Proceedings of the 2010 Forum on specification & Design Languages

Southampton, UK
September 14th-16th, 2010

General Chair:

Prof. Tom Kazmierski
School of Electronics and Computer Science
University of Southampton

FDL is an ecsi event!

Table of Contents

Mixed Signal Simulation with SystemC and Saber

Tobias Kirchner, Nico Bannow,
Christian Kerstan
Robert Bosch GmbH
Corporate Sector Research and
Advance Engineering
P.O. Box 30 02 40, 70442 Stuttgart, Germany
Tobias.Kirchner@de.bosch.com

Christoph Grimm

Vienna University of Technology
Institute of Computer Technology

Grimm@ICT.TUWien.ac.at

***Abstract*—Increasing complexity and heterogeneity leads to systems that combine the aspects of both digital hardware/software and mixed-signal embedded systems. A major difficulty is the fact that the components for mixed-signal systems are designed bottom-up, while a digital hardware/software system is designed top- down. Often this requires co-simulation, in practice involving multiple simulators from different vendors and on different platforms. Unfortunately, setting up co- simulations is a time- consuming task which is therefore done only a few times for verification purposes. In this paper we show how a plain SystemC simulation can be connected to Saber. A proxy module interfaces to the SystemC simulation and relays signals to Saber. A special signal synchronisation and update scheme ensures the availability of current analogue values to SystemC starting from the very beginning of each time step. Furthermore we introduce a mechanism for automatically connecting SystemC modules and show how it can be used to implement a graphical SystemC editor. A design example which compares a SystemC to Saber co-simulation to a functionally identical SystemC-AMS simulation is also included.**

I. Introduction

With the complexity and heterogeneity of today's systems increasing at an above average rate [2], the design methodology needs to be adapted in order to keep pace. This is especially important for maintaining a low error rate and to ensure a short time to market [23], [4]. Overall system simulation for executable specification and validating system integration are state-of-the-art [8]. Overall system simulation, due to heterogeneity of nowadays systems involves several simulators, e.g. a circuit simulator for analogue parts, a Verilog simulator for digital circuits, maybe a VHDL simulator for IP components and SystemC for multi-processor systems. In practice, these simulators are from different vendors, and run on different operating systems. Co-simulation based approaches for heterogeneous systems are [5], [17], [19]. It has to be noted that a combined simulation can be more detailed and precise than a conventional single tool simulation [16]. Although the benefits of co- simulation are obvious, the effort of a designer to set up co-simulation has not yet been considered. This effort is especially high in the case of analogue components that are integrated in a "digital" overall system simulation. A major effort is the development of wrapper modules that convert signal representations and interfaces.

In this paper we present a method for co-simulation of SystemC and Saber with minimal effort for the designer. A practical design example will provide essential code fragments and explain all necessary steps in detail. For a prove of concept, the presented approach is compared to an otherwise identical SystemC-AMS co-simulation, which in addition will reveal how simple switching between different simulators can be.

II. Related Work

Algorithm, architecture and hard- and software design are often done sequentially [18]. During this design phase and the following refinement process, simulation has a central role as it is executed several times. It makes use of various different simulators which are linked in order to co-simulate across more than one domain. Examples for simulators often found in electrical heterogeneous systems co- simulation environments are SystemC, Spice, Saber and MatLab Simulink [15], [19].

Although heterogeneous systems often unite digital hardware, analogue hardware and software in one model, each part requires a different simulator or model of computation (MoC) [13], [7]. Non standardised simulator interfaces as well as different MoCs make connecting simulators difficult [9].

There has been previous work aiming at a multi domain simulation environment. A hardware- software co-simulation environment which uses a "bridge" to connect multiple abstract modules is described in [6]. Details of the simulation are moved to the bridge module in order to raise the level of abstraction. The bridge is described and compiled using a bridge specification language (BSL).

Another framework is PeaCE which is based on Ptolemy [11], [20]. It also provides the possibility to connect external tools to the design flow. The limitation to that approach is that it is bound to use Ptolemy as a central element.

A framework for software- hardware co-simulation which uses a proprietary bit stream to transfer data to the simulation peer is proposed in [14]. The project is aimed at verifying a synthesisable SystemC implementation with an existing VHDL design at a very low level of abstraction. The link

between the simulators is optimised for speed rather than simplicity.

