TRAIN: A Virtual Transaction Layer Architecture for TLM-based HW/SW Codesign of Synthesizable MPSoC

Wolfgang Klingauf, Hagen Gädke, Robert Günzel
Technical University of Braunschweig, Dept. E.I.S.
Mühlenpfordtstr. 23, D-38106 Braunschweig, Germany
[klingauf, gaedke, guenzel]@eis.cs.tu-bs.de

Abstract

Our concept of a virtual transaction layer (VTL) architecture allows to directly map transaction-level communication channels onto a synthesizable multiprocessor SoC implementation. The VTL is above the physical MPSoC communication architecture, acting as a hardware abstraction layer for both HW and SW components. TLM channels are represented by virtual channels which efficiently route transactions between SW and HW entities through the on-chip communication network with respect to quality-of-service and realtime requirements. The goal is to methodically simplify MPSoC design by systematic HW/SW interface abstraction, thus enabling early SW verification, rapid prototyping and fast exploration of critical design issues. With TRAIN, we present our implementation of such a VTL architecture for Virtex-II Pro and PowerPC and illustrate its efficiency by experimentation.

1. Introduction

Modern multiprocessor System-on-Chip (MPSoC) architectures require the system engineer to juggle with an increasing number of heterogeneous processor cores, signal processing components, peripheral controllers, memory subsystems, and on-chip communication architectures. As a result, in current HW/SW codesign methods one of the most important issues is mapping the system's communication infrastructure to the target architecture, which particularly affects the HW/SW integration phase, i.e. development of the HW/SW interfaces [19].

Raising communication modeling to the transaction level (TLM) is considered as a key technology for fast design space exploration and early embedded software development. However, in current design flows, the abstract point-to-point transaction channels of the TLM model cannot be mapped directly to the target architecture. Rather, they have to be refined to a synthesizable representation, implementing both the cycle-accurate timing protocol and the low-level signals of the target communication architecture. This results in a significant TLM-to-RTL design gap, which not only affects productivity and design reuse, but also prevents the early exploration of critical design issues. Moreover, the final results of the communication refinement process in terms of design constraints such as timing, area, and power consumption are not deterministically predictable, which renders accurate prognosis in the TLM model impossible.

In order to make TLM-based HW/SW codesign of MPSoC more comfortable, we introduce our concept of a *virtual transaction layer* (VTL). The VTL is a logical communication layer which is placed above the physical communication architecture of the target MPSoC. The goal is to systematically facilitate communication refinement with TLM, and in particular, to simplify the HW/SW integration phase. To this end, a flexible and scalable HW/SW communication architecture is required which from the programmer's point of view fully retains all TLM channels as *virtual channels* in the final system implementation. The key features that such an architecture should provide are:

Flexibility: The hardware part of the architecture should be highly flexible and scalable, enabling each processor core to be connected to an arbitrary number of on-chip communication networks.

Performance: The HW/SW interface should deliver highest possible data transfer rates according to the underlying communication subsystems.

Adaptability: The VTL should be highly retargetable, rendering any re-engineering of MPSoC components unnecessary. For software components, TLM channel interface method calls such as `put` and `get` should be transparently mapped to the HW/SW interface. Hardware components should be accessible independently from their communication protocol.

Configurability: Each TLM channel of the high-level system model should be represented by a virtual communication channel in the implementation model. Virtual channels should be routable to different on-chip communication networks, and should provide quality of service (QoS) management.

In this paper we present the TRAIN *TRAnsaction INterchange* architecture which meets all of the above requirements and, thanks to its modular design, can be efficiently adapted to virtually any target architecture. To justify our design decisions, we will first briefly outline current TLM design methods for MPSoC and give an overview on related work. We then will describe the TRAIN architecture in detail and look at how we map HW/SW communication efficiently to the target system using a hardware abstraction layer. We have implemented this architecture on the Virtex-II Pro FPGA fabric and illustrate our approach with experimental results.

