Optimistic Synchronization of Mixed-Mode Simulators[*]

Peter Frey, *Radharamanan Radhakrishnan*,
Harold W. Carter, and *Philip A. Wilsey*
Dept. of ECECS, PO Box 210030, Cincinnati, OH 45221–0030

Abstract

Mixed-Mode simulation has been generating considerable interest in the simulation community and has continued to grow as an active research area. Traditional mixed-mode simulation involves the merging of digital and analog simulators in various ways. However, efficient methods for the synchronization between the two time domains remains elusive. This is due to the fact that the analog simulator uses dynamic time step control whereas the digital simulator uses the event driven paradigm. This paper proposes two new synchronization methods which are general enough to support several algorithms within the same simulation. The capabilities of the synchronization protocols are presented using a component-based continuous-time simulator integrated with an optimistic parallel discrete event simulator. The results of the preliminary performance evaluation leads us to believe that while both synchronization methods are functionally viable, one has superior performance.

1 Introduction

Mixed-mode simulation is the simulation of a combination of two distinct mathematical models. Zeigler [15] classifies these two mathematical models as the class of discrete event models and the class of differential equation models. Both models are characterized by their behavior. Differential equation models exhibit continuous changes in time and state; thus, the time derivatives (*i.e.*, rates of change) are governed by differential equations. Cellier [4] reports that differential equation models, in general, are simulated as a discrete-time model on a digital computer to avoid infinitely many state changes. Values in between the discrete-time stamps are then interpolated. On the other hand, discrete event models exhibit discrete changes in time and state. State changes only occur at individual time points (events) and are always discontinuous. No changes in state occur in between these time points.

Due to the difference in the mathematical models, mixed-mode simulation results in two different simulation paradigms. There are many simulation algorithms, each with its own properties, that can simulate either differential equation models [2, 10] or discrete event models [5, 7]. However, these algorithms cannot be used for mixed-mode simulation as the two paradigms have to be combined. Figure 1 shows an example of a mixed-mode simulation model and its results. The model contains two discrete-event processes with signals (A,B) and one process which solves differential equations, with an internal signal. The model description in Figure 1a includes interface functions (IF) between the different domains. The IF maps discrete signals to continuous signals and vice versa. This is illustrated by the vertical lines in Figure 1b. Although the IFs are critical for mixed-mode simulation (as two different signal types are being connected), they introduce new problems during design and simulation [12]. For example, how is a digital signal described at the behavioral level connected to the same signal in the analog domain (described at the electrical level)? Hence, the IFs are not only crucial for correct modeling, but also act as a bridge between the two different simulation paradigms.

It is obvious that for correct simulation of a combination of these models interaction via the interface functions will have to be supported. This implies that the simulation is divided into distinct simulation time[1] intervals in which no interaction between the two simulation models takes place, leaving distinct points where the interaction (or communication) does takes place. To model this interaction, the simulation model is defined as interacting processes. Discrete-event processes define the behavior of the discrete-event model whereas differential equation processes define the behavior of the differential equation model. The interface functions are entrusted with the task of the communication between these processes; the different notions of time remain the responsibility of the simulator.

Resolving the different notions of time is critical for correct simulation. Figure 2 illustrates the temporal behavior of a discrete-event process and a differential equation process. As seen in Figure 2, execution of a discrete-event process is instantaneous. Time is not advanced during execution. On the other hand, a differential equation process's execution may advance the simulation time during execution. This means that the execution of a differential equation process may result in the execution influencing itself. This continuous time increment is the reason why a differential equation process is referred to as a *self-advancing* process. In the remainder of this paper, differential equation processes are denoted as self-advancing processes.

A mixed-mode simulator needs to coordinate between the different simulation processes. In current implementations of integrated mixed-mode simulators, this coordination is achieved by partitioning the self-advancing process into a large set of discrete-event processes [1]. However, this results in high communication costs, which have to be avoided if a network of workstations (NOW) is the target architecture. As this study is restricted to parallel mixed-mode simulation on a network of workstations, intelligent partitioning is essential for reducing the communications costs. Current implementations accomplish this partitioning by combining a set of processes belonging to the same self-advancing process and executing this partition on one processor. The same goal is achieved if the self-advancing process is considered as a single heavyweight process with respect to network synchronization. But this assumes that the simulator can synchronize the two different types of processes. To facilitate the synchronization of the self-advancing process with other discrete-event processes, a new synchronization scheme is introduced in this paper. This new scheme adopts a process based approach towards synchronization, and is called *process synchronization*.

