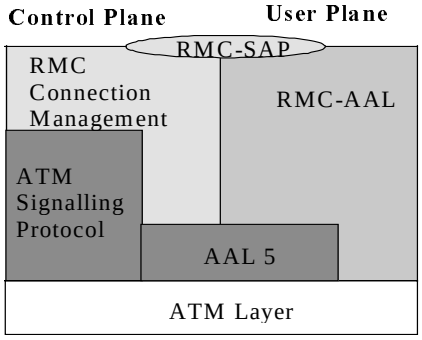

Figure 6. RMC-AAL as part of the B-ISDN Protocol Reference Model for ATM networks

This functionality is provided by the state machines of the RMC Connection Management Protocol. RMC-AAL is a special multicast protocol which can not only be used within transmitters and receivers of ATM end systems (i.e., systems attached to an ATM network which terminate ATM connections). It can also be used within dedicated servers (so-called Group Communication Servers, GCSs). Such servers allow to perform error detection prior to the receivers, and are able to collect and preprocess acknowledgments. Servers are also able to per-

form retransmissions in case of errors which occurred on links between server and receiver, and also allow to modify protocol parameters on individual sublinks [6]. Therefore, the use of servers within the network allows to improve the service quality in several cases significantly. Depending on the characteristics of the connections to be supported by a server, and depending on the network scenarios, servers are to be implemented differently. Therefore, the existence of a semi-automated design method which insures high performance results is very beneficial for the implementation of a server.

For both RMC-AAL and RMC Connection Management an SDL specification was developed. State-of-the-art SDL-to-C compilers are adequate to generate a software implementation of the connection management protocol which meets the performance requirements. However, a software implementation of RMC-AAL generated by this method does not meet the performance requirements. The semi-automatic generation of a software implementation for a user plane protocol is only useful for the purpose of protocol testing.

3.2 Formal specification of RMC-AAL

RMC-AAL has been formally specified in SDL for both end systems and GCSs. Figure 7 shows the SDL processes of RMC-AAL for a GCS that supports only frame-based ARQ. Each octagon represents a process in SDL.

For illustration purposes, the cooperation of the different processes is presented in the following. A data frame arriving from the sender enters the diagram at the upper right corner and is lead to the process FM_Receive by the process Filter_snd, which forwards frames depending on their types. The process RM_Receive allocates memory and stores the frame. The frame is then scheduled for transmission by the process Send_Manager. The process FM_Send assembles the head of the frame and passes the frame to the process Switch_rcv. This takes care that no cells of different frames are interleaved. Acknowledgments arrive at the process Filter_rcv. The process RM_Repeat interprets the acknowledgments. The results are passed to the process Pool_Manager. The core of the error control mechanism is formed by the process Frame_Control. RM_Receive starts a Frame_Control process for each data frame the retransmission of which has been requested. Acknowledgment frames are created by the process FAck_Creator. The process Switch_snd is necessary only in the case of cell-based ARQ; thus, in this specification it just forwards the acknowledgment frames generated by the process FAck_Creator.

This explanation shows the high degree of process interaction in the selected protocol. Our target architecture GAPPU supports this high degree of interaction by providing multiple support for inter-process communication: communication using hardware-support and synchronized access to local memory; communication using the global memory, and communication using dedicated addresses within the host memory.

3.3 Synthesis results

After applying all the design steps as explained in section 2, we generated two different implementations, both based on the same SDL-specification and GAPPU as generic target architecture.

