Cycle and Phase Accurate DSP Modeling and Integration for HW/SW Co-Verification

Lisa Guerra[*], Joachim Fitzner[⊥], Dipankar Talukdar[*], Chris Schläger[⊥], Bassam Tabbara[+], Vojin Zivojnovic[⊥]

[*]Conexant Systems
4311 Jamboree Rd., MC 510E-602,
Newport Beach, CA 92660, USA
[lisa.guerra,dipankar.talukdar]@conexant.com

[⊥]AXYS GmbH,
Kaiserstr. 100
52134 Herzogenrath, Germany
[jf, cs, vz]@axys.de

[+]UC Berkeley EECS Dept.
211-150 Cory Hall
Berkeley, CA 94720, USA
tbassam@eecs.berkeley.edu

ABSTRACT

We present our practical experience in the modeling and integration of cycle/phase-accurate instruction set architecture (ISA) models of digital signal processors (DSPs) with other hardware and software components. A common approach to the modeling of processors for HW/SW co-verification relies on instruction-accurate ISA models combined (i.e. wrapped) with the bus interface models (BIM) that generate the clock/phase-accurate timing at the component's interface pins. However, for DSPs and new microprocessors with complex architectural features this approach is from our perspective not acceptable. The additional extensive modeling of the pipeline and other architectural details in the BIM would force us to develop two detailed processor models with a complex BIM API between them. We therefore propose an alternative approach in which the processor ISAs themselves are modeled in a full cycle/phase-accurate fashion. The bus interface model is then reduced to just modeling the connection to the pins. Our models have been integrated into a number of cycle-based and event-driven system simulation environments. We present one such experience in incorporating these models into a VHDL environment. The accuracy has been verified cycle-by-cycle against the gate/RTL level models. Multi-processor debugging and observability into the precise cycle-accurate processor state is provided. The use of co-verification models in place of the RTL resulted in system speedups up to 10 times, with the cycle-accurate ISA models themselves reaching performances of up to 123K cycles/sec.

1. INTRODUCTION

The trend of developing increasingly complex systems under shrinking time-to-market conditions continues. Typical current-day, single-chip electronic system implementations include a mix of a number of microcontroller, DSP, shared memory, dedicated logic, and interconnect components. With increased hardware and software design complexity, greater use of components from various design teams and third parties, and rising number of gates per pin, verification has emerged as a critical bottleneck surrounding the system-on-chip design paradigm.

In addressing this verification bottleneck, concurrent simulation of full systems-on-chip with interacting hardware and software is now widely recognized as an important and viable verification approach. Co-simulation/verification of hardware and software has been proposed for a number of environments such as [8] [3][12][5][10][13][4][15][2] and is also offered commercially by companies such as Mentor Graphics [9], Synopsys [14] and others.

In this work our interest is system co-simulation used during cycle-accurate system verification, after the system architecture has been selected and hardware and software models have been developed. Co-simulation enables the verification of actual processor application code running on ISA processor models in conjunction with accurate models of the remainder of the hardware system.

In general, for cycle-accurate processor modeling, there have been efforts which develop C/C^{++}-based cycle-accurate ISA models to verify against the RTL models (e.g., for the UltraSparc [11] and CRISP 32-bit RISC microprocessors [6]). On the other hand, in the co-verification domain, the most common approach for cycle/phase-accurate modeling uses instruction-based ISA models extended by simple interface models (commonly referred to as bus interface models (BIM) or wrappers) which interpret external interaction events, and thereby generate cycle/phase-accurate simulation traces at the component's pins. This method is used, for example, in the ARM7 processor model [7] integrated into commercial HDL-based environments such as [14] and [9].

