

Simulating Multimedia Systems with MVPSIM

IN DESIGNING AND TESTING

modern, highly complex, high-speed systems, a clear consensus among designers is that simulation plays a critical role in successful system development. Such simulation becomes more important due to the inapplicability of conventional prototyping practices, such as wire-wrapping and breadboarding, to the design and test of high-performance systems. The competitive and fast-moving VLSI, electronics, and computer fields also increase the need for fast, efficient system simulation.

A highly integrated cosimulation approach (see the box) is ideal for such systems, and is particularly beneficial when applied to the design of embedded or special-purpose systems, such as those for multimedia. Generally, these systems have two characteristics in common: strict application-specific requirements and strong interaction between hardware and software components. The goal in designing these systems is to meet application requirements with the optimal allocation of software and hardware resources. Using hardware-software cosimulation, designers can be confident that their system meets its requirements.

JIHONG KIM
YONGMIN KIM
University of Washington

MVPSIM allows highly integrated system level simulations of complex, high-speed multimedia systems. It also aids multimedia software development by providing a uniform working environment to hardware architects and algorithm developers. MediaStation 5000 designers used MVPSIM to compare different architectures, testing and refining their design with various system level scenarios.

MVPSIM can detect system modeling and design errors and provide accurate performance estimates of multimedia systems under development.

Our project

We have spent several years developing a series of multimedia systems using multimedia video processor MVP (also known as the Texas Instruments TMS320C80) as the main computing en-

gine.¹ Each system must meet strict requirements and targets specific applications, such as real-time MPEG video processing, real-time image analysis, and interactive 3D volume visualization. To facilitate system level simulations and build an infrastructure for future projects, we developed a highly integrated simulation environment. MVPSIM supports a unified interface for developing both the hardware and software of MVP-based systems. By using MVPSIM extensively, we successfully completed the first multimedia project in the series, the MediaStation 5000.

MVP overview

MVP^{1,2} is a single-chip multimedia processor with multiple internal processors and a peak performance of over 1 billion RISC operations per second. Its internal architecture is highly parallel at both the chip and individual processor levels. A large on-chip memory (25 modules of 2 Kbytes each) reduces the data transfer overhead with the external memory and devices. To meet the data transfer requirements of multimedia applications, the chip supports up to a 2.4 Gbytes/s communication bandwidth in

Why cosimulation?

Currently achievable device speed, chip density, and design complexity are so high that if a system were not thoroughly simulated, the first prototype system would probably not work as designed. Design errors found after the system's fabrication would require another design iteration, increasing development cost and time to market significantly.

To facilitate simulations, several hardware description languages (HDLs) such as Verilog and VHDL are available. Designers can express their designs at a high level using HDLs. With the accompanying HDL simulators, they can verify their designs and detect logic errors as well as modeling errors.

Once designers simulate every section of the system and find each to be functioning correctly, they can integrate the models for all the sections and perform system level simulations with a wide range of scenarios. This allows designers to test their designs more rigorously at the full-board level and predict the system design performance more accurately than by using back-of-the-envelope calculations and unproven assumptions.

If the models include software components of the system, more extensive, system level, hardware-software cosimulation^{1,2} is possible. Once designers build the cosimulation environment, they can simulate target applications of the system on top of hardware simulation models. This highly integrated approach not only detects subtle design errors that are difficult to find without

close hardware-software interaction, but also produces very accurate system performance estimates.

In addition, the cosimulation environment provides a software development platform for programmers, and naturally encourages interactions among designers and programmers. These early interactions can minimize or eliminate many potential design problems and allow more balanced hardware-software task partitioning.

While system level simulation provides significant advantages to the design team, it also takes time and resources to build the required models. We divide simulation models into three classes: behavioral, structural, and hardware.

Behavioral models represent device behavior at the functional level, usually with high-level program languages or HDLs. Structural models mimic the actual internal structure of a device using HDLs. Hardware models couple an actual device with the simulator.^{3,4} Each type of model differs both in its intended application and cost. For example, behavioral models are relatively simple to develop, but are appropriate only for coarse design verification early in the design and test cycles. On the other hand, structural models take substantially more time to create, but can be used during the system's detailed functional and timing verification stage.

Fortunately, commercial models exist for many common devices, alleviating much of the modeling overhead. For example, the SmartModel Library

and LM-Family Hardware Model from Logic Modeling Corporation of Beaverton, Oregon, offer models for thousands of different components.⁵ The SmartModel Library includes behavioral and structural models.

But if a system contains components that commercial libraries do not support, or components currently under development, the overhead of building accurate models can become a burden. For example, if a high-performance programmable microprocessor is a system's main computing engine, building an efficient and accurate model is not a straightforward task.