- *RISC-processor based*: The first solution is based on RISC-processor and special hardware for time-critical functions as shown in Figure 2. For this configuration we synthesized 6 DLX processors, timer and list processing components, interfaces, and a crossbar switch (CBS) with full connectivity. The gate count for these components is shown in Figure 8. The complete GAPPU configured with these components uses ca. 150000 gates, and allows for a clock speed of 50 MHz using 0.7 µm CMOS (for comparison: the Intel Pentium II processor core is designed using a 0.35µm process and comprises ca. 7.5 million transistors, i.e., ca. 1.8 million gates). To avoid having a separate RISC-processor for every process shown in Figure 7, we are able to map several processes onto one processor depending on the performance requirements (c.f. section 3.4).
- *Fully synthesized protocol automata*: To compare different solutions we also synthesized every process in Figure 7 onto its own hardware component using StoV. Figure 9 shows the resulting gate count for each process. This shows clearly, that the gate count for these specialized components is less than the gate count even for a simple RISC processor as the DLX. Therefore, this is a proper solution if power consumption and high performance are design goals. The SDL specification for this synthesis consists of ca. 1700 lines of code, the generated VHDL code for hardware synthesis is 15k lines of code. Each of these full-custom protocol automata can run at a clock speed of at least 100 MHz (the internal critical paths are between 5ns and 9.5ns).

3.4 Mapping of SDL specification onto implementation architecture

A simple mapping of the 11 SDL processes shown in Figure 7 onto the implementation architecture GAPPU would require 11 protocol processing units. This simple one-to-one mapping results in a large number of proc-

essing units, and in intensive interprocess communication. The SDL processes of RMC-AAL do not have identical processing requirements. This would allow to use processing units with different processing capabilities. However, several using processing units with identical processing capabilities, and mapping SDL processes onto these processing units in a way that bottle-necks are avoided allows to reduce communication overhead, and allows to meet the processing requirements with lower hardware complexity at the same time. We searched for a mapping which results in high protocol processing performance similar to the one-to-one mapping, while reducing the implementation complexity. The solution we identified as suitable maps the 11 SDL processes of RMC-AAL onto 6 RISC-Processors, and uses additional protocol functional units (PFUs) for processing of functions involving FEC, CRC, timers and lists. The GAPPU configuration shown in Figure 2 represents this mapping result.

4. Related Work

The automatic derivation of a high-performance communication subsystem from a formal specification is already for a relatively long time subject of ongoing research. While existing methods like [20] could be successfully applied to lower-layer protocols (e.g., for protocols of the MAC sublayer), the successful application of these methods for complex, transport-level protocols has not yet been demonstrated. In addition, many existing approaches are based on specialized specification languages, such as e.g. ASPL [21] which is not supported by any other tool then the one especially developed for this language. This means that a developer cannot utilize the many person years of experience implicitly available in many commercial tools.

Table 1 shows a comparison of some of the existing approaches for communication systems with hardware support. PSi [22], [1] and ASPL [21], [20] are two approaches that deal directly with the synthesis of full custom hardware based on a formal specification.

