INTRODUCTION

From the dawn of the first use of microprocessors and microcontrollers in embedded systems, the software has been blamed for products being late to market. This is due to software being developed after hardware is fabricated. During the past few years, the use of Hardware Description (or Design) Languages (HDL’s) and digital simulation have advanced to a point where the concurrent development of software and hardware can be contemplated using simulation environments. This offers the potential of 50% or greater reductions in time-to-market for embedded systems.

This paper is a tutorial on the technical issues that underlie software-hardware (sw-hw) co-simulation, and the current state of the art. We review the traditional sequential hardware-software design paradigm, and suggest a paradigm for concurrent design, which is supported by co-simulation of software and hardware. This is followed by sections on HDL’s, modeling and simulation; hardware-assisted approaches to simulation; microprocessor modeling methods; brief descriptions of four commercial products for sw-hw co-simulation and a description of our own experiments to develop a co-simulation environment.

DESIGN PARADIGMS
The Traditional Hardware-Software Development Cycle

The workflow in a traditional embedded processor project is shown in Figure 1. Software is typically developed after a hardware design has begun to stabilize (some actually do!) and prototypes are available for integrating the software and hardware. Add to this the limited observability of the operation of the hardware as the software executes, and the inability to control all of the elements of the design, especially peripheral components running synchronously from the microprocessor or microcontroller, it becomes obvious why the hardware-software integration and test becomes the most time consuming part of the project (almost all engineering veterans have at least one horror story of some heroic effort that involved sleeping under their desk...).

Attempts at concurrent hardware and software development have resulted in only a small portion of software bugs being found early. Software still waits for hardware prototypes before significant progress is made on integration. So, sw / hw integration problems are still found late in development and solved in software. Historically, the nature of technology has made this a fact of life. However, advances in hardware modeling and simulation offer the potential to develop a new design paradigm using software-hardware co-design and co-simulation.

Hardware-Software Co-design and Co-simulation.

The process of concurrently developing hardware and software encompasses two
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
main areas of study, co-design and co-simulation. In the context of this paper the term co-design refers to the process of translating the requirements for a desired system level functional behavior into a partition of hardware and software designs which, taken together, provide and maintain the desired system behavior. The term co-simulation, is simulation of a model of the hardware running the software, simultaneously providing visibility (and control to a lesser degree) of the hardware model while allowing the execution of the software to be controlled and observed at any and all levels of detail necessary for the comprehension and understanding of the behavior of the system. The co-simulation environment should provide a user interface that is consistent with the current state-of-the-art for both the hardware simulators and the software emulators currently used by the development team. In the ideal world this user interface would provide the same look and feel as the equipment used to test and verify the actual hardware after it has been produced.

A New Paradigm for Developing Embedded Systems.

Figure 2 shows an end-to-end development process that uses modeling and simulation. The system’s functional requirements are specified as the inputs and outputs of a model, and restrictions on its implementation. A behavioral model is developed to relate the inputs to the outputs. Simulations of the behavioral model are used to validate that the requirements are satisfied. The model becomes part of the specification, and simulating the model becomes the method for interpreting the specification. Then, design of the hardware and software is much less likely to suffer the problems of misinterpretation of an English language description of that behavior. Maintaining the same behavioral description of the system throughout the development process, and filling in the details needed to achieve that behavior at each step in the design, coupled with a co-design and co-simulation environment, allows designers to partition a system into smaller functional elements which, together retain the overall behavior. Sub-system elements are each described by their own behavior which is derived from the system level behavioral model. This decomposition and stepwise refinement of the behavioral model of the system and sub-systems (and sub-sub-systems) continues until the entire system is composed of functional elements that are small enough that the detailed design of that element is straight...
Figure 2. A new design paradigm for concurrent software and hardware development supported by co-design and co-simulation of software on hardware models.

forward. Part of this decomposition process also includes partitioning functional elements into hardware and software components which together produce the desired behavior of the functional element. This hardware / software partitioning is carried out and determined based on traditional design tradeoffs such as performance, cost, volume, etc.

