A Backplane Approach for Cosimulation in High-Level System Specification Environments

S. Schmerler Y. Tanurhan K. D. Müller-Glaser

Forschungszentrum Informatik Karlsruhe
Dept. of Electronic Systems and Microsystems
FZI/ESM
Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany
esm@fzi.de

Abstract

The design of microelectronic systems including hardware and software for open-loop and closed-loop control requires the combination of expertise from different domains. However, no integrated approach to the specification and design nor to the analysis and simulation of the overall system is available. The design of such complex and heterogeneous systems mandates a systematic, computer aided approach to requirements definition, specification and design as well as to the verification and validation of the results. In this paper a simulation backplane concept is presented for parallel mixed-mode cosimulation using a standardized interface. Synchronization mechanisms are developed and used that allow an efficient concurrent cosimulation.

1 Introduction

During the past few years the design of electronic systems with analog and digital components has made it necessary to use different tools within one simulation environment and to couple them such as in a simulation backplane. An efficient coupling between modern simulators often can only be achieved in a distributed simulation. Cosimulation has become a very important issue in mixed digital/analog circuits and system design because many tools are specialized either to discrete-event or continuous simulation. In this paper we present a simulation backplane for coupling continuous and discrete-event simulators in an environment that is able to run the cosimulation in a very efficient mode. Special algorithms must be used to ensure a parallel mixed-mode cosimulation. This work is part of our integrated design environment for real-time systems IRTISD for heterogeneous systems consisting of analog and digital hardware and software as well [1].

There are several possibilities to simulate systems containing analog and digital components. First of all, there are mixed-mode simulators such as Saber™ from Analogy Inc. using algorithms that allow a coupling of digital and analog models by introducing interface models performing the analog/digital conversion. This solution requires that all simulators, as well as the design data format, come from the same tool manufacturer, or at least are adapted to each other. In other words, it is in general not possible to use a simulator of one's choice in such an environment if there is no interface provided by the tool manufacturer.

Other coupling technologies are so called "ad-hoc couplings", which can be thought of as a very specific solution for two or more simulators. A coupling in this sense may be obtained by a code integration process for the source code created by each tool. Coupling another simulator to this environment would require the same work to be done once again, i.e. there is no advantage in the knowledge of the integration process for other simulators. Building such a cosimulation environment also requires the knowledge of detailed information about the single simulator or source code representing the simulator. Coupling tools in this way is hard work. Moreover, this is just a specific solution and the next simulator to couple requires the same effort.

Commercial simulation backplanes offer a much more flexible way in coupling technology. Using a procedural interface, a high degree of modularity and independency in tool integration is achieved. Tool manufacturers provide their simulators with a standardized interface, and thus enable each customer to use a coupling with other simulators of their choice. Besides, already existent integrations are not affected by this new interface. The adaptation of simulation data of each simulator to the new simulation environment is performed automatically.

2 Common Backplane Concept

The simulation backplane concept used by most available backplanes is in a standardization process of the CAD Framework Initiative (CFI) which has not yet been finished [2]. In the following, this concept will be called the "Common Backplane Concept". A simulator backplane can be thought of as a superior instance that controls the processing of all client simulators coupled to it. Thus, cosimulation is obtained. The whole simulation progress is controlled by the backplane - all a simulator can do is tell the backplane when it needs input from partner-simulators or other dependencies. Based on this information, the backplane calculates the order and time period of simulator runs in a way that all clients can satisfy their data requests registered by the backplane.Within this proceeding algorithm, the backplane does not only have to prevent the simulating clients from working further than a request to a particular solver but it must also ensure that it does not run to a time point later than *another* solver is able to make a request. This is a bidirectional requirement.

