Using microcode in the functional verification of an I/O chip

S. P. Goldman
L. M. Mohr
D. R. Smith

The IBM pSeries® clustered and parallel processing systems require high-speed, low-latency communication among processor nodes. The 2-Link Switch Network Interface and 4-Link Switch Network Interface for the pSeries High Performance Switch are the adapters which provide the communication infrastructure for the pSeries p655 and p690 servers. A unique approach was used during the functional verification of these adapters that yielded benefits over the methodology used for the previous-generation product—the SP™ Switch2 Adapter. The approach used on the Switch Network Interface introduced the concept of using microcode during the functional verification process. This paper gives an overview of functional verification, followed by a description of the SP Switch2 Adapter and the Switch Network Interface. The verification methodologies used on these adapters are described and compared. Finally, the benefits of implementing hardware/software co-verification on the Switch Network Interface throughout the development cycle are described.

Introduction

In clustered and parallel processing systems, high-speed, low-latency communication among processor nodes is essential. The hardware to support this high-performance network for the IBM pSeries* p655 and p690 servers consists of adapters within the node complex and external switches. The 2-Link Switch Network Interface and the 4-Link Switch Network Interface for the pSeries High Performance Switch (HPS) play a key role in these systems, because they offload much of the communication workload from the processor nodes. Each node connects to the pSeries HPS through a Switch Network Interface (SNI), as shown in **Figure 1**.

The SNI enables high-speed communication among servers. Each server can contain multiple adapters (SNIs) that communicate with one another over the network. Data is transferred between servers via message-passing protocols implemented through a combination of hardware and software. To send information between servers, the software issues tasks to the hardware [or, more specifically, the Switch Interface (SI) chip on the SNI], which then sends the data to the appropriate destination server.

The SI chip is an application-specific integrated circuit (ASIC) chip and the primary component on the SNI. A special-purpose processor within the SI chip is driven by microcode. This paper focuses on the chip-level verification of the SI chip and how inclusion of the microcode in the verification environment decreased escapes of errors into the hardware and improved development efficiency and overall time to market compared with those of the previous product.

The first section of this paper is an overview of functional verification and the techniques that are commonly used. Next, the hardware-only functional verification of the chips on the previous adapter is discussed. The third section describes the architecture of the SI chip. This is followed by a section which describes the functional verification environment used to test this chip. The benefits of using hardware/software co-verification for the SI chip are summarized in the last section.

Functional verification overview

Before a chip is fabricated, many types of verification are performed to ensure that the chip functions properly. These include technology rules checks, timing analysis,

0018-8646/05/$5.00 © 2005 IBM

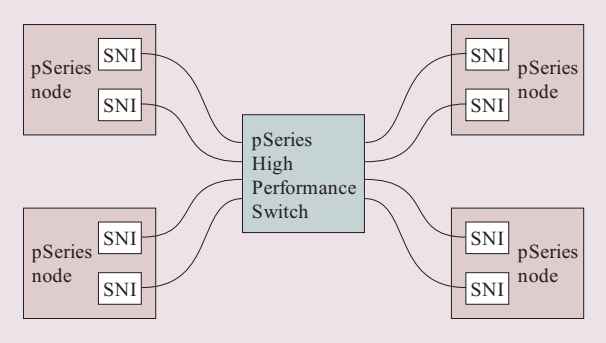

Figure 1

pSeries system view.

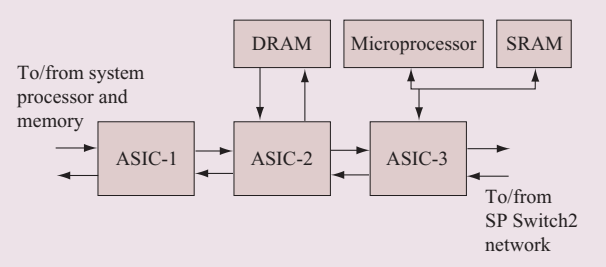

Figure 2

Block diagram of the SP Switch2 Adapter.

and functional verification, which is the focus of this paper. Functional verification is concerned with the validation of all of the chip functions in normal operating modes as well as after an error condition. If a chip or system design contains a processor or a sequencer, hardware/software co-verification may be used to verify system operation before the hardware is manufactured. In hardware/software co-verification, the verification environment includes the software, firmware, or microcode that executes on the processor or sequencer, as well as the model of the chip hardware.