For architectures with a simple timing, this can provide a satisfactory level of accuracy under most circumstances. Recent DSP architectures and a growing number of new microcontrollers, however, have deep pipelines with greater memory access bandwidth, memory accesses distributed over multiple stages, and complex stalling logic. For these types of processors, bus interface models become very complex, and must include modeling of pipeline and architecture information in order to recover the necessary timing. Our contribution described here offers an alternative to detailed interface models; we model the processor ISA itself in a full cycle-accurate fashion. Complete modeling of the deep, partly-protected pipeline with all the interrupt, wait state, and stall effects is performed. The wrapper is thus reduced to a thin layer describing the interconnection to pins and the API between the ISA model and the BIM is simple.

Our solution provides full cycle and phase-accuracy for the proposed implementation under real operating conditions for system-on-chip designs. The accuracy is verified cycle-by-cycle against the gate/RTL-level model. For the processor models, the simulation slowdown that one might expect by the increased accuracy does not occur. The SuperSim compiled ISA simulator of the processor [15][16] successfully compensates for slowdown due to increased accuracy.

In this paper we also describe the integration of these models with other hardware and software components. A fully cycle-accurate system simulation has been achieved, incorporating the C-based ISA DSP models and mixed-level VHDL models. The integration supports full multi-processor debugging, visibility, and controllability of instantiable, configurable models of the DSP processor with low development effort. Experimental results demonstrate processor ISA model speeds of up to 123K cycles/second and co-verification model speeds of up to 70K cycles/second. For a complete RTL VHDL-based system-on-chip simulation, speedups of a factor of 10 over conventional RTL simulations were attained.

The paper is organized as follows. Section 2 presents our DSP processor co-verification model. An integration of these models into a VHDL simulation environment is presented in Section 3. Experimental results are discussed in Section 4 and finally conclusions are presented in Section 5.

2. PROCESSOR CO-VERIFICATION MODEL

This section presents the design options for a processor co-verification model and discusses the reasons that lead to our choice of a cycle-accurate over an instruction-based (bit-accurate) ISA core model. Model requirements, possible model structures, and aspects of model reuse are also presented.

The target for our co-verification model is a state-of-the-art, Conexant in-house, low-power DSP. It is a pipelined, two-way SIMD (single -instruction-multiple-data) machine with a Harvard memory architecture capable of handling up to three data memory accesses per cycle. It has a fixed length 32-bit instruction set and supports 16 and 32-bit operations. The DSP is being used in several systems-on-chip embedded in communication and consumer products developed at Conexant Systems.

2.1 Model Requirements

Models that will be used within system simulation have to be designed to include the following features:

Modeling accuracy: In terms of functionality and processor state representation, bit-accuracy [12] is necessary in order to reflect bit-length effects and obtain correct results. In terms of timing, the accuracy requirements depend on the processor's mechanisms to connect to the system. For the case of the systems and verification stage considered here, cycle-accurate timing is critical for verifying the performance impact of certain components and architectural features such as memory wait states, variable interrupt latency, or non-interruptible loops. Since the targeted processor as characterized above has interactions with memory, memory-mapped I/O, and I/O on a phase basis, modeling the related components' interaction with phase-accuracy is inevitable.

Debug capability: The internal state of the model needs to be observable. For processors, this includes the ISA registers, memory, and pipeline. It is desirable to observe the correct state at every clock cycle, not just at instruction boundaries, as this greatly enhances processor understanding and programming. The execution control should comprise all standard debugging features like stepping, and breakpoint handling for a multiple processor system.

High simulation speed, and reliability: The performance of the overall system is drastically influenced by the speed of its components. Therefore, each component should be optimized for speed. This is not only beneficial for run-time simulation but in addition enables thorough model verification against its reference. This helps ensure high reliability not only of the components but also of the system.

IP reuse: The models should be generic with respect to the number of instances of processors, memory sizes, memory maps, and interface protocols. Models must have a well-defined API, and provide flexibility in their use and execution since different design tools may be used by different teams as the design evolves. One requirement in this class is that the model has to be slavable, i.e. capable of being activated by a master scheduler. While it may appear a simple requirement, a number of models that we have encountered are not slavable to another simulation master and thus integration alternatives are less flexible.