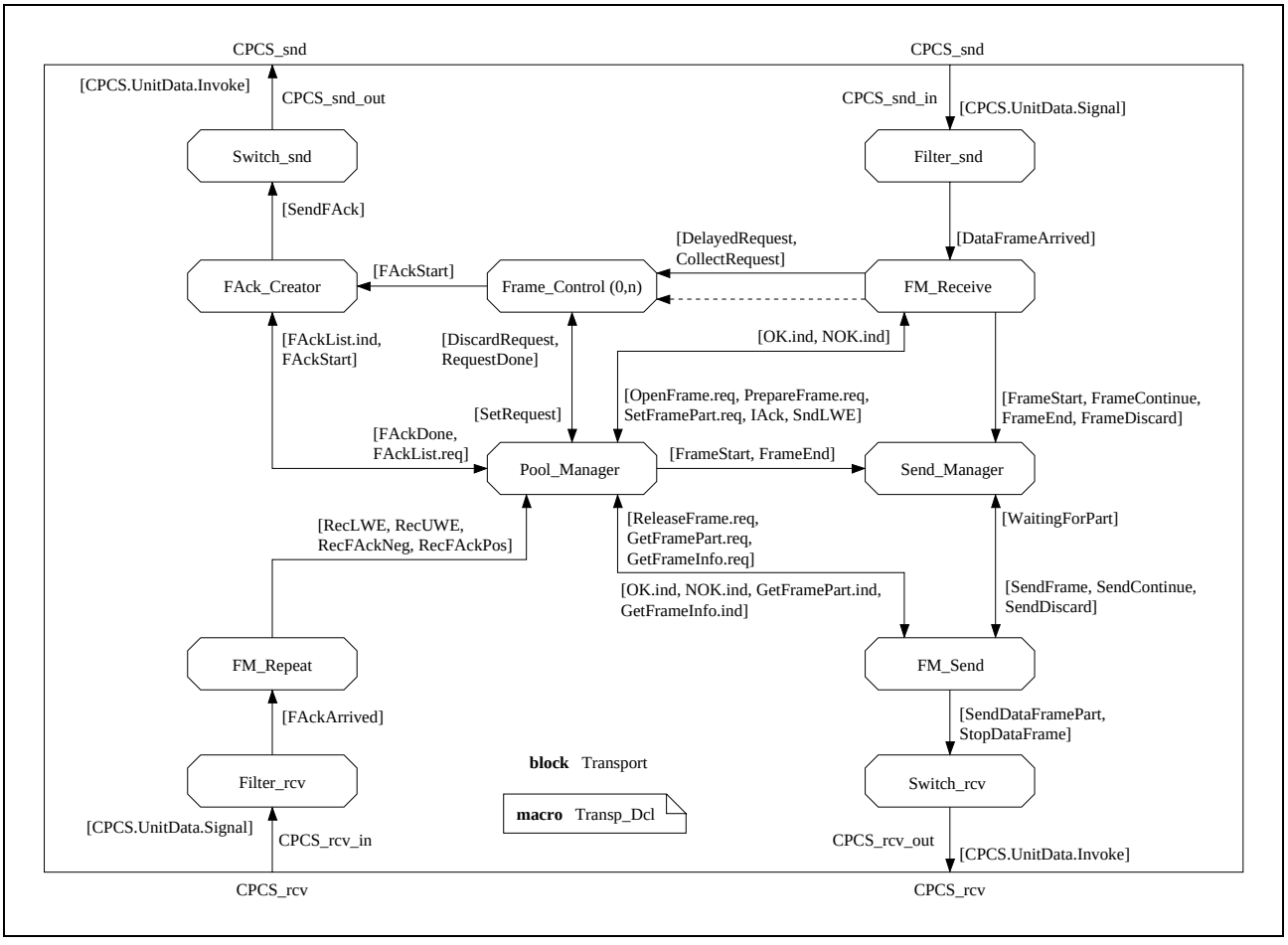

Figure 7. SDL specification of RMC-AAL for GCSs

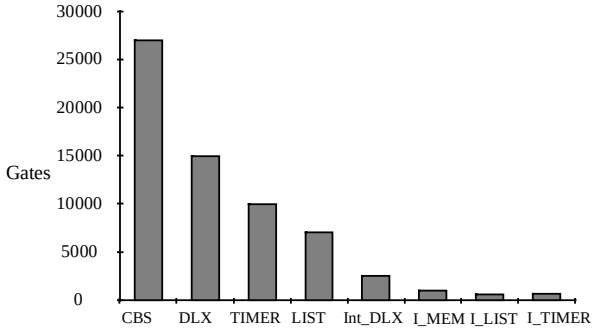

Figure 8. Gate count for the RISC-processor based solution

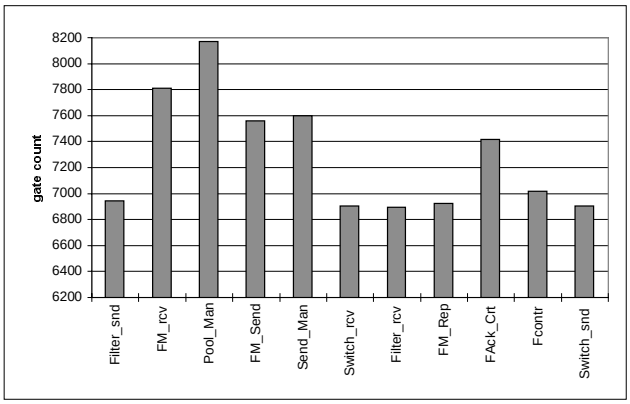

Figure 9. Gate count for the solution based on synthesized protocol automata

Both approaches are limited to lower layer protocols and lack aspects of system integration. Axon [26] and Afterburner [11] are examples of network adapters which are fully designed by hand to derive a very high performance. Therefore, they lack the support of specification or synthesis. The Washington University Broadband Terminal (WUBT, [24]) and the Desk Area Network (DAN, [23]) are two examples of complete systems, i.e., they include not only the network adapter, but also host system, storage, and operating system. Typically, those systems have a quite high performance, good system integration, but no support for formal techniques like specification or semi-automated synthesis.

Special features of the proposed approach are a very high performance, high modularity, but also the ability to support a wide range of specification, simulation, and synthesis techniques. A better system integration is subject of ongoing work. A relatively new approach to support implementations which are able to meet largely varying performance requirements on different platforms is the use of Java. The advantage of this approach is the existence of a language which allows to use a single description and implementation on many different platforms. A number of disadvantages can be identified. Java was not designed with the intention to support the implementation of communication protocols, and therefore lacks several features which can be found in protocol-specific languages. Examples are support for validation by simulation, and also verification by tools which apply strong mathematical methods for proving correctness. In addition, Java, as many other programming languages, is already close to an implementation of, e.g., processes and data structures, whereas a specification with an appropriate language gives more freedom in this respect.

Table 1. Comparison of proposed method

Name	System	Performance	Flexibility	Modularity	Specification	Synthesis
PSi	-	+	o	-	+	+
ASPL	-	o	o	o	+	o
AXON	+	+	o	o	-	o
WUBT	+	+	+	o	- -	- -
DAN	+ +	+	+	+	- -	-
Afterburner	+	+ +	+	o	- -	-