2. TLM-based SoC design and related work

Recent and upcoming SoCs contain sophisticated communication architectures ranging from common bus architectures to novel networks-on-chip (NoC). Numerous subsystems include application-specific processors, peripheral IP and distributed memories (figure 1). According to [8], 90% of new ASICs already contain a CPU and the trend of building heterogeneous MPSoCs will even be accelerated.

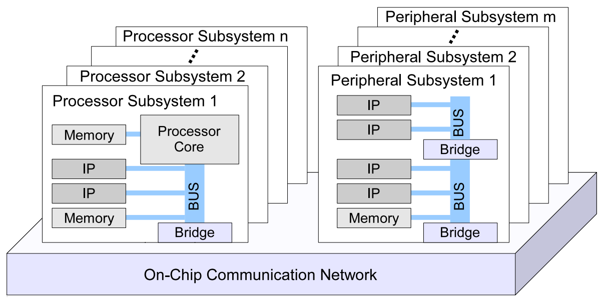

Figure 1. Complex multiprocessor SoC

Large interest has been attracted by transaction level modeling with system level design languages such as SpecC [4] and SystemC [6]. Based on the orthogonalization of system functionality and system communication, these languages support virtual prototyping and mixed-mode simulation [1, 5, 9] as well as communication analysis [11, 16, 17] using several TLM abstraction levels. Faced with the task of generating a synthesizable MPSoC implementation, however, the crucial issue is communication refinement.

Methodical communication modeling is considered as a key technology for straightforward SoC design. Thus, many recent publications deal with standardization proposals for TLM channels [12, 15] and TLM abstraction layers [2, 13]. In [1] and [17], the authors tackle automatic communication refinement, and Kogel et al. [11] address virtual architecture mapping using a generic bus model. Fummi et al. [3] and Ramaswamy et al. [14] show application-specific techniques for HW-assisted verification with SystemC. However, all of these approaches do not provide generic solutions for TLM-to-RTL communication synthesis.

A straightforward approach for improving MPSoC design efficiency is using wrappers to connect heterogeneous IP to the on-chip communication network. Outstanding research in this field has been done by Jerraya et al. [19]. Their methodology, however, requires the manual adaptation and configuration of architecture templates, and HW/SW communication refinement is not generically taken into account, which in our opinion is crucial for automatic SW synthesis from TLM.

We believe that in current and future MPSoC there is a rising demand for more and more generic communication architectures which overlay the complex infrastructure of the on-chip communication networks, thus providing a generic yet flexible TLM-based view of communication to the system engineer. In the following, we present our approach for such an architecture, which in combination with our systematic TLM design method [9] and automatic refinement strategy [10] can considerably improve MPSoC design efficiency.

3. TRAIN top-level architecture

With TRAIN, we developed a specification and reference implementation of a number of generic HW and SW adapters to interconnect processor cores, HW components, and SW modules to a virtual transaction layer architecture in a MPSoC. Our HW and SW communication adapters are runtime-configurable and map virtual transaction-level point-to-point channels between HW and SW components to physical communication paths on a target platform, using a tunneling protocol. By this means, TRAIN represents an abstract communication layer for MPSoC.

When designing TRAIN, we aimed at a scalable, adaptable and highly configurable architecture which includes quality-of-service features required by realtime applications. TRAIN covers bidirectional communication between both HW and HW as well as SW and HW components, e.g. between a processor core and multiple HW entities such as IP cores, peripheral components, memories, etc. In a multiprocessor system, TRAIN can be used to connect each processor core to an arbitrary number of HW entities.

TRAIN-based *virtual* communication is mappable to *physical* point-to-point connections, on-chip buses and networks-on-chip (NoC) as well as a mix of these architectures. Each HW component is equipped with a communication adapter called *accessor*, and each processor core connects to the on-chip communication subsystems by a *CPU adapter*. On the SW side, TRAIN comprises of a platform-specific device driver and a *hardware abstraction layer* (HAL). An overview of TRAIN applied to a processor subsystem of an MPSoC (figure 1) is depicted in figure 2.

