Hardware/Software Codesign and Rapid Prototyping of Embedded Systems

Frank Slomka

Matthias Dorfel

Ralf Munzenberger

Richard Hofmann
University of Erlangen-Nuremberg

This article describes tools for the analysis, synthesis, and rapid prototyping of distributed embedded real-time systems and presents a complete design flow from specification to implementation.

THE COMPLEXITY and the short time to market of embedded systems require the use of automated techniques during the specification, implementation, and testing phases of such systems. Due to the cost requirements and the timing constraints of such systems, application-specific hardware solutions are often needed, making the codesign of hardware and software a major topic for the design automation of embedded systems.

Often, simulation tools are used for exploring the design space and for validating the functional and timing behaviors of embedded systems. During the last few years, a lot of different approaches for simulation have been developed. Hardware can now be simulated at different levels (e.g., electrical circuits, logic gates, or register-transfer level) or behavioral very high speed integrated-circuit (VHSIC) hardware description language (VHDL) descriptions. Furthermore, cosimulation environments have been formed to support the codesign of embedded systems. In some environments, software development tools can be coupled with hardware simulators, while in others, the software is executed on the simulated hardware. The latter approach is feasible only for small parts of the embedded system (like device drivers together with their related hardware). With a formal specification language like Statecharts or Specification and Description Language (SDL), it is possible to validate the functional behavior of an embedded system on a high abstraction level. But on this level, it is impossible to consider the timing behavior of the system and the interaction application-specific device drivers with the hardware.

In order to close the gap between specification and implementation and to support the codesign and validation of large embedded systems, rapid prototyping with configurable and programmable hardware/software systems allows us to validate the complete system in the

0740-7475/00/$10.00 © 2000 IEEE

real environment. Typical examples of today's complex embedded systems are ATM switching fabrics, Internet Protocol network routers with Quality of Service requirements or switching centers, and base stations for mobile communication. A prototyping system can interact with the real environment in real time, and the observed time frame for testing is larger than when using simulation.

Shortening the design cycles for such large embedded systems is the goal of our integrated framework Codesign and Rapid-Prototyping System for Applications with Real-Time Constraints (known as Corsair) that we are currently developing. Corsair supports the formal specification of embedded systems with the widely used specification languages SDL[9] and Message Sequence Chart (MSC).[10] SDL and MSC are well-supported by commercial CASE tools for the analysis, design, and verification of real-time systems. These commercial tools also support the automatic generation of software descriptions for the implementation of the system. Codesign and hardware aspects are not covered yet. These two languages are chosen for our framework because the focus of our research is the development of real-time communication systems (like base stations for mobile communication). In contrast to lower-level description languages such as C/C++, Java, or VHDL, SDL has a formal semantic, and it is possible to use standard verification techniques like deadlock or lifelock analysis on the state space. SDL and MSC have been used in the telecommunication industry during the last few decades for the specification of reactive systems. The focus of SDL is the specification of typical communication protocol automata that use signals with parameters of data to communicate.

To be able to describe the particular requirements of real-time embedded systems, we extended these specification languages to SDL* and Performance MSC (PMSC).[12] After an automatic optimization and partitioning step, the resulting specification will be synthesized to the hardware description and software implementation of the system. In contrast to former SDL codesign frameworks like Cosmos[13] and ODE,[8] Corsair supports the full development cycle from early performance evaluation (using PMSC) to the final implementation (with integrated measurement and validation on a programmable rapid-prototyping platform).

Design process

In order to reduce complexity, the design process is divided in four major steps: specification, system synthesis, implementation synthesis, and performance evaluation of the prototype (see Figure 1, next page).

Specification

During this part of the design process, the informal requirements of the analysis are transformed to a formal specification. The formal specification comprises use case descriptions given in MSC and a full functional specification in SDL. PMSC describes timing requirements together with supplemental information, such as traffic sources and resource requirements. An automated transformation of all PMSC requirements to SDL leads to an integrated codesign specification in SDL*.

System synthesis

For performing an automatic hardware/software partitioning, the system synthesis step translates the SDL* specification to an internal system model. This system model contains two graphs: a problem graph (PG) describing the functional system behavior and an architecture graph (AG) describing the prototyping platform. The PG is derived from the SDL* specification, while the AG is loaded from the Library. For each possible mapping of a problem node to a processing element of the AG, the cost and the timing behavior are estimated by the Estimator. Based on this estimation, the Optimizer searches for a hardware/software implementation of the system. The goal is to find an implementation with a minimum cost, which we discuss later in this article. After system synthesis, the resulting system model is translated back to SDL*.