Process synchronization of mixed-mode simulation processes involves the design of a synchronization interface which maps self-advancing processes to discrete-event processes. Instead of dividing the self-advancing process into individual discrete-event processes (as is the norm with current implementations of mixed-mode simulators), the process synchronization approach deals with the self-advancing processes as a single discrete-event process. Process synchronization protocols are required to bridge the

[*]This work was supported by the Defense Advanced Research Projects Agency under contracts J–FBI–93–116 and DABT63–96–C–0055.

[1]Every discrete event simulation contains a state variable called the simulation clock to model the flow of time during a simulation.

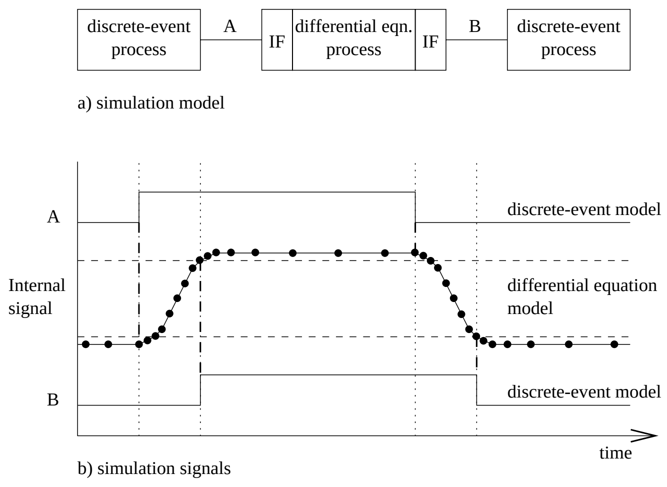

Figure 1. Mixed-mode simulation

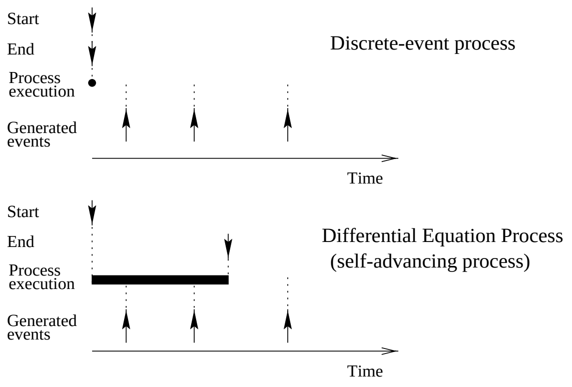

Figure 2. Mixed-mode processes

differences between the self-advancing process and a discrete representation of the same process. This has several advantages:

(a) There is a large set of algorithms available for differential equation model simulation. Each of these algorithms have different properties and performance factors. Depending on the model descriptions, the algorithm for best performance may vary. With the synchronization protocol approach, the algorithm can be dynamically selected during the simulation or statically determined before the simulation. Special attributes of a circuit may suggest the simulation algorithm to use (static selection). For example, existence of capacitive loads in a circuit suggests that waveform relaxation methods [10] are the best simulation algorithm. In addition, as the process synchronization protocols are process based, several different kernels can be used in the same simulation. Also, as self-advancing processes are mapped to a single discrete-event process no additional partitioning overhead is incurred.

(b) As basic optimistic discrete-event simulation stores states after each activation of a process, memory requirements are reduced in the case of a single self-advancing process. This is because the self-advancing process advances in its own time domain, such that state saving is only required at synchronization points. As synchronization is limited to specific simulation intervals, the number of states saved will be much lower than the number of states saved in the traditional optimistic discrete-event simulation. It is important to note that state and event histories are the major memory intensive components of the discrete-event simulation and by saving state only at synchronization points, a considerable amount of memory consumption is reduced. This is essentially a trade off between process granularity and memory requirements [6].

(c) Communication is reduced to interface function communication in the case of the self-advancing processes. As self-advancing processes have high internal communication demands, this communication is kept local (and not through the network of a NOW).

