Host-Compiled Simulation of Multi-Core Platforms

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas, 78712
Email: gerstl@ece.utexas.edu

***Abstract*—Virtual platform models are a popular approach for virtual prototyping of multi-processor/multi-core systems-on-chip (MPCSoCs). Such models aid in system-level design, rapid and early design space exploration, as well as early software development. Traditionally, either highly abstracted models for exploration or low-level, implementation-oriented models for development have been employed. Host-compiled models promise to fill this gap by providing both fast and accurate platform simulation and prototyping. In this paper, we aim to provide an overview of state-of-the-art host-compiled platform modeling concepts, techniques and their applicability and benefits.**

I. Introduction

In recent years, virtual platform models have gained tremendous popularity for early and rapid prototyping of embedded system designs. With inherent and continuously growing complexities, such models enable system-level design space exploration to derive optimized system architectures for a given application (or set of application scenarios) under tight performance and real-time constraints. At the same time, virtual platform prototypes can serve as the basis for early software development, validation and debugging before the physical hardware is available.

To facilitate design space exploration and application development, designers require models that simulate fast yet can provide accurate feedback about platform effects. Traditional models for exploration at high levels of abstraction provide fast evaluation but rely on inaccurate, coarse-grain estimation or worst-case static analysis. By contrast, models for actual software and application development tend to be at low, implementation-oriented levels using relatively slow instruction-set simulation (ISS) or even cycle-accurate micro-architecture and register-transfer level (RTL) hardware descriptions.

An active area of research is the search for modeling solutions that can fill the gap in between high-level exploration and low-level implementation models. Most recently, a large number of solutions that can generally be classified as host-compiled models have begun to emerge. These approaches pair a high-level functional model with back-annotation of statically determined low-level timing estimates in order to achieve fast *and* accurate simulation. An open question that is not yet well understood, however, is the right choice of granularity that achieves the best balance between the capability to accurately analyze tight static bounds and to model all necessary dynamic effects with little to no overhead.

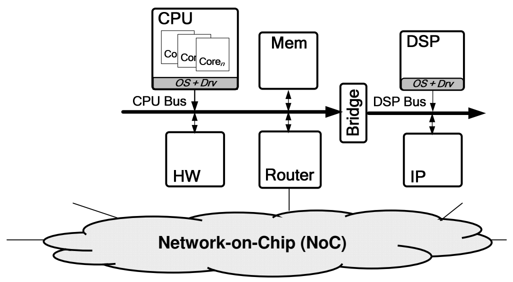

Fig. 1. MPCSoC platform architecture.

In this paper, we aim to provide an overview of host-compiled platform modeling concepts, benefits and challenges. The remainder of this paper is organized as follows: after a brief introduction to platform target architectures and general modeling concepts, we will provide an overview of virtual platform modeling and related work in Section II. Section III will further detail host-compiled processor modeling techniques, and Section IV will show an industrial-strength case study to demonstrate the benefits of host-compiled approaches. Finally, we will conclude the paper with a summary in Section V.

A. Platform Architectures

Driven by ever increasing application demands and technological advances, we have entered an era where complex multi-processing platforms beyond a single CPU are integrated on a single chip. Various approaches aim to manage complexities and provide high performance in a scalable fashion through regular, homogeneous structures adapted from general-purpose computing, including Network-on-Chip (NoC) or multi- and many-core architectures. However, the need for optimization will continue to push for specialization, and architectures are likely to be highly heterogeneous, hybrid and hierarchical in nature, such as homogeneous single- or multi-core processors as part of heterogeneous, bus-based subsystems that are interconnected via a homogeneous or heterogeneous backbone on- or off-chip network (Fig.1).

