REIHE INFORMATIK
4/96

Hardware/Software Co-design of Communication Protocols

S. Fischer, J. Wytrebowicz und S. Budkoswki
Universität Mannheim
Fakultät für Mathematik und Informatik
D-68131 Mannheim

Hardware/Software Co-design of Communication Protocols

Stefan Fischer
University of Mannheim, Praktische Informatik IV
D-68131 Mannheim, GERMANY
Phone: +49 621 292 5053, Fax: +49 621 292 5745
Email: stefis@pi4.informatik.uni-mannheim.de

Jacek Wytrebowicz
Warsaw University of Technology
ul.Nowowiejska 15/19
PL-00–665 Warszawa, POLAND
Phone: (+48 2) 660.77.15, Fax: (+48 22) 25.16.35
Email: jwt@ii.pw.edu.pl

Stanislaw Budkowski
Institut National des Télécommunications
9, rue Charles Fourier
F–91011 Evry Cedex, FRANCE
Phone: +33 1 60 76 47 20, Fax: +33 1 60 78 39 27
Email: stan@int-evry.fr

Abstract

An important aspect in providing high performance distributed systems such as multimedia systems is the combined use of hardware and software in the end systems. System design techniques should allow hardware/software co-design to integrate both means of implementation. In this paper, we show how the *standardized formal language Estelle* can be used to facilitate co-design. The system will first be designed in Estelle. At the point in time of final decision on which parts to implement in software and which in hardware, the original specification will be split into several partial specifications. The software parts are translated into C code, while the hardware parts are translated into VHDL code for further analysis and development. We present *a tool environment* which supports the protocol developer

in the design and implementation process. A simple Video-on-Demand example shows the usefulness of the tool environment.

1 Introduction

The use of both software and hardware in the realization of communication protocols seems to be a key issue in high performance networking. Some parts of a communication system are better implemented in hardware in order to achieve a higher speed (e.g. an MPEG video decoder), while other parts are preferably realized in software, thereby yielding a greater flexibility.

In the design process of such systems, *formal methods* play an important role. At the beginning, requirements of a system may be formulated and verified. Later, the abstract specification may be refined and then validated using formal test methods; simulations or even the skeleton for implementations may be generated automatically.

If, in a design process, both hardware and software parts are integrated into one specification and handled together, this is known as *hardware/software co-design*. The designer usually will not want to distinguish between such parts in the beginning; sometimes, he does not even know which parts to implement in hardware and which in software. The first, abstract specification may thus be written using only one formal method, allowing the description of architectural building blocks of a system. This will, in the further development, allow for *co-simulation*, because a precise formal semantics exists when only one technique is used. However, when moving towards implementation, it may be useful to use techniques which are better adapted for hardware resp. software. Hardware description languages like VHDL [4] allow much better handling of the hardware parts, while software may be directly implemented in programming language code like Pascal, C or C++.

In this paper, we present a tool environment which supports the design process as described above. The formal language we use for system specification is Estelle [10]. Estelle is based on the theory of extended finite state machines. It is especially well-suited for the description of communication protocols and distributed systems. Estelle's syntax is a superset of Pascal which makes it very easy to understand such specifications; the Pascal part is used to describe sequential data processing. In addition, deriving simulations and efficient implementations is easier under Estelle's operational semantics than in other formal techniques.

The road from first design steps to implementation is as follows: a protocol designer builds an Estelle specification to refine his concepts, to validate his ideas and to analyze properties of the protocol under design. The Estelle specification is useful for test generation. A protocol implementor extracts from this specification descriptions of distributed elements of the system, and divides it into the parts to be implemented in hardware and in software. (Due to the increasing demands of protocols' throughput, the hardware solutions become more and more frequently investigated.) Subsequently, the implementor

generates software and hardware descriptions of the computation and the communication units. Finally, he adds the implementation details and incorporates the implementation specifications of the designing units into the realization, through e.g., C compilers and VHDL high-level synthesis tools.

In our integrated toolset, the overall specification will be handled by the EDT environment [2], supporting specification and simulation in Estelle. To allow for separated handling of hardware and software parts once simulation is finished, the Estelle specification may be split up into several parts using the tool described in [13]. The software parts may then be generated using EDT and its so-called *implementation motor*, while the hardware parts may be translated into VHDL for further processing (e.g. automatic chip board layout), using the Estelle-to-VHDL translator *e2v* described in [24].

Related Work. Several projects currently in progress are trying to integrate both hardware and software into the same design process: COSMOS from TIMA/INPG [8], SpecSyn from Irvine [6], CODES from Siemens [1], Thomas from CMU [21], Gupta and DeMicheli from Stanford [7] and Ptolemy from Berkeley [3].

