Platform Modeling for Exploration and Synthesis

Andreas Gerstlauer

Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712
e-mail: gerstl@ece.utexas.edu

Gunar Schirner

Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
e-mail: schirner@ece.neu.edu

Abstract— Ever increasing complexity and heterogeneity of system platforms drive the need for a move to higher levels of abstraction accompanied by corresponding design automation tools. The basis for any automated flow are well-defined design models. In this paper, we present an overview and taxonomy of platform modeling at various levels. Experiments demonstrate the benefits of fast yet accurate intermediate models at varying levels for rapid, early design space exploration. Furthermore, paired with automatic model generation and hardware/software synthesis, an automated path from specification to implementation becomes possible.

I. Introduction

Driven by ever increasing application demands and technological advances that allow us to put complete Multi-Processor and Multi-Core Systems on a Chip (MPCSoCs), design complexities continue to grow exponentially. System-level approaches promise to tackle the complexity crisis by raising the level of abstraction. While often misunderstood as traditional high-level synthesis of a single hardware unit, true system-level design is a unified process of implementing a system application on a system architecture spanning hardware and software processors interconnected by a network of busses or other communication media.

The ultimate goal is to apply design automation techniques to this process. The first approaches to appear at any level are always modeling and simulation solutions. Models allow designers to validate decisions, explore the design space and reason about properties before a system is built. At each step of the design flow, models provide an abstract view of a design, representing essential aspects while hiding others that are not relevant or not yet known. A key aspect is thereby what features to abstract and how many models to provide.

There are several approaches for modeling of systems both at the application and the architecture level. At the application level, capturing and validation of algorithmic functionality is often based on formalized Models of Computation (MoCs) [24, 30]. MoCs are the basis for many commercial, domain-specific modeling and simulation environments, such as Matlab/Simulink or LabView. Academic research, on the other hand, has been focused on ways to relate different MoCs, e.g., through co-simulation [33, 23, 2, 12] or translation [7, 43]. At the architecture level, C-based System-Level Design Languages (SLDLs) are widely used to assemble virtual prototypes and co-simulate components under a common backplane [28, 21]. Furthermore, to facilitate model integration and exchange, there are several attempts at standardization of IP interfaces in extended netlist form [22, 41]. In all cases, simulation-centric approaches at the application or architecture level enable the horizontal integration of various components or application domains. However, approaches for vertical integration through synthesis and verification are lacking.

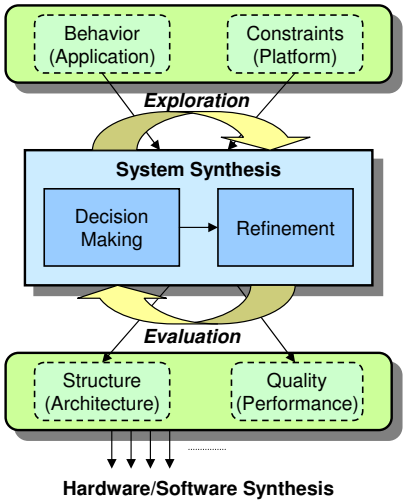


Fig. 1. Electronic system-level (ESL) design.

Fig.1 depicts the requirements of an ESL synthesis flow [17]. System synthesis starts from a specification that describes application behavior and associated constraints, which include the database of available components and, in so-called platform-based design, even complete definitions of predesigned and reusable "platforms" as customizable and parameterizable architecture templates. System synthesis is an iterative process of decision making, refinement and evaluation to perform automated design space exploration. In each iteration, the application is mapped onto a selected platform following given design decisions. Resulting design instances are captured in the form of platform models that describe the structure and quality of the selected architecture. Quality is determined by annotating models with performance metrics. Platform models are evaluated through analysis or simulation in order to drive the next iteration of the exploration. At the end of the system-level synthesis process, final platform models also serve as the input to subsequent hardware and software backend synthesis all the way down to an implementation at the instruction-set and register transfer levels.

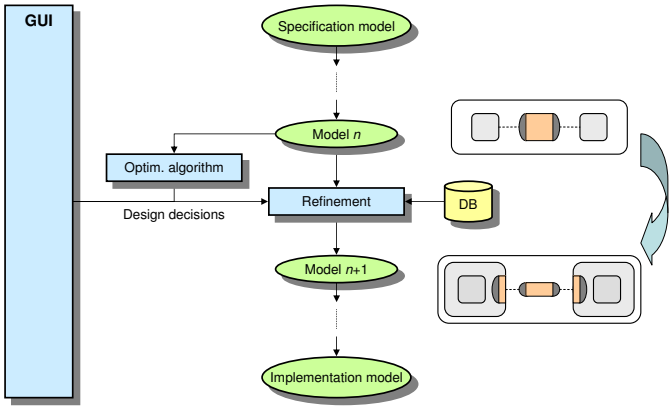


Fig. 2. Synthesis process.

