Virtual Architecture Mapping: A SystemC based Methodology for Architectural Exploration of System-on-Chip Designs

Tim Kogel, Andreas Wieferink,
Rainer Leupers, Gerd Ascheid, Heinrich Meyr
Integrated Signal Processing Systems
Aachen University of Technology, Germany
{kogel, wieferin, leupers, ascheid, meyr}@iss.rwth-aachen.de

Denis Bussaglia, Manoj Ariyamparambath
Intellectual Property and Design Services
Synopsys, Inc.
{denis.bussaglia,manoj.ariyamparambath}
@synopsys.com

***Abstract*— The ever increasing complexity and heterogeneity of modern System-on-Chip designs demands early consideration and exploration of architectural alternatives, which is hardly practicable on the low abstraction level of implementation models.**

In this paper, a system level design methodology based on the SystemC 2.0 library is proposed, which enables the designer to reason about the architecture on a much higher level of abstraction. Goal of this methodology is to define a system architecture, which provides sufficient performance, flexibility and cost efficiency as required by demanding applications like broadband networking or wireless communications. The methodology also provides capabilities for co-simulating multiple levels of abstraction simultaneously. This enables reuse of the simulation environment for functional verification of synthesizable implementation models against the abstract architecture model.

During a cooperation with Synopsys Professional Services, this methodology is applied to the development of a 2.5 GB IP forwarding chip with Quality-of-Service (QoS) support. In this paper we share our experiences of this real-life case-study with special emphasis on the architecture exploration phase, where several architectural alternatives are evaluated with respect to their impact on the system performance.

I. Introduction

One of the most challenging tasks in modern System-on-Chip design projects is to map a complex application onto a heterogeneous platform architecture in adherence to the specified flexibility, performance and cost requirements. Under stringent cost constraints, the required flexibility and performance is best delivered by a heterogeneous platform employing standard as well as application specific programmable architectures and dedicated hardware blocks, which are connected by a sophisticated communication topology. As a result, the designer faces a huge design space and has to compose a system architecture from various kinds of building blocks and communication resources in order to meet the constraints of the specific application.

The traditional design flow comprises only two decoupled phases of textural specification and architecture implementation and is no longer feasible for the design of large heterogeneous systems on a single chip, because quantitative architectural considerations are difficult to estimate on paper, prior to the implementation phase. Systems are either over-engineered, thus impacting the cost, or fail to deliver the expected performance.

Due to the high level of detail inherent to implementation models, so far they can only be optimized locally and system architecture tradeoffs and optimizations are not exploited. For that reason we advocate an intermediate System Level Design phase in the design flow, where the functionality of the system is mapped to the platform architecture in an abstract manner to enable architecture optimizations across heterogeneous computational components.

The following methodical aspects have been identified to cope with requirements of System Level Design:

- orthogonalization of concerns [1] with respect to timing and behavior allows efficient profiling of functional blocks mapped to alternative architectures.
- separation of interfaces and behavior according to the interface based design paradigm motivated by Rowson [2] enables successive communication and structural refinement as well as IP reuse.
- high simulation speed and modeling efficiency is mandatory to handle the high complexity of SoC designs.
- incorporation of hardware semantics like reactivity, concurrency and determinism to express impact of the platform architecture.
- seamless transition from system to gates to avoid long iteration cycles caused by gaps in the design flow
- intuitive visualization comprising system level debugging and performance analysis to enable efficient validation of the system model.

The foundation for our methodology is provided by the SystemC library [3], which is widely considered as the emerging EDA industry standard language for bringing together today's disjunctive worlds of system conceptualization and implementation. Hardware semantics as well as interface based design are already incorporated into the 2.0 release of SystemC and also synthesis tools become commercially available. We have supplemented SystemC with a methodology specific library to enable System Level Design to address all requirements listed above.

The following section discusses related work in the area of system level architecture exploration. After that the projected design flow followed by the technical details of methodology specific library are presented. Section V contains our experi-

ences from an industry cooperation, where the framework has been deployed in the design of a NPU platform. Finally we conclude our approach and give an outlook on future research topics.

II. Related Work

The issues of System Level Design have attracted a growing attention from both university and industry research teams. It is commonly accepted to cope with the growing system complexity by raising the abstraction level of the initial specification to explore architecture tradeoffs decisions.