Implementation synthesis

The SDL* specification is then translated into conventional implementation languages (such as VHDL for the hardware modules and C for the software parts of the system). This is

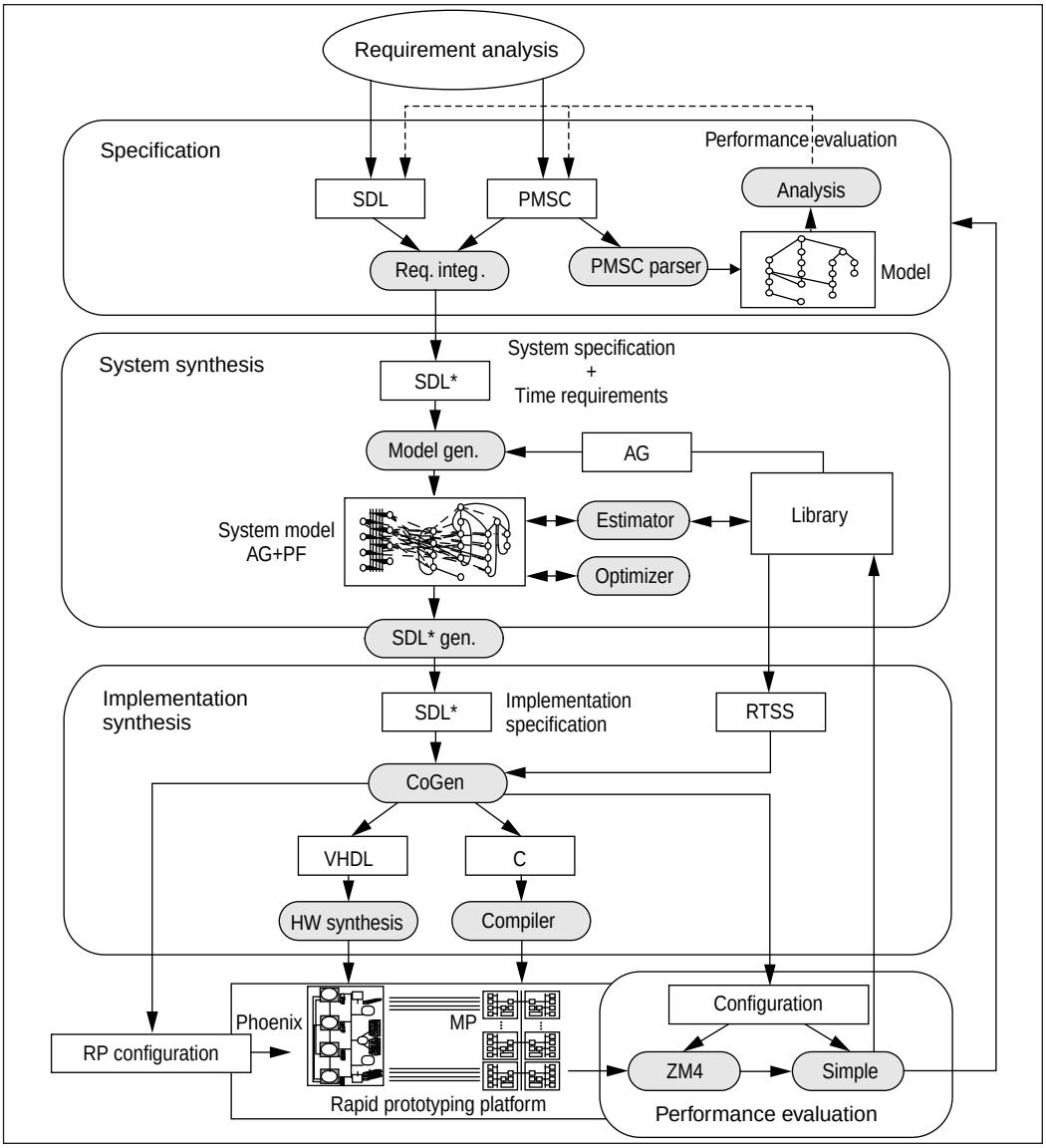


Figure 1. Design flow.

done by the implementation synthesis tool, which is called Codesign Generator (CoGen). CoGen translates the behavior of the SDL* processes into an implementation description and adds the SDL Run-Time-Support-System (RTSS) to this implementation. The RTSS supports all functions of the computational model of SDL (such as communication mechanisms and timers). The RTSS contains different hardware (VHDL) and software modules (C) stored in the framework's Library. To perform an automatic implementation of the system, the CoGen additionally generates configuration descriptions for implementation and measurement.

Prototyping

On a prototyping platform, the implementation of the system under development is executed, with the software parts running on a multiprocessor unit and the hardware parts running on a field-programmable gate array (FPGA) board known as Phoenix (Prototyping Hardware for Embedded Network Interface Accelerators). Phoenix contains four Altera FPGAs supported by large memory modules to implement the computational model of SDL and programmable clock generators to test different architectures. Another important step in Corsair is the automatic performance evaluation

of the resulting system. The data of the measurements are back-annotated to the Library to support the Estimator and the System Synthesis.