To address this problem, Accellera designed a Standard Co-Emulation Modelling Interface (SCE-MI). It is the purpose of this interface to provide a standardised way for linking simulators in a co-simulation environment [1].

Commercial design tools which provide a SystemC co-simulation with an analogue simulator are available from Synopsys, Cadence and Mentor Graphics. Some of them use a modified SystemC kernel. However, all vendors merely mention external interfaces in their documentation.

An open source alternative to extend simulation capabilities is SystemC AMS Extensions. It is a native extension to the SystemC language that aims at analogue designs, especially linear models like transmission lines [10], [8], [3]. It cannot replace a fully featured analogue simulator, but it provides models of computation like the synchronous data flow to simplify connecting other simulators [12].

Intellectual Property (IP) reuse is more important than ever. This is why the SPIRIT consortium [22] has defined IP-XACT, a standard exchange format for IP interface descriptions. The SPIRIT Consortium consists of numerous major EDA and semiconductor companies. IP-XACT uses XML style like meta- data to configure, integrate, and verify IP in SoC design environments. It can describe objects used by the design, i.e. registers and memories by holding attributes like name, sise, and fields, and address space address offset and width.

III. Synchronisation

One of the main differences between an analogue simulator and a digital simulator is the concept of timing they use. Analogue simulators usually are continuous time (CT) simulators. CT simulators advance in time by making small time steps. The simulator calculates its result depending on the current time ($f(t)$). New simulation results are only calculated once every time step.

In order to combine analogue and digital simulators in one co-simulation their execution needs to be synchronised at specific points in time. Figure 1 shows the basic synchronisation principle we use. There are other, more efficient ways to synchronise with an analogue simulator but they require tighter interaction with the solver than the Saber smmi interface, which we use, provides.

A. Analogue Events

We chose to synchronise SystemC and Saber when a digital event occurs. This allows SystemC to access and process analogue values whenever it needs to. This however does not permit modelling of an interrupt pin as this would require Saber to generate an additional event in SystemC.

To achieve this we use certain conditions of the analogue signal which if they occur could be called analogue events. Analogue events are generated by the analogue simulation if a group of values matches certain criteria. These may be extrema, threshold values or turning points. Applied to the

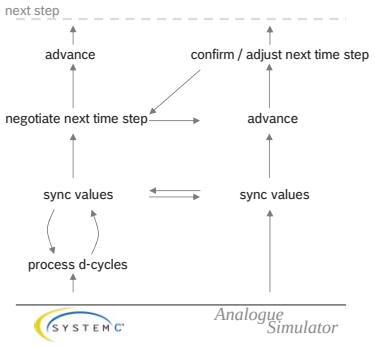

Figure 1. SystemC to analogue synchronisation.

synchronisation described in III the next event used by the analogue simulator would not be the next time step but an analogue event. When asked for its next event during the synchronisation process, Saber would now run until the next SystemC event unless it encounters an analogue event on its way.

This way analogue events perform two vital tasks. First of all they are an essential part of a analogue- digital simulator coupling which requires little communication. Second analogue events resemble very much trigger like they are used in oscilloscopes. Mixed signal designers usually are familiar with the concept of oscilloscope triggers and should therefore be able to use analogues events intuitively.

B. Sampled Signals

Through the use of analogue events In the SystemC to analogue co-simulation values are updated between the simulations at every SystemC time step. Synchronisation between simulators is done by the last process of each time step to avoid unnecessary communication of unsettled SystemC signals. The synchronisation causes signals to be updated with current analogue values. Processes which get triggered by these signal updates may use the updated value right away. Unfortunately this also means that processes which ran before this update may have used an outdated value

For pure SystemC designs this is no problem, on the contrary, delta cycles are an integral part of the SystemC methodology. As the simulation ticks away towards its steady state, some processes may be executed repeatedly, using updated values.

Processes which only read a signal without being sensitive to it would require a signal update at the beginning of each time step. Unfortunately there is no way to make sure that the proxy module which handles the signal updates runs as fist of all modules. There is, however, a SystemC function which can check if there are any more delta cycles to follow in the current time step. This allows the proxy module to make itself wait repeatedly until it is last. It can then update current SystemC simulation results to the analogue simulator.

