

Connecting Hardware and Software Design Environments: Realities in Bridging the Gap

John K. Bartholomew and Geoffrey J. Bunza
Eagle Design Automation
Beaverton, Oregon

Abstract

This paper presents a new approach towards support of rapid system virtual prototyping, melding current hardware and software development environments, called Virtual System Integration. It provides for the rapid execution of target software applications as they are simulated in a virtual system modeled in VHDL or Verilog. This approach also supports the respective development tools and debugging environments available today to hardware and software engineers. Indications from early adopters of this new technology appear very promising. One such case is discussed.

1. Introduction

There have been few successful attempts to run application software packages on simulated hardware (VHDL or Verilog). Most of the problem has been attributed to a lack of processor performance in hardware simulation, but there is more to making this marriage work than processor cycles. Hardware and software development teams are separated by different tools, specification interpretations and development methodologies. Among the differences in tools are dissimilar human interfaces, which discourages a unified environment for Virtual System Integration or a virtual prototyping effort. Our goal is to provide a unified design verification environment that can be used earlier in the design process than is typical today, enabling designers to find integration level defects before a costly hardware prototype is built. This earlier integration point produces several other benefits, discussed below.

We have chosen to focus our hardware/software co-verification efforts on the growing embedded systems market for several reasons. Some 70-80% of all hardware designers work on embedded systems. The number of software designers is also high; typically, there are four software engineers for every one hardware engineer. An estimated 65-75% of ASICs are designed for use in embedded systems. Reflecting the growth in embedded systems design, the embedded systems tools market is growing currently at over 30% per year, as well. The embedded systems market is an active and growing one, with ample opportunities for Virtual System Integration efforts.

2. Current State of Affairs

The face of embedded systems design is changing rapidly today. Larger and more complicated microprocessors are being used, with an increasing number of designs heading towards a 32-bit architecture. In addition, a larger number of custom ASICs are appearing in these designs. The typical size of software applications running on an embedded design is also growing rapidly. The overall design complexity of these systems, whether measured in ASIC gate count or lines of software code, is estimated to be doubling every two years. In addition, product market lifetimes are decreasing; a personal computer which might have enjoyed a market lifetime of 3 years not so long ago, now may only last 9 months. Hitting the "market window" is now more important than ever before. In short, embedded systems designers are being asked to deliver much more, in less time, with limited resources and at a higher level of quality than ever before.

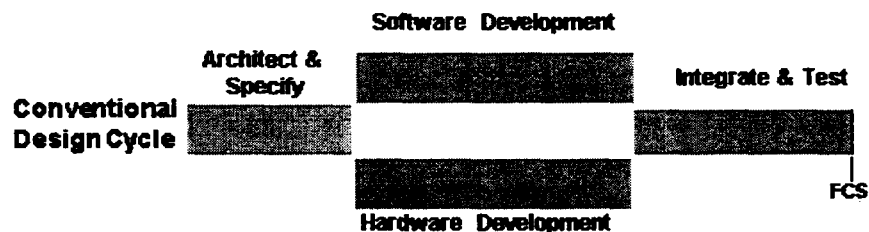


Figure 1. Conventional Embedded System Development Life Cycle

Obviously, it is becoming more important for the embedded systems designer to "do it right the first time." Figure 1 shows a typical embedded system development life cycle. Defects caught in the integration phase are about ten times more expensive to fix than those caught during development. Defects caught *after* product delivery are also likely to be at least ten times more expensive to fix than at integration. A hardware defect, caught at integration or beyond, may cause the redesign/remanufacture of an ASIC, at a cost of anywhere from \$40K to \$250K or more, and require three to twelve months' time. Many times, this ASIC respin is avoided by designing and implementing workarounds in the software application. Figure 2 shows the "real" development life cycle, with the hardware and/or software rework cycles included. In the worst case, a product may contain either reduced functionality or performance, or be shipped weeks to months behind schedule.

