Fast and Unified SystemC AMS - HDL Simulation

Yaseen Zaidi, Christoph Grimm, and Jan Haase
Institute of Computer Technology
Vienna University of Technology
Vienna, Austria
{*zaidi,grimm,haase*}*@ict.tuwien.ac.at*

***Abstract*—An agile methodology for mixed signal simulation is presented allowing seamless connection of simulators on as needed basis eliminating overheads of the communication backplane, sophisticated synchronization and kernel modification. The methodology uses the SystemC AMS synchronization layer which supports user defined solvers and simulators. The cosimulation is wrapped in a statically scheduled timed dataflow node. The simulated executable specification enables co-design, partitioning, refinement, virtual prototyping and architecture exploration of the design space.**

I. Introduction

Heterogeneous systems often comprise of analog, digital, DSP, RF, software or even MEMS components. Such systems are common in communications, automotive, biomedical and defense applications. System behavior is often described at varying level of abstractions using a variety of design methods, languages and tools. The functionality of each subsystem is usually worked out by a specific team with knowledge in a certain discipline. Often there is not a single tool that supports all Models of Computation (MoC) describing the heterogeneous system. At times teams have models written already that they like to reuse. At the system definition phase the architects like to explore architectures or experiment with system tolerance or performance by model refinement. In many cases the system description is an executable specification that is used to backtrack system requirements. Such scenarios demand quick simulations where models can be "dropped-in." Cosimulation is probably the only means where designers prior to design implementation can bring together their pieces of work and study full heterogeneous system behavior in a unified fashion. The advantage of simulator coupling is to use the simulator that has the most appropriate solver for the subsystem [1]. The specification and validation of a heterogeneous system can be realized in a distributed simulation scheme where the system block functions are simulated together by consorting EDA tools. The resulting coupling allows understanding of the heterogeneous model inter-dependencies so as long the coupling is relax and its mechanics are clear-cut. Distributed cosimulations are often a necessity because tools have best performances on specific operating systems and different machine architectures.

Our methodology is "meet in the middle" approach starting with top-down executable specification while cosimulating possible problem areas of the system as fine grained in a bottom-up manner. The remainder of the paper is organized as follows. In Section II we introduce difficulties with simulator coupling and touch on previous works. Section III details our approach and rationale for mitigating problems of Section II. In Section IV we demonstrate a small example. Section V presents outlook and finally, we conclude in VI.

II. Previous Work and Issues

Designers generally abstain from simulator coupling for many reasons. They anticipate that the process would take too long tying the engineering hours in coupling effort than the design itself primarily because:

1) Simulators' interfaces are not known [1].
2) Signals, ports, data types of module blocks and semantics of MoCs would need interface components e.g. polymorphic signals [2], converter channels [3], channel adapters [4] or generic converters [5].
3) A common synchronization scheme [6] leading to the correctness of computation is missing because simulators of different domains consist of different kernels and employ different algorithms or run at different time bases e.g. whether to synchronize at every simulation cycle, i.e. never simulate in future, or use causality constraints to block the computation until constraining conditions clear or let a simulator run ahead of others then rollback restoring some previous known state.
4) A communication [6] mechanism for data transfer among simulators would be needed.
5) Choice of a master or top-level simulator is intricate to make because it must support connectivity and coordination with cosimulators.
6) Simulation speed and accuracy is highly debatable.
7) Designers are not application programmers.

All major tool vendors by and large have some sort of interface for issue 1, typically C language interface [7][8]. Issue 2 is not a showstopper as some adaptation is always necessary. Issue 3 has varying solutions e.g. [9] depending on cosimulation topology. Issue 4 is broadly handled as a backplane [1] [6], a simulation controller [10], a central

simulation coordinator [11] and a cosimulation bus [7]. However, these solutions are only viable if tight coupling is required, centralized exchange at system level is usually a needless overhead [8] at the cost of accuracy. Issues 5 and 6 are absorbed by SystemC AMS itself as a master simulator that is flexible, open source and provides hooks (MoC and synchronization layer) for external simulator connection and its synchronization to SystemC AMS. Finally the interfacing effort as identified in 7 requires no intensive software development. Our technique shall touch on all these problems.

