KAHN PROCESS NETWORKS APPLIED TO DISTRIBUTED HETEROGENEOUS HW/SW COSIMULATION

Dylan Pfeifer
Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas, U.S.A.

Dr. Jonathan Valvano
Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas, U.S.A

***Abstract*—Heterogeneous, distributed hardware/software cosimulation techniques using the backplane method encounter complex interface protocols for simulator communication and synchronization, limiting their adoption or abstraction. We simplify the dynamics of backplane cosimulation to the properties of a Kahn Process Network (KPN), such that tokens of the KPN are interpolated events. This simplifies the backplane API and reduces the coordination problem to a parameterization of token update rates. The performance of this method is reported on a timed ISS model for Freescale HC12 microcontrollers (TExaS) coordinated with a Spice (Ngspice) circuit simulation.**

Keywords: distributed cosimulation, heterogeneous cosimulation, Kahn Process Networks, hardware/software cosimulation, cosimulation backplanes, system level design

I. INTRODUCTION

System level design (SDL), considered the "next frontier in EDA" [1], requires heterogeneous cosimulation of hardware and software models at flexible levels of abstraction. Distributed cosimulation requires coordinating multiple simulations across spatially separated computing resources, for parallelism and other benefits [2], although there is a communication overhead cost. The use of software backplane techniques to implement distributed, coordinated cosimulation is covered in [3][4][5], and formalisms for discrete and continuous model coordination are given in [6][7]. Backplane techniques surrender simulator control and communication to an independent software agent that distributes signals and enforces event synchrony. The coordination protocols for these solutions, however, are complex. We offer, rather, a lightweight protocol that implements the rules of a Kahn Process Network (KPN) [8]. This simplifies connector software implementation and abstracts backplane and discrete/continuous synchronization to a data flow problem. "Full" or "predicted events" discrete/continuous coordination [7] may be achieved by adjusting signal producer rates of interpolated events in the KPN. We describe the backplane and interface implementing the KPN, define "interpolated" events as the tokens of the KPN with the formalism of [9], and measure the performance of an implementation coordinating a microcontroller timed ISS and a Spice circuit simulation. We introduce a Spice socket device that provides concurrent distributed Spice cosimulation without netlist exchange.

II. PRIOR ART

The Coordination Problem

Homogenous and heterogeneous cosimulations coordinating discrete-event and continuous-time models of computation must solve the "coordination problem." We define coordination to be both the time synchronization and event communication between process-separated simulators, each with independent criteria for time advancement. In short, simulators must exchange signal information across connections, but must, per the *local causality constraint* (LCC) [10], ensure that local time advancement does not exceed events not yet communicated from other simulators, unless rollback is offered. This is challenging for discrete-event based simulators (which may have zero-delay events) coordinated with continuous simulators, which may experience dynamic time stepping, non-convergence, or other different internal criteria for time point acceptance. Solutions however are offered in the *-AMS languages (SystemC-AMS, Verilog-AMS, VHDL-AMS), and hybrids such as Xspice [11]. Performance comparisons of *-AMS environments are given in [12]. Heterogeneous two-simulator connections (called "ad-hoc" solutions in [3]) nearly span the combination set of any two popular environments, but coordinated synchronizations among three or more simulators exhibit techniques seen in backplanes.

Improving Backplane Techniques

Cosimulation backplanes distribute information across connected simulators and control simulator time advancement. An *interface* must be constructed for each simulator that connects to the backplane software agent and implements the coordination API. Because existing solutions implement variants of the discrete/continuous coordination described in [6], the interfaces and APIs are complex, and may not be available for all simulators desired. Although the Common Framework Initiative (CFI) [3] offers a standard for simulator connection, it is a complex protocol that may not be fully implemented in every EDA environment.

We offer both a simplified interface, and an abstraction for the backplane synchronization that reduces to the properties of

a Kahn Process Network, gaining implementation simplicity. Simulators connect to the backplane in a client/server relationship over inter-process communication (IPC), and simulator interfaces perform read or write operations on connection FIFOs following the rules of a KPN. The simulator interface therefore only implements token passing, and the backplane only implements the KPN FIFOs and the node connection graph. The backplane forwards tokens from signal producer to signal consumer through a FIFO, and nodes are connected, concurrent, independently running simulators. Blocking reads on FIFOs combined with signal producer rates realize the requirements of simulator coordination. An interpolated event token, read from an input FIFO, communicates both signal value and value expiration, providing a future event time when the simulator must re-sample the input FIFO. This removes time step and synchronization control from the responsibility of the backplane and interface API. The synchronization and control is implied in the token data, and synchronization is parameterized by token update rates of signal producers. These can be assigned statically or dynamically. The properties of Kahn Process Network and interpolated events will now be discussed.