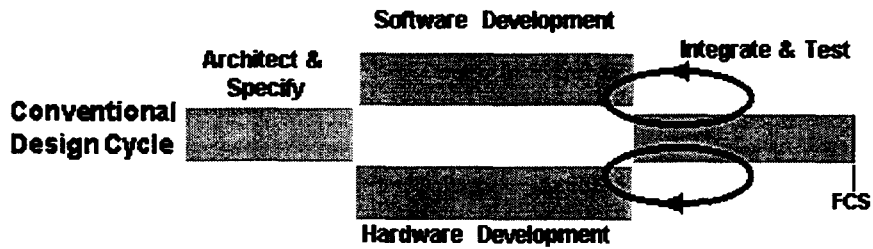


Figure 2. Real Embedded System Development Life Cycle

Figure 3 shows that a significant portion -- very nearly the largest part -- of the development life cycle is typically spent by design teams performing hardware/software integration by debugging their prototype system.

System Specification & Design	HW & SW Design/Debug	Prototype Debug	System Test
37%	20%	31%	12%

Source: Dataquest

Figure 3. Where Design Time Is Spent

Over the years, a number of approaches have been developed to aid teams in their design and debugging of combined hardware/software systems. [1] Figure 4 summarizes some of the more popular approaches.

	Speed	Cost	Silicon?	Debug	ASICs?	uProcessor	SW	Timing
HW Prototype	Fast	High	Yes	Low	Real	Real	Real	Yes
In-Circuit Emulation	Fast	Medium	Yes	Medium	Real	Real	Real	Yes
HW Emulation	Fast	High	No	Low	In FPGAs	Real	Real	Yes
Instruction Set Simulation	Medium	Low	No	SW Only	None	SW	'Real' Reduced	No
HW Simulation	Slow	Low/Med	No	HW Only	HDL	Bus Func HW Model	None/Reduced	No
HW Simulation With/ISS	Slow	Low/Med	No	HW/SW	HDL	Bus Func ISS	'Real'	Locked
Full Software System Sim	Slow Deve High Sim	High	No	Algorithm	C Model	C Model	C Model	No

Figure 4. Hardware/Software Integration Tools and Techniques

Hardware-Based Approaches:

- *Hardware prototypes* -- These appear late in the development process, making significant design changes difficult and expensive. Little to no support exists for software debugging, and hardware debugging requires time and instrumentation.
- *In-circuit hardware emulation* -- This approach replaces the physical processor on the hardware prototype with an emulator, which adds the software controllability and debug visibility lacking in the hardware prototype alone. Hardware visibility is often limited to devices on the processor bus. This approach also suffers from appearing late in the development process.
- *Hardware emulation* -- No prototype is required. Instead, a mock-up of the system is built from reconfigurable hardware components and high-complexity standard ICs. Hardware designers end up debugging the system mock-up, not the system itself. Software support is often very limited. This is an expensive approach, as the entry level cost typically starts at \$90K.

Software-Only Approaches:

- *Instruction set simulators (ISS)* -- These simulate a processor-centric view of the system (instruction execution engine plus a large linear memory for program and data), with very limited support for other system hardware. Timing detail is limited or lacking.
- *Hardware simulators* -- These EDA tools simulate the function, interaction and timing of system hardware at a detailed level. Full functional models of a processor are often too slow (1-4 microprocessor instructions/second) to gain meaningful results of hardware/software interaction. Typically no software support (debuggers, etc.) exists for "executing" the software on the processor model.

- *Hardware simulation with instruction set simulation* -- An ISS is used in place of a full functional processor model, as described above, running in lock step with the hardware simulator. Only modest gains result in software execution speed over hardware simulation alone.
- *Full software system simulation* -- A high-level programming language is used to model both the hardware and software of the target system. This approach is very time consuming, difficult and fraught with errors. In addition, the results are neither directly transferable to the hardware design nor to system integration development phases.

Obviously, none of these approaches has truly solved the hardware/software integration issue. We believe a better approach is *Virtual System Integration*, a unified environment allowing the integration of hardware and software during the design phase.

3. New Unified Integration Environment

Virtual System Integration offers a new approach to hardware/software co-verification, in that it builds a unified integration environment from existing hardware simulators and software development tools and methodologies; both hardware and software designers are working in a familiar environment. The debugging capabilities of these simulators and tools are still available to the designer, as well. Two new tools are added to this new environment, however: the Virtual Software Processor (or VSP) and the Virtual System Integration Console. A VSP actually exists in two halves, one used by the software designer, and the other by the hardware designer. [2] These new tools are detailed below.

