Concepts for Model Compilation in Hardware/Software Codesign

S. Schulz, and J.W. Rozenblit
Dept. of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721
USA
sschulz@ece.arizona.edu
jr@ece.arizona.edu

K. Buchenrieder, and M.Mrva
Infineon Technologies AG
Otto-Hahn Ring 6
81739 Munich
Germany
klaus.buchenrieder@infineon.com
michael.mrva@infineon.com

Abstract

The model-based design process has established itself in the domain of hardware/software codesign. It has been applied to many complex embedded systems. Nevertheless we still observe a discontinuity in the design process when the system model evolves into its eventual implementation. We present here the concepts for a model compiler which addresses this model continuity problem. In addition, a candidate tool is evaluated for our proposed design approach.

1 Introduction

Recent advances in embedded systems hardware have resulted in rapid emergence of high performance products such as global positioning systems, efficient electronic fuel injection control units, portable multimedia players, or palm top computers. Stringent processing and reliability requirements as well as the need for a low cost and short time-to-market solution make a revision of the traditional design methods necessary.

Model-based approaches have been proposed to design these systems at a higher level of abstraction in order to reduce the ever increasing design complexity. We have presented one such approach, called model-based codesign (MBC) [1,11], which relies heavily on simulation modeling techniques to explore the feasibility of virtual prototyping. It provides a methodology for the design of embedded systems which fosters rapid development, late partitioning, and assessment prior to implementation as well as component reuse. A critical point in any model-based design methodology is the transition from an abstract system model to an efficient implementation. Research literature refers to this transition as the model continuity problem [9].

Current practice is to build simulation models from embedded system specifications for system development and early design assessment. From there the design information gets handed to a set of hardware and software developers which start their implementations virtually from scratch. This hand-off represents a discontinuity in the design process of the application. Next to initial conceptual errors, miscommunications between the design teams can lead to additional problems in the implementation of the embedded system.

In this paper, we introduce concepts for model compilation. It provides a structured approach for the transformation of abstract system models into design

implementations, and addresses the model continuity problem in our design methodology.

2 Model-Based Codesign

Conventional heterogeneous systems design is based on multiple, subsequent development steps in which designers refine hardware and software specifications to construct a system prototype. Based on experiments and system profiling, functionality is moved from software to hardware and vice versa in each iteration [5,6,7]. Early approaches practiced immediate partitioning into hardware (HW) and software (SW) components, pursue HW and SW development threads in isolation from each other, and often place a stronger emphasis on hardware rather than software [1,5,9].

MBC uses extensively implementation independent computer models to prototype embedded systems under design. Our work focuses on the development of design techniques in which models can be synthesized and tested for a number of objectives, taken individually or in trade-off combinations. MBC lets developers create and refine models of embedded systems independently of their eventual hardware and software implementation enforcing a late partitioning of the system design. Designers use simulation to explore the feasibility of virtual prototypes and then interactively map the specifications onto a mixed hardware/software architecture. Figure 1 provides an abstract overview of the design flow advocated by our methodology [3].

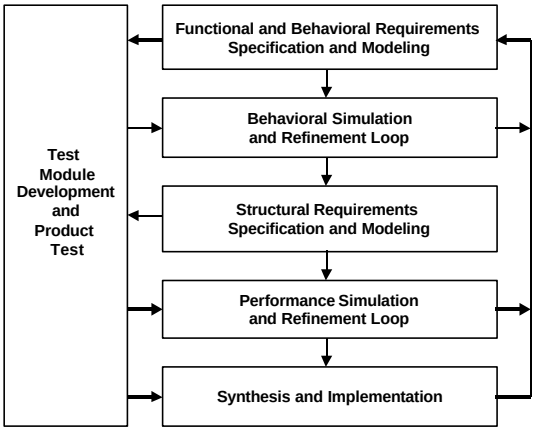

Figure 1. Abstract Design Flow in Model Based Codesign

Functional and Behavioral Requirements Specification and Modeling encompasses the solicitation and documentation of requirements as well as the development of an executable model. The Behavioral Simulation and Model Refinement Loop iteratively refines the design model until it is functionally correct. Structural Requirements Specification and Modeling relates physical design constraints to a proposed processing architecture. In the Performance Simulation and Model Refinement Loop, the model is enhanced with performance measures for computation and communication. Performance measures are obtained from a preliminary, reconfigurable system prototype which implements the chosen architecture. Synthesis and Implementation involves extracting design descriptions from the models in order to produce a physical prototype. Test

Module Development and Product Testing creates a set of test scenarios from System Requirements which can be used to assess the design at all levels of the design process.