2.2 Model Structure

Most of the processor models used in existing HW/SW co-verification environments are instruction-accurate, providing precise processor state modeling only at the instruction boundaries. A simulation step in such a model atomically simulates one instruction's behavior through all stages of the pipeline. Each executed instruction can trigger one or more software events such as reads, writes, and signal assignments.

The co-verification model consists of the ISA simulator and a Bus Interface Model, BIM (Figure 1a). The interface model itself is also partitioned into two parts, the bus pin model and the bus cycle scheduler. The latter collects the information about the software events, identifies the corresponding hardware events, and schedules them in proper order for each clock cycle. It schedules these transactions along with already pending transactions using cycle count information from the ISA and architectural information of the processor to properly take into account external inputs, such as interrupts and memory wait states. The bus pin model provides the interconnection to the simulation environment and the rest of the system. For example, in an HDL environment, it provides the pin delay back-annotation and connects to the HDL over the FLI/PLI interfaces. For processor architectures with relatively simple cycle-based timing, like some standard microcontrollers and embedded RISC architectures, the use of instruction-accurate models might be sufficient. However, even for these architectures this rather low level of accuracy can cause modeling problems especially for whole systems involving multiple cores, interrupts, and heterogeneous memory banks. Despite this fact, the community of model users and providers have had several strong arguments to develop and use this type of model in the past. The main ones are ISA model availability, high simulation speed, high degree of IP protection, and relatively simple ISA model development. The best known model from this class is the ARM7TDMI HW/SW co-verification model developed by ARM, Ltd. [7].

In an alternative approach, a cycle-accurate ISA model (Figure 1b) is used. The increase of the timing accuracy within the core model reduces the complexity of the bus cycle scheduler since the only remaining task is to translate ordered cycle based events into phase based hardware events. The overlapped execution of

instructions and other pipeline effects are already handled by the core model.

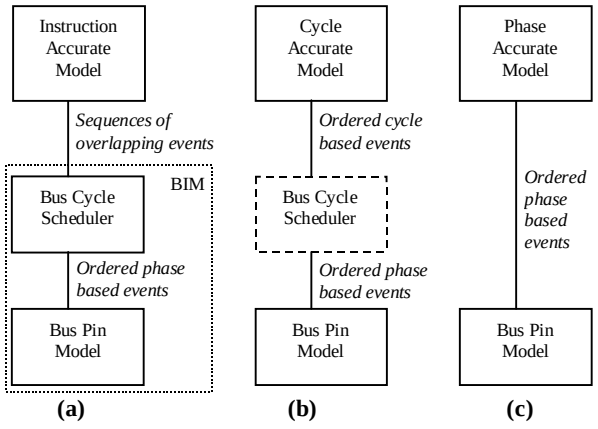

Figure 1: Possible Model Structures

Finally, in a third approach a phase-accurate ISA model is used (Figure 1c). This models the bit-true state of the processor at each phase transition. The interface model in this case does not require any kind of bus cycle scheduler because the software events of each phase can be directly translated into the corresponding hardware events.

The tradeoff among these approaches lies in the complexity of the ISA model versus the complexity of the interface model, as well as the complexity of the API between the ISA and the interface model. In the following section, we provide arguments as to why for DSPs the development and integration of cycle-accurate ISA models is unavoidable.

2.3 Arguments for a Cycle-Accurate Core Model

Considering the existing commercial HW/SW co-verification environments, it can be seen that models of DSP architectures are rarely provided and if they are, they show serious HW/SW modeling deficiencies if modeled using the instruction-accurate approach. The question that arises is why the integration of a DSP causes these problems.

Main characteristics of today's DSPs are a high memory bandwidth, short interrupt latencies, and deep pipelines. All of these features have an impact on the ability of the processor to communicate with other components and therefore are critical for the design of a co-verification model.

