Virtual Synchronization for Fast Distributed Cosimulation of Dataflow Task Graphs

Dohyung Kim Chan-Eun Rhee Youngmin Yi
Sungchan Kim Hyunguk Jung Soonhoi Ha
CAP Laboratory
Computer Engineering Dept.
Seoul National Univ., Seoul, Korea
+82 2 8807292
{dhkim, chaeni, ymyi, ynwie, noirsens, sha}@iris.snu.ac.kr

ABSTRACT
Fast distributed cosimulation is a challenging problem for the embedded system design. The main theme of this paper is to increase simulation speed by reducing the frequency of inter-simulator communications, reducing the active duration of simulators and utilizing the parallelism of component simulators, which is accomplished by combining event-driven and data-driven simulation methods. The proposed technique is applicable when the simulated tasks follow dataflow execution semantics. Experimental results show that the proposed technique can boost the cosimulation speed significantly compared with the previous conservative approaches.

Categories and Subject Descriptors
B.7.2 [**Design Aids**]: Simulation and Verification

General Terms: Performance, Verification

Keywords
Cosimulation, distributed simulation, time accurate simulation, virtual synchronization.

1. INTRODUCTION
A complex embedded system usually consists of software modules and hardware modules that are mapped to heterogeneous components such as programmable processors, customized ASIC, and IPs. Though several efforts are invested to model all modules in a single simulation platform, a current practice of system-level simulation is likely to involve communication and synchronization between component simulators, sometime geographically distributed, to make it a distributed cosimulation. For time-accurate simulation, component simulators are basically event-driven or time-driven that the simulator processes events in the chronological order [1].

As system complexity increases and fast design turn-around time is required, increasing the simulation performance becomes more important specially because the simulation complexity is a super-linear function of design size. The performance bottleneck of distributed event-driven simulation comes from the huge overhead of time synchronization and data communications.

A typical technique to minimize data communication overhead is to group data samples and send them all at once. Minimizing time synchronization overhead is more difficult. A naive but expensive solution is to let the main simulator marshal the local clock advancement of each component simulator by paying extra communication overhead and waiting delay. Better solutions allow the component simulators to advance their clocks asynchronously and to exchange clock information as few times as possible. They are classified into optimistic [2] and conservative [3] approaches. In this paper, we focus on conservative distributed approaches to accommodate existent component simulators without rollback capability.

Another performance bottleneck is the simulator performance because time accurate simulation for software or hardware is very slow. Compiled co-simulations make it faster but sacrifice accuracy [4][5]. In-circuit emulators boost the speed [6] but they require special hardware components.

The main theme of this paper is to increase the simulation speed by reducing the frequency of inter-simulator communications, reducing the active duration of simulators, and utilizing the parallelism of component simulators, without modifying the component simulators. Though the proposed technique is based on the assumption that the simulated tasks follow dataflow execution semantics, it is applicable to more general cases where task execution results do not depend on the arrival times of input data but their arrival order.

First, we propose a new distributed event driven simulation technique of component simulators. Our approach uses a cosimulation backplane as the master process in the distributed event driven (DED) cosimulation. Furthermore, we present a novel technique to reduce the time synchronization and communication overheads significantly by combining event-driven and data-driven simulation methods. Previous distributed event-driven simulation techniques do not assume any special execution semantics of the simulated tasks. Execution semantics allows us to minimize the communication overhead, while not violating the causality condition of the event driven simulation.

In the next section, we explain the execution model of task graphs and the environment assumed in this paper. Section 3 explains the proposed distributed event driven (DED) cosimulation technique and compares it with the existent techniques. Section 4 discusses the main theme of this paper: how to use data-driven scheduling information for the DED cosimulation. Experimental results and conclusions will follow in section 5 and 6 respectively.

2. COSIMULATION ENVIRONMENT

In our proposed environment, we specify control and function modules separately on the codesign/cosimulation backplane using FSM model for control modules and synchronous dataflow (SDF) model for function modules. In a SDF graph, a block represents a function that transforms input data streams into output streams. An arc represents a channel that carries a stream of data samples from the source block to the destination block. The number of samples produced (or consumed) per block firing is predefined. An SDF model has a restriction that a block becomes runnable only after all input ports have the predefined number of input samples.