COSMOS is a hardware/software co-design environment based on the SOLAR intermediate format for system-level modeling and synthesis. SOLAR supports the system and behavioral levels of specification. COSMOS includes system-level synthesis tools for communication synthesis and partitioning as well as AMICAL - the behavioral synthesis tool for hardware. The current version of COSMOS proceeds from the SDL language and produces a heterogeneous architecture including hardware descriptions in VHDL and software descriptions in C.

SpecSyn is a system design framework that helps a designer to obtain synthesizable descriptions of implementation modules starting from an abstract system specification given in SpecCharts or VHDL languages. The main SpecSyn design tasks are: allocation of structural objects (i.e., modules and buses), partitioning of the system specification (functional objects are mapped onto the structural objects), and binding the structural objects to library and to generic components.

CODES is a design environment, which integrates a new modeling tool with several existing tools for system specification, for hardware and for software implementation. The modeling tool is based on the abstract algebraic Parallel Random Access Machines model and on an extended Petri Nets model. The input for this tool can be a Statemate or an SDL specification. The existing tools integrated into this environment are: the C and VHDL generators for StateChart and SDL, the Matrix design and simulation tool, and the SIDECON knowledge-based configuration tool for printed circuit boards. Currently, CODES targets the design of processor-based systems.

The Thomas co-synthesis approach consists of specifying a system as a set of tasks, and of allocating these tasks to processes implemented in specialized hardware and application software running on general-purpose CPUs. Thomas proposes to specify in the Verilog Hardware Description Language the processes to be implemented in hardware, and to build Unix processes for the software. Hardware and software communicate by means of

BSD Unix sockets and by a Verilog module that corresponds to an abstract bus interface. A co-simulation of a system specified in this manner can be performed using a simulator for Verilog.

Gupta and DeMicheli capture system functionalities using the hardware description language HardwareC which is the input language of the hardware synthesis system Olympus. A data flow graph derived from the system specification is used to evaluate a partition cost function.

The Ptolemy design environment supports the development and simulation of functional and behavioral models of hardware and allows the generation of assembly code for DSP microprocessors. A Ptolemy user can synthesize software, model hardware and simulate algorithms. He specifies on the Ptolemy input: a synchronous data flow structure, and functional graphics with abstraction levels from the gate level to the behavioral level. He can obtain as output: a DSP assembly code, and netlist descriptions of the hardware configuration that can be fed to logic synthesis tools.

The application domain considered by each approach depends closely on the description power of the input specification language chosen. The approaches that can be used for protocol design are those that accept a language used in protocol engineering, e.g., Estelle or SDL.

This paper is organized as follows: Section 2 identifies the implementation blocks of a protocol and relates Estelle constructs to them. Section 3 describes how to split up Estelle specifications to allow for distinct handling of software and hardware parts. Section 4 gives insights into the complete tool environment using an example, and Section 5 concludes the paper.

2 Implementation blocks of a protocol

A protocol specification defines a cooperation of communication processes. The processes are implemented as *computation units* and cooperate by means of *communication units*. A communication unit represents a communication medium together with hardware/software means of accessing the medium. In some cases, the communication units already exist within the target system, and only links between computation units and the existing communication units should be done (e.g., operations on the UNIX TCP sockets). The communication units should behave in the same way as the abstract communication models of the specification. Because we have selected the Estelle language for an abstract protocol specification, the communication models are the unbounded FIFO queue and the exported variables[1].

A computation unit may be built on the base of the ASICs, PLDs, microcoded devices, standard microprocessors widely used, and standard *computers*. When the unit is implemented on a standard computer, the link to the communication unit is implemented by

[1]The use of exported variables is very limited in Estelle; e.g., there are no shared variables between Estelle `systems`.

system calls. When the computation unit is implemented on a standard microprocessor, the link is realized as a hardware-related assembler routine. When it is implemented on micro-controlled data paths, then the link is done by a microcode routine. Finally the computation unit may be implemented entirely in hardware, in which case the link is built from RTL structures (e.g., multiplexers, registers).

Frequently, a communication unit is a complex system. In such a case, it can be implemented using a lower-level communication protocol that may be broken down into simple building elements, i.e., subprogram calls and hardware functional blocks. The implementation may be done in many different ways, and it is not possible to arbitrarily select the best one when starting from an abstract protocol specification. A design library for interactive use by a designer or a synthesis system can be created. It should be open to any user modifications and should contain generic templates and communication units, as well as descriptions of the existing and commonly used interface units, e.g. a set of TCP socket functions, drivers to Ethernet or Token Ring cards, VHDL description of 16550, 16450, 8250 UARTs etc.

