Seamless CVE™
Hardware/Software Co-Verification Technology

by Russ Klein, Ross Nelson
Mentor Graphics Corporation

Abstract
Embedded systems rely on a close, integrated relationship between software and hardware. However, in a traditional embedded systems design cycle, hardware and software are often developed separately, then integrated after the hardware has been built. Not surprisingly, often the hardware and software do not work correctly together the first time. In many cases the real software and integration debugging begins only after the hardware has been built. This debug and integration can take up to half of the project schedule.

To remedy this problem, Mentor Graphics, in partnership with Microtec, has developed the Seamless Co-Verification Environment (CVE). Seamless CVE enables designers to link software execution to the hardware simulation and co-simulate the hardware and software. Software integration can be started much earlier in the design cycle, without having to wait for the hardware prototype to be built.

Seamless CVE™ Co-Simulation
Traditional logic simulation is too slow to execute software. All hardware/software co-simulation tools must augment the performance of the logic simulator to allow software to be executed and debugged. The Seamless CVE approach to accelerated co-simulation is two-fold. First, the processor's functionality is separated from its interface. Second, users are able to selectively suppress bus cycles in the hardware simulation.
The components of Seamless CVE are as follows (see Figure 1):

- The **Co-Simulation Kernel** controls communications between the software and hardware portions of a co-simulation session. This kernel also permits users to set up various aspects of the co-simulation session through the Configuration Manager. The Co-Simulation Kernel services all address-space accesses from the Instruction-Set Simulator (ISS). The kernel also determines whether to pass those requests to the Bus Interface Model or service the requests directly through local memory, without hardware bus cycles. The Instruction Set Simulator reports the number of clock cycles required for a given instruction to the Co-Simulation Kernel.
- The **Instruction-Set Simulator or Native Compiled Software** (NCS) performs the software portion of a co-simulation session. The Instruction Set Simulator executes machine code produced by cross-assemblers or compilers for specific processors. NCS is software written in a high-level language and compiled for execution directly on the workstation.
- The **Configuration Manager** connects the communication path between the Instruction Set Simulator and the Bus Interface Model, executing in the hardware part of the co-simulation.
- The **Debugger Interface** controls execution of the Instruction Set Simulator and NCS. The Debugger Interface also performs operations such as single-stepping, examining registers and memory, and other typical debugger functions.

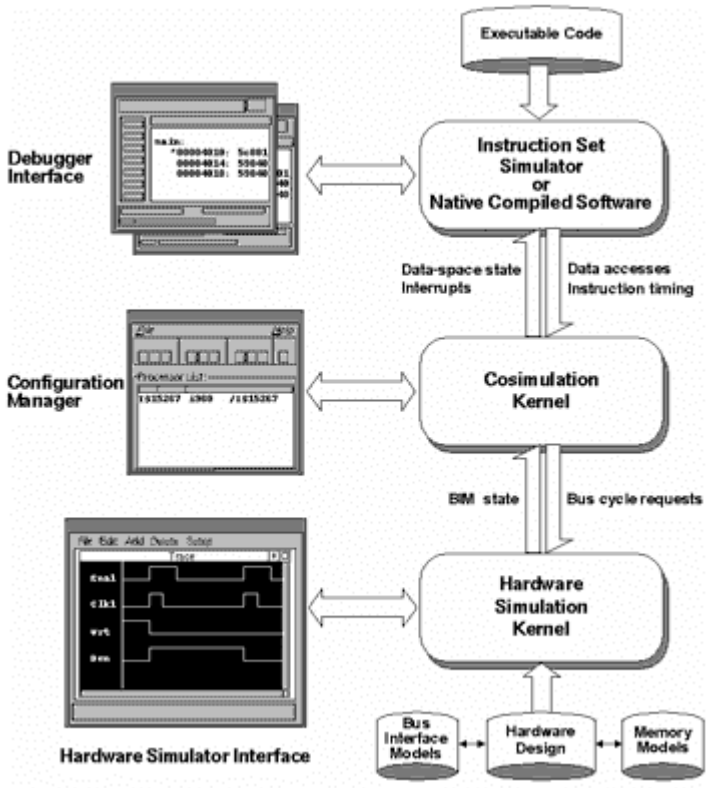

Figure 1: Seamless CVE™ Architecture

The **Hardware Simulator Interface** and **Hardware Simulation Kernel** perform the hardware portion of a co-simulation session. The kernel executes a hardware design that has instantiated within it one or more Bus Interface Models and special Seamless CVE memory models.