A. Synthesis Process

Generally, the gap between specification and implementation is too large to be bridged in a single step. Therefore, as show in Fig.2, a specification is typically brought down to an implementation through successive, stepwise refinement and exploration. A particular sequence of design steps defines an overall design methodology and within a methodology, design models provide the interfaces in between. Each model therefore serves as both documentation at the output of one step and specification at the input to the next.

Within in each step, refinement tools take given design decisions and insert new layers of functionality into the model. Following fundamental principles for separation of concerns, system-level design is permeated by an orthogonalization of computation and communication aspects [26]. Starting with the application MoCs, systems are described as a set of application processes that communicate via channels. With each refinement step, new layers are inserted by hierarchically decomposing processes and channels into finer and finer spatial and temporal granularity until a fully structural and timing-accurate representation of physical components connected by signals and wires is reached. In the process, some of the layers might be partially inserted out of databases containing various platform component models.

B. MPCSoC Architectures

Recent trends have increasingly led to a shift towards complex multi-processing platforms beyond a single CPU [46]. Various approaches aim to manage complexities and provide high performance in a scalable fashion through regular, homogeneous structures adapted from general-purpose computing, including Networks-on-Chip (NoCs) for communication or multi-core processor architectures. However, the need for optimizations will continue to push for specialization. Future architectures are likely to be hybrid in nature. Such systems might provide a mix of homogeneous, symmetric and heterogeneous, asymmetric multi-processing. We can generally define an MPCSoC to be a hierarchical composition of shared-memory multi-core processors that are components in an overall distributed multi-processor system[1]. Furthermore, communication structures can consist of bus-based subsystems that are interconnected via a homogeneous or heterogeneous back-bone NoC. All in all, complexity and heterogeneity as determined by the large numbers and wide variety of components are the main challenges in synthesizing applications into current and future MPCSoC architectures.

In this paper, we will present requirements and solutions for MPCSoC platform modeling in support of an automated system-level exploration and synthesis flow. In Section II, an overview of concepts and techniques for computation and communication modeling is provided. Furthermore, we describe a taxonomy of system models at various levels of detail and abstraction. Section III demonstrates some of the resulting trade-offs and benefits as applied to an industrial-strength cellphone design example. Finally, the paper concludes with a summary and outlook in Section IV.

II. Platform Modeling

Platform models are at the core of the system design process. In order to apply algorithms and tools, models must be machine-readable. Furthermore, models should be executable for fast yet accurate simulation-based evaluation of functionality and design quality. For these reasons, system models are usually captured in an event-driven SLDL such as SpecC [16] or SystemC [21]. However, to support synthesis and verification, corresponding tools must be able to reason about the meaning of objects and their compositions in a model. Therefore, vertical integration requires well-defined and unambiguous semantics of models on top of basic execution semantics. In the following, we will outline various concepts for sound modeling of computation and communication at various levels and across a wide variety of target architectures [14].

A. Communication Modeling

With communication and component integration being at the core of and increasingly dominating most platforms, Transaction-Level Modeling (TLM) has become tremendously popular and almost universally accepted as a vehicle for acceleration of platform design and validation. TLM allows orders of magnitude faster simulations by abstracting away pin- and wire-level details with little to no loss in accuracy [5, 20]. Over the years, many different TLM styles and levels have been developed [38, 32, 8], and TLM concepts have been applied to modeling of various popular bus architectures [6].

More recently, there have been efforts to standardize TLM syntax on top of existing SLDLs, such as the SystemC TLM2.0 standard currently undergoing IEEE certification [31]. A standardized syntax allows for easier exchange of component, IP and system models across users and tools. However, while the TLM2.0 standard defines some guidelines for so-called loosely timed and approximately timed coding styles, model semantics, even for issues of actual timing granularity, are not well defined. For example, data granularity and interpretation of generic transaction payloads are intentionally left open. In general, standards often provide flexibility to ensure wide applicability. But this comes at the expense of limited interoperability, in some sense defeating the purpose of standardization.

Furthermore, to support synthesis and verification, models have to unambiguously define timing and data granularity issues, which typically go hand-in-hand. Reasoning about se-

[1] Where each processor can have multiple cores that share a common subset of resources such as bus interfaces, memories or the operating system.

TABLE I
PROTOCOL LAYERS.

