Simulation Bridge: A Framework for multi-processor simulation

G.D. Nagendra
Software Development Systems
Texas Instruments India Ltd.
Bangalore, India
nagendra@india.ti.com

V.G. Prem Kumar
Software Development Systems
Texas Instruments India Ltd.
Bangalore, India
vadapall@india.ti.com

B. S. Sheshadri Chakravarthy
Software Development Systems
Texas Instruments India Ltd.
Bangalore, India
shesha@india.ti.com

ABSTRACT

Multi-processor solutions in the embedded world are being designed to meet the ever increasing computational demands of the emerging applications. Such architectures comprise two or more processors (often a mix of general purpose and digital signal processors) together with a rich peripheral mix to provide a high performance computational platform. While there are many simulation solutions in the industry available to address the system partitionaing issues and also the verification of HW-SW interactions in these complex systems, there are very few solutions targetted towards the SW application developers' needs.

The primary concern of the SW application developers is to debug and optimize their code. Hence, cycle accuracy and performance of the simulation solution becomes the key enablers. Desired observability and controllability of the models are additional careabouts. Secondly, application devlopers are more comfortable at instruction level simulations than they are with RTL or gate level simulation. These specific requirements have a bearing on the choices in the simulation solutions.

This paper describes the design of a generic, C based multi-processor instruction set simulator framework in the context of the above parameters. This framework, termed the "simulation bridge", facilitates highly accurate, yet efficient simulation. The SimBridge performs clock correct lock-step simulation of the models in the architecture using a global simulation engine that handles both intra-processor and inter-processor communication in a homogenous fashion. It addresses the multiple key issues of execution control, synchronization, connectivity and communication.

The paper concludes with the performance analysis of the SimBridge in an experimental test setup as well as in the Texas Instruments (TI) TMS320C54x-based simulators.

Keywords

Multiprocessor simulation, Instruction Set Simulator, Simulation Framework

1. INTRODUCTION

Emerging embedded applications have necessitated the advent of high performance DSPs to meet their computing needs. Among the multiple options available to the design community, of late, one has witnessed the emergence of multi-processor DSP solutions as well as higher level of integration to satiate the MIPS requirements. For instance, in the Texas Instruments' context, there are the TNETC4320, OMAP1510 and the TMS320C5471 platforms integrating microcontroller and DSP onto a single chip, and devices like TMS320C5421 & TMS320C5441 comprising 2 to 4 DSP cores together with a rich mix of shared peripheral subsystems such as DMAs, memories, serial ports and other device controllers.

The application developer (we use the term "application developer" to refer to "sw application developer") has to comprehend a number of features of such architectures in order to implement and optimize solutions. These features include the individual instruction set architectures of the constituent processors, access characteristics of shared memory, necessary address translations, characteristics of background memory transfers, inter-processor communication semantics, use of shared peripheral subsystems, interrupt behaviour, boot-up modes, and inherent master-slave relationships. The developer is, thus, most often interested in running real applications on such architectures to gather architecture understanding, perform validations and application optimizations. It is also important to note that application developers are more often happier with instruction level visibility than with RTL or gate level details. This requires cycle accurate yet abstracted, high-performance and full-featured simulation platforms that make realistic application simulation feasible, both in terms of performance as well as modeling detail.

Given this recurring need for multi-processor simulators, it is intuitively appealing to construct an architecture- agnostic simulation framework that hosts common simulation infrastructure of instruction set simulators. In general, there are several factors that influence the design decisions for such a framework. From application development stand-

point, two of the key issues are simulation performance and cycle accuracy. These are key enablers for large-scale application debug and optimization. Further, other factors such as the observability (to a debugger), controllability and configurability of the simulation come to bear on the design choices.

These criteria thus define the scope and available choices for designing the multi-processor simulation framework. This paper describes the *Simulation Bridge* (*SimBridge*, for short) which has been designed to be a generic multi-processor simulation framework. SimBridge is an architecture-agnostic framework that schedules and synchronizes an arbitrary number of processor models at clock granularity. It also provides mechanisms for processor models and other components to be "wired up" and to be able to communicate with one another in a homogenous fashion.