Tahawy *et al* [14] were the first to investigate synchronization of processes for mixed-mode simulation. However, the process synchronization scheme introduced by Tahawy only considered sequential event-driven simulation kernels. Due to the requirement of parallel simulation methods of today's large scale simulation models, the utility of Tahawy's scheme is limited. The new synchronization protocols proposed herein have been developed to support optimistic parallel simulation. Optimistic parallel simulation is considered to be one of the most promising paradigms for the simulation of large simulation models [13].

The remainder of this paper is organized as follows. As the synchronization protocols are specifically designed for optimistic discrete-event simulation, Section 2 reviews parallel discrete-event simulation (PDES) protocols and, in particular, the Time Warp [8] protocol. Section 3 discusses the new synchronization protocols. Some performance results and an analysis of the results is presented in Section 4. Section 5 presents some conclusions.

2 Background

In this section, a brief overview of PDES [7] is presented. In PDES, the model to be simulated is decomposed into *physical processes* that are implemented as *simulation objects*. Each simulation object is assigned to a *Logical Process* (LP); the simulator is composed of a set of LPs concurrently executing their simulation objects. Simulation objects communicate by exchanging time-stamped messages through the LPs. Thus, each LP (which can be associated with multiple simulation objects) receives messages from other LPs and forwards them to the destination objects. In order to maintain causality, LPs must process messages in strictly non-decreasing time-stamp order [8, 9]. There are two basic synchronization protocols used to ensure that this condition is not violated: (i) *conservative* and (ii) *optimistic*. Conservative protocols [5, 11] avoid causality errors, while optimistic protocols, such as Time Warp [7, 8] allow causality errors to occur, but implement some recovery mechanism.

In a Time Warp simulator, each LP operates as a distinct discrete event simulator, maintaining input and output event lists, a state queue, and a local simulation time (called *Local Virtual Time* or LVT). As each LP simulates asynchronously, it is possible for an LP to receive an event from the past (some LPs will be processing faster than others and hence will have local virtual times greater than others) — violating the causality constraints of the events in the simulation. Such messages are called *straggler* messages. On receipt of a straggler message, the LP must rollback to undo some work that has been done. The state and output queue are present to support rollback processing. Rollback involves two steps: (i) restoring the state to a time preceding the time-stamp of the straggler and (ii) canceling any output event messages that were erroneously sent (by sending *anti-messages*). After rollback, events are re-executed in the proper order.

3 Synchronization Protocol Approach

Correct parallel simulation depends on the behavior of the discrete-event processes, the self-advancing processes and the interface between the different time domains. To achieve correct parallel simulation, two different synchronization protocols are presented. The self-advancing processes simulate time intervals which have start and end times. In accordance to this, synchronization protocols are grouped by activation on the first event (start of the interval) or second event (end of the interval). A graphical representation (synchronization diagram) of the protocols is provided for simplifying the discussion of the protocols.

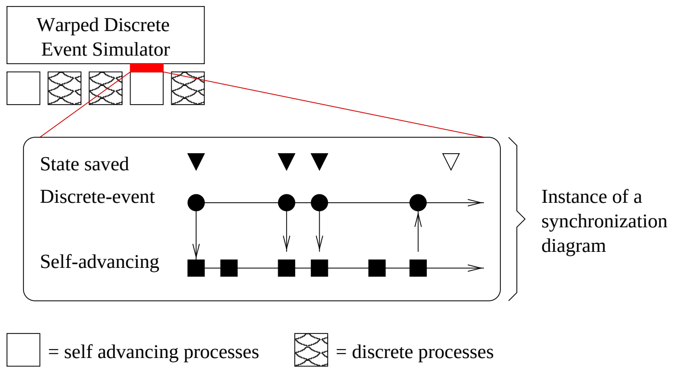
Warped Discrete Event Simulator
State saved
Discrete-event
Self-advancing
Instance of a synchronization diagram
= self advancing processes
= discrete processes

Figure 3. Synchronization diagram

3.1 Synchronization Diagrams

The relationship between the synchronization diagram and the simulation system is illustrated in Figure 3. A single synchronization diagram represents a snapshot of the interface between a single self-advancing process and the parallel discrete-event simulator. The state of a self-advancing process is marked by a state indicator. All state indicators shown in the diagram belong to the monitored self-advancing process. The same applies to the events shown in the diagram (i.e diagram does not show all the events in the system).