Impl.	Layer	Semantics	Functionality	OSI
	Application	Channels	Computation	**7**
Middleware	**Presentation**	End-to-end typed messages	Data formatting	**6**
	Session	End-to-end untyped messages	Synchronization, multiplexing	**5**
	Transport	End-to-end data streams	Packeting, error correction, flow control	**4**
	Network	End-to-end data packets	Subnet bridging, routing	**3**
Driver	**Link**	Point-to-point logical links	Station typing, synchronization	**2b**
	Stream	Point-to-point control/data streams	Multiplexing, addressing	
HAL	**Media Access**	Shared medium byte streams	Data slicing, arbitration	**2a**
HW	**Protocol**	Media word/frame transactions	Protocol timing	
	Physical	Pins, wires	Driving, sampling	**1**

mantics of communication is not a new problem. In fact, we can adapt the ISO/OSI 7-layer model developed for this purpose in the networking community (Table I) [18]. In the process, ISO/OSI concepts are tailored specifically to SoC/NoC requirements, e.g. by splitting or combining layers to reflect hardware/software boundaries[2].

To support modeling and design with stepwise, successive refinement, functionality is divided into layers based on separation of concerns, grouping of common functionality, dependencies across layers, and support for early validation of critical issues through rapid and efficient design space exploration by humans or automated tools. Layers are stacked on top of each other where a layer provides services to the next higher layer by using the services provided by the layer below. At its interface, each layer provides services for establishing communication channels and for performing transactions over those channels. Each layer is implemented in the form of hardware, firmware or middleware as part of final system.

Semantics of transactions and channels vary from layer to layer. Therefore, each layer immediately corresponds to a certain level of abstraction. Furthermore, with each new layer inserted and implemented in the design, we can define a new class of models where communication channels connecting components at the level of a certain layer abstract away and encapsulate functionality of all layers below (Fig.3). At the highest specification level, application processes communicate via abstract message-passing primitives. At the lowest level, components in a Pin- and Bus Cycle-Accurate Model (PAM/BCAM) are interconnected through physical pins and wires. In between, we can define TLMs at various levels with communication being described in the form of TLM channels realizing a particular layer interface. Note that almost all existing TLM approaches, including the SystemC TLM2.0 standard, are aimed at solutions at the protocol layer or below.

Layer definitions have been successfully adopted for modeling and synthesis of bus-based MPCSoC communication architectures [18]. Following a layer-based approach, tools can automatically generate application- and target-specific protocol stacks that realize semantics of desired specification-level communication primitives over a given communication architecture. Combined with similar layer-based approaches for computation, well-defined modeling concepts and refinement techniques will build the basis for a wide variety of system models at varying levels of abstraction.

[2]Note that the ISO/OSI model was never intended as a blueprint for code. In implementations, layers should be merged and optimized across boundaries.

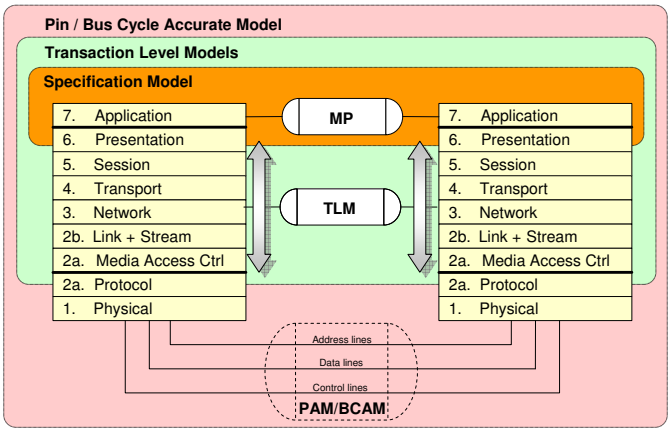


Fig. 3. Communication models [14].

B. Computation Modeling

On the computation side, the basic system component to perform computation is a processor. We can observe that there is no fundamental difference between a custom hardware processor, a programmable software processor or any of the reconfigurable or Application-Specific Instruction Set Processor (ASIP) solutions in between. In all cases, computation is implemented by a controller and a datapath [1]. How much of the datapath is specialized and how much of the control is hardwired defines a spectrum of processor implementation choices. In the most general case, control is fully programmable and the processor executes universal instructions from a stack of firmware, middleware and application software.