Section 2 takes a quick look at existing literature on multiprocessor simulation in the light of addressing application developers needs. This is followed by Section 3 which motivates the need for the simulation bridge by discussing key factors involved in the design of a generic simulation framework. Section 4 dives into a detailed presentation of the SimBridge design. Section 5 presents quantitative measurements of SimBridge performance on two case studies: an experimental test setup and the TI TMS320C54x based simulator. The paper concludes with a note on future work in section 6.

2. CURRENT STATE OF THE TECHNOLOGY

Multiprocessor simulation is a problem already well addressed in the EDA world. Specifically Seamless (refer [9]), VCC (refer [1]), and other HDL simulation environments provide us with the ability to create and simulate multiprocessor designs. HDL simulation environments typically work with RTL level capture of the models, which puts a big constraint on the simulation performance. Seamless does better by allowing one to interface instruction set simulators with HDL simulations, thus allowing for instruction level abstractions from the application developers perspective. While we can get better performance using Seamless and its optimizations modes, the speed is still constrained by the slowness of the HDL simulation. Seamless works well in the area of low level HW-SW interactions, but is still slow from full application execution perspective. VCC allows for various levels of capture of the target system, often trading off performance for accuracy. While this approach works well for HW-SW exploration & partitioning, it is not best suited for application debug and tuning.

There are a few other published works on various efforts in multi-processor simulations. Most of these provide evaluation frameworks for memory architecture evaluation (refer [5], [12], [2] and [10]). They work with a specific target instruction set while allowing for change in the number of processors and the memory architecture. Luis Barriga et. al. (refer [8]) give a very good overview of different approaches on multi-processor simulation and their relative merits and demerits.

A few other solutions are more relevant to the problem that we are trying to address, but no one of them seems to adrress all the issues. Augmentation based SPARC simulator (refer [2]) and the Wisconsin Wind Tunnel (refer [10]) discuss synchronizations across multiple threads and the granularity for synchronization, but both of them work with a specific instruction set simulator and are not generic to be applied for multiple instruction set architectures. Pia(refer [6]) and the Hierarchical Architecture Simulation Environment (refer [3]) discuss frameworks that comprehend communication and abstraction of communication, but both separate the simulation frameworks from the model that needs to be simulated. While appealing as frameworks, they lead to loose coupling between the framework and the simulation models.

3. KEY CAREABOUTS

Instruction set simulators have done well to address the needs of the application developers in the single processor domain. They are at the right level of abstraction from the developers perspective and give good performance. In moving to system level solutions we should be able extend these abstracted C based models into the multi-processor domain. In this section we will look at some of the key careabouts for a *simulation framework* targeted to enable the extension of Instruction set simulators into the system level models.

3.1 Execution Control of simulated processors

Simulators are typically partitioned into two parts: a debugger that provides the user interfaces and a back-end that provides the simulation content. Debuggers need to provide multiple execution controls to the user: single step processors (*group step*), free run processors (*group run*), halt processors etc. In the multi-processor context, a whole bunch of semantics needs to be implemented for their support. Some of them include: *how are group run & step implemented - by serially running each processor or by running them all in a synchronous fashion ?, how is a run halted - is there a globally correct state at which it is safe to halt the system ?, how are breakpoints handled - if any one processor hits a breakpoint, should all processors be stopped ?, if so, at what simulation time - immediately or aligned to some state ?* etc.

An important distinction is that the debugger only provides user interfaces in support of the above controls. It does not enforce any semantics of these controls. The definition and implementation of the semantics, as well as addressing related concerns of accuracy & efficiency are implemented in the framework.

3.2 Scheduling & synchronization of simulated components

An important consideration is the granularity of the atomic simulation step: *does the simulation advance on a clock tick by clock tick basis ?, or does it advance in some larger chunks of time ?.* This affects the choice of synchronization and scheduling strategy.

