# A Heterogeneous Hardware-Software Co-Simulation Environment Using User Mode Linux and Clock Suppression

Hannes Muhr, Roland Höller, Member, *IEEE*, and Martin Horauer, Member, *IEEE*

***Abstract*— With the ever increasing complexity of electronic systems and the fact that networked embedded systems become more and more widespread, substantial research has already been devoted to such issues as hardware-software co-design and co-verification, virtual prototypes, and heterogeneous simulation environments including instruction set simulators.**

**The work presented herein, however, presents a new and efficient heterogeneous hardware-software co-simulation environment for network or communication centric distributed embedded systems. It is new, since for the first time it integrates a complete operating system kernel and thus enables testing of driver software and protocol stacks. It is efficient, since it allows intelligently turning off unnecessary simulation loads to allow for the long simulation run times typically needed for communication protocol simulation.**

**The correctness and validity of this virtual prototyping approach are shown by comparing simulation results with real-life measurements in a prototype system developed with the presented simulation environment.**

## I. Introduction

With shrinking feature sizes of semiconductor process technologies today several microprocessors, digital signal processors, memories, and application specific circuitry are often integrated on a single chip forming a system on a chip (SoC). Those SoCs typically make part of embedded systems of which a large number is distributed in space and is interconnected via a local area network, e.g. in factory automation systems, building automation, or sensor networks.

Not only the design of such complex systems is a challenge, but especially ensuring functional verification is of utmost importance. And it is time consuming, with the cost of undetected design errors tremendously increasing at the same time. This is why methodologies like hardware-software co-verification or virtual prototypes are in frequent use to speed up and enhance the overall quality of the verification process [14], [4], [2].

As always, there is a tradeoff between simulation speed and the detail in which the models represent the actual physical entities of the system. And often - especially for real-time systems - it is crucial to have reliable timing information available. This is the case e.g. when using an instruction set simulator for a certain processor to execute the software to be verified against not only functional but also timing constraints.

Certain systems, however, will not require the same level of detail for reaching the verification goal, and the cost and loss of computing performance of an instruction set simulator (ISS) can be avoided, even with the performance of state of the art ISS of about one million cycles per second [1], [10]. Instead the software can be executed by a special software module, User Mode Linux in this case, if information about the execution time is not of importance. In parallel a gain in simulation performance is possible, if portions of the hardware design that are coded in a hardware description language (typically VHDL or Verilog) on register transfer level (RTL) and are run by an event-driven simulation engine, can be turned off[1] during the time they are not needed. This is exactly what has been done within the simulation environment under consideration in this work.

The remainder of this paper is structured as follows. Section 2 discusses related work and Section 3 presents tools and languages used to build the simulation environment. Section 4 thereafter explains in detail the architecture underlying the simulation setup and Section 5 shows the methodology for using it for the design and verification of a prototype distributed system for precision clock synchronization in Ethernet networks. Section 6 presents results from some special simulation runs and comparisons to real-life measurements with the prototype system to underline the validity of the approach. Section 7 finally concludes this paper.

## II. Related Work

The issue of hardware-software co-simulation has already been dealt with in numerous articles. However, as to our best knowledge no virtual prototype simulation environment has been presented up to now that allows to simulate a complete operating system kernel including a file system together with custom driver and application software as well as custom hardware designs. We therefore consider the presented heterogeneous hardware-software co-simulation environment as new.

Clock suppression was first mentioned by Ulrich in [16] but relates to gate-level simulations. The virtual clock synchronization technique presented by Youngmin et al. [17], [9] uses a very similar approach to speed up simulations.

Hannes Muhr is with the Institute of Computer Technology, Vienna University of Technology, Vienna, 1040, Austria (email: hannes.muhr@tuwien.ac.at).

Roland Höller and Martin Horauer are with the Department of Embedded Systems, Vienna University of Applied Sciences Technikum Wien, Vienna, 1200, Austria (email: {hoeller,horauer}@technikum-wien.at).

This work has been partly funded by the Austrian FHplus research initiative within the scope of the DECS (Design Methods for Embedded Control Systems) research project, grant number 811414.

[1] Just like for power saving in power sensitive devices, where clock signals literally are switched off, turning off a clock signal during an RTL simulation saves a lot of simulation events and thus can tremendously speed up the simulation.