Traditionally, processors are validated using accurate RTL or microarchitecture simulations. For software processors, Instruction Set Simulator (ISS) models are commonly used and integrated into an overall platform co-simulation environment [3, 13, 45]. Such models can provide functional and timing simulation on a host at very fine, instruction- and potentially cycle-accurate granularity. Several commercial solutions aim to provide fast and cycle-approximate ISS-based virtual platform prototypes, either building on standard TLM concepts [9] or in the form of proprietary, CPU-centric simulation frameworks [42, 44]. However, especially when high timing accuracy is desired, ISS approaches are slow and do not scale well in multi-processor and multi-core contexts. Due to their nature, there is a significant overhead to co-simulate multiple processor cores on a cycle-by-cycle basis in a full system simulation.

Therefore, ISS-based models are not sufficient to match the needs for rapid, early validation and debugging at the sys-

TABLE II
PROCESSOR LAYERS.

Target approx. computation timing	Appl.
Task mapping, dynamic scheduling, task communication, middleware	OS
Interrupt handlers, low level drivers	HAL
HW interrupt handling & scheduling	HW-TLM
Pin-/cycle-accurate communication	HW-BFM
Cycle-accurate computation	BFM - ISS

tem level. Similar to the approach taken with TLMs for communication, computation models at higher levels of abstraction are needed. To that effect, so called host-compiled [40, 35, 4, 25, 10] or hybrid [15, 29] software simulation approaches have recently garnered widespread interest. The goal is thereby to combine fastest possible host execution with the accuracy of an ISS. Applications in the form of C code are compiled to natively emulate functionality on the simulation host. C code is wrapped into SLDL modules and back-annotated with estimated or measured timing information. Furthermore, application wrappers are augmented with lightweight models of middleware, firmware and processor hardware to describe, with little to no overhead, all relevant operating system, microarchitecture and other effects.

Again, to support not only simulation but automated synthesis and verification, host-compiled processor models at various levels of detail and accuracy have to be constructed in a well-defined and sound manner. In reality, application software usually runs on top of a Real-Time Operating System (RTOS), which in turn sits on top of the actual processor hardware providing physical bus interfaces and interrupts for external communication. All of these affect overall functional and timing behavior. For example, interrupt handling and can introduce a significant processor load [47]. Hence, all such effects need to be considered and modeled in order to provide an accurate (but fast) model of the software execution environment.

Following a similar approach as in the case of communication (Section A), we can identify processor layers at varying feature levels (Table II) [39]. Basic algorithmic functionality, including execution timing, is part of the application. The operating system layer adds dynamic scheduling and bus drivers. A Hardware Abstraction Layer (HAL) provides canonical interfaces for accessing external communication hardware from the OS and application side. Finally, the hardware layer includes the physical interrupt logic and bus interfaces in TLM or pin-accurate (also called Bus-Functional Model, BFM) form. We can contrast these high-level processor layers with a traditional pin-accurate RTL or ISS model that describes cycle-by-cycle behavior at the microarchitecture level. Note that custom hardware processors in which OS and HAL are empty are special instances of such a general processor model.

Based on these definitions, we can construct processor models in a layer-based fashion in the same way as for communication (Fig.4) [39]. The *Application* is described as a set of processes that run on top of a model of the operating system [19]. The *OS* layer also includes models of the middleware stacks for communication with other processors in the system. A *HAL* provides low-level bus drivers and interrupt handlers. It is the lowest layer implemented in software, and its border marks the boundary of the HW/SW interface. Finally, the *HW* layer contains accurate model of external bus communication

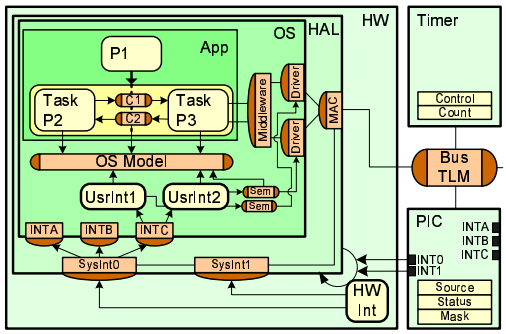


Fig. 4. Processor model.

and interrupt logic. To increase accuracy, it may also include models of advanced dynamic microarchitecture features, such as caches or branch predictors [34]. Bus interfaces are described at the protocol level in TLM or BFM form. Besides a number of HW cores, the model can include any interrupt controllers and other peripherals (such as timers) immediately associated with the processor. All in all, high-level processor models provide a complete model of computation running in its execution environment.