3.1 Virtual Software Processors

On the software side, a VSP appears as a library of functions that the software designer uses to build the driver functions of the software application. Typically, an embedded software designer must write a library of driver routines to handle processor specific instructions, such as reading from, or writing to, memory in the hardware design. The VSP library is designed to provide a one for one substitution of VSP routines for the application's driver functions. The VSP library functions are not "stub" routines; they interact with the hardware simulation at runtime. The software application also must make a single call to initialize the VSP before any other VSP library calls are made. The software application is cross-compiled onto a workstation or PC, so that it may execute in parallel with the hardware simulator.

On the hardware side, the entire HDL design description (in VHDL or Verilog) is used, with one exception. The processor model is replaced with one provided with the Virtual System Integration toolset, as either a VHDL entity/architecture pair, or Verilog module. This hardware portion of the VSP resembles the replaced HDL model in all respects, with

an identical set of pins and bus cycle behavior. However, it is implemented not as a fully functional HDL description, but as a foreign language model (a PLI application, in Verilog), written in 'C', which interacts with both the hardware simulator and the software application.

From the user's viewpoint, when the system simulation is started, both the software application and the hardware simulation start executing together. When the software application executes a VSP library "I/O write" call, the hardware portion of the VSP performs the corresponding write in the hardware simulation. Likewise, when a VSP library "I/O read" call is executed, the hardware portion of the VSP performs a read cycle and returns the result to the software application. When a simulated hardware device interrupts the VSP, this interrupt is reported to the software application, where a user-defined interrupt handler is automatically invoked. Multi-VSP simulations are also possible, as each VSP maintains local synchronization with its software program.

3.2 Virtual System Integration Console

Coordinating this co-simulation is the Virtual System Integration console. The console acts as the "launch pad" for starting both the hardware simulator and the software application. It also acts as a simple configuration management tool, allowing users to save and recall different configurations of hardware simulator/design and software application combinations to execute. Once the details of invoking the appropriate hardware and software components of a co-simulation run have been entered and saved via the console, users can recall the saved configuration and restart the co-simulation. This reinforces the unified integration environment approach: hardware design team members are not required to remember unfamiliar details like software application invocation and arguments. Likewise, software design team members do not have to remember the details of how to set up and run the hardware simulation. The console can also monitor the flow of control between the software and hardware sides of a VSP, displaying the results in a number of graphical formats, or as a textual trace.

3.3 Rationale

We believe that Virtual System Integration is a sound approach for several reasons:

The software application is exercising the "real" hardware design; fake "stub" functions and the like are not employed. Integration can begin early in the design process, when the hardware design is still likely to exist at a high level of abstraction (behavioral and RTL level models). This allows for efficient simulation.

The hardware design is tested using "real" software application code, above and beyond any test vectors written by the hardware design team. The software application is cross-

compiled to execute on a UNIX workstation or PC, so its execution is at full workstation/PC speed. It is not interpreted or "executed" directly by the HDL simulator.

The Virtual Software Processor is a new piece of simulation technology. Beyond the library of software routines and the HDL foreign language model described above, a VSP is a new form of an HDL processor model. Consider the interaction between the software, processor and "other hardware" as shown in Figure 5. A high level of interaction exists between the processor and the instruction memory subsystem in any design, as instruction fetching and execution are primary activities of such a system. Generally, a relatively low level of interaction exists between the processor and other hardware on the processor bus, typically in the range of five percent or less of total system execution time.

If the processor and the instruction memory subsystem were to be combined into a single model along with the software application, this model would act as an accelerated software execution engine in an HDL simulator. One can assume the basic memory fetch and execute cycle of the processor has been designed and tested by the hardware design team prior to integration. Why should simulation time be spent performing the instruction fetch and execute cycle over and over again, especially if the design utilizes an off-the-shelf processor? The VSP allows the fetch and execute cycle to be handled by the software application running at workstation speed, thereby freeing the HDL simulator to handle only the direct "other hardware" read and write cycles on demand from the software application. In short, the VSP is running a high level abstraction of the software application against the hardware design, and does so very fast.