Fig. 1 shows a typical simulation backplane environment regarded in this scope. Client simulators are connected to the backplane either directly over a tool specific or over a C-code interface. Thus, a backplane environment is a convenient solution for the multi-simulator problem. Various independent simulators, each optimized for a special task, are combined in one common environment. The open architecture of a backplane allows to integrate special simulators such as DSP modelling tools, self-written simulators, and even not yet existing tools to the backplane if they are provided with a specific interface. For this reason, the integration of the best tool in each case is possible and it is no more necessary to use a specific tool just because it already is coupled to another simulator in a special environment. Thus, existing model libraries, which often represent an immense value, can still be used. There are currently two strands of work: First, commercial backplanes are used to simulate models containing reactive, time-discrete behaviour sub-models and the models of closed-loop control systems. Here, two different simulators are connected to the backplane. There is one VHDL-simulator which is responsible for the reactive sub-model and the other simulator is a differential equation solver written in C-Code for the closed-loop control system. Unfortunately, all tested backplanes offered a bad performance which is a very important drawback in simulation. In the evaluation phase we tested the performance of different commercial simulation backplanes. Several performance marks have been compared by running the same simulation model both on a backplane environment and on a single-core simulator. As shown in Fig. 3 the ratio of computation time backplane system to single-core system was up to 40:1. To ensure a representative comparison, four different simulation courses (S_1 to S_4) were chosen The reason for this lack of performance can be caused by additional communication levels or inefficient synchronization algorithms.

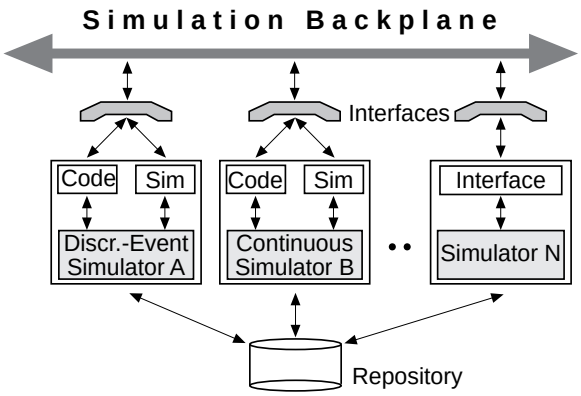

Fig. 1: Backplane concept of cosimulation

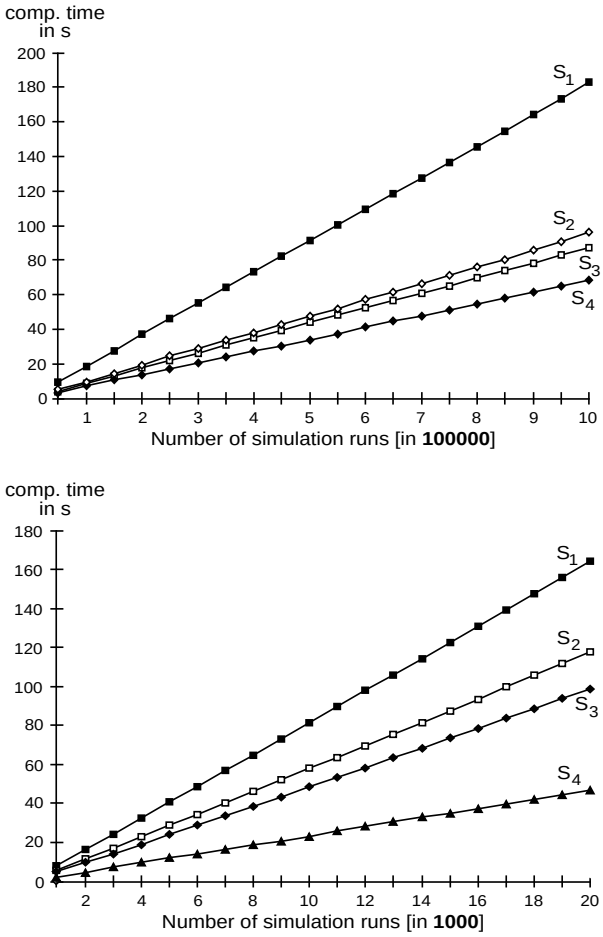

Fig. 2: Performance ratio backplane system to single-core system

3 An Optimistic Backplane Approach

Our main goal was to develop a superior simulator processing strategy with high efficiency and a minimum of communication between backplane and clients. Very important in this context is the fact that simulators working in the continuous time space are connected. One major problem in coupling time-discrete and continuous simulators is data exchange - especially requests from a continuous to a time-discrete working solver. A typical digital simulator only knows the results of its calculations at the current time but not in the past and not at all at time points in the past that do not fit on the time-scheme. Several enhancements have been made to additionally improve the performance during cosimulation.