The co-simulation environment provides the means to verify the behavior of each functional element composed of hardware and software. The behavior of elements implemented in hardware only is verified by hardware simulation. In all cases, starting from the top level simulation and working down to each functional element of each sub-system, the behavior defined at each sub-system level is used to verify the behavior of the lower level elements making up that sub-system, and so on down to the simulation of the lowest functional elements. Hardware models used in co-simulation, while primarily behavioral models to maintain simulation speed, are developed with synthesis of that model into silicon as a goal. Synthesis of the model (or verification of the model with an equivalent commercial part) and subsequent testing in the system allows the model to be fine tuned, thereby improving the fidelity of the models used to perform simulations. Finally, by maintaining the behavior that was defined at the highest level of the system down through each subsequent level, the fidelity of the product is ensured and that behavior can be used to establish the tests used for product acceptance.

HARDWARE DESCRIPTION LANGUAGES, MODELING AND SIMULATION

Modern digital design is often performed using hardware description languages such as VHDL (VHSIC Hardware Description Language) and Verilog HDL. Many textbooks about the VHDL and Verilog languages are available, among them ARM93, ASH96 and TH096. VHDL and Verilog are similar to programming languages, having their roots in Ada and C, respectively. (The software engineer will take great satisfaction in the observation that hardware design is reduced to programming!) In this section we will touch on several subjects pertinent to software-hardware co-simulation, including the
history of HDL's; how HDL models can describe circuits in different levels of detail to support hierarchical modeling; schematic capture from models; model verification; some different simulation methodologies; and, model availability.

VHDL is the older of the two, and was originally developed in the DARPA Very High Speed Integrated Circuit (VHSIC) program as a means of providing precise descriptions of circuitry. As time went on, the HDLs were developed for purposes of modeling, logic design, simulation of models, and synthesis of circuitry from logic designs. HDL's are quite flexible, and allow digital components to be represented at various levels of detail. An 8-bit adder, for example might be described at a behavioral level using an algebraic expression, or it could be described using Boolean expressions. The behavioral description will be quite satisfactory for modeling and simulating many aspects of a system. However, the Boolean description is more readily synthesized into a circuit using an automatic synthesis tool.

Synthesizing circuits from the models used to simulate the design is a very powerful design methodology because of the intimate tie between the model and the final circuit. The model becomes the circuit specification, and simulating the model becomes the method for interpreting the specification. Issues arising from a writer’s and a reader’s different interpretations of written language are obviated. Top-down design methodologies can begin with high-level behavioral models which are progressively synthesized into register transfer level (RTL) models, and finally gate-level models.

The largest users of VHDL and Verilog are integrated circuit designers. ASICs and gate arrays are commonly designed using these languages, and tools are available for synthesizing into circuits or for automatically routing connections in the gate arrays. For board-level design, VHDL seems to be the language of choice when an HDL is used. IC designers tend to use a different set of design tools and methodologies than do the IC users.

A high level VHDL description of an Intel 8051 microcontroller is in Figure 3. The top level of the design is the entity which contains an optional generic statement and a port statement. The entity's name is typically the name of a component. If used, the generic statement contains constants, such as setup times and delay times. The port describes the signals which are the inputs and outputs of the microcontroller. The port statement maps directly to the pins of the device being modeled. In this example, signals are a data type called standard logic, which can have nine different values, to take account for electrical properties of real signals. The architecture of the entity is where actual work is done on the inputs to produce the outputs. We show two different architectures for the 8051. The first, called RTL_MODEL, is intended to contain a detailed model suitable for synthesis into an IC. The second, called BEHAVIORAL_MODEL, produces the same behavior at the port, with a simpler internal representation. The architectures may be primitives that contain no lower level models themselves, or they may be constructed out of lower level components, which are also entity-architecture pairs.
entity 8051 is
  generic (tval: 8051 delays:=Our_CMOS_delays);
  port(ean, rst, xtall, xtal2: IN std_logic;
       ale, pSEN: OUT std_logic;
       p0 : INOUT std_logic_vector(7 DOWNTO 0);
       p1 : INOUT std_logic_vector(7 DOWNTO 0);
       p2 : INOUT std_logic_vector(7 DOWNTO 0);
       p3 : INOUT std_logic_vector(7 DOWNTO 0) );
end 8051;

architecture RTL_MODEL of 8051 is
  code
end RTL_MODEL;

architecture BEHAVIORAL_MODEL of 8051 is
  code
end BEHAVIORAL_MODEL;

Figure 3. The entity, generic (timing) and port statements for an Intel 8051 microcontroller, and two architectures which may be used to implement the 8051.