Specification techniques

The greater the complexity of a modern embedded system, the more its design requires a systematic approach based on formal description techniques. Therefore, Corsair supports the extended languages PMSC and SDL*. These languages are based on the ITU languages MSC[10] and SDL.[9]

PMSC

As a system specification represented in SDL necessarily contains all internal details, it is often difficult to see how it works. Therefore, MSC is used to display the interactions between communicating processes. The MSC use cases define the external behavior of processes and can be used to cross-check the implementation with the specification.

For real-time systems, the sequence of the interactions between processes clearly has to be obeyed, too. Timing requirements, resource requirements, and traffic sources need to be specified to deduce the performance of the system. Therefore, MSC has been extended by annotations that allow us to express these nonfunctional aspects, resulting in PMSC.

SDL

SDL allows the specification of functional aspects of systems and supports a variety of formal checks on the specification level. This helps prevent severe design errors in early design stages. In SDL, a system can be decomposed into blocks with communication via signal channels. Blocks themselves are divided into subblocks, which are further refined to processes that are specified as extended finite-state machines that send and receive signals asynchronously and that perform transitions. This asynchronous execution model implies that each process has an implicit message queue from which it receives its messages. Correspondingly, its signals to other processes go to the input queue of the destination process. Because SDL abstracts from implementation details like the scheduling of processes on processing units or limited message queue lengths, pure SDL is not feasible for an automated codesign framework for embedded systems.

To remedy this restriction, we have developed an extension to SDL that uses comments in a special syntax that our own compiler recognizes. With these annotations, SDL migrates to SDL*, which allows us to express the nonfunctional and implementation-specific aspects of system design in real-time environments.

Methodology

The development cycle starts with the formal system specification of the embedded system. During this phase, the functional and timing requirements of the system are determined. The interaction between different system components is described formally with PMSC, which also describes the estimated timing behavior of the system to perform an early performance analysis.[7] This can be a stochastic graph analysis using the analysis tool PEPP[2] or a real-time schedulability analysis, as described in Slomka et al.[14]

If the designer finds performance bottlenecks during this analysis, he or she can manually change the system's interaction diagrams. After exploring the PMSC use cases, the designer starts with the specification of the functional structure with SDL. Next, the designer specifies the complete functional behavior of the different SDL processes. After the system is fully specified in SDL, the designer starts the functional evaluation of the system. If the SDL specification does not contain more design errors (like deadlocks or livelocks), the performance requirements specified with PMSC must be inserted into the SDL specification, resulting in an SDL* specification. This transformation is performed by the Requirement-Integrator, which is based on a technique called specification-driven monitoring.[3]

Synthesis

System synthesis

System model. An embedded system's specification, given in SDL*, is translated into a PG, an extended directed control-data-flow graph (CDFG). The extensions of the CDFG are addi-

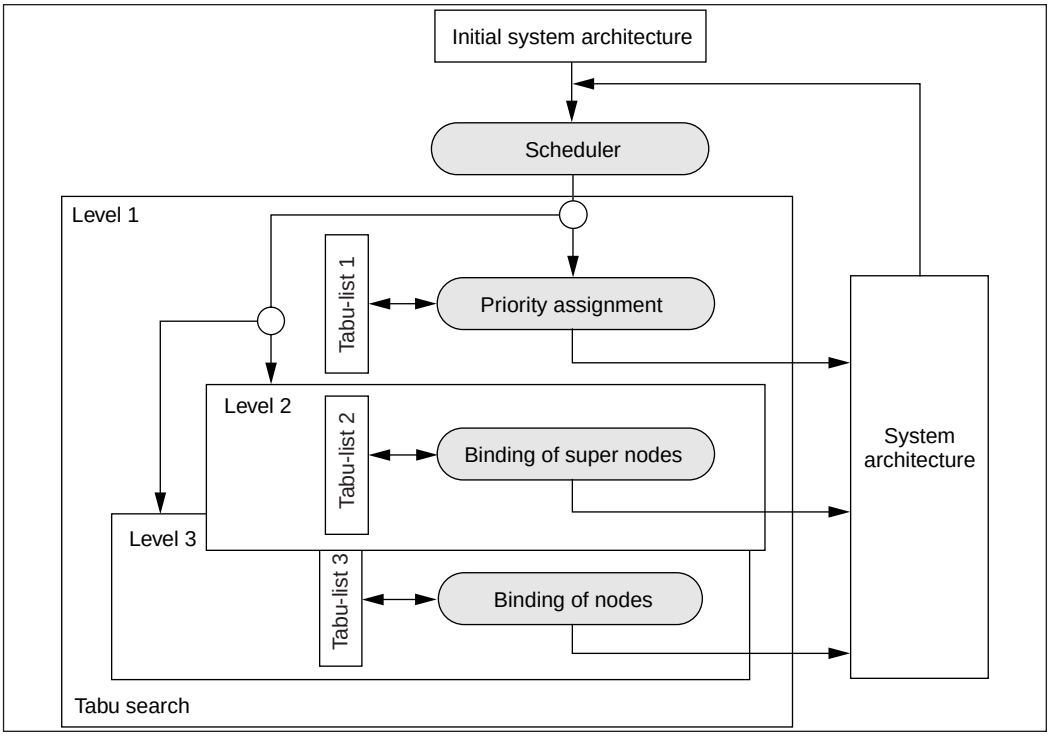


Figure 2. Three-level tabu-search algorithm.

tional node types (like sending or receiving nodes) to model the asynchronous communication model of SDL and a concept called supernodes to model the SDL processes. Each supernode is a set of CDFG nodes with the same priority related to the scheduling mechanism of the underlying RTSS.