Such Multi-Processor and Multi-Core Systems-on-Chip (MPCSoCs) provide a mix of homogeneous, symmetric and heterogeneous, asymmetric multi-processing (SMP and AMP). We can generally define an MPCSoC to be a hierarchical composition of shared-memory multi-core (i.e. SMP) processors that are components in an overall distributed-memory multi-

17072/10 $25.00 (C) 2010 IEEE DOI 10.1109/RSP_2010.SS3

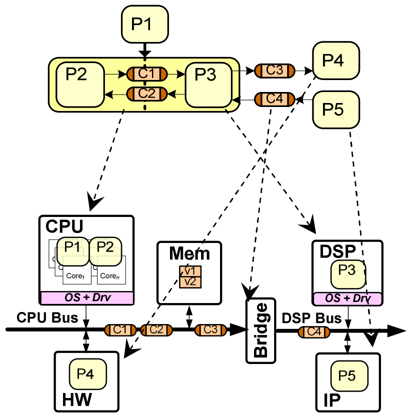

Fig. 2. System-level design.

processor (i.e. AMP) system. As such, each processor has one or more cores that share a common subset of resources, such as bus interfaces, local memory and a single real-time operating system (RTOS). On the other hand, different single- or multi-core processor are each run their own, independent software stack and are loosely connected via networks of busses or other communication structures.

B. Modeling Levels

At the system level, the design process is a mapping, i.e. partitioning and scheduling of application computation and communication onto a target platform consisting of processing elements (PEs) and communication media (busses) (Fig.2). The result is a structural system model of application processes and channels running on the platform architecture. Platform modeling is concerned with capturing such structural system-level descriptions at various levels of detail and abstraction.

Approaches can generally be classified according to the level of granularity at which target functionality and timing is modeled. Based on an orthogonalization and separation of concerns, we generally distinguish between the computation and communication side. As shown in Fig.3, we can furthermore separate computation and communication functionality into layers with well-defined interfaces. A model is then defined by the amount of implementation detail, i.e. by the amount of target-specific computation and communication layers explicitly included. Any implementation layers below a certain interface are abstracted away and replaced by an abstract model of the underlying target functionality and timing.

Functionality is usually modeled by mapping elements at the chosen interface directly into equivalent code on the simulation host. For example, an instruction-set simulation (ISS) model will translate target assembly code into functionally equivalent host instructions. Matching timing models are constructed through back-annotation of interface objects, such as instructions, with estimated timing information. In the simplest case, estimates are derived through a static table lookup (or ignored for a purely functional simulation). Especially at higher levels, however, timing data is often obtained by statically emulating and estimating execution on a detailed target model that

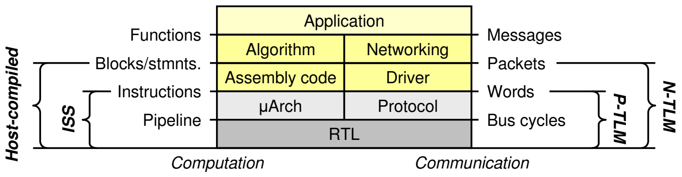

Fig. 3. Modeling layers.

may reach all the way down to the lowest RTL layer. Since estimation and back-annotation is performed before runtime, simulation speeds do not suffer but accuracy is limited by any static assumptions made about dynamic effects. Alternatively, back-annotation can happen dynamically, or in a hybrid static and dynamic fashion by running a more or less detailed timing model next to the abstracted functional simulation. This allows for tradeoffs between speed and dynamic accuracy, where parallelism and hardware acceleration in the simulation host can be exploited [1].

There exist a variety of static analysis approaches that abstract the system at the granularity of complete applications based on static models of process [2], channel and system-level interactions [3], [4]. Such high-level analysis is necessarily based on static estimation or pre-characterization, e.g. of task execution times on a given processor under average- or worst-case conditions. At lower levels, restricted sets of operations, well-defined interactions and limited dynamic dependencies make a static analysis and worst-case operation feasible and desirable. However, flexibility in applications, complex system-wide interactions and wide variations under different dynamic operation conditions often result in sub-optimal or overly conservative static analysis at the system level.

For these reasons, simulation-based evaluation and exploration solutions are popular. Similar to purely static analysis, a class of solutions for early and fast exploration simulates application processes and channel interactions at the granularity of functions and messages [5], [6]. Such approaches utilize back-annotation through similar static estimation or pre-characterization of target-specific function and message timing. As such, they suffer from corresponding inaccuracies in determining worst- or average-case bounds. Furthermore, while dynamic interactions (e.g. through scheduling or arbitration) can be simulated, their accuracy is limited by the coarse granularity of the model itself. Nevertheless, such models can provide very rapid feedback to drive initial pruning of the design space of clearly infeasible solutions.