Figure 1(a) shows an example system specification which is hierarchical and compositional. We assume that blocks A1 and A2 generate simulation vectors and block E displays the result. Figure 1(b) shows a corresponding architecture template. We assume that hierarchical blocks B and D are mapped to the DSP and the ASIC component respectively while block C is to the IP component and the control module to the micro-controller. After mapping between function modules and architecture components is determined, we establish a distributed cosimulation environment as illustrated in Figure 1(c). All component simulators establish their outside connections only to the backplane. The backplane is basically an event-driven simulator and a centralized backbone for communication between component simulators [7][8].

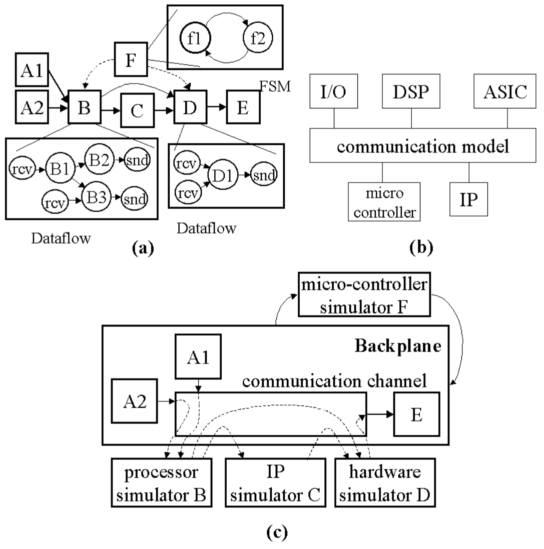

Figure 1. (a) An example system specification in the pro-posed methodology. (b) An example architecture template. (c) Cosimulation environment that consists of the cosimulation backplane and component simulators.

3. DISTRIBUTED EVENT DRIVEN COSIMULATION

In this section, we propose a new distributed event driven cosimulation technique which enhances parallelism between component simulators. The main difficulty to speed up the distributed event driven simulation is to synchronize the simulators so that no simulator violates causality condition of event processing. Several techniques for accelerating distributed discrete event simulations have been proposed since late 1970's and largely classified into two approaches: optimistic [2] and conservative [3].

In the optimistic approach, a component simulator may advance its local time optimistically assuming that no past event will arrive. If that assumption fails, it rolls back its local time to the event arrival time or the earlier checkpoint time, and cancels all processing results after the adjusted time, paying huge over-head [9].

To accommodate component simulators without rollback capability [10], we take the conservative approach where a component simulator advances its local time and processes events only after it is guaranteed that no event will arrive earlier [11][12][13]. There are two schemes to satisfy this causality condition: centralized approach and distributed approach.

In the centralized approach, the central controller manages the component simulators with the information how far the local clock of the simulator can advance. The centralized approach serializes all communication activities between simulators in the global queue although there is no dependency between them. It is the main cause of low performance.

In the distributed approach, there is no need of a central controller. Instead, each simulator should wait until it receives input events from all input ports and process the event of the smallest time-stamp. The main difficulty of the distributed approach is deadlock possibility. If there is a cycle among simulators, no simulator can receive events from all input ports.

One solution is to use null messages that carry the lower bound information of the time-stamp value of the next event [14]. On the other hand, Chandy and Misra [3] allow deadlock situation. When deadlock situation occurs, a central controller initiates the recovery phase in which each simulator exchanges time information with other simulators and updates the time until when it can safely advance its local clock. Since the overhead of deadlock detection and recovery is significant, the simulation performance degrades proportionally to the deadlock frequency.