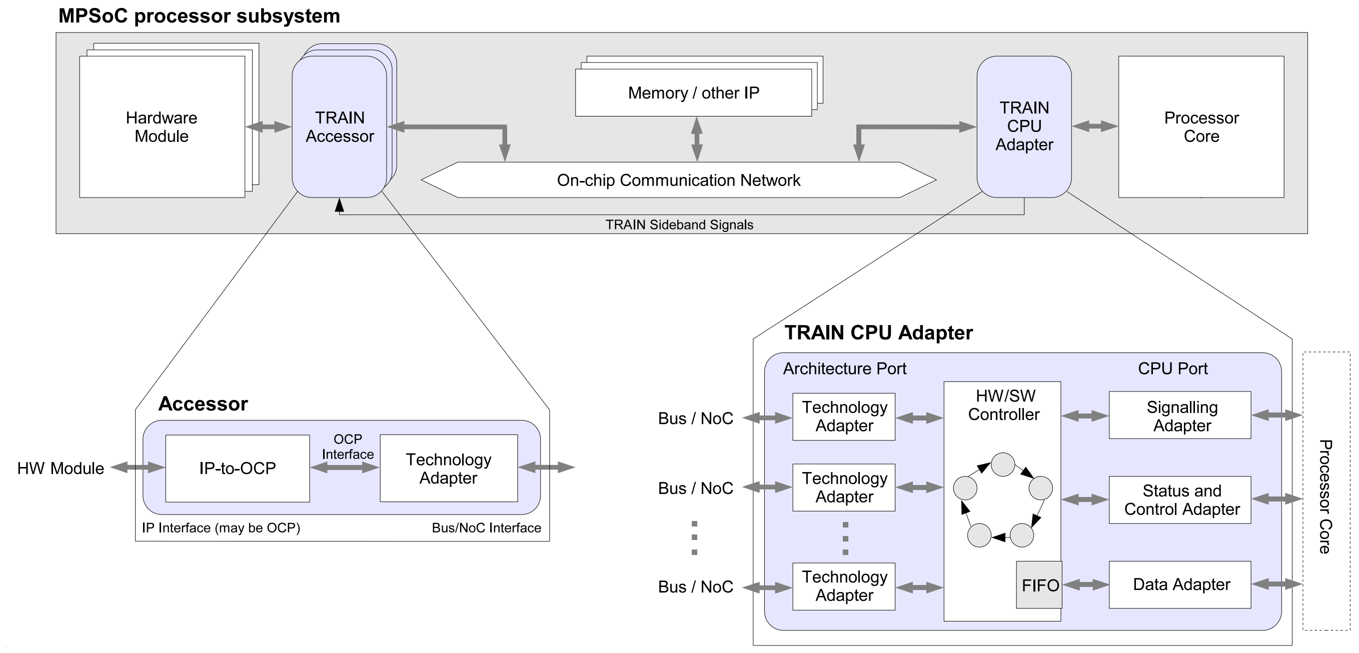

Figure 2. TRAIN architecture overview

3.1. CPU adapter

The TRAIN CPU adapter routes transactions between SW processes and HW modules connected to the on-chip communication network. It consists of three different types of components: The *HW/SW controller*, *CPU port adapters* and *technology adapters*.

The HW/SW controller's task is to coordinate CPU-to-HW and HW-to-CPU data transfers in terms of scheduling and routing. The data is transferred in a block-by-block manner using a FIFO buffer of configurable size. By the technology adapters, the data blocks are transferred to the appropriate destination on-chip bus or network. The mapping from address ranges to technology adapters is given by a routing table which can be configured at runtime. Currently, 16 technology adapter slots are provided by our reference implementation.

The CPU port's signalling, data, and status-and-control adapters map our TRAIN specific CPU interface to a target CPU. A HW/SW data handshake mechanism including interrupt handling ensures data integrity, while error detection guarantees transfer reliability.

3.2. Accessors

To integrate HW IP cores into the virtual transaction layer, accessors can be used. They connect the HW IP's interface to the target bus or NoC using a technology adapter. To allow for advanced QoS management independent from the underlying architectures' capabilities, they also can be connected to the CPU adapter by sideband signals. For example, an ongoing transfer can be aborted by the CPU adapter to give way for a high-priority transaction.