At the other end of the spectrum, low-level, implementation-oriented platform models have traditionally been constructed using RTL simulations at the micro-architecture and bus protocol level. However, while being cycle-accurate, such models are clearly too slow for full-system simulation of increasingly complex platforms. On the computation side, aforementioned binary translating ISS approaches have replaced or complemented traditional cycle-accurate micro-architecture simulators [7], [8], [9], [10]. Such simulators can provide significant speedups (reaching simulation speeds of several hundred MIPS), but often focus on functionality and speed at the expense of limited or no timing accuracy.

On the communication side, Transaction-Level Modeling (TLM) has become tremendously popular and almost universally accepted as a vehicle for acceleration of platform model integration. TLM allows orders of magnitude faster simulations by abstracting away pin- and wire-level protocol details into an interface at the level of bus read/write transactions [11], [12], [13], [14] (protocol TLM, P-TLM) or complete data packets (network TLM, N-TLM) [15], [16]. Internally, bus TLM descriptions realize an abstracted, yet accurate model of the functionality and timing of such bus transactions. With the exception of dynamic effects such as arbitration or preemption, the predictability of bus protocol timing enables TLM-based descriptions with little to no loss in accuracy. For dynamic effects, accuracy is inherently linked to the granularity, i.e. the level at which the TLM simulation is realized. Note, however, that several techniques have been developed that implement principles of optimistic timing prediction and subsequent correction of dynamic disturbances in an attempt to enable fast yet accurate simulations at higher levels [17], [18].

Following the success of TLM concepts, the idea of host-compiled modeling is to similarly push computation modeling to higher abstractions above the level of instructions. The idea is to define a level below the granularity of complete functions where all major dynamic effects can be simulated but simulation speeds remain close to the native execution of functionality on the simulation host. For this purpose, host-compiled approaches model computation at the source code level (typically in C-based form). This allows a functional model to be natively compiled onto the host for fastest possible execution of given algorithms. Timing information is added by prior back-annotation of the source code.

In order to accurately capture dynamic data dependencies while minimizing overhead, back-annotation is usually performed at the basic block level [19], [20]. Several nearly identical approaches [21], [22] use a standard compiler frontend to first bring the code down to an intermediate representation. This allows typical source-level compiler optimizations to be accurately considered. In all cases, back-annotation is based on static emulation of basic block execution on a timing model of the target processor. In some cases, this may be as complex as using the target tool-chain to compile and simulate each block on a cycle-accurate target model. Since back-annotation is performed off-line on a block-by-block basis, each block only needs to be analyzed once. As such, complexities of timing estimation are deferred to a pre-processing step while time-critical simulation of repeated block executions is performed with statically back-annotated values. Several host-compiled approaches also support hybrid static and dynamic timing models and back-annotation, either by including dedicated simulation models of critical dynamic micro-architecture features, such as caches or branch predictors [19], [21], [23], or by toggling between host-compiled and ISS-based models dynamically at simulation time [24], [25].

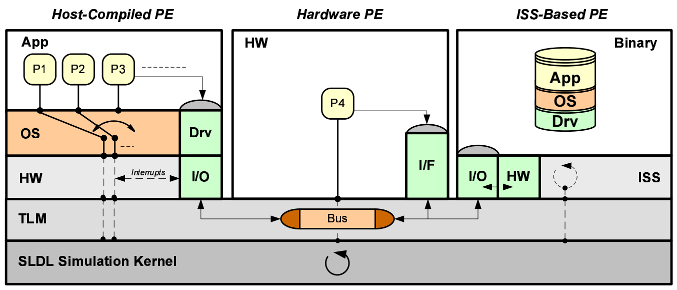

Fig. 4. Platform modeling.

II. Platform Modeling

To assemble a complete platform model, computation and communication descriptions at specific levels are combined into PE models that are integrated into a common platform simulation framework. Most commonly, the simulation framework is provided through a standardized TLM backplane [26] that sits on top of a system-level design language (SLDL), such as SystemC [27]. To allow integration and simulation of application computation and communication at a certain level of abstraction, a platform model has to provide the necessary execution environment on top of the basic SLDL kernel and TLM backplane[1]. Specifically, this requires processor and bus models that abstract all target functionality and timing not explicitly included in the description of the application implementation itself (Fig.4).

