A Framework for Fast Hardware-Software Co-simulation

Andreas Hoffmann, Tim Kogel, Heinrich Meyr
Integrated Signal Processing Systems (ISS), RWTH Aachen
Templergraben 55, 52056 Aachen, Germany
hoffmann[kogel,meyr]@iss.rwth-aachen.de
http://www.iss.rwth-aachen.de/lisa

Abstract

We present a new hardware-software co-simulation framework enabling fast prototyping in system-on-chip designs. On the software side, the machine description language LISA allows the generation of bit-true models of programmable architectures on various levels – from instruction-set to phase accuracy. Based on these models, a complete tool-suite consisting of fast compiled processor simulator, assembler, linker, HLL-compiler as well as co-simulation interface can be generated automatically. On the hardware side, the SystemC simulation class library is employed and enhanced with our generic co-simulation interface that enables the coupling of hardware and software models specified at various levels of abstraction. Besides that, a hardware modeling strategy using abstract macro-cycle based C++ processes to increase hardware modeling efficiency and simulation speed is presented.

1 Introduction

Today, typical single chip electronic system implementations include a mixture of microcontrollers, digital signal processors (DSPs) as well as shared memory, dedicated logic (ASICs) and interconnect components. Driven by the ever increasing hardware and software design complexity, components from various design teams and third parties (intellectual property blocks) are employed. Due to the heterogeneity of these components and the drastically increased number of gates per chip, verification of the complete system has become the critical bottleneck in the design process [14]. Hardware-software co-simulation integrates hardware and software design techniques which are typically using various languages, formalisms and tools into a single design methodology. Using a single framework for this task accelerates the design process, enabling hardware-software trade-offs to be made dynamically as the design progresses, and eases verification significantly. For verification and evaluation of hardware-software trade-offs of the complete system, large test-vector sets are needed. Considering that the amount of test-vectors needed for verification rises by a factor of 100 every six years [2], which is ten times the increase of the number of gates on a chip as stated by Moore's law, it becomes clear that simulation speed of the overall system is crucial when designing a complex system.

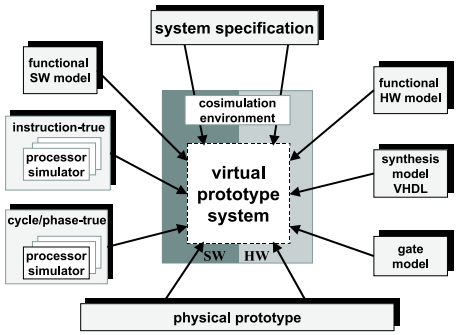

Figure 1. Virtual prototype system

To increase the productivity and shorten time to market it is important to be able to verify a heterogeneous system-on-chip (SOC) design at an early stage of the development process to prevent expensive re-designs. Here, heterogeneity is not only referring to hardware and software models but also to models specified on either side on different abstraction levels that have to be coupled. For example parts of the system's hardware taken as intellectual property (IP) from previous designs might be specified on a low abstraction level as register-transfer or gate-level VHDL/Verilog code, whereas new functionality added to the system is specified on a high abstraction level in the programming language C. On the software side the model accuracy can also vary from phase accuracy over instruction-set accuracy to functionally correct C-code specifying the behavior of the application running on the target architecture. So it is compulsory to have one simulation environment that understands the semantics of all models and settles the interfaces to enable communication among them. Coupling and verifying of the different parts of the target system at any time of the design process is what we call building a *virtual prototype* (VP) of

the system in software (see figure 1).

The approach presented in this paper of using the machine description language LISA [11] for the software side and a C++ framework based on the SystemC class library [9] for high simulation speed on the hardware side and for integrating different hardware-software models into one simulation environment fulfills the posed requirements. LISA allows the specification of programmable architectures on various abstraction levels and the automatic generation of fast processor simulators, assemblers, linkers, HLL-compilers as well as co-simulation interfaces. The enormous speedup achieved by employing the compiled simulation technique [17] over the commonly used interpretative simulation technique of more than two orders of magnitude even makes the usage of phase accurate processor models permissible. This degree of model accuracy is required when using architectures with complex pipelines keeping up simulation speed on the software side.