The proposed scheme lies in the middle of the centralized and the distributed approaches. As a centralized approach, all data exchange between component simulators goes through the cosimulation backplane. Unlike the centralized approach, however, the backplane does not serialize the communication activities. Instead, each component simulator waits until it receives input events from all input ports, and processes the event of the smallest time-stamp like the distributed approach. Therefore, the proposed simulation is based on the idea of deadlock detection and recovery. The difference between Chandy and Misra's approach is that the cosimulation backplane plays the role of the central process without paying extra communication overhead to find a runnable simulator.

Figure 2 illustrates how the proposed scheme works. We need to introduce a simulator interface between the backplane and a component simulator, which contains wrap-up code for deadlock detection and recovery since the component simulator is not expected to have this capability. The simulator interface has a local queue to check causality conditions, which confirms that there arrives no earlier event after processing a current event. And it manages a local clock related to the simulator. The event produced from the predecessor is delivered to the associated local queue of the destination block. Because events are stored in the distributed local queues, the backplane executes the simulators in parallel as long as they meet causality conditions. When a deadlock occurs, it finds the earliest event in all local queues and makes it delivered although it does not meet local causality condition [3].

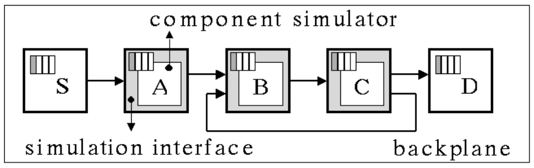

Figure 2. The proposed cosimulation scheme

There is a significant difference between the cosimulation problem and the typical DED simulation problem. In a typical DED simulation, each process is assumed to finish processing the current input event before accepting the next event. In other words, the local process is non-preemptive. On the other hand, the component simulator performs itself an event-driven simulation so that it advances the local clock while processing events. Without the support of rollback mechanism, the local clock should not be ahead of the time-stamp of the next event. During processing the current event, the local clock should be compared with the time-stamp of the next earliest event and the current processing should be preempted if the next event arrives before completion. If the next event time is not known a priori, such time-synchronization requirement charges huge overhead of exchanging time information between simulators during the event processing.

In the proposed scheme, we can do much better if we consider the underlying computation model of the simulated tasks. Dataflow tasks can be regarded as non-preemptive process if we use a clever time adjustment scheme as explained in the next section.

4. EVENT-DRIVEN SCHEDULING WITH DATA-DRIVEN MODELS

As explained in section 2, the simulated task in a component simulator is a SDF subgraph. Recall that a SDF block becomes active after all input ports have the required number of data samples. By utilizing the data-driven execution model, we could reduce both the data communication overhead and the time synchronization overhead significantly. Although we restrict our discussion to the SDF model, our approach is also applicable to more general cases when the simulated tasks are data-driven and have predictable access patterns to shared resources [5].

4.1 Virtual Synchronization

Time synchronization problem explained in the previous section has been the main performance bottleneck for the distributed cosimulation. The difficulty lies in the fact that the local clock of a component simulator should be synchronized with the global clock of the backplane in order not to violate the causality condition. We observe that such synchronization requirement is no more needed if the simulated tasks are data-driven. In the proposed scheme, we do not synchronize the local clock with the global clock, but still preserve the correctness. We will explain the key idea with a simple example of Figure 3.

Figure 3(a) shows a simple synchronous dataflow subgraph that a component simulator should simulate, where a, b, and d denote the data samples and a(k) indicates the k-th data sample. The execution time of each block is also displayed. For synchronous dataflow graphs, the local schedule of block executions is pre-determined at compile-time. Figure 3(b) and (c) show Gantt chart representations of two local schedules, one for a software component where blocks are sequentially executed, and the other for a hardware component where blocks can be executed concurrently.

Suppose that data samples from the backplane arrive following the scenario as displayed in Figure 3(d). After a(1) and b(1) are both available, the component simulator simulates the task graph as shown in Figure 3(e) and computes the time duration between d(1) and a(1), $\Delta(1)$. When the component simulator returns the result d(1) to the backplane, the time stamp of d(1) is adjusted to the sum of the time stamp of a(1) plus $\Delta(1)$ as shown in Figure 3(f). Similarly the component simulator simulates the task graph again with the next input data a(2) and b(2) and computes $\Delta(2)$. The time stamp of d(2) is adjusted as displayed in the figure.