The nodes of the PG can be mapped to processing elements (PE) of an AG, with the different types of PEs being used to model different scheduling strategies of the RTSS (preemptive, nonpreemptive, and parallel execution) and different PE classes modeling different architectures and technologies of processors. Each PG node has a set of attributes related to the different classes of the AG nodes to model the different costs of mapping or binding a PG node to different PEs (e.g., the costs of the program code and the timing behavior of the node). The overall system cost is calculated by adding the binding costs of all PG nodes to the fixed costs of all PEs that are allocated by at least one PG node.

Estimation of execution times and costs. For each node of the PG, the software and hardware execution times and the costs of the allocated architecture components are estimated. The internal graph representation contains two different types of nodes: nodes with an underlying data flow graph (DFG) and nodes that must be mapped on library components (e.g., signal queues, timer modules, and communication interfaces). The software and hardware execution times as well as the costs of elements are loaded from the Library.

For each node with a DFG, the Estimator calculates the execution time and costs with the left-edge algorithm in case of hardware mapping. This algorithm calculates the number of resources needed to guarantee the shortest latency. Based on the calculated hardware resources, the area of the application-specific circuit is estimated. In case of a software implementation of a PG node, the latency, depending on the given execution resources like registers and arithmetic logic units, is calculated.

The estimated execution times and implementation costs are further refined in steps: during implementation synthesis by Caddy-II[1] and the floorplanning tools for the FPGAs and dur-

ing the evaluation phase on the prototype by the hardware monitor ZM4 and the performance evaluation tool Simple, which we will discuss later.

Optimization. The Optimizer searches for a system architecture with a minimum cost that meets all given constraints. To perform this search, the optimizer calculates the mapping of nodes of the PG to nodes of the AG. Thus, the optimizer subsequently modifies the system architecture to improve its quality. The search for an optimized implementation of the system is based on a tabu-search method. If a solution does not meet the constraints, the Optimizer allocates new hardware components and/or changes the mapping of the PG nodes.

The evaluation of the system quality is performed using two different tools: a schedulability analysis[14] and a modified list-scheduling algorithm (see Figure 2). While tabu-search itself can be optimized if the neighborhood of moves that changes the system architecture is well-defined, each PG node will be attributed by quality analysis tools. For example, if a node or a set of nodes does not satisfy their timing constraints, the node is marked. The node with the highest number of violations is the first to be chosen for changing by the optimization heuristic.

To reduce the complexity of the problem, a three-stage heuristic for optimizing the systems architecture has been developed. The first stage optimizes the priorities of supernodes to search for a possible scheduling strategy. The second stage changes the mapping of complete supernodes to processing elements, while the third stage allows us to change the mapping of each PG node itself. Each stage of the heuristic has its own tabu-list to avoid cycles during the search.

Implementation synthesis

Software. A commercial CASE tool is used to translate SDL specifications to software implementations in C. Each SDL process is mapped to exactly one operating system process. Process communication is supported by the mechanisms of the operating system. As we describe below, the communication between software and hardware processes is performed by library components. While the commercial SDL software compiler does not support the full set of SDL constructs for such a tight integration of processes and also does not support the functionality of our partitioning algorithm, an SDL CoGen is under development. This tool will integrate hardware and software generation using a single SDL compiler.

Hardware. The hardware part of the system is translated to VHDL by the code generator SDL2VHDL.[1] For the implementation of an SDL process, there are two alternatives: In one case, the SDL process contains more than a few data operations, so behavioral VHDL is generated. The hardware architecture is synthesized by the high-level-synthesis system Caddy-II that generates an application-specific data path with a hard-wired controller. In the other case, Caddy-II would produce too much hardware overhead. For this, register transfer level VHDL descriptions generate more-efficient implementations. In addition to the VHDL descriptions of the SDL processes, the system architecture is described by a structural VHDL description that is also generated by the code generator SDL2VHDL.

RTSS. For facilitating the work of code generators, a set of library functions has been developed that forms the RTSS in hardware and software and the communication between hardware and software. The software RTSS is based on the real-time operating system RTEMS. All functions needed to implement the model of computation of SDL (e.g., receive/send or timer functions) are resolved to corresponding functions of the operating system.

To implement the hardware RTSS, a set of scalable library components has been developed.[6] Such a component encapsulates the needed functionality (like message queues and timers) and connects each SDL process with one or more communication links (e.g., global buses with different bus protocols). Using such components, it is also possible for SDL processes to share SDL functionality and memory resources. The connection between SDL processes and the library components is performed by VHDL functions belonging to the component, which are resolved by Caddy-II during the high-level synthesis step.