On the hardware side the proposed methodology is based on the SystemC simulation library that is extended by the results of our GRACE++ project [12]. This introduces abstraction from the hardware by using macro-cycle based functional C++ processes and accelerates simulation speed significantly. In addition to that the GRACE++ co-simulation interface allows integration of various hardware and software simulation environments into one co-simulation framework. Moreover, it is possible to couple processor, ASIC and FPGA prototypes via the RAVEN-board [8] to the simulation environment.

2 Related Work

Several researchers have proposed methodologies for hardware-software co-verification and fast prototyping of digital systems but primarily aiming at automated hardware-software partitioning and co-design. Simulation speed of the overall system is not in the primary focus thus leading to a significant bottleneck in the SOC design.

In [4] and [5] a system level design environment aiming at system-on-chip designs including real-time embedded software is proposed. Here, software can be functionally tested in combination with hardware and successively refined from the system level model to the software source-code implementation.

The COSMOS co-design environment [3], which is now commercially available from AREXSYS [1] takes an SDL system specification to perform automated design space exploration, partitioning into hardware and software parts and code generation. The approach of *CoWare* [16] uses C/C++as the base language for the system specification and allows besides synthesis and interface generation the mapping of the software onto various off-the-shelf target architectures to explore different hardware/software combinations. The *COSYMA* system [10] specifies the system in the C* language which is similar to *C* and aims primarily at the partitioning of the system functionality in hardware and software parts.

Moreover, co-simulation/verification tools are offered commercially by companies such as Mentor Graphics [7] and SYNOPSYS [15] which couple VHDL/Verilog simulators with software simulators via well defined interfaces. All approaches have in common that the instruction set simulators integrated into the system environments are commercial interpretive simulators and thus too slow for an efficient process of verification and performance measurement. On the hardware side, the C++ models are based on clock cycles which ruins modeling efficiency and simulation speed in early design stages.

3 LISA Language and Tools

The language LISA is aiming at the formalized description of programmable architectures, their peripherals and interfaces. It was developed for cycle/phase-accurate simulation purposes of a wide range of modern programmable architectures (DSPs and microcontrollers).

The development of LISA was motivated by the fact that the task of building a custom simulator for a new architecture is extremely tedious and error-prone. It is a very lengthy process of matching the simulator to an abstract model of the processor architecture. These efforts can be significantly reduced by using a retargetable simulator which is generated from machine descriptions. At the same time, simulation speed is critical and thus important in simulator design. The principle of compiled simulation [17] is to take advantage of a priori knowledge and move frequent operations from simulation run-time to compile-time with the goal of providing the highest possible simulation speed. In contrast to interpretive simulators, this approach requires a transformation step to be performed before simulation can be run.

3.1 Model requirements

Indeed, depending on the complexity of the employed architecture, either instruction-set or cycle/phase accurate models are needed to enable co-simulation with the surrounding hardware. For relatively simple architectures having either very rudimentary or no pipelines at all, an instruction set model of the architecture is sufficient. This model is then coupled via a bus interface model (BIM) to the hardware environment. The BIM thereby interprets external events and generates cycle/phase accurate simulation traces at the component's pins. For more complex architectures, though, employing heavy pipelining and interlocking mechanisms as recently seen in both the DSP and microcontroller area, this methodology is no more applicable [6]. Here, cycle and phase accurate simulators for co-simulation purposes are needed to cope with the problem

of instructions being split into smaller pieces and the associated interactions between the processor and other components across multiple cycles. Moreover, phase accurate models for any architecture on the software side enable to keep co-simulation interfaces simple, since reads and writes are made directly on processor resources (buses and pins). Thus BIMs become redundant (see figure 2).

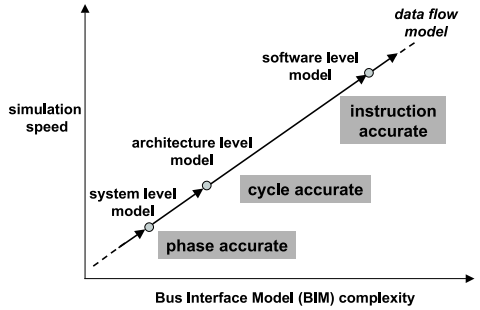

Figure 2. BIM complexity vs. speed

The simulation slowdown due to the increased model accuracy can be completely compensated by employing fast compiled processor simulators generated from a LISA description of the target architecture. Typical speedups of compiled processor simulators versus interpretive simulators range in the area of two orders of magnitude.