Our goal is to go beyond the executable specification from early on in the engineering cycle i.e. add refined models in the specification which would assist in trading decisions related to co-design, system partitioning, refinement, architecture exploration, virtual prototyping and verification right from the system definition phase. We use SystemC AMS for continuous time computation on top of SystemC which is a discrete event (DE) simulator. We attach SystemC AMS client (master) to VHDL (slave) simulator at server. VHDL computed outputs are ported into SystemC AMS and synchronized by the synchronization layer above the kernel. The external simulator is called from within AMS timed dataflow block and loaded with data driven on block input. The output is computed on remote simulator, received in TCP sockets and is then driven to the output port of AMS block which invoked the cosimulation. The access to remote simulator is possible through the simulator's procedural interface. Since SystemC AMS and SystemC types are native to C/C++ and cosimulator interface is also C based, no special converters or channels are needed for signal and data type conversion.

III. Methodology

A. Client Server Configuration

Figure 1 shows interfaces involved in the distributed cosimulation topology. SystemC AMS acts as a master simulator running at the client computer which connects to the simulator at the server computer. The simulation runs in full automation requiring no user intervention. Two binaries are obtained. The main SystemC AMS description is compiled with the client cosimulating wrapper discussed in Sub-section C. Another wrapper runs at the server as a simple program. Both wrappers communicate via sockets. The client wrapper passes data tokens and necessary information to the server wrapper to compute the MoC which then sets up the simulator after scrutinizing the received information. Setting up the simulator suite is a major task since a series of tools has to be called in the right order and with many options for the simulator environment. This job can be done in two ways: either executing a setup script written in simulator's native scripting language from inside the wrapper; or embedding the setup calls in the wrapper itself as if they would be made from the simulator command line interface. In either case the simulator tools have to be passed the data tokens. Besides the data token, the information related to MoC evaluation such as any other signals required in computation that do not come from the SystemC AMS description e.g. clock, reset are generated in the local testbench at the server. The testbench is an important element in cosimulation, it must drive the model with properly timed and locally generated signals since the model to be evaluated on a mature and commercial simulator is finer than the system level model and thus needs more input signals than simply the input data. Furthermore, the refined models should never be modified for the sake of cosimulation.

B. Timed Data Flow

The Timed Data Flow (TDF) [12] modeling formalism in SystemC AMS stems from the Synchronous Data Flow (SDF) paradigm for describing streaming DSP applications running concurrently. SDFs produce and consume a predetermined number of tokens in each firing. The TDF is a synchronous reactive MoC best understood as a compromise between timed and untimed MoCs. TDF uses predetermined, discrete constant time steps which result in a pool of tokens equally spaced in time [13]. TDF sampled data are tokens whose sampling rates are determined by the system clock. Figures 2 and 3 show a typical TDF cluster and static firing schedule.

At the core of TDF class is the `processing()` member function that is called in `SCA_TDF_MODULE()` whenever module's continuous time behavior is to be evaluated. Discrete event models are precisely timed where every clock tick is accounted for transitions. However such precision at system level modeling is unwarranted because important design decisions become obscure in detailed timing information. Therefore TDF model is suited for streaming DSP applications with constant sampling rates, systems described by transfer functions (input output) that mimic continuous time waves and high sampling systems that signify analog signals.

C. Cosimulation Wrapper at Client

When a module in a TDF cluster is to be cosimulated, a wrapper is instantiated inside `processing()`. The module is modeled as a blackbox that has input ports, port attributes, output ports and `processing()`. But `processing()` does not implement continuous time evaluation behavior as it normally would in absence of cosimulation; rather it calls the wrapper to perform the computation task on dedicated remote simulator. The wrapper encapsulates all necessary functionality needed to evaluate the module. This functionality includes:

1) Format input and TDF attributes and data as strings.
2) Transport data, attributes and socket file descriptor to remote simulator and receive computed results. Optionally echoes are received from the server ensuring