For a clock accurate simulation, it may be necessary for the framework to schedule the models to run in such a way as to mimic true hardware clocking. In a multi-processor context a host of issues crop up: *are all processors running at the same clock speed ?, if not what are the relative clock speeds of the constituent processors ?, what is the strategy for advancing all the processors ?.*

This brings up the important question of *granularity* of synchronization: *how far ahead should any one processor be*

allowed to run before synchronizing its time with the rest of the system ? It is intuitively clear that there are tradeoffs among granularity, simulation performance and accuracy. While a coarse granularity approach favors higher performance, it impacts accuracy. To maintain accuracy, either a fine-grained (pessimistic) synchronization or a *rollback* mechanism is needed (refer [7]).

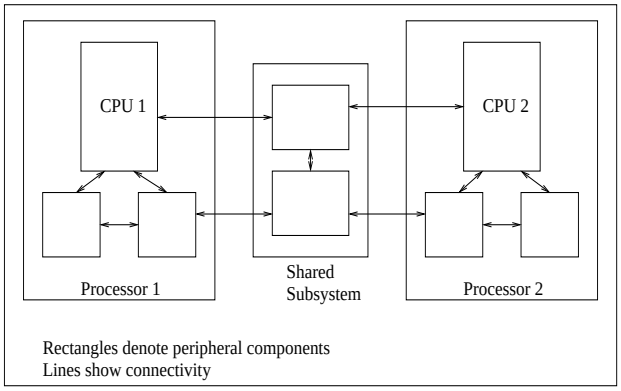

Figure 1: Multi-processor system block diagram

Further, the framework has to address the issue of clocking the non-processor shared components in the system. While the user interface concerns itself with providing "processor views" of the multi-processor system, the framework has to advance not only the processor simulators, but also the shared active components (such as shared DMAs, memories etc.) that can not be categorized as belonging to any single processor (see figure 1). Relevant questions include: *when should the shared active system be clocked ? should it be clocked only when all processors are run (group run) ? or should it be clocked even if any one processor is single-stepped ? should the shared system's clock be synchronized with the rest of the system ?*

3.3 Connectivity between components

Different components have to be interconnected via pins (as in the silicon) or at other levels of abstraction. Interaction between components occurs by means of pin-level transactions or, as is often the case in a software simulation, at a higher level of abstraction.

No one processor by itself is aware (nor should it be) of the system-wide connectivity. For instance, a processor is not aware of how its output pins are hooked up to the rest of the system. This calls for an entity that is *outside* the realm of any one processor to set up the desired connectivity. Thus, the framework must facilitate specification and implementation of this connectivity.

3.4 Communication between components

During the course of simulation, components read from or write to pins (or other abstractions). This data communication has to be implemented behind the scenes: no component is (should be) aware of inter-component connectivity. Ideally, components read from and write to abstract, well-defined interfaces which implement the physical communication channels. The communication system should also deal with the issue of order of processing of communication requests: this has impact on both efficiency of communication as well as on accuracy of simulation.

In the following sections we discuss the SimBridge solution and some analysis on its expected performance. We will see how SimBridge is designed to ensure speeds of multi-processor systems comparable to those of single processor instruction set simulators while ensuring the ability to maintain system wide cycle accuracy thus addressing the needs of application developers.

4. SIMULATION BRIDGE DESIGN

4.1 Overview

SimBridge is a C-based multi-processor system simulation framework. It is designed to be able to build over existing processor and subsystem instruction set simulators, and allow for capture of the shared system in an extendable way. This makes it suitable for capture of any multi-processor based system, using C-based simulation models of the different processor subsystems, and abstracted models of other modules in the system. By facilitating tight integration of the models into a single threaded executable, it provides very high performance.

Figure 2 captures the overall architecture of a SimBridge-based solution for multi-processor simulation. The SimBridge sits between the front end (FE) layers and the simulation models. It is designed to *bridge* all the processor models in the multi-processor simulation system. During construction of this system it sets up the connectivity between the different subsystems and enables communication at execution time. It intercepts & implements all execution control flow from the FE to the simulation models, supplying semantics, and synchronization. Below, a more detailed description of SimBridge design and how it addresses the above considerations is presented.

