UNISIM: An Open Simulation Environment and Library for Complex Architecture Design and Collaborative Development

David August[†], Jonathan Chang[†], Sylvain Girbal[*], Daniel Gracia-Perez[‡], Gilles Mouchard[‡], David Penry[§], Olivier Temam[*], Neil Vachharajani[†]

[*] INRIA, Orsay, France
[†] Dept. of Computer Science, Princeton University, Princeton, New Jersey
[‡] CEA, Gif-Sur-Yvette, France
[§] Dept. of Electrical and Computer Engineering, Brigham Young University, Provo, Utah

***Abstract*—Simulator development is already a huge burden for many academic and industry research groups; future complex or heterogeneous multi-cores, as well as the multiplicity of performance metrics and required functionality, will make matters worse. We present a new simulation environment, called UNISIM, which is designed to rationalize simulator development by making it possible and efficient to distribute the overall effort over multiple research groups, even without direct cooperation. UNISIM achieves this goal with a combination of modular software development, distributed communication protocols, multi-level abstract modeling, interoperability capabilities, a set of simulator services APIs, and an open library/repository for providing a consistent set of simulator modules.**

I. Introduction

The UNISIM project philosophy stems from the following six key observations on the current state of micro-architectural simulation:

1) Many existing simulators are monolithic or designed for a single architecture. As a result, it is difficult to extract a micro-architecture component from the simulator for reuse, sharing or comparison.
2) The trend towards Chip Multi-Processors (CMPs) implies that the forcus of simulation will shift from in-core behavior to system behavior; this shift will occur for two reasons: (1) a large number of cores will slow cycle-level simulators to unusable levels, and (2), communication between cores may have a greater impact on overall chip performance than in-core details.
3) For CMPs, the behavior of the *software system* (operating system) is more important, requiring full-system simulation.
4) Another major trend is architectural customization and heterogeneity. Heterogeneous architectures, much like embedded System-On-Chips, bring together a large set of IP blocks. These blocks vastly increase the potential design space. As a result, rapid coarse-grain prototyping phases of these systems will become increasingly necessary.
5) Simulators now require many more features than just performance evaluation: power and temperature modeling, sampling for simulation speed, debugging support for parallel computing, etc. These features are currently implemented in an ad-hoc and non-reusable manner, even though these issues are often orthogonal to the micro-architectures being modeled.
6) The current lack of interoperability among the different academic simulators hampers reuse, sharing, and comparison of ideas and hinders the take-up of academic results by companies which cannot afford the effort to integrate multiple academic simulators.

These observations form the rationale of UNISIM; the key contribution of UNISIM is that it addresses all of these issues in a unified fashion while still supporting distributed development. The following five features of UNISIM permit this goal to be met:

1) In order to address issue 1, UNISIM is a *modular* simulation environment, implemented as a layer on top of the industry standard SystemC [1]. Besides modularity, a key contribution of the UNISIM environment is a particular focus on the reuse of control logic, which corresponds to a large share of simulator code, and which is often overlooked by simulation environments.
2) In order to address issues 2 and 4, UNISIM supports an abstract level of modeling, called Transaction-Level Modeling (TLM), in addition to the more common detailed Cycle-Level Modeling (CLM). TLM simulators are less accurate but much faster than CLM simulators. UNISIM allows hybrid CLM/TLM simulators which can zoom in on only the important architecture details.
3) In order to address issue 3, UNISIM contains full-system functional simulators capable of booting a complex operating system like Linux. These functional simulators can be plugged into CLM or TLM simulators which are compliant to a functional simulator API.
4) In order to address issue 5, UNISIM provides APIs for a set of services. Any module implementing this set of standardized calls automatically benefits from the corresponding services. Moreover, since these services are provided at the simulation engine level, i.e., independently of any simulator, they can be easily modified or replaced.
5) In order to address issue 6, UNISIM provides two

Supported by the HiPEAC Network of Excellence under contract No. IST-004408.