New Approach	+	+ +	+	+ +	+	+

5. Conclusions

The approach for semi-automated design of protocol components presented in this paper represents a framework which can be applied to a large number of protocols, and for a large number of implementation platforms. Furthermore, a wide range of already commercially available tools are supported allowing for extensive simulation and state-of-the-art synthesis. Suitable communication protocols can be found in the ISO/OSI communication layers 2, 3 and 4. Example network types are conventional ATM networks where high throughput and exact match of traffic characteristics are required, and also wireless cellular networks, where high error correction capability is a major concern. Example target implementation platforms range from network adapters for conventional PCs and workstations, where high protocol processing capability is a major concern, to mobile terminals where low energy consumption is of high importance.

The design approach was applied to a relatively complex multicast protocol with error control by FEC and ARQ (RMC-AAL, [8]). The synthesis results show that this approach not only supports the design of different communication systems based on one common specification (full custom, RISC processors, sub-system design), but also results in very small and efficient hardware components. Future work will concentrate on system in-

tegration aspects such as the interaction of the synthesized communication systems with operating systems and host architectures.

6. References

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

[2] Beer, I.; Ben-David, S.; Geist, D.; Gewirtzman, R.; Yoeli, M.: Methodology and System for Practical Formal Verification of Reactive Hardware, 6th International Conference on Computer Aided Verification, CAV '94, Stanford, California, USA, June 1994, pp. 182-193

[3] Burch, J. R.; Clarke, E. M.; Long, D. E.; McMillan, K. L.; Dill, D. L.: Symbolic Model Checking for Sequential Circuit Verification, Technical Report CMU-CS-93-211, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 1993

[4] Biersack, E.: *Performance Evaluation of Forward Error Correction in* an ATM Environment, IEEE Journal on Selected Areas in Communication, Volume 11, Number 4, May 1993

[5] Cadence Design Systems, Inc.: Documentation of DFW II (Design Framework II), Cadence Design Systems, Inc., San Jose, California, 1994