Starting from a formalized system description, comprehensive co-design frameworks like Polis [4] from UC Berkeley address architecture exploration and synthesis of implementation models. In contrast to such co-design frameworks we do not address automatic synthesis, but jointly model the system functionality and the performance impact of the architecture on the highest possible level of abstraction.

Several design environments, both commercial (e.g. VCC [5]) and academic (e.g. Artemis [6]) also address abstract mapping of functional models to architecture, but since these are based on proprietary languages, the resulting performance evaluation models cannot be reused in later implementation steps. Instead, our methodology provides a seamless path to implementation by using the SystemC language and enables the reuse of the abstract architecture model as a reference for functional verification of later refinement steps [7].

In general, C/C++ based system modeling languages like SpecC [8] and OCAPI-xl [9] and object oriented system conceptualization methodologies with support in tooling like YAML [10] are considered as the most promising vehicles to cope with the ever increasing complexity of SoC designs.

In this context the major contribution of this paper is a system level design methodology, which is based on the SystemC library to accomplish interoperability with other SystemC based environments [11] and to provide a seamless path to implementation [12]. Whereas most of the tools and methodologies listed above intent to automate the design of small and medium range embedded systems, we are focused on the conceptualization and verification of large scale SoC designs with high performance requirements as demanded by wireless and networking communication devices at the edge of silicon feasibility.

III. System Level Design Methodology

System Level Design is all about filling the gap between specification and implementation. In this section we first classify the abstraction levels enabled by System 2.0 and then elaborate on the proposed architecture exploration methodology.

A. Transaction-Level Modeling

SystemC 2.0 has been conceived to realize a *Transaction-Level Modeling (TLM)* style [13], where communication is abstracted from the low-level implementation details of the Register Transfer Level (RTL). The resulting improvement in terms of simulation speed and modeling efficiency enables the system architect to create an executable specification of the complete SoC architecture.

Comm. Accuracy	Timing Accuracy	Data Accuracy	Addressed Design Problems
packet level TLM	untimed-functional	Abstract Data Type	functional specification
	timed-functional	Abstract Data Type	architecture exploration
cycle level TLM	cycle true	bit true	SW development, ISS co-simulation
RTL	cycle true	bit true	HW implementation

TABLE I
Abstraction Levels

As depicted in table I, the TLM paradigm can be further subdivided by applying abstraction w.r.t data and timing accuracy. By that the manifold design problems during the definition of the system architecture can be resolved in the appropriate design step. Note that the entry level depends on the design complexity: small scale and homogenous designs start immediately at RTL whereas cycle level TLM is usually sufficient for the design of medium scale embedded systems.

This paper addresses the conceptualization of large scale heterogenous systems, which need heterogenous computational modules and a customized communication infrastructure to meet performance and cost requirements. Thus we have conceived a *packet level TLM* modeling style for architectural exploration, i.e. the considered data granularity are sets of functionally associated data, which are combined to Abstract Data Types (ADTs).

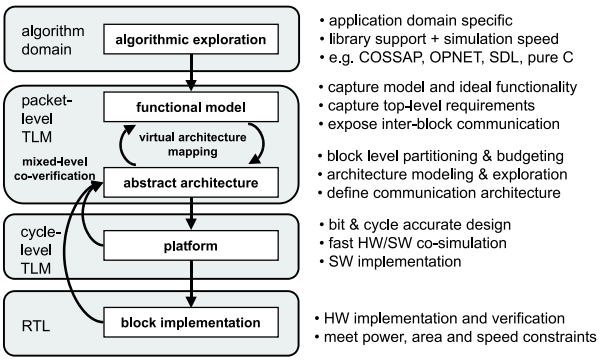

Fig. 1. SoC design flow

All phases in our refinement methodology depicted in figure 1 are based on SystemC and are thus interoperable. However, in this paper we particularly address the design of dedicated Hardware, so we intentionally leave out the discussion of the cycle-level TLM phase, which is mainly dedicated to the integration of Instruction Set Simulators. In the following we describe the sub-phases of our approach, i.e. functional and abstract architectural modeling as well as mixed-level co-verification.