Suppose that the time stamp of a(2) is changed to 10 instead of 16. Then, the component simulator detects the next execution of the task graph should be delayed by 3 units after the previous execution completes. The time duration between d(2) and a(2), in this case, should include such delay.

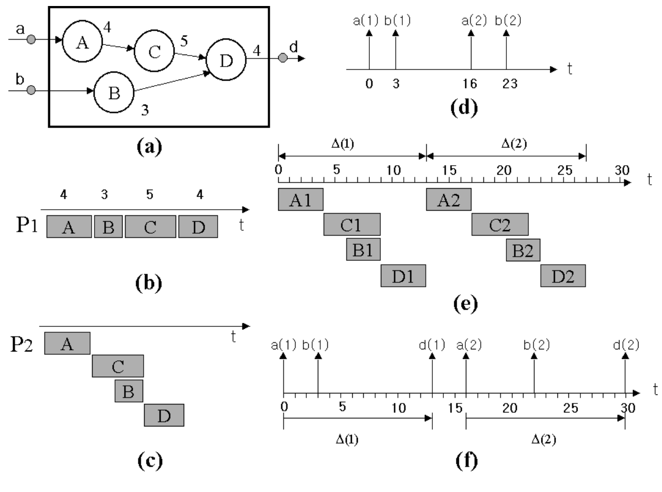

Figure 3. Virtual synchronization of the synchronous dataflow sub-graph: (a) example graph, (b) SW schedule, (c) HW schedule, (d) assumed input arrival pattern, (e) simulation behavior with virtual synchronization, (f) observed behavior from the backplane.

In summary, a component simulator does not perform any synchronization with the backplane during execution. It returns a time difference between the start time and the end time of the task. Then, we reconstruct the accurate simulated time of the output data in the backplane. We call this scheme as *virtual synchronization* because the local clock is apparently synchronized with the global clock while not actually.

Virtual synchronization removes the need of time synchronization overhead during the execution of the current schedule. Synchronization between the backplane and the component simulator is accomplished at the time of sample exchange. As a result, we reduce the time-synchronization overhead near to zero except adjusting overhead of the time stamp.

Since the virtual synchronization preserves the processing order of data samples and the data-driven execution model is dependent only on the arrival order of data samples, data values will not be corrupted. In Figure 3(e), sample *a(2)* does not affect the value of sample *d(1)*. Therefore, we arrive at the following theorem.

Theorem 1.

If the simulated task is a synchronous dataflow graph, virtual synchronization preserves the correctness of cosimulation.

Cockx [5] proposed a similar approach for compiled cosimulation which allows out-of-order executions of SW modules. Correct execution order is recovered from later time adjustments. The proposed technique is similar to those in that we also use time adjustment technique for timing correctness. But, there are three significant differences. First, our technique is for the system-level distributed cosimulation, which is more complicated. Second, we can utilize parallelism between simulators. Third, we do not synchronize local clocks of component simulators while they do.

4.2 Reduction of Active Simulation Duration

Virtual synchronization not only removes the synchronization overhead but also reduces the active duration of component simulators. A simulator does not need to increase the local clock until it receives a new input data after processing the last data samples as shown in Figure 3(e). The absolute value of local clock is no more important. Instead time difference between output production and input arrival matters for timing accuracy. Therefore, we may shorten the simulation time significantly.

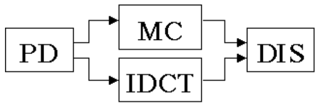

Figure 4. A simple video decoder example

Consider a simple video decoder example of Figure 4, where a packet decoder (PD) block and a display block (DIS) are executed on a SW simulator. And we assume that an inverse DCT (IDCT) and a motion compensation (MC) blocks are run on a HW simulator.