An Estelle specification can describe a distributed communication system. As shown in Figure 1, an Estelle specification consists of systems, which in turn consists of modules. We assume that a single system will be implemented as a closely coupled architecture, i.e, on one machine, however two ore more systems can be physically distributed. A module behavior is represented by an extended finite state automaton. The behavior is described by a set of transitions. A transition is defined by a set of selection clauses and statements to evaluate. These clauses and statements do not precise the access mechanism to the module interface. They can send a message or receive it via a FIFO queue, or they can access an exported variable. These actions are atomic. Thus, the module's transitions describe an algorithm performed by a protocol computation unit. A communication unit is modeled by either a pair of connected Estelle interaction points together with the associated FIFO queues or by a set of Estelle's exported variables. In both cases, the Estelle specification does not give any premises about the communication unit implementation.

The decision to partition a system's functionality among interacting hardware and software must be made by a designer. Though there are some research efforts to completely automatize this task [22], we argue that the choice should be made by a designer. Most of the partitioning algorithms are based on the assumption that a system designer's goal is to implement a system using a minimal amount of application-specific hardware to satisfy required performance. But in reality, the designer considers more goals, e.g. the time required to put a product on the market, the volume of the final realization, the length of production runs (Does assembly cost exceed the design cost of an ASIC?). Moreover the price of application-specific hardware design decreases continuously. The designer does not need an HW/SW partitioning resolver, but a specification evaluator that supports some metrics about complexity and throughput of computation units.

Some metrics for an Estelle specification are already defined [16, 15]. These were developed to measure a specification's quality, to improve its modifiability, reusability and

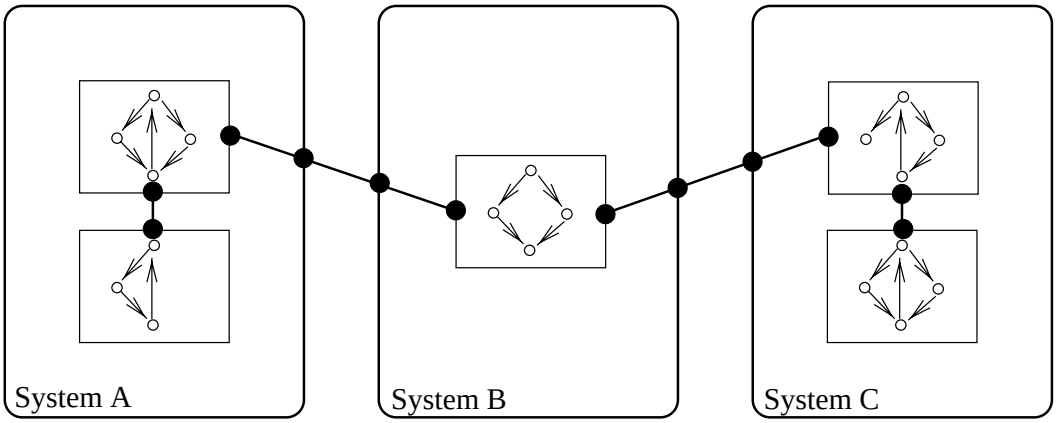

Figure 1: Example for the structure of an Estelle specification

readability. They give us an idea about particular modules' complexity. However the metrics for HW/SW partitioning of an Estelle specification are still subject to future research.

3 Distribution of an Estelle specification

For Estelle, there exist several toolsets allowing easy production of software simulations for a given system specification, e.g. [23, 17, 19]. Some of them are already headed towards implementation, e.g. by allowing distribution on several nodes of a network [12, 20] or parallelization on multiprocessors to achieve higher speed [5, 14]. With respect to co-design, system distribution is an especially important aspect.

However, currently available tools all follow the same distribution pattern: the specification is translated *as a whole* from Estelle into programming language code, e.g. C or C++. The resulting code is then distributed over the network and translated into executables on each node. This "classical" approach has the following disadvantages:

- During the code generation process, only **one** Estelle compiler can be used. That means that at every site the same type of programming language code is available, e.g. C or C++. This is usually not a problem, as on most systems compilers exist for all common programming languages. However, some code types may be not suitable for certain machines. Experience shows, for example, that it is quite difficult to port generated C++ code to transputers, due to the very huge executables produced from it. Thus, it would be desirable to be able to use different compilers at each site.

- Hardware and software co-design becomes very difficult. Partitioning of C functions between hardware and software design units and building an interface among them is a laborious and error-prone task.