III. KAHN PROCESS NETWORKS

Kahn Process Networks (KPNs), named for Gilles Kahn, authoritatively defined in [8], are dataflow networks with these high-level properties:

- The KPN is a directed graph with arcs, representing simplex FIFOs, and nodes, representing concurrent compute elements without interdependent side-effects.
- Nodes may read from input FIFOs and write to output FIFOs, but reads are blocking (the node stalls) if the FIFO is empty, while writes always succeed (the node does not stall).
- Nodes may not conditionally execute by FIFO sniffing.
- FIFOs are unbounded (infinite depth).
- The KPN is independent on the order of node execution if the KPN dataflow rules are followed. The KPN is completely determined by its node set, arc set, initial conditions, and FIFO producer rates.

KPNs reduce to synchronous data flow networks if token production rates are static and known a priori [9]. KPNs are deterministic based on initial conditions if the FIFO rules are followed. We suggest the KPN rules are sufficient to solve the simulator coordination problem if tokens are interpolated events, if FIFOs are simulator connections, and if nodes are concurrent, distributed simulators. This greatly simplifies the cosimulation interface and backplane architecture, while providing a strong abstracted model (KPN).

IV. INTERPOLATED EVENTS

We formally define interpolated events as the tokens of the KPN FIFOs. Using set-theory formalism, Lee and Sangiovanni-Vincentelli [9] define an event as an element of the Cartesian product $V \times T$, where V is a set of values, and T is a set of tags. The event (v, t) is an element of $V \times T$ with value $v \in V$ and tag $t \in T$. If T is the set of real numbers $\boldsymbol{R}$ or integers $\mathbf{Z}$, events can be totally ordered by the total order on the tag set. If T is empty, the events are "untimed" [13]. A signal is defined as a set of events.

We introduce to this formalism the "interpolated event" (IE), an element of the set $V \times T \times T$. This is an element (v, t_m, t_n). If T is totally ordered, we define the value v to be constant on the interval $[t_m, t_n)$ for the interpolated event specified by the tuple (v, t_m, t_n). It includes all single events (v, t) of the signal such that v is constant and $t_m \le t < t_n$. Interpolated events collapse to single events *iff* $t_m = t_n$. Interpolated events collapse to untimed events *iff* $t_m = t_n$ = NULL, where NULL is defined to indicate that no tag values (t_m , t_n) are given for the interpolated event value v.

If a simulator consumes an interpolated event, it may assume the value v for the signal is stable on the tag range $[t_m, t_n)$, and not need sample the value again until t_n. The interpolated event therefore communicates a signal value v, a value start tag t_m, and value expiration tag t_n. Interpolated events are appropriate for models that sample, or for which time is discrete (even computed continuous time models advance discretely, however small, since computed number formats are finite). Although the signal may change from t_m to t_n in reality, it does not change from the operational perceptive of a sampling consumer.

We define operations and relations on interpolated events as follows:

Concatenation: $(v_0, t_j, t_k) \oplus (v_1, t_m, t_n) = (v_0, t_j, t_k)$ *iff* $v_0 = v_1$, $t_n > t_k$, and t_m is the supremum of (t_j, t_k). Concatenation extends the stable time of v_0 and can be used to extend rendezvous time of a signal consumer.

Order: $(v_0, t_j, t_k) < (v_1, t_m, t_n)$ *iff* $t_j < t_m$. Order is required to select the greatest IE in a FIFO containing a selection time t.

V. SIMCONNECT AND SIMTALK

We created a KPN server "SimConnect" which implements the FIFOs and directed graph of a KPN. Clients, the nodes of the KPN, are connected simulators which communicate through the "SimTalk" API message passing protocol. They connect in a client/server relationship with the SimConnect server to read and write interpolated event tokens to their FIFOs. Because the backplane/simulator relationship is an applied KPN network, SimConnect and SimTalk do not concern themselves simulator instantiation, lockstep, or time advancement. Rather, the t_n value of each produced IE token indicates the next time a consuming simulator should read the input FIFO for the signal value. The consuming simulator may operate independently from t_m to t_n on an interpolated event with the constant value v for that input token, and produce output tokens while operating. When reading a new token, the simulator is blocked, following KPN.