However, there models of the operating system are used, rather than an instantiation of a complete OS with kernel and file system.

The GEZEL [13], [12] language and design environment for exploration, simulation and implementation of domain-specific architectures supports only one clock domain, thus is limited to synchronous designs, and its approach to skip cycles, does not improve the simulation performance in case of permanent state activity, e.g. if the hardware design includes periodic counters.

## III. Used Tools

For heterogeneous simulation environments typically used for the design of distributed embedded systems, where hardware, software and network connections are involved, co-simulation of different simulation engines is an essential requirement for a verification setup to be useful. The different tools and languages used throughout this work are hence briefly discussed in the following, before explaining how they interoperate in the next section.

### A. SystemC

SystemC in principle and its application to hardware-software co-verification in particular have been already broadly discussed. Basically SystemC is a system level design language, which is purely based on C++ [5], [15]. By extending a programming language such as C++ with concepts to describe hardware (concurrency, hardware data types, hierarchy, signals, events, timing) a bridge between the two different worlds, hardware and software, is built. It is thus widely used for hardware-software co-simulation efforts and environments.

Since most operating systems are based on C/C++, for co-simulation essential communication techniques such as inter process communication (IPC) are directly supported by C functions, which can be inserted into SystemC code to enable efficient communication between different simulation engines.

In this work, however, SystemC is used for building an interface between a kernel for software execution and hardware models as well as for abstract modeling of hardware parts in an early design phase.

### B. Event-based simulation engine

The industry standard hardware simulator Modelsim from Mentor Graphics supports both standardized hardware description languages VHDL and Verilog. It enables mixed-language simulation at any level of hierarchy. Traditionally, two methodologies exist for the interfacing to the outside world:

- Foreign Language Interface (FLI): A set of Modelsim specific C functions for controlling and observing the simulation. VHDL architectures und subprograms can be replaced with appropriate C functions.
- Verilog Programming Language Interface (PLI): Standardized in the IEEE Standard 1364.

These interfaces are very powerful in terms of controlling the simulation kernel but the encapsulation of foreign functions into a hardware description language requires a lot of code overhead and makes the code less readable.

Since version 5.8 Modelsim supports also SystemC simulation, which is directly integrated into the simulation kernel and thus allows seamless co-simulation of VHDL, Verilog and SystemC. That is actually also the reason, why this simulation engine has been used during the work presented here. This is with no limitation to generality, since the integration of VHDL, Verilog and SystemC could also be done manually, however, at the cost of substantial amount of additional work.

### C. User mode Linux

User Mode Linux is a specialized architecture of the Linux operating system kernel [3]. While User Mode Linux is most frequently used in fields of real-time network simulations, firewalls, and teaching Linux, this work presents as a novelty its application in hardware-software co-simulation.

User Mode Linux runs as a single process within the host operating system. Thus a process that hangs, can be easily suspended and recreated. The whole operating system including drivers and network protocol stacks can be tested in user space.

The necessary synchronization on the hardware-software boundary in the process of co-simulation demands for the software side – as well as for the hardware side – to pause until some access is done. Because user space processes can be halted at any time, co-simulation with a hardware simulator and debugging (stepping through code) via standard tools, e.g. gdb is possible. Hence, User Mode Linux greatly supports driver and software development.

Another reason for the usage of User Mode Linux in a networked environment is its integrated TCP/IP stack, which is well tested since it is utilized without modifications in millions of computers running the Linux operating system all over the world.

Linux' growing importance in the embedded world together with the fact that it is available as open source motivated its usage. However, in principle, the mentioned properties might also apply to other operating systems as soon as they are ported to run in user space.

The virtual prototype presented in the next section makes use of all before mentioned features the User Mode Linux kernel offers.

## IV. Architecture

The purpose of the distributed system developed and verified with the simulation environment presented in this paper is to precisely synchronize the local clocks of networked embedded systems distributed in space and interconnected via Ethernet network technology.