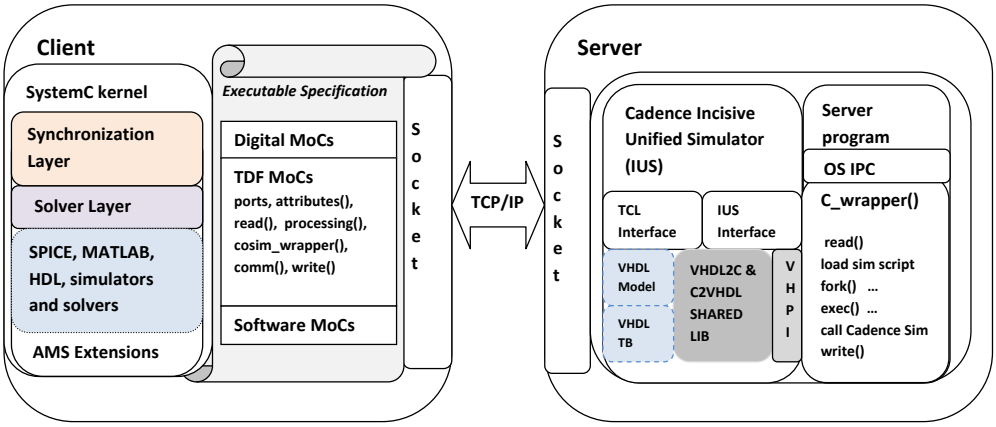

Figure 1. Layered interfaces of client server based distributed cosimulation

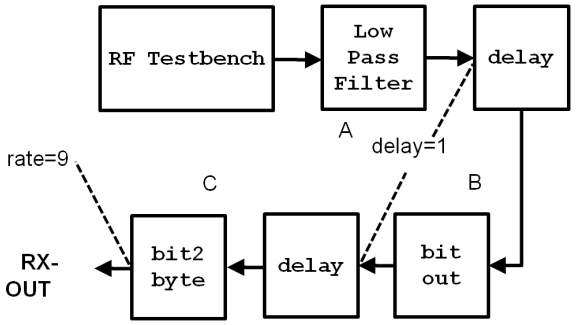

Figure 2. TDF cluster with attributes

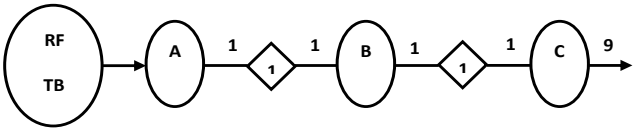

Figure 3. Multirate TDF graph with and schedule vector ABCCCCCCCCC

corrupt data or incorrect buffer sizes are recognized by the server.

3) Communicate any other information, commands, options, delays, etc. that may be useful for the simulator before acting.

The input data and signals are either SystemC or SystemC AMS hardware types. These types are formatted using overloading, static type casting and other built-in C/C++ constructs. All data is eventually converted to C-strings to be accepted by the socket. At the server side, the data is read and written using the simulator's C interface as well. Note that such distributed simulation topology using socket calls with SystemC AMS as master completely eradicates the need of special converters or channels. The wrapper provides a flexible interface for cosimulation as the interface is unique for each cosimulating model with respect to its ports and attributes. The wrapper arguments are the input data and attributes that will be used by the simulator for model evaluation and socket file descriptor set by client OS. The wrapper return argument is the computed result by the remote simulator. For DE models that do not take a clock signal, the wrapper is instantiated in SystemC `SC_METHOD()` and `SC_MODULE()` classes. The execution semantics of `SC_THREAD()` class of SystemC in suspension, forever termination after execution or containing infinite loops are not naturally suitable for hardware modeling and therefore for cosimulation either. In fact `SC_THREAD()` decelerates simulation speed many fold compared to `SC_METHOD()` [14]. The TDF cluster is not traversed to the next TDF module until a return value is received which then is driven to the output port of the module that invoked cosimulation. Thus the main simulation and cosimulation continue in a dataflow cycle visiting each node and executing it in the graph. Multiple modules in the TDF cluster can have their own wrappers e.g. in transmit path of a mixed signal transceiver module encrypting data and in the receive path also for decryption. Therefore multiple cosimulations can be triggered from the top level system simulation.