B. Functional model

Compared to the detailed register transfer level (RTL), simulation speed and modeling efficiency can only be improved significantly by modeling the system behavior on a much

higher level of abstraction. In the functional model, the complete system behavior is partitioned only into a small number SoC building blocks instead of scattering the functionality over numerous processes as often required for a synthesizable RTL description. Abstract Data Types (ADT) replace the bit-true data representation of RTL models, such that a whole set of functionally associated data is represented as a single token, as for example an IP packet. Since the system state only changes on the arrival of a new token, we can employ pure reactive communication channels to model the data exchange between the functional blocks. This minimizes the number of the activations in the event-driven SystemC simulation kernel, which effectuates maximum simulation speed.

At the end of the functional modeling stage, the complete system behavior is captured and validated. The simulation speed as well as the modeling efficiency (measured in lines of code) is about two orders of magnitude better compared to the corresponding RTL model, which models the same functionality on a much higher level of architectural detail. The SystemC model is now prepared for the annotation of timing information, which is described in the next section.

C. Virtual architecture mapping

In the next design step, the functional model is mapped to the intended target architecture in order to create a performance model of the resulting system architecture. The mapping is performed virtually by annotating the timing characteristics of the target architecture to the functional model, thus the methodology enables a very fast exploration of different design alternatives. The process of timing annotation is completely orthogonal to the functionality, hence the previously validated functional system behavior is preserved.

The methodology is based on the following observation: for performance profiling purposes, the basic timing characteristics of the target architecture can be expressed by the temporal relationship of consuming, processing and producing ADT tokens.

- Pipelined architectures are able to consume and produce a token every cycle but introduce a static latency, which is determined by the number of pipeline stages.
- Data dependent modules show varying delays until the processing of the actual token is finished. In the case of a cache module for example, the processing delay of a cache read depends on whether the requested data set is in the cache or has to be fetched from the main memory.
- Resource shared modules are afflicted with an initiation interval, i.e. they are blocked for a varying amount of time during a token is processed.

As soon as the estimated timing parameters are annotated to the channels, the simulation results reflect the performance of the final system. A statistical evaluation system is associated with the channels to detect and eliminate architecture bottlenecks very early in the design flow before the time consuming implementation starts. The access statistics gathered by the bus model during a simulation run guide the selection and configuration of a specific on-chip bus.

The key mechanism of virtual architecture mapping is separating behavior inside the functional modules from timing aspects, which are captured by the communication channels. Thus we achieve a threefold orthogonalization of system level design concerns in terms of behavior, interface and timing.

D. Mixed-level Co-verification

After the architecture is finalized, the abstract architecture model is converted into a synthesizable model. Here the same functionality has to be realized at a much lower level of detail according to the refinement steps in the SystemC RTL synthesis guidelines [14]. By using the SystemC synthesis [12], the implementation phase can be performed within the SystemC design environment itself. This enables a smooth transition from the refined model to the implementation model.

Since this implementation phase is highly error prone, functional verification is the most important and time consuming task in the overall design flow [15]. In our approach the effort for functional verification is drastically reduced by reusing the abstract system model as a reference for the synthesizable implementation models. For that we have generalized the well known adapter concept for communication refinement [13] to bridge the abstraction gap between packet-level and cycle-level models.

The next section describes the channel library, which has been developed to enable the exploration and co-verification methodology.

IV. Methodology Specific Channel Library

SystemC 2.0 provides a well defined interface for its underlying event-driven simulation kernel to enable implementation of methodology specific communication channels, like e.g. data-flow fifos or HW signals. We have developed such a SystemC 2.0 compliant channel library, which provides intuitive visualization, a comprehensive set of communication and timing annotation primitives as well as support for mixed-level co-simulation.

A. MSC Visualization

Our way of modeling complex systems is illustrated by a snapshot of our graphical debugger depicted in figure 2. The simulation is visualized according to the Message Sequence Chart (MSC) principle, which is a well known tracing mechanism usually employed in SDL design environments like the TAU SDL suite [16]. For our methodology, MSC tracing provides a very intuitive visualization of a network of coarse-grain SystemC processes exchanging ADTs.

SystemC modules are represented by vertical lines, which are labeled with the module name at the top. The progress of time is vertically notated from top to bottom. Communication events are displayed by horizontal arrows between the lines of the sending and receiving processes. These arrows are labeled with signal name, the point in time and the ADT type name.