To this end hardware support in the network equipment of network interface cards (NIC) and Ethernet switches is mandatory given the project goal of achieving a synchronization precision of several ten nanoseconds. Furthermore,

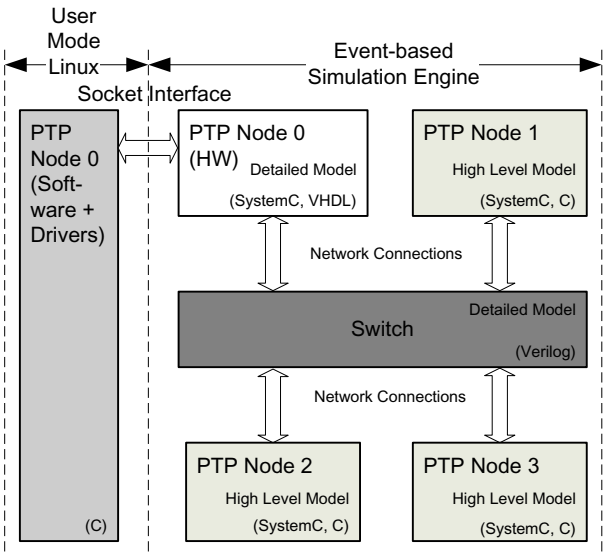


Fig. 1. Overview of the co-simulation setup used for virtual prototyping of a distributed system. Shown are four nodes interconnected via a custom Ethernet switch. On one node (PTP Node 0) a standard TCP/IP socket interface is used for the communication between the event-based hardware simulator and User Mode Linux, where the software is executed. The hardware part is further detailed in Figure 3

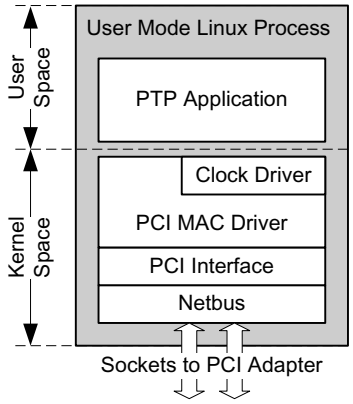


Fig. 2. The User Mode Linux process, which runs on a workstation together with other simulation engines, consists of drivers (in the Linux kernel) together with the application in user space. It serves to model the base system of a node including even a virtual file system.

the IEEE1588 Precision Time Protocol (PTP) not only had to be supported, but protocol and hardware enhancements were at the same time to be built into the system in an orthogonal fashion [8], [7].

By using the tools described in the previous section a system consisting of hardware and software in a networked environment was to be built. Four network nodes running a distributed clock synchronization algorithm should be connected together over a network switch. The flexibility and possibilities of a virtual prototype promised to help in developing and designing such a distributed system (see Figure 1).

The hardware part (VHDL, Verilog, SystemC) was simulated solely with Modelsim. The application software was executed within Modelsim on nodes, which are modeled without User Mode Linux and in the User Mode Linux process for the nodes possessing the socket interface to User Mode Linux. In the hardware simulator the application is encapsulated in three high level PTP nodes using SystemC wrappers. On the fourth node the application is embedded into User Mode Linux as a dedicated process. The boundary between hardware and software on the User Mode Linux node is the PCI bus, which interconnects the host CPU with the NIC. Accesses over the PCI bus happen over standard TCP sockets, which are used for the communication between the hardware simulator and User Mode Linux[2].

Because the PCI core supports both master and target functionality two sockets are utilized, one in blocking mode for master accesses from the host and one for target accesses initiated from the PCI core. The latter operates in non-blocking asynchronous mode to avoid deadlocks during simulation and also to be able to process (asynchronous) interrupts from the PCI core [6].

As User Mode Linux does not contain any bus implementation the PCI functionality has to be added to the kernel. Usually, PCI drivers within Linux only use a small number of functions from the kernel to access the bus[3] [11]. To be able to link a PCI driver to the User Mode Linux kernel these so-called unresolved symbols have been implemented within a PCI interface module (see Figure 2). The netbus module serves as an interface to the communication socket, i.e. it maps the PCI accesses from the PCI interface to the sockets and vice versa.

The actually developed driver software is composed of the PCI Ethernet Media Access Controller (MAC) driver and the clock driver for the hardware clock core. Both drivers and the application software running in User Mode Linux user space are later executed on the real-world target system without altering any code.