D. Cosimulation Synchronization

The DE kernels of SystemC at client and VHDL simulator at server run on their own time bases which are not synchronized to each other. However, SystemC AMS synchronization layer registers all the discrete time and TDF MoCs with their signals. The layer synchronizes AMS continuous time models to the DE models (SystemC). It also synchronizes external simulators to the SystemC kernel. An application specific simulator is just another solver to the synchronization layer. The layer determines the fixed step time points at which analog and digital models in static dataflow will be synchronized as well as the execution order of the solvers. This scheme does not burden the kernel by continually synchronizing solvers; rather the decision when to synchronize is determined by the analog stepping which

is an estimation of the truncation error [15] of the infinite Taylor series involved in computation. The truncation error arises due to numerical integration of the finite number of steps. The selection of step size by the layer is related to system clock and sampling rates and it must be large enough to establish that signals have changed their values in the step window. When the C wrapper is invoked at the client requesting cosimulation, this is the point in TDF flow at which the node waits for the computed tokens to return from the cosimulator. Since there is no data available in the node to output, the TDF MoC simply waits for the external simulation to finish. Once values are returned, the external simulator connection is lost and the main simulation continues until a new input is loaded at the MoC blackbox that would again instigate cosimulation. The TDF MoC acts as if its own `processing()` solver is busy computing the outputs. Thus the TDF semantics and AMS synchronization layer when used by the wrapper need no fancy synchronization.

E. C Wrapper at Server

The wrapper at the server runs in the background as a small program in a parent process with a listening socket. The program also checks out the licenses for the requested features of the tools needed in cosimulation. When a connection is made from the client, the wrapper validates the incoming data then spawns into several child processes. Each child process executes a specific tool of the functional simulator. There are as many child processes as the number of tools that would be called. Each child is dead after executing the tool. Figure 4 shows server `fork-exec` calls during execution of the tool chain. At the end of all successful `exec()` calls only the parent process is alive that waits for a new connection from client for next simulation iteration. The Cadence Incisive Unified Simulator (IUS) is a comprehensive functional simulator. The chain of toolset consists of among others a VHDL simulator (NCVHDL), an elaborator (NCELAB) that generates a simulation object file referred to as a snapshot image and the simulator (NCSIM) that simulates snapshot images. Both NCELAB and NCSIM are unified single kernel engines for mixed VHDL, Verilog and SystemC simulations. Precompiled models are elaborated and simulated bypassing NCVHDL child process and preelaborated models are directly simulated by NCSIM bypassing both NCVHDL and NCELAB child processes.

During the simulator execution, the wrapper loads a shared library that accesses VHDL objects discussed in the next Sub-section. The library contains bootstrap or task registering functions that enable C level access and simulation control. This library is created using the Cadence PLI Wizard. For real valued DE ports, the VHDL tools must suppress restrictions imposed by VHDL Initiative Toward ASIC Libraries (VITAL) level 0 compliance (IEEE 1076.4-2000 VITAL ASIC Modeling Specification). These models will not be accelerated.