4 The SystemC platform

The SystemC class library enables the building of synthesizable C++ hardware models on the behavioral- and register-transfer-level. This section proposes a methodology to employ SystemC from the beginning of the specification phase in order to perform high level system modeling and successive refinement to synthesizable models within one single framework. The objectives we pursue with this methodology are twofold: at first to improve modeling efficiency by exploiting the object oriented features of C++ for abstract hardware modeling and successive refinement and at second to increase simulation speed to cope with the rising number of test-vectors needed for system verification and performance evaluation. For coupling models specified at various abstraction levels of both the hardware and the software side, we enhanced the SystemC simulation library with a generic simulation interface which enables the integration of external simulators.

4.1 Hardware Simulation with SystemC

The SystemC class library provides a synthesizable C++ subset in order to establish a unitary implementation language for both hardware and software parts. Therefore the SystemC library is furnished with a set of classes to express the behavior of hardware blocks by means of C++ processes and communication happens by exchanging data via signal routes. Of course any synthesizable SystemC description has to cope with the demands of the subsequent architecture and logic synthesis tools, hence the propagated modeling style corresponds to the behavioral- or register-transfer-level of pure hardware description languages (HDLs) like VHDL or Verilog . However by just applying HDL semantics to the C++ syntax, the methodical gap between algorithmic system specification and hardware implementation is not resolved, neither is simulation speed perceptibly improved.

Our methodology targets to fill the gap between hardware specification and implementation. Thereby hardware is first modeled at a higher level of abstraction and within the SystemC framework successively refined to the synthesizable subset. In our approach abstraction applies to structure, data and time. The system specification is first structured into course grain functional blocks that exchange abstract data types. This specification is subsequently divided into subcomponents and the abstract data types are refined towards their bit-level representation manually.

Key concept for raising the abstraction level is the introduction of a hierarchical time scale. Of course system performance validation needs a time base for latency annotation and throughput measurement, but the high resolution to hardware clock cycles is a significant drawback in modeling efficiency and simulation speed. Indeed, many applications in the area of high speed networking and wireless communication are packet based with a fixed length of data packets (e.g. ATM cells, SDH frames, GSM packets, UMTS slots). On a high level of abstraction only the state changes at packet arrival times need to be modeled and the identification of a macro-cycle is straightforward. The introduction of a logical macro-cycle enables performance profiling by using large test-vector sets and hides superfluous timing details.

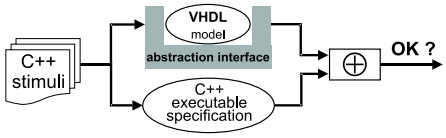

Figure 3. Co-verification of C++ vs. implementation model

The choice of an appropriate macro-cycle depends on the considered application. In typical SOC designs several programmable cores and peripheral hardware components communicate via one or more asynchronous buses at different data rates. Here the designer can trade-off between modeling accuracy on the one hand and simulation speed and modeling efficiency on the other hand to determine the ideal time division into macro-cycles.

Within our design methodology system specification starts with an untimed description, where all operations are performed within one macro-cycle. Accuracy of sys-

tem performance measurement is then subsequently refined by back-annotated latency information extracted from later synthesis results. The synthesizable implementation model can be verified at any time against the executable macro-cycle based model as depicted in figure 3 using C++ stimuli derived from the system context and the abstraction interface described in the following subsection.

4.2 Virtual Prototyping of the System

For integration of various hardware-software models jointly with the macro-cycle based C++ processes an adaptable co-simulation interface is employed. Coupling LISA software simulators, independent from their underlying model accuracy (i.e. phase, cycle or instruction accuracy), to VHDL/Verilog simulators is straightforward, since they both employ a similar interface that reads and writes hardware resources. Coupling models with different underlying time scales though needs some effort. An abstraction interface fulfills the task to adjust different time and data abstraction levels of SystemC, VHDL/Verilog and software simulation environments.