Instruction-Set Simulators and Debuggers

An Instruction Set Simulator is a software application that models the functional behavior of a processor's instruction set. An Instruction Set Simulator is not a hardware model of the processor; instead, it is an abstract model of the data processing that occurs during instruction execution. An Instruction Set Simulator can run much faster than a hardware simulation because it does not have to evaluate those signal transitions that occur in the gates and registers within the processor.

Native Compiled Software

Native Compiled Software is not machine code executed by an Instruction Set Simulator. Instead, it is compiled for and then executed directly on the workstation. One advantage to using NCS is that it runs much faster than machine code running on an Instruction Set Simulator. In addition, portions of the software for an embedded system could be implemented in a high-level language and tested directly with the hardware simulation, without requiring the machine code to be available.

The link to the hardware simulation process is through an Application Program Interface (API). The API exports a set of data-access functions, connecting the compiled code to the Bus Interface Model in the hardware simulator.

Bus Interface Models

For hardware simulation, a processor is defined by the way in which its input and output pins interact with the surrounding hardware. For Seamless CVE co-simulation, this interaction is modeled with a Bus Interface Model (BIM). The Bus Interface Model is a simulation model that can be instantiated in a design and executed on a hardware simulator. Since the Bus Interface Model is a more abstract model and only models pin transitions, it simulates much more quickly than a fully functional model.

In general, the Bus Interface Model is able to execute standard processor bus cycles, including:

- Read and write memory
- Read and write I/O ports
- Read-modify-write
- Reset and halt
- Exceptions, interrupts, and faults
- Bus request and grant

Configuration Manager

The co-simulation process for a particular design begins with the Configuration Manager. Using the Configuration Manager (see Figure 2), the user sets up all aspects of a co-simulation, including the following operations:

- Associating a software program with a corresponding Bus Interface Model instance
- Mapping memory address ranges to hardware-only, software-only, optimizable, and illegal access regions
- Mapping Seamless CVE memory instances into optimizable memory ranges
- Optimizing memory regions and simulation time
- Starting up the software and hardware simulators

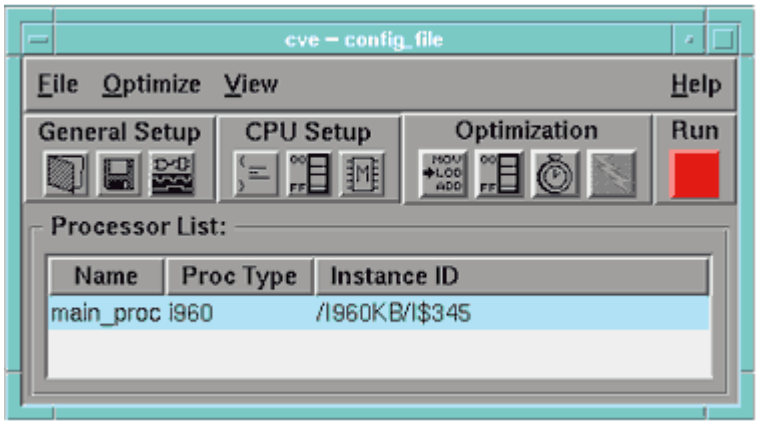

Figure 2: Configuration Manager User Interface

Data Access Cycles

Data access requests are reads and writes to memory and I/O ports. These requests originate from the software being modeled in the Instruction Set Simulator or NCS. When the Bus Interface Model receives a data access request, it executes the requested bus cycle by modeling pin transitions. The timing of bus cycle pin transitions is resolved to processor clock transitions.

Processor data books usually describe bus cycles in terms of the minimum number of clock cycles necessary to complete the bus cycle. In many cases, the actual number of clock cycles used to execute the bus cycle can be greater. For example, the memory device being accessed might be slow, and wait states might need to be added to the bus cycle. In this case, the Bus Interface Model reports the additional wait states used when the bus cycle completes. This information can be used with the instruction-cycle count from the Instruction Set Simulator to determine the total number of clock cycles required to execute the instruction.

Exceptions and Bus Control Cycles

Data access cycles originate from the execution of embedded software instructions. Most processors also have events, such as faults, interrupts, bus-arbitration requests and grants, and coprocessor operations initiated from the surrounding hardware design. The Bus Interface Models provided with Seamless CVE simulate these bus cycles. In the case of interrupts, the interrupt data is passed back to the software simulation to be serviced by the software interrupt-service routines.