Figure 3 shows the hardware part of the NIC design. The existing VHDL testbench (tb_nica) consisting of a PCI master/target model and the NIC was extended with a SystemC PCI adapter and another SystemC unit (tb_nica_ctrl), which improves the simulation performance by clock suppression as explained in more detail below. The PCI adapter bridges the socket interface from User Mode Linux to the PCI master/target model, which does the actual access to the PCI bus.

## V. Methodology

Having discussed the architecture of the simulation environment in Section IV, this section is devoted to the methodologies applied: clock suppression and stepwise refinement of the parts of the system. Both are not claimed as completely new herein, however they are very fundamental in building such a complex virtual prototype.

[2]The distribution of the simulation onto several workstations is easily possible using the socket interface for partitioning.

[3]Some read and write functions for different word sizes (readl, readw, writel, writew,...) and initialization functions.

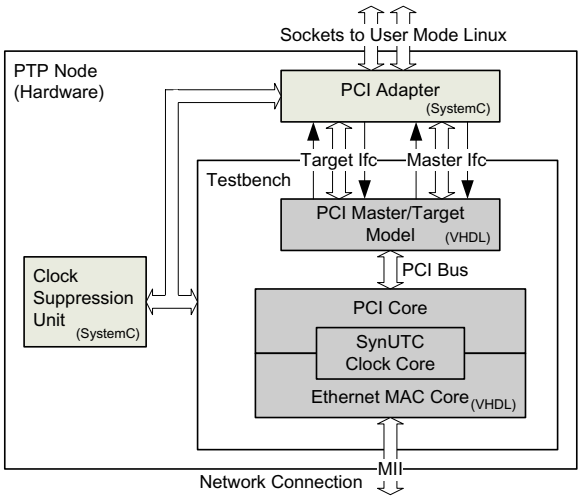


Fig. 3. Simulation setup for the Network Interface Card within the virtual prototype. The PCI Adapter block provides the interface between the software part plus the base system (User Mode Linux) on one hand and the hardware testbench (VHDL, SystemC) on the other hand. One subblock of the testbench is used to implement the presented clock suppression technique.

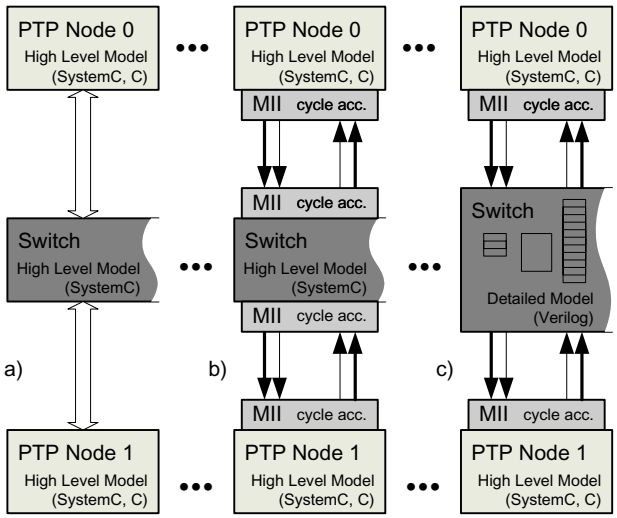


Fig. 4. Basic simulation setups. a) The high level simulation setup with SystemC allowed early evaluation of algorithms and protocols. b) Refined interfacing for connecting four nodes over a switch via the media independent interface. c) The Ethernet switch design is now simulated with all implementation details on register transfer level.

Because of the need to significantly speed up simulation runs – the synchronization of a system can take up to several minutes in real time – clock suppression is implemented in a testbench submodule (see Figure 3). This is necessary, because the clock synchronization protocols exchange information in periods of several seconds rather than in fractions of a second. Whenever there is no activity on the Media Independent Interface (MII), all data packets are processed, and no PCI accesses have to be scheduled the clock signal is turned off to avoid unnecessary events during simulation. To have an accurate system time available within the SynUTC hardware clock core a mechanism for setting the correct time values in the registers of the clock core after a period of clock suppression is necessary.