There are two kinds of state indicators. The black triangles represent the saved states (i.e states that were saved in the previous execution cycle). These states are accessible and rollbacks to these states can be initiated. The white triangles represent states which are going to be saved at the completion of the current execution cycle. Note that each saved state is associated with a discrete-event. This implies that a self-advancing process saves state only if there is an interaction with the discrete-event simulator. In between these points of interaction, the self-advancing process may have any number of changes to its state which will not be recorded.

Each synchronization diagram has two time lines. The upper one with circular markers shows the discrete events whereas the lower with the square markers depicts the changes in the state of the self-advancing process. The diagrams show how the discrete-events trigger the self-advancing process such that it advances in time. The exchange of information is indicated by arrows between the self-advancing and discrete-event time line. Again, only events concerning the depicted process are represented in the diagrams. The discrete-event time line can also be thought of as the communication (input/output) of the self-advancing process with any other process in the system. Because of the relaxed causality constraint of an optimistic simulator, the discrete time line does not necessarily portray the complete set of events related to the self-advancing process. In the following subsections synchronization diagrams are used to explain the individual synchronization protocols.

3.2 First Event Synchronization Protocol

In the First Event Synchronization (FES) protocol, the self-advancing process is handled as a discrete event process that is triggered by an event arriving at the start of the simulation interval. The process synchronization has to handle some exceptions due to the process's self-advancing nature.

Figure 4 illustrates the FES protocol. The self-advancing process is activated by the first event (t_n). The previous state stored contains the system state up to the event time (t_n). The self-advancing process attempts to simulate to the time of the next scheduled event (t_{n+1}) of the process. If no event is generated, the self-advancing process calculates all intermediate values, stops computation at t_{n+1} and stores its state. The state stored will be associated with the activation event (t_n) but it will contain the self-advancing process state upto time t_{n+1}. After the state is saved, a new synchronization point is reached. In case a threshold cross (event generation) occurs during computation of the internal values of the self-advancing process, the FES protocol requires the storage of the state and an artificial event at time $t_{n'}$ to reach another synchronization point. The synchronization is reached because the artificial event is responsible for reactivation and the state stored has the self-advancing process's values stored up to time $t_{n'}$.

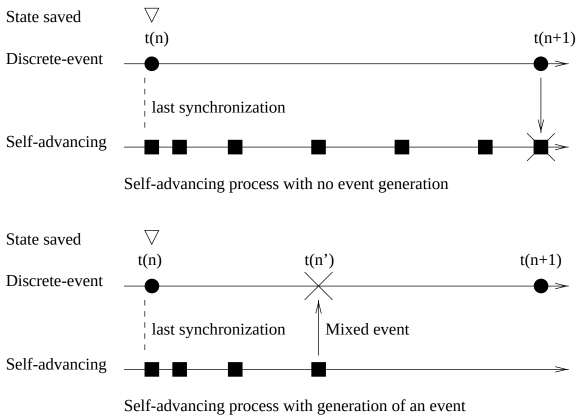

Figure 4. First Event Synchronization

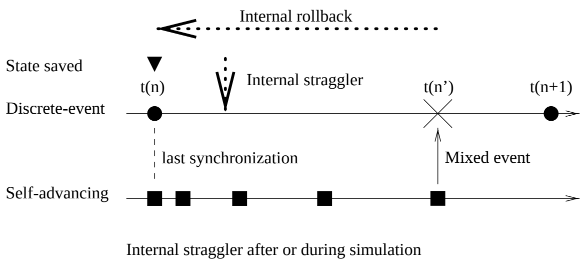

Figure 5. First Event Synchronization error

Several assumptions have to be satisfied to guarantee correct execution. The most obvious assumption is that the protocol assumes the time stamp t_{n+1} of the next event is available. This can not be guaranteed. Therefore, if no further event is scheduled, a stop time has to be picked arbitrarily (denoted as the maximum advance time). This is not necessarily a problem if we assume that the self advancing process is the bottleneck of the simulated system and the next event is generally available. A more serious problem arises from the use of the optimistic discrete event simulator. Although the optimistic protocol does not enforce the execution of events in causal order, it will take care of events before the current event (t_n). This, however, is not enough in a mixed-mode simulation because the computation in the self-advancing process has its own notion of time.