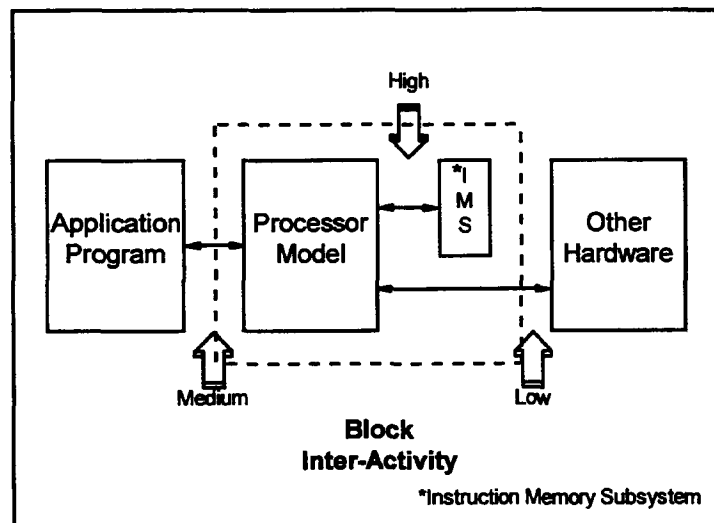


Figure 5. Software, Processor and Hardware Interaction

The observed speedup in simulation runtime comes at a cost, however. In this case, the cost is a loss of timing accuracy. Each VSP is timing accurate at the bus cycle level, but due to the nature of the VSP abstraction (i.e., it subsumes the processor and instruction memory subsystem), not all bus cycles are, in fact, executed by the hardware simulator. For example, the cycles associated with a cache miss and subsequent refresh are not supported, although those cycles can often be approximated with the basic VSP I/O read and I/O write cycles. This was a conscious technical choice on our part, as we believe the benefits of increased simulation speed far outweigh the loss of timing accuracy, especially when the remainder of the hardware design exists at a behavioral or RTL level.

4. Benefits of Virtual System Integration

4.1 Earlier Design Integration

By providing a unified design environment, both a high level hardware design and prototype software can be combined for integration testing much earlier in the development life cycle than ever before. Most importantly, the products of hardware and software design can be integrated before a hardware prototype has been constructed. This can have a significant impact on the system's development life cycle, as shown in Figure 6.

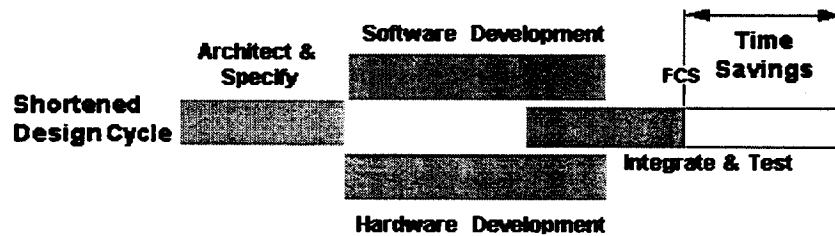


Figure 6. Shortened Development Cycle

By providing an earlier integration point, Virtual System Integration can accelerate the design cycle, thus reducing overall development costs, and saving ASIC turns and software upgrades. A higher quality product can be delivered to market on time and to specification.

4.2 Faster System Simulation

Virtual System Integration can reduce the overall time a design team spends on integration by providing a simulation runtime speedup of well over three orders of magnitude, as compared to a similar full functional HDL model. Figure 7 details a single board computer design, containing a Motorola MC68040 microprocessor (modeled by a VSP), 4KB of ROM, 4KB of RAM, decoder logic, I/O ports, etc. A hardware diagnostic

test suite was written to exercise this design, running a battery of typical tests: ROM checksum test, RAM write/read test, RAM address size test, RAM word size test, and processor interrupt tests.