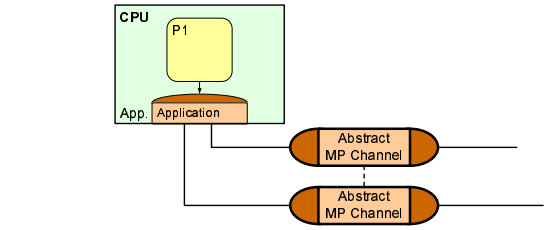
C. System Modeling

Combining approaches for modeling of computation and communication, a complete platform model emerges. Communication layers are thereby implemented inside computation layers based on inherent relationships and dependencies of the former on services of the latter. Depending on how many layers are inserted, system models at varying levels of detail can be constructed (Fig.5).

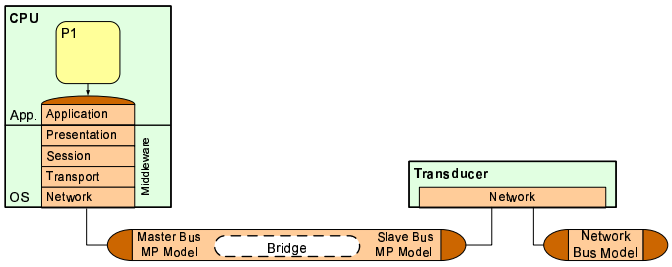
A specification model (Fig.5(a)) contains application layers for both computation and communication. It describes the system as a virtual architecture in which application processes, variables and channels are mapped onto PEs and memories of a system platform. Application layers annotate processes with timing and resolve synchronization, storage and complex channels down to a set of canonical communication primitives for abstract message-passing, memory interfaces and events. As such, a specification model can be used to rapidly validate the feasibility of a given application-to-platform mapping.

A network TLM (Fig.5(b)) inserts OS layers and refines communication to include middleware down to the level of link layer packet transfers. In the process, models of bus gateways and routers (so-called transducers) are inserted. Insertion of bridges that transparently connect busses directly at the protocol level is deferred and bridging is encapsulated as part of bus channel models. Overall, a N-TLM accurately reflects the overall topology of the communication network. Universal, abstract bus channels provide models of logical, point-to-point packet transfers and memory accesses in each segment of the network. Bus channels can include estimated timing for each individual transfer based on the bus bandwidth in relation to the size of the transferred data block. Since bus traffic is simulated at a coarse, block-level granularity, simulations are fast but results are inaccurate in terms of timing effects resulting from sharing of underlying media.

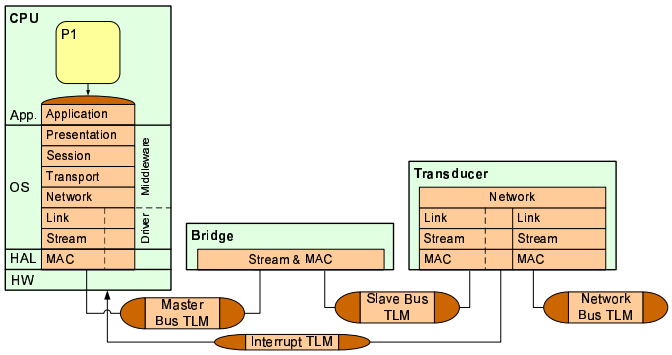
The protocol TLM (Fig.5(c)) further refines computation down to the hardware and communication down to the media



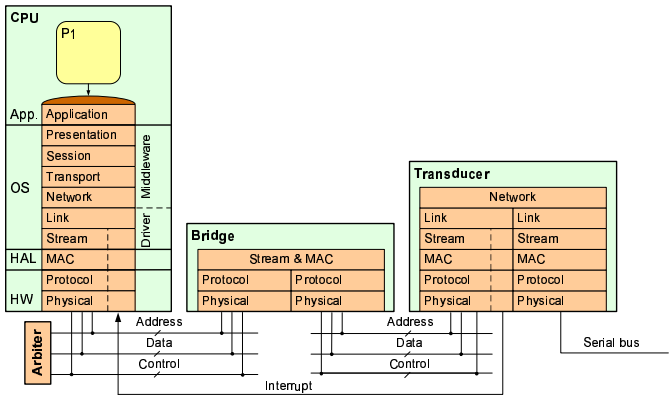
(a) Specification



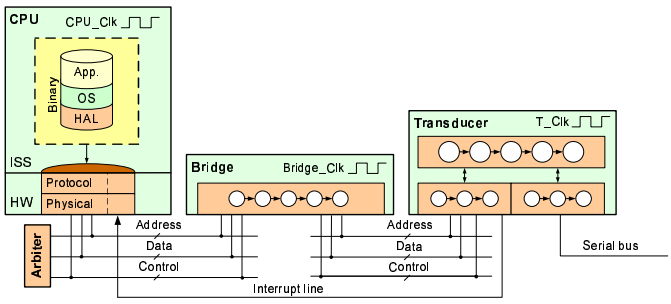
(b) Network TLM (N-TLM)



(c) Protocol TLM (P-TLM)



(d) Pin-/Bus Cycle-Accurate Model (PAM/BCAM)



(e) Cycle-Accurate Model (CAM)

Fig. 5. System models [14].