References

1. D. Becker, R.K. Singh, and S.G. Tell, "An Engineering Environment for Hardware/Software Co-Simulation," *Proc. 29th ACM/IEEE Design Automation Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1992, pp. 129-134.
2. D.E. Thomas, J.K. Adams, and H. Schmit, "A Model and Methodology for Hardware-Software Codesign," *IEEE Design & Test of Computers*, Vol. 10, No. 3, Sept. 1993, pp. 6-15.
3. D. Giles et al., "Maintaining Simulation Accuracy Through Physical Device Models," *Proc. Int'l Test Conf.*, IEEE CS Press, 1985, pp. 692-695.
4. L.C. Widdoes and H. Stump, "Hardware Modeling," *VLSI Systems Design*, Vol. 9, No. 7, July 1988, pp. 30-38.
5. *Model List*, Logic Modeling Corp., Beaverton, Ore., June 1993.

two ways. It supports this bandwidth among the processors via a crossbar to the multiple on-chip shared-memory modules. The second way is among processors and external memory systems via an intelligent direct memory

access (DMA) controller.

Figure 1 (next page) shows a high-level block diagram of the major functional blocks. The master processor (MP) is a general-purpose RISC processor with an integral IEEE Std 754-compatible

floating-point unit. In typical operating mode, the master processor serves as the main supervisor and distributor of tasks within the MVP. Also, the master processor is preferred for performing high-precision floating-point operations.

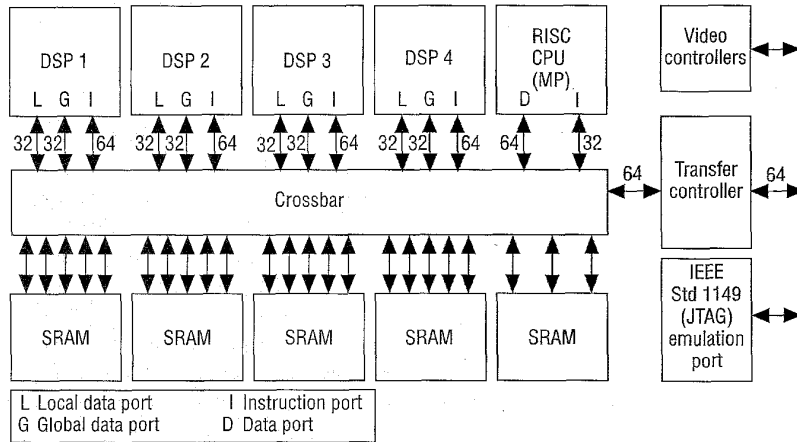


Figure 1. MVP high-level block diagram.

Digital signal processors 1 through 4 have a highly parallel architecture optimized for image and video processing. Each DSP can perform many operations in a single clock cycle via a very long instruction word mechanism.³

A dedicated memory controller with sophisticated data transfer logic, the transfer controller (TC) services the data transfer requests and cache misses of the five internal processors. The video controllers (VCs) provide support for programmable video timing to control both capture and display. The processors and on-chip shared-memory modules are fully interconnected through the crossbar switch network.

MVPSIM

Our primary goal in developing MVPSIM was to provide hardware architects and algorithm developers with a uniform working environment. At the same time, we sought to make it serve the different simulation needs of hardware and software designers efficiently and accurately. For example, software developers generally are not concerned with the system memory controller's detailed timing behavior, which, on the other hand, is of vital interest to hardware designers.

Since the MVP was developed concurrently with our system, no commercial simulation model was available for it. We therefore developed the MVP simulation model by extending an instruction set simulator.

MVPSIM supports a software development environment and a hardware-software cosimulation environment. The first environment consists of the instruction set simulator plus its debugger interface. In the second environment, actual MVP programs serve as the stimulus language driving the hardware simulation models.

Two-level model. To support efficient software development and accurate system level simulation, we divided the MVP model into two levels. For software development, we based the MVP model on an instruction set simulator. In this simulator, most of the master processor and DSP blocks are quite accurately simulated for up to a half-cycle resolution (the MVP operates internally in two phases). However, we have overly simplified the transfer controller and video controller blocks for a fast response in debugging software.

In the cosimulation environment, the MVP model consists of an instruction

set simulator and Verilog models. This hybrid modeling approach provides accurate timing support when necessary, yet allows faster simulation by sacrificing unnecessary details when they are not critical. We extended the MVP instruction set simulator to support the external interfaces of the transfer controller and video controller within half-cycle resolution.

The transfer controller and video controller Verilog models handle timing resolutions finer than a half cycle to generate the MVP's I/O signals according to specification. Users can access the MVP models using the same interface in both environments.

Architecture. Figure 2 shows the overall MVPSIM architecture. In addition to the two environments, MVPSIM includes MVP software development tools. The currently supported tools include C compilers and assemblers for the master processor and DSPs, and MVP linker tools. In the figure, boxes with the bold outlines indicate the software development tools.

MVP programs are compiled, assembled, and linked to executable object files (step 1 in Figure 2). Once these files are prepared, the software development environment can simulate the execution of these programs up to the instruction set level exactly as would occur on a working MVP chip. Detailed timing could differ somewhat (step 2 in Figure 2).