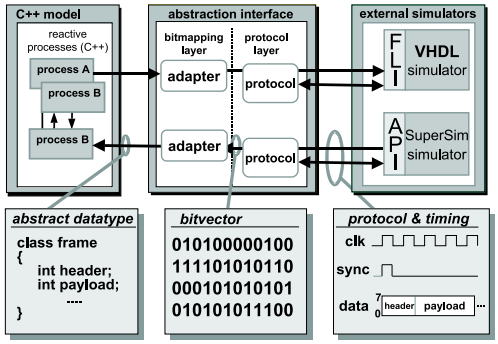

Figure 4. Abstraction interface

To achieve a flexible and generic Co-simulation interface we incorporated a two-step approach as depicted in figure 4. The abstract data types of the C++environment are first mapped to a binary representation by the bitmapping layer. The resulting bit-streams are transferred to the protocol layer, cut into slices according to the respective data bus width and forwarded into the external simulator. The protocol layer adds all the required control signals (e.g. data-valid, sync, enable) to perform the specified bus protocol. Changes in the interface specification can be easily applied to the C++protocol classes.

This approach provides great flexibility since bit-level communication can be easily established on the one hand to refined synthesizable SystemC models and on the other hand to any external simulator providing a C language interface: e.g. towards VHDL/Verilog hardware simulators via the foreign language interface (FLI) or towards the compiled simulators generated from LISA processor models via the co-simulation interface (API).

5 A case study

In a case study we successfully applied the proposed methodology to the port-processor of an ATM switch design. The port-processor identifies incoming ATM cells, performs local cell scheduling and cell flow-control by running the control dynamic transfer protocol (CDT)[13]. The functionality of the port-processor can be divided into a high speed cell processing part realized in dedicated hardware and a low speed part for signaling, configuration and maintenance implemented in software on an ARM 7 microcontroller.

Firstly, we realized a phase accurate model of the ARM 7[1] with LISA. Due to the high modeling efficiency of LISA the description and verification of the ARM 7 took less then four weeks. It comprises approx. 1500 lines of code including comments and empty lines. Based on that the complete LISA tool-suite was generated automatically. The LISA simulator of the ARM 7 runs at a speed of 4.5 mega cycles per seconds (see table 1).

Table 1. ATM port-processor simulation

Model	Speed [kCycles/sec]
VHDL (*VSS* compiled) *phase accurate*	0,27
SystemC *marco-cycle based*	98,5
LISA simulator (ARM 7) *phase accurate*	4500
Virtual prototype (system) SystemC + LISA	52,5

On the hardware side, the functionality was partitioned into several communicating blocks that were at first all modeled as abstract SystemC processes. The underlying macro-cycle was straightforward chosen to ATM cell boundaries. The abstract C++ models were then successively refined to VHDL models, since synthesis tools for SystemC were not available at that time. These models were integrated into our simulation environment and co-simulated with the rest of the system to ensure their correctness (see figure 5). By this approach, 37 implementation errors on the hardware side and 12 errors in the embedded software code were detected within 3 weeks until the virtual prototype passed all tests. The external stimuli used for verification (arriving ATM cells and configuration) were specified in C++ eliminating the tedious and error-prone task of writing VHDL/Verilog test-benches.

By integrating the LISA simulator of the ARM 7 into the SystemC simulation, functional verification and performance assessment of the complete port-processor was pos-

[1]The authors would like to thank Tim Hopes, Ian Phillips and Mark Burton of ARM Ltd. for their organizational and technical support in the project.

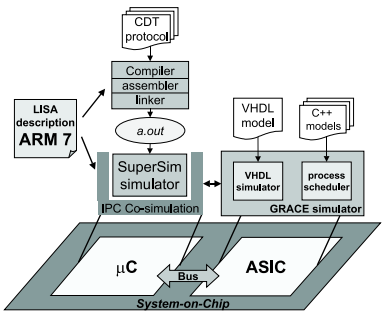

Figure 5. Verification of a port-processor

sible at an early stage of the design process. Due to the high simulation speed of the heterogeneous system, large test-vector sets could be processed resulting in a high state coverage in verification. Simulating the complete hardware as abstract C++ processes lead to a speedup of 365 compared to simulation of event-driven compiled VHDL models with *VSS* (the hardware parts comprised approx. 50k logic gates plus 0.5 MBit memory). All results were obtained on a 300 MHz Sun Ultra 10 with 2 GBytes of RAM.