Figure 5(a) shows the simulated execution time. HW blocks (IDCT, MC) are executed simultaneously and run faster than SW blocks. Figure 5(b) illustrates a typical implementation where two HW blocks are simulated in a same HW simulator because multiple hardware simulators need more inter-process communication with huge overhead. The simulation time of HW simulator can be divided into two sections. The first gray section indicates the period during which the HW simulator just increases its local clock to be synchronized with the global clock. The second section indicates the active duration processing two HW blocks.

The proposed technique changes the execution profile as shown in Figure 5(c). First the idle times are removed and sample times are adjusted by the backplane. Second multiple HW simulators can be invoked to process separate blocks. Since no time synchronization is necessary, benefits from distributed simulation is bigger than the increased IPC overheads

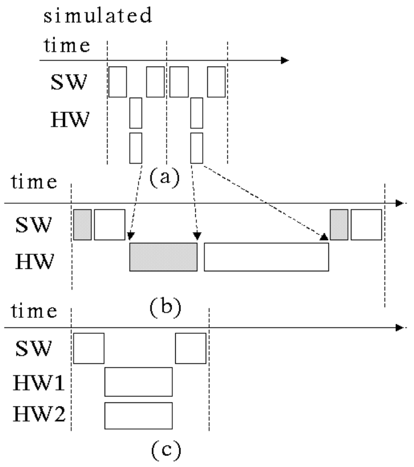

Figure 5. (a) Simulated execution time (b) Conventional cosimulation profile (c) Cosimulation profile from the proposed technique

4.3 Message Grouping

Using the backplane as a central simulator doubles the amount of data communications among simulators. Even though the total number of data samples becomes double, we can reduce the communication overhead by grouping the samples into a large packet. This grouping optimization is possible if it does not hurt the local causality condition of the task. *Message grouping* in the proposed scheme comes from the data-driven property. Because a dataflow block becomes active only when all input ports have data, the backplane does not need to send a separate packet to each input port.

5. PRELIMINARY EXPERIMENTS

Performance improvements from the proposed technique result from several factors. First, virtual synchronization remove the time synchronization overhead and reduce the active duration of simulators. Second, it enhances parallelism between component simulators. Last, message grouping reduces the data communication overhead. In this section, we present the experimental results on the first two factors separately. At last, we compare the proposed approach with the optimized operation of Seamless CVE co-verification environment.

5.1 Reduced Time Synchronization Overhead

To isolate the performance gain due to the reduced time synchronization overhead, we make a simple example which consists of a source, a HW simulator, and a display. We compare two centralized approaches and the proposed approach: "normal" indicates a conservative approach and "optimized" indicates an optimized technique proposed in [15].

The first set of experiments assumes that the source block generates an event every 200 ns and HW takes 140 ns. And the clock period of the HW simulator is set to 20 ns. Figure 6 shows the experimental results varying the total simulated time. In the "normal" case, there need nine time-synchronization activities for each invocation of the simulator. Each synchronization overhead is measured 66 ms on the average. Since the "optimized" one already knows when the next event arrives, it needs not any time synchronization activity in this experiment. But, compared with the proposed approach, it has another cause of performance penalty. That is, it needs to advance the local clock during the idle period to be synchronized with the global clock while the virtual synchronization does not need to synchronize two clocks but adjusts the time stamp values after executions.

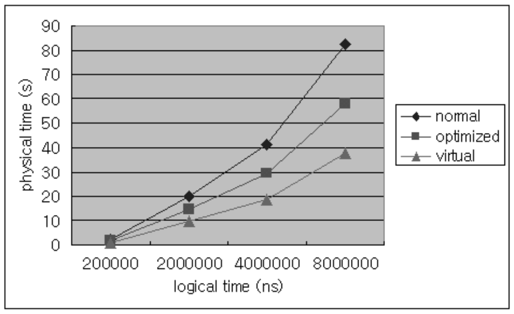

Figure 6. Comparison between conservative approaches and proposed virtual synchronization technique

Now, we increase the source period to 10000 ns, the performance gain by the virtual synchronization becomes drastic as shown in Figure 7. The performance gain from the proposed scheme becomes 45 times in Figure 7 compared with the "optimized" one mainly because of the removal of time synchronization overhead.