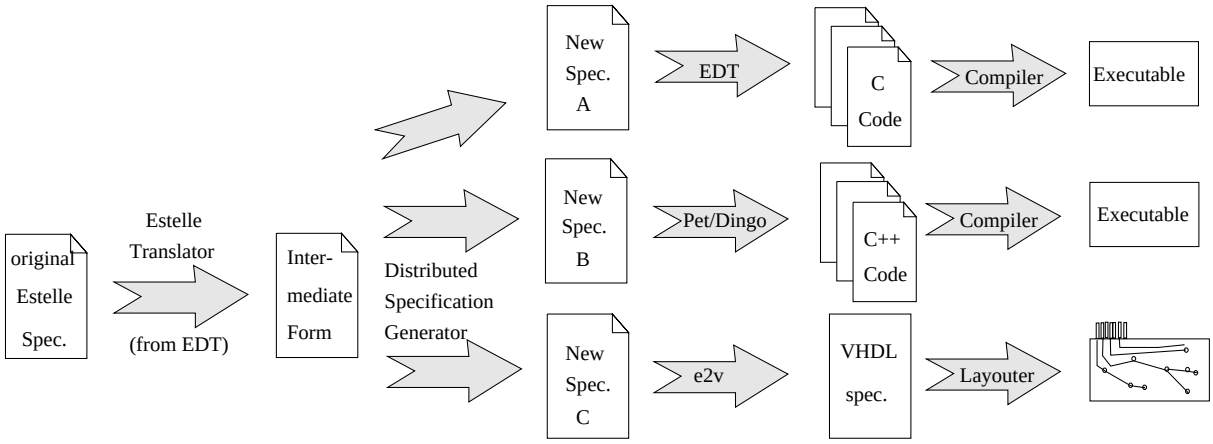

Figure 2: Splitting up Estelle specifications

- The generated C code can be easily linked with implementation-oriented functions, but it is almost illegible for a human — thus making *code refinement* very difficult.

We therefore propose a new methodology, in which not programming language code, but specifications are distributed. Instead of producing code from the original specification, we split up the latter into several partial specifications. Each of these can then be handled separately. Specifications intended to be realized in software can be transformed by one of the existing code generators into programming language code; hardware parts may be translated into hardware description techniques like VHDL and then further processed. The whole process of splitting up a specification may be seen in Figure 2.
Three main requirements on the new methodology have to be taken into account:

1. We need, on each node, a *compilable* Estelle specification. This means that the splitting is more a constructive process rather than a simple extraction of a given subsystem definition; some definitions of data types, communication channels etc. defined globally elsewhere should be included in the generated parts.

2. Each specification part must offer an *additional interface* to provide a means of communication between these parts. To ensure better performance, the traffic through these interfaces should be minimized.

3. The derived implementations have to be *executable* within the existing environments (compilers/runtime systems). This means that within the generated specification parts, any handling of the additional interface has to be specified explicitly within Estelle transitions, as otherwise, the runtime systems would have to have special knowledge of how to handle certain Estelle constructs used for the external interface.

Described next are the main design decisions and the structure of the new partial specifications; for the reasons behind those decisions as well as for a more detailed description, please have a look at [13].

To fulfill requirement (1), a new module has to be added to each partial specification to make it syntactically correct. This module is a model of a communication unit part, which is the external interface of the separated Estelle system. It is equipped with transitions (requirement (3)) describing the external interface mentioned in requirement (2). There is one transition for each possible incoming or outgoing message with respect to this partial specification. Every incoming message is immediately forwarded to the part split off from the original specification; every message outgoing from this part is sent via the external interface to the environment.

In the new transitions, some function and procedure calls are used to address the external interface. A library of their implementations has to be provided. Typically, these libraries offer a TCP socket or an RPC interface; for hardware-software interfaces, one has to provide calls to a device driver as described in Section 2.

Two important issues to be discussed are correctness and performance of the new method. Again, details on this may be found in [13]. There, we showed that the method works correctly. Some experiments and measurements with sample Estelle specifications show that performance is also very good.

4 EDT environment in a co-design process

The Estelle Development Toolset has been used for several years by more than 30 universities, research centers and industrial laboratories in protocol validation and software implementation. Its current commercial version consists of the following tools:

- Compiler (translator and C code generator with the implementation motor),
- Simulator/debugger,
- Universal Test Drivers Generator.

Some new tools under development whose prototype versions are already available include:

- Graphic Mouse-Menu Interface,
- Documentation Generator,
- Distributed Specification Generator,
- Metrics Evaluator,
- Estelle to VHDL Translator.

The enhanced EDT environment can be used in both hardware/software co-simulation and co-design. Figure 3 shows how an EDT user can put an Estelle specification into hardware or software implementation environments. Let's assume we have a complete