Pipelining is a key implementation technique used to increase processor performance by overlapping the execution of multiple instructions. One problem with increased pipelining is that it splits the execution of an instruction into smaller pieces, and thus provides the potential for a number of interactions between the processor and other components across multiple cycles. For a load-store-architecture the memory read and write always occur in the same stage but for an architecture capable of memory-to-memory operations those accesses are located in two different stages. Since the duration of one cycle tends to shorten with longer pipelines, memory accesses can not be limited to one clock cycle and need to be spread over stage boundaries to allow the usage of standard memory.

Consider a hypothetical 5-stage pipeline with instruction fetch (IF), instruction decode (ID), operand fetch (OF), execute (EX), and write back (WB) stages. The execution of an ADD instruction in this pipeline generates read accesses to both program and data memory (Harvard architecture). Consider the data memory access. In the decode stage, the ADD's operand memory address is available and issued to an external, cycle-accurate shared memory component. In the next cycle, when the ADD is in the operand fetch stage, the read data is returned to the processor (Figure 2).

Cycle	1	2	3
Stage	IF	ID	OF
ADD	Prog. Addr. (out)	Prog. Value (in) Data Addr. (out)	Data value (in)

Figure 2: Multi-Stage Memory Access

To model this behavior with an instruction based core model and a bus cycle scheduler it is not only necessary to schedule the address bus but also to read the data back from the data bus in the next cycle. This value has to be propagated to the ISA model which has to remain idle until this value becomes available in order to compute the sum and update the flag register. The interaction between the ISA model and the bus scheduler becomes even more complicated, if two read accesses are scheduled by the same instruction. Both access requests have to be sent to the bus model and the ISA must wait for two responses from the system. The interface between the core processor model and the bus model needs to be able to take multiple reads into account plus additional side effects like variable length wait states for the response. The high memory bandwidth in DSPs is achieved by either using multiple memory banks or by using memory modules with multiple ports. For example, up to two reads and one write can be caused by one instruction. As a consequence, two overlapping instructions in the pipeline can result in three data memory accesses scheduled for a single cycle (two reads from one instruction and one write from the other). All of this results in an enormous number of software events, all of which have certain interdependencies.

Some architectures allow a fetched instruction to be subsequently killed later in the pipeline in order to handle issues such as mispredicted branches. An instruction-based simulator usually does not model the fetch of these killed instructions, since without pipelining, the branch condition from the previous instruction is always determined. Data memory accesses initiated by killed instructions are sometimes visible to the system. Thus while they do not have an impact on the internal state of the processor, they still might impact the system.

Short interrupt latencies are a strong requirement in the DSP domain in order to allow applications to meet their real time constraints. Some processors have variable interrupt latency since they are not able to accept interrupts under every condition. Hardware loops, stalls, memory wait states, program flow instructions, and branch prediction are examples that might cause a delay in the servicing of interrupts. Consider the example shown in Figure 3 for a different 5-stage pipeline. An interrupt is received in cycle one. At this time a conditional branch instruction with one delay slot is in IF. The processor delays the interrupt by a cycle to ensure the execution of the delay slot instruction prior to the interrupt. The branch target address is thus used as the return address after the interrupt routine has been completed. It would be far more complicated to save the branch target address and then resume the program execution with the delay slot instruction. The BR's delay slot instruction is thus fetched before the interrupt is accepted, and in cycle three the interrupt jump

target instruction is fetched. The sources of variable interrupt latencies are not orthogonal and have certain processor specific dependencies which are rather difficult to model in an instruction-based ISA providing a cycle counter.