The structure of the language lends itself well to schematic capture. A symbol for a component can be readily generated from an entity's port statement. The architecture of the entity can be associated with the symbol. Symbols are then linked together into a schematic drawing, compiled, and passed to a simulator. The same circuit can be simulated with different underlying models by changing the architectures used for the symbols.

To illustrate the difference in complexity between RTL and behavioral descriptions, consider how registers can be represented and incremented. An RTL description of an 8-bit register will be done using an entity-architecture pair. The register inputs are a data byte signal, clock and clear signals, and an output data byte signal. Their data type will be standard logic. Storing the signals will probably take eighteen bytes, one for each bit. In our 8051 model, the architecture is about fifteen lines of code, and quite a number of steps need to be performed to move data from the input to the output. In contrast, to represent the behavior of a register as seen from outside of the model, it is enough to have a one-byte variable which contains the register contents. To simulate incrementing the RTL register, its contents will be transferred to the accumulator, incremented, and written back to the register, taking hundreds of machine instructions. In contrast, incrementing a register in a behavioral model may take only three instructions; fetch to accumulator, increment, and store to register. This comparison makes several points for software-hardware co-simulation. First, an HDL model may be complex or simple. One needs to
ask about the nature of the models being used. Second, the choice of model type can affect the execution time of a simulation by orders of magnitude. Third, one needs to be concerned with the fidelity of the model, i.e. how accurately the behavior of a circuit will be modeled. Verifying fidelity of a behavioral model of a register to an RTL model should be trivial, but in more complicated examples this becomes a very difficult problem in its own right.

Elementary logic primitives are available with most HDL simulators. Models of common circuits in behavioral and synthesizable forms are available from vendors. Behavioral HDL models of many IC's are commercially available from companies such as Logic Modeling, CAST (Pomona, NY), Sand Microelectronics (Santa Clara, CA). Models of microprocessors are more difficult to obtain. Synthesizable models used by manufacturers are, of course, very valuable intellectual property. Some vendors sell only compiled versions of models. Working with a compiled version of a model limits your ability to see what is going on inside it, a potentially important part of debugging software in a virtual environment. The availability of models may influence how an application is modeled.

HDL simulators are available from many different vendors (for example, Synopsys, Viewlogic, Cadence, Model Tech, and just look in EE Times for more) but there are also two major simulation algorithms to choose from. Most HDL simulators on the market today are event-driven simulators. An event-driven simulator keeps track of events happening at arbitrary times, which is necessary for simulating asynchronous logic. Cycle-based simulators are just being introduced in the market. Cycle based simulation requires all logic transitions to occur on clock cycle boundaries, i.e. fully synchronous logic designs. This dramatically reduces the scheduling overhead, and reduces simulation time by as much as 10-20x for suitable models.

The execution time of simulations is important for sw-hw simulation. Our RTL model of the Intel 8051 executes 3-5 instructions per second using a Viewlogic simulator running on a Sun Sparc 20. Before concentrating on tradeoffs for sw-hw simulation, it is worth briefly considering alternatives to software-based simulation.

HARDWARE-ASSISTED SIMULATION OF MODELS

Simulation time is an issue that is also faced by hardware designers who must verify the correctness of large designs. Hardware designs are verified at the gate level to the extent possible. Verification of large gate-level designs can be addressed by hardware-assisted simulation and by emulation.

In hardware-assisted simulation, a high-powered co-processor is used to execute the simulation of the gate-level model. Other parts of the simulation, such as peripheral circuits and test benches, are left in the software simulator. Zycad is a major accelerator vendor that recently introduced a product called Lightspeed which executes about 4000
instructions per second for an 80486-class processor, a ~1000x speedup over a software simulation. A Lightspeed accelerator will cost several hundred thousand dollars.

In emulation, the entire gate-level design is loaded into a programmable gate array. Quickturn is the largest vendor of this type of product. The major components of a “Quickturn box” are field-programmable gate arrays, the electronics to monitor the arrays, and the software to control the system. The gate arrays can be clocked at 1-4 MHz, so a model can be executed at speeds that are very fast compared to simulation, 1% to 10% of the full design speed. Emulation is considered to be an important part of IC design verification [VIV96]. Maintaining and using an emulation system is also a substantial amount of work [DIC96], and the price is in the hundreds of thousands of dollars.