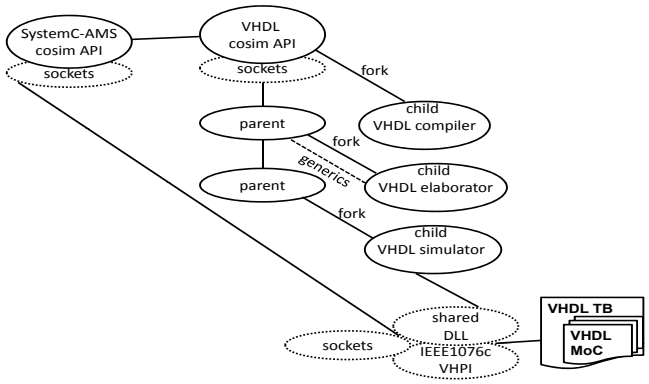

Figure 4. Execution of cosimulation tool chain

F. VHPI based C Level Access

The C level access of VHDL design objects and data for read and write tasks is possible by implementing the VHDL Programming Interface (VHPI). Cadence IUS simulator also provides an extensive C language interface to integrate, import and simulate foreign models written in C. Earlier HDL based cosimulation interfaces [8] have been built around Verilog Procedural Interface (VPI), the VHPI is a new procedural interface released by IEEE. VHPI allows run time access and modification of postelaborated and simulated VHDL design objects in a hierarchal traversal using a reference handle. The VHPI enables simulation interaction and control using C routines. When a VHDL simulation object, no matter how deeply nested in the hierarchy, is to be accessed e.g. component, port or signal a handle (`vhpiHandleT`) defined in the `vhpi_user.h` is returned representing that object. VHPI supports access of wide range of VHDL data types. The ports are written on time resolution base on the clock driven by the testbench while the ports are read off on value change base. For serial lines, the value changes are correctly aligned on time base to assemble complete words. Any number of ports and signals can be monitored for value changes.

For models that require several tokens e.g. an ADC that samples a continuous time wave and produces thousands of samples, large data when received at the server is written to a file and then applied to the VHDL model in the order it was sampled. The corresponding outputs are obtained in order as well and then sent to SystemC AMS. If the sample values do not change much in dynamic range then the values that reflect a borderline are used for faster simulation.

For static models (no new values during computation), data can be input to the elaborator as a C-string argument using VHDL `generic`. The socket file descriptor is also passed as `generic` to the VHDL testbench which outputs it on a port for the shared library to be read. The shared library reads computed data from VHDL

model and writes it to the set file descriptor. A new file descriptor is set by the OS at every new cosimulation evoking by the client and this value is communicated to the VHDL testbench at server. Since port types can be long bit vectors, VHDL `std.textio.all` package converts C-strings passed as `generic` to VHDL data type. All VHPI callbacks for example `vhpi_get_value()`, `vhpi_put_value()` are registered. The simulation time is tracked by `vhpi_get_time()`.

All VHPI related functionality is compiled into the shared library that is dynamically loaded by the NCSIM simulator `exec()` call inside the server wrapper. The simulation start and stop times are passed to the NCSIM `exec()` call as well. If there are various VHPI applications built for different types of designs, it is possible to compile VHPI application as static shared library into the NCSIM simulator executable. Thus new simulator executables can be obtained each augmented with specific VHPI application. Optionally the simulator can be kept online with the waveform analyzer if the design is to be debugged, else the simulator will exit when reaching the stop time. Once the output values are obtained they are saved in the (value, time) data structure. The set socket file descriptor is also available to the shared library and therefore the output data is sent to the SystemC AMS client simulator directly from the VHPI application in the server wrapper. If the client is expected to refine more than one models using scalable cosimulation, multiple wrappers with their own socket ports and shared libraries of VHPI routines can be run within a single server application e.g. one wrapper simulating encryption and other decryption behaviors.

G. Mixed Language Simulation

The methodology can be extended for mixed VHDL-Verilog designs often desirable. The VPI which is Verilog equivalent of VHPI uses similar callbacks for object retrieval, property access and synchronization. In fact VHPI has its roots in VPI which is a more mature interface. The methodology is also equally applicable to any vendor's simulator compliant of VPI/VHPI e.g. Synopsys. Most simulator suites consist of a chain of executables of various tools which can be individually executed by `fork-exec` calls. The methodology allows the use of the most popular languages of ASIC and SoC design spectrum: HDLs, SystemC and their AMS extensions.