SimTalk therefore is a very lightweight backplane API, with four basic messages from client to server:

- *Subscribe <signal name>* – allocate an input FIFO arc to receive interpolated events on signal <name>.

- *Broadcast <signal name>* – allocate an output FIFO arc to post interpolated events on signal <name>.
- *Get <signal name, t>* – A FIFO read operation. Get the greatest interpolated event (v, t_m, t_n) on the signal <name> FIFO such that $t_m \leq t < t_n$. The consuming simulator blocks until an IE is available from the server.
- *Set <signal name, v, t_m, t_n>* – A FIFO write operation. Post an interpolated event (v, t_m, t_n) on signal <name> FIFO. The server delivers it to all FIFOs registered to receive interpolated events on signal <name>.

From server to client there are messages Kill and Reset, directing the simulator to self-exit, or self-reset, and messages supporting flow control, rollback, and dynamic IE resolution.

- *TxACK* – After a signal producer posts, it can optionally wait for a server-to-client TxACK message indicating the posted IE has been consumed by at least one consumer node. This can flow control signal producers in acyclic networks so that producing nodes do not excessively out-pace consuming nodes.
- *Rollback to <t>* – For optimistic execution, a rollback message from server to client directs the client to rollback to an earlier local time to process a "straggling" event [10]. The node must support state saving and restoration to support rollback messages.
- *Reply back at <t_k>* – For dynamic IE signal resolution, the server can instruct the client to post the next IE at the node's local time t_k. For example, if a producer posts an IE (v, t_m, t_n), due to update again at time t_n, the server can send a "reply back at <t_k>" message instructing the producer to post the next IE at time t_k instead. If $t_k > t_n$, the signal resolution has been relaxed. If $t_k < t_n$, the signal resolution has been increased. For the posted IE (v, t_m, t_n), the server replaces the t_n value with t_k when the IE is delivered to the FIFO.

Dynamic resolution affects global simulation speed by reducing the number of SimTalk messages posted by signal producers, effectively changing the sampling rate on a signal and duration a consumer executes on an IE before re-querying the FIFO. This is useful if during a period in the simulation a signal is ignored (such as when interrupts are off in the simulated microcontroller), or when high resolution is not required (such as when a signal polling thread is not running in a simulation of multi-tasking).

VI. System Configuration

Figures 1 and 2 illustrate how a hardware/software cosimulation maps to the SimConnect/SimTalk client-server architecture. Consider an example problem of source-level debugging of a real-time operating system (RTOS), network protocol stack, and application layer code over two virtual 9S12 microcontroller targets in TExaS. Cycle accurate soft IP for the CAN controller peripherals is provided in Verilog or VDHL, and an Ngspice deck simulates the CAN physical layer. The microcontrollers communicate with the CAN controllers through a memory-mapped register interface. Figure 1 provides a conceptual hierarchy of the system from an electronic design automation (EDA) point of view.

Figure 2 illustrates the client/server mapping of the EDA components. The SimConnect server provides the cosimulation backplane, servicing SimTalk protocol messages from connected EDA components. Each simulator or connected consumer (logger, plotter) runs in its own process, which can be distributed across multiple machines where TCP/IP sockets provide the IPC. Simulators register to receive IE tokens from the SimConnect server on input signal names, and deliver IE tokens to the SimConnect server on output signal names. The SimConnect server iteratively delivers IE's from signal producer to consumer through the KPN FIFOs, buffering IE's in each FIFO until a consuming simulator sends a get request.

In this manner the simulators have no state, control, or shared memory "awareness" of one another (per KPN rules); they only depend on their dataflow tokens from input FIFOs as they subscribe to them, and output FIFOs as they broadcast to them. Global simulation advancement is therefore driven by IE token producers, rather than centralized simulator step control. To avoid startup deadlock, initial condition IE's are assigned statically on each FIFO at time t = 0 with an appropriate initial duration [0, t_n), set by the user according to an appropriate property of the system, or a known initial sampling rate for the IEs on the FIFO.