3 Model Compilation

We address the model continuity problem in our model-based codesign methodology during model compilation. Model compilation follows the Structural Modeling of the embedded system prototype. It encompasses the process of translating an abstract, formal, executable model into a detailed design description of an embedded computing system. The implementation of this description should result in a digital, mixed hardware/software prototype.

A formal, executable system model specification consists of a set of mathematical model components. A model component is specified by its inputs, outputs, and state. A transition and output function describe state changes of a component [4,12]. A network of modular components can be represented in a coupled model specification.

A design description here refers to a list of integrated components which consist of: hardware components, their configurations, a board layout for the hardware components, software processes, and an operating system configuration. The design description specifies a physical prototype rather than a fine-tuned or optimized final implementation. A mixed processor/FPGA prototyping environment [10] is shown in Figure 2. Here a PC monitors the software executing on a processor while a logic analyzer provides insight into the execution of the reconfigurable hardware components.

Figure 2. Mixed Hardware/Software Prototyping Environment

3.1 Simulation Information Analysis

During the structural modeling phase in the MBC design process the system model is refined to a sufficient amount of detail for model compilation while still meeting all of the imposed behavioral system requirements. An appropriate level of granularity is reached when the interface to external peripherals follows the specified protocol faithfully at the level of bits and bytes. In the same manner interfaces between model components should be using bit or byte message based communication.

From our executable system model static and dynamic simulation information can be collected. While static information is extracted from the model description dynamic information can be obtained from the simulation engine.

Static model information like model structure, size, instruction mix, and behavior of each model component provides the foundation for the following architecture generation stage in the model compilation. Figure 3 depicts some static information contained in a composed system model.

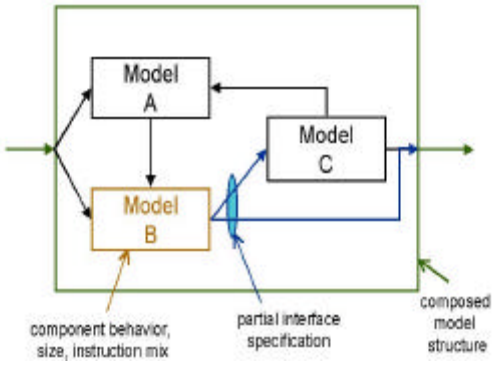

Figure 3. Static Information in a Composed System Model

When we use a modular, executable discrete event model specification [2,11] to describe our system model we can also access dynamic information generated by the model components as well as their simulators as shown in Figure 4. The dynamic information of the composed model provides important insights into component scheduling, concurrency, execution behavior, and data flow of the application as well as the bandwidth of their communication

channels.

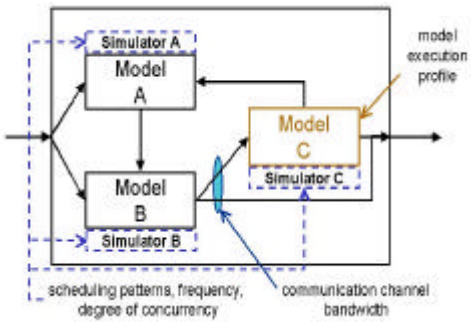

Figure 4. Dynamic Information in a Composed System Model

3.2 Architecture Generation

In the next stage of the model compilation hardware architectures are obtained by introducing structural requirements to the system model, e.g., available hardware components, and evaluating the previously collected simulation information of the composed model. Pure modular modeling specifications support concurrent design naturally. Therefore, each model component generates either a real or virtual processor with its own memory space. Model component couplings are converted into communication channels which follow a specified protocol. The implementation of these protocols is defined in the system model and adapted to the target architecture.

Model components are transformed into custom processors for pure hardware implementations. Each dedicated custom processor realizes a single model component and its simulator. It consists of a hardwired control unit, and memory for communication and data manipulation purposes.

Software implementations of model components are realized as virtual processors, i.e. by processes, on a general purpose processor in the generated architecture. Here, multiple model components can be grouped on a processor. Best groupings are determined from the dynamic simulation information of the system model. On each of the general purpose processors a multi-processor operating system creates a virtual environment of concurrently executing processors. This virtual concurrency does not only help in our mapping of model components to processors but it also increases the utilization of the processors which can be a problem in a pure custom processor design. Additional hardware blocks can also be generated from the model components to accelerate the execution on general purpose processors. In this case, however, these blocks will execute sequentially with the processes running on the processor.

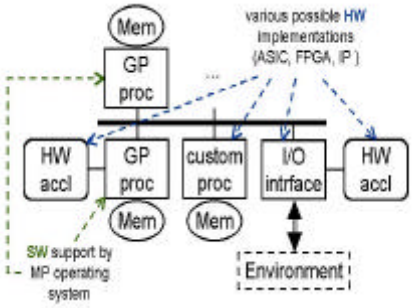

Figure 5. A Multi-Processor Target Architecture

The operating system abstraction also enforces a well-defined software communication interface. Communication and synchronization complexity will be hidden from the system designers and ensure a safe computing environment. In order to guarantee high performance the operating system should only require a minimum overhead. Figure 5 depicts the discussed target architecture.