Supported by National Science Foundation grant NGS-0305617. All opinions, findings, conclusions, and recommandations expressed are those of the authors and do not necessarily reflect the views of any of our supporters.

Manuscript submitted: 03-Jul-2007. Manuscript accepted: 30-Jul-2007. Final manuscript received: 2007-Aug-07.

features: a library of compatible modules and models which comes with the environment and is open to external contributions, and the ability to inter-operate with other simulators by wrapping them into UNISIM modules.

The following sections describe these UNISIM features in more detail and outline the current status of the platform and initial experiments.

II. Modularity and Control

UNISIM is a modular simulation platform in which all architecture blocks are mapped to corresponding software modules. They can only communicate through explicit software links corresponding to hardware wires. Beyond proper software practice, this approach has two benefits: it provides an intuitive mapping between the architecture block diagram and the simulator, and it reduces implementation inaccuracies, such as early signal access due to an incorrect variable read. Other environments, such as SystemC, the Liberty Simulation Environment (LSE) [15], and Asim [5] already propose modular simulation decomposition.

While mapping the architecture block diagram to a modular simulator is both attractive and intuitive, it can conflict with the objective of reusing simulator components. The key difficulty is the reuse of *control logic*. Control is often implemented, or simply represented, as centralized or only partially decentralized; a symptomatic example is the DLX block diagram from the famous textbook [7], where control is centralized into one block. While this implementation is modular, it is difficult to reuse: any modification in any of the hardware blocks would require a modification of the control block. And while control logic may correspond to a small share of transistors, it often corresponds to the largest share of simulator code. For instance, a cache bank can account for many more transistors than the cache controller, but it is just an array declaration in the simulator, while the cache controller can correspond to several hundred simulator lines.

In order to address these conflicting decomposition/reuse objectives, UNISIM uses a solution pioneered by LSE [15] and the MicroLib environment [13]. This solution provides a customizable representation of control which lends itself well to reuse and modularity. This control *abstraction* takes the form of a handshaking mechanism between modules: a module makes no assumption about the other modules beyond its incoming and outgoing signals. All control logic corresponding to interactions with other modules are embedded in these signals. This approach has two key benefits: (1) by construction, it provides a *distributed* simulator implementation of control, and (2) it defines a rigorous and standardized interface between modules, which, in turn, improves modules' interoperability. In contrast, SystemC and Asim define signals for communicating between modules, but no communication protocol for control, so that most SystemC and Asim modules are not compatible among themselves.

III. Simulation Speed and Transaction-Level Modeling (TLM)

In spite of significant gains in simulator development productivity, a caveat of modular simulators is simulation speed. In a monolithic simulator such as SimpleScalar, a hardware block sending data to another hardware block amounts to writing a variable. In a modular simulator, additional overhead is required to inform the receiving block that the data is available and schedule the receiving block's execution. As a result, a modular simulator based on UNISIM or pure SystemC can be more than 10 times slower than a monolithic simulator like SimpleScalar.

We can address this issue in two ways. The first solution is to use sampling. State-of-the-art sampling and checkpointing techniques such as SimPoint [14] and TurboSMARTS [16] require the simulation of a tiny fraction (usually much less than 1%) of the total program in order to achieve an accuracy on the order of 3% [14]. Unlike previous simulators, UNISIM provides sampling and check-pointing techniques at the engine level, independently of simulator modules, allowing any simulator compliant with a check-pointing API to benefit from the check-pointing service implemented within the engine.