C. Pre- Synchronisation

For performance reasons it is not sensible to make SystemC modules sensitive to constantly changing analogue signals. However, when modelling e.g. an A/D converter, it may be of great advantage for the designer to be able to sample the analogue signal. Since the analogue synchronisation is done last in a time step, the modules running before that process outdated analogue values. Because SystemC and Saber are synchronised only every SystemC time step, the actual age of the outdated analogue value is neither predictable nor constant. The problem of indeterminably outdated analogue samples only occurs when the signal is to be read from the analogue side rather than it being actively written.

SystemC-AMS basically shows the same behaviour only here analogue tokens are generated and updated in fixed intervals. The maximum age of an analogue SystemC- AMS token is given by its set sampling rate. Assuming that this sampling rate is set to sensible values, the potential error introduced is much smaller. [21]

To resolve this the update procedure has been extended. When SystemC asks Saber to make a time step it will be executed being a little bit shorter than requested. After that analogue results are transferred to the SystemC side. At this point in time no scheduled SystemC task runs and therefore only the simulation results are processed. The SystemC time step which was originally scheduled follows, however, it can now access analogue values which have just been updated.

SystemC assumes that Saber has not yet reached the desired simulation time because it has encountered an analogue event. However, since no trigger have fired no port updates happen and finally no SystemC processes run. Due to this intermediate step SystemC simply asks Saber to proceed to the final time of the step in progress. Saber does indeed proceed to that simulation time because it detects that the time step that is being requested now is sufficiently small. Analogue values which are available on the digital side are already up to date.

IV. Usability

A. SystemC Proxy Module

The proxy module is used for transferring SystemC signals to the analogue Simulator. Figure 2 shows its main components.

The signal map is used by the proxy module to keep references to all signals that have to be synchronised with Saber. The `synchronise()` routine synchronises all signals in the signal map between Saber and SystemC. It iterates through the map of signals and checks for modifications. If a signal has changed, its value is handed over to the protocol. The protocol receives and transmits values between SystemC and Saber.

The most important part of the proxy module is the `LastDeltaCycle()` function. It is registered as a process within SystemC. Its only task is to

- wait() repeatedly until there are no pending processes in this delta cycle
- call the synchronise() routine
- schedule itself for the next time step

Listing 1 shows the corresponding code. The `sc_pending_activity_at _current_time()` function is publicly available as of SystemC 2.2.0. It returns true if there are any more processes ready to run in the current time step.

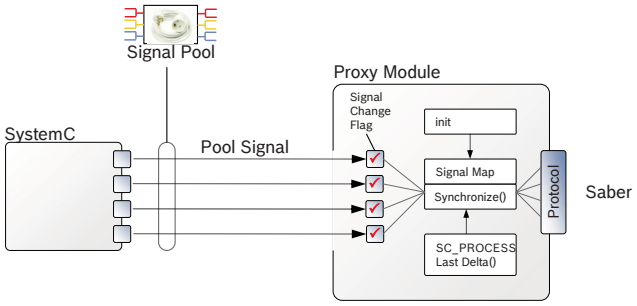

Figure 2. The proxy module connects SystemC signals to Saber.

```
1
2  SC_THREAD(lastDeltaCycle);
3  ...
4  voidSystemCCoSim::lastDeltaCycle() {
5      while(true) {
6          do{          // repeat until everything is fine!
7                       // wait until this thread is the very last one
8              while(sc_pending_activity_at_current_time()) wait(SC_ZERO_TIME);
9                       // perform data exchange & consistency check
10             if(!Synchronise()) sc_stop();
11                      // are there any pending events caused by external changes?
12         } while(sc_pending_activity_at_current_time() || PortsChanged());
13                      // set next relevant event (schedule myself)
14         CoSimNextRelevantEvent.notify(sc_time(NextExternalEventTime,
               SC_SEC) - sc_time_stamp());
15                      // wait for next external or internal event
16         wait(CoSimNextRelevantEvent);
17     }
18 }
```

Listing 1. The proxy module repeatedly waits until it is the last module in the current time step. It then synchronises signals with Saber.

The proxy module is an `sc_module`. Instead of using ports, it accesses signals directly. This makes it independent from the design. Without any ports which need adapting it can easily be re-used for different designs. This also brings the benefit of having a large piece of known good code with various plausibility checks already available. For connecting the proxy module and the design we use the signal pool.