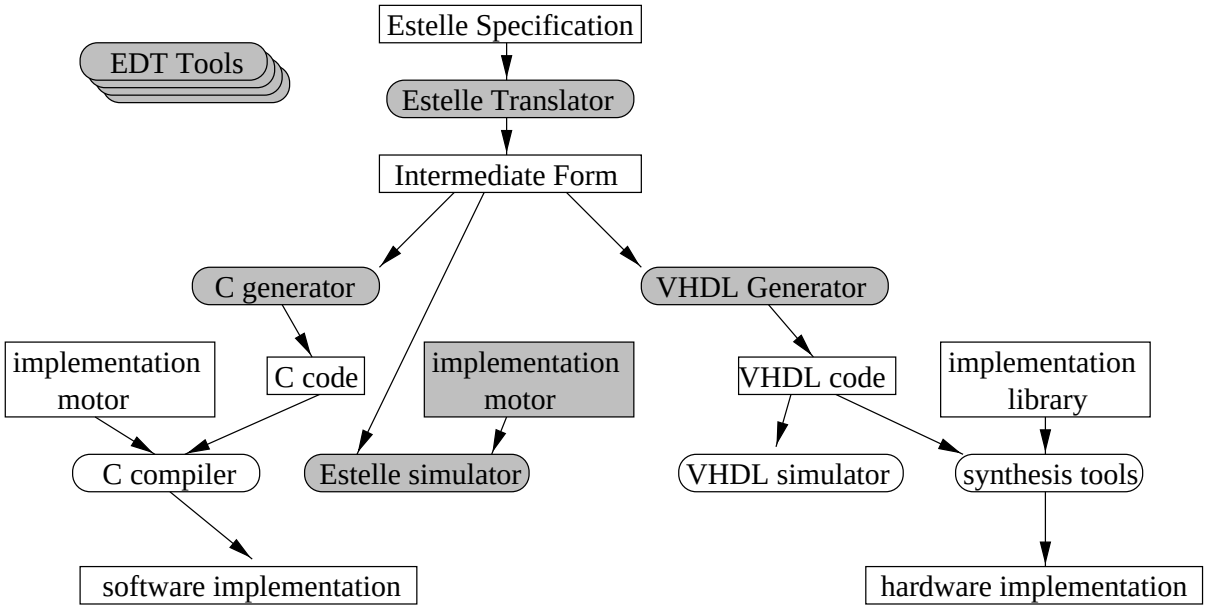

Figure 3: Working with the Estelle Development Toolset

Estelle specification of a system. Using EDT's *Estelle translator*, we generate an Intermediate Form (IF) file from an Estelle specification. The IF file is built only for syntactically correct specifications. All other tools take as their input the IF file.

In the next step, the Estelle specification of the overall system should be validated and verified with respect to designer requirements by using the *Estelle Simulator*. It helps to analyze the system's behavior, to eliminate deadlocks, livelocks, unspecified events, incompleteness of the specification and dead code errors. Test modules can be written in Estelle to either stimulate the whole system or each individual module. In addition, the *Universal Test Drivers Generator* can be used to produce test modules automatically, providing a very flexible means for interactive testing.

The designer can use the *Metrics Evaluator* to analyze the quality and complexity of the specification. Some iterations of specification refinements and simulations have to be done to develop a system specification which may enter the implementation process. The first step in this process is hardware/software partitioning. The Metrics Evaluator can be helpful at this step, but here, the application behind the protocols, with its algorithms and requirements, must also be considered. Some time analysis can be done using the Estelle Simulator facilities. For a more detailed time analysis, the designer can translate all system specifications into VHDL to profit from the VHDL time expressiveness.

When the system has been tested to the designer's satisfaction, the next step is headed toward implementation. Using the *Distributed Specification Generator* described in Section 3, the overall specification may be split up into its part, following the result gained from the tools described above.

The partial specifications will then be distributed over the network, and each part is further processed at the network site where it will later be implemented. For the software parts, EDT's *C Code Generator ec* may be used to produce C-code from the Estelle specification. Using the *Estelle-to-VHDL Translator e2v*, the hardware parts may be transformed into a more hardware-oriented language. The resulting VHDL code may then be put into a VHDL behavioral level environment, e.g. AMICAL from the TIMA Laboratory [11].

The generated C code and VHDL specification have to be refined. All subroutines that are relevant to the communication rules (and omitted in the Estelle specification or only mentioned as so-called prototype functions) have to be integrated.

To refine a generated VHDL specification much more work is needed than to refine a generated C code. The C code is ready to be compiled into object files. The VHDL specification is ready for simulation, but it must be reworked to obey the restrictions of selected synthesis tool and to fit the interfaces of computation and communication units.

Finally, the C code has to be compiled and linked with the different communication unit interfaces provided in object code libraries. In EDT, there exists one such library, offering a TCP Socket interface for interprocess communication. Interfaces for standard hardware devices are currently under development.

Chipboard layout tools may be used to produce hardware implementations. The resulting chip boards contain standard communication units as described in Section 2, for which hardware drivers are available. The communication connection between software and hardware parts is established using the device drivers and the communication interfaces provided in the libraries.

5 Example: A Simple Video-on-Demand System

To show how the tool environment works for a real-life system, we will now have a look at a simple Video-on-Demand system. In our system, movies are stored on a server disk in MPEG format [9]. Clients may connect to the server and ask for the transmission of such movies. Incoming frames at the client site have to be directed to an MPEG decoder/player which finally displays the movie on the screen. Incoming control data such as connection information is directed to the user. The overall system specification using Estelle systems and modules is shown in Figure 4.