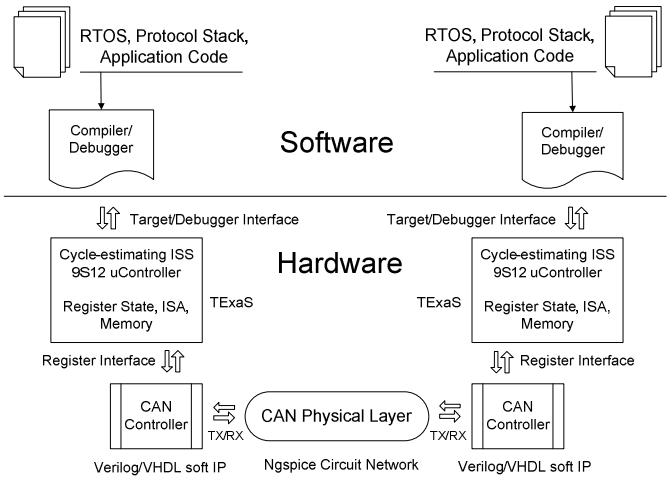

Figure 1. System Cosimulation, EDA Hierarchy

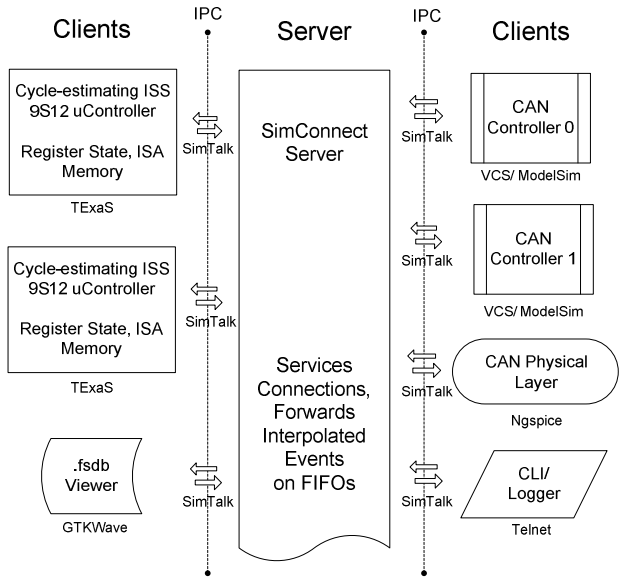

Figure 2. System Cosimulation, Client-Server Hierarchy

Implementing the SimTalk protocol can be done through function calls in a local process or IPC in distributed processes. We first implemented SimTalk through BSD socket calls with blocking reads, non-blocking writes, and ASCII string message content. SimConnect ran as a daemon servicing simulator connection requests and forwarding interpolated event tokens from signal producer to signal consumer queues.

Since signal token producers drive global simulation time advancement, IE rates can be assigned statically based on a desired signal resolution, the sampling rate of a consumer simulator (such as the minimum clock on a cycle-accurate simulator), or adjusted dynamically against speed/accuracy tradeoffs at desired times in the simulation.

VII. Implementation

We implemented the SimConnect server in C with BSD Socket-based IPC on a Linux OS 2.6.16 kernel workstation. We wrote socket-based SimTalk connectors for the open-source Ngspice suite [14], which combines Berkeley Spice 3 and Georgia Tech's Xspice [11], and for the Freescale HC12* based TExaS (TExaS Execute and Simulate) timed ISS [16]. We modeled a very simple microcontroller software-based voltage controller of an Ngspice-modeled RC network. Feedback was modeled based on a voltage output driver on a 9S12 GPIO output pin, PT[0], and threshold comparators wired to GPIO input pins, PM[1:0].