	Clock cycle				
	1	2	3	4	5
IF	BR	DelaySlot	ISR		
ID		BR	DelaySlot	ISR	
OF			BR	DelaySlot	ISR
EX				BR	DelaySlot
WB					BR

interrupt — interrupt pending — interrupt accepted

Figure 3: Variable Latency Interrupt Example

The interdependencies between instructions causing pipeline stalls and instructions to be killed affect the timing and the functionality of the bus interface significantly. As a consequence, either the bus cycle scheduler has to be supplied with more information about these dependencies or the timing accuracy of the core model has to be increased, providing ordered cycle or even phase events. Making the bus cycle scheduler more aware of the instruction dependencies has the drawback that it would result in building a second pipeline within the bus cycle scheduler and the decentralization of architectural information.

```
if (phase1){
       Get phase1 inputs, write to ISA model
       ExecutePhase1()
       Drive phase 1 outputs.
}
if (phase2) {
       Get phase 2 inputs, write to ISA model
       ExecutePhase2()
       UpdateState()
       Drive phase 2 outputs.
}
```

Figure 4: Phase Accurate Wrapper

Since our target processor has features such as high memory bandwidth, memory wait states, stalls, variable interrupt latency, and branch prediction, we therefore developed a cycle-accurate ISA core model that provides ordered cycle events to the bus scheduler. Since the interaction between the processor and the other system components is required on phase boundaries, we in some cases allow the ISA to be advanced in phases (Figure 4). However, only those parts of the processor state that are visible at the pins get updated on the phase edge; the complete processor state is still updated only at the cycle boundary. This enables 100% phase accuracy of the co-verification model.

2.4 Model Reuse and Configuration

This section describes a number of elements that enable the reuse of our model. First, the model provides a well-defined set of interface functions and modes. Supported interface functions include initialization, reset, termination, phase execution, state access such as reading and writing registers and memories, setting and resetting of interrupts and pipeline stalls for memory wait states. The model also has interfaces from which the system integrator can customize multi-stage memory and I/O reads (split reads), and memory and I/O writes. Modes include master mode in which the simulation runs as an independent process and slave mode in which it can be called from an external simulation master.

These interfaces are critical to enable reuse of the model within a number of environments. For example, for co-verification, these processor models have been integrated directly into an HDL environment as shown in Figure 5a. Also, they have been integrated into a cycle-accurate C^{++} environment to form a sub-system "island" which itself has been integrated into a HDL environment (Figure 5b).

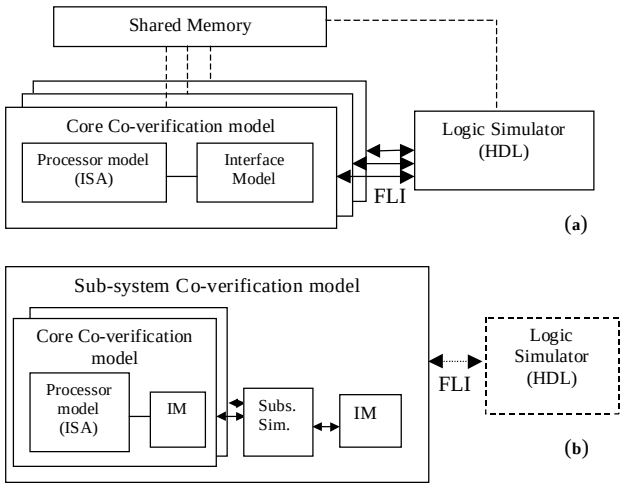

Figure 5: Reusing the Model in Multiple Environments

3. INTEGRATION: HDL ENVIRONMENT

This section presents our experience in integrating the before-mentioned models into a VHDL environment. Since the final verification of the design will be done on a HDL gate-level netlist, a mixed-level VHDL-based environment provides a uniform verification platform that supports model refinement from the abstract behavioral description, to the cycle-accurate, then finally the gate level. Another benefit is model availability and reliability since a large amount of legacy and third-party hardware components are modeled primarily in HDL. At this point, integration is done directly into a commercial HDL simulator but can also be done for standard co-verification tools.

In addition to the DSP, other modeled system components include dedicated and shared memories, programmable/hardwired accelerators, DMA controllers, and processor interface busses.

3.1 Overview

In the VHDL environment, the VHDL simulator acts as the central event scheduler. We execute the processor model by a call through the C Foreign Language Interface (FLI) of the VHDL simulator. The phase-level synchronization with the rest of the system is maintained by making the call on every transition of the clock. Before the call, inputs are read, and after the call, outputs are driven. Processor co-verification models are compiled into shared objects and directly integrated without the need for inter-process communication (IPC).