3.1 Synchronization Algorithm

The most important influence on the efficiency of a cosimulation environment is the basic synchronization algorithm used. We consider the used synchronization algorithm to have a favourable effect on the performance of the coupled system in mixed-mode cosimulation. Most simulation backplanes use a PDES-(Parallel Discrete-Event Simulation) algorithm based on Lock-Step which results in a sequential processing or a locked technique. Only one simulator may run at a time, exchanging results afterwards. This causes a lack of performance which may be acceptable when discrete-event systems are coupled. Unfortunately, when regarding continuous simulators such as differential equation solvers it means a drawback.

In our approach, we use a partially optimistic synchronization mechanism first proposed by Jefferson [3], [4]. Jefferson introduces the term *local virtual time* which is the simulation time point to which a subsystem has carried out its simulation. The virtual time paradigm is a method of synchronizing and organizing distributed systems. Jeffersons implementation of this paradigm, the *Time Warp* mechanism, is a very important protocol for parallel simulation. In most cases, it is used to organize discrete-event systems. To correctly simulate a subsystem (design partition), a process must execute messages in the timestamp order. This may not be the order of their real arrival time. Because subsystems act independently and asynchronously, incoming event messages are not guaranteed to arrive at a receiving process in timestamp order. Nevertheless, the logic of simulation demands, for the maintenance of causality, that all event messages must be processed in timestamp order for each process. The management of virtual time is handled in an asynchronous manner. This means that virtual times and the simulation progress may differ in a wide range. If a process local virtual time is t, then the process has (at least temporarily) simulated to this time point. Event messages with timestamps $t^* < t$ arriving at this process have been processed but none of a higher timestamp. Of course, there is no guarantee that all messages destined to arrive with timestamps less than t have already arrived. This would mean a *causality error*: if the execution of an event A causes or affects the execution of an event B, then the execution of A and B must be scheduled in real time so that A is completed before B starts. A Time Warp system characteristically processes continuously event messages from its input queue in timestamp order until there are no more events in the queue. It never waits until it can "safely" process the next message. Instead, it always charges ahead and the only reason for stopping is an empty message queue. It continues working when the next message arrives. This "charge ahead"-policy is optimistic, for it takes a calculated risk that no straggler message (a message with a timestamp less than or equal to the local virtual time of the receiving process) will arrive. In the case of a straggler, the state of the receiving process must be reset to that belonging to the timestamp of the straggler event. This procedure is called *rollback*. In addition, all side effects caused by the use of computation results for a time point greater than the timestamp of the straggler must be eliminated. This means that a rollback also causes rollbacks of other processes. A process that has been rolled back may reprocess some messages that it processed before, but this time the straggler will be processed first in its correct sequential position. In the case of a rollback some computation time is "wasted" when a projected future is thrown away. A conservative[1] mechanism [5] in the same circumstance would keep the process blocked for the same amount of time so that time would be wasted "anyway". In difference to a conservative algorithm, there is a higher degree of parallelism in cosimulation. In the mixed-mode cosimulation, an optimistic synchronization algorithm can be advantageously applied. For example, requests from continuous clients directed to discrete-event simulators with a lower local virtual time can now be answered by optimistically assuming no changes from the last calculated time point to the time in request. Simulation can go on only if the slower client calculates another result for the specified time point. A straggler message must be sent and rollbacks must be performed. Alternatively, the asking client would have waited for the slower partner to complete its simulation. The whole coupled system would have been as slow as the slowest client simulator. A state restoration is performed by sending so called *anti-messages* cancelling all messages at or after the straggler event message.

1. Conservative systems execute events only when they can guarantee that doing so does not violate the causality constraint.

3.2 Extensions to Time Warp

The Time Warp-based mechanism has been successfully used for simulations of digital hardware with a promising speed-up [6]. We are working on a modification of this synchronization algorithm to synchronize cosimulation of mixed digital and analog hardware.