IV. Example

Figure 5 shows part of a mixed signal transceiver. The shadowed blocks are modeled in TDF domain whereas regular blocks are of DE domain. The module byte2bit, a boundary block between TDF and DE domains, is a simple parallel to serial converter that uses multirate TDF to convert 9-bit words to single bits. The proposed methodology is applied to this block with a cosimulation wrapper:

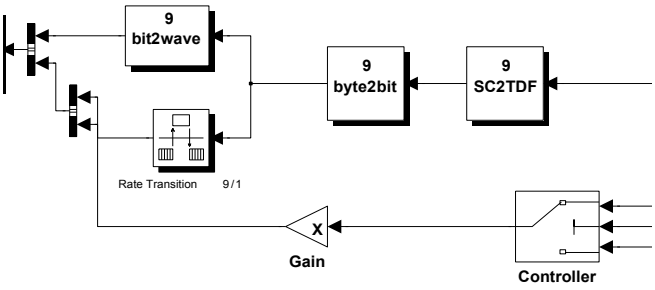

Figure 5. Part of a transceiver model in mixed TDF and DE domains

```
SCA_TDF_MODULE(byte2bit) {
  sca_tdf_in<sc_bv<9> > in port;
  sca_tdf_out<bool> out port;
  void attributes() {
  out.set_rate(9);}
data = in.read();
void processing(){
token = cosim_wrapper
     (c_strings(data, rate, file_desc));
     {socket_comm(data, rate, file_desc);
     commands to spawn Cadence IUS;
     echoes();
     status();}
out.write(to_bool(token));}
}
```

The equivalent model is simulated in VHDL while a testbench uses rate attribute to clock a shift register as:

```
sampling_rate <= conv_integer(rate_slv);
token_vec <= conv_std_logic_vector(token, rate);
file_desc <= conv_std_logic_vector(fd, 8);
s <= s(7 downto 0) & '1';
sca_tdf_out_vhdl <= s(8);
```

Upon receiving tokens, the wrapper at server creates child processes to execute the tool chain as follow.

```
daemon_parent_pid();
nbr_of_bytes = read(new_sock, buf, sizeof buf-1);
validate_messages();
(token, rate, fd) = extract_data(buf);
fork_calls();
execv(NCVHDL, opts, args, "set_rate.vhd", "byte2bit.vhd",
"testbench.vhd");
execv(NCELAB, opts, args, "generic", "token",
"worklib.entity_tb:beh");
execv(NCSIM, opts, args, -input @tcl_script,
"+loadvhpi VHDL2C_DLL", "+start=180", "+stop=350",
"+inst=:uut:comp_byte2bit", "worklib.entity_tb:beh");
```

The VHDL to C shared library reads and records register output events (value toggles and times). For example:

```
EXPIT void
print_vhdl(const vhpiCbDataT *cbPtr)
{sigListP array =
(sigListP)cbPtr->user_data;
fprintf(vcd,"#%llu\n", time2int64(cbPtr->time));
fprintf(vcd,"b%s %s\n",
strtok(cbPtr->value->value.str, "'"),array->id);
if (strcmp(array->id, "%") == 0)
fprintf("cbPtr->value->value.str, b\"%s\");}
..
EXPIT int delVcl(p_cb_data cbPtr)
{
```

```
..
status_code = write (fd, buffer, sizeof buffer);
..
}
```

To form complete words the values are backward/forward filled (carry over) for the positions in which they did not change. The processed values are written to out port of byte2bit module.

The cosimulated results are equivalent to pure SystemC AMS simulation which validates the methodology and interfaces. The run time of the interface computation is dominated by the complexity of VHDL model that is simulated. For precompiled and elaborated models, the times depend only on the execution of NCSIM, the output token rate for each input, the output token length and forward/backward filling for serial outputs. Regardless of the CPU usage, the benefit of cosimulating RTL models with executable specification related to overall design cycle is significant.

V. Future Work

The methodology shall be applied to evaluate complex VHDL models in high level SystemC AMS description. These models will be pure digital, real valued behavioral analog models and mixed signal models. The methodology will be verified for cosimulating multiple models in a single top level description.

VI. Conclusion

Very often designers are faced with interfacing of system level models with refined models written for different domains. We present a simple methodology around single kernel SystemC AMS simulator that uses the TDF model of computation which is at a level of abstraction where various models can nicely co-interact. The methodology is not software intensive to discourage hardware engineers to construct cosimulation interfaces. The TDF paradigm, though not cycle accurate, delivers timed data. SystemC AMS supports hooking peripheral simulators and facilitates their synchronization. The C language based coupling of simulators ensures interoperability. The framework allows mixing different levels of abstraction, domains, languages and tools for an overall simulation with adequate speed.