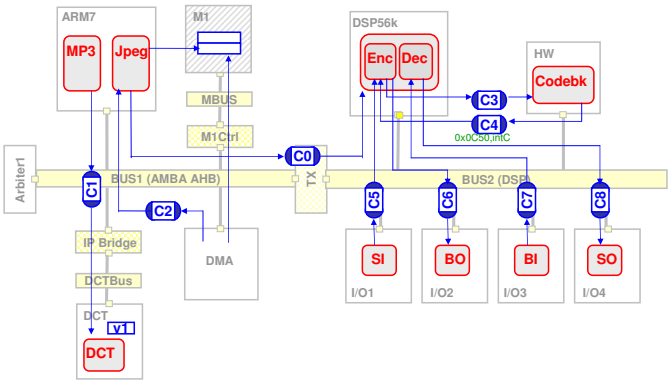


Fig. 6. Cellphone baseband MPCSoC example.

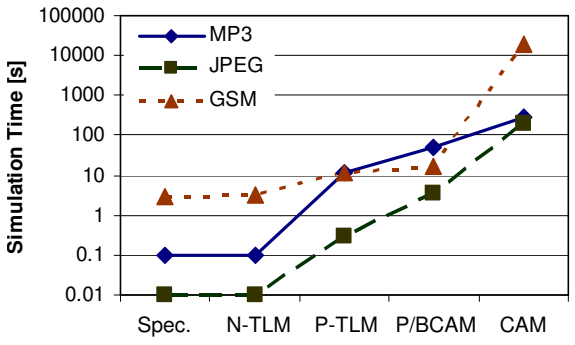
access (MAC) layer. All drivers are implemented to a level of bus reads/writes and interrupt handlers. In a P-TLM, all software is known and CPU models can be easily exchanged for equivalent ISS models running corresponding binary code. Bridge models are inserted and components communicate over models of physical bus media at the level of target protocol transactions. Bus protocol channels can accurately model the known protocol timing and arbitration effects. Therefore, the P-TLM provides results comparable to a PAM/BCAM but at significantly faster simulation speeds.

A Bus Cycle-Accurate Model (BCAM, Fig.5(d)) is the final result of the system design process and implements all layers of computation and communication functionality. Compared to a P-TLM, a BCAM adds protocol and optionally physical communication layers. A BCAM refines all bus interfaces and bus media down to a structurally accurate assembly of component BFMs connected by busses. This includes any components that are necessary as part of the bus structure, such as arbiters or muxes. Note that a BCAM may or may not be pin-accurate. A Pin-Accurate BCAM (PAM/BCAM) describes the complete pin- and wire-level system netlist as a basis for further layout and synthesis. On the other hand, if physical layers are not included, components communicate through transactions at the level of abstracted address, data and control cycles. In both cases, the BCAM provides the cycle-by-cycle behavior needed for accurate modeling of advanced bus features such as split transactions or preemption.

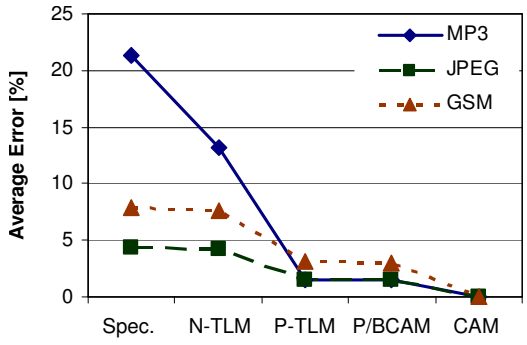
A BCAM is the basis for hardware and software implementation of system components. As a result of the backend synthesis process, a fully Cycle-Accurate Model (CAM) is produced (Fig.5(e)). In a CAM, behavioral component models are replaced with their cycle-accurate RTL or ISS representations. On the software side, ISS models of individual processors are wrapped and integrated into the overall BCAM framework. On the hardware side, processors and interfaces are replaced with finite state machine models of computation and communication protocols. The CAM is the final result of the system design process and allows for cycle-accurate full-system validation as the final signoff before further logic and physical synthesis.

III. Modeling Tradeoffs

To evaluate various system modeling options, we have applied modeling concepts to a cellphone MPCSoC that combines an MP3 decoder and a JPEG encoder running on an



(a) Speed



(b) Accuracy

Fig. 7. Subsystem simulation results.