4.2 Building blocks

The SimBridge is made up of three modules: the execution manager, the event engine and the processor integration module (see figure 2). The following sections highlight the key functionalities provided by each of these modules.

4.2.1 Execution Manager

This module supports the user execution commands: run, step, group run & group step. As seen from figure 2, the SimBridge *straddles* all the back-end processor simulation models. This allows the execution manager to take complete control of the execution of the entire multi-processor system.

It provides the following semantics for the group execution commands (single processor run and step are treated as special cases of the group run & step, the group size being 1) :

- Group Run: Advance the processors constituting the group in *lock-step*: interleave the clocking of the processors as to ensure that a global time line is followed. The termination of the group run occurs whenever:

```
Any processor completes its run quantum
                or
Any processor hits a breakpoint
```

- Group Step: This is similar to group run except that the termination of this command occurs when all the processors finish one instruction each.

Section 4.3 details the functionality of this module.

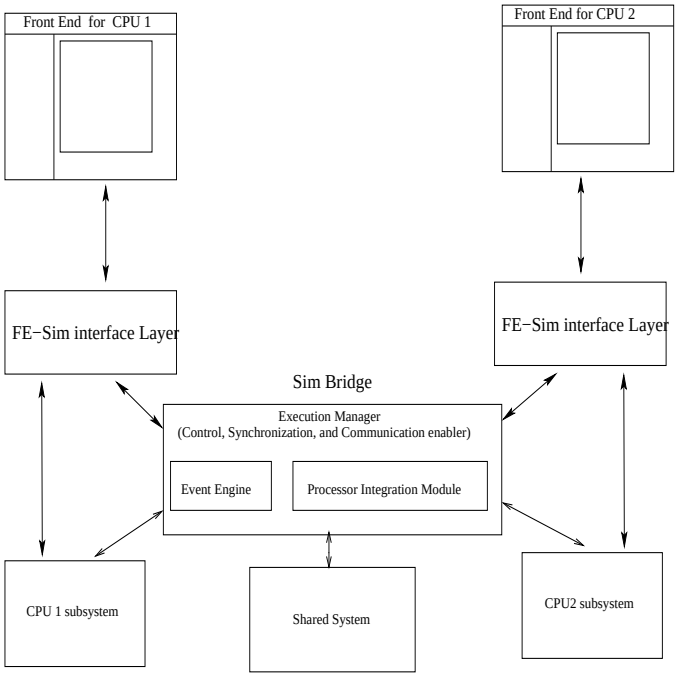

Figure 2: Multiprocessor Simulation with SimBridge

4.2.2 Event Engine

The event engine, based on the Core System Simulation Interface (refer [4]), supports event notification and data communication between components throughout the system. Components post events (along with event-specific data) into the engine. The engine, in turn, posts the event notifications to its listeners. A listener typically reads the associated data and may initiate further events of its own in response. Thus, no component ever needs to know which component(s) listen(s) to its event outputs. The event engine guarantees that the posted events are notified *in order* obeying the notion of *simulation delta*: all notifications are made after the current round of cycle activities complete. This provides the basis for highly accurate simulation.

Events and associated data pins are initially configured into the engine using the processor integration module discussed below. Listener components also register themselves into the engine, indicating which event(s) they would like to be notified on. The engine does not distinguish between inter- and intra-processor notifications. Thus, the global event engine provides full flexibility and visibility for components to "talk" to one another.

4.2.3 Processor Integration Module

While the SimBridge, by design, is generic, it requires input information on the connectivity of the system. This information comprises of events and pins that components use. The SimBridge captures this "netlisting" information via the processor integration module. This module provides the event engine with the list of events and pins that components use for communicating amongst one another. It may be noted that this module has to be implemented specifically for each multi-processor configuration.

4.3 Execution support & clock synchronization

This section describes the internals of the execution manager in greater detail and also shows how this design addresses the clock synchronization issues covered earlier.