B. Signal Pool

The signal pool is like a name server for SystemC signals. SystemC modules may request to be bound to a signal with a given name. If the requested connection name matches a signal which is already available inside the signal pool, the module is connected to it. If there is no matching signal available it will be created. The signal is created by the signal pool but exists independently. The signal pool does not interfere with SystemC, leaving all runtime and elaboration checks intact. By using a singleton the signal pool itself can be accessed throughout the entire simulation by its name.

A designer who wants to establish a connection between two modules would first acquire the reference to the signal pool and then use its `bind` routine. It requires the signal

name and the port reference as parameter. The `bind` function also accepts the name of a trace file as optional parameter. An example can be seen in listing 2. Like the signals themselves, trace files will also be created on demand. This saves manually creating trace files and lets the designer tag signals for tracing right where they are created. The signal pool can be used as interface for the proxy module. In that case the proxy module registers itself with the signal pool as a further source of signals. If the signal pool is asked to connect a signal, it tries to establish a connection to the proxy module first. If a requested connection is available from the proxy module, the signal is created and connected to the proxy module, i.e. added to the signal map.

C. Graphical Design Entry

It is the nature of SystemC to be a programming based digital simulator, however, a graphical editor would greatly improve its usability. This is especially important for those contributing to a mixed signal design without being a digital expert.

Such a graphical editor would preferably have the ability to export the design using IP-XACT, a XML based format specially designed to store structural information of model based designs. It can hold information about the used components, their interfaces and interconnections.

This stored design information can be read and processed by SystemC. This means that first objects or models need to be created. In a second step they are then connected by signals. Both steps need to be completed before the elaboration phase to take advantage of all SystemC run time tests. Once elaboration starts, the simulation behaves like any other SystemC simulation. Dynamically creating objects before the simulation starts has no influence on the simulation performance.

In order to create the SystemC objects according to the information contained in the IP-XACT file we use a factory. It is a piece of code which can create objects at runtime- but only those which were known when it was compiled itself. The factory is necessary because C++ cannot create arbitrary objects at run time with nothing but the object type name available.

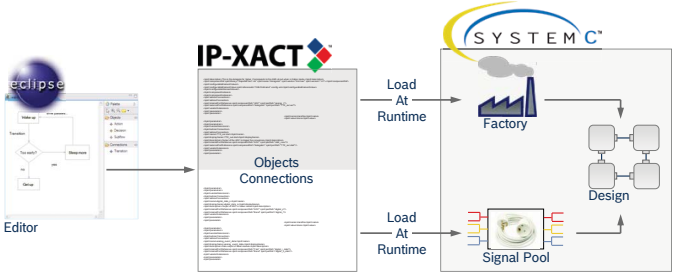

Figure 3. At simulation runtime the design structure is read from a IP-XACT file. A factory then creates SystemC modules and the Signal Pool connects them.

Once all objects have been created, the second part of the IP-XACT file is evaluated by the signal pool. It automatically reads the signals name from the file and consults the factory to create the correct type of signal.

Large parts of the simulation such as its structure or model parameters can now be entered through the IP- XACT file at run time without the need to re-compile the simulation. SystemC as simulator remains hidden from the user. This reduces the complexity of designs and makes SystemC more usable for engineers. In fact, they may not even know they are using it.

.

V. Design Example

The example resembles common engine control unit (ECU) structures in the automotive industry. Data acquisition is mostly done in hardware. Data processing is done by tasks which run independently and asynchronously. Timing sensitive actions are supported by timer units which can be programmed to perform a certain action at a pre-defined time. A overview of this example can be seen in Fig. 4.

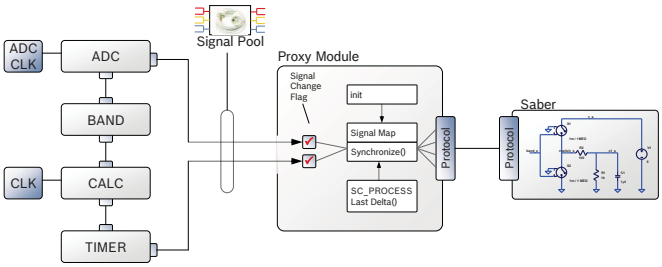

Figure 4. SystemC to Saber co-simulation example.