Functional verification may be performed at the unit, chip, subsystem, and/or system levels. A unit is a logical partition of the design which performs a specific function. For example, the portion of the design which provides an interface to an external bus might be considered a unit. A chip comprises units and is an entity that can be fabricated in silicon. A subsystem is a collection of two or more chips that communicate directly with each other. In functional verification, a system is a collection of two or more chips including a processor chip and memory. The focus of the testing is different at each level. In unit verification, testing is targeted toward the function of an individual block. This can be accomplished through various verification methods, including formal verification, which can provide full proofs of functional properties; deterministic tests, in which each separate test targets a specific operation; random tests, in which operations are mixed in a random manner; and biased random testing, which allows the user to control the randomness for better coverage. As verification moves to environments that include more of the design hierarchy (unit to system), the testing is targeted toward interconnections and interactions between the smaller pieces. At the chip level, where all of the functional blocks of the chip are integrated, random verification is often used to create complex test scenarios to comprehensively test the chip. In subsystem verification, multiple chips are included in a verification environment to test their interoperability. For I/O chips, such as the SI chip, system-level verification includes processors, memory, and the I/O chips [1].

For the chip-level verification of the SI chip, a biased random transaction-based simulation environment was used. The SI chip simulation model was built from the Hardware Description Language (HDL) design, and a cycle simulator was used. This is similar to the environments used on recent pSeries [1] and zSeries* processor chips [2] and I/O chips. The object-oriented C++ simulation code consists of driver code, which presents stimuli to the chip, and monitor code, which predicts and checks the behavior of the chip. Both the driver and monitor code operate "on the fly," creating inputs and checking results every simulation cycle. Parameters are used by the driver to bias the random generation of the transactions that are driven into the SI chip [3]. These parameters allow the verification engineer to focus the testing toward required scenarios.

The SP Switch2 Adapter and its verification environment

To better understand the advantages of the SI chip verification methodology, it is useful to review the design and functional verification of the previous product. The SP Switch2 Adapter, the previous-generation message-passing adapter, provided connectivity between nodes, just as the Switch Network Interface does, and it fit into the system just as the SNI does in Figure 1. The SP Switch2 Adapter contained three special-purpose ASIC chips, a PowerPC* microprocessor, and static random access memory (SRAM), as shown in **Figure 2**. ASIC-1 served as an interface to the system processor bus. ASIC-2 bridged the other two ASICs and provided access to dynamic RAM (DRAM) storage on the adapter which was used to gather the message data that was being sent or received over the SP Switch2 network. ASIC-3 interfaced with the PowerPC microprocessor and SRAM and provided the links to the SP Switch2

network. In system operation, firmware running on the microprocessor controlled the flow of message-passing traffic between the system processor/memory complex and the network.

In the functional verification of the SP Switch2 Adapter, each of the ASIC chips was initially tested separately; the three chips were then combined into a larger verification environment. Since the chips were developed well in advance of the firmware, the chip verification environments could not include any of the firmware. Because there was no firmware, it was not necessary to include the PowerPC microprocessor in the simulation models. A C++ bus functional model was used to emulate the behavior of the bus at the PowerPC microprocessor interface. The simulation driver essentially replaced the firmware function in the adapter subsystem by randomly generating the commands needed for message-passing operations and sending them to ASIC-3. After the ASIC chips were fabricated and the first SP Switch2 Adapters were built, testing began. When the firmware became available, additional releases of the chips were required to change the chip designs to work better with the firmware in the system. Each chip release required additional functional verification prior to its release and additional testing after the chips were manufactured. As a result of this experience, the chip microcode was designed as an integral part of the primary verification environment for the new SI chip.