Firstly, it may be recognized that by performing lock-stepped clocking of all the processors, the SimBridge provides highly accurate simulation. Since no processor is allowed to run ahead of the rest (pessimistic synchronization), it entirely avoids the issue of rollback. Further, the accurate execution model has no performance impact: *all that the SimBridge does is supply explicit clock triggers to the processor simulators instead of letting the processor simulators generate their own.*

4.3.1 Implementation of the execution manager

At start-up, it first acquires the relevant parameters: available processors and their clock configurations. It also acquries the clock configuration of the shared system. Thus set up, it implements the group run & step commands by running the clock generation algorithm:

```
termination_condition = false;
while ( termination_condition == false)
{
   move global time to nearest clock edge.
   generate appropriate clock trigger(s).
        ( this runs the simulation )
   compute next, nearest clock edge.
   update termination condition.
}
```

As may be observed, this algorithm is generic and clock-ratio-agnostic, scaling up to any number of processors and clock ratios.

Caveats:

1. Termination condition becomes true either if any processor completes its pre-assigned quantum of simulation (expressed in cycles or instructions) or if any processor hits a breakpoint.

2. Since not all the processors would finish their quanta together, it becomes necessary to either immediately stop all of them (aligned to a global time) or to let them run upto their instruction completions.

3. In the implementation, the processor layers use callbacks into the execution manager to notify instruction step completion & breakpoint hit conditions.

4. The execution manager clocks the shared subsystem along with the processor layers to ensure accurate simulation of the entire system. Even when a single processor is run or stepped, the shared subsystem is clocked (even though it breaks the concept of a global time). The intuition behind this choice is that:

   - when a processor is run, it typically needs to interact with the shared system
   - shared components are of a request-response nature & not absolute clock value based.
   - user is anyway not keeping the processors' execution in lock-step.

- If the system time was held close to the individual processor time, then it would lead to starvation of processor requests.

4.4 Customizing the SimBridge

The SimBridge is designed to be a generic framework, capable of working with any number of processors, different clock ratios and so on. With additional effort it is possible to derive specific versions of the SimBridge for better performance. For example, for commonly encountered configurations (such as a two processor system with identical clock speeds), the efficiency of the SimBridge can be boosted by specializing its implementation to take advantage of the settings. This also enables the SimBridge to be deployed in uni-processor simulations. The execution manager facilitates this capability by allowing its functions to be overridden in custom ways.

5. CASE STUDY & PERFORMANCE RESULTS

The evaluation of the SimBridge was conducted on two test setups: A. an experimental test bench that mimicked processor models and B. the TI TMS320C54x based simulator.

5.1 Experimental test setup

The experimental setup consists of dummy processor models. These models do not model any architecture but simply get clocked by the SimBridge. This setup was intended to serve functional validation and exploration of the SimBridge.

Since the SimBridge has two main parameters, namely number of processors and clock ratios, following two performance characteristics were measured: one, simulation time versus the number of processors, and two, simulation time for changing clock ratios.