“Hardware modeling” is a hybrid technique which runs software on a hardware processor that is interfaced to a software simulator which runs models of the peripheral circuitry. The speed of the entire simulation is controlled by either the software simulation or the communications overhead between the hardware processor and the simulator. Hardware modelers typically execute 10-50 instructions per second [ROW94]. However, the microprocessor is much easier to buy than the model the vendor wrote to design it!

So, what is the message for sw-hw co-simulation? If your company’s hardware design path uses accelerators or emulators, it will be valuable to debug software on these models, if you can attach a software debugger to them. If not, they are probably inappropriate for sw-hw co-simulation because the accelerators and emulators work with gate-level models which take a lot of work to build, and have more detail than necessary for most of the software development.

SOFTWARE-HARDWARE CO-SIMULATION: PROCESSOR MODELS

Figure 4 shows the major elements of a sw-hw co-simulation environment. It is composed of software, a processor with memory, and peripheral hardware. Note that most processor interactions will be with the program and data memory; at most a few percent of its activity will be communication with peripherals. The software is executed on a model of the processor in order to interact with the peripherals. The software and hardware development tools allow the developers to view and control the simulation of the software and hardware models. Depending upon the models chosen, the processor and memory may or may not have internal states visible from the outside. The method of modeling the processor and its interaction with memory is one of the most important choices to be made. As one considers the choices, the 20-80 rule (20% of the effort yields 80% of the results) should be kept in mind, and balanced against the possibility that a serious error will be missed! Table 1 shows the execution speed and ease of use for various combinations of processor models and simulation and emulation methods.

Paraphrasing Rowson [ROW94], the tradeoffs to be evaluated in this choice are between (1) model and timing accuracy, (2) visibility of the internal state of the processor.
Figure 4 The elements of an embedded system in a co-simulation environment with software and hardware development tools. The internal state of the processor and memory may be visible to the software and hardware development tools.

and peripheral models for debugging purposes, (3) model availability, and (4) simulation speed. How you make tradeoffs depends upon where you are in the design process, and the type of design (board vs. ASIC). Modeling and co-simulation can be used early in a design to evaluate software-hardware tradeoffs which will impact the hardware design; a co-design activity by our definition. For example, early in a design the required processing power must be established. Later in a design, the processor is fixed, and you may be doing the software-hardware integration in advance of receiving a prototype, what we defined as co-simulation. The objective of the modeling and the types of models which are available for the hardware simulation also depend upon whether the design is of a board level system which will be built with commercial ICs, a board with a commercial processor and custom ASICs for peripherals or a fully custom ASIC with an embedded processor core. Re-work cost becomes progressively larger with these three options!

At the co-design level, high-level models which run high-level code, without showing internal details of processor operation may be satisfactory. Some provision needs to be made for the occasional interaction with the peripherals. For board level design with discrete ICs, you will probably find only behavioral models of the processor and the peripherals. If you use custom ASICs, you may have access to RTL models used to design them. After receipt of a board-level prototype, you will still have good visibility of the interconnections between the processor and the peripherals, so that conventional methods can still be used for final integration. For an ASIC with an embedded processor, models may be available down to the RTL level for the entire design. In this case in particular, detailed models may be desirable for portions of the sw-hw integration because it will be the only prototype available before a very expensive and time-consuming ASIC turn. And, after the ASIC is fabricated, you will have little visibility into it for de-bugging; the software and hardware viewers may need to be your in-circuit emulator and logic analyzer. RTL processor models give the best accuracy and internal visibility, but they are difficult to obtain. If the processor and peripherals are asynchronous, event-driven simu-
lation is required to represent their interactions "exactly", and only a few instructions per second will be executed. At the cost of only evaluating the simulation at clock-cycle boundaries a cycle-based simulator increases speed to about 100 instructions per second. The value of using an RTL model is that it will have nearly absolute fidelity with the hardware synthesized from it.

Simplified processor models may be acceptable to the software developer. Instruction set simulator (ISS) models of processors are commonly included in software development environments. An ISS model includes the internal registers and memory, although it probably assumes sequential execution of the instructions, an assumption which may not be satisfied for pipelined processors. An ISS can be written in an HDL and be executed by the HDL simulator with the peripheral models. Execution speeds can be increased to the 2000-20,000 instructions per second range. An ISS can also be written in a conventional programming language and executed in a process separate from the HDL simulator if there are means for synchronization and communication. An ISS model keeps track of execution time pretty well, and it can be used to generate many of the bus-cycles produced by a real processor. For a software developer, an ISS can provide most of the processor functionality, while having a very realistic connection to the peripherals. A disadvantage of ISS models, as with any behavioral model, is that fidelity to the actual hardware may be poorly understood.