Micro-controller Peripherals

Many micro-controllers contain peripheral devices as part of the component. These peripherals include devices such as timers, interrupt controllers, Direct Memory Access (DMA) controllers, A/D and D/A converters, and others. In Seamless CVE peripheral components of the supported micro-controllers are modeled as well.

Seamless CVE Memory Models

Seamless CVE memory models are simulation models that can be instantiated in a design and simulated like any other memory model. However, they have special characteristics that allow a coherent view of memory to be maintained between the hardware and software portions of a co-simulation session. When a conventional memory model is used in a hardware simulation, its contents are not available for direct manipulation by the software simulator. Consequently, any access to this memory from software must be from the hardware simulator.

By providing a single coherent view of memory, a Seamless CVE memory model allows both the hardware and software simulator to access memory. Regions of memory can then be optimized to eliminate unneeded hardware bus cycles, while allowing critical hardware devices in the design (such as DMA or a co-processor) access to the same memory from the hardware simulation.

Seamless CVE is supplied with generic memory models that include dynamic RAM, static RAM, dual-port RAM, FIFO, and a register element. These models are parameterized, allowing a user to specify address-bus and data-bus widths. Additional parameters allow the user to specify timing delays.

Optimizations

A Seamless CVE co-simulation that includes an instruction set simulator can simulate hardware and software interaction to a high degree of accuracy. However, detailed co-simulation can be slow when processing a substantial amount of code. Seamless CVE provides optimization features to increase simulation speed. These optimizations fall into three general categories:

- **Data Access Optimizations** disable hardware bus cycles during accesses to selected addresses or address ranges.
- **Instruction Fetch Optimizations** disable hardware bus cycles during all instruction fetches.
- **Time Optimizations** de-couple the time synchronization between hardware and software simulation, and only insert hardware simulator bus cycles for address ranges which have not been optimized.

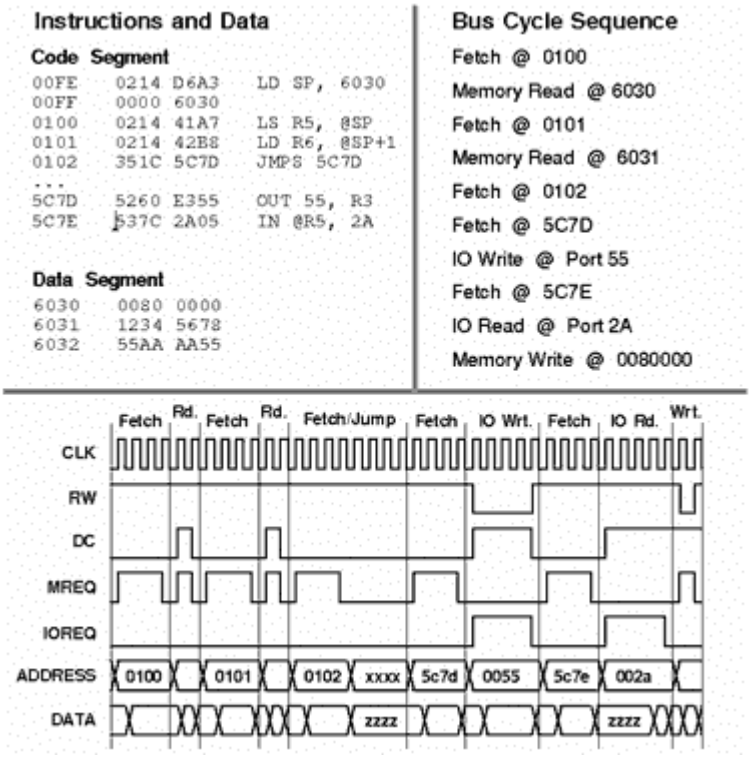

Figure 3. Full Simulation Bus Activity

Figure 3 shows an example of the bus activity of a full simulation. Once the proper operation of the bus cycles has been verified, most of the cycles shown in Figure 3 can be eliminated, since they supply no additional useful information to the simulation.