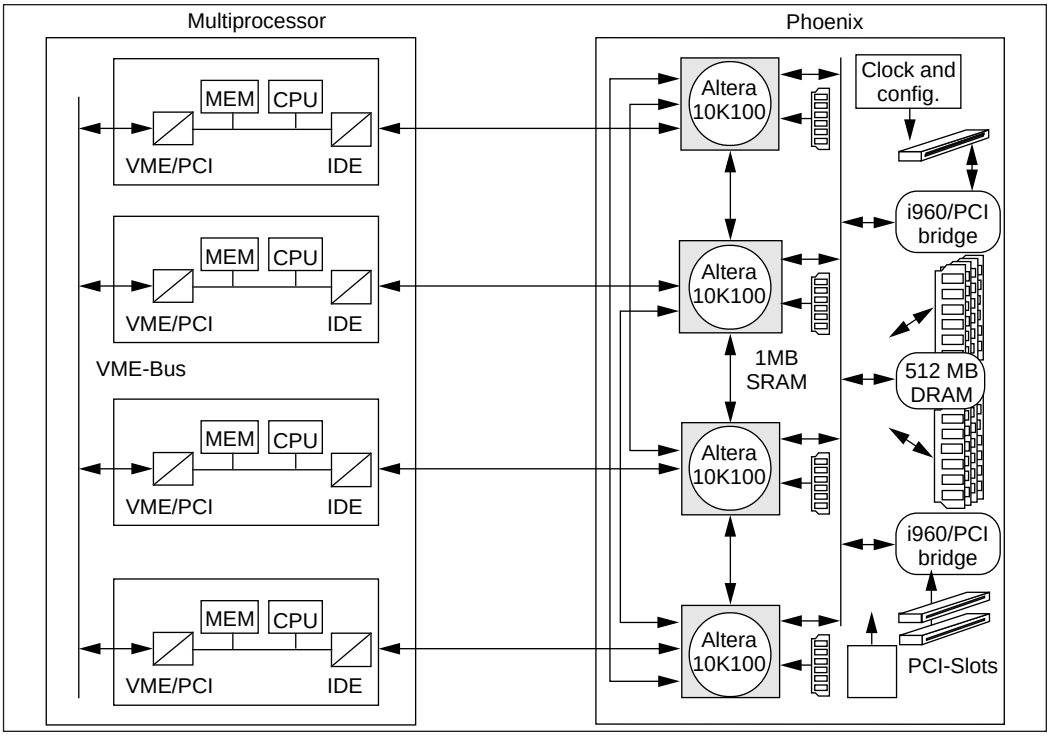


Figure 3. The prototyping platform.

Rapid prototyping environment

The main goal of the prototyping platform (see Figure 3) is the realization of large embedded systems as distributed applications. According to the optimizing model, several execution units are needed for both the software and the hardware parts. Thus, the prototyping system consists of a multiprocessor system and the FPGA board Phoenix[5] connected by four parallel links with 16 megabytes per second bandwidth per link. Both parts have been designed to implement large systems with high computational power and high-bandwidth demands.

FPGA board

The FPGA board Phoenix shown in Figure 3 contains four Altera Flex10k100 FPGAs. They are interconnected with high-bandwidth links between every two chips, which can be used to communicate between processes in different FPGAs but also to split SDL processes among multiple FPGAs. The realization as a single multilayer board gives a total communication bandwidth between the FPGAs of up to 2 gigabytes per second.

To aid the implementation of SDL systems, we added memory modules for the implementation of signal queues. Each FPGA has its own, fast SRAM of 1 megabyte size and a bandwidth of 100 megabytes per second. A second memory array of up to 512 megabytes and a bandwidth of 133 megabytes per second is connected to a global bus that is shared among all FPGAs. Phoenix supports two PCI slots for extension cards and a high-speed link to a host PC. The configuration logic, programmable clock generators, and interfaces to the hardware monitoring system ZM4 assist the automated execution of different design alternatives of the application under development.

Multiprocessor system

The multiprocessor system is based on a VME bus to connect the individual processor nodes. This lowers the cost of the total system by the use of standard processor boards while enabling the use of different processor architectures at the same time.

In the cage, a total of 10 processor nodes can be used with two currently installed. These boards are diskless industrial Pentium boards with 32 megabytes of memory, a network inter-

face for the system boot loader, and the standard interface ports. The bridge between the PCI and the VME bus offers memory access, interrupt generation and acception, and memory transfers without processor interaction (direct memory access). These features are used by the communication routines defined in the RTSS.