SI chip overview

The SI chip is the new hardware that enables system software to implement message-passing transactions between IBM pSeries p655 and p690 servers. The software makes requests to the hardware to send data from one server to another. Each of these requests, which are known as descriptors, is placed in a descriptor list which exists in memory on the server. Multiple descriptor lists can be processed simultaneously by the SI chip. Descriptors can exist on the source server, on the destination server, or on both source and destination servers. Once the software issues an indication that everything has been prepared for a message-passing transaction, the SI chip at the source uses the information in the descriptor to obtain data from system memory and send it through the network to the SI chip in the destination server. The destination SI chip uses a descriptor to determine where to write the data in the memory of the destination server. Information sent through the network is contained in direct memory access (DMA) packets that are constructed within the SI chip. The DMA packets consist of two parts: The first part is a header that provides control information to the destination node, and the second part is the actual message data that is being transferred, also referred to as payload data.

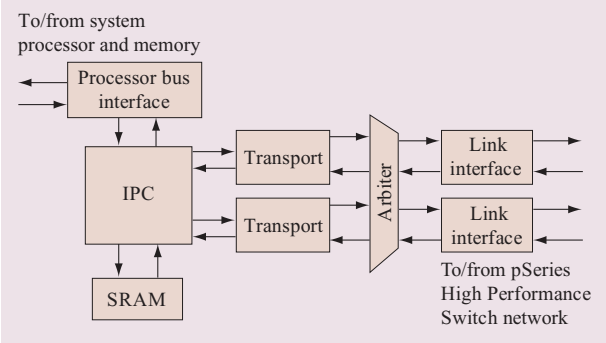

Figure 3

Block diagram of the SI chip.

Figure 3 is a block diagram of the SI chip. The processor bus interface logic handles the communication between the chip and the server processor complex. Data flow within the SI chip is controlled primarily by the inter-partition communication (IPC) block. The IPC is a new custom-designed block that contains a 64-bit arithmetic and logic unit (ALU), a sequencer, and a hardware dispatch unit. Together, these units process instructions contained within microcode that is loaded in an on-chip SRAM. This microcode was newly developed specifically for the IPC. The IPC determines which data it wishes to retrieve from the memory on the server and sends the appropriate requests to the processor bus interface. It then controls the construction of the DMA packet to be sent over the link. The transport logic directs the message to one of the two link interface ports, and the arbiter controls the flow of data from the two transport units to the ports. The reverse process is followed when a message is received from the network (i.e., entering from the right side of Figure 3).

Since most of the control functionality within the SI chip is implemented in IPC microcode, rather than with a hardware state machine, the SI chip provides a highly programmable environment. This approach makes it easier to implement functionality that would alternatively have resulted in overly complex and inflexible hardware. The ability to modify the microcode can be useful for adapting to alternate message-passing protocols.

SI chip functional verification

The verification environment for the SI chip comprises a hardware model, the C++ verification code, IPC microcode, and simulation parameters. The hardware model is a subsystem model made up of two SI chips connected by a cable macro that allows different cable lengths to be simulated between the chips. **Figure 4**

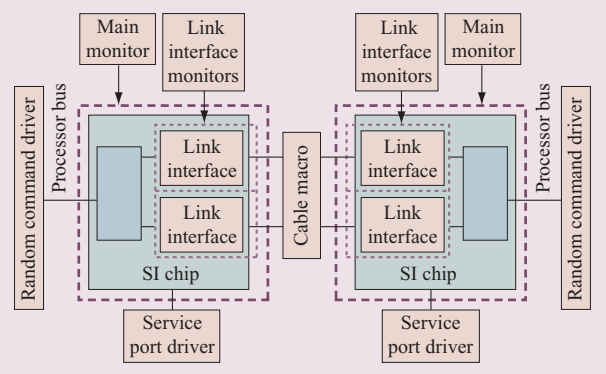

Figure 4

Functional simulation environment of the SI chip.

illustrates how the verification code logically works with the model. Each chip has a random command driver for the processor bus interface, a service port driver, and a chip monitor. There is also a small monitor at the link interface which provides an additional level of internal checking and failure isolation. The presence of the IPC in the model is important in that this allows its microcode to be loaded before simulation begins, permitting hardware/software co-verification. This ability to load and run the microcode is a key difference between this methodology and the verification methodology used for the SP Switch2 Adapter. Finally, a set of control parameters is used to guide the test case. The entire environment utilizes the SimAPI programming interface [4] in conjunction with a high-performance cycle simulator [5].