Interactive control is available via the MVP debugger. The debugger can display the master processor's and DSPs' memory and register contents, single-step through instructions, load images into memory, and stop at a specified breakpoint address. After chips became available, we used the MVP emulator for software debugging as well as system testing and integration. Programmers typically spend the most time in this environment, iterating the cycles of coding, simulation, and debugging (steps 1 and 2 in Figure 2). After de-

bugging programs, we can load the same executable COFF (common object file format) files into the cosimulation environment (step 3 in Figure 2).

The cosimulation environment consists of the Verilog XL simulator with MVP extensions, the MVP debugger interface, transfer controller and video controller Verilog models, and other Verilog models for non-MVP system components. The extended Verilog XL simulator combines the original Cadence Verilog XL simulator and the extended MVP instruction set simulator through the C/C++ Verilog glue interface. The MVP dataflow plot helps users understand the program's dataflow characteristics, and the image display feature shows the content of the specified memory block in a separate window.

The user interacts with the environment through two interfaces. In addition to the debugger interface, which is same as the one supported in the software development environment, the Verilog graphical user interface can also control cosimulation using its control commands. While the debugger interface provides functional information (for example, results of a fast Fourier transform operation stored starting in memory location 34000000) on the current simulation, the Verilog graphical user interface is useful for timing verifications.

Figure 3 (next page) shows a snapshot of the MVPSIM environment. The debugger interface initiates simulation, and the user controls the simulation using the Verilog control commands or the MVP debugger control commands. The dataflow plot shows the simulated program's dataflow, and the image display renders the image generated from the requested memory area in a separate window. The Verilog graphical user interface describes the system's detailed timing behavior. The master processor memory window opens from the debugger control window and dis-

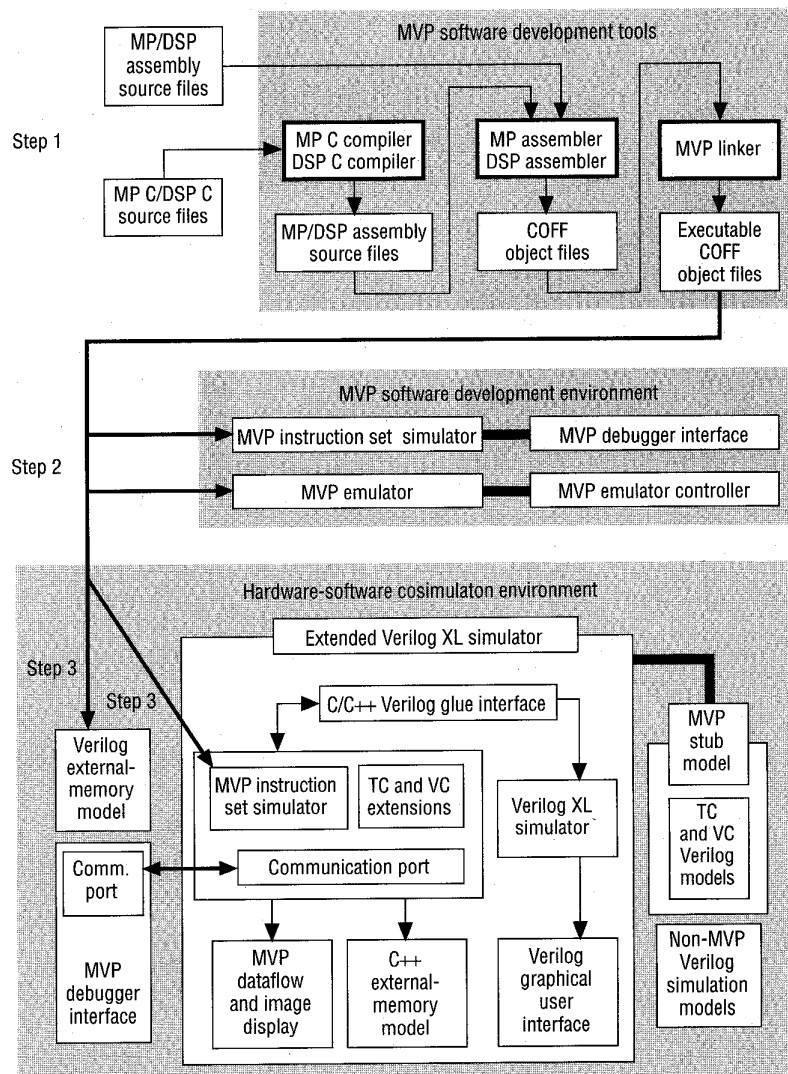


Figure 2. Overview of the MVPSIM simulation environment.

plays the master processor program in assembly format.

Cosimulation environment

We describe each component of the cosimulation environment, the overall architecture of which appears in Figure 2.

Glue interface. Cadence's programming language interface (PLI) utility⁴ supports communications between the

extended MVP instruction set simulator written in C++ and the original Verilog XL simulator. This utility associates a user-written subroutine with a Verilog task name. Invoking a user-defined task in a Verilog program causes the Verilog XL simulator to call the C/C++ function associated with that task name.