3.2 Memory Modeling

Processor memory was modeled in the following three ways. All memory falling only within the DSP's address space (not shared with any other components) is modeled locally within the ISA model, in C. Using the SuperSim compiled simulation approach,

all program memory is pre-compiled into the model. Thus, no program memory interaction takes place at run time, although such interactions could be made visible within the system.

All other memory accesses (shared with possibly multiple processors, I/O, accelerators and DMA controllers), are modeled using either standard VHDL transactions or are abstracted using VHDL'93 *shared variables*. Abstraction is achieved without sacrificing cycle-accuracy even for memories shared between multiple processors and components. It is used whenever verification of the interaction is not needed. The shared variables are aliased in the core's VHDL wrapper and freely accessible through the C FLI of the commercial VHDL simulator. This mechanism provides for efficient processor-based shared memory access by reducing the number of VHDL transactions. In the system-level memory components, the shared memory is used as the storage element instead of the regular VHDL variables. All interactions of the memories with other components such as a DMA controller or accelerator take place through VHDL signals and the existing memory interface protocols. The use of shared variables results in only one copy of the storage resources to represent multiple-owner memory components, thus obviating the need for coherency control.

3.3 Reuse

In addition to the reuse properties described in Section 2.4, reuse in the VHDL integration was enabled by keeping all instance and memory configuration information separate from the model itself. Models are re-instantiable and configuration information such as instance identifiers are passed to the models using VHDL *Generics*. Another manner in which reuse is enabled is in the actual definition of the model pin boundaries. With an in-house processor, boundaries are flexible and thus the core was partitioned to keep system-dependent parts of the DSP, such as the multi-bus protocol interface, in VHDL. This left a generic memory access interface to the core.

3.4 Debugging

Debugging includes visibility of state and source level code running on the processor cores as well as controllability of the simulation. By using the cycle-accurate ISA models, the programmer is able to see register values committed in the precise cycle and to watch the instructions currently in the different pipeline stages. This aids greatly in understanding the impacts of interrupts, stalls, wait states and mispredicted branches and thus in more efficient processor programming. Debug of co-verification models within the system simulation is provided with the same debug interface provided for the stand-alone ISA model. In both cases, debugging is achieved using a standard C-language debugger (e.g. dbx or gdb). The debugger and the SuperSim simulator are adapted to execute the C code of the simulation and at the same time display target assembly instructions. This debug approach is enabled by a certain use of compiled simulation [15][16] (also shown in Figure 6a). While for the stand-alone ISA model, debugging is done directly on the compiled model (<file>.exe in Figure 6a), for the VHDL-based system simulation, debugging is done on the VHDL simulator which includes the compiled co-verification model as a shared object (Figure 6b). Whereas the proprietary VHDL simulator code is not visible to the debugger, the compiled model code is. In this way, breakpoints can be placed within the co-verification model to step, break, and perform all other standard software debug operations. Even multiple instances of a processor can be debugged in this manner at the same time, with compound multi-processor breakpoint conditions. A key advantage of this approach is that very little code development/modification is needed to support debug. Also, this is achieved without the run-time and development overhead of IPC. This approach of attaching a standard C-language debugger can be used for debug of any C-based processor or ASIC models integrated within the HDL simulation. It was found quite useful for debugging of C FLI code in general.