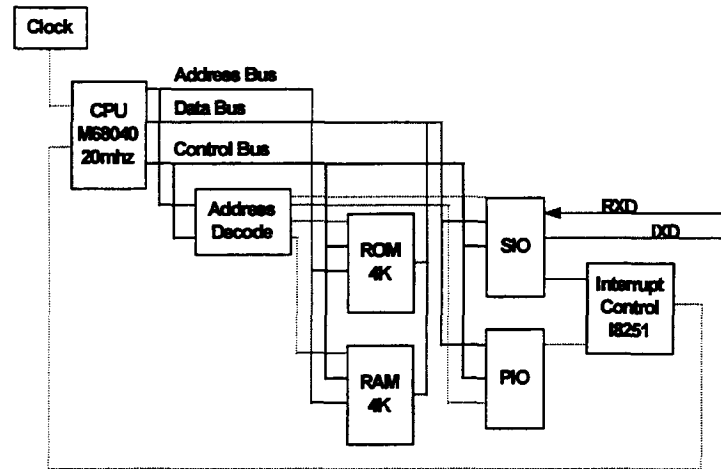


Figure 7. Single Board Computer Design

VSPs gain their main performance advantage by using the HDL simulator as a resource only on demand from the executing software, when the application software must read from or write to a memory location in the hardware design, or service a hardware interrupt. For example, the diagnostic RAM test, above, runs to completion in 83 seconds. This same code was cross-compiled for the MC68040, and the assembly code was examined, yielding an equivalent of 262,000 microprocessor instructions to run the RAM test. This translates to an execution rate of over 3100 microprocessor instructions per second. This code, if "executed" as part of a full functional VHDL design (at about one microprocessor instruction per second) would consume an estimated 72 hours of simulation time.

These numbers become even more impressive when the nature of the diagnostic test suite is considered. The ROM, RAM and processor interrupt tests cause the VHDL simulator to be utilized on a majority of the lines of the software being executed. This is not typical of many embedded system applications which (apart from the instruction memory fetch subsystem, subsumed by the VSP) will utilize these "other hardware" components often no more than five percent of the time. Our diagnostic test suite is therefore a worst case co-verification performance scenario. Further runtime performance may be gained by distributing the execution of the hardware simulation, software application and the Virtual System Integration console across multiple workstations on a network.

5. Early Adopter Results

Several aspects of Virtual System Integration, as described above, are the subject of patent applications. One early adopter of this new technology was a producer of high-performance RAID systems (fault-tolerant, high availability disk servers) in the San Francisco Bay area. Their design included four large ASICs (200K gates), four smaller ASICs (80K gates), two AM29040 processors modeled by VSPs, and a PCI bus model.

A member of the hardware design team said the Virtual System Integration approach allowed him to use a high-level language (in this case, the software application, in 'C') to create stimulus vectors more quickly and easily. This code could also be used for prototype power-up testing and power-on self-test in the final product. Over twenty hardware bugs were discovered and fixed during co-verification testing that were not discovered during initial hardware design testing. A software design team member said he now had a six week window before the hardware prototype arrived to test his hardware diagnostic code by running it against the simulated hardware design. Such a window had never before existed for this design team; this software designer now had the opportunity to verify his code before it was loaded onto the hardware prototype in the integration lab. The team plans to continue using the Virtual System Integration approach in future design and prototype development.

6. Summary

In the authors' experience, the current "state of practice" of co-design is actually one of independent hardware and software design, followed by a long and tortuous prototype integration phase. This is frequently followed by some amount of hardware and/or software redesign and remanufacture, often at considerable expense. Virtual System Integration offers design teams the ability to perform design co-verification, executing actual software application code on the prototype hardware design, earlier and faster than any other approach currently available. This approach provides a unified environment for hardware and software designers, building upon existing hardware simulator and software tools and practices, with no loss of debug capabilities; designers work in the environment they are already accustomed to. We believe Virtual System Integration is the "next step" solution for embedded system designers.

References

- [1] Bunza, Geoffrey J. "A Journey Into Parallel Worlds: Exploring Hardware/Software Systems Integration". *Electronic Engineering Times*, Feb. 5, 1996.
- [2] Wilson, John. "A New Methodology for Co-Design: Virtual System Integration". *Electronic Engineering*, Sept., 1995.