However, while sampling techniques for user-level applications in single-core architectures are mature and efficient, sampling techniques for multi-cores or for full-system simulation are still in their infancy and may not prove to be accurate. As a result, UNISIM currently focuses on a second, readily available solution: abstract-level modeling, also called Transaction-Level Modeling (TLM) [6]; this solution retains the modularity properties of CLM simulators through a standard interface, but it relies on messages rather than a handshaking protocol. TLM simulators are less accurate but much faster than CLM simulators; for instance, on a PC with a 2 GHz Centrino, a CLM simulator of the PowerPC 750 processor only (no system hardware except for an SDRAM and a simplified bus) runs at 250 KIPS (Kilo-Instruction Per Second), while our TLM simulator of a full hardware system based on the PowerPC 750 (processor with caches, memory, PCI bus and chipset, SDRAM, and system devices) runs at 6.5 MIPS, i.e., about 26 times faster. The rationale for TLM is that, as the number of cores increases, detailed in-core behavior may no longer have a critical impact on overall architecture performance; it is therefore sufficient to focus on *transactions* between cores. A typical TLM simulator for performance evaluation is a functional simulator with timing annotations indicating the time between transactions (memory requests or communications with other cores/IP blocks). While SystemC provides some support for TLM simulators, neither LSE nor Asim do so.

UNISIM augments TLM in two ways. First, UNISIM adds timing annotations to requests and transparently updates these timing annotations based on the overall system behavior, e.g., delaying the effective start date of a load request due to earlier contentions in the network. Second, UNISIM enables the creation of modular hybrid TLM/CLM simulators. Hybrid simulators allow the architect to focus on only on the most critical portions of the design, which can significantly speed up both development time and simulation time. Moreover, hybrid simulators allow for incremental design of the architecture, where details are added as they become available in the design process. This latter asset will become increasingly important as architectures shift towards customization and heterogeneous designs. With hybrid simulators it is possible to sketch architecture designs before hardware, or even software, is fully available.

IV. Full-System Simulation

With the advent of CMPs, it is increasingly hard, if not impossible, to ignore the impact of operating systems activity. Whenever the number of threads can change during the program execution, the operating system scheduler will come into play, deciding which threads are executed where and for how long. This is the minimum support for operating system activity. However, if the simulation is bound to increasingly focus on transactions, the overall operating system activity should be considered, not just the scheduler.

Thus a simulation environment must support full system simulation. Several alternatives exist and are already used, but have undesirable limitations. Simics [9] is the most popular platform. It is capable of supporting multiple different operating systems and ISAs, and it is very fast, up to 1 GIPS on a recent PC, thanks to binary translation. However it is a commercial and closed platform. For instance, CMP support is publicly available only for Sun and PowerPC ISAs; as a result, in order to simulate CMPs based on other ISAs, it is necessary to run multiple instances of Simics and synchronize them. This workaround has been implemented in GEMS [10], but it remains cumbersome. Moreover, there is no way to alter the internal workings of Simics, for instance to change the simulated ISA. Other fast full-system simulators are Bochs and QEMU, but they are geared towards virtualization rather than hardware exploration. Another alternative is M5 [2], a full-system simulator under BSD license with good modularity properties, but it is a *simulator* targeted to a single architecture rather than an *environment* for developing an array of simulators. As a result, full-system support is intertwined with the simulator, and it is not possible to reuse it for other micro-architectures or ISAs.

Consequently, UNISIM proposes a reusable and free alternative. Full-System capabilities are implemented at the functional simulator level in the form of TLM modules (chipsets, interrupt controlers, peripherals and their buses among others) through a UNISIM functional simulator API. Any UNISIM micro-architecture simulator can use this API to access full-system functional simulation. The UNISIM PowerPC functional simulator is capable of booting Linux in 2 minutes on a PC with a 2 GHz Centrino. This PowerPC full-system functional simulator has also been used to build a full hardware system emulating a PowerMac G3 (PowerPC 300 MHz) and a PowerMac G4 (with PCI and a PowerPC G4 at 350MHz), down to the chipset behavior.

V. Simulator Services

Beyond measuring the number of cycles, simulators tend to include functionality which varies from a tool to another, depending on the expertise of their authors. Some simulators include power models, such as WATTCH or CACTI, others include temperature models, or mechanisms for fast simulation such as SMARTS and TurboSMARTS. Other functionality are becoming essential for CMPs such as debugging support, which is complex in a parallel environment, or check-pointing due to long program execution times, but they are only rarely available in current simulators.