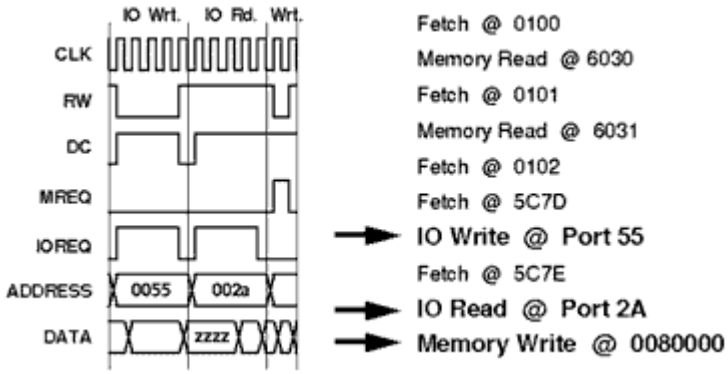

Figure 4: Simulation Bus Activity After Optimization

Figure 4 illustrates the effect of eliminating unnecessary bus cycles from the full simulation bus activity shown in Figure 3. All instruction fetches and two data accesses to general-purpose memory have been eliminated from hardware simulation. In addition, the time optimization has eliminated associated hardware clock cycles. The I/O read and write accesses are still simulated by the hardware, as well as the memory-mapped access to location 0080000. However, the amount of hardware simulation required is reduced to a fraction of the example shown in Figure 3.

Data Access Optimization

In a fully detailed co-simulation, each memory access generates a read or write bus cycle in the hardware simulation. The majority of memory accesses in most embedded programs store and retrieve data that only the program needs. The bus cycles generated by the accesses are not needed for the hardware simulation, except possibly to account for the simulation time that elapses during their execution.

Seamless CVE allows the user to enable and disable bus cycles during accesses to selected addresses or address ranges. When a data access is optimized, no bus cycles are generated for that access, and the Instruction Set Simulator reads or writes directly to memory. For timing accuracy, any bus acquisition logic that the processor executes is carried out. Additionally, Seamless CVE maintains the same number of clock cycles in the hardware simulation that would have elapsed in a non-optimized data access.

An optimized access accelerates the hardware simulation despite the fact that the same number of clock cycles are being simulated. Had the bus activity been present, it would have generated additional simulation events on all pins sensitive to changes on the affected nets, increasing the work done by the hardware simulator.

Instruction Fetch Optimization

Each instruction that the processor executes must be read from memory. In a detailed hardware simulation, this generates instruction-fetch bus cycles. The user can eliminate these bus cycles by optimizing data access to the appropriate address ranges. Alternatively, Seamless CVE allows the user to optimize all instruction fetches. This is a convenience optimization for use in an otherwise fully detailed or mostly detailed simulation. It quickly eliminates from each executed instruction a class of bus cycles that will rarely have an important impact on the co-simulation.

Time Optimization

For any given elapsed simulation time, Instruction Set Simulator simulation is faster than hardware simulation. But in non-optimized co-simulation, the Instruction Set Simulator must wait for the hardware simulator to "catch up" at synchronization points occurring no less than once per instruction.

Seamless CVE time optimization allows the Instruction Set Simulator to run ahead of the hardware simulator. The hardware simulator is allowed to execute bus cycles each time the software makes a non-optimized data access. However, if there are few non-optimized data accesses, the Instruction Set Simulator can run with few delays and greatly improve the speed of the total simulation.

With time optimization, it is sometimes desirable to allow the hardware simulator to execute a few clock cycles every now and then, even when non-optimized accesses are few and far between. This might be appropriate in an interrupt-driven system where the hardware simulator needs to stay active in order to generate interrupts to the software. Time optimization provides two optional parameters. The first specifies how many clock cycles the Instruction Set Simulator is allowed to run before the hardware simulation can proceed. The second indicates how many clock cycles the hardware simulator can execute once the Instruction Set Simulator has yielded.

Memory Mapping

Memory mapping configures a co-simulation session so that software access to memory is correctly correlated with the hardware simulation access. Seamless CVE does not analyze the design to produce memory maps; instead, it relies on the user to define the mapping. Although Seamless CVE does not generate the memory map, it does verify the correctness of the map that the user specifies.
There are two types of memory mapping:

- **Address Range Mapping** defines different ranges of memory and how they are accessed.
- **Memory Instance Mapping** indicates which memory instances in the hardware design cover which software addresses for which processor.

Address Range Mapping

An example of an address-range map is shown in Figure 5. This memory map has been partitioned into five address ranges. Each region has been assigned a label that appears to the right of the diagram.