An ISS is useful for software integration relatively late in a project, but it may also be used for system behavioral modeling earlier in the design. In conjunction with behavioral models of peripherals it can be used to help evaluate processor requirements, sw-hw tradeoffs, and validate system conformance to requirements, before proceeding with detailed design.

Further simplifications are possible, as well. The application software could be a program written in a high-level language [WIL94], executing on the workstation in the

Table 1. Comparison of execution speed and ease of use of processor models using simulation and emulation methods. Scores are subjective on a scale of 0-10.

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Speed (instruct/s)</th>
<th>Control &amp; Visibility</th>
<th>Model Availability</th>
<th>Model Difficulty</th>
<th>Fidelity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Driven RTL</td>
<td>1-10</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Cycle-Based RTL</td>
<td>~100</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Event-Driven ISS</td>
<td>$10^4$</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Cycle-Based ISS</td>
<td>$10^5$</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>&quot;Processor-less&quot;</td>
<td>$10^3$-$10^5$</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Hardware Modeler</td>
<td>$10^6$</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Gate-Level Emulation</td>
<td>$10^5$</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>
workstation's native assembly language, communicating with the hardware simulation by special function calls. Synchronizing the software execution with the simulation of the peripheral hardware will be imperfect, but perfect synchronization may not be necessary.

SOFTWARE-HARDWARE CO-SIMULATION PRODUCTS

Tools and environments for really doing software-hardware co-simulation began to appear in 1995. An effective environment must tie together tools familiar to both the software and hardware engineers. This is a difficult environment to produce because of the many tradeoffs to be made, and there is no single right way to accomplish the task. We are aware of four companies which provide capabilities for sw-hw co-simulation. In this section we give brief descriptions of their products, as we understand them.

EagleI and EagleV - Eagle Design Automation

Eagle Design Automation offers, EagleI [BUN96, EAG96, WIL96], now available, and EagleV, due to be introduced in September, 1996. Both are relatively high-level modeling tools, with two major components. The first is the Virtual Software Processor (VSP), and the second is the Virtual Product Console (VPC).

The Virtual Software Processor executes high level language code compiled into the native machine language of the workstation. The only internal behavior of the target processor and its memory that is modeled is the part that handles interactions with external devices, e.g. i/o ports. High-level language function calls communicate through these ports with the peripheral device models which are executed by an HDL simulator. The strengths of this method are the simplicity of the model, and rapid execution speed (it can run 5000 instructions per second). The weaknesses of the method are that model verification is difficult because the models are far-removed from the detailed behavior of the target processor, and execution of the software and the hardware simulation are difficult to synchronize. Three methods are used for synchronizing the VSP with the HDL simulator [ZAC95]. The first is Polled Status synchronization in which a hardware action is initiated by the software, which polls a status register until a flag is set signifying that the hardware action is complete. The VSP can also respond to "hardware" driven interrupts much as a hardware processor would respond. The third way is for the software to estimate how long a simulation of a hardware action will take.

The Virtual Product Console links together the software development environment, the VSP, the hardware simulator and its development environment. Commercial software development tools are supported by the VPC. The VPC connects the VSP to VHDL and Verilog simulators from most of the major CAE vendors. One of the virtues of EagleI is that it should fit into many existing design environments. Another interesting feature is that the VPC can link multiple processor models to the HDL simulator.
Zach and Wilson [ZAC95] describe use of an Eagle1 prototype in the design of a complex ASIC. Three serious errors were caught in simulation which would have resulted in ASIC redesign and re-spin.

VIRTUAL ICE - YOKOGAWA ELECTRIC CORPORATION

Yokogawa offers products called Virtual ICE™ and VMLink™ [YOK96]. Virtual ICE provides an in-circuit emulator environment for code development on HDL models. Conceptually, it plugs into an HDL model of an application (ASIC or board) in the same way that a hardware ICE plugs into a processor socket in a circuit board. VMLink is an interface for connecting to the MATLAB™ simulator. MATLAB is a more general modeling tool, so that the simulation of a digital application can be coupled to a simulation of its surroundings. For example, a motor controller model can be connected to a motor model. This looks like a very useful tool if it will fit into your design environment [SAT94].