In the last few years, many extensions to Time Warp have been made to improve performance. In *lazy cancellation* [7], a rolling back process does not immediately send the correspondent anti-message but waits to see if the reexecution of the computation causes any of the *same* messages to be regenerated. If the result is the creation of the same message, it may not be cancelled. The other approach is an *aggressive* cancellation mechanism. Once a computation is executed out of timestamp, *all* of the computation as well as other computations that may have been affected by the execution of this causal wrong message are incorrect and cancelled. However, it is possible that the erroneous computation in spite of the error generated correct event messages, which should not be cancelled. This drawback is partially avoided in lazy cancellation. [8] and [9] show that lazy cancellation tends to perform as well as, or better than, aggressive cancellation in practice.

In [10] and [11] Madisetti, Walrand, and Messerschmitt propose the so called *WOLF*-mechanism. In WOLF, a straggler message causes a process to send special control messages to stop the spread of erroneous computations. Processes that may be affected by the execution of the non-causal message are notified when a straggler message is detected. Optimizing work on the area of state saving (necessary for rolling back in Time Warp) has been done in [12] and [13] in form of *periodic checkpointing* and *incremental state saving*. The intention was to find the optimal checkpoint interval, i.e. a minimum of overhead and memory required to perform state saving.

3.3 Coupling Mechanisms

Coupling discrete-event and continuous systems means combining two completely different timing mechanisms. Requests among discrete-event or continuous clients can easily be answered because of the same timing scheme: discrete-event clients need to have the same time base (i.e. system state can change only at the same time points), whereas continuous solvers a priori do not make demands to the timing scheme. Difficulties can be seen in requests from continuous to discrete-event clients when the time point in question does not fit the time scheme of the discrete-event client. This problem can easily be solved within the simulation backplane by interpolation and the assumption that there is no change between any points on the timing scheme, for example in Fig. 3, the

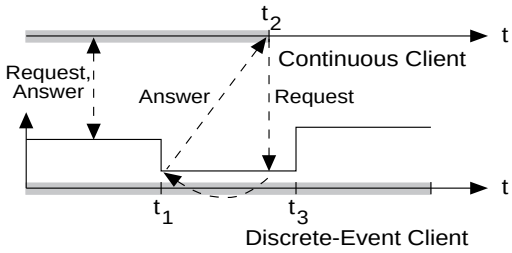

Fig. 3: Interpolation scheme

request of the continuous client at t_2 will be answered with the computation results at the last simulated time step (t_1) of the discrete partner. Requests as shown in Fig. 3 can be answered immediately, whereas a timing according Fig. 4 in a conservative system would cause the asking client to wait until the asked partner has worked to a time point (t_2), when it can provide the data in request by itself, which means for the time (t_2-t_1). In our optimistic approach, we assume that there is no change since the last calculated value (at t_1) and thus go on simulating without interrupt.

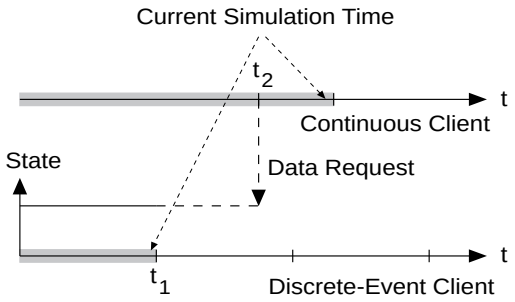

Fig. 4: Optimistic approach

Requests from time-discrete to continuous clients are treated in a similar way. Interpolation is used to answer requests that do not fit on the calculation plan of the continuous client (according to Fig. 3). Optimistic approaches for pure continuous systems are very hard to realize, but the developed extrapolation algorithms make it possible to perform a look-ahead according to Fig. 4. It can be shown that the resulting error is not larger than any internal calculation error during the solution of differential equations. Of course, there is a limitation of this extrapolation interval (simulation time) by reason of the exactitude of the estimated solution.

3.4 Procedural Interface

Communication for synchronization and data transfer is done by a Procedural Interface (PI) containing a set of callbacks. A minimal set of these callbacks can be found which represent the minimal PI required by mixed-mode cosimulation over a simulation backplane. This PI is the

only interface to the backplane. Therefore, it should become standardized in a way that tool manufacturers are able to provide their simulators with this functionality. CFI [2] mainly concentrates on the coupling of discrete-event systems but a lot of specific communication that typically must be used in a mixed-mode cosimulation has not yet been defined. In this approach we propose an interface which eliminates this deficit.