Performance evaluation

To support the automatic performance evaluation of the system under development and the back annotation for the refinement of the estimated execution times of the PG nodes, we used the performance evaluation environment ZM4/Simple.[11] These companion tools comprise the universal distributed hardware monitor system ZM4 and the monitor-independent trace evaluation environment Simple. Both of them have been elaborated with the goal of universal applicability, which now allows us to tailor them to this novel application area by configuration files. Based on instrumentation specifications given in SDL*, which can be performance requirements of the specification or directives of the system synthesis, the CoGen generates the configuration files for ZM4/Simple and inserts instrumentation commands into the resulting C and VHDL code of the application. This information is automatically propagated to the software and hardware parts of the system generated through the steps in the design cycle described below.

The measurement is performed by ZM4, a universal distributed hardware monitor system. In contrast to measurement tools like logic analyzers, ZM4 supports a synchronized global clock that allows the exact causal relationship of events in a distributed system. This includes the system under test and its environment. ZM4 is based on off-the-shelf personal computers that serve as monitor agents. Every monitor agent can be equipped with up to four event recorders that perform the actual monitoring task. Connected to Phoenix and the multiprocessor system by simple interfaces, the event recorders recognize the events, equip them with global time stamps of high resolution (100 ns), and store them in a 32,896-entry FIFO buffer. During a monitoring session, the monitor agent can read the measured data from the FIFO buffer, store them to the local hard disk or to a network device, or perform online evaluation.

In addition to serving as the source for instrumenting the generated system, the requirements, as denoted in the SDL* specification, are used for configuring the evaluation environment Simple. As Simple uses an abstraction layer for accessing the event trace files, it can be configured to work with arbitrary monitor systems (e.g., logic analyzers, software monitors, event-driven simulators, and ZM4).

Above the abstraction layer are several separately configurable evaluation tools. Among them are the following tools:

- the general tools Merge for merging local traces to a global trace and Glesti for estimating a global time base from local event traces
- the statistical tools Tracestat for computing overall trace statistics and Fact for determining sequences of events as described in the fact definition language
- the model-based tool Varus (validating rules checking system) that checks user-definable assertions on the trace
- the graphical tools Gantt for a flexible display of state-time diagrams and Hasse for representing causal relationships

All of these tools can be configured by a configuration file, written in the appropriate language.

Case study: multimedia terminal

For evaluating the methodology and the tools, a terminal of a real-time multimedia conference system was specified in SDL*/PMSC. Such a terminal has high computing requirements and hard real-time constraints. The Multimedia Protocol Suite (MMPS) is a scalable protocol architecture to guarantee Quality of Service requirements for delivering real-time traffic in Internet Protocol (IP) networks.

An important part of the MMPS is the Integrated Service Architecture (IntServ) developed by the Internet Engineering Task Force. The IntServ includes two sorts of services: controlled load service and guaranteed service. In order to be able to guarantee Quality of Service requirements (such as delay and bandwidth)