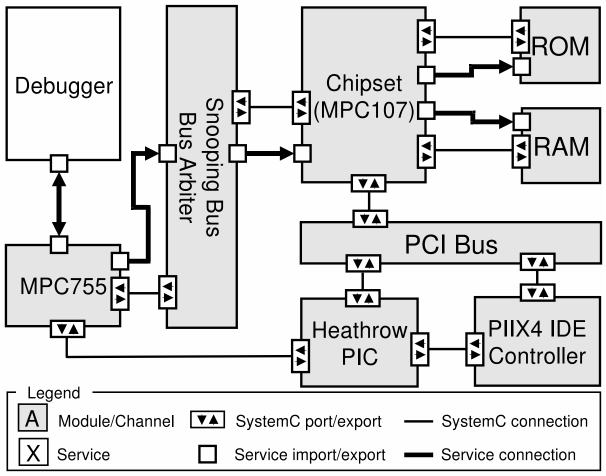

Fig. 1. *Full-System PowerMac G3 simulator with debugging services.*

In order to both increase the functionality of simulators, and to provide a more efficient way to leverage the work of different research groups, UNISIM implements a set of *Service APIs*. These APIs are essentially standardized function calls; any simulator module implementing these function calls automatically benefits from the corresponding services. For instance, a cache module providing statistics on its activity can get an evaluation of power consumption, provided a power model is plugged in the engine. For instance, Figure 1 shows the modules of our full-system Mac simulator, each with ports for accessing a debugging API; the debugging tool plugged into the UNISIM engine is the standard `ddd` debugger. This is the second benefit of the *Service APIs* approach: the services are plugged in at the engine level. Therefore, not only can any simulator benefit from a service as long as it implements the corresponding API, but it is also easy to replace a service tool by another, provided again that it is API compliant. For instance, two power models can be easily compared on any simulator with that approach. For the aforementioned `ddd` example, a small adapter (44 lines) must be developed to make a UNISIM module, such as the TLM PowerPC 750, `gdb` compliant.

VI. Simulator Interoperability

Not only is simulator functionality difficult to reuse and inter-operate, but the simulator implementations of different hardware blocks themselves are often excruciatingly difficult to extract from a simulator and reuse in another one. While SystemC is based partly on the vision of a standardized environment which allows reuse and sharing of simulator modules, in practice, reuse does not occur often. Most of the SystemC simulators are not interoperable because of insufficiently clear communication and development guidelines. Beyond enforcing stricter development rules as mentioned in Section II, UNISIM takes two additional steps for achieving interoperability.

The first step consists in acknowledging that simulator development is a huge effort, and any group which has invested several man-years on a tool will not drop that effort easily.

As a result, UNISIM is designed to create heterogeneous simulators by *wrapping* existing simulators within a UNISIM module, allowing existing simulators to interact with UNISIM modules, or even with other simulators. Besides the syntactic wrapping, an adapter must sometimes be developed to translate the simulator *Model of Computation* [4] into the UNISIM one, i.e., the method through which the different modules or parts of the simulator are called or woken up. In order to illustrate that this approach is pragmatic, we have wrapped the full SimpleScalar simulator into a UNISIM module, stripped it off its memory model, which is too simplistic, and connected it to another UNISIM module which contains a detailed SDRAM module developed into a third simulation environment. Only 50 source lines of SimpleScalar had to be modified in order to break the pipeline loop, and the resulting simulator is only 2.5 times slower despite the more complex memory model and the wrapping.

The second interoperability action is to build an open library or repository providing a set of consistent and interoperable models and modules. The initial models are meant to replicate existing industrial cores, and to build upon them, e.g., by implementing multi-core models. The repository maintains information about inter-module compatibility and module history and allows users to easily locate modules meeting their needs, thus improving the reuse of modules. Finally, this library is open, allowing anyone to upload modules, while retaining intellectual property rights to the modules and applying a license of the author's choice. This open library development effort distinguishes UNISIM from LSE, which did not develop a large library of models, and Asim, which is neither open nor publicly available.