Figure 5 shows this case. Due to the event at time t_n, the self-advancing process is triggered to calculate the system behavior till $t_{n'}$ (the same error would occur if the self-advancing process did not cross the threshold). Now any event which arrives between the current trigger event at t_n and the final time $t_{n'}$ (or t_{n+1}) produces a self-advancing process error. Events in this interval (internal straggler) are not considered to be a causality error by the discrete system because the event receive time is greater then the time of the trigger event. On the other hand, the self-advancing process has already advanced to the end time $t_{n'}$ (or t_{n+1}). Therefore a rollback is required to restore correct initial conditions, remove unnecessarily created events, and resume execution from t_n up to the new received event time. The FES protocol is responsible for all tasks from the recognition of the internal straggler to the initiation of a rollback. This implies that there is a relationship between the time interval for the

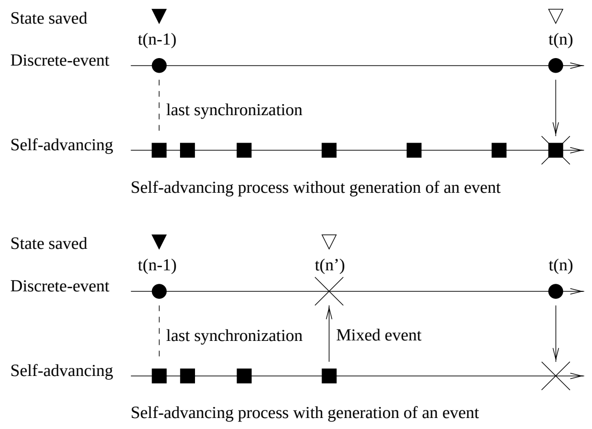

Figure 6. Second Event Synchronization

self-advancing process and the internal rollbacks. Therefore, the selection of the time step (which is picked arbitrarily) is a critical performance factor in the basic FES protocol.

Optimistic discrete-event simulators require a saved state at any time point to which the simulation may rollback. This requires that every process called during initialization returns a valid state. However, if the self-advancing process always advances in time (as described in the previous paragraph), this can not be ensured. Therefore, the FES protocol has to assure that this property can be supported by the self-advancing process. A straight forward solution is to prohibit time advancement during simulation initialization and enforce re-execution by scheduling an event with no delay. This results in a safe synchronization state from which the protocol can continue.

3.3 Second Event Synchronization Protocol

For the Second Event Synchronization (SES) protocol, synchronization is performed at the receive time of the event(s) determining the end of the self-advancing simulation interval. With synchronization at the end of the interval, the protocol does not need to pick a simulation end time arbitrarily.

The general approach is presented by the synchronization diagram in Figure 6. The previously executed event at time t_{n-1} marks the previous synchronization point to which the self-advancing process has simulated. The self-advancing process is reactivated by the event at time t_n and continues to simulate from t_{n-1} to t_n. If no threshold is crossed, the simulation reaches the time t_n and stops. A new synchronization point is automatically generated after the state is stored. The synchronization is complete because self-advancing simulation time as well as discrete time is t_n and all causality errors will be handled by the discrete-event system. In case an event is generated during the self-advancing simulation interval, the self-advancing process has to be interrupted. In addition to sending the generated event at time $t_{n'}$, the state associated with the time $t_{n'}$ has to be stored and the discrete-event simulator notified that the self-advancing process did not complete the calculations up to time t_n. One possible solution is to insert a dummy event at time $t_{n'}$ and make the system believe that the self advancing process was activated by this event. The change of the triggering event involves reactivating the event at time t_n. The state will then automatically be stored with the correct time stamp because of the change in the triggering event. A synchronization point is now reached as the time of the self-advancing process is the same as the discrete-event time.

Although there is no necessity for additional rollback detection, there exists some additional processing overhead. The state has to be saved if an event occurs during the self-advancing process execution. If not, the system can enter deadlock as shown in Figure 7.