Another aspect to mention when integrating simulators to a backplane is the amount of work the integrator has to do, and to which degree a reuse of already done work is possible. To ensure an easier way of integrating clients to a backplane, we formulated a distinct set of control commands a solver must dispose of before being connected (PI). These commands also include coupling of analog/continuous simulators. Thus, it is easier for tool manufacturers to provide a set of callbacks without publishing their know-how. When a tool provides this set of callbacks it can be very easily - and always in the same manner - be connected to a backplane by writing the interface code. Moreover, another advantage of such a standardized interface is the similarity of the integration code that is necessary to complete the integration. Coupling yet another client to the backplane does not cause a change of the integration for the other tools and can be done in a relatively short time.
The advantage of a standardized PI is the fact that in the future manufacturers will be able to take care of this additional functionality in the development phase of their tools. Unlike in the case of other coupling techniques, there is no danger of giving away the internal know-how or algorithms of the client simulators. Many direct tool couplings in the past have failed or resulted in a bad performance because manufacturers did not open interfaces that should have been used for an efficient cosimulation for they feared to give away know-how. An interface, as mentioned in this paper, is a possibility to avoid this conflict.

3.5 Design Partitioning

To a high degree, the performance of a coupled simulation system is influenced by the result of the design partitioning performed in the elaboration phase. Characteristics for a good partitioning include a minimum of communication over partition boundaries, i.e. over the backplane. Minimizing inter-simulator connections is extremely important. Each connection (so called boundary pin) is a potential carrier of events, and some may exhibit very high levels of activity and thus may cause an intensive communication. The most common application of PDES to analysis of large designs is to intelligently partition the design into sub-sections, with the dual goal of minimizing event traffic among the partitions and relying on the algorithms implemented in the backplane for run control and synchronization. Clearly, an efficient cosimulation requires that the majority of the time is spent inside the simulator clients rather than in a synchronizing module. On the other hand, design partitioning for parallel simulation can be approached automatically. During a special phase in elaboration, the design data must be processed and based on a set of rules or heuristics that the whole design is partitioned around. The goal is to minimize cross-simulator communications.

There are two important techniques for automatic partitioning: optimization and path analysis. Optimization scheme means a form of combinatorial optimization as it is used for instance for the automatic placement of standard cells. Another approach is based on the analysis of circuit structure, i.e. it provides a geometrical solution of the problem. Combining MINCUT and Simulated Annealing algorithms it is possible to achieve several sub-optimal but acceptable partitionings in a relatively short time. The algorithm manipulates functional blocks arbitrarily placing them in different partitions. The best solution, i.e. the partitioning with fewest wires crossing the partition boundaries, is found by combinatorial optimization. A drawback of this algorithm is that it does not take into account the activity-level of partition crossings. A further improvement would be possible if run-time statistics were collected and weighted crossings lead to a better partitioning via back-annotation. We now have an algorithm working on a combination of dynamic (run time statistics) and static (structural) data.

The most promising technique, however, we see in path analysis-based approaches. Algorithms of the graph theory are used to find the optimal partitioning. A graph representing the design is traversed from the output pins to the inner design. Sub-graphs are formed which describe circuitry that has effects on the value of each output signal. Those graphs that do not intersect with others form a partition. Intersection areas may be replicated in multiple partitions (resulting in larger independent partitions) or cut using optimization technique described above.

Another aim is to work out algorithms for automatic partitioning to achieve a well-balanced load concept for coupled client simulators. These simulators will minimize causality errors, resulting in a better performance. Rolling back as a consequence of the occurrence of a causality error must also be limited to a minimum of neighbour partitions for evident reasons. This also results in constraints for good partitioning algorithms based on graph theory. In Fig. 5 modifications of the boundaries between partition *B* and *E* and between *A* and *D* stop spreading a rollback initiated by partition *A*.