Virtual ICE consists of a processor model and an ICE interface to the processor. The processor model is a bus-cycle-accurate ISS written in Verilog. Their method for model verification is not disclosed. The present version of Virtual ICE can be run using the Verilog-XL and VCS Verilog simulators. Future versions of Virtual ICE will run under “bilingual” simulators which will accept designs with a mix of VHDL and Verilog components. Off-the-shelf processor model offerings are somewhat limited, but custom models are available. The Verilog language includes a Programming Language Interface, which is used to connect to ICE software. The ICE software provides windows for source code, tracking assembly language, internal registers, memory and breakpoints. Execution of the model is controlled using the ICE interface. We have seen little emphasis on visibility into the hardware side of the design, but presumably there is some access through the Verilog simulator.

System Lab and Behavioral Verification Technology - CPU Technology

CPU Technology’s main business is design of custom microprocessors for which they have developed their own design methodology and proprietary tools. King [KIN96] describes the design of a MIPS quad-processor system with this approach. It appears to be one of the most advanced top-down design methodologies in use today. They offer two products called SystemLab™ and Behavioral Verification Technology™ (BVT). CPU Technology does not make their language or simulators available for sale. What they do offer is a modeling service which provides their customer a model and the environment for working with it. They work with customers to take models to silicon.

SystemLab is an environment for developing behavioral models of systems and their components. Models of processors and peripherals are constructed using CPU Technology Design Language, a proprietary HDL. The models are executed using a proprietary cycle-based simulator. Their design approach is to design the functionality of all system components at this behavioral level before going to detail design of the compo-
nents themselves. Very high visibility into designs is available using virtual instruments such as a logic analyzer and oscilloscope. The microprocessor models run real code and contain representations of all internal registers. They have models of x86, Motorola 68000, MIPS and various DSP processors. A unique feature of their simulator is that it will run backwards, so that once a problem is observed, it’s easy to back up a few cycles to examine what went wrong, rather than re-starting the simulation. SystemLab models can be further synthesized into circuitry through a path using Verilog and Synopsys synthesis tools. Verifying model fidelity to the final circuit with this design path is relatively straightforward. However, when the methodology is applied to model existing components, model verification becomes a critical issue.

BVT is CPU Technology’s method for verifying the accuracy of models. It is a semi-automated method for generating extremely comprehensive sets of test vectors which can run on behavioral models, gate-level models, or real hardware. Quantitative measurements are made of the behavioral model fidelity to the gate-level model or hardware. We were quoted a price for BVT test modules for a model of a small system that appeared to be 20-30 times greater than the price of the model. This price disparity highlights the difficulty of comprehensive verification of behavioral models.

**Seamless Co-Verification Environment - Mentor Graphics and Microtec Research**

Mentor Graphics and Microtec Research merged in 1995 [GOE95] in order to link software and hardware development products. They have recently announced a product called Seamless Co-Verification Environment (Seamless CVE) [LEE96, MEN96].

Seamless CVE combines the Microtec X-Ray Software Development environment with Mentor Graphics’ HDL simulation. Microtec has provided instruction set simulators for many processors for some time. Standalone execution speeds of up to 100,000 assembly instructions per second are quoted for the ISS models. Seamless CVE is an interface between the ISS model and the HDL simulator which executes models of the peripheral devices. The interface functions include synchronization of the ISS with the HDL simulator, memory management functions, and transferring information between the ISS and HDL processes. Seamless CVE allows its user to take advantage of the observations that (1) most CPU activity is with memory, and (2) less than 1% of the instructions communicate with peripherals. The architecture of CVE allows the user to run memory models in the HDL simulation, or in the CVE. They quote execution of 450 instructions per second with memory fetches from the HDL simulation. CVE allows the user to choose (dynamically during execution) various levels of filtering of the transactions between the ISS and the HDL simulation. The first filter is instruction fetch masking. Additional clock signals can be suppressed by the user when he knows that there is no processor-peripheral interaction taking place. With full filtering, execution may speed up to as much as 15,000 instructions per second. This methodology should give good timing fidelity. The fidelity of the ISS to a real processor should still be verified.
Sandia’s Approach

We have been experimenting with ways to provide a co-simulation environment using commercial off-the-shelf tools for over a year. We thought this was an important capability that would shorten our development cycle; improve the integrity, reliability, safety and security of our systems; and provide an excellent spring board to the next major hurdle, a co-design environment. When we began no tools were available, although this is rapidly changing. Coupling commercially available tools allowed us to take advantage of each tool's strengths, providing a kind of synergism not typically found in a single vendor solution to a multi-discipline problem. This gave us the greatest flexibility to configure the co-simulation environment, allowing us to decide what level of detail is needed based on which stage of the development cycle we are in.