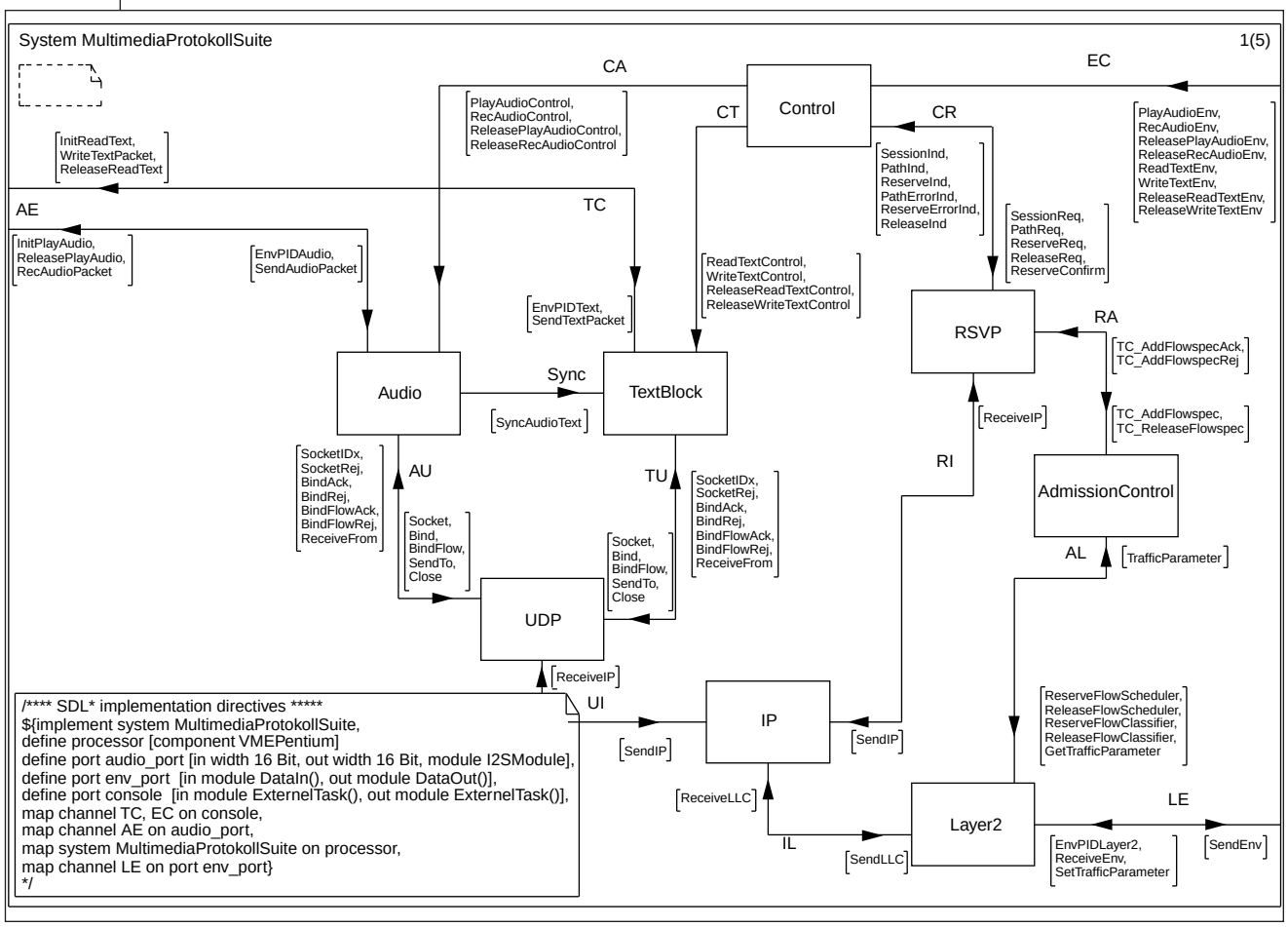


Figure 4. SDL* specification on system level of the Multimedia Protocol Suite. RSVP = Reservation Setup Protocol. RTP = Real-Time Transport Protocol. UDP = User Datagram Protocol.

in IP networks, resource reservations in each router and the terminals are required. Therefore, the MMPS includes the Internet Engineering Task Force's Resource Reservation Setup Protocol. The Real Time Transport Protocol reconstructs the ordering of packets, identifies the payload type, detects the loss of packets, and synchronizes data streams.

Usually, the Real Time Transport Protocol is used as a part of the application on top of the user datagram protocol. The interaction with the user is done by the Application Layer, which is split into parts: a part for the man–machine interface (Control), a part for coding and encoding speech data (Audio), and a part for whiteboard data (Text). After a user starts a multimedia session, Control initializes the session, including the sending of path and reservation messages. After a path in an IP network is reserved, the server terminal starts to send packets.

At first, PMSC diagrams were made in accordance with our methodology during the development of the MMPS. Therefore, in an early stage, it was possible to understand the interfaces and to avoid specification errors. Starting with PMSC diagrams, all protocols and components of the MMPS were specified in SDL*, including IP as well as the components admission control, packet classifier, and packet scheduler of the IntServ (see Figure 4). During the simulation of the SDL specification, the functional behavior of the MMPS was validated. After specifying the hardware/software partitioning by hand, the software parts of the system were translated with a commercial code generator, and the hardware parts were translated with the code generator SDL2VHDL.

Additional software modules were required to perform the communication of the SDL system and the environment. Because it is not possible to simulate either such software modules or the runtime environment at specification level, monitoring and performance evaluation of the running system are inevitable.

During the execution of the MMPS with real audio data, unacceptable audio quality was perceived from the prototype. Systematic monitoring of this system isolated a performance bottleneck that was critical for the real-time behavior, which resulted from IP messages fragmented into too-small pieces. After the performance bottleneck was fixed, the communication time on the VME backplane was one third that before the bottleneck was fixed.

An important advantage of the Corsair methodology is that specifications can be implemented on real hardware (i.e., the prototyping platform) and do not stop at the level of cosimulation. As described in Daveau et al.,[4] a complete cosimulation cycle of a full IP/ATM stack requires 120 minutes. Because of the long simulation times, it is not possible to simulate the transfer of a real-time data stream and to evaluate the audio quality of the stream. As a result, our methodology combines the strength of hardware/software codesign with rapid prototyping of embedded systems.

THE FRAMEWORK Corsair contains several tools for the automatic implementation of formal specified embedded systems. Based on the extended specification languages SDL/MSC, the methodology supports system synthesis, implementation synthesis, and performance evaluation for rapid prototyping. The framework contains standard tools (such as high-level synthesis and a software compiler) for the synthesis of hardware and software modules.

Using a configurable heterogeneous hardware platform that includes standard industrial multiprocessor cards and four FPGAs, the validation of an embedded system in real time in its real environment is possible. Specifying and implementing a multimedia terminal on the rapid-prototyping platform, we tested the methodology and the framework. The case study shows the importance of rapid prototyping in addition to formal verification and cosimulation to prove the functional and performance correctness of large embedded systems with high computational requirements, hard real-time constraints, and quality-sensitive data traffic.