The virtual time during periods of clock suppression is calculated as

$$v = v_0 + \sum_{n=1}^{N} (t_n - t_{n-1}) \frac{s_n}{s_0},$$

where $v_0$ is the initial virtual time, $t_n$ is the real (simulator) time at timestamp point n, $s_n$ is the step size at timestamp point n and $s_0$ is the initial step rate of the clock.

The whole distributed system was verified alongside ongoing implementation efforts using four simulation setups, which differ in modeling, accuracy, and simulation performance.

### A. Simulation setup I: high level SystemC

Starting point for the development of the simulation environment was a high level SystemC model of network nodes running the PTP application connected together over a high level switch model (see Figure 4a). The number of nodes as well as the number of ports of the switch is parameterisable to easily permit exploration of different network configurations. Transaction Level Modeling (TLM) was used to allow a fast exploration of the distributed systems behavior [5]. Thus a network message is transmitted using a single event on a signal, which carries the whole information of the message.

A PTP node consists of models for the hardware clock, for the network port and a wrapper for the pure C application software. This is an ideal environment for early software development and integration and furthermore starting point for the refinement task.

### B. Simulation setup II: refined MII

The first step of refinement addresses the interfaces between the PTP nodes and the switch (see Figure 4b). Preparing for the next step of inserting the cycle-accurate Verilog switch design, a cycle-accurate Media Independent Interface (MII) model was attached to the high-level SystemC interfaces. Still, this simulation setup allowed to verify and operate the algorithms and protocol stacks with a large number of interconnected nodes to investigate e.g. investigation of scaling of algorithms to large numbers of nodes.

### C. Simulation setup III: replaced switch

The previously generated testbench was then used for the verification of the switch hardware design, which was coded in Verilog HDL for later synthesis, by simply replacing the bus cycle-accurate SystemC switch model with the RTL design. The setup now is restricted to four network nodes, because the specification of the prototype system limited the custom Ethernet switch implementation to four connections. This is not a limitation of the simulation setup.

### D. Simulation setup IV: replaced NIC

Finally, one SystemC PTP node was replaced by the Network Interface Card (NIC) design, which was coded in

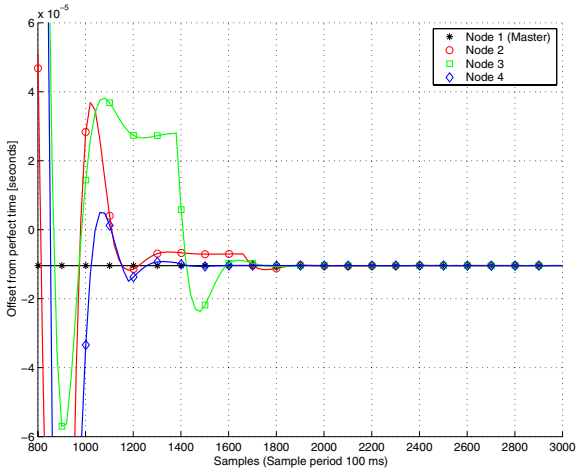


Fig. 5. Local time traces of four nodes showing the project goal of achieving a tight synchronization as long as nodes are connected. The absolute value of the maximum offset between the nodes converges within a few seconds. The nodes reach synchronicity in the 100 ns range at sample 2000. This simulation run has been conducted with simulation setup IV.

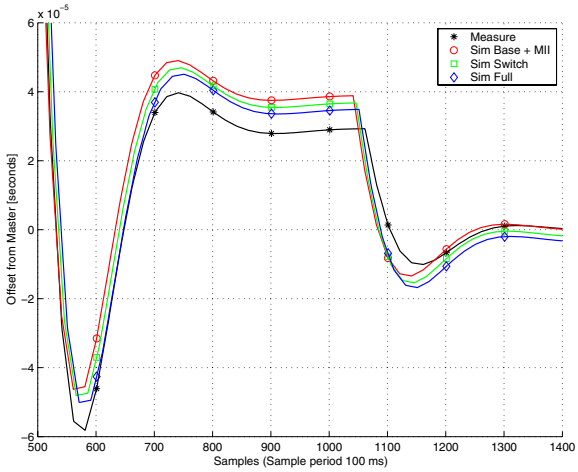


Fig. 6. Comparison between measured and simulated startup behavior of a PTP Node. The difference between the traces is in the us range. The measured offsets have been gained from the comparison of the ten pulse-per-second output signals of the implemented prototype system.