The example maintains a constant voltage across a capacitor by controlling a push pull stage in Fig 5. The SystemC modules on the left side of Fig. 4 sample an analogue signal (ADC), apply threshold values (Band), calculate the next output stage event (Calc) and execute it (Timer). The analogue circuit is simulated with Saber. It is connected through the proxy module.

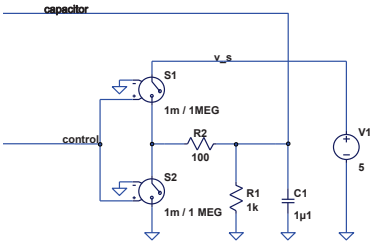

Figure 5. Schematic of the design example.

The output of the circuit can be seen in Fig. 6. The ripple is caused by the hysteresis of the control algorithm, the jitter is owed to the asynchronous calc process.

A. Design Effort

In a classical SystemC design setting up the SystemC of a Saber co-simulation would require three main steps. First of

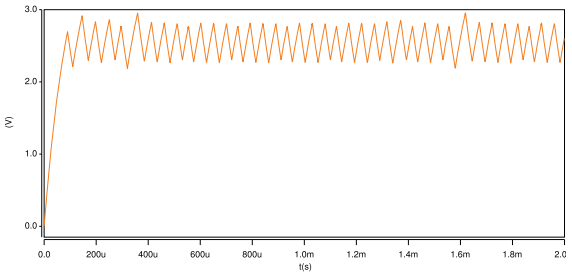

Figure 6. Analogue results of SystemC-AMS and Saber are identical.

all the interface of the proxy module has to be defined. Ports used to connect it to the SystemC design have to be created. Then, the internal synchronise function has to be modified to transfer data from and to the interface ports. Finally the proxy module has to be connected to the design. Through the use of the signal pool these steps can be reduced significantly.

```
1  Testbench::Testbench(sc_module_name mn) : sc_module(mn) {
2     calc_task_clock  = newsc_clock ("calc_task_clock", sc_time(10,SC_US));
3     SystemCSignalPool* signalpool = SystemCSignalPool::getSignalPool("TLD");
4
5  #ifdefSABER
6      SCProxyModule* SaberModule = new SCProxyModule("nm", "TLD", "config.xml");
7  #else
8     i_amscircuit = new amscircuit("i_amscircuit");
9      signalpool->bind(i_amscircuit->switch_control_port,"VLeft.value");
10     signalpool->bind(i_amscircuit->capacitor_voltage_port,"TTR.out","Trace");
11 #endif
12
13 i_adc = new adc("i_adc");
14     signalpool->bind(i_adc ->digital_o,"digital_data_s",sc_traceFile);
15     signalpool->bind(i_adc->analog_i,"TTR_out.out",sc_traceFile);
16
17 i_band = new band("i_band");
18     signalpool->bind(i_band ->digital_o_data,"analog_event_data","Trace");
19     signalpool->bind(i_band -> digital_o_time,"analog_event_time");
20     signalpool->bind(i_band ->digital_i,"digital_data_s");
21   ...
22 }
```

Listing 2. First part of the top level design (TLD) of the example.

Listing 2 shows parts of the top level design (TLD).The number of lines of code for the top level design has been reduced by about 75%. The signal pool provides the means for Signal definition, instantiation, binding and tracing with just one line of code. The proxy module can be used directly without any modification. Both, signal pool and proxy module are SystemC modules and therefore they can easily be included in any design. There are no modifications of the SystemC kernel required.

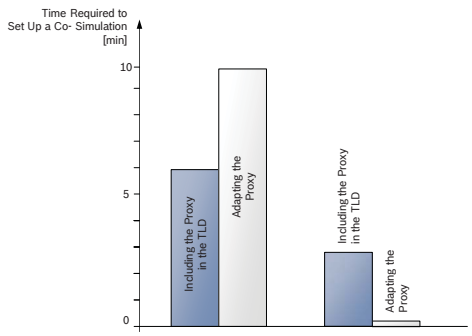

Figure 7. Design effort for the proxy module (in minutes).

The design efforts in lines of code can be seen in Fig. 8. The corresponding time effort can be seen in Fig. 7.