VII. Current Status and Future Work

Currently, the library/repository at http://www.unisim.org contains a full-system functional simulator for a PowerPC 750, user-level functional simulators for ARM-v3 to ARM-v5te, and ST231, cycle-level models for a PowerPC405 single-core and an ARM-v5te-like single-core, a cycle-level multi-core model for a Shared Memory PowerPC 405 CMP with snooping-based coherence protocol. In addition, the repository contains two service APIs: a debugging API and a power API.

The future work is composed of three parts: the environment, the library, and research activities based on UNISIM. For the environment, we are finalizing the development of hybrid CLM/TLM simulators, and the next steps will consist in implement fast simulation techniques at the engine level, such as sampling, statistical simulation, and native execution. In addition, the modular nature of UNISIM enables us to automatically parallelize the simulation as well as use hardware accelerators. Future work will both integrate the automatic parallelization techniques from [12] and provide automated partitioning and synthesis tools for reconfigurable hardware [11].

Several groups are developing models which will be included in the library: the full-system PowerMac G3/G4 was jointly developed by CEA and BSC, a cycle-level model of the IBM Cell was developed at UPC/BSC [3], a model of an ST231 VLIW processor, and later on a distributed-memory multi-core, is being developed at INRIA, an ARM9 cycle-level and full-system simulator is being developed at CEA.

Finally, several research efforts are being undertaken or contemplated around UNISIM. For instance, we are developing a library browser for automatically scanning the design-space provided by the set of modules and models, in order to keep a permanent ranking of the best possible architectures, according to various criteria. This ranking would be updated as new modules are uploaded. A parameter API is being investigated to serve for this DSE browser. We also plan to investigate how to automatically derive timing annotations for TLM simulators using ANN-based [8] modeling.

References

[1] "SystemC, OSC Initiative," OSCI, Tech. Rep., 2003.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26b, no. 4, pp. 52–60, 2006.

[3] F. Cabarcas, A. Rico, D. Rodenas, X. Martorell, A. Ramirez, and E. Ayguade, "Implementation and validation of a cell simulator using unisim," in *In 3rd HiPEAC Industry Workshop, Haifa, Israel*, Palo Alto, California, April 2007.

[4] J. Davis, M. Goel, C. Hylands, B. Kienhuis, E. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuerdorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong, "Overview of the Ptolemy project," EECS, University of California at Berkeley, Tech. Rep. UCB/ERL No. M99/37, 1999.

[5] J. Emer, P. Ahuja, E. Borch, A. Klauser, C.-K. Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan, "Asim: A performance model framework," *IEEE Computer*, vol. 0018-9162, pp. 68–76, February 2002.

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

[7] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[8] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *PPoPP '07: Proceedings of the 12th ACM SIGPLAN Symp. on Principles and practice of parallel programming*. New York, NY, USA: ACM Press, 2007, pp. 249–258.

[9] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[10] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[11] D. Penry, Z. Ruan, and K. Rehme, "An infrastructure for hw/sw partitioning and synthesis of architectural simulators," in *WARP 2007: 2nd Workshop on Architectural Research Prototyping*, June 2007.

[12] D. A. Penry, D. Fay, D. Hodgdon, R. Wells, G. Schelle, D. I. August, and D. Connors, "Exploiting parallelism and structure to accelerate the simulation of chip multi-processors," in *Proc. of the Twelfth Int. Symp. on High-Performance Computer Architecture*, February 2006, pp. 29–40.

[13] D. G. Pérez, G. Mouchard, and O. Temam, "Microlib: A case for the quantitative comparison of micro-architecture mechanisms," in *Int. Symp. on Microarchitecture*. ACM, Dec 2004.

[14] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS X*, October 2002, pp. 45–57.

[15] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August, "Microarchitectural Exploration with Liberty," in *Proc. of the 34th Annual Int. Symp. on Microarchitecture*, Austin, Texas, USA., December 2001.

[16] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "TurboSMARTS: Accurate Microarchitecture Simulation Sampling in Minutes," *SIGMETRICS '05*, June 2005.