These functions are called under several circumstances. One possible configuration is to have the C++ function called whenever we need to change an

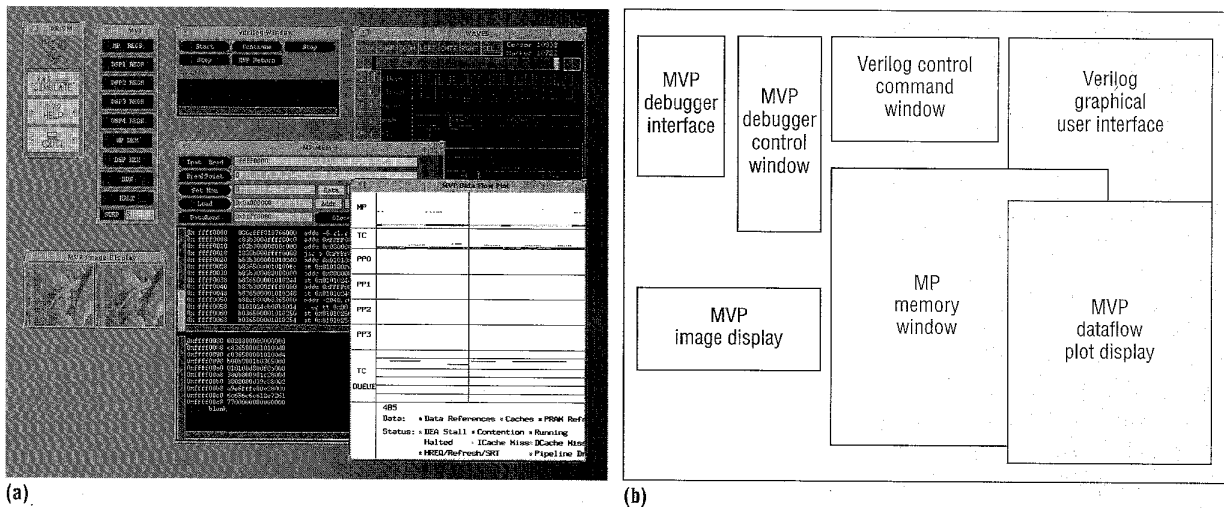


Figure 3. Snapshot (a) of the MVPSIM environment and schematic diagram (b).

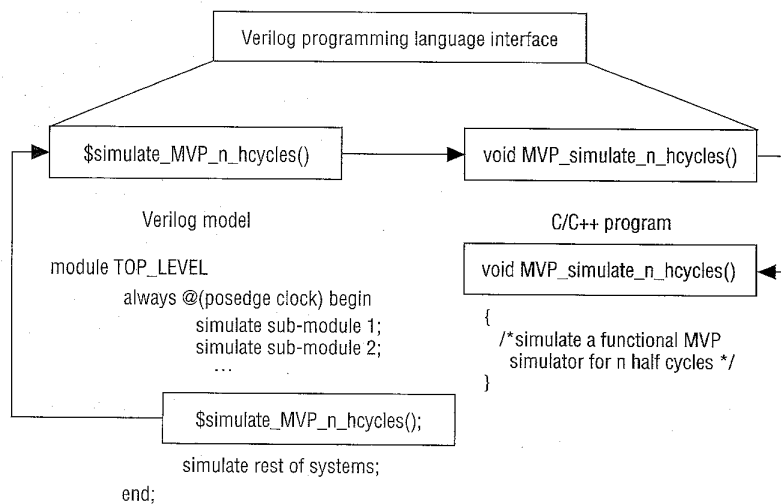


Figure 4. Control flow between Verilog module and C/C++ function.

argument of the Verilog task during simulation. Or the user can explicitly request invocation of the C++ function under specific conditions. Using parameters for the user-defined task allows movement of signal values between the Verilog module and C/C++ function.

Figure 4 illustrates the changes in control flow caused by invoking a user-defined task in a Verilog model. User-defined task \$simulate_MVP_n_

hcycles() is associated with C function MVP_simulate_n_hcycles() by specifying the relationship in the table provided by the PLI utility. Calling user-defined task \$simulate_MVP_n_hcycles() automatically calls its associated C function for execution. After the MVP is simulated for the requested number of half cycles, control reverts back to the Verilog module.

Table 1 lists currently supported user-

defined tasks. Group I tasks initialize the cosimulation environment. For example, \$load_input_file() loads the executable MVP COFF file into the C++ external-memory model of the instruction set simulator. \$set_simulation_flag() sets various simulation options like maximum simulation time, history logging, and so forth.

We use tasks in Group II during actual simulation. \$simulate_MVP_n_hcycles() calls the associated C routine to simulate the MVP for *n* half cycles. The \$write_ext_mem{8/16/32/64}() tasks update the C++ external memory with the data provided. Group III tasks control the transfer controller's monitoring process, while the tasks in Group IV update the video controller registers.