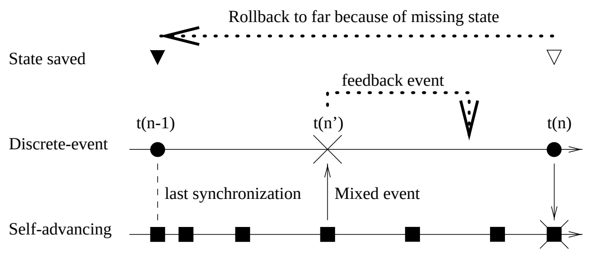

Figure 7. Second Event Synchronization error

In this case, an event occurred at $t_{n'}$. However, no state was saved and the self-advancing process continued up to t_n. If the discrete event at time $t_{n'}$ is responsible for an event in the interval $[t_{n'}, t_n]$ for the self-advancing process, a system rollback will be determined. Through the initiated rollback the system is reset to the last synchronization point. However, this removes both the event at time $t_{n'}$ and the feedback event which initiated the rollback. The system will result in the same state as before (namely at time t_{n-1}). Because no change in the system occurred, the simulation would again create the event at $t_{n'}$ which would result in a loop.

To ensure the existence of a second event at any time, the SES protocol needs to guarantee that at least one element is scheduled at the end of the simulation time. As this marks the end of the simulation interval, a large self-advancing process simulation interval could result. As for the FES protocol, the size of the simulation interval is an important performance issue. A large simulation interval represents a higher probability of a rollback. Due to this phenomenon, the SES protocol can also support a maximum simulation interval. The maximum advance time denotes the length of this maximum simulation interval. This maximum time value is again an arbitrary value and the optimum is dependent on the simulation model. Compared to the FES protocol, the maximum simulation interval is an additional option for the simulation protocol and not a requirement since simulation end time can be used as the interval end time.

4 Evaluation

As the field of mixed-mode simulation is a comparatively new research area, large simulation models are difficult to obtain. In addition, there is no standard mixed-mode simulation language with which standardized sets of benchmarks can be created. If such a standardized set of benchmarks were available, then the capabilities of mixed-mode simulators can be easily compared. The limitations in the hardware description language of ISPLICE3 for example, make even the comparison of a simple example circuit (*e.g.,* a resetable clock generator) impossible. This is because there is no way to incorporate input files or test benches.

However, performance comparisons are not entirely complete as the metrics for a reasonable performance comparison are not defined for mixed-mode simulators. The set of parameters concerning parallel mixed-mode simulation is very large. Memory requirements, communication demands, execution time, event granularity, and process granularity are just a few of the metrics that exist in parallel discrete-event simulation [3]. These metrics have to be expanded to address the attributes of differential-equation simulation to obtain an accurate index of parallel mixed-mode simulation performance. Some metrics native to differential-equation simulation include the accuracy of the solution, the sample interval of the signals, the generality of the simulation method and several others parameters that are completely alien to the discrete-event simulators. The issue of generality of the differential equation simulator is rather important as performance varies by orders of magnitude if only a subset of equations is supported.

The performance evaluation of the mixed-mode simulator with the new

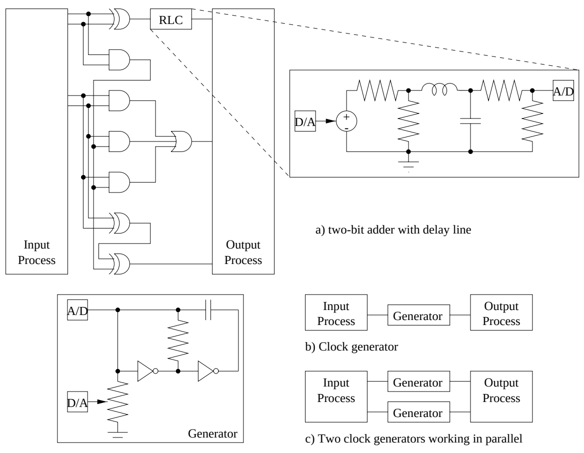
RLC
Input Process
Output Process
D/A
A/D
a) two-bit adder with delay line
Generator
b) Clock generator
c) Two clock generators working in parallel