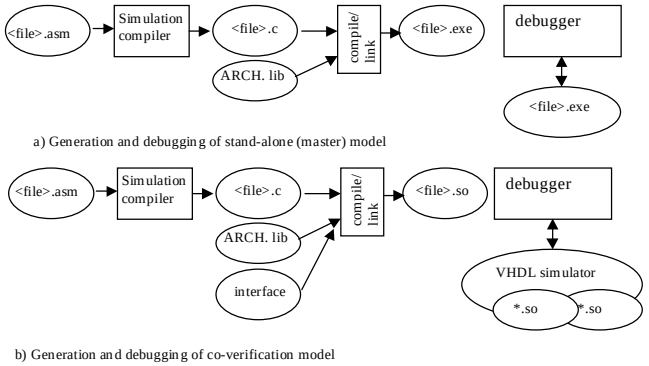

Figure 6: Debugging Approach

4. EXPERIENCE AND RESULTS

4.1 Accuracy

The described model has been implemented and successfully verified against the phase-accurate reference RTL VHDL model. The verification process consisted of running millions of random instructions as well as hundreds of focus vectors on the processor model to test the core functionality. A focus vector is a complex sequence of instructions designed to test specific sub-sections of a design, such as the memory interface. Several hundred system-level focus vectors consisting of sets of programs and HDL testbenches were used to test the system level interactions. An automated trace mechanism was used to compare the results to that of a VHDL gate-level netlist. The desired visibility and controllability have also been achieved. Figure 7 shows a screen shot of the user interface. Mentor Graphic's MTI ModelSim simulator is used for logic simulation and hardware debug. The SuperSim simulator with customized Solaris Workshop debugger is used for software simulation and debug.

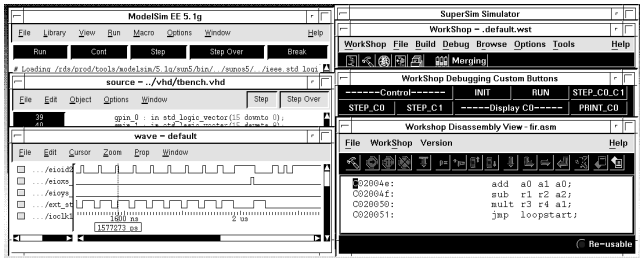

Figure 7: Screen Shot Showing Waveform and Software Debug

4.2 Speed

A number of experiments have been run to quantify the model simulation speeds. All results were collected on a 296 MHz Sun Ultra2 with 2 Gigabytes of RAM. Experiments were performed on a set of examples with varying core computational complexity and

IO activity.

The first set of results (Table 1) shows the simulation times for the processor core. RTL HDL is the phase-accurate behavioral RTL VHDL of the core. MaxSim is the co-verification model presented in this paper, placed in a VHDL wrapper. SuperSim is the stand-alone cycle-accurate ISA C-model. The simulation was performed on three different programs. Simple1 is an example of low computational complexity, basically an idle (NOOP) loop. Complex1 is of high computational complexity involving arithmetic operations and significant shared memory activity. IO1 is an I/O-intensive application where the core initiates larger number of interactions with the peripheral and external memory components. Compared to the RTL model, the MaxSim model provides a speedup of 600, 875, and 158 on these examples, respectively. What is noteworthy is that the speedup is achieved with a phase accurate model of the core which is relatively more complex than a traditional instruction set accurate model. The examples also highlight the effect of the interface activity. In the IO1 example, the external pin activities cause numerous events in the HDL simulator domain, which slow down the simulation. Reference [1] presents the effect of parameters such as I/O activity and number of cores on the simulation time. The trends presented in that paper were observed in our system also and can be used to extrapolate the results to other examples.

Examples	Speed (cycles per sec) / Speedup		
	RTL HDL	MaxSim	SuperSim**
Simple1	130 / 1	78K / 600	123K / 946
Complex1	80 / 1	70K / 875	94K / 1175
IO1	90 / 1	14.3K / 158	110K / 1222
** Developed by AXYS [16].			

Table 1: Processor Core Only

Table 2 shows simulation times for a system containing four instances of the processor core, shared memory, and peripheral components. The MaxSim column represents the system with the RTL cores replaced by the co-verification models, and the VHDL'93 shared memory implementation. Roughly an order of magnitude speedup was obtained by replacing processor RTL models with the C-models. This resulted in simulation speeds on the order of hundreds-of-cycles/second (200-300). The presence of RTL peripheral components in the system reduced the speedup gain that was achieved with only the core.