[6] Carle, G.; Schiller, J.: Enabling High Bandwidth Applications by High-Performance Multicast Transfer Protocol Processing, 6th IFIP Conference on Performance of Computer Networks, PCN95, Istanbul, Turkey, October 1995

[7] 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, Marathon, Florida, March 1995

[8] 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

[9] Carle, G. : Zuverlässige Gruppenkommunikationsdienste in ATM-Netzen (in German), PhD Dissertation, University of Karlsruhe, Faculty of Computer Science, December 1996

[10] Diot, C.; Dabbous, W., and Crowcroft, J.: "Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms, IEEE Journal on Selected Areas in Communications, Vol. 15, Nr. 3, April 1997, pp. 277-290

[11] Dalton, C.; Watson, G.; Banks, D.; Calamvokis, C.; Edwards, A.; Lumley, J.: Afterburner, IEEE Network, July 1993

[12] European Silicon Structures: Documentation of 1.0µm and 0.7µm-library, European Silicon Structures, Rousset, France

[13] Floyd, S.; Jacobson, V.; McCanne, S.; Liu, C.; Zhang, L.: A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing, Computer Communications Review, Vol. 25, Nr. 4, Proceedings of ACM SIGCOMM'95, Cambridge, Massachusetts, August 1995

[14] Fujitsu: ALC (MB86687A) Adaptation Layer Controller, Fujitsu,
http://www.fmi.fujitsu.com/products/network/atm.html, 1996

[15] IEEE: IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987

[16] IEEE: IEEE Standard VHDL Language Reference Manual, ANSI/IEEE Std 1076-1993

[17] Intel Corp.: MMX technology, Intel Corp., http://www.intel.com/sites/mmx/, 1997

[18] ITU-T (formerly CCITT): Functional Specification and Description Language (SDL), Recommendations Z.100-Z.104, Blue Book, October 1989

[19] Hennessy, J.L.; Patterson, D.A.: Computer Architecture a Quantitative Approach, Morgan Kaufman Pub., 2nd ed., San Francisco, California, 1996

[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. (ed.): High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 383-396

[22] Morales, F.A.; Abu-Amara, H.: *Design of a Header Processor for the PSi Implementation of the Logical Link Control Protocol in LANs*, 3rd IEEE International Symposium on High Performance Distributed Computing, San Francisco, April 1994, pp. 270-277

[23] McAuley, D.R.: *Operating System Support for the Desk Area Network*, 4th International Workshop on Networking and Operating System Support for Digital Audio and Video, NOSSDAV '93, Lancaster, U.K., LNCS 846, Springer-Verlag, November 1993

[24] Richard, W.D.; Costa, P.; Sato, K.: *The Washington University Broadband Terminal*, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, February 1993

[25] Sailer, P.M.; Kaeli, D.R.: *The DLX Instruction Set Architecture Handbook*, Morgan Kaufman Pub., San Francisco, California, 1996

[26] 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

[27] Sun Microelectronics: *The VIS Instruction Set*, Sun Microelectronics, http://www.sun.com/sparc/vis/, 1997

[28] Synopsys Inc.: *Documentation of Simulator, Design Compiler, Design Analyzer* etc., version 3.2a, Synopsys, Inc., Mountain View, California, USA, 1995

[29] Systems & Networks: *Documentation of BONeS (Block oriented Network Simulator)*, Systems & Networks, Foster City, California, USA, 1995

[30] Verilog SA: *Documentation of GEODE*, Verilog SA, Toulouse, France

[31] Virtual Computer Corporation: *EVC1s Technical Reference*, Virtual Computer Corporation, Reseda, California, USA, May 1995

[32] Xilinx: *The Programmable Logic Data Handbook*, Xilinx, Inc., San Jose, California, USA, 1994

[33] ITU-TS Recommendation Z.100 - Appendix: *Appendices I and II: SDL Methodology Guidelines, SDL Bibliography,* Telecommunication Standardization Sector of International Telecommunication Union, Geneva, Switzerland, 1988