Extended instruction set simulator. The original MVP instruction set simulator did not include a transfer controller external-interface model and a video controller model. We extended the transfer controller simulation model to support external interfaces, added the video controller simulation model, and divided these extensions into two portions. The C++ portion is responsible for the control and arbitration logic,

Table 1. User-defined tasks.

Group	Name	Description
I	\$load_input_file(coff_file_name) \$set_start_address(address) \$init_MVP_simulator() \$set_simulation_flag(option,value)	Load a binary executable file into the MVP simulator Update program's starting location Initialize simulator Set option by value
II	\$simulate_MVP_n_hcycles(n, hreq, ...) \$write_ext_mem{8/16/32/64} (address,data)	Simulate MVP simulator for <i>n</i> half cycles Update C++ external memory with 1/2/4/8-byte bus size
III	\$start_tc_monitor() \$end_tc_monitor()	Start transfer controller monitoring End transfer controller monitoring
IV	\$run_vc0_fclk() \$run_vc1_fclk() \$run_vc0_sclk() \$run_vc1_sclk()	Update Verilog frame timer 0 registers Update Verilog frame timer 1 registers Increment the pixel address in frame memory 0 Increment the pixel address in frame memory 1

and a Verilog model handles the timing-critical portion of the transfer controller and video controller models. We did this to minimize the number of timing-critical signals that pass between the two portions of the simulation models. The MVP C++ simulator operates on a half-cycle resolution; we specify signals requiring finer timing in the Verilog MVP stub model.

Since the transfer controller's external interface extension is one of the key additions that make the system level simulation possible, we give a detailed description of it as an example. (Other extensions are similar in principle, though the simulated operations are different.)

The MVP's host interface is a simple handshake mechanism that allows the transfer controller to share the bus with external devices. A host request is the highest priority request to the transfer controller. Since the data transfer request is serviced in the transfer controller's internal pipeline, the higher priority request cannot preempt the lower priority request until the transfer controller's internal pipeline is empty. The Verilog portion of the transfer con-

troller model samples the host request from external devices. It passes the sampled value to the transfer controller model's C++ portion through the hreq_ signal in the \$simulate_MVP_n_hcycles() task. Once hreq_ is active, the transfer controller starts to drain its internal pipeline, allowing operations already in the pipeline to finish, but not placing any new requests into the pipeline. When the pipeline is completely empty, the transfer controller model's C++ portion asserts hack_ signal to the Verilog portion. The Verilog portion then generates the actual host-acknowledge signal with the required timing delay.

MVP stub Verilog model. This model defines the MVP Verilog module, which provides the interface for instantiating the MVP model in board-level modeling. In the module, the transfer controller and video controller Verilog modules are instantiated, and these modules communicate with their counterpart C++ functions through the Table 1 user-defined tasks. The transfer controller Verilog module controls the host interface and memory interface signals

based on the parameters passed from the C++ portion of the transfer controller model. Communication between the transfer controller Verilog module and its C++ function are supported through the \$simulate_MVP_n_hcycles() task.

The video controller Verilog module generates all video timing signals, and we use Group IV tasks in Table 1 for communication between this module and its C++ function. Figure 5 (next page) uses a simple board example to describe the hierarchical structure of the MVP stub Verilog model up to the level of user-defined tasks. Calling the MVP module interface in the Verilog board module BOARD instantiates the MVP. The modules located below the dashed line in Figure 5 are supported from the MVP stub Verilog model.

Debugger interface. We based the debugger interface on the PRISM (Portable Retargetable Instruction Simulator for Multiprocessor) architecture, which provides a flexible interface building environment in the X Window System.⁵ Using PRISM's configuration capability, we can quickly build an easy-to-use interface for running multi-

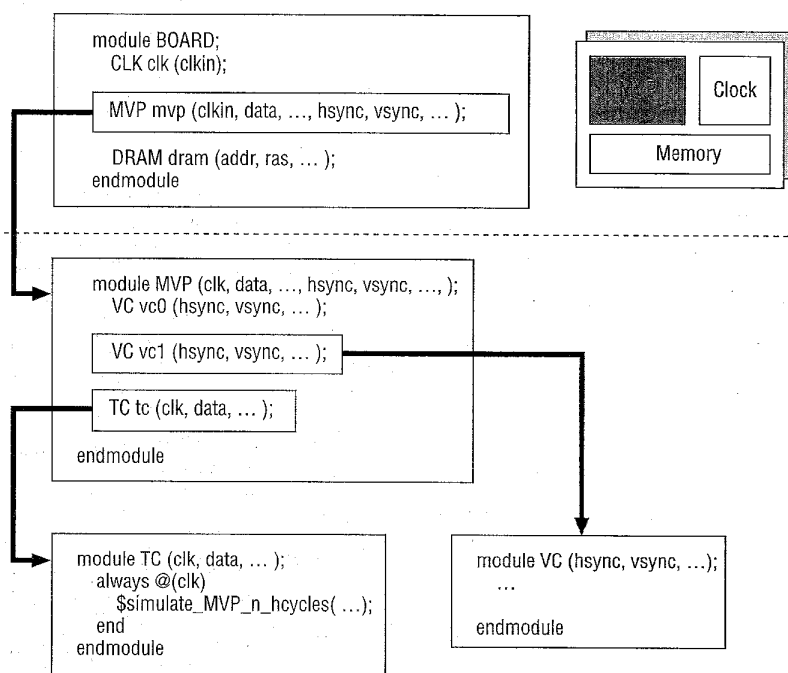


Figure 5. Hierarchical structure of MVP stub Verilog model.