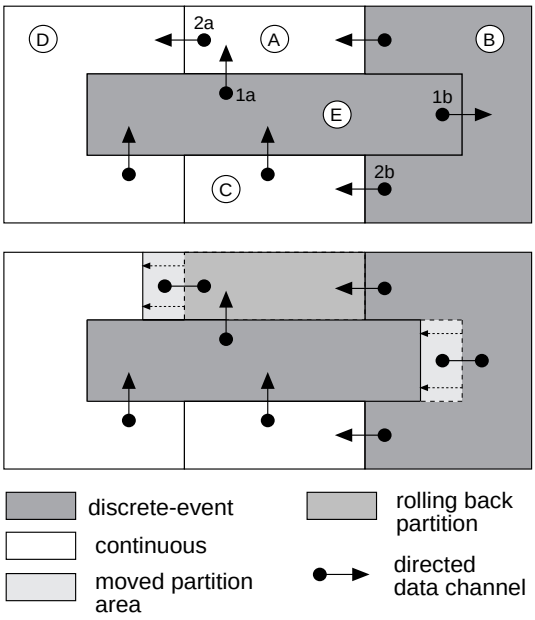

Fig. 5: Partitioning and rollback behaviour

4 Conclusions

A backplane concept is an appropriate solution for the cosimulation problem in heterogeneous design environments. Existing simulation backplanes are very powerful tools as far as the standardized interface and data exchange mechanisms are concerned. On the area of partially optimistic working synchronization algorithms we see a large potential for improving the performance of the coupled mixed-mode system simulation. Optimistic approaches have yet to find their way into commercial products. On the other side, the Lock-Step-based distributed simulation has been used and evaluated on the commercial software market several times. These attempts resulted in improvements of simulation capacity, but delivered relatively little (if at all) concerning the performance in the simulation of coupled systems.

5 Future Work

Future work will be concentrated on partitioning algorithms and techniques for microelectronic systems both in discrete and continuous areas with constraints mentioned above. Tests of the developed synchronization and partitioning algorithms in a multiprocessor environment must be performed to achieve performance data. Furthermore, the efficiency of optimistic synchronization for mixed discrete-event and continuous systems must be proven in concrete applications.

References

[1] Y. Tanurhan, S. Schmerler, and K. D. Müller-Glaser, "System Level Specification and Simulation for Microsystem Design in the METEOR-Project", *Proceedings of Microsystems Technologies*, Berlin, Germany, 1994.

[2] CAD Framework Initiative, Inc., Simulation Backplane Programming Interface Specification, *Working Group Draft Proposal, Version 0.6.0*, Austin/Texas, 1993.

[3] D. R. Jefferson, and H. Sowizral, "Fast co current simulation using Time Warp mechanism", part 1: *Local control, Technical Report N-1906-AF, RAND Corporation*, 1982.

[4] D. R. Jefferson. "Virtual Time", *ACM Transactions on Programming Languages and Systems*, 7(3):404-425, 1985.

[5] K. Chandy, and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations", *Commun. ACM 24*, 440-452, 1981.

[6] D. Ball, and S. Hoyt, "The adaptive Time Warp concurrency control algorithm", *Proc. of the SCS Multiconference on Distributed simulation*, 22(1):174-177, 1990.

[7] A. Gafni, "Rollback mechanisms for optimistic distributed simulation systems", *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):61-6, 1988.

[8] D. Baezner, J. Cleary, G. Lomov, and B.Unger, "Algorithmic optimizations of simulations on Time Warp", *Proc. of the SCS Multiconference on Distributed Simulation*, 21(2):73-78, 1989.

[9] S. Bellenot, "Global virtual time algorithms", *Proc. of the SCS Multiconference on Distributed Simulation*, 22(1):122-127, 1990.

[10] V. Madisetti, J. Walrand, and D. Messerschmitt, "WOLF: A rollback algorithm for optimistic distributed simulation systems", *1988 Winter Simulation Conference Proceedings*, pp. 296-305, 1988.

[11] V. Madisetti, J. Walrand, and D. Messerschmitt, "Synchronization in message-passing computers-models, algorithms and analysis", *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1): 25-48, 1990.

[12] Y.-B. Lin, and E. D. Lazowska, "The optimal checkpoint interval in time warp parallel simulation", *Technical Report 89-09-04, Depqrtment of Computer Science and Engineering, University of Washington, Seattle, Washington*, 1989.

[13] Y.-B. Lin , B. R. Preiss, W. M. Loucks, and E. D. Lazowska, "Selecting the optimal checkpoint interval in time warp parallel simulation", *7th Workshop on Parallel and Distributed Simulation. Society for Computer Simulation.*, 1993.