In a traditional ISS-based PE model (Fig.4 on the right), an instruction set simulator emulates execution of a cross-compiled target binary that includes instruction-level implementations of all application, OS and driver code. The ISS kernel is integrated into the overall TLM backplane through a thin wrapper that relays all I/O and interrupts between the simulated processor and the external TLM environment [28], [29]. Internally, the ISS kernel can include a separate scheduler that simulates processor-internal parallelism, e.g. when modeling a SMP CPU with multiple cores. In the process, the ISS can also integrate and co-simulate models of processor peripherals or other external hardware. Note that most ISS frameworks also allow for stand-alone simulation of complete multi-processor platforms exclusively assembled on top of the proprietary ISS kernel [30].

By contrast, in a host-compiled PE model (Fig.4 on the left), an application implementation is provided in the form of C code for each process. C code is back-annotated with target-specific execution timing and simulated on top of abstract models of the operating system (OS) and processor hardware (HW) that integrate into the TLM and SLDL backplane [31]. Some of the earliest host-compiled approaches were centered around accurate models of OS effects [32], [33]. On top of the concurrency model provided by the basic SLDL kernel, the OS model simulates dynamic real-time scheduling of application processes on a given number of processor cores. More

[1]A common simulation backplane thereby allows for mixed-level co-simulation of PE models at different abstraction levels.

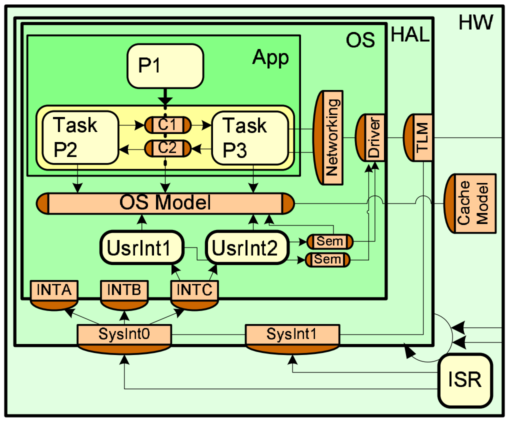

Fig. 5. Host-compiled processor model.

```
...
a[i][j] += sum;
/* __BA: enqueue cache access */
__alist[__idx] = A_BASE + 4*(i*A_WID+j);
/**/

...

/* __BA: accumulate delays */
cache_delay = cache.upd(__alist, __idx);
os.wait(BASE_DELAY + cache_delay);
/**/
```

Fig. 6. Application code back-annotation.

recently, OS modeling approaches have been extended into full processor models that include abstract descriptions of the processor hardware [34], [35]. Hardware models integrate with OS models to accurately describe hardware timing effects, e.g. due to processor suspension and interrupt handling [36]. In addition, processor hardware models interface with the external TLM environment. Depending on the level of communication abstraction, the application may provide implementations of networking, driver and interrupt handler code, while any remaining external communication is abstracted into calls to the corresponding bus TLM channel interface (e.g. at the network or protocol level)[2].

Lastly, note that a host-compiled model of a software processor is essentially the most general case that covers any type of PE. Specifically, a model for a custom hardware PE (Fig.4 middle) becomes a special case of a host-compiled PE that does not include OS and hardware interrupt models. In a hardware PE model, the given application code is back-annotated with timing information and co-simulated with the rest of the system through a SLDL-based description of the bus interface implementation that integrates into the TLM backplane.

III. Processor Modeling

In the following, we will present some more details about host-compiled modeling of processors to be integrated into an overall TLM-based platform simulation environment. Fig.5 shows an example of a host-compiled processor model [34]. The model is constructed in a layer-based fashion following the general organization of application and modeling levels outlined in the previous section. This allows processor models at varying feature and abstraction levels to be easily composed and generated.

The *Application* is described as a set of back-annotated, C-based processes that describe basic algorithmic functionality and execution timing. Application processes run on top of a model of the operating system that adds realizes dynamic scheduling as described previously. In addition, the *OS* layer also includes code for any networking and driver stacks provided with the application. A Hardware Abstraction Layer (*HAL*) then provides canonical interfaces for translating all external communication into appropriate calls to the TLM-based platform environment. The HAL also includes implementation code or models for low-level interrupt handlers. Finally, the *HW* layer contains accurate models of logic for processor suspension and handling of interrupt service requests in conjunction with an externally provided interrupt controller model.