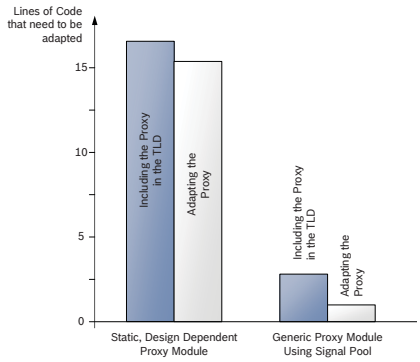

Figure 8. Design effort in lines of code for the proxy module.

B. *Performance and Accuracy*

To compare the design efficiency to SystemC-AMS the analogue circuit has also been simulated with SystemC- AMS. Fig. 9 shows the modified structure. The proxy module is not needed. The signal pool remains in the design because it allows to simply plug in the AMS simulation without any further changes to the design. Switching between simulators is done in the #ifdef command of Listing 2.

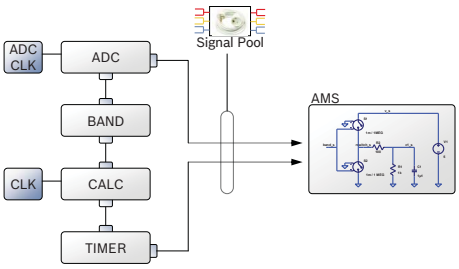

Figure 9. The example from Fig. 4 shall also be co-simulated with SystemC-AMS.

Comparing the two simulation results one can see that they are very similar. The result of the example being co- simulated with SystemC- AMS is almost identical to the example being co-simulated with Saber. Fig.10 shows the difference between the simulations. The deviations are given in per mille.

Deviations are a little bit lager at the beginning of the simulation. The reason for this is that SystemC- AMS and Saber seem to have different strategies for the DC bias point calculation. In Fig. 6 Saber starts with 0V while SystemC-AMS claims that the voltage of the circuit at $t = 0$ is several micro volts.

Figure 11 shows the simulation times. Each simulation was compiled using Visual Studio Express 2008. They were both run in release and debug mode. It can be seen that SystemC- AMS greatly benefits from running in release mode as the whole simulation is affected. The Saber co-simulation execution time is almost identical. The reason for that is that most time is consumed by Saber itself. Taking into

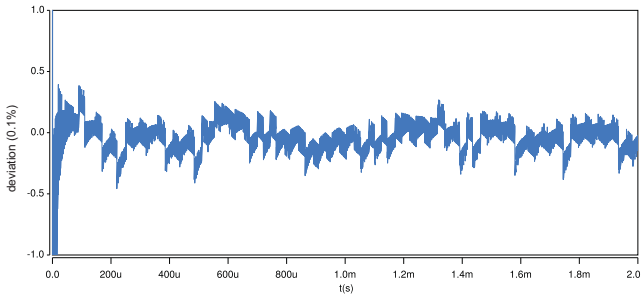

Figure 10. Deviation of Saber and SystemC-AMS results differences in per mille.

consideration that Saber is a fully featured analogue simulator, simulation times are well within a competitive range.

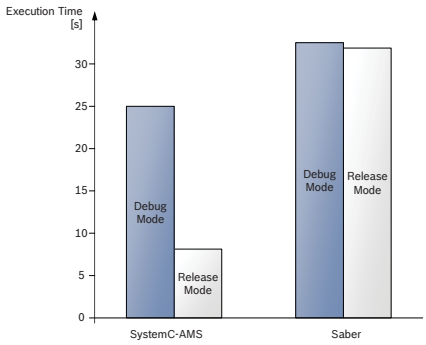

Figure 11. Simulation times for Saber and SystemC-AMS co- simulations in seconds.

VI. Conclusion

This paper has shown how SystemC can be connected to Saber with little effort. It has outlined a synchronisation strategy which only updates simulation results on and immediately before SystemC events. Through the introduction of analogue events which are generated by the analogue simulator, SystemC signals can now become sensitive to analogue signals.

The proxy module which is responsible for signal synchronisation on the SystemC side of the co- simulation uses the signal pool to interface to the simulation. It is therefore completely independent from the design. By making signals accessible by their name, the signal pool is also a key element for a graphical SystemC editor.

The design example has demonstrated improved usability and the correctness of our approach. It has also shown the competitive speed with which a fully featured analogue simulator like saber can be used inside a SystemC simulation. Co-simulation effort set up times have been reduced by approximately 75 per cent. At the same time the simulation overhead and thus complexity of the co-simulation were reduced.

In times with steadily increasing system complexity and rising need for overall heterogeneous system simulations improved usability is more important than ever.