Examples	Speed (cycles/sec) / Speedup	
	RTL HDL	MaxSim
Simple	37 / 1	300 / 8
Complex1	22 / 1	300 / 13
IO1	26 / 1	208 / 8

Table 2: Four-Core System-On-Chip

On a similar comparison with a VHDL gate-level netlist of the system, an order of magnitude speedup was attained (speedups of 12-28). The added speedup was attained since the relative complexity between MaxSim and gate-level netlist models is greater than that between MaxSim and RTL models.

5. CONCLUSION AND FUTURE WORK

Simulation is a major issue and a critical approach for debugging complex embedded system-on-chip designs. This paper has presented our experience in the modeling of cycle and phase accurate DSPs for HW/SW co-verification, including our decision to use a cycle-accurate ISA. Despite the timing complexity of the target architectures, all the effects of interrupts, memory wait states, and deep, partly-protected pipelines have been modeled and verified. While integration into a general VHDL framework has allowed us to easily incorporate existing HDL hardware IPs, parallel work at Conexant Systems has also integrated these models into a C/C^{++}-based cycle-accurate simulation environment for earlier system co-verification.

6. ACKNOWLEDGEMENTS

Thanks to Ulrich Bortfeld from Conexant and Davorin Mista from AXYS for their feedback and contributions to this project.

7. REFERENCES

[1] T. Albrecht, J. Notbauer, S. Rohringer, "HW/SW CoVerification Performance Estimation & Benchmark for a 24 Embedded RISC Core Design," *DAC*, pp. 808-811, 1998.

[2] F. Balarin, M. Chiodo, P. Guisto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, B. Tabbara, "Hardware-Software Co-Design of Embedded Systems: The POLIS Approach," *Kluwer Academic Publishers*, 1997.

[3] D. Becker, R. Singh, S. Tell, "An engineering environment for hardware/software co-simulation," *DAC*, pp. 129-134, 1992.

[4] W.T. Chang, A. Kalavade, E. Lee, "Effective heterogenous design and co-simulation," *NATO Advanced Study Institute Workshop on Hardware/software codesign*, June 1995.

[5] S. Coumeri, D. Thomas, "A simulation environment for hardware-software codesign," *ICCD*, pp. 58-63, 1995.

[6] D. Ditzel, A. Berenbaum, "Using CAD tools in the design of CRISP," *IEEE Design & Test*, 21-31, June 1987.

[7] R. Earnshaw, L. Smith, K. Welton, " Challenges in cross-development," *IEEE Micro*, pp. 28-36, July/Aug. 1997.

[8] R. Gupta, C. Coelho, G. De Micheli, "Synthesis and simulation of digital systems containing interacting hardware and software components." *DAC*, pp. 225-230, 1992.

[9] R. Klein, "Miami: A hardware software co-simulation environment," *IEEE Int'l workshop on rapid system Prototyping*, pp. 173-77, 1996

[10] B. Lin, K. Van Rompaey, S. Vercauteren, D. Verkest, I. Bolsens, H. De Man, "Designing single chip systems," *ASIC* , 1996.

[11] G. Maturana, J. Ball, J. Gee, A. Iyer, J. M. O'Connor, "Incas: A cycle accurate model of UltraSPARC," *ICCD*, pp. 130-135, 1995.

[12] J. Rowson, "HW/SW co-simulation," *DAC*, pp. 439-440, 1994.

[13] B. Schnaider, E. Yogev, "Software development in a hardware simulation environment," *DAC*, pp. 684-689, 1996.

[14] Synopsys Eagle tool. http://www.synopsys.com/products/hwsw/.

[15] V. Zivojnovic, H. Meyr, "Compiled HW/SW co-simulation," *DAC*, pp. 690-695, 1996.

[16] AXYS SuperSim simulators. http://www.axys.de/products.