For the Ngspice connector, we wrote a novel "socket_device" as an Xspice device which compiled into the Ngspice environment as a user-defined behavioral device. No change to the Ngspice kernel was required. The socket device, hostname of the SimConnect server, connection port, and signal name FIFO of the device were specified in the text-file Spice deck for Ngspice as in Figure 3. We assigned one input or output FIFO for each socket device. The socket device consumed 4-state digital values and bridged them through Xspice adc_bridge and dac_bridge devices to the analog RC circuit.

```
.model socket_output_PM0 d_socket_output
+            (signal  name="PM0"
+             hostname="SimConnect" port=8000)
```

Figure 3. Ngspice deck socket device for SimTalk FIFO connection

The control software in TExaS polled Ngspice socket device values through GPIO instructions such as:

```
keepCharging
    brset PM,#$01,turnoff    ; high voltage limit signal on PM[0]?
    bra keepCharging         ; no -> keep charging
```

Figure 4. 9S12 assembly polling PM[0] for Ngspice output FIFO values delivered from SimConnect

VIII. Synchronization

We achieved conservative, predicted-events [7] synchronization between the TExaS simulator, a time-driven, clocked synchronous model of computation (MoC), and Ngspice, a time-driven, dynamically stepped model of computation with statically timed IEs of 125 ns duration. In the clocked synchronous MoC, input and output signal exchanges occur at ends of a fixed period, but a one evaluation cycle delay occurs from signal input to output result [13]. For example, the TExaS simulator continually executes a GetInputs(), Evalutate(), PostOutputs() cycle in its local evaluation. With a local evaluation cycle of 125 ns, there is a 125 ns delay from the operational effect of an input appearing on an output if they are related.

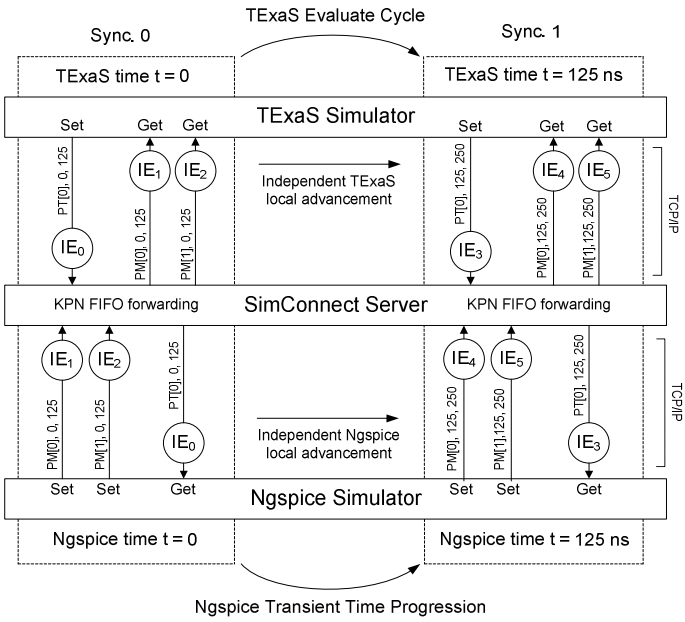

Figure 5. Conservative, Predicted Events Synchronization

In Figure 5, the Xspice socket devices and TExaS were configured to post output IE's of $(t_n - t_m)$ = 125 ns duration, the predicted event interval, or TExaS evaluation cycle. Initial condition IEs of 125 ns duration were posted to FIFOs at startup so each simulator could advance and post after the first Get operations. The IEs posted by TExaS, consumed by Ngspice, allowed the Spice kernel compute freely for 125 ns, posting IEs before blocking at sync point 1 in the figure to re-query the SimConnect server again. Up to each sync point, established by the expiration time of an input IE, simulators advance independently, without local time coordination. For resolution, if the IE durations are less than the TExaS evaluate cycle, or $(t_n - t_m) < 125$ ns, the FIFOs oversample. If the IEs are greater in duration, $(t_n - t_m) > 125$ ns, the execution is optimistic, since TExaS advances to the next evaluation cycle on an IE that could change during the evaluation cycle, but which was declared to be constant on an IE range larger than the cycle. If the IE duration increases further, $(t_n - t_m) >> 125$ ns, the signal resolution decreases, decreasing message count, increasing the time between synchronizations, and decreasing accuracy. However, for low frequency input signals compared to the TExaS clock, resolution can be decreased to an appropriately large $(t_n - t_m)$ period, say the period of the Nyquist frequency of the input signal. If an appropriate resolution is unknown, it can be observed first at a high resolution rate $(t_n - t_m) << 125$ ns, then adjusted to a lower resolution to increase simulation speed. The *local causality constraint* (LCC) is observed because simulators do not advance beyond time points on inputs until an input IE is provided for that time point with a future expiration value (t_n).

IX. Results

We tested the speed of the SimConnect backplane as a function of geographical distribution and interactive plotting.