References

[1] Accellera. Standard Co-Emulation modeling interface (SCE-MI) reference manual, March 2007.

[2] Daniel Amyot and Gunter Mussbacher. Bridging the requirements/design gap in dynamic systems with use case maps (UCMs). In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 743–744. IEEE Computer Society, 2001.

[3] Martin Barnasconi. SystemC AMS extensions: Solving the need for speed. *White paper on Analog/Mixed-signal (AMS) 1.0 standard*, May 2010.

[4] A. Bernstein, M. Burton, and F. Ghenassia. How to bridge the abstraction gap in system level modeling and design. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 910–914, 2004.

[5] M. Briere, L. Carrel, T. Michalke, F. Mieyeville, I. O'Connor, and F. Gaffiot. Design and behavioral modeling tools for optical network-on-chip. 1:738–739 Vol.1, 2004.

[6] Ping Hang Cheung, Kecheng Hao, and Fei Xie. Component-Based Hardware/Software Co-Simulation. pages 265–270, 2007.

[7] Moo-Kyoung Chung and Chong-Min Kyung. Enhancing performance of HW/SW cosimulation and coemulation by reducing communication overhead. volume 55, pages 125–136, 2006.

[8] Karsten Einwich, Alain Vachoux, Christoph Grimm, and Martin Barnasconi. SystemC AMS extensions draft 1. December 2008.

[9] L. Gheorghe, F. Bouchhima, G. Nicolescu, and H. Boucheneb. Formal definitions of simulation interfaces in a Continuous/Discrete Co-Simulation tool. pages 186–192, June 2006.

[10] Christoph Grimm, Karsten Einwich, and Alain Vachoux. SystemC-AMS reference manual, April 2005.

[11] Soonhoi Ha, Sungchan Kim, Choonseung Lee, Youngmin Yi, Seongnam Kwon, and Young-Pyo Joo. PeaCE: a hardware-software codesign environment for multimedia embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):1–25, 2007.

[12] Fernando Herrera and Eugenio Villar. A framework for heterogeneous specification and design of electronic embedded systems in SystemC. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):1–31, 2007.

[13] A. Herrholz, E. Oppenheimer, P.A. Hartmann, A. Schallenberg, W. Nebel, C. Grimm, and M. Damm. Analysis and design of Run-Time reconfigurable heterogeneous systems. In *International Conference on Field Programmable Logic and Applications*, 2007.

[14] A. Hoffmann, T. Kogel, and H. Meyr. A framework for fast hardware-software co-simulation. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 760–764, 2001.

[15] Open SystemC Initiative. IEEE 1666-2005 standard SystemC language reference manual. December 2005.

[16] A. Jantsch and I. Sander. Models of computation and languages for embedded system design. *Computers and Digital Techniques, IEE Proceedings* -, 152(2):114–129, March 2005.

[17] A.A. Jerraya, F. Rousseau, A. Bouchhima, M.-W. Youssef, A. Grasset, W. Cesario, L. Kriaa, and A. Sarmento. Service dependency graph: an efficient model for hardware/software interfaces modeling and generation for SoC design. pages 261–266, 2005.

[18] E. W. Johnson and J. B. Brockman. Measurement and analysis of sequential design processes. *ACM Trans. Des. Autom. Electron. Syst.*, 3(1):1–20, 1998.

[19] Tobias Kirchner, Nico Bannow, and Christoph Grimm. Analogue mixed signal simulation using spice and SystemC. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, pages 284–287, April 2009.

[20] Edward A. Lee. Overview of the ptolemy project. *Technical Memorandum No. UCB/ERL M03/25*, University of California, Berkeley, July 2003.

[21] Rajeev Narayanan, Naeem Abbasi, Mohamed Zaki, Ghiath Al Sammane, and Sofiene Tahar. On the simulation performance of contemporary AMS hardware description languages. In *Microelectronics, 2008. ICM 2008. International Conference on*, pages 361–364, 2008.

[22] Schema Working Group of The SPIRIT Consortium. IP-XACT Draft/D5: a specification for XML meta-data and tool interfaces, May 2009.

[23] G.K. Rauwerda, P.M. Heysters, and G.J.M. Smit. Towards software defined radios using Coarse-Grained reconfigurable hardware. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(1):3–13, 2008.