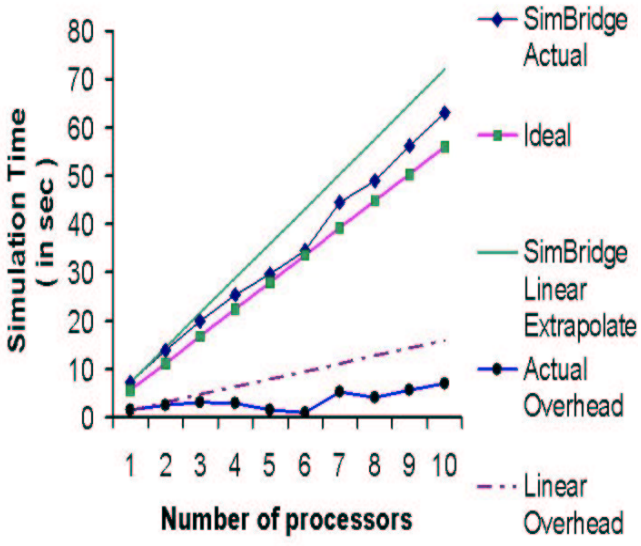

Figure 3: Simulation Time vs Number of Processors

Figure 3 shows the variation of simulation time with increase in number of processors. All processors were clocked at the same speed and the simulation was run for 5M cycles. The "ideal" line was obtained by simply scaling the time for a single processor simulation in a non-SimBridge setup by the number of processors. SimBridge linear extrapolate was obtained by scaling the time for a single processor simulation in the SimBridge setup by the number of processors. Actual overhead is the difference of the SimBridge actual and the ideal. Linear overhead is the single processor overhead scaled up. It is clear that the overhead of SimBridge does not multiply with the number of processors.

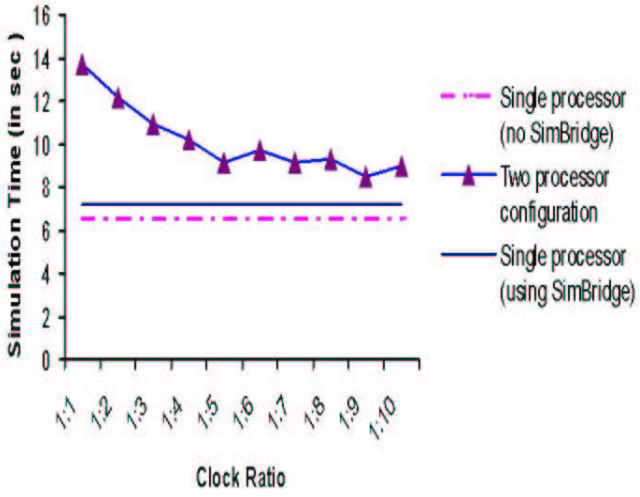

Figure 4: Simulation Time for Changing Clock Ratios

Figure 4 shows simulation times for changing clock ratios in a two-processor configuration. The simulation was run for 5M cycles of the fastest clock. The curve shows that the simulation time *falls* as clock ratio is increased. This is because, as the clock ratio becomes more and more skewed, fewer and fewer ticks of the slower clocks are issued, resulting in lesser synchronization and simulation. Thus, as the clock ratio becomes very skewed, the dominating cost is that of clocking only the fastest processor. Due to implementation overheads, this asymptotic cost is higher than the ideal uniprocessor simulation cost.

While SimBridge has been optimized for efficient scheduling and synchronization, owing to its generic nature it suffers performance losses. To further improve performance, a handful of customizations were done (exploiting knowledge of clock ratio, and number of processors). The performance of the customized system was compared against an ideal, home-grown solution that we refer to as a "stand-alone" setup. The stand-alone setup essentially hard programmed the clocking sequence. Table 1 gives an indicative summary. It shows that the overheads can be cut down appreciably by exploiting domain-specific knowledge.

5.2 The TMS320C54x-based simulation setup

The TMS320C54x-based simulator comprised cycle accurate C models of the TMS320C54x (refer [11]) CPU together with on-chip, off-chip memories, timers and serial ports. To measure real-life simulation performance, the simulator was instantiated upto 3 times & integrated into SimBridge to

Table 1: Performance comparisons of generic, customized, and stand-alone setups (all times in seconds)

clock ratio	Stand-alone setup	Customized SimBridge [% overhead]	Generic SimBridge [% overhead]
1:1	11.2	12.15 [8.4]	13.80 [23.2]
1:2	8.7	9.75 [12.0]	12.16 [39.7]
1:3	7.8	9.05 [15.8]	10.91 [24.4]
1:4	7.6	8.78 [15.2]	10.20 [33.8]
1:5	7.5	8.75 [16.6]	9.16 [22.1]

Table 2: TMS320C54x simulation performance

Configuration	Time (seconds)	Cycles simulated	Speed (kcps)
Single processor (without SimBridge)	14.4	8195740	570
Single processor (with SimBridge)	20.3	8195740	404
Two processors	32.4	16391480	506
Threee processors	45.6	24587220	539

mimick a true multi-processor scenario. Data was collected by running identical copies of the matrix multiplication application on all the instantiated models.