Figure 5 shows the environment we developed. There are three software components, running on two workstations across a network. The first is a commercial HDL simulator, the second is a commercial software debugging tool, and the third is our translator program to connect them. For VHDL simulation we chose Viewlogic Systems' Viewsim VHDL simulator and Viewtrace, a waveform viewer that provides a “real time” display of hardware signals. For the software debugging tool we chose SourceGate II (SG II) from Huntsville Microsystems, Inc., which is the GUI for their In-Circuit Emulators (ICE). We chose SG II because it supports an Internet communication interface which facilitated interprocess controls and there is a published list of the emulator commands and responses. The translator program, called “Vemulator”, provides ICE re-

Figure 5. A co-simulation environment constructed from commercial, off-the-shelf software and hardware development tools.
responses to SG II, and controls the VHDL simulation. The computer in the lower left of Figure 5 displays the GUIs for the hardware and software and the computer in the upper right is a fast, high powered workstation running the VHDL simulator. All of the elements will run on a single workstation without modifications, or, as shown, coupled across a network, allowing additional computing power to be utilized.

Our work centered on Vemulator, which executes on the same workstation as Viewsim. Vemulator starts and initializes Viewsim and the VHDL models which it runs. Vemulator is a server for SG II, accepting ICE commands, translating &em into Viewsim commands, gathering the results of the simulation from Viewsim, and returning a response to SG II. Vemulator also starts Viewtrace, providing a hardware waveform display.

We think this approach is valuable because, for the price of writing the translator program, we retain the development environment already familiar to software and hardware engineers and add powerful new capabilities while maintaining flexibility. Developing Vemulator allowed us to couple a tool used by software developers for sw-hw integration to the hardware simulation environment used by our hardware designers. This allows the two sides to work concurrently while keeping their accustomed development environments. Existing tool knowledge and experience is maintained throughout the development process and training for similar tools used at different stages is kept to a minimum. An even more important advantage to us is the ability to change the level of detail contained in the hardware models at various stages in the development process. At the front end of the development effort, fairly simple behavioral models and an ISS (written in VHDL) for the processor can be used to define the system level behaviors. Later, when more detailed design information is available, simpler models can be replaced by the more detailed models, allowing the checkout of critical interfaces and high consequence elements of the system like security critical functions. The design team, rather than the tool, is allowed to make the decision as to the amount of detail necessary.

**SUMMARY AND CONCLUSIONS**

Significantly reducing the time-to-market for embedded processor systems will require adoption of development paradigms which greatly increase the degree of concurrency of software and hardware development compared with today's sequential development of hardware and software. We see software-hardware co-design and co-simulation as two elements of the new paradigms. In this paper we have described how the technology of hardware modeling and simulation has advanced to a stage at which software can be sensibly developed on models of hardware, the co-simulation part of a new paradigm. A key to successful modeling and simulation is to make models with no more detail than necessary, and to choose an appropriate simulation method. We have described styles of HDL modeling and the main simulation methodologies. We reviewed four co-simulation products that are available in the market. We also described how we developed a tool at our company by integrating commercial products that presents the user a familiar look-and-feel, while giving him significant choices in modeling style and level of detail.
are many ways to construct sw-hw simulation environments. The trick will be to identify methods that will improve productivity at your company.

ACKNOWLEDGMENTS
We wish to thank Huntsville Microsystems for their assistance with interpreting Source-Gate II commands in our development of Vemulator. This work was supported by the U.S. Department of Energy under contract DE-AC04-94AL85000.

REFERENCES
DIC96 W. Dickerson, “IC Emulation Technology a Mixed Bag”, p. 82, EE Times, May 13, 1996.
YOK96 Yokogawa Electric - http://www.yokogawa.co.jp/Eda/eda_ice