Figure 8. Benchmark Circuits

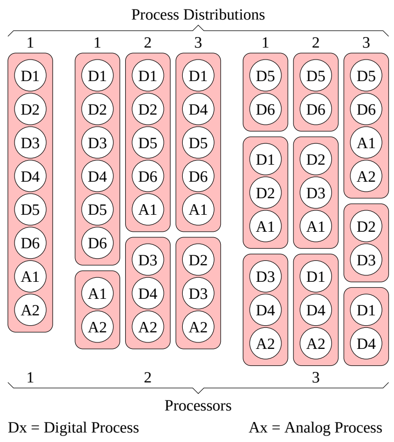

Figure 9. Example Process Distribution

synchronization protocols is therefore an attempt to present the capabilities of the new protocols. Currently automatic generation of test cases from the input model description to the simulation backend is not possible. Therefore, all performance models had to be hand coded; large test cases could not be used. However, the small test cases represent realistic test cases for mixed-mode simulation. Saleh [12] presents six test circuits which specifically stress the mixed-mode simulation interface issues. The generator circuit is from this set. Three circuits were considered: a two-bit adder with an RLC circuit as the delay line (8a), a clock generator consisting of a RC circuit connected to two NOT gates (8b) and a combination of two clock generators working in parallel (8c).

All the circuits are driven by an input testbench which provides either input vectors (Example 8a) or information to switch the generators on and off (Examples 8b and 8c). Output is reported through another discrete-event process which writes out the changes to the disk. Continuous values are also monitored by means of trace file outputs that provide detailed information about the internal node voltages in the analog islands.

The timings presented in Table 2 are the average of three different runs. Averaging was done to reduce the effect of system load fluctuations. In addition, in case of multiprocessor execution, three different process distributions were simulated. Figure 9 illustrates the process distribution for the two clock generators example. Table 2 presents a comparison of the different synchronization protocols. The effect of increased computation power and communication cost on the synchronization protocols can be

Clock Generator on two processors		
	Execution Time (sec)	
ext. Time Step	FES	SES
1E-09 s	286.12	278.55
1E-07 s	266.10	262.98
1E-05 s	270.78	254.39
1E-03 s	255.83	237.53
max. (Infinity)	263.59	238.25
Two Clock Generators on three processors		
	Execution Time (sec)	
ext. Time Step	FES	SES
1E-08 s	798.90	806.94
1E-06 s	985.81	807.68
1E-04 s	7215.03	5925.56
1E-02 s	4442.05	5188.90
max. (Infinity)	5017.95	6040.64

Table 1. Variation of maximum step size

investigated by varying the number of processors. Keeping the number of processors constant but changing the process distributions, illustrates the effect of locality of related processes on the protocol performance.

Because of the relatively small size of the differential equation islands, the internal step size of the differential equation process was used to increase the self-advancing process's workload. The internal step size in the differential equation process determines the maximum possible advancement of the differential equation simulator. Restricting this value results in a higher number of internally calculated time points and an increase in the self-advancing process's execution time (due to the higher granularity). A second set of measurements were taken to study the effect of varying the maximum advance time when the maximum internal time step is kept constant. To keep the number of measurements reasonable, the time step variation was investigated on only two cases; the clock generator on two processors and the two clock generators on three processors. The results are presented in Table 1.

The preliminary performance evaluation illustrates some interesting facts about the individual protocols. The performance values illustrated in Table 2, favor the SES protocol when it is a multiprocessor simulation. In most cases, the SES protocol performs as good as or better than the FES protocol. A reason why SES may be more efficient than FES is that SES is a less optimistic approach (with the synchronization performed on the second event). Unlike FES, where a lot of unnecessary computation may occur (and be rolled back), SES limits the number of such occurrences and reduces the wasted simulation time. Reducing the amount of wasted computation is a primary goal in mixed-mode simulation using an optimistic simulation approach.

One interesting observation is the effect of partitioning on the performance of the simulator, especially in the two clock generator example. If both self-advancing processes are assigned to the same partition (same processor), the high process granularity affects the simulator's performance adversely.

The influence of the step size provides additional insight into the algorithms. The SES algorithm performs slightly better if the internal step size (accuracy) is fixed (independent from the external step size) and the algorithm is allowed to proceed in greater steps. However, as the second example shows, larger external time steps make the synchronization protocols vulnerable to the distribution of the processes. In the case of the FES protocol, unnecessary computation is permitted, whereas in the SES protocol, the event in the far future (high risk of rollback) is used as a trigger.