References

[1] W. Wünsche, C. Clauß, P. Schwarz, and F. Winkler, "Microsystem Design Using Simulator Coupling", In Proceedings of the 1997 European Conference on Design and Test.

[2] C. Grimm, R. Schroll, and K. Waldschmidt, "Refinement of Mixed-Signal Systems: Between Heaven and Hell", in *Advances in Design and Specification Languages for SoCs:* Selected Contributions from FDL04, Springer USA, 2005.

[3] M. Damm, J. Haase, C. Grimm, F. Herrera, and E. Villar, "Bridging MoCs in SystemC Specifications of Heterogeneous Systems", EURASIP Journal on Embedded Systems, vol. 2008.

[4] F. Gharsalli, S. Meftali, F. Rousseau, and A. A. Jerraya, "Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC", In Proceedings of the 39th Conference on Design Automation, New Orleans, LA, USA, June 2002.

[5] S. Orcioni, G. Biagetti, and M. Conti, "SystemC-WMS: Mixed-Signal Simulation Based on Wave Exchanges", in *Applications of Specification and Design Languages for SoCs:* Selected papers from FDL'05, Springer Netherlands, 2006.

[6] B. Agrawal, T. Sherwood, C. Shin, and S. Yoon, "Addressing the Challenges of Synchronization/Communication and Debugging Support in Hardware/Software Cosimulation", In Proceedings of the 21st International Conference on VLSI Design, January 04-08, 2008.

[7] P. L. Marrec, C. A. Valderrama, F. Hessel, A. A. Jerraya, M. Attia, and O. Cayrol, "Hardware, Software and Mechanical Cosimulation for Automotive Applications", In Proceedings of the 9th IEEE International Workshop on Rapid System Prototyping, June 03-05, 1998.

[8] S. Kajtazovic, C. Steger, A. Schuhai, and M. Pistauer, "Automatic Generation of a Coverification Platform", in *Applications of Specification and Design Languages for SoCs:* Selected papers from FDL'05, Springer Netherlands, 2006.

[9] X. Liyi, Y. Yizheng, and L. Bin, "A New Synchronization Algorithm for VHDL-AMS Simulation", Journal of Comp. Science and Tech. vol. 17, number 1, pp. 28-37, Jan. 2002.

[10] C. Lochert, A. Barthels, A. Cervantes, M. Mauve, and M. Caliskan, "Multiple Simulator Interlinking Environment for IVC", In Proceedings of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks Cologne, Germany, September 2005.

[11] A. Maili, D. Dalton, and C. Steger, "A Generic Timing Mechanism for Using the APPLES Gate-Level Simulator in a Mixed-Level Simulation Environment", In 14th International Workshop on Power and Timing Modeling, Optimization and Simulation, Patmos, Greece, September 2004.

[12] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich, "An Introduction to Modeling Embedded Analog/Mixed-Signal Systems Using SystemC AMS Extensions", SystemC AMS Extensions Draft 1, OSCI Analog/Mixed Signal Working Group, June 2008.

[13] J. L. Pino, and K. Kalbasi, "Cosimulating Synchronous DSP Applications with Analog RF Circuits", Signals, Systems & Computers, Conference Record of the 32nd Asilomar Conference on , vol. 2, pp. 1710-1714, Nov. 1-4, 1998.

[14] A. Varma, M. Y. Afridi, A. Akturk, P. Klein, A. R. Hefner, and B. Jacob, "Modeling Heterogeneous SoCs with SystemC: A Digital/MEMS Case Study", In Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis For Embedded Systems, Seoul, South Korea, October 22-25, 2006.

[15] H. Al-Junaid and T. Kazmierski, "An Extension to SystemC to Allow Modelling of Analogue and Mixed Signal Systems at Different Abstraction Levels", In SoC Design, Test and Technology Seminar, Loughborough University, United Kingdom, September 15, 2004.