ple simulators concurrently. The debugger interface is the same under both simulation environments. Once PRISM starts, it displays a simple window loading the configuration file for the MVP simulator, and creates a Unix process used in the extended Verilog XL simulator for the cosimulation. It also creates the MVP instruction set simulator in a separate process. Command buttons control the simulation and read the MVP's various registers and the system's external memory.

The debugger interface connects to the Verilog XL simulator (or MVP instruction set simulator) through the named-pipes interprocess communication of the Unix system. The named pipe is created when the interface generates a child process, which, in the cosimulation environment, is the extended Verilog XL simulator. The pipe is bidirectional so that messages can travel back and forth. Commands from the interface go to the MVP simulator, and

results return to the debugger interface through the pipe.

External-memory models. As shown in Figure 2, the instruction set simulator in the cosimulation environment has its own C++ external-memory model. Since hardware designers also provided a Verilog model for external-memory systems, there exist two copies of the same external memory. This redundant memory modeling allows the easy detection of system design and modeling errors because the transfer controller external-interface model checks for inconsistencies between the two memory models during cosimulation. For example, if the memory controller of a system does not respond correctly to an MVP read request, the inconsistency can be easily checked by comparing data from the C++ memory model with that from the Verilog model.

We can also detect inconsistent data and inconsistent memory parameters.

Detecting either type of inconsistency causes an error to be reported back to the user and halts cosimulation. Inconsistent data is detected when the contents for the same memory location in the two memory models differ. When the MVP reads data from external memory, the transfer controller external-interface model checks whether data from the two memory models are the same.

Inconsistent memory parameters arise when the two models have different memory parameters for the same location. The transfer controller model checks for parameters like memory type (for example, 2 cycles per column memory), page size (a 1-Kbyte page), and bus size (a 32-bit bus). Memory parameters are checked during row-time access. For the C++ memory model, file system.mmap specifies memory parameters. For the Verilog model, the parameters are embedded in the Verilog program.

Setting up the cosimulation initializes both memory models with the same contents. For example, when an MVP program is loaded into the cosimulation environment, the same contents are loaded into the C++ and Verilog memory models. The C++ model assumes a contiguous memory configuration, so the executable MVP program can be loaded directly.

Various possible memory configurations are available for the Verilog model. For example, to organize a 64-bit-word memory system, we can use multiple 4- or 8-bit memory devices. If several small-size devices organize a long memory word, we may need to split the executable MVP program into multiple files before loading to the Verilog memory model.

We have developed a tool to split the executable MVP program into several Verilog memory-loadable ASCII files. In addition, when the memory update is initiated by system components other than the MVP, such as peripheral de-

vices, we need to update the C++ memory model, too. To support this kind of memory operation, we provide user-defined tasks `$write_ext_mem8()`, `$write_ext_mem16()`, `$write_ext_mem32()`, and `$write_ext_mem64()`, as listed in Table 1.

Example. Once we establish the complete cosimulation environment by integrating non-MVP Verilog models with MVPSIM, we can simulate various MVP programs on top of it.

To start, the user initializes a small portion of Verilog code. We describe the initialization steps as an example of using the cosimulation environment. Figure 6 shows the initialization file typically used for system level simulation. After the MVP program is compiled, assembled, and linked, the executable COFF file (application.coff in this example) is produced.

We use `$set_simulation_flag()` twice in Figure 6 to set simulation options. The first use limits the total simulation time to less than 2,000 seconds to avoid a runaway process. The second use causes display of the dataflow plot and image display windows. The `$load_input_file("application.coff")` task loads the MVP program into the C++ memory model. Then the same program initializes the Verilog memory model, but how it does so varies depending on the external-memory configuration. The `$set_start_address()` task sets the program's starting address for the MVP simulator while `$init_MVP_simulator()` initializes it.

Using MVPSIM

MVPSIM proved instrumental in the development of the MediaStation 5000 (for an overview of the system, see the box, next page). We constructed the complete system level simulation environment using MVPSIM and Verilog models for non-MVP system components. MVPSIM's cosimulation capability facilitated simulation of many target

application algorithms on top of the MediaStation 5000 hardware simulation models. Application algorithms tested include those for MPEG compression and decompression, median filtering, and convolution.

Case study

Our work on MediaStation 5000 took 4 years and 30 man-years. It helped us refine the design of MVPSIM.