		Simulation Execution Time (seconds)					
	Process	RLC Adder		Clock Generator		Two Generators	
Processors	Distributions	FES	SES	FES	SES	FES	SES
1	1	193.76	157.62	302.83	308.22	1826.14	1773.85
2	1	177.09	170.76	277.26	285.43	1973.30	1728.37
	2	181.02	156.08	264.20	272.42	1256.25	930.92
	3	265.97	303.73	271.29	261.49	1158.35	863.38
3	1	312.62	213.81	271.85	263.00	883.59	784.49
	2	310.52	183.35	302.15	271.16	858.34	825.55
	3	329.91	167.01	326.17	283.80	1672.81	1575.86

Table 2. Performance values of the synchronization protocols

Both these characteristics explain the poor performance in the last three measurements in Table 1.

5 Conclusion

When compared with a purely event-driven mixed-mode simulation approach, the process synchronization approach exhibits several advantages. They are itemized as follows:

(a) Reduction in memory usage: States are saved only at the beginning/end of the simulation interval and not for every intermediate value. This results in a decrease in the amount of memory needed for simulation given that they are an infinite number of intermediate values in a typical mixed-mode simulation.

(b) Naturally partitioned: Partitioning is trivial. The structure of the model is already partitioned. The modeler can improve the simulation performance by subdividing circuits into separate islands.

(c) Algorithm freedom: There is a large set of algorithms available for differential equation model simulation. Each of these algorithms have different properties and performance factors. Depending on the model descriptions, the algorithm for best performance may vary. With the synchronization protocol approach, the algorithm can be selected during simulation. As the synchronization protocols are process based, several different kernels can be used in the same simulation. Parallel execution of individual islands and local parallelism within the islands are areas of further improving the performance.

As a preliminary evaluation, the performance results indicate satisfactory simulation performance. The lack of other mixed-mode simulators or their performance measurements prevents any sort of absolute comparisons. In general, the synchronization protocols are suitable and can be be easily applied to any Time Warp based simulator. The preliminary results also show that none of the basic implementations actually allow optimal simulation performance. As the maximum simulation interval is dependent on the user specified accuracy (internal step size), it needs to be adjusted. This seems to be a necessary optimization as indicated by the huge differences in the measured simulation times. The determination of the optimal external time step by either a static or a dynamic algorithm is of great importance if further performance improvement is desired. Research is ongoing in this direction. Also the trade off between this approach with larger granularity vs. number of events through the network has to be investigated more throughly.

References

[1] E. L. Acuna, J. P. Dervenis, A. J. Pagones, F. L. Yang, and R. A. Saleh. Simulation techniques for mixed analog/digital circuits. *IEEE Journal of Solid-State Circuits*, 25(2):353–363, Apr. 1990.

[2] B. A. A. Antao and A. J. Brodersen. Behavioral simulation for analog system design verification. *IEEE Transactions on VLSI Systems*, 3(3):417–429, Sep. 1995.

[3] V. Balakrishnan, P. Frey, N. Abu-Ghazaleh, and P. A. Wilsey. A framework for performance analysis of parallel discrete event simulators. In *Proceedings of the 1997 Winter Simulation Conference*, Dec. 1997. (forthcoming).

[4] F. E. Cellier. *Continuous System Modeling*. Springer-Verlag, 1991.

[5] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–206, Apr. 1981.

[6] J. Fleischmann and P. A. Wilsey. Comparative analysis of periodic state saving techniques in time warp simulators. In *Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS 95)*, pages 50–58, June 1995.

[7] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[8] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):405–425, July 1985.

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM*, pages 558–565, July 1978.

[10] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-1(3):131–145, Jul. 1982.

[11] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, Mar. 1986.

[12] R. Saleh, S.-J. Jou, and A. R. Newton. *Mixed-mode simulation and analog multilevel simulation*. Kluwer Academic Publishers, 1994.

[13] L. P. Soulé and A. Gupta. An evaluation of the Chandy-Misra-Bryant algorithm for digital logic simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 1(4):308–347, Oct. 1991.

[14] H. E. Tahawy, D. Rodriguez, S. Garcia-Sabiro, and J.-J. Mayol. VHD$_e$LDO: A New Mixed Mode Simulation. *EURO DAC 1993*, 9/20–9/24 1993.

[15] B. P. Zeigler. *Theory of Modelling and Simulation*. John Wiley & Sons, 1976.