VHDL for later synthesis, extended through PCI models and other testbench units for clock suppression (see Figure 3).

This setup is well suited for the verification of the NIC hardware as well as the driver software for the PCI bus and the hardware clock inside User Mode Linux (see Figure 2). Due to performance reasons not all of the four high level nodes were replaced with the detailed RTL NIC design. As already shown in Figure 1 this final simulation setup is heterogeneous in terms of design languages (C, SystemC, VHDL and Verilog) as well as in terms of simulation engines (Modelsim and User Mode Linux).

A big advantage of the described refinement procedure is, that only the necessary details are added in each step until the whole distributed embedded system and all developed instances including switch, NIC with hardware clock core and MAC/PCI IP-cores, PTP application and device drivers is simulated. The top level testbench can be used with only slight modifications throughout the whole verification process.

## VI. Results

Throughout the design process of a prototype system for precision clock synchronization, where chip designs, driver software, protocol stacks, and application software had to be developed from scratch, a new and efficient heterogeneous hardware-software co-simulation environment has been set up in parallel for functional verification of all parts of the system that were to design anew. This approach, as detailed in Section 4 with respect to its architecture and in Section 5 with respect to the methodology, showed positive effects on the progress of the design process and other contributions as summarized in the following:

- Protocol stack and application software verification was possible as soon as the first, very basic SystemC model of the system was available, thus significantly contributing to the quality of the software and disburden the subsequent system integration process.
- Important system features (in this case the achievable precision of the clock synchronization between the nodes connected via an Ethernet local area network, see Figure 5) were accessible from the very beginning and their assessment and improvement already during the design process was made possible.
- User Mode Linux enabled the development of kernel drivers early in the design cycle with only using the virtual prototype and without having prototype hardware available.
- The usage of User Mode Linux provided the software developers with a real implementation and not just a model of their interfaces very early in the design flow. At the same time, however, still allowing simulation runs of sufficient duration to check the properties of the protocol under development and other system features.
- SystemC allowed the implementation of high level models, which in turn enabled the verification of the distributed synchronization algorithms and communication protocols.
- The presented clock suppression technique impressively helped to speed up the simulation runs of the complete distributed system.
- All previously mentioned items together led to a successful first time right design of a complex distributed system using a virtual prototype.

The simulation performance of the four distinct simulation setups has been compared to show the effect of clock suppression. The results for a 300 seconds simulation run on a Pentium 4 CPU with 3.4 GHz and 1 GB main memory running a 32-bit Linux operating system are listed in Table 1. All setups where cycle-accurate modeling is involved (Sections 5.2 - 5.4) reach a speedup factor of over four orders of magnitude.

| | Simulation Time [sec] | | Speedup |
|---|---|---|---|
| | without C.S. (estim.) | using C.S. | |
| High level (Section V.A) | n.a. | 1,0 | n.a. |
| Refined MII (Section V.B) | $9{,}0 \cdot 10^4$ | 8,0 | $1{,}1 \cdot 10^4$ |
| Included Switch (Section V.C) | $2{,}7 \cdot 10^8$ | $7{,}8 \cdot 10^3$ | $3{,}5 \cdot 10^4$ |
| Full design (Section V.D) | $4{,}5 \cdot 10^8$ | $3{,}0 \cdot 10^4$ | $1{,}5 \cdot 10^4$ |

TABLE I

COMPARISION OF SIMULATION PERFORMANCE FOR A 300 SECONDS SIMULATION RUN WITH AND WITHOUT CLOCK SUPPRESSION (C.S.)

Finally the quality of results of the final hardware-software co-simulation environment has been assessed by comparing the simulated system behavior with the measurements performed with the real-life prototype system that has been developed with the presented simulation environment. As can be seen in Figure 6, the qualitative and quantitative behavior very accurately matches the measured system behavior, delivering impressive evidence of the quality and correctness of the presented heterogeneous hardware-software co-simulation environment using User Mode Linux and clock suppression.