Moreover, accessors can provide remote direct memory access (RDMA) capabilities. Using this feature, the CPU adapter can initiate an arbitrary length data burst between an accessor and a memory block. The transaction is controlled by the accessor and does not require further CPU interaction, which enables fast SW-controlled HW-to-memory transfers.

As a standard interface to HW IP, we decided to use the Open Core Protocol (OCP) [12], because this interface is platform-independent, versatile and commonly used. For OCP-compliant HW cores, accessors basically are composed of a technology adapter, which maps the OCP interface to the target bus or NoC. If a HW core utilizes another communication interface, its accessor additionally has to be equipped with an appropriate communication wrapper (figure 2).

3.3. Technology adapters

Technology adapters (TA) are used to connect the TRAIN CPU adapters and accessors to the platform's communication architecture, which in MPSoC typically incorporates multiple buses or networks-on-chip. The TA's intention is to hide the complexity of the underlying communication network and simultaneously utilize as much of the architecture's features as possible. Each TA can be configured to work as master, slave, or both. For interfacing to the TAs, we use a light-weight OCP subset, which allows for using TAs both in accessors and CPU adapters. The TA translates this protocol into a certain bus or network protocol and vice versa.

4. HW/SW interface abstraction

4.1. Hardware abstraction layer

A primary objective of the virtual transaction layer approach is to use the same SW implementation for the target system as for high-level TLM simulation or even hardware-in-the-loop verification. To this end, the VTL includes a hardware abstraction layer (HAL) providing an API that lets SW modules communicate with HW modules by simple TLM interface method calls. It communicates with the CPU adapter of a processor core by a HW/SW handshake protocol. In order to support realtime applications, the HAL should also allow for configuring each virtual HW/SW communication channel with QoS requirements. Moreover, the HAL should be highly portable in terms of supported target architectures and operating systems. This results in the following architectural demands:

- For each processor core in the MPSoC, one HAL instance should aggregate all virtual HW/SW channels that exist for this processor.
- The HAL should provide a single common API applicable to any target architecture.
- The platform dependent part of the HAL should be kept as small as possible.

These requirements can be achieved by splitting the HAL implementation into three layers, namely *device driver*, *virtual channel manager*, and *HAL API*. This concept is outlined in figure 3. The advantage of this approach is that the device driver is the only platform-dependent part of the HAL implementation, thus minimizing re-engineering effort when switching to another target processor or operating system.

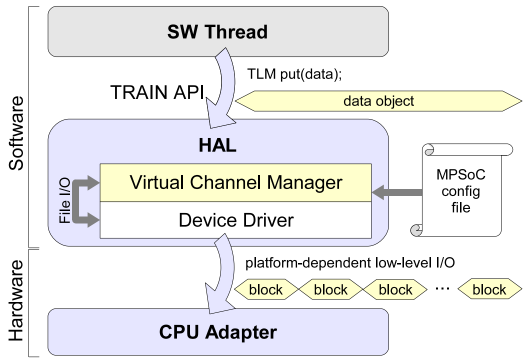

Figure 3. HW/SW interface abstraction

Device driver

The low-level driver provides communication primitives that gain access to the TRAIN CPU adapter, i.e., IRQ handling, register I/O and data transfer. Its intended purpose is to decouple the mid-layer from the hardware.

Many MPSoC operating systems such as embedded linux or QNX shield kernel space from user space to safeguard system integrity. In such systems, the HAL driver would be part of the kernel space and thus should provide a common interface to the user space. As file I/O is part of every OS implementation, we propose to represent HW/SW communication channels by means of *virtual file pointers* to the user space. Thus user applications can establish HW/SW communication by simply reading and writing data streams to a file, using standard OS function calls. The higher-level HAL API uses these file pointers as a base to provide more sophisticated communication services to the user application, i.e. SystemC-like transaction-level channels and events for thread synchronization.