TABLE I. COSIMULATION SPEED WITH SIMCONNECT/SIMTALK

Case	TExaS/Ngspice Software-Based Voltage Controller				
	Simulation Time Elapsed	***Run Time***	***HC12 Cycles***	***Spice Time Points***	***SimTalk Messages***
A	100 us	85.4 s	800	966	2400
B	100 us	81.1s	800	966	2400
C	1 ms	18.9 s	8k	9810	24k
D	1 ms	13.2 s	8k	9810	24k

A. Metropolitan WAN, interactive Ngspice plotting and IE prints

B. Metropolitan WAN, no Ngspice plotting or IE prints

C. Corporate LAN, interactive Ngspice plotting

D. Corporate LAN, no Ngspice plotting

The communication overhead cost of backplane cosimulation manifests in the differences in speed between cases A and B, where TExaS and Ngspice were separated from the SimConnect server over a metropolitan WAN, and cases C and D, where all components were hosted on a corporate LAN. In cases A and B, with the network latency, the interactive plot workload of Ngspice had a lesser effect on speed of cosimulation versus cases C and D, where communication latency was lower. In case D, the local workload of interactive Ngspice signal plotting affected latency. SimTalk and SimConnect used TCP/IP packets, but could have used UDP packets on the LAN for further speedup.

The socket wait time on SimTalk Get responses was measured on the Xspice input socket devices to determine the relative duration of the blocking overhead versus the execution/post cycle. Using the IA-32 Time Stamp Counter (TSC) register, we found that over ninety-eight percent of the cycle, from one Get operation to the next, Ngspice was blocked waiting on an IE. This means that during remaining the two percent of the compute time, it posted two non-blocking IEs, advanced the transient simulation to the next IE break point, sent plot values, and queried the socket again for a new IE. Since Ngspice spent most of its runtime sleeping, there is considerable margin for decreasing the IE delivery delay in the simulation until the cycle evaluation time becomes the majority delay.

Case A and B speeds, although quite slow, yet facilitated observing the interaction of blocks of 9S12 instructions (on the order hundreds of instructions) with electrical circuit effects through GPIO pins. Case C and D speeds allowed for monitoring 9S12 software task switching at a 10 ms tick time, but generally, simulation speed in this environment must increase to allow efficient runtime monitoring of system dynamics on the range of full seconds.

X. Future Work

Implementing the SimTalk connectors did not require many lines of code (around 100 lines each) for TExaS and Ngspice, since the connectors only consisted of the socket initiation ritual, socket data servicing, and parsing of SimTalk message content for delivery into local data structures. The Ngspice connector, for example, inserted Spice breakpoints for socket re-query based on the input IE t_n values, and the rest of the time (IE duration), returned the consumed IE value to the Spice kernel when polled for its value. Since IPC-based sockets, or IPC shared memory are sufficient to implement the KPN FIFO read/write rules, it is desired to expand the set of SimTalk-supported simulators by writing plugins for popular simulation environments with OS host interfaces. For performance, we only profiled the distributed performance of SimConnect, we have not yet profiled a shared-memory IPC implementation, which can witness a reduction in latency compared to sockets. Is it also possible to optimize message processing if the SimTalk messages are assigned a binary packet structure rather than ASCII strings.

We intend to pursue an empirical study of scalability of SimConnect (increasing the number of connected simulators). The cost of adding simulators is mainly loaded on the SimConnect backplane, which must ferry more messages, but lightly loaded on individual simulators such that their input FIFOs remain the same. That is, a simulator has no awareness of the number of total system simulators, only its own I/O ports, so latency from proliferation can only affect the

simulator at its wait time on an input FIFOs. The compute time of the simulator, rather, during the evaluation cycle, is only lightly affected by adding more simulators, since writes do not block. However, the collective time advancement of the global simulation is affected by adding simulators with cyclic dependencies of a high resolution IEs verses components of low resolution IEs. The global simulation will only pace at the fastest execution of the high resolution cyclic components if the LCC is observed and the simulation advances conservatively.