3.3 Model Mapping

In the final stage of the model compilation the system model components are mapped to a selected hardware architecture. The basic model mapping occurs at the component level. In a structured approach each formal system model component description and its data are converted into either hardware descriptions of concurrently executing custom processors or software process descriptions.

The conversion of a model component into a custom processor involves the creation of custom processor control unit and memory. The needed

memory size is determined by the data structures which are associated to corresponding model component. A software implementation is achieved by converting a model component into process. Here the model transition functions are implemented with send and receive primitives. The model component mapping is illustrated in Figure 6.

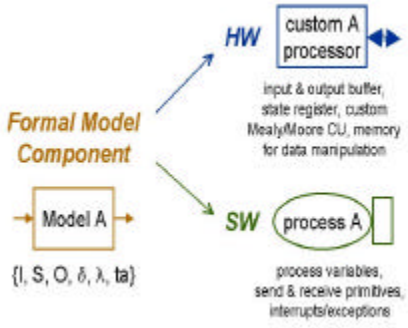

Figure 6. Model Component Mapping

We are left with fine-tuning software processes with hardware support for the interfacing or acceleration of computation to increase their performance. Custom hardware functions can be used, e.g., to convert data between different formats. It can also implement frequently used instruction sequences or component specific instructions which are not supported by the general purpose processor. This requires the model compiler in the simulation result analysis stage to recognize these functions and to synthesize code accordingly.

4 A Tool for Model-Based Design

The first step in our ongoing research efforts was to find a tool where we could apply the concepts described in the previous sections. The tool which we found to fit best for our design flow (see Figure 1) is the Cierto Virtual Component Codesign (VCC) environment available from Cadence Design Systems [2]. In 1998, Cadence, Philips, and Infineon Technologies (then Siemens) started the COSY [1] project.

The goal of the COSY project was to develop and deploy a design flow and methodology with the necessary supporting tools and models for system design based on re-usable hardware and software Intellectual Property components. COSY and Cadence´s Felix initiative led to the commercially available Cierto VCC environment.

The VCC tool enables designers to create functional and architectural models for an application at the system level. They can trade-off design alternatives early on in the design process by evaluating and comparing the impact of architectural choices. Functional model components can be described in a variety of formats. White Box C, a specialized subset of procedural C, is among these formats. Composed models can be created by connecting these components graphically. For simulation of the application the entire functional model is compiled into C++ code, executed, and can be evaluated in a functional analysis.

Similarly an architectural model can be specified. The functional components can then be mapped onto the architectural model. In the performance analysis the description of the architectural model is used to compute execution delays for each functional component depending on the architectural component it is mapped to. This information is annotated in the code of the functional model and then again compiled into C++ code. Delays are computed from abstract hardware component descriptions and represent only execution estimates.

The tool also supports easy graphical evaluation of various model execution parameters. Figure 7 shows part of a task schedule in form of a Gantt chart for our application example of an Embedded Java Virtual Machine (EJVM), which we produced during a performance analysis of its functional model on a single processor target architecture [8].

[1] ESPRIT Project No. 25443 COSY: COdesign Simulation and sYnthesis

Figure 7. Performance Analysis Results of EJVM model

Finally VCC can assist the designer in the mapping from the model to an implementation in an implementation analysis. Here the tool checks the integrity of hardware descriptions, which underlie the components of the target architecture, and generates software components.

5 Results

VCC was found to be very useful in the design process in gathering simulation information for various design applications. In the functional analysis can be used to validate the correct behavior of each model component as well as the composed model. Nevertheless the tool does not provide a direct link for feeding the analysis results into the specification of the target architecture as proposed in our approach. The implementation analysis comes closest to our idea of a model mapping. Here though the tool mainly supports only mappings to software processes.

These drawbacks can be compensated by introducing a more formal model specification front-end to the tool and adding some programs based on our previously introduced concepts of architecture generation and model mapping. Although VCC does not yet address the model continuity problem directly it can be used to complement our suggested approach.

6 Conclusions

In previous work we presented a high-level system design approach called model-based codesign that relies heavily on simulation modeling techniques to explore the feasibility of virtual prototyping. This paper described concepts for a model compiler. It enables a smooth transition from an application model to an implementation prototype in model-based design. We propose a three stage model compilation process: simulation information analysis, architecture generation, and model mapping.