The random driver handles all interactions with the model for both message-passing and non-message-passing operations. The latter include actions for service and interrupts. For message-passing operations, the driver creates message-passing requests to be sent to an adapter. These result in specific hardware commands being placed on appropriate facilities in the simulation model, initiating further processing by the chip. This in turn leads to requests exiting the chip, which are also handled by the driver. The chip that sends packets across the link makes outbound requests to the driver for descriptor data, source payload data, and address-translation data. Address-translation data is required when the addressing mode of a request indicates the use of virtual addresses (instead of real addresses) for data locations in memory. For a chip that is receiving packets, the requests to the driver can be to supply data (e.g., descriptor data or address-translation data) or to write payload data to memory. Whenever necessary, the driver provides appropriate handshaking responses, for example returning a "write done" acknowledgment after payload data is written to memory.

To complete the robust checking required in the verification effort, two levels of checking are incorporated in the environment. First, the driver includes code to check all traffic over the processor bus interface in order to ensure adherence to proper processor bus protocol. Second, a major benefit is derived from the main chip monitor, which examines all inputs to the chip in order to predict and then verify that the correct output occurs.

To guide the various testing situations required, the driver uses a large number of simulation parameters and controls. The vast majority of these are maintained in one base file. A set of secondary parameter files contains overrides to the base set to produce different test scenarios. During the early stages of development, a number of specialized controls were added to the base and secondary files to restrict various aspects of the testing rather than allowing them to be completely random. For example, special controls were put in place to allow only one or two descriptor lists to be processed at any one time, and to cause activity on these lists to be serialized instead of possibly overlapping. The general controls, which are more widely used than the special controls, take the form of probability tables that are used to create random biasing both within a single test case and across many test cases. Some examples of the probability table controls used include types of commands, frequency of commands, and amount of payload data.

The microcode used in the SI chip verification environment was developed incrementally to be used for final-release-level microcode (i.e., to be shipped with the product). The source code written by the microcode developers utilized the C programming context. It was written in a macro- and function-oriented manner, and as such does not have the outward appearance of many C program source files. An example portion of the source code for an error-handling routine appears in **Figure 5**.

When the source code is compiled, several output files are produced. One of these is the listing file, which is a key source of information for use in debugging microcode and hardware problems. The listing file uses both hexadecimal and textual representations of the actual microcode instructions and notes where they exist in relation to each other. The portion of a listing file resulting from compilation of the error-handling routine source code noted above is shown in **Figure 6**. The microcode compile process also creates a model initialization file (MIF) to be used in simulation. This file contains the microcode instructions, along with information about where the instructions are to be loaded in the IPC SRAM. The MIF is loaded into the model in a single simulation cycle just prior to the start of testing. Each line of the MIF contains a model SRAM facility name, followed by array-element and bit-position

```
label(R_ERR_ACTIV_IA_R);

  XORI08_B_ACTIVE_IA(D_ALU, A_GR_AF, R_ERR_TYPE_WIN_KEY, R_ERR_ACTIV_IA_R);

    cmt("Update counter in GPR for the dropped packet");
    win_state_tab_entry.dw = 0;
    win_state_tab_entry.bit.receive_packets_dropped = ALL_ONES_LL;
  ADDMODI08_M(D_GR_AD, A_GR_AD, win_state_tab_entry.dw, 0x1);

    win_state_tab_entry.dw = 0;
    win_state_tab_entry.bit.key_mismatch_count = 1;
  ORI08_M_BNE(D_ALU, A_PASS_MASK_VAL, win_state_tab_entry.dw, 0, R_ERR_WR_BAK);

    cmt("Increment key_mismatch_count");
    win_state_tab_entry.dw = 0;
    win_state_tab_entry.bit.key_mismatch_count = ALL_ONES_LL;
  ADDMOD_M(D_GR_AD, A_GR_AD, win_state_tab_entry.dw, B_ALU);
```