A general drawback of the MSC representation is the amount of communication events for realistic systems with tens and hundreds of SystemC modules, because the user can observe only roughly up to 15 modules and up to 40 events simultaneously. Therefore our debugger provides advanced filter mechanisms to systematically reduce the displayed data to the currently interesting communication events.

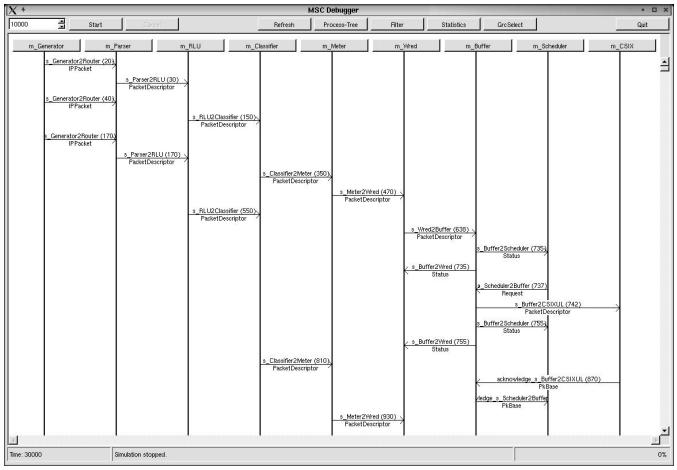

Fig. 2. Message Sequence Chart Visualization of the SystemC Simulation

To enable dynamic filtering, the MSC debugger maintains a data-base of all communication events. This data base can be searched in multiple dimensions, e.g. time, packet type, connection name or SystemC process hierarchy. Furthermore the event filter can introspect and search for the member-fields inside the ADTs, so the MSC debugger can filter e.g. all IP-packets with a certain destination address.

MSC visualization is a built-in feature of all channels in our library, so it is enabled by just instantiating the channels without any additional code in the user modules. Further channel features like timing annotation for architectural exploration are described in the next sections.

B. Timing Annotation

Our goal is to model the architecture specific timing independently from the behavior, thus a functionally correct system model can be easily mapped to different architectures. According to the observations listed in III-C, the channels provide methods to annotate processing delay and initiation interval. The channels are implemented hierarchically (see chapter 11 of [17]), since they incorporate internal processes to implement the mechanism for timing annotation.

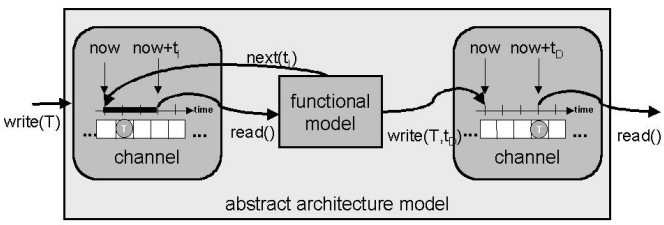

Fig. 3. Abstract Architecture Model

Figure 3 illustrates the creation of an abstract architecture model from the combination of the original functional model together with the channels to capture the impact on performance of the intended architecture. The right part of figure 3 shows the annotation of a processing delay t_D, which is passed as a second parameter of the `write()` method of outgoing connections. Transparently the channel takes care, that this token does not arrive at the consumer process before the specified delay t_D. In the same way, the left part of figure 3 shows the annotation of an initiation interval t_I by calling the `next()` method of incoming connections. This causes the channel to suppress the arrival of tokens for the specified amount of cycles, which captures the effect of a blocking module.

Besides dedicated point to point communication, modern SoC architectures very often employ on-chip busses to improve utilization of interconnect resources. However shared busses cause a non-deterministic communication delay, which can have negative impact on the overall system performance in case of insufficient bus bandwidth. To capture this effect on the high abstraction level, we provide a generic bus model, which connects an arbitrary number of master and slave modules with a parameterizable bandwidth. A priority based bus arbitration scheme is used to resolve conflicts in case more than one master wants to perform a bus transaction at the same time. Alternatively, a Time Division Multiple Access (TDMA) based arbitration scheme allows interleaved service of simultaneous bus requests.