In a hybrid between static and dynamic timing back-annotation, the processor model may include runtime models of advanced dynamic microarchitecture features, such as caches or branch predictors. Fig.5 shows an example of a behavioral cache model integrated into the processor hardware layer [23]. As part of the back-annotation process, application code is instrumented at the basic block level with both best-case execution time estimates and additional calls to the cache model (Fig.6). The cache model is driven by information about cache lines being accessed in each block, which is obtained by deriving base addresses of each variable in the code from the symbol table of the application compiled into a target binary[3]. Internally, the purely behavioral cache model maintains line tags according to a specified associativity and replacement policy. On each call, the cache model updates its internal state and returns the number of cache misses. In this way, the cache model can simulate effects of data locality based on the dynamic code execution sequence. Cache miss information is dynamically added to the static timing annotations in order to simulate the effect of miss penalties. Results show that integration of models of dynamic target behavior significantly increases accuracies. However, simulation overhead is similarly incurred with each additional dynamic feature, up to the point where a complete cycle-accurate timing model is run next to the host-compiled functional simulation.

[2]Note that if there is a mismatch in communication abstractions of different PEs, the TLM backplane can provide necessary transactors that transparently translate between different levels and interfaces.

[3]In case of array accesses, address information is computed at runtime based on the array index and the array base address.

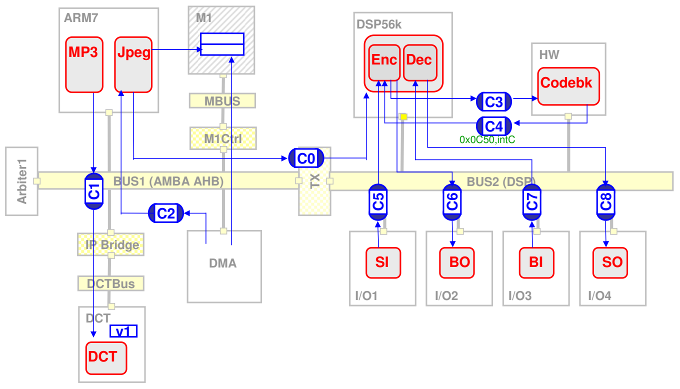

Fig. 7. Cellphone baseband MPCSoC example.

IV. Case Study

We have applied host-compiled platform modeling concepts to a cellphone MPCSoC that combines an MP3 decoder and a JPEG encoder running on an ARM7 processor with a GSM voice encoder/decoder running on a Motorola DSP6600 (Fig.7) [34]. The cellphone system was modeled at various levels of computation and communication abstraction. On the computation side, models with and without inclusion of the OS layer were constructed. On the communication side, N- and P-TLMs were compared against a pin-accurate model (PAM) at the RTL level. Subsystems were exercised with 55 MP3 frames, 30 116×96 pictures and simultaneous encoding/decoding of 163 frames of speech, respectively. Accuracy was measured as the average absolute error in simulated frame/picture delays when compared to a cycle-accurate ISS-based reference model.

Fig.8 shows speeds and accuracies for various models in a simulation of 3 s real time with 180 million DSP and 300 million ARM cycles. For single processor systems, simulation speeds of 2000 MIPS peak and 600 MIPS sustained can be achieved. For the full system cellphone simulation, the P-TLM runs at 300 MIPS. To isolate modeling from estimation errors, back-annotation of average execution timing at the function level was performed using perfect ISS measurements. Resulting timing errors of models at various levels range from 12.5% down to less than 3%. In all cases, however, models exhibit 100% fidelity across various explored architectures.

In general, results confirm expected speed and accuracy tradeoffs with increasing abstraction. On the computation side, models at the highest application level are grossly inaccurate due to a mismatch in concurrency models. Moving to the function level with subsequent inclusion of OS models significantly increases accuracy with practically no overhead. Further refinement of communication from messages down to packet or protocol levels (paired with corresponding inclusion of bus drivers and interrupt models) results in additional accuracy gains at moderately decreased simulation speed. Finally, switching to RTL modeling of communication and finally computation leads to exponential growth in simulation times with only minor reduction in modeling errors.