The first stage is concerned with the evaluation of the information which is encoded and generated by an executable formal model description. Based on this collected information, an appropriate, application-specific target architecture is selected from a set of possible multi-processor architectures. Finally the system model is mapped to this architecture which results in a set of hardware descriptions for models which are mapped to hardware, and a set of processes for models which are mapped to software.

The Cierto Virtual Component Codesign (VCC) environment was tested as a candidate tool for the application of these concepts in a model-based design process. It was found to be useful for assisting the designer in the suggested approach. Nevertheless there is yet no direct support for an architecture generation or

an approach for transforming abstract functional model components into hardware descriptions.

So far we have not encountered any other tool candidates. Therefore, our future research will address analysis methods which will enable us to use simulation information for target architecture generation, and the mapping of abstract model components to both - hardware or software - implementations.

References

1. K. Buchenrieder and J.W. Rozenblit, "Codesign: An Overview", in J.W. Rozenblit and K. Buchenrieder (Eds.) *Codesign: Computer-Aided Software/Hardware Engineering*, pp. 1-16, IEEE Press, 1994.
2. Cadence Cierto Virtual Component Codesign (VCC) Environment, URL: http://www.cadence.com/technology/hwsw/ciertovcc/
3. S.J. Cunning et al., "Towards an Integrated, Model-Based Codesign Environment", *Proceedings of the IEEE Conference and Workshop on Engineering of Computer Based Systems*, Nashville, TN, 136-43, March 1999.
4. S.J. Cunning, S. Schulz, and J.W. Rozenblit, "An Embedded System's Design Verification Using Object-Oriented Simulation Techniques", *Simulation*, 72(4), 238-49, April 1999.
5. G. De Micheli, and R.K. Gupta, "Hardware/Software Co-Design", *Proceedings of the IEEE*, 85(3), 349-65, 1997.
6. R. Ernst, J. Henkel, and T. Benner, "Hardware-Software Cosynthesis for Microcontrollers", *IEEE Design and Test of Computers*, 10(4), pp. 64-73,1993.
7. D. Gajski, S. Narayan, F. Vahid, and J. Gong, Specification and Design of Embedded Systems, Englewood Cliffs, NJ: Prentice-Hall, 1994.
8. R. Kress, "Demonstration of COSY Results on a Robust Mobile Controller", *COSY report*, ESPRIT Project No. 25443: COdesign Simulation and sYnthesis, May 2000.
9. S. Kumar, *A Unified Representation for Hardware/Software Codesign*, Ph.D. Dissertation, University of Virginia, 1995.
10. A. Pyttel, and A. Sedlmeier, "Hardware/Software System Prototyping with Statically and Dynamically Reconfigurable FPGAs", *Proceedings of Design, Automation and Test Conference in Europe*, Paris, 234-47, February 1998.
11. S. Schulz, J.W. Rozenblit, M. Mrva, and K. Buchenrieder, "Model-Based Codesign", *IEEE Computer*, 31(8), 1998.
12. B.P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation*, 2nd Ed., Academic Press, 2000.

Stephan Schulz is a Ph.D. candidate in electrical and computer engineering at the University of Arizona. He received a B.Sc. from the State University of New York, Binghamton, and an M.Sc. in electrical and computer engineering at the University of Arizona. His research interests include embedded systems applications, model-based design, real-time operating systems, continuous and discrete-event simulation, and hardware/software codesign.

Jerzy W. Rozenblit is Professor of electrical and computer engineering at the University of Arizona. His research and teaching are in complex systems design and simulation modeling. He received a Ph.D. and M.Sc. in computer science from Wayne State and a M.Sc. in computer engineering from The Technical University of Wroclaw. He is an associate editor of ACM Transactions on Modeling and Computer Simulation and IEEE Transactions on Systems, Man and Cybernetics. He is a senior member of the IEEE and the Society for Computer Simulation and a member of the ACM. He's also a department editor for the IEEE Computer column - Integrated Engineering.

Klaus Buchenrieder heads research in Hardware-Software Codesign at the Central Development Laboratories of Infineon Technologies AG. He received a Dipl.Ing. in electrical engineering from the Fachhochschule in München, and a M.Sc. and Ph.D. in computer science from Ohio State University. He is the founding chair of the Codes/Cashe workshop series. He is professor of computer science at the University of Tübingen and adjunct professor of computer and electrical engineering at the University of Arizona.

Michael Mrva is Head of the Advanced Design Methods Department at Infineon Technologies AG and an adjunct professor in Electrical and Computer Engineering at the University of Arizona. His research interests include Hardware-Software Codesign, high-level specification and verification of hardware systems, object-oriented codesign, VHDL-based modeling, and hardware design reuse. He received a Ph.D. in mathematics from the University of Vienna.