C. Mixed-Level Adapter Channel

The mixed-level adapter channel bridges the gap between packet-level and RT-level modules to enable block-wise functional verification, i.e. the RTL implementation is plugged into the abstract architecture model. Thus stimuli are derived from the system context and comparison of the synthesizable bit and cycle true implementation is done against the abstract architecture model. By that the significant effort for writing and verifying RT-level testbenches is avoided.

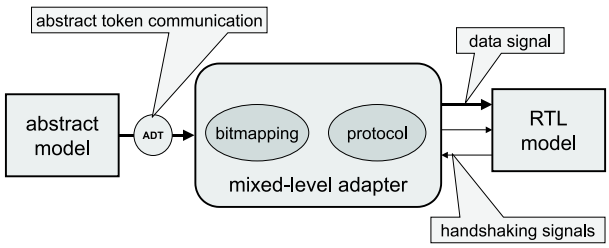

Fig. 4. Data- and Timing- refinement

The structure of the mixed-level adapter channels is depicted in figure 4: in the *bitmapping* engine, the adapter channel maps the ADT to the respective fields in the corresponding bit-accurate data representation. The resulting bitstream is then transfered to the protocol layer, where it is cut into slices according to the respective data width and forwarded to the RTL implementation. The *protocol* engine adds all the required control signals to reform the specified interface protocol.

A similar adapter channel implements the reverse direction to feed the output of the RTL implementation back into the packet-level environment. Here the ADT is reconstructed from the output bitstream provided by the protocol engine. Note that this concept is also capable to bridge packet-level and cycle-level TLM. Here the protocol engine calls the TLM interface methods instead of wiggling the RTL pins.

V. IP Forwarding Chip

In this section we present the results of an IP forwarding chip design project, where the proposed system level design methodology has been applied in cooperation with Synopsys Professional Services. We will introduce and characterize the design and illustrate the architecture mapping methodology by means of experimental results.

A. IP Forwarding with QoS Support

IP forwarding is a central part of the IP network infrastructure. This challenging application domain combines sophisticated functionality for QoS support with highest performance requirements: at OC-48 wire speed (corresponds to 2.5Gbit/s) the timing budget in the NPU for the processing of a minimum size 48Byte packet is as short as 147ns.

IP Forwarding with QoS Support according to the DiffServ proposal [18] requires a set of complex functional blocks. These are displayed in the shaded part of figure 5 together with the basic inter-block communication. The *Parser* performs checks on incoming packets and provides the following units with packet descriptors, which hold all the relevant header information of the respective IP packet. The *Route Lookup Unit* (RLU) performs forwarding of IP Packets based on the longest match table search algorithm. The forwarding decision is based upon destination IP address and the routing table. The *Classifier* classifies incoming packets into Classes of Service (CoS), so the packets are processed according to their negotiated Quality of Service parameters by the following blocks. The *Meter* measures the IP packet rate and drops packets exceeding the negotiated traffic characteristics to protect the succeeding queuer unit from unfriendly traffic streams. The *WRED* unit drops additional packets according to the weighted RED algorithm [19] to avoid throughput degradation especially for TCP traffic due to congestion. The *Queuer* stores IP packets according to their class of service until they can be forwarded to the CSIX unit. The *Scheduler* decides on the basis of the priority and the actual fill status of the packet queues in the queuer unit. Finally the *CSIX* unit segments IP packets into fixed size packets according to the standardized CSIX bus protocol [20] to interface the switch fabric.

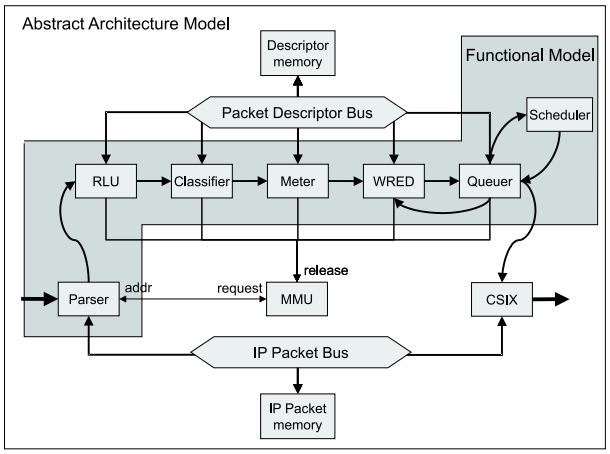

Fig. 5. IP Forwarding Chip

According to the methodology described in section III, first a functional SystemC model is created from the functional specification document. This functional model is well suited to validate completeness and functional correctness, but does not yet impose any assumptions on the architectural realization.