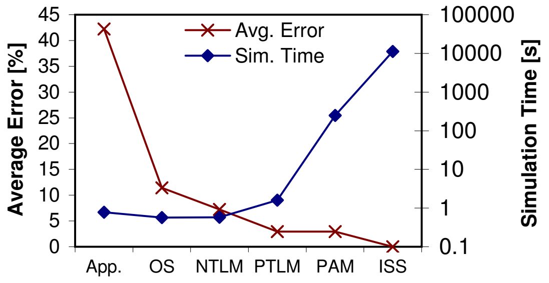

Fig. 8. Cellphone MPCSoC modeling results.

V. Summary and Conclusions

Platform models are at the core and form the basis of any system design methodology. Following a layer-based approach, host-compiled models at various levels of abstraction can be constructed in a systematic manner. Experimental results on an industrial-strength case study demonstrate the benefits of host-compiled modeling at the N- or P-TLM level. Models are fast and accurate for rapid, early design space exploration and software development. However, all models are Pareto-optimal and none clearly outperforms any other. Thus, ideally a variety of models is desired to support a design flow with gradual design space pruning while successively converging down to more and more accurate solutions.

Traditionally, system models are manually written, which is a tedious, error-prone and time-consuming process. This makes it infeasible to explore a large number of design alternatives at varying levels within a given time-to-market window. However, based on sound layer-based model definitions, tools for automatic model generation can be developed. Furthermore, well-defined TLMs and PAMs support automatic synthesis down to final hardware and software implementations at the RT or ISS level. Using such tools [37], all models shown for the cellphone example were generated within seconds. Furthermore, tools can easily create models at varying levels of abstraction. If done manually, writing and debugging of equivalent models would take months. Overall, automatic model generation paired with fast and accurate simulation are the key to unlocking significant productivity gains in system-level design.

Acknowledgments

The author would like to thank all former colleagues at the Center for Embedded Computer Systems (CECS) at UC Irvine, where many of the concepts presented here were originally developed.

References

[1] D. Chiou, D. Sunwoo, J. Kim, N. Patil, W. Reinhart, E. Johnson, J. Keefe, and H. Angepat, “FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators,” in *MICRO*, Dec. 2007.
[2] G. C. Buttazzo, *Hard Real-Time Computing Systems*. Kluwer, 1999.
[3] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich, “SPI — a system model for heterogeneously specified embedded systems,” *IEEE TVLSI*, 2002.

[4] E. Wandeler, L. Thiele, M. Verhoef, and P. Liverse, "System architecture evaluation using modular performance analysis: A case study," *Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 649–667, Oct. 2006.

[5] C. Erbas, A. D. Pimentel, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP JES*, vol. 2007, no. 82123, 2007.

[6] M. Streubühr, C. Haubelt, and J. Teich, "System level performance simulation for heterogeneous multi-processor architectures," in *Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*, Jan. 2009.

[7] F. Bellard, "QEMU, a fast and portable dynamic translator," in *USENIX*, 2005.

[8] M. Reshadi, P. Mishra, and N. Dutt, "Hybrid-compiled simulation: An efficient technique for instruction-set architecture simulation," *ACM TECS*, vol. 8, no. 3, Apr. 2009.

[9] G. Braun, A. Nohl, A. Hoffmann, O. Schliebusch, R. Leupers, and H. Meyr, "A universal technique for fast and flexible instruction-set architecture simulation," *IEEE TCAD*, vol. 23, no. 12, pp. 1625–1639, 2004.

[10] W. S. Mong and J. Zhu, "DynamoSim: a trace-based dynamically compiled instruction set simulator," in *ICCAD*, San Jose, CA, 2004.

[11] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *CODES+ISSS*, Newport Beach, CA, Oct. 2003.

[12] F. Ghenassia, *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[13] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based communication architectures at the CCATB abstraction," *ACM TECS*, vol. 7, no. 2, pp. 22:1–22:32, Feb. 2008.

[14] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia, "IPSIM: SystemC 3.0 enhancements for communication refinement," in *DATE*, Munich, Germany, Mar. 2003.

[15] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM TECS*, vol. 8, no. 1, pp. 4:1–4:29, Dec. 2008.

[16] A. Gerstlauer, D. Shin, R. Dömer, and D. D. Gajski, "System-level communication modeling for network-on-chip synthesis," in *ASPDAC*, Shanghai, China, Jan. 2005.