ARM7 processor with a GSM voice encoder/decoder running on a Motorola DSP6600 (Fig.6). Fig.7 shows performance and accuracy results for simulations of individual subsystems at various levels [39]. 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 the CAM. Results shows typical tradeoffs where accuracy increases linearly while simulation speed drops exponentially with increasing detail at lower levels.

Fig.8 shows speeds and accuracies for various models of the complete, combined cellphone system 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 TLM runs at 300 MIPS. To isolate modeling from estimation errors, back-annotation of execution timing at the application 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. TLMs at various levels support these tradeoffs for different use cases. A P-TLM can be as accurate as a BCAM for systems that do not utilize busses with complex arbitration schemes. In cases where there is no arbitration at all, such as busses with single masters only, even the N-TLM can provide the same accurate feedback at much improved speeds. In the end, simulations with arbitrary combinations of behavioral and cycle-accurate component models are possible at both the P-TLM and BCAM levels. Such mixed-level simulations allow for validation of individual component implementations in a faster system context.

Recently, a lot of research has been performed on modeling techniques to improve simulation accuracy and speed. Many of these techniques, such as concepts based on temporal decoupling [31, 27] or on timing prediction with subsequent correction [36, 37], are general in the sense that they can be applied to models of both computation and communication and at varying levels of granularity. Under many general conditions and in many common situations, such techniques allow a significantly shift in speed/accuracy tradeoffs, e.g. by improving speeds without a loss in accuracy.

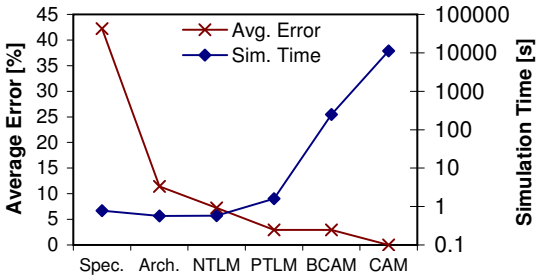


Fig. 8. Cellphone MPCSoC modeling results.

IV. Summary and Conclusions

Platform models are at the core and form the basis of any system design methodology. Following a layer-based approach, models at various levels of abstraction can be constructed in a systematic manner. Experiments confirm that intermediate system TLMs can simulate fast at reasonable accuracy. However, all models are Pareto-optimal and none clearly outperforms any other. Thus, 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 PAMs/BCAMs support automatic synthesis down to final hardware and software implementations at the CAM level. Using such tools [11], 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, sound model semantics beyond simulation are the key to unlocking corresponding significant productivity gains by enabling rapid, early design space exploration with an automated path to implementation.

Acknowledgments

Many of the insights presented in this paper are the result of countless discussions with members back at the Center for Embedded Computer Systems (CECS) at UC Irvine. We specifically would like to acknowledge the contributions of our co-authors on [14], which provided the material for the summary on platform modeling concepts as presented here.

REFERENCES

[1] Accellera. *RTL Semantics*, Feb. 2001. Version 0.8.

[2] A. Bakshi, V. K. Prasanna, and A. Ledeczi. MILAN: A model based integrated simulation framework for design of embedded systems. In *LCTES*, Snowbird, Utah, June 2001.

[3] L. Benini, et al. MPARM: Exploring the multi-processor SoC design space with SystemC. *Journal of VLSI Signal Processing*, 41(2):169–184, 2005.

[4] A. Bouchhima, et al. Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration. In *ASPDAC*, Shanghai, China, Jan. 2005.

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

[6] M. Caldari, et al. Transaction-level models for AMBA bus architecture using SystemC 2.0. In *DATE*, Munich, Germany, Mar. 2003.

[7] P. Chandraiah and R. Dömer. Code and data structure partitioning for parallel and flexible MPSoC specification using designer-controlled re-coding. *IEEE TCAD*, 27(6):1078–1090, June 2008.

[8] M. Coppola, et al. IPSIM: SystemC 3.0 enhancements for communication refinement. In *DATE*, Munich, Germany, Mar. 2003.

[9] CoWare. Virtual Platform Designer. http://www.coware.com.

[10] R. Dömer. Transaction level modeling of computation. Technical Report CECS-06-11, Center for Embedded Computer Systems, University of California, Irvine, Aug. 2006.

[11] R. Dömer, et al. System-on-Chip Environment: A SpecC-based framework for heterogeneous MPSoC design. *EURASIP JES*, 2008(647953):13, 2008.

[12] J. Eker, et al. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE*, 91(2), Jan. 2003.