Figure 5

Example of an error-handling routine in a microcode source file.

```
R_ERR_ACTIV_IA_R:
 0361   4e38007e  XOR08_B_ACTIVE_IA  D_ALU  A_GR_AF  R_ERR_TYPE_WIN_KEY  R_ERR_ACTIV_IA_R
                                          Mask 00000000ffffffff (H = 32 L = 63 n = 0)
//Update counter in GPR for the dropped packet
 0362   5ce8107e  ADDMODI08_M        D_GR_AD  A_GR_AD  win_state_tab_entry.dw  0x1
                                          Mask 0000000100000000 (H = 31 L = 31 n = 0)
 0363   46380fbe  ORI08_M_BNE        D_ALU  A_PASS_MASK_VALUE  win_state_tab_entry.dw  0
                                           R_ERROR_WRITE_BACK
//Increment key_mismatch_count
                                          Mask ffffffff00000000 (H = 32 L = 63 n = 1)
 0364   1ce8107f  ADDMOD_M           D_GR_AD  A_GR_AD  win_state_tab_entry.dw  B_ALU
```

Figure 6

Example of post-compile listing for a microcode error-handling routine.

information used to identify exactly where the data is to be loaded. Following the SRAM and position information is the hexadecimal microcode instruction data.

Before providing the microcode to the SI chip verification environment, the microcode developers used a special model to perform some initial testing. This model was a single SI chip with its link outputs connected to its link inputs, referred to as a wrapped model. The wrapped model was also used to test the microcode bootstrap load process. In the laboratory and customer environments, the microcode is loaded into the SI chip using a bootstrap process in which the microcode is sent from system memory to the SI chip. This happens quickly in real time, but it takes too many simulation cycles to do this for every functional test case. In light of this, the

bootstrap load process was verified separately from the random functional testing for the SI chip. In the SI chip verification environment, a shortcut process is used in which the microcode is loaded from the MIF.

To properly stimulate the hardware and microcode, the SI chip random driver basically emulates a portion of the server software function. This differs from the SP Switch2 Adapter verification, since each chip in that environment was tested with pseudo-random stimuli to the limits of its specified function with minimal regard to how the software might eventually drive it. Because descriptor lists are at the heart of message-passing operations, each instance of the SI chip driver individually handles building and managing all aspects of its lists. In order to implement support with this breadth, numerous variables related to the actual processing of descriptors had to be considered. In turn, solutions for these issues had to be incorporated either in the code or as control parameters. Some examples of these variables include

- Length of descriptor lists and their location in memory.
- Amount of payload data.
- Addressing mode (real or translation) used for descriptor list and/or payload data locations.
- Memory boundary considerations for descriptors and payload data.
- An image of the descriptor lists and payload data storage (must be maintained throughout a test).
- Start and end locations of descriptor lists and payload data [must be determined and varied within the context of adapter requests being on cache-line (128-byte) boundaries].
- Number of descriptor lists in process at a given time.
- Frequency with which active channels make requests to the adapter.

Several of the issues were actually multidimensional in nature. For example, in dealing with boundary considerations, the requirements mandate that descriptors must start on a 16-byte boundary and lists may not cross a page boundary, whereas payload data can start on any boundary. Another example is the addressing mode. If address translation is to be used, it requires the creation of appropriate addressing schemes along with page tables and other related support. Most elements of this support are established prior to the start of simulation and many remain static during the run, while some change throughout the simulation. These controls and descriptor list variables force the random driver to produce valid transactions while also allowing the monitor to perform its prediction of results correctly.

Aside from the base mainline test focus, additional effort was put into exercising other situations in the hardware. Such scenarios included exercising boundary and stress conditions, such as causing full buffer conditions, holding off traffic/data flow, and flooding the device with requests. In addition to these tests, complete support for recovery and error testing was added.

Manipulating all of the available control parameters in a random manner drives the microcode in various ways, which in turn causes the hardware to be utilized in different ways. This randomization, done across literally millions of test cases, provides the desired level of test coverage required for this type of hardware verification environment. While the primary goal of this simulation effort is to remove hardware design flaws, a substantial amount of the microcode is also exercised. This significantly increases the chances of success when the fabricated chip and microcode first come together in the laboratory.