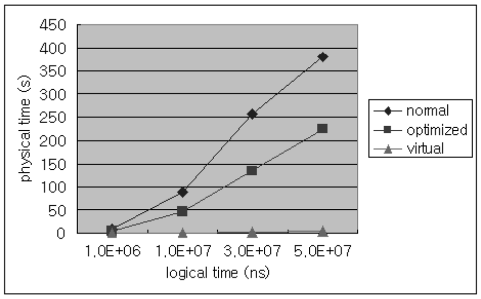

Figure 7. Another comparison between conservative approaches and the proposed technique

5.2 Enhanced Parallel Execution

In this subsection, we examine the performance improvement due to the distributed execution of simulators. We use six different experiment sets with diverse experiment conditions by varying graph topologies and machine specifications.

Figure 9 shows three different graph topologies. And three machines are used in the experiments. Machine 1 is Pentium III 600Mhz using dual CPUs and machine 2 is Ultra Sparc II 450Mhz. Both are connected to an 100M fast Ethernet switch while Machine 3 is Pentium III 350Mhz and connected through a 10M Ethernet hub. Six different experiment sets are listed in Table 1. The first three sets have a single task graph and the next three sets have two disconnected task graphs. Tasks mapped to the SW component are simulated sequentially in a SW simulator while we use a separate HW simulator process for each task mapped to the HW component.

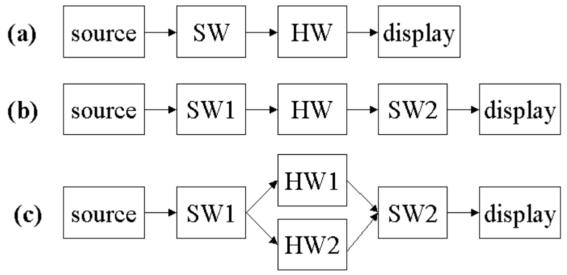

Figure 8. Experiment graphs

Table 1. Experiment sets

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
Graph	(a)	(b)	(c)	(a),(b)	(a),(c)	(b),(c)
Machine	1	1	1,2	1,2	1,2,3	1,2,3

For each set of experiment, we perform five different experiments by intentionally changing the execution time ratio between a SW task and a HW task as 1(SW):1(HW), 1:2, 2:1, 1:3 and 3:1. Figure 10 illustrates the summary of performance improvement by the distributed execution of simulators compared with the serialized execution. Both adopt the virtual synchronization. The more simulators are involved in the cosimulation with similar execution times, the more performance gain is achieved.

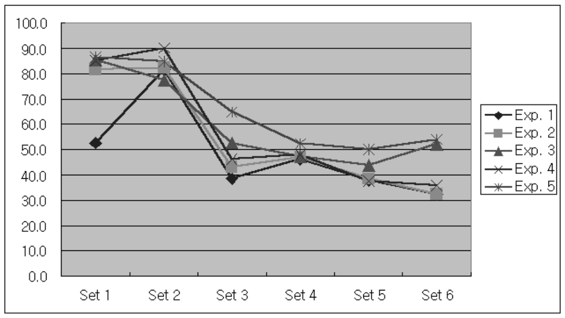

Figure 10. Cosimulation applied with parallelism/ normal serial cosimulation *100 (%)

5.3 Comparison with Seamless CVE

We compare the proposed approach with Seamless CVE co-verification environment [16] using an H.263 decoder example which is composed of a HW IDCT block and remaining SW blocks. The approach uses Armulator for a processor simulator and ModelSim for a HW simulator as Seamless CVE does on the Ultrasparc II 450Mhz machine. In the Seamless CVE, we apply the instruction fetch optimization and the data access optimization [16] to enhance the simulation speed while maintaining the cycle accuracy.

The experiment result shows a significant performance enhancement. On Seamless CVE, the simulation takes 2031.71s to decode one frame of QCIF format. But the proposed approach ends the simulation at 303s and is 6.7 times faster. Such performance gain comes from reducing synchronization overhead and the active duration of simulators by the virtual synchronization. In our approach, HW simulation time is major bottleneck 78%. And IPC overhead and SW simulation times are 18% and 4% respectively.