[17] G. Schirner and R. Dömer, "Result Oriented Modeling a Novel Technique for Fast and Accurate TLM," *IEEE TCAD*, vol. 26, no. 9, pp. 1688–1699, Sep. 2007.

[18] R. S. Khaligh and M. Radetzki, "Efficient parallel transaction level simulation by exploiting temporal decoupling," in *IESS*. Langenargen, Germany: Springer, Sep. 2009.

[19] J. Schnerr, O. Bringmann, A. Viehl, and W. Rosenstiel, "High-performance timing simulation of embedded software," in *DAC*, Anaheim, CA, Jun. 2008.

[20] R. Dömer, "Transaction level modeling of computation," Center for Embedded Computer Systems, University of California, Irvine, Tech. Rep. CECS-06-11, Aug. 2006.

[21] Z. Wang and A. Herkersdorf, "An efficient approach for system-level timing simulation of compiler-optimized embedded software," in *DAC*, San Francisco, CA, Jul. 2009.

[22] Y. Hwang, S. Abdi, and D. Gajski, "Cycle approximate retargettable performance estimation at the transaction level," in *DATE*, Munich, Germany, Mar. 2008.

[23] A. Pedram, D. Craven, and A. Gerstlauer, "Modeling cache effects at the transaction level," in *IESS*. Langenargen, Germany: Springer, Sep. 2009.

[24] L. Gao, K. Karuri, S. Kraemer, R. Leupers, G. Ascheid, and H. Meyr, "Multiprocessor performance estimation using hybrid simulation," in *DAC*, Anaheim, CA, Jun. 2008.

[25] M. Krause, D. Englert, O. Bringmann, and W. Rosenstiel, "Combination of instruction set simulation and abstract RTOS model execution for fast and accurate target software evaluation," in *CODES+ISSS*, Atlanta, GA, Oct. 2008.

[26] *Transaction Level Modeling Library, Release 2.0*, Open SystemC Initiative (OSCI), Jun. 2008.

[27] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Kluwer, 2002.

[28] M. Gligor, N. Fournel, and F. Petrot, "Using binary translation in event driven simulation for fast and flexible MPSoC simulation," in *CODES+ISSS*, Grenoble, France, Oct. 2009.

[29] F. Fummi, M. Loghi, M. Poncino, and G. Pravadelli, "A co-simulation methodology for hw/sw validation and performance estimation," *ACM TODAES*, vol. 14, no. 2, pp. 23:1–23:32, Mar. 2009.

[30] L. Benini, D. Bertozzi, A. Bogoliolo, F. Menichelli, and M. Olivieri, "MPARM: Exploring the multi-processor SoC design space with SystemC," *Journal of VLSI Signal Processing*, vol. 41, no. 2, pp. 169–184, 2005.

[31] T. Kempf, M. Dörper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout, "A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms," in *DATE*, Munich, Germany, Mar 2005.

[32] A. Gerstlauer, H. Yu, and D. D. Gajski, "RTOS modeling for system level design," in *Design, Automation and Test in Europe: The Most Influential Papers of 10 Years DATE*, R. Lauwereins and J. Madsen, Eds. Springer, 2008.

[33] H. Posadas, J. A. Adamez, E. Villar, F. Blasco, and F. Escuder, "RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model," *Design Automation for Embedded Systems*, vol. 10, no. 4, Dec. 2005.

[34] G. Schirner, A. Gerstlauer, and R. Dömer, "Fast and accurate processor models for efficient MPSoC design," *ACM TODAES*, vol. 15, no. 10, Feb. 2010.

[35] A. Bouchhima, I. Bacivarov, W. Yousseff, M. Bonaciu, and A. Jerraya, "Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration," in *ASPDAC*, Shanghai, China, Jan. 2005.

[36] H. Zabel, W. Müller, and A. Gerstlauer, "Accurate RTOS modeling and analysis with SystemC," in *Hardware-dependent Software: Principles and Practice*, W. Ecker, W. Müller, and R. Dömer, Eds. Springer, 2009.

[37] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-Chip Environment: A SpecC-based framework for heterogeneous MPSoC design," *EURASIP JES*, vol. 2008, no. 647953, p. 13, 2008.