B. Architecture Refinement and Exploration

Referring to the complete system view of figure 5, we first have to add an IP Packet Memory and a Memory Management Unit (MMU) to the functional model, which are necessary to store IP packets during the processing is performed by the functional units.

Given the specified constraints on latency and cost, the architect now has to define the general organization of the architecture. In our case, we decided to replace the expensive and inflexible point to point exchange of packet descriptors with a shared memory architecture, which is accessed through an on-chip bus.

The resulting SystemC architecture model of the IP router is running at 140k cycles per second on a 2 GHz Linux PC, thus in our case the simulation of 1 second IP traffic takes less than 12 minutes. In the following steps, the top-level structure of the router does not need to be modified. Instead, different configurations of the general system architecture depicted in figure 5 are evaluated in a very effective way by adjusting the generic model parameters and running the simulation.

An initial timing budget in terms of pipeline delay and initiation interval is annotated to each module as explained in section III-C. The initial annotations represent an educated guess depending mostly on the experience of the designer. In our case, the timing annotations for the functional blocks are taken from a implementation on the Intel IXP2400 Network Processing Platform [21].

Based on simulations, the architect may relax some constraints, choose to tighten some others, or find out that a budget is required that is not realistic for a module. In the latter case, the architecture will be modified, e.g. resources are added or the algorithm is changed. Final budgets provide a requirement specification for later RTL implementation. The statistic evaluation of the functional blocks and the channels is used to detect bottlenecks in the system. A well-balanced system architecture provides sufficient processing power for all components.

An excerpt of the exploration results dealing with the dimensioning of the packet descriptor communication is printed in table II. Besides the interconnect-description and -configuration we list the resource utilization and the mean delay per transaction of on the considered bus. The right column contains the overall packet latency introduced by the complete IP processing chain. Since the timing annotations of the functional units are constant during the depicted experiment, the measured packet latency clearly unveils the impact of the selected communication architecture on the system level performance.

The point-to-point (p2p) communication of the functional model generates an initial traffic profile. The unconstrained p2p setup performs all communication with zero overhead, so the IP latency in line 1 reflects only the timing annotations of the functional units. The 32 bit wide p2p engine in line 2

line	interconnect description	interconnect configuration	utili-zation	trn. delay in cycles	IP latency in cycles
1	p2p	unconstr.	0%	0	910
2	p2p	32bit	8%	22	1117
3	IP_p2p	32bit	52%	75	1117
4	IP_bus	64bit_prio	78%	149	1191
5	IP_bus	64bit_tdma	78%	93	1125
6	pd_p2p	32bit	4%	20	1117
7	pd_bus	32bit	82%	51	1582
8	pd_bus	64bit	51%	36	1355
9	pd_bus	128bit	46%	23	1142

TABLE II
EXPLORATION RESULTS

serves as a 'lower-bound' reference configuration and not as a realistic design option. Among other unrealistic assumptions, this would implicate a packet descriptor memory with 6 read and 6 write ports , i.e. one for every connected module.

Besides an insignificant amount of local communication between the buffer and its neighboring units, the statistics generated by the p2p engine admits the classification of the on-chip traffic into two domains: (a) Parser, RLU and CSIX unit regularly access the IP memory (DRAM) to store and forward the arriving IP Packets and (b) all master blocks heavily access the packet descriptor memory.

Lines 3–5 refer to the exploration of traffic domain (a), where we examined a dedicated configuration corresponding to a multi-port IP memory and two different bus arbitration schemes connected to a single port memory. The priority based bus arbiter is obviously outperformed, because the uninterrupted transfer of long IP Packets has a negative influence on short packets. The preemptive TDMA scheduler minimizes the average delay by interleaving short and long packet transfers. The decision between configuration 3 and 5 is mainly driven by the technology dependent implementation cost of memories and wiring.

The third section in table II is dedicated to the exploration of a customized communication architecture for the access to the packet descriptor memory. The complete p2p mesh in line 6 is way too expensive and inefficient. Lines 7 to 9 shows the significant impact on performance of the shared bus bandwidth. Here the designer has to trade implementation cost against system performance.