Advantages of the hardware/software co-verification of the SI chip

The methodology of simulating the hardware and microcode together on the SI chip yielded advantages over the SP Switch2 Adapter methodology. One of the key advantages was reduced time to market. Time was saved because it was not necessary to develop a driver that would mimic the behavior of the microcode. This would have been a large effort, and it would have duplicated the work done by the microcode developers. Also, since the driver would not be able to generate exactly the same command sequences as the final microcode, some hardware defects might not have been found until late in the development cycle, when the hardware and microcode were tested together. By using co-verification, late releases of additional passes of the hardware were avoided, thus improving overall time to market. Analysis of the SI chip defect data shows that about 38% of the 255 hardware defects identified in this simulation environment were found as a direct result of running with the microcode.

Microcode development was a major contributor to total system development time. The SI chip simulation environment found 87 microcode defects. The process of finding many of these microcode errors was enhanced by monitoring the current microcode address. The SI chip monitor code used this address to derive microcode state information. Knowing that the microcode should always be in an idle state upon completion of a test case, the monitor helped identify microcode defects that would otherwise go undetected if only message-passing transactions were monitored. Also, through the use of current address information collected after a test-case failure, the microcode team was given a good starting

point to begin its debugging process, resulting in greater efficiency. Overall, microcode development was able to progress at a faster rate when compared with the alternative of waiting to develop the microcode until the hardware had been manufactured.

In addition to faster time to market, another advantage of simulating the hardware with the microcode was increased flexibility when deciding where to fix system problems. For example, a microcode defect that would have been difficult to fix in microcode was instead fixed in hardware. Alternatively, hardware defects were either temporarily worked around or permanently fixed by making adjustments to the microcode. Using a microcode fix to temporarily work around a hardware problem allowed the verification team to make progress until a hardware fix was available. This methodology also has the ability to uncover architectural flaws. System function that had mistakenly been omitted from hardware and microcode was identified and marked for implementation in higher-level software. An example of this type of architectural problem involved error identification and recovery. When errors occur on the interface between adapters, it is possible for the source adapter, which sends information to the destination adapter, to resend a packet. It is the responsibility of the destination adapter to ignore any duplicate packets that it may receive. A defect was found in which a particular type of packet, called a sync packet, would not be ignored by the destination adapter if duplicate versions of this packet were received. This resulted in duplicate information being incorrectly written to the destination server. The architectural description had failed to account for this scenario, and left both the hardware and microcode without the ability to identify duplicate sync packets. Performance issues, which can often be difficult to remedy in hardware late in the design cycle, were identified early using the visibility into hardware/microcode interaction available in the SI chip verification environment. By simulating hardware and software together, which more closely represented the actual manufactured system, bring-up times for the Switch Network Interface were reduced.