Basically, all the functionality of the Movie Protocol module is specified in Estelle. For the server module, only the disk access has to be specified using a primitive procedure in Estelle, which has to be provided in C later. All other server functions are written in Estelle. In this example, the User module is generated using the Test Drivers Generator. As a result, we obtain an interactive test module. A human user may arbitrarily select transitions and parameters of this module to test all the system's functions.

The only module which is very rudimentarily described in Estelle, is the MPEG decoder. This is because the MPEG algorithm is very complex which makes it a hard task to

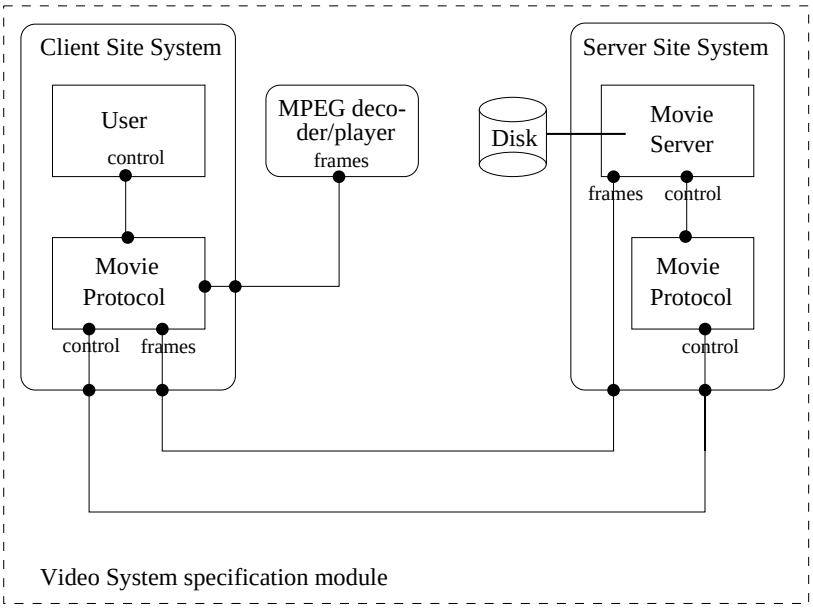

Figure 4: System Level Specification of a Simple Video-on-Demand System

specify it in a high-level language. Basically, the Estelle transitions inside the module look as follows:

```
TRANS
   when data.I-frame begin
       decode-frame;
       display-frame;
   end;
```

The procedures `decode-frame` and `display-frame` are primitive; they have to be provided by the implementor.

The testing and validation process is based on this specification. We use the EDT Simulator to test the protocol functions. It is also possible to get first impressions of the system's performance. Before advancing to the implementation process, intensive testing should be performed in order to keep mistakes away from the implementation. Possibly, a redesign of the specification is necessary.

Let us assume, that we have tested our video example, removed all faults and now start to implement the system. The first step is to partition the overall specification into its parts, using the Distributed Specification Generator. For this example, we obtain three partial specifications: one for the server system, one for the client system and one for the MPEG decoder. We then have to finally decide which parts to implement in software and which in hardware. In this example, the User, Server, and Movie Protocol module are translated into software using the C Code Generator. The Movie Protocol Module can be

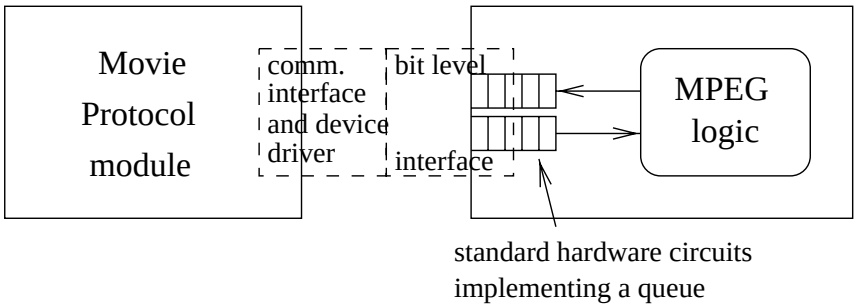

Figure 5: Connection from Movie Protocol to MPEG chip board

used as is. The C code for the User module should be refined to fulfill the needs of a real user application. In the Server module, the C functions for disk access have to be added.

The main problem is the implementation of the MPEG module. In the following, we consider two possible implementations: a pure software decoder and a hardware solution.

- **Building a software decoder**
  In the software solution, the primitive functions `decode-frame` and `display-frame` have to be filled with the corresponding C code. For our example, we do not develop a completely new MPEG software decoder. Instead, we use the existing Berkeley MPEG Player's functionality [18]. It is not a difficult task to integrate this code into the module framework code generated for the MPEG Decoder module. This decoder also includes the functions necessary to display frames in an X-Windows environment.