VI. CONCLUSION

In this paper, a system level design methodology based on the SystemC 2.0 library is presented. The outlined approach is capable to capture the complete system functionality as well as all performance relevant architecture features on the highest possible level of abstraction. The resulting modeling efficiency measured in lines of code and the simulation speed is about two orders of magnitude better compared to an RTL architecture model.

During a research cooperation with Synopsys Professional Services, the outlined methodology has been applied to the architecture conceptualization of an IP forwarding chip. By employing the proposed system level design methodology, a team of 3 engineers was able to demonstrate the feasibility and define a scalable and cost effective architecture within 2 months. The resulting system architecture model also serves as an executable specification and as a fast co-verification environment for the HW implementation.

In the second project phase, the HW/SW co-design aspects of the methodology are currently applied to the IP Router case study. Here we consider the cycle-level TLM phase and investigate the mapping of several computational tasks to Application Specific Instruction-set Processor (ASIP) cores to improve the system flexibility. The ASIP design is performed using the LISA based Processor Design methodology [22], [23].

REFERENCES

[1] K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer-Aided Desig of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1523–1543, December 2000.

[2] J. Rowson and A. Sangiovanni-Vincentelli, "Interface-Based Design," in *Proceedings of the Design Automation Conference (DAC)*, 1997.

[3] SystemC initiative, *http://www.systemc.org*.

[4] Balarin et al., *Hardware-Software Co-Design of Embedded Systems : The Polis Approach* . Kluwer Academic Publishers, 1997.

[5] VCC: Virtual Component Co-Design, *Cadence, http://www.cadence.com*.

[6] Andy D. Pimentel, Louis O. Hertzberger, Paul Lieverse, Pieter van der Wolf, Ed F. Deprettere, "Exploring Embedded-Systems Architectures with Artemis," *IEEE Computer*, vol. 34, no. 11, pp. 57–63, November 2001.

[7] A. Hoffmann, T. Kogel, H. Meyr, "A Framework for Fast Hardware-Software Co-simulation," in *"Proc. Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2001.

[8] D. Gajski, J. Zhu, R. Dömer, A.Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[9] G. Vanmeerbeeck, P. Schaumont, S. Vernalde, M. Engels, I. Bolsens, "Hardware/Software Partitioning of embedded system in OCAPI-xl," in *"Proc. Int. Symp. on Hardware/Software Codesign (CODES)"*, 2001.

[10] V. Sinha, F. Doucet, C. Siska, R. Gupta, S. Liao, A. Ghosh, "YAML: a tool for hardware design visualization and capture," in *Proc. Int. Symp. on System Synthesis*, 2000.

[11] CoCentric System Studio, *Synopsys, http://www.synopsys.com*.

[12] CoCentric SystemC Compiler, *Synopsys, http://www.synopsys.com*.

[13] T. Grötker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[14] Describing Synthesizable RTL in SystemC, V1.0, *Synopsys Inc., USA*.

[15] A. Evans, A. Silburt, G. Vrckovnik, T. Brown, M. Dufresne, G. Hall, T. Ho, and Y. Liu, "Functional Verification of Large ASICs," in *Proceedings of the Design Automation Conference (DAC)*, 1998, pp. 650–655.

[16] TAU SDL suite, *Telelogic, http://www.telelogic.com*.

[17] *Functional Specification for SystemC 2.0*.

[18] An Architecture for Differentiated Services, *http://www.ietf.org/rfc/rfc2475.txt*.

[19] S. Floyd, and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 379–413, August 1993.

[20] The Network Processor Forum, *founded by CSIX/CPIX members in 2001 http://www.npforum.org*.

[21] S. Lakshmanamurthy, K.-Y. Liu, Y. Pun, L. Huston, U. Naik, "Network Processor Performance Analysis Methodology," *Intel Technology Journal*, Aug. 2002.

[22] A. Hoffmann, T.Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, "A novel methodology for the design of application specific instruction-set processors using a machine description language," *IEEE Transactions on Computer-Aided Desig of Integrated Circuits and Systems*, vol. 20, no. 11, pp. 1338–1354, 2001.

[23] LISATek Product Line, *CoWare, http://www.coware.com*.