6. CONCLUSION

As the embedded systems are complex and the design turn-around time is shortened, the performance of co-simulation becomes more important to verify such a system. And the component simulators used in the cosimulation are likely to be heterogeneous or geographically distributed. So the fast distributed cosimulation is essential for the embedded system design.

Our proposed backplane approach uses the "deadlock detection and recovery" approach to enhance the parallel execution of component simulators. The key contribution of the proposed approach is to combine data-driven and event-driven simulation technique for fast distributed cosimulation. Data-driven properties of the specification model reduce the number of communication packet exchanges by message grouping and minimize time-synchronization overhead by virtual synchronization. Virtual synchronization also reduces the active duration of component simulators significantly. The preliminary experiments give promising results on the performance improvement specially when there are more component simulators involved.

7. ACKNOWLEDGMENTS

This work was supported by National Research Laboratory Program (number M1-0104-00-0015) and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.

8. REFERENCES

[1] Cassandras, C., "Discrete event systems, modeling and performance analysis", Irwin, Homewood IL, 1993.

[2] Jefferson, D.R., "Virtual time", ACM Trans. Prog. Lang. and Syst. 7, Jul. 1985, pp404-425.

[3] Chandy, K.M., and Misra, J. "Asynchronous distributed simulation via sequence of parallel computations", Commun. ACM 24, Nov. 1981, pp. 198-205.

[4] Desmet et al. "Timed executable system specification of an ADSL modem using a C++ based design environment: A case study", CODES '99, 1999, Page(s): 38 –42

[5] Cockx, J., "Efficient modeling of preemption in a virtual prototype", RSP 2000, 2000, Page(s): 14 –19

[6] Meerwein, M.; Baumgartner, C.; Wieja, T.; Glauert, W., "Embedded systems verification with FGPA-enhanced in-circuit emulator", System Synthesis, 2000. Proceedings. The 13th International Symposium on , 2000, Page(s): 143 -148

[7] Kim, D. and Ha. S., "System level specification for multimedia applications", the 6th Conference of Asia Pacific CHip Design Languages, Fukuoka, Japan, Oct. 6-8, 1999

[8] Sung, W. and Ha, S., "Efficient and Flexible Cosimulation Environment for DSP Applications" , IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Special Issue on VLSI Design and CAD algorithms, Vol.E81-A, No. 12, pp. 2605-2611, December, 1998

[9] Yoo, S. and Choi, K., "Optimistic Distributed Timed Cosimulation Based on Thread Simulation Model", Proc. of Proc. 6th Int'l Workshop on Hardware/Software Co-Design, Mar. 1998.

[10] Nicol, D. and Heidelberger, Ph, "Parallel Execution for Serial Simulators", ACM Transactions on Modeling and Computer Simulation, vol. 6, no. 3, July 1996, pp. 210-242.

[11] Atef, D., Salem, A. and Baraka, H., "An Architecture of Distributed Co-Simulation", 42nd Midwest Symposium on Circuits and Systems 2000, Volume: 2 , 2000, Page(s): 855 - 858 vol. 2

[12] Klein, R., "Miami: A Hardware Software Co-Simulation Environment", In proceedings. 7th IEEE International Workshop on Rapid System Prototyping, June 1996.

[13] Hines, K. and Borriello, G., "Dynamic Communication Models in Embedded System Co-Simulation" In Proceedings of the 34 th Design Automation Conference, June 1997

[14] Fujimoto, R. M., "Parallel Discrete Event Simulation", Communication of the ACM, Oct. 1990, Vol. 33, No. 10, pp. 30-53

[15] Sung, W. and Ha, S., "Optimized Timed Hardware Software Cosimulation without Rollback", 1998 Design Automation and Test in Europe, Paris, France, Feb. 1998.

[16] Mentor Graphics Seamless CVE Home Page (http://www.mentorg.com/seamless/)