- **Producing an MPEG decoder chip board**
  To produce a hardware implementation, we use the Estelle-to-VHDL Translator to transform the MPEG Decoder module into VHDL code. The translator generates one MPEG entity (the computation unit) and two queue entities (the communication unit). The generated MPEG entity contains only the description of the MPEG decoder behavior, which is initially specified in Estelle. The generated queue entities represent the framework for designing a bit level communication interface, which is needed to access the hardware implementation of the MPEG logic. The building blocks existing in a VHDL design library can be taken to create the final implementation of the communication interface. To allow software parts to access the hardware, a device driver has to be written. The Distributed Specification Generator allocates the device driver calls within the Movie Protocol specification. Thus, it is easy to connect the software and hardware parts. This connection is shown in Figure 5.

  The MPEG logic, which is doing all the decoding work, is designed using a VHDL development tool as described in the previous section. These environments allow

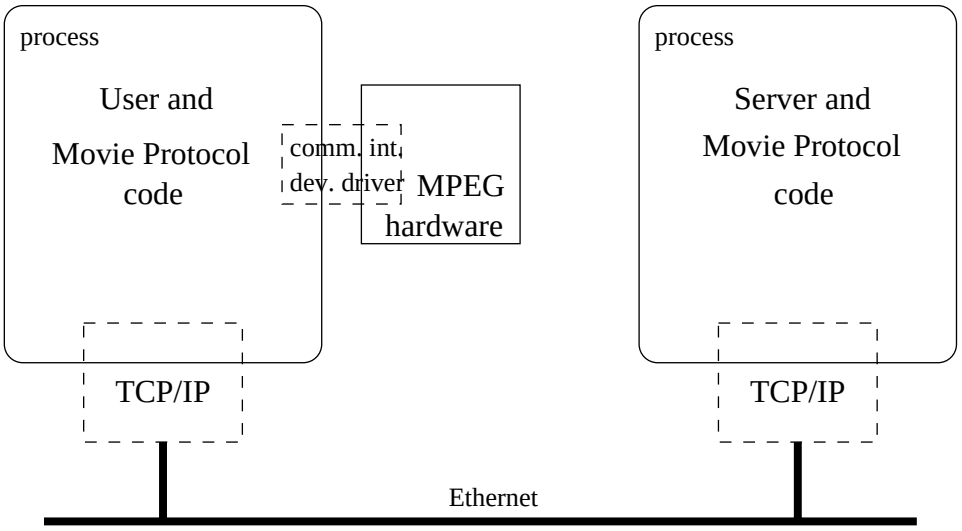

Figure 6: Implementation of the MPEG movie system

testing of the hardware before really implementing it. As a final step, the chipboard layout for the MPEG decoder may be generated from the VHDL specification.

The last step in the production of the distributed implementation is to compile and link the programs. Let us look at the hardware solution: The Movie Protocol module code has to be linked with the communication interface accessing the device driver for the standard queue circuit. It also needs an interface to TCP sockets, because it communicates with its peer. The resulting implementation is shown in Figure 6 in terms of processes, chip boards and networks.

6 Conclusions

We have presented a toolset which supports hardware/software co-design for communication protocols by using the formal language Estelle. The abstract notion of the Estelle communication mechanisms simplifies specification and analysis of such protocols. A designer can describe in Estelle a system that is composed of hardware and software blocks that communicate through unbounded FIFO queues. A hardware/software co-simulation of the protocol is based on one global specification. C code generation and derivation of a VHDL specification from a formal specification is easier and much less time-consuming than its derivation from a specification given in natural language. A formal specification already contains a structure and interprocess communication control flow. An automatic translation does not introduce additional errors and is better able to relate implementation back to the original requirements. Thus, developing a distributed system by stepping down from Estelle to C and VHDL can speed up the design process. Test modules translated into C or VHDL are useful in subsequent verification steps.

The designer can start a rapid hardware/software co-designing taking advantage of the following tools: a generator of distributed Estelle specifications, a translator from Estelle to a programming language (C, C++) and a translator from Estelle to a hardware specification language (VHDL). A library of communication units will speed up integration of the synthesized modules.

The Estelle tools presented can be a front end to these system analysis environments which have an input for VHDL specification. For example the COSMOS and SpecSyn frameworks can be useful to allocate the library and generic components for the communication and computation units and to obtain a synthesizable VHDL code.

The Distributed Specification Generator and the Estelle-to-VHDL-Translator are currently being integrated into the EDT distribution and will thus be available to the public in the near future. The existing libraries providing interfaces to communication units are extended to include more hardware and software interfaces.

References

[1] K. Buchenrieder, A. Sedlmeier, and C. Veith. HW/SW Co-Design with PRAMs Using CODES. In *Proceedings of the International Conference on Computer Hardware Description Languages (CHDL'93), Ottawa, Canada*, Apr. 1993.

[2] S. Budkowski. Estelle Development Toolset. *Computer Networks and ISDN Systems, Special Issue on FDT Concepts and Tools*, 25(1), 1992.