Virtual channel management and TRAIN-API

The virtual channel manager in the mid-layer implements the main functionality of the HAL. It uses the low-level device driver for platform-independent data handshake with HW components via the TRAIN CPU adapter, and provides the TRAIN-API to applications.

QoS requirements can be announced by special TRAIN-API method calls. The following SystemC example shows the QoS configuration of a high-level TLM channel in the top-level model of the system:

```
// create TLM channel
train_fifo<myobj> channel;
// connect moduleA with moduleB using the channel
moduleA.outPort(channel);
moduleB.inPort(channel);
// configure QoS requirements
channel.setProperty("QoS_Mode", "Burst");
channel.setProperty("QoS_Bandwidth", "30 MB");
```

For QoS transactions, the HAL and the CPU adapter treat HW/SW communication with respect to the special requirements, as far as this is supported by the underlying communication architecture. For normal channels, a best effort service is delivered.

PowerPC implementation

We have created a reference implementation of these concepts using SystemC for the TLM channel library and C/C++ for the HAL. The HAL was compiled with the GNU cross compiler for the PowerPC 405 processor, which is part of the Virtex-II Pro FPGA fabric. As depicted in figure 3, the HAL uses a system configuration file to become aware of the available HW slaves in the system, their adresses, and their module names in the TLM model. This configuration file can be generated by a simple script from a SystemC model of the whole HW/SW system and allows for automatic initialisation of all HW/SW communication channels in the final MPSoC implementation. To this end, the QoS requirements specified in the SystemC model are also included.

4.2. HW/SW communication

For SW-to-HW data transfers, the HAL divides the SW's transaction data into blocks appropriate to the CPU adapter's FIFO size and sends them via a technology adapter to the destination. HW-initiated transfers are signaled to the HAL by interrupts, which in response reads the CPU adapter's FIFO buffer and reassembles the data blocks. Each TRAIN transaction starts with a 64bit header indicating transfer mode, target port address and transaction length.

If multiple data transfers through one CPU adapter take place at the same time, data blocks are time-division multiplexed according to the QoS configurations of the actuating virtual channels. As a result, a high-priority transaction can slow down or even stall a lower-priority transaction when both transactions share one medium on the chip.

5. Design flow

TRAIN was designed as a back-end for automating MPSoC synthesis from abstract transaction-level communication models. The integration of TRAIN into our TLM-based design flow (see [9, 10]) is outlined in figure 4.

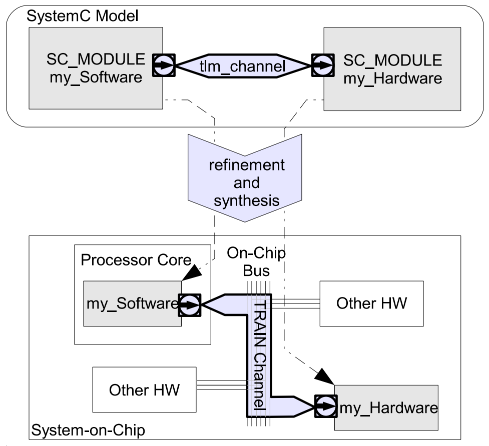

Figure 4. Design flow integration

In the high-level model, transactions between system components are described by abstract TLM channel method calls like `put(data)` and `get(data)`. In the implementation model, the TLM channels have to be replaced by physical on-chip buses and networks with architecture-specific protocols and complex RTL signal interfaces.

To bridge this gap, in the first step of the design flow TLM method calls are refined to synthesizable interfaces, e.g. OCP (see [10]). Afterwards, TRAIN CPU adapters and IP core accessors are chosen from a target platform library. The result is a fully synthesizable RTL implementation of the MPSoC which still reflects the original TLM communication channels by means of virtual channels.

In figure 4, this concept is demonstrated by an example TLM model, in which data is sent from one module to another by a TLM channel. In the final RTL implementation, this data is sent from a processor core to a HW module via an on-chip bus. However, by using a TRAIN CPU adpater and a HW accessor, the complex interface and protocol of the on-chip bus is completely hidden from the HW and SW modules. From the HW and SW module's point of view, a simple point-to-point connection is established.