Table 2 shows that the SimBridge delivers good multi-processor simulation performance (of the order of a few hundred *kcps* - kilo cycles per second). While the single processor scenario shows some overheads w.r.t a native simulation setup, the two- and three-processor versions show that the SimBridge integrates multiple processors in a very efficient manner, scaling up with very little synchronization overheads. The data demonstrates that it is feasible to perform real application simulation on multi-processor architecture simulators by means of tightly coupled, accurate C models.

6. CONCLUSIONS

In this paper, we have seen that there is a gap in the system level simulation solutions available today, namely, meeting the needs of SW application developers. As instruction set simulators have been very successful in addressing the developers' needs in the uni-processor space, we looked at some of the key careabouts in being able to extend these solutions into the multi-processor domain. The SimBridge framework is specifically designed to address these careabouts, and do so without adding too much of a simulation overhead. As shown in Section 5, it achieves performance comparable to that of stand-alone uni-processor instruction set simulators while still being able to provide cycle level accuracy.

Going forward, we need to see how SimBridge can be designed differently to be able to further reduce the overheads and improve performance. Some of the considerations here would be increasing the granularity of synchronization, considering non-interpretive simulation and distributed techniques. Also, given the trend towards building applications on top of minimal operating systems running on DSPs, it may be profitable to switch between accurate and functional simulation styles dynamically.

Also, it is planned to make the SimBridge configurable enough to let the end user put together a multi-processor simulation system and run it. This involves reading up the configuration information from a front-end or a database and generating the corresponding Processor Integration Module. Another extension is to be able to integrate simulators from different vendors via the SimBridge. While it is believed that these extensions fit into the current framework, these capabilities are yet to be demonstrated.

7. REFERENCES

[1] Cadence, "*Cadence Virtual Component Codesign Environemt*", www.cadence.com/datasheets/vcc_environment.html

[2] Dwight Sunada, David Glasco and Michael Flynn, "*ABSS v2.0: a SPARC Simulator*," Technical Report CSL-TR-98-755, Comp. Systems Laboratory, Stanford Univ., April 1998.

[3] Howell F. W., Williams R., Ibbett R. N., "*Hierarchical Architecture Design and Simulation Environment*," Dept. of Computer Science, Univ. of Edinburgh.

[4] Ish Kumar Dham, "*Towards extending cycle accurate CPU Instruction Set Simulator to Chip Level Simulator - a case study*,"TI India Technical Conference, May 1998.

[5] Jack E. Veenstra and Robert J. Fowler, "*MINT Tutorial and User Manual*," Technical Report TR-452, The Univ. of Rochester, Comp. Sci. Dept., June 1993.

[6] Ken Hines,"*Pia: A Framework for Embedded System Co-simulation with Dynamic Communication Support*," Univ. of Washington, 1996

[7] Lamport, L., "*Time, Clocks, and the Ordering of Events in a Distributed System*,"Comm. of the ACM, Vol. 21, no. 7, July, 1978.

[8] Luis Barriga and Ayani R., "*Towards Efficient Simulation of Parallel Architectures*," Technical Report TRITA-IT R 94:07, Dept of Teleinformatics, Royal Institute of Technology, Stockholm, Sweden, March 1994.

[9] Mentor, "*Seamless Hardware/Software Co-Verfication*", www.mentor.com/seamless/datasheets/

[10] Shubhendu S. Mukherjee et. al., "*Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator*," Workshop on Performance Analysis and Its Impact on Design (PAID), June 1997.

[11] Texas Instruments, "*TMS320C54X, TMS320LC54X, TMS320VC54X Fixed-point digital signal processors*", Literature Number: SPRS039C, February, 1996.

[12] Vijay S. Pai et. al.,"*RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors*," Dept. of Electrical and Comp. Engg., Rice University, 1997