Component-Based Hardware/Software Co-Design

Péter Arató, András Orbán, Zoltán Ádám Mann

Abstract— **The unbelievable growth in the complexity of computer systems poses a difficult challenge on system design. To cope with these problems, new methodologies are needed that allow the reuse of existing designs in a hierarchical manner, and at the same time let the designer work on the highest possible abstraction level.**

Such reusable building blocks are called components in the software world and IP (intellectual property) blocks in the hardware world. Based on the similarity between these two notions the authors propose a new system-level design methodology, called component-based hardware-software co-design, which allows rapid prototyping and functional simulation of complex hardware-software systems. Moreover, a tool is presented supporting the new design methodology and a case study is shown to demonstrate the applicability of the concepts.

I. Introduction

The goal of this research was to enable the design of embedded systems using a component-based methodology. A typical embedded system consists of both hardware and software components, therefore our methodology has to support them both. Moreover, the design of embedded systems is typically constrained in several non-functional aspects: the design has to respect costs and timing requirements such as hard real-time constraints.

In order to cope with the complexity of the designed system, the design methodology has to allow the designer to work at a sufficiently high level of abstraction. This means that the designer works with behavioral components, focusing mainly on functionality, while implementation issues are handled automatically. This also implies that components are handled regardless of whether they are implemented in hardware or in software. Moreover, the methodology has to be supported by powerful tools that provide a high-level view on the design while hiding and automating lower-level implementation issues.

II. Basic concepts

Our methodology is based on the following concepts. We define a component as a functional unit. The composition of components is based on their functionality. This functionality is captured by the interface of the component, which is completely decoupled from its implementation. It is also possible to have more than one implementation for the same interface. What is more, it is possible that there is a hardware implementation and a software implementation for the same interface. An important feature is hardware/software transparency, which means that a change between the two implementations is transparent to the rest of the system.

Thus we can identify three different kinds of components. There can be components for which there is only a software implementation. For instance, GUI elements or a database component are typically implemented in software. Similarly, there can be components for which there is only a hardware implementation. For example, it does not make sense to implement a video card in software. And finally, there can be components for which there is both a hardware and a software implementation. For instance, a cryptographic algorithm can be realized either by a program or by a special-purpose hardware unit [1].

During the design process, we face two consistency problems that are special to hardware/software co-design. We call the first one interface inconsistency. The question to answer here is whether or not two implementations can form a partitionable component. The other consistency problem, called state consistency, relates to partitioning. Namely, it is possible that repartitioning the system results in swapping the two implementations for a given component, and this way the state of a component can change.

III. Tool support

We have also developed a prototype tool for component-based hardware/software co-design. It is an extension of the Component Workbench [7]. Beside software components, the extended workbench also handles hardware component models by means of wrappers. Moreover, it also supports partitionable components. We have also integrated a partitioning algorithm that is based on integer linear programming [6] and we implemented the discussed consistency check mechanisms.

IV. Demo Description

The demo is concerned with frequency measurement, which has a number of important applications including mobile devices, car electronics, hard disk drives and so on.

In the demo, the frequency measurer is a partitionable component. That is, we have a cheap software implementation for it, which works on a micro-controller, and a more expensive hardware implementation based on a Field Programmable Gate Array. The software realization

is much slower, enabling it to measure frequency values under 25kHz, whereas the FPGA realization even works in the MHz domain appropriately.

The partitionable frequency measurer is part of a bigger application. The architecture consists of a signal generator that can generate signals of different frequency, the two implementations of the frequency measurer, as well as a PC, which is responsible for displaying the measured frequency values over time.

The demo shows the easy composition of the different software, hardware, and partitionable components in the Component Workbench. It also demonstrates consistency checking mechanisms, as well as automated partitioning between hardware and software implementations of a component.

V. Related Work

System-level design issues are addressed in [3], [4], [5]. In [9] a fast prototyping approach of complex systems is considered. Another related issue is interface synthesis (see [2] and references therein). For automatic component matching, see [10].

Our approach is different from these works in that it provides the designer with a very high-level view on the system, in which implementation details are completely hidden. Moreover, our system provides several automation possibilities.

VI. Integration

The methodology and tool presented here is an extension of the Component Workbench and the underlying Vienna Component Framework [8].

VII. Summary

Our methodology supports hardware and software components in a uniform way. It can also handle partitionable components, which can be partitioned between hardware and software using the developed algorithms. We also developed methods for checking both interface consistency and state consistency. Our methodology is also supported by a graphical tool, which is an extension of the Component Workbench.

References

[1] P. Arató, S. Juhász, Z. Á. Mann, A. Orbán, and D. Papp. Hardware/software partitioning in embedded system design. In *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, 2003.

[2] A. Basu, R. Mitra, and P. Marwedel. Interface synthesis for embedded applications in a co-design environment. In *11th IEEE International conference on VLSI design*, pages 85–90, 1998.

[3] P. Chou, R. Ortega, K. Hines, K. Partridge, and G. Borriello. Ipchinook: an integrated ip-based design framework for distributed embedded systems. In *Design Automation Conference*, pages 44–49, 1999.

[4] Ph. Coussy, A. Baganne, and E. Martin. A design methodology for integrating ip into soc systems. In *Confrence Internationale IEEE CICC*, 2002.

[5] S. J. Krolikoski, F. Schirrmeister, B. Salefski, J. Rowson, and G. Martin. Methodology and technology for virtual component-driven hardware/software co-design on the system level. In *ISCAS*, 1999.

[6] Z. Á. Mann and A. Orbán. Optimization problems in system-level synthesis. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2003.

[7] Johann Oberleitner and Thomas Gschwind. Composing distributed components with the Component Workbench. In *Proceedings of the 3rd International Workshop on Software Engineering and Middleware*, volume 2596 of *Lecture Notes in Computer Science*. Springer-Verlag.

[8] Johann Oberleitner, Thomas Gschwind, and Mehdi Jazayeri. The Vienna Component Framework: Enabling composition across component models. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 25–35. IEEE, May 2003.

[9] F. Pogodalla, R. Hersemeule, and P. Coulomb. Fast protoyping: a system design flow for fast design, prototyping and efficient IP reuse. In *CODES*, 1999.

[10] P. Roop and A. Sowmya. Automatic component matching using forced simulation. In *13th International Conference on VLSI Design*. IEEE Press, 2000.