6. Experimental results

In order to analyze the practicability of our approach, we created a TRAIN reference implementation for the Xilinx Virtex-II Pro [18] FPGA fabric. We chose the V2P30 as a target platform. It boasts 13,696 logic slices and two immersed PowerPC 405 processor cores that connect to the on-chip CoreConnect [7] bus architecture.

We implemented a C/C++ hardware abstraction layer as described in section 4 and compiled it for the PowerPC with Xilinx EDK 7.1. To equip the PowerPC with a TRAIN HW interface, we implemented a CPU adapter which is utilizing the PowerPC's on-chip memory interface DSOCM. Furthermore, we implemented technology adapters and accessors for the FPGA's On-chip Peripheral Bus (OPB). To set up a test system, we created various HW modules. These modules provide TRAIN-compliant OCP bus interfaces and are connected to the OPB by accessors. They are runtime configurable and can produce and consume programmable amounts of data.

Table 1 lists the ressource allocations and timing estimations of our TRAIN library for the Virtex-II Pro. All components were implemented with Verilog and compiled with Xilinx ISE 7.1. As can be seen in the table, the additional ressource allocation by TRAIN in a MPSoC is quite low. For example, when both processor cores of the V2P30 are equipped with a CPU adapter, 93% of the FPGA are still unallocated.

Table 1. TRAIN ressource allocation on V2P

Component	Slices	Clock (MHz)
CPU Adapter	488	98
Tech. Adapter OPB (master)	97	283
Tech. Adapter OPB (slave)	67	195
OCP-2-OPB Accessor (master)	223	188
OCP-2-OPB Accessor (slave)	202	129

To analyze the performance of our TRAIN implementation, we first measured the transfer times of 1MB transactions from SW to HW using different FIFO sizes in the CPU adapter. The results are shown in the left-hand diagram of figure 5. It can be seen that due to fixed arbitration times for each block transfer, small block sizes are not suitable for bus-based communication. On the OPB, a block size of 4KB turned out to be applicable.

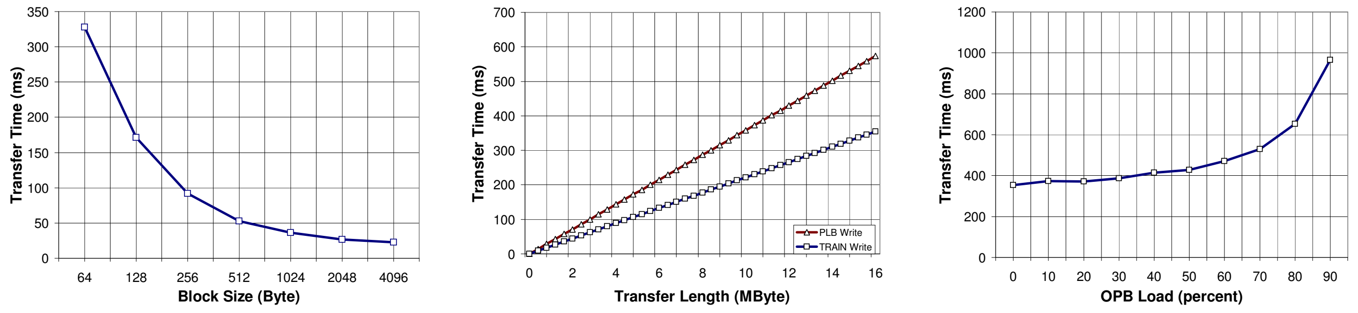

Figure 5. Experimental results

In the middle diagram, we then measured the throughput of our CPU adapter when routing transactions from SW threads to HW modules via the OPB. In addition, the PLB curve in this diagram depicts the transfer time for standard PLB-based RAM writes by the PowerPC without TRAIN. The comparison shows that TRAIN outperforms normal shared-memory-based HW/SW communication on the Virtex-II Pro, since it can take advantage of the fast on-chip memory interface of the PowerPC.