The proposed methodology of using LISA processor simulators for the software side and macro-cycle based reactive C++ models for the hardware side leads to simulation speedups of more than two orders of magnitude compared to the commonly used verification techniques, interpretive software simulators and event-driven hardware simulators.

6 Conclusion and Future Work

In this paper we presented a new methodology for early hardware-software co-verification by fast prototyping coping with the enormous design complexity. The machine description language LISA allows the bit-true specification of programmable architectures on various abstraction levels and the automatic generation of fast processor simulators. A C++ simulation framework based on the SystemC class library integrates various hardware/software models and enhances simulation speed by employing functional macro-cycle based C++ processes on the hardware side.

In an ATM switch design we successfully employed the introduced methodology. Thereby we modeled the hardware side of the port-processor with SystemC processes by abstracting the time to the granularity of ATM cells. On the software side, the ARM 7 microcontroller was employed. The programmable architecture was described with LISA and a LISA simulator was generated. Due to early verification of the complete system the total system development time went down by a factor of four compared to a similar design realized before using traditional verification techniques.

Our future work will focus on applying the proposed technique to further SOC designs as well as enabling hardware synthesis from LISA architecture descriptions, which is mainly targeting at the generation of the control-path. Besides, we will investigate the automatic derivation of stimuli for both hardware and software parts from an SDL description to improve the coverage of our verification methodology.

References

[1] Arexsys. http://www.arexsys.org.

[2] R. Camposano. Automating System Implementation from System Specification, Oct. 1997. Talk at Synopsys University Day "Towards System on Silicon", Aachen.

[3] J. Daveau, G. Marchioro, T. Ben-Ismail, and A. Jerraya. *Hardware/Software Co-Design and Co-Verification*, volume 8, chapter COSMOS: An SDL Based Hardware/Software Codesign Environment, pages 59–87. Kluwer Academic Publishers, 1997.

[4] D. Desmet and *et al.* Timed executable system specification of an adsl modem using a c++ based design environment: A case study. In *Proc. of the Int. Workshop on Hardware/Software Codesign*, May 1999.

[5] D. Desmet, D. Verkest, and H. De Man. Operating System based Software Generation for Systems-on-Chip. In *Proc. of the Design Automation Conference (DAC)*, Jun. 2000.

[6] L. Guerra, et *al.* Cycle and phase accurate DSP modeling and integration for HW/SW co-verification. In *Proc. of the Design Automation Conference (DAC)*, Jun. 1999.

[7] Mentor Graphics. *Seamless http://www.mentor.com/seamless.*

[8] A. Müller, G. Post, and M. Vaupel. RAVEN - A Real-Time Analysis and Verification Environment. In *Proc. Int. Conf. on Signal Processing Application and Technology (ICSPAT)*, pages 297–301, Toronto, Sep. 1998.

[9] Open SystemC Initiative. *http://www.systemc.org*.

[10] A. Österling, T. Brenner, R. Ernst, D. Herrmann, T. Scholz, and W. Ye. The COSYMA system. In *Hardware/Software Co-Design: Principles and Practice*. Kluwer Academic Publishers, 1997.

[11] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr. LISA – Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures. In *Proceedings of the Design Automation Conference (DAC)*, New Orleans, June 1999.

[12] G. Post, A. Müller, and T. Grötker. A System-level Co-Verification Environment for ATM Hardware Design. In *Proceedings of the European Conference on Design, Automation and Test (DATE)*, pages 424–428, Paris, Feb. 1998.

[13] Quantum Flow Control Specification, Rev.2.0, July 1995. WWW: http://www.qfc.org/.

[14] A. Silburt et.al. Accelerating Concurrent Hardware Design with Behavioural Modelling and System Simulation. In *Proc. of the Design Automation Conference (DAC)*, Jun. 1995.

[15] Synopsys. *Eagle, http://www.synopsys.com.*

[16] D. Verkest, K. Van Rompaey, and I. Boolsens. Co-Ware – A Design Environment for Heterogeneous Hardware/Software Systems. 1, Nov. 1996.

[17] V. Živojnović, S. Tjiang, and H. Meyr. Compiled simulation of programmable DSP architectures. In *Proc. of IEEE Workshop on VLSI in Signal Processing, Osaka, Japan*, pages 187–196, Oct. 1995.