Hybrid design approach. This approach to building the cosimulation environment enabled us to build it quickly (in about four man-months) using the existing MVP instruction set simulator. This produced a relatively fast cosimulation environment due to the fast C++ instruction set simulator. Reusing most of the MVP instruction set simulator code minimized the environment-building overhead. The current environment with the full set of MediaStation 5000 simulation models can simulate about 30 cycles per second on a Sun Sparcstation 10. It takes about 15 hours to fully simulate 1/30 of a second, which is a single frame time in MPEG and H.261. This speed was adequate for most of our system level scenarios.

Accuracy. The cosimulation environment provided hardware designers with a very accurate picture of the system under development. They gathered realistic system performance estimates and optimized the system architecture. For the MediaStation 5000, real-time

```

module SETUP;
  initial begin
    $set_simulation_flag("stime",
      "2000", "s");
    $set_simulation_flag("graphics");

    $load_input_file("application
      .coff");
    // Verilog memory models should
    // be initialized
    // with the same data as
    "application.coff".

    $set_start_address(32'hfff0000);

    $init_MVP_simulator();
  end
endmodule

```

Figure 6. Cosimulation setup file.

MPEG video compression and decompression is a major application. An early architecture dedicated a fair amount of hardware resources to fast data movement during MPEG encoding. But after extensive system level testing, we concluded that the performance requirements could be met with much less hardware. Based on the cosimulation's performance estimates, we made the MediaStation 5000 more efficient in every revision.

We measured actual performance after the system was fabricated and made functional. Table 2 lists the results for several representative algorithms and shows that simulation estimates closely match actual system performance. For example, with a 20-ns cycle time, a 3x3 arbitrary-kernel convolution operation on a 512x512-pixel image took

Table 2. Execution time comparison between the MediaStation 5000 and system level simulator for a 512x512 image.

Algorithm	Actual (cycles)	Simulation (cycles)
3x3 convolution	938,398	938,874
3x3 median filtering	554,131	573,724
Window and level	347,746	355,700
Histogram equalization	636,539	659,946

MediaStation 5000 overview

The MediaStation 5000 system resides on a VESA local bus/ISA form factor card and transforms a personal computer into a programmable high-performance multimedia workstation.^{1,2} It offloads computing-intensive multimedia tasks from the host computer using an MVP, custom programmable logic devices, and other supporting chips. It supports real-time MPEG compression and decompression,³ still picture processing (such as JPEG⁴), and image processing (such as convolution, Fourier transform, contrast enhancement, and geometric transformation), as well as 2D and 3D computer graphics.

The workstation combines a powerful computing engine, high-bandwidth data transfer, and support for programmable display resolution (1,024×768 in 16 bits/pixel or 1,280×1,024 in 8 bits/pixel). This makes the MediaStation 5000 a good add-on board for medical imaging and telemedicine workstations as well as multimedia authoring, desktop audio/video teleconferencing, industrial machine vision, desktop publishing, scientific computing, and simulation and animation.

Figure A shows the block diagram of the MediaStation 5000. Based on a single-bus architecture, the MVP is the core computing engine of the system while the memory and peripheral controller (MPC) arbitrates data transfer requests among different system components and controls various peripheral devices. The video decoder digitizes incoming video signals, which are stored into the

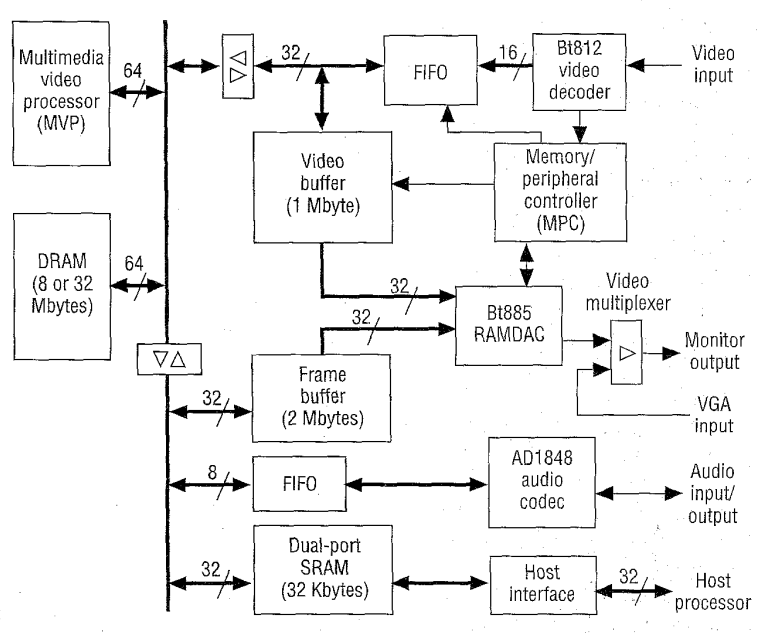


Figure A. MediaStation 5000 system overview.

video buffer, processed in the MVP, and sent to the RAMDAC (random access memory digital-to-analog converter) for display. CD-quality audio signals are digitized by the audio codec and processed in the MVP as well. The dual-port SRAM and host interface logic provide bidirectional communication between the host personal computer and MediaStation 5000.