Conclusion

In contrast to the verification methodology used on the SP Switch2 Adapter, the hardware/software co-verification methodology developed for the SI chip produced an environment that more closely reflected the final product. This was accomplished by modeling the verification environment around a system-level architectural description of the data transfer protocol. Taking this approach required developing driver and monitor code for the verification environment that closely conformed to the architecture. Using real microcode, instead of a simulation approximation, also brought the verification environment closer to emulating the final production system. The benefits realized by this approach were decreased time to market and increased flexibility when defects were encountered in the system before and after manufacturing.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. J. M. Ludden, W. Roesner, G. M. Heiling, J. R. Reysa, J. R. Jackson, B.-L. Chu, M. L. Behm, J. R. Baumgartner, R. D. Peterson, J. Abdulhafiz, W. E. Bucy, J. H. Klaus, D. J. Klema, T. N. Le, F. D. Lewis, P. E. Milling, L. A. McConville, B. S. Nelson, V. Paruthi, T. W. Pouarz, A. D. Romonosky, J. Stuecheli, K. D. Thompson, D. W. Victor, and B. Wile, "Functional Verification of the POWER4 Microprocessor and POWER4 Multiprocessor Systems," *IBM J. Res. & Dev.* **46**, No. 1, 53–76 (January 2002): see *http://www.research.ibm.com/journal/rd/461/ludden.pdf*.
2. D. G. Bair, S. M. German, W. D. Wollyung, E. J. Kaminski, Jr., J. Schafer, M. P. Mullen, W. J. Lewis, R. Wisniewski, J. Walter, S. Mittermaier, V. Vokhshoori, R. J. Adkins, M. Halas, T. Ruane, and U. Hahn, "Functional Verification of the z990 Superscalar, Multibook Microprocessor Complex," *IBM J. Res. & Dev.* **48**, No. 3/4, 347–365 (May/July 2004); see *http://www.research.ibm.com/journal/rd/483/bair.html*.
3. B. Wile, M. P. Mullen, C. Hanson, D. G. Bair, K. M. Lasko, P. J. Duffy, E. J. Kaminski, Jr., T. E. Gilbert, S. M. Licker, R. G. Sheldon, W. D. Wollyung, W. J. Lewis, and R. L. Adkins, "Functional Verification of the CMOS S/390 Parallel Enterprise Server G4 System," *IBM J. Res. & Dev.* **41**, No. 4/5, 549–566 (July/September 1997); see *http://www.research.ibm.com/journal/rd/414/mullen.pdf*.
4. G. G. Hallock, E. J. Kaminski, Jr., K. M. Lasko, and M. P. Mullen, "SimAPI—A Common Programming Interface for Simulation," *IBM J. Res. & Dev.* **41**, No. 4/5, 601–610 (July/September 1997); see *http://www.research.ibm.com/journal/rd/414/hallock.pdf*.
5. J. Darringer, E. Davidson, D. Hathaway, B. Koenemann, M. Lavin, B. Lee, J. Morrell, S. Ponnapalli, K. Rahmat, W. Roesner, E. Schanzenbach, and L. Trevillyan, "EDA in IBM: Past, Present and Future," *IEEE Trans. Computer-Aided Design, Integrated Circuits & Syst.* **19**, 1476–1497 (December 2000).

Received October 4, 2004; accepted for publication March 4, 2005; Internet publication August 10, 2005

Steven P. Goldman *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (sgoldman@us.ibm.com)*. Mr. Goldman received a B.S. degree in computer engineering from Rutgers University in 1999, joining IBM that same year as a hardware verification engineer working on pSeries I/O adapter chips. In 2003, he received an M.S. degree in computer engineering from National Technological University. He is currently working as a logic designer for the next-generation zSeries microprocessor.

Lisa M. Mohr *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (lmohr@us.ibm.com)*. Mrs. Mohr received a B.S. degree in electrical engineering from Rutgers University in 1981. That same year she joined IBM in Kingston, New York, where she worked on the development of IBM 3270 display systems and PC monitors. She received an M.S. degree in electrical engineering from Syracuse University in 1984. In 1986, she joined IBM Graphics Systems, where she designed peripherals, boards, rendering chips, and RAMDACs for IBM 5080 and 6090 systems. For the last ten years, Mrs. Mohr has led the hardware verification of high-performance communication adapters and switches for IBM RS/6000* SP and pSeries systems. Her current focus is the verification of I/O hub and bridge chips for future IBM systems. Mrs. Mohr is a Senior Engineer and leads several I/O chip verification teams. She is a member of the Institute of Electrical and Electronics Engineers.

David R. Smith *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (davrsmit@us.ibm.com)*. Mr. Smith joined IBM in Poughkeepsie, New York, in 1979 after receiving a B.S. degree in electrical engineering from the University of Bridgeport, Connecticut. Mr. Smith initially worked on IBM 3090* Service Processor development; in 1986 he joined the Engineering Design Tools group, where he worked on IBM internal use engineering software tools. In 1990, Mr. Smith joined the IBM Test Design Automation group in Endicott, New York, working on static and dynamic logic fault simulators and analysis tools. In 2000 he returned to Poughkeepsie, where he has worked on hardware verification of pSeries I/O chips, including team leader responsibilities for some of them. Mr. Smith is currently an Advisory Engineer working on the verification of I/O chips for future IBM products.