[3] M. Chiodo, P. Giusto, A. Jurecska, L. Lavagno, H. Hsieh, and A. Sangiovanni-Vincentelli. A Formal Specification Model for Hardware/Software Codesign. In *Handouts of the International Workshop on Hardware Software Co-design, Cambridge, Mass.*, Oct. 1993.

[4] D. Coelho. *The VHDL Handbook*. Kluwer Academic Publishers, 1989.

[5] S. Fischer and B. Hofmann. An Estelle Compiler for Multiprocessor Platforms. In R. L. Tenney, P. D. Amer, and M. Ü. Uyar, editors, *Formal Description Techniques, VI*, pages 171–186. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1994.

[6] D. Gajski, F. Vahid, and S. Narayan. A Design Methodology for System Specification Refinement. In *Proceedings of the Euro Design Automation Conference (EDAC'94), Paris, France*, Feb. 1994.

[7] R. Gupta and G. DeMicheli. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design and Test of Computers*, pages 29–41, Sept. 1993.

[8] T. B. Ismail and A. Jerraya. Synthesis Steps and Design Models for CoDesign. *IEEE Computer*, 28(2):44–52, Feb. 1995.

[9] Information technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 MBit/s (MPEG). International Standard ISO/IEC IS 11172, 1993.

[10] Information processing systems — Open Systems Interconnection — Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074, 1989.

[11] A. Jerraya, M. Aichouchi, E. Berrebi, H. Ding, P. Kission, M. Rahmouni, and P. V. Raghavan. AMICAL: Interactive Architectural Synthesis Based on VHDL. Technical report, INPG/TIMA Grenoble, France, 1994.

[12] D. Kreuz and R. Gotzhein. A Compiler for the Parallel Execution of Estelle Specifications. In H. König, editor, *Formale Methoden für Verteilte Systeme*, pages 161–178. K. G. Saur München, New Providence, London, Paris, 1993.

[13] E. Lallet, S. Fischer, and J.-F. Verdier. A New Approach for Distributing Estelle Specifications. In G. von Bochmann, R. Dssouli, and O. Rafiq, editors, *8th International Conference on Formal Description Techniques (FORTE'95)*, pages 439–448, Montréal, Kanada, 1995.

[14] R. Plato, T. Held, and H. König. PARES – A Portable Parallel Estelle Compiler. In P. Dembiński and M. Średniawa, editors, *Protocol Specification, Testing an Verification XV, Warsaw*, pages 403–418. Chapman & Hall, London, 1995.

[15] J.-L. Raffy. Mastering Complexity at Design Level. In *AOWSM'95, Portland, USA*, June 1995.

[16] J.-L. Raffy. Mésures de complexité de specifications écrites en Estelle. In *Colloque Francophone sur l'Ingenierie des Protocoles (CFIP'95), Rennes, France*, pages 497–516, May 1995. (in French).

[17] J.-L. Richard and T. Claes. A Generator of C–Code for Estelle. In M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azema, and V. Chari, editors, *The Formal Description Technique Estelle*, pages 397–420. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1989.

[18] L. A. Rowe and B. C. Smith. A Continuous Media Player. In P. V. Rangan, editor, *Network and Operating System Support for Digital Audio and Video (Third International Workshop, La Jolla, California, USA, November 1992)*, Lecture Notes in Computer Science 712, pages 376–386. Springer-Verlag, Berlin Heidelberg, 1993.

[19] D. P. Sidhu and T. P. Blumer. Semi–automatic Implementation of OSI Protocols. *Computer Networks and ISDN Systems*, 18:221–238, 1990.

[20] R. Sijelmassi and B. Strausser. The PET and DINGO tools for deriving distributed implementations from Estelle. *Computer Networks and ISDN Systems*, 25(7):841–851, 1993.

[21] D. E. Thomas, J. K. Adams, and H. Schmitt. A Model and Methodology for Hardware/Software Codesign. *IEEE Design and Test of Computers*, pages 6–15, Sept. 1993.

[22] F. Vahid, J. Gong, and D. Gajski. A Binary-Constraint Search Algorithm for Minimizing Hardware during H/S Partioning. In *Proceedings of EURO DAC'94*, pages 214–225, 1994.

[23] S. T. Vuong, A. C. Lau, and R. I. Chan. Semiautomatic Implementation of Protocols Using an Estelle–C Compiler. *IEEE Transactions on Software Engineering*, 14(3):384–393, Mar. 1988.

[24] J. Wytrebowicz and S. Budkowski. Communication Protocols Implemented in Hardware: VHDL Generation from Estelle. In J.-M. Berge, O. Levia, and J. Rouillard, editors, *Current Issues in Electronic Modeling: Languages for System Abstraction*, pages 77–98. Kluwer Academic Publishers, 1995.