[13] F. Fummi, et al. Scalable and flexible cosimulation of SoC designs with heterogeneous multi-processor target architectures. *ACM TODAES*, 14(2):23:1–23:32, Mar. 2009.

[14] D. D. Gajski, et al. *Embedded System Design: Modeling, Synthesis, Verification*. Springer, 2009.

[15] L. Gao, et al. Multiprocessor performance estimation using hybrid simulation. In *DAC*, Anaheim, CA, June 2008.

[16] A. Gerstlauer, et al. *System Design: A Practical Guide with SpecC*. Kluwer, 2001.

[17] A. Gerstlauer, et al. Electronic system-level synthesis methdologies. *IEEE TCAD*, 28(10):1517–1530, Oct. 2009.

[18] A. Gerstlauer, et al. Automatic, layer-based generation of system-on-chip bus communication models. *IEEE TCAD*, 26(9):1676–1687, Sept. 2007.

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

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

[21] T. Grötker, et al. *System Design with SystemC*. Kluwer, 2002.

[22] A. Haverinen, et al. SystemC based SoC communication modeling for the OCP protocol, Oct. 2002. http://www.ocpip.org.

[23] F. Herrera and E. Villar. A framework for heterogeneous specification and design of electronic embedded systems in SystemC. *ACM TODAES*, 12(3), Aug. 2007.

[24] A. Jantsch. *Modeling Embedded Systems and SoCs: Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2004.

[25] T. Kempf, et al. A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms. In *DATE*, Munich, Germany, Mar 2005.

[26] K. Keutzer, et al. System level design: Orthogonolization of concerns and platform-based design. *IEEE TCAD*, 19(12), Dec. 2000.

[27] R. S. Khaligh and M. Radetzki. Efficient parallel transaction level simulation by exploiting temporal decoupling. In *IESS*, Langenargen, Germany, Sept. 2009. Springer.

[28] T. Kogel, R. Leupers, and H. Meyr. *Integrated System-Level Modeling of Network-on-Chip enabled Multi-Processor Platforms*. Springer, 2006.

[29] M. Krause, et al. Combination of instruction set simulation and abstract RTOS model execution for fast and accurate target software evaluation. In *CODES+ISSS*, Atlanta, GA, Oct. 2008.

[30] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich. Models of computation for embedded system design. In A. Jerraya and J. Mermet, eds., *System-Level Synthesis*. Kluwer, 1999.

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

[32] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of bus-based communication architectures at the CCATB abstraction. *ACM TECS*, 7(2):22:1–22:32, Feb. 2008.

[33] H. D. Patel and S. K. Shukla. *SystemC Kernel Extensions for Heterogeneous System Modeling: A Framework for Multi-MoC Modeling and Simulation*. Kluwer, 2004.

[34] A. Pedram, D. Craven, and A. Gerstlauer. Modeling cache effects at the transaction level. In *IESS*, Langenargen, Germany, Sept. 2009. Springer.

[35] H. Posadas, et al. RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. *Design Automation for Embedded Systems*, 10(4), Dec. 2005.

[36] G. Schirner and R. Dömer. Result Oriented Modeling a Novel Technique for Fast and Accurate TLM. *IEEE TCAD*, 26(9):1688–1699, Sept. 2007.

[37] G. Schirner and R. Dömer. Introducing Preemptive Scheduling in Abstract RTOS Models using Result Oriented Modeling. In *DATE*, Munich, Germany, Mar. 2008.

[38] G. Schirner and R. Dömer. Quantitative analysis of the speed/accuracy trade-off in transaction level modeling. *ACM TECS*, 8(1):4:1–4:29, Dec. 2008.

[39] G. Schirner, A. Gerstlauer, and R. Dömer. Fast and accurate processor models for efficient MPSoC design. *ACM TODAES*, to appear, 2010.

[40] J. Schnerr, et al. High-performance timing simulation of embedded software. In *DAC*, Anaheim, CA, June 2008.

[41] SPIRIT Consortium. *IP-XACT, Release 1.4*, Mar. 2008.

[42] VaST Systems. http://www.vastsystems.com.

[43] S. Verdoolaege, H. Nikolov, and T. Stefanov. PN: a tool for improved derivation of process networks. *EURASIP JES*, 2007(75947), 2007.

[44] Virtutech, Inc. Simics. http://www.virtutech.com.

[45] A. Wieferink, et al. A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In *DATE*, Paris, France, Feb. 2004.

[46] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor system-on-chip (MPSoC) technology. *IEEE TCAD*, 27(10):101–1713, Oct. 2008.

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