## VII. CONCLUSION AND FUTURE WORK

In this paper a heterogeneous simulation environment has been presented that allows co-verification of hardware and software components of networked embedded systems. The method deployed instantiates a complete operating system kernel so that driver software or protocol stacks can be simulated together with supporting hardware designs on RTL while it is still possible to perform long simulation runs needed especially for protocol verification.

The presented methodology has been used and proofed very useful throughout the design process of a networked prototype system, where drivers, protocol stacks, and hardware for network equipment had to be designed from scratch. The usage of User Mode Linux instead of an ISS together with techniques to significantly speed up long simulation runs with clock suppression during idle periods of the hardware designs, proved to be the most efficient way for early functional verification in this case of an networked distributed system.

The simulation environment itself, however, has not yet been further optimized with respect to e.g. memory consumption or simulation performance. This is subject to ongoing and future research efforts. Extending the presented environment to allow for fault-insertion to asses fault tolerance and to infer timing information into the software execution, namely the User Mode Linux kernel, are also to be investigated in the near future.

## REFERENCES


[1] T. W. Albrecht, J. Notbauer, and S. Rohringer. HW/SW CoVerification Performance Estimation & Benchmark for a 24 Embedded RISC Core Design. In *Proceedings of the 35th Design Automation Conference*, pages 808–811, June 1998.

[2] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino. SystemC Cosimulation and Emulation of Multiprocessor SoC Designs. *IEEE Computer*, pages 53–59, April 2003.

[3] J. Dike. A user-mode port of the Linux kernel. In *Proceedings of the 5th Annual Linux Showcase and Conference*, Atlanta, Georgia, USA, 2000.

[4] F. Fummi, M. Loghi, S. Martini, M. Monguzzi, G. Perbellini, and M. Poncino. Virtual Hardware Prototyping through Timed Hardware-Software Co-simulation. In *Proc. Design Automation and Test in Europe Conf. (DATE 05)*, pages 798–803, 2005.

[5] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[6] V. Guffens and G. Bastin. Running virtualized native drivers in User Mode Linux. In *Proceedings of the USENIX 2005 Annual Technical Conference*, pages 33–39, Anaheim, CA, USA, 2005.

[7] R. Höller, T. Sauter, and N. Kerö. Embedded synutc and ieee 1588 clock synchronization for industrial ethernet. In *Proceedings of the ETFA 2003 IEEE Conference on Emerging Technologies and Factory Automation*, pages 422–426, Lissabon, Portugal, Sept. 16–19, 2003.

[8] IEEE Std 1588-2002. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA, Nov. 2002.

[9] D. Kim, Y. Yi, and S. Ha. Trace-driven HW/SW Cosimulation Using Virtual Synchronization Technique. In *Proceedings of the 42nd Design Automation Conference*, pages 345–348, June 13–17, 2005.

[10] W. Qin and S. Malik. Flexible and formal modeling of microprocessors with application to retargetable simulation. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition DATE'03*, pages 556–561, Munich, Germany, Mar. 3–7, 2003.

[11] A. Rubini and J. Corbet. *Linux Device Drivers*. O'Reilly, second edition, 2001. http://www.xml.com/ldd/chapter/book/.

[12] P. Schaumont, S. Sandeep, and I. Verbauwhede. Extended Abstract: A Race-free Hardware Modeling Language. In *Proceedings of the Third ACM and IEEE International Conference on Formal Methods and Models for Co-Design MEMOCODE'05*, pages 255–256, July 2005.

[13] P. Schaumont and I. Verbauwhede. Interactive cosimulation with partial evaluation. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 2004*, pages 642–647, Paris, France, Feb. 2004.

[14] L. Sèmèria and A. Ghosh. Methodology for Hardware/software Co-verification in C/C++. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 405–408, New York, USA, 2000. ACM Press.

[15] L. Singh, L. Drucker, and N. Khan. *Advanced Verification Techniques: A SystemC Based Approach for Successful Tapeout*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[16] E. Ulrich. A design verification methodology based on concurrent simulation and clock suppression. In *Proceedings of the Design Automation Conference*, pages 709–712, Miami Beach, FL, USA, June 1983.

[17] Y. Yi, D. Kim, and S. Ha. Virtual Synchronization Technique with OS Modeling for Fast and Time-accurate Cosimulation. In *Proceedings of the first International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–6, Oct. 1–3, 2003.