XI. SUMMARY

SimConnect and SimTalk offer a lightweight API and dataflow based synchronizer to the field of cosimulation backplane techniques. This simplifies connector implementation considerably, and abstracts the simulator coordination problem away from time-step control of connected simulators. Rather, through the rules of KPN FIFOs, and the construct of interpolated event types as the KPN data tokens, both full and predicted events synchronization may be achieved between distributed discrete and continuous time simulations. This offers another study of KPN instantiation [15], and quickly allowed us to connect a timed ISS microcontroller model (TExaS) to a system conservative, dynamically time stepped Ngspice circuit. We introduced the user-defined Xspice socket device model to the Ngspice environment that implements the SimTalk API between Ngspice and the SimConnect server without modifying the Ngspice kernel. Additionally, we introduced the interpolated event (IE) data type to simulation formalism which encapsulates event value, start, and expiration information in a single token for synchronization through distributed dataflow networks rather centralized time control.

REFERENCES

[1] Sangiovanni-Vincentelli, A., "Quo Vadis, SLD? Reasoning about the Trends and Challenges of System Level Design," *Proceedings of the IEEE* , vol.95, no.3, pp.467-506, March 2007.

[2] Amory, A., Moraes, F., Oliveira, L., Calazans, N., and F. Hessel, "A Heterogeneous and Distributed Cosimulation Environment," *Integrated Circuits and Systems Design, 2002. Proceedings. 15th Symposium on* , vol., no., pp. 115- 120, 2002.

[3] Schmerler, S., Tanurhan, Y., and K.D. Muller-Glaser, "A Backplane Approach for Cosimulation in High-level System Specification Environments," *Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European* , vol., no., pp.262-267, 18-22 Sep 1995.

[4] Atef, D., Salem, A., and H. Baraka, "An Architecture of Distributed Cosimulation Backplane," *Circuits and Systems, 1999. 42nd Midwest Symposium on* , vol.2, no., pp.855-858 vol. 2, 1999.

[5] Sung, W., and S. Ha, "A Hardware Software Cosimulation Backplane with Automatic Interface Generation," *Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific* , vol., no., pp.177-182, 10-13 Feb 1998.

[6] Gheorghe, L., Bouchhima, F., Nicolescu, G., and H. Boucheneb, "Formal Definitions of Simulation Interfaces in a Continuous/Discrete Cosimulation Tool," *Rapid System Prototyping, 2006. Seventeenth IEEE International Workshop on* , vol., no., pp.186-192, 14-16 June 2006.

[7] Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., and E.M. Aboulhamid, "A SystemC/Simulink Cosimulation Framework for Continuous/Discrete-Events Simulation," *Behavioral Modeling and Simulation Workshop, Proceedings of the 2006 IEEE International*, vol., no., pp.1-6, 14-15 Sept. 2006.

[8] Kahn, G., "The Semantics of a Simple Language for Parallel Programming," *Information Processing*, pages 471-475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[9] Lee, E.A., and A. Sangiovanni-Vincentelli, "Comparing Models of Computation," *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on* , vol., no., pp.234-241, 10-14 Nov 1996.

[10] Zeng, Y., Wentong C., and S.J. Turner, "Causal Order Based Time Warp: A Tradeoff of Optimism," *Proceedings of the 2003 Winter Simulation Conference* , vol.1, no., pp. 855- 863 Vol.1, 7-10 Dec. 2003.

[11] Cox, F.L., W.B. Kuhn, H.W. Li, J.P. Murray, S.D. Tynor, and M.J. Willis, "Xspice User's Manual." Computer Science and Information Technology Laboratory, Georgia Tech Research Institute. Atlanta, December 1992.

[12] Narayanan, R., Abbasi, N., Zaki, M., Al Sammane, G., and S. Tahar, "On the Simulation Performance of Contemporary AMS Hardware Description Languages," *Microelectronics, 2008. ICM 2008. International Conference on* , vol., no., pp.361-364, 14-17 Dec. 2008.

[13] Jantsch, A., Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation. Morgan Kaufmann, 2003.

[14] Nenzi, P., and V. Holger, "Ngspice Users Manual." V. 22, Sep 2010, ngspice.sourceforge.net.

[15] Vrba, Z., Halvorsen, P., and C. Griwodz, "Evaluating the Run-Time Performance of Kahn Process Network Implementation Techniques on Shared-Memory Multiprocessors," *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on* , vol., no., pp.639-644, 16-19 March 2009.

[16] Valvano, J., Embedded Microcomputer Systems: Real Time Interfacing, 3rd edition. CL-Engineering, 2012.