Finally, the graph on the right-hand side shows the transfer times of a 16MB SW-to-HW transaction at different additional OPB loads. For this experiment, a configurable master and a corresponding slave module have been used to pollute the OPB with additional bus traffic, using a higher priority than the SW-to-HW virtual channel. The diagram shows that our TRAIN implementation is able to operate the underlying shared communication architecture to full capacity, with respect to its QoS configuration.

All experiments were done using a 50MHz clock.

7. Conclusion

In this paper, we introduced our concept of a virtual transaction layer which overlays the physical communication architecture of an MPSoC. The goal is to reflect high-level channels of a TLM model as virtual channels in the final implementation. By tunneling communication through the on-chip buses and networks, virtual channels provide runtime-configurable point-to-point links with standard interfaces to HW and SW components. We have shown that this approach enables the mapping of TLM channels to various MPSoC communication architectures and hence can facilitate automatic TLM-to-RTL communication refinement. Our TRAIN reference implementation for Virtex-II Pro provides QoS support and includes a platform-independent hardware abstraction layer for SW components.

In our ongoing work, we address runtime-reconfigurable distributed embedded systems and implement additional TRAIN CPU adapters and accessors. Furthermore, we are developing a SystemC framework for fast communication architecture analysis and synthesis based on the VTL approach.

References

[1] S. Abdi and D. Gajski. Automatic Communication Refinement for System Level Design. *Proc. DAC*, 2003.

[2] L. Cai and D. Gajski. Transaction Level Modeling: An Overview. *Proc. CODES+ISSS*, 2003.

[3] F. Fummi, M. Loghi, S. Martini, and M. Monguzzi. Virtual Hardware Prototyping through Timed Hardware-Software Co-simulation. *Proc. DATE*, 2005.

[4] D. Gajski, J. Zhu, and R. Domer. Specc: Specification Language and Methodology. *Kluwer*, 2000.

[5] A. Gerstlauer, D. Shin, R. Doemer, and D. Gajski. System-Level Communication Modeling for Network-on-Chip Synthesis. *ASP-DAC*, 2005.

[6] T. Groetker, T. Liao, and S. Martin. *System Design with SystemC*. Kluwer, 2002.

[7] IBM. The CoreConnect Bus Architecture. *IBM*, 1999.

[8] A. Jerraya. Long Term trends for Embedded System Design. *Proc. CEPA 2 Workshop*, 2005.

[9] W. Klingauf. Systematic Transaction Level Modeling of Embedded Systems with SystemC. *Proc. DATE*, 2005.

[10] W. Klingauf and R. Guenzel. From TLM to FPGA: Rapid Prototyping with SystemC and Transaction Level Modeling. *Proc. FPT*, 2005.

[11] T. Kogel, M. Doerper, A. Wieferink, R. Leupers, G. Ascheid, and H. Meyr. A Modular Simulation Framework for Architectural Exploration of On-Chip Interconnection Networks. *Proc. CODES+ISSS*, 2003.

[12] OCP-IP. Homepage. www.ocpip.org. 2005.

[13] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration. *Proc. DAC*, 2004.

[14] R. Ramaswamy and R. Tessier. The Integration of SystemC and Hardware-assisted Verification. *Proc. FPL*, 2002.

[15] A. Rose, S. Swan, J. Pierce, and J. Fernandez. Transaction Level Modeling in SystemC. *OSCI TLM-WG*, 2005.

[16] A. Siebenborn, O. Bringmann, and W. Rosenstiel. Communication Analysis for System on Chip Design. *Proc. DATE*, 2004.

[17] R. Siegmund and D. Mueller. SystemC-SV: An Extension of SystemC for Mixed Multi-Level Communication Modeling and Interface-Based System Design. *Proc. DATE*, 2001.

[18] Xilinx. Virtex-II Pro Product Specification. *Xilinx*, 2003.

[19] N. Zergainoh, A. Baghdadi, and A. Jerraya. Hardware/Software Codesign of On-chip Communication Architecture for Application-Specific Multiprocessor System-On-Chip. *Int. Journal of Embedded Systems*, 2004.