Within the presented framework, the following key tools have been implemented as prototypes: The SDL2VHDL code generator is operational in a first version, the prototyping hardware is ready to use, and the system synthesis tool is still under test and tuning. Further work in the project is to perform an automatic hardware/software partitioning of the multimedia terminal and to integrate the tool set to a user-friendly framework. We also plan the specification, implementation, and evaluation of different queueing and scheduling strategies for real-time traffic in IP-based networks. ■

Acknowledgment

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG) under grant He1408/4-2 as part of the program Rapid Prototyping of Embedded Control Systems with Real-Time Constraints.

■ References

1. O. Bringmann, W. Rosenstiel, A. Muth, G. Farber, F. Slomka, and R. Hofmann, "Mixed Abstraction Level Hardware Synthesis from SDL for Rapid Prototyping," *10th IEEE Int'l Workshop on Rapid System Prototyping*, Clearwater, Florida, June 1999.
2. P. Dauphin and A. Quick, "PEPP: Performance Evaluation of Parallel Programs," 7th ITG/GI-Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, RWTH Aachen, *Short Papers and Tool Presentation,* B. Walke, ed., Aachen, Germany, Sept. 1993.
3. R. Hofmann and F. Lemmen, "Specification-Driven Monitoring of TCP/IP," *8th Euromicro Workshop on Parallel and Distributed Processing*, Rhodes, Greece, Jan. 2000.
4. J.M. Daveau, G. Marchioro, and A.J. Jerraya, "Hardware/Software Co-Design of an ATM Network Interface Card: A Case Study," *6th Int'l Workshop on Hardware/Software Codesign*, Seattle, 1998.
5. M. Dorfel and R. Hofmann, "A Prototyping System for High Performance Communication Systems,"

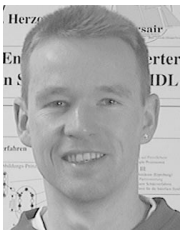
9th IEEE Int'l Workshop on Rapid System Prototyping, June 1998.

6 M. Dorfel, F. Slomka, and R. Hofmann, "A Scalable Hardware Library for the Rapid Prototyping of SDL Specifications," *10th IEEE Int'l Workshop on Rapid System Prototyping*, Clearwater, Florida, June 1999.

7. N. Faltin, L. Lambert, A. Mitschele-Thiel, and F. Slomka, "An Annotational Extension of Message Sequence Charts to Support Performance Engineering," *SDL 97, Time for Testing, SDL, MSC and Trends, Proc. 8th SDL Forum*, Elsevier Science Publishers, 1997.
8. T. Hadlich and T. Szczepanski, "The ODE-System—a SDL-Based Approach to Hardware/Software-Codesign," *Embedded Systems*, OMI, 1995.
9. ITU-T, "Z.100, Appendix I," *ITU, SDL Methodology Guidelines*. ITU, 1993.
10. ITU-T, *Z.120, Message Sequence Chart*. ITU, 1996.
11. R. Hofmann, R. Klar, B. Mohr, A. Quick, and M. Siegle, "Distributed Performance Monitoring: Methods, Tools, and Applications," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 6, June 1994.
12. A. Mitschele-Thiel and F. Slomka, "Codesign with SDL/MSC," K. Buchenrieder and A. Sedlmeier, eds., *CONSYSE 97, Int'l Workshop on Conjoint Systems Engineering*. Chicago: IT Press, 1999.
13. J. Rozenblit and K. Buchenrieder, *Codesign—Computer-Aided Software/Hardware Engineering*. Los Alamitos, Calif.: IEEE CS Press, 1995.
14. F. Slomka, J. Zant, and L. Lambert, "Schedulability Analysis of Heterogeneous Systems for Performance Message Sequence Chart," *6th Int'l Workshop on Hardware/Software Codesign*, Seattle, Mar. 1998.

Frank Slomka has been a member of the Rapid Prototyping Group at the University of Erlangen since 1996. He received the diploma degree in electrical engineering and microelectronics at the Technical University of Braunschweig in 1993. The focus of his work is the system-level synthesis of distributed embedded real-time systems.

Matthias Dorfel joined the Computer Architecture and Performance Evaluation Group at the University of Erlangen-Nuremberg in 1996. He studied computer science at the Technical University of Munich. His main interests are the design of embedded real-time systems and their performance evaluation.



Ralf Munzenberger has been a member of the Rapid Prototyping Group at the University of Erlangen since 1996. He holds a degree in electrical engineering from the University of Kaiserslautern. He works on formal specifications of distributed embedded real-time systems and their implementations.

Richard Hofmann heads the Rapid Prototyping and Performance Evaluation Group at the University of Erlangen, where he received his diploma degree in electrical engineering and his doctoral degree in computer science. His research interests include HW/SW codesign, monitoring, and parallel and distributed systems.

■ Direct comments and questions about this article to Frank Slomka, University of Erlangen-Nuremberg, Department of Computer Architecture and Performance Evaluation, Martensstr. 3, 91058 Erlangen, Germany; slomka@informatik.uni-erlangen.de.