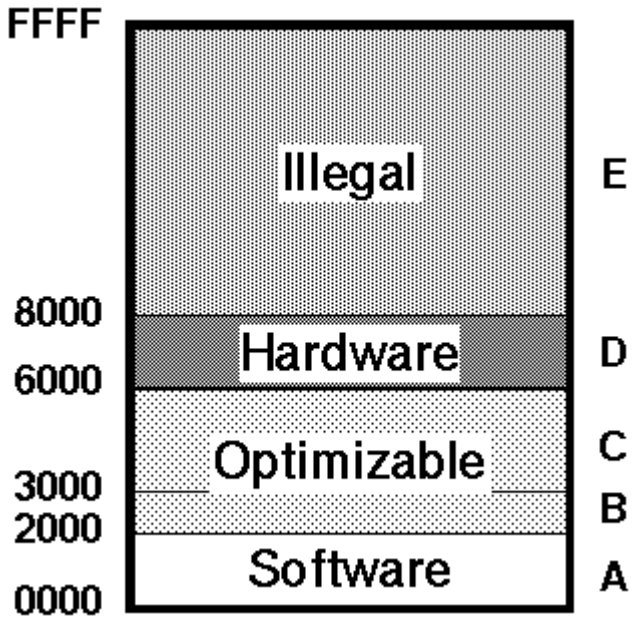

Figure 5: Address Range Map

Seamless CVE provides four access types, each of which is included in the diagram. The access types are as follows:

- Software: The Instruction Set Simulator models this area of the memory for itself. The Instruction Set Simulator never generates bus cycles to access a software-only region, and the hardware cannot access this memory. If anything from the hardware accesses an equivalent address, the result is determined by the hardware simulation and will not be consistent with what the software sees. In Figure 5, the software-only region is labeled A. This region's physical existence in the hardware has either not yet been modeled, or it has been modeled using a model that cannot be loaded by Seamless CVE.
- Optimizable: These ranges of memory can be accessed with or without generating bus cycles in the Bus Interface Model. They are the only ranges that respond to data optimizations. In Figure 5, two adjacent regions (labeled B and C) are optimizable. Optimizable mapping is used for an address range where memory is simulated using Seamless CVE memory models. In these regions, any data access can be optimized and restored as many times and in as many sub-regions as needed. The Co-Simulation Kernel maintains a consistent view of the data for access by either side.
- Hardware: The Instruction Set Simulator always causes the Bus Interface Model to generate the necessary bus cycles to access this type of memory. The hardware and software views of the data are consistent under program execution. In Figure 5, the hardware region is labeled D. Hardware mapping is used in designs where bus cycles access memory-mapped I/O devices. Hardware mapping is also appropriate for address ranges where memory is modeled with non-Seamless CVE memory models accessed by both hardware and software.

- Illegal: Illegal addresses are addresses that the software should never access. If an access is attempted, an error message is generated. Hardware can access the equivalent address; the result is determined by the logic simulation. In Figure 5, the illegal region is labeled E.

The user can associate the following additional parameters with software-only and optimizable access address ranges:

- Initial Content: This parameter allows the initial data contents to be defined for all locations within a specified address range of software-only memory.
- Read Wait States: This parameter is the number of wait states (clock cycles) to be added to each simulated instruction that reads memory within this range.
- Write Wait States: This parameter is the number of wait states (clock cycles) to be added to each simulated instruction that writes memory within this range.

The read and write wait state parameters do not apply to any given non-optimized access to optimizable memory. In this case, the wait state counts are obtained from the hardware simulation.

Memory Instance Mapping

Optimizable address ranges need to be simulated using Seamless CVE memory models, since only Seamless CVE memory models have the ability to communicate directly with the Co-Simulation Kernel. These memory models allow both hardware and software to have a consistent view of the memory, even if the memory access is not run in the logic simulation. In this case, Seamless CVE needs to know how memory instances map into the target processor's address space, which is accomplished through memory-instance mapping.

Seamless CVE warns the user before simulation has started that an optimizable region is defined that has not been completely mapped to Seamless CVE memory instances. This case will still work if the design contains non-Seamless CVE memory models in that address space, even though hardware and software views of the memory could become inconsistent.

Conclusion

Today’s embedded systems are composed of complex hardware and software components with a high degree of interaction between the two. The integration process can be long and risky, yet combined hardware/software verification normally begins only after the hardware prototype is built. Co-verification using a virtual prototype allows this integration task to begin before the hardware prototype is built. The optimization algorithms in Seamless CVE allow large amounts of software to be verified in concert with the simulated hardware before a hardware prototype is built and debugged.