References

1. D. Kim et al., "A Real-Time MPEG

Encoder Using a Programmable Processor," *IEEE Trans. Consumer Electronics*, Vol. 40, No. 2, May 1994, pp. 161-170.

2. W. Lee et al., "MediaStation 5000: Integrating Video and Audio," *IEEE MultiMedia*, Vol. 1, No. 2, June 1994, pp. 50-61.

3. D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. ACM*, Vol. 34, No. 4, Apr. 1991, pp. 46-58.

4. G.K. Wallace, "The JPEG Still Picture Compression Standard," *Comm. ACM*, Vol. 34, No. 4, Apr. 1991, pp. 30-44.

about 18.8 ms in simulation as well as on the actual system.

Small differences between the actual and simulation times are due to the hybrid modeling of the MVP. One factor contributing to this difference is the mod-


eling of the transfer controller's internal-pipeline drainage mechanism. To make the extension of the MVP instruction set simulator simple, detecting the empty transfer controller pipeline takes several cycles more than necessary.

Dual external-memory model.

The redundancy of these models and the automatic detection of inconsistent memory conditions caught many system modeling and design errors. We detected not only memory system errors

but also most data path modeling errors. For example, the automatic checking mechanism detected a wiring error between the video FIFO buffer and the video transceivers several days before we sent the printed circuit board design out for fabrication. If the error had gone undetected, the returned circuit boards might have required modifications during system debugging.

Prototype testing. Due to extensive system level simulation, prototype testing was very smooth. In spite of complex arbitration logic, multiple request sources, and sophisticated dataflow and processing mechanisms, we did not experience a single logic error during system testing. In fact, there was one logical error in the MediaStation 5000, but it resulted from an MVP specification error. Since the MVP and MediaStation 5000 developed concurrently, our MVP documentation did not include up-to-date modifications, and one timing-specification error existed in an early version of the MVP's manual.

OUR POSITIVE EXPERIENCE using MVPSIM in developing the MediaStation 5000 system strongly suggests that it is possible to build a highly integrated simulation environment with reasonably low overhead. This experience also indicates that the benefits of using such an environment throughout system design and test cycles far outweigh the expense. We have developed two more image computing systems with MVPSIM, and several companies have used it to design MVP-based systems. We are also extending MVPSIM to efficiently support multiple MVPs and to add new functions such as support for the MVP program execution profile. 

Acknowledgments

Our sincere thanks go to Paul Fuqua of Texas Instruments, Robert Gove of Compression Labs, Inc., and Walt Bonneau of Sony for their input and timely assistance in developing MVPSIM.

References

1. K. Gutttag, R.J. Gove, and J.R. Van Aken, "A Single-Chip Multiprocessor for Multimedia: The MVP," *IEEE Computer Graphics & Applications*, Vol. 12, No. 6, Nov. 1992, pp. 53-64.
2. R.J. Gove, "The MVP: A Highly-Integrated Video Compression Chip," *Proc. Fourth IEEE Data Compression Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 215-224.
3. J.A. Fisher, "Very Long Instruction Word Architectures and ELI-512," *Proc. 10th Symp. Computer Architecture*, IEEE CS Press, 1983, pp. 140-150.
4. *Programming Language Interface Reference Manual*, Vols. 1 and 2, Cadence Design Systems Inc., San Jose, Calif., 1992.
5. T.Y. Sinn, *A Portable Retargetable Instruction Simulator for Multiprocessors*, master's thesis, Image Computing Systems Laboratory, Dept. of Electrical Eng., Univ. of Washington, Seattle, 1994.



Jihong Kim is a PhD candidate in the Department of Computer Science and Engineering at the University of Washington. His research interests include image computing, multimedia systems, algorithm design and software tools, performance analysis, real-time systems, and simulation. Kim re-

ceived a BS in computer science and statistics from Seoul National University and an MS in computer science and engineering from the University of Washington. He is a member of the IEEE Computer Society and the ACM.



Yongmin Kim is a professor of electrical engineering and an adjunct professor of bioengineering, radiology, and computer science and engineering at the University of Washington. His research interests are in multimedia, image processing and computer graphics, medical imaging, advanced workstation design, and modeling and simulation. Kim holds a BS in electronics engineering from Seoul National University, and an MS and PhD in electrical and computer engineering from the University of Wisconsin. He received the Early Career Achievement award from the IEEE Engineering in Medicine and Biology Society in 1988. He is a senior member of the IEEE, and a member of the Computer, Engineering in Medicine and Biology, Signal Processing, and Education Societies. Kim also serves as an Accreditation Board of Engineering and Technology (ABET) evaluator for computer engineering.

Address questions or comments about this article to Yongmin Kim, University of Washington, Department of Electrical Engineering, Box 352500, Seattle, WA 98195-2500; kim@ee.washington.edu.