Functional verification of the POWER5 microprocessor and POWER5 multiprocessor systems

D. W. Victor
J. M. Ludden
R. D. Peterson
B. S. Nelson
W. K. Sharp
J. K. Hsu
B.-L. Chu
M. L. Behm
R. M. Gott
A. D. Romonosky
S. R. Farago

This paper describes the methods and simulation techniques used to verify the functional correctness and performance attributes of the IBM POWER5™ microprocessor and the eServer™ p5 systems based on it. The approaches used were based on migrating the best practices that had been used to verify the POWER4™ chip. The POWER5 chip design posed new challenges to the simulation team with the addition of simultaneous multithreading (SMT) and dynamic power management (DPM). In addition, there was further integration of cache and memory subsystem function onto the POWER5 chip. Since the design complexity had increased from the POWER4 design, the use of test plan coverage tools and techniques was expanded to ensure the maximum effectiveness of each simulation cycle run. A new toolset was also employed to improve the utilization of the large pool of computers used to run batch simulation jobs and to provide more efficient fail reproduction and bug fix management. For the system-level verification, a new test-case-generation tool was utilized which allowed for more targeted testing through a deeper knowledge of the system topology. In parallel with the mainline functional validation, verification of reliability functions and performance attributes also had increased focus for the POWER5 design.

1. Introduction

Architectural and functional verification of the POWER5 multiprocessor

The initial POWER4* high-end eServer* systems were introduced by IBM in late 2001. Since that time, POWER4 and POWER4+* microprocessors have continued to serve as the engines for not only the high-end iSeries* and pSeries* machines, but also have been introduced in the midrange and low-end iSeries and pSeries spaces [1]. The POWER5 microprocessor is the next generation in this processor family [2]. Much of the verification team, methodologies, and guiding principles that were brought to bear on the verification of POWER4 [3] design were deployed in an evolutionary way in the POWER5 verification effort. The fundamental goal of the verification team was to effectively validate the POWER5 chip utilizing the lessons learned from the POWER4 verification experience. To this end, the team not only added efficiencies to the overall verification process, but enhanced the tools and strategies deployed to meet the unique challenges introduced in the POWER5 design.

POWER5 design changes

This paper presents the verification strategy used to verify features unique to the POWER5 design. While the POWER5 design point maintains both binary and structural compatibility with the POWER4 design, allowing existing executable files to continue to execute properly and application optimizations to advance, enhanced performance and functionality was introduced in the POWER5 processor and system design. New features of the POWER5 processor design that required a novel approach by the verification team included simultaneous multithreading (SMT) and dynamic power management. Beyond these new design features, functional enhancements introduced in the POWER5 chip include 1) greater virtualization; 2) better reliability,

0018-8646/05/$5.00 © 2005 IBM

availability, and serviceability (RAS) characteristics of the chip and system; 3) improved elastic interface characteristics; 4) enhanced memory tracing; and 5) introduction of the GX+ bus. Each of these changes provided unique challenges to the verification effort.

Virtualization

The POWER4 processor was the first implementation of the POWER* logical partitioning (LPAR) architecture. This capability allowed a single system to be divided into multiple logical partitions such that each partition could run a different operating system. POWER4 LPARs required each processor to be placed in a single partition when the system was powered on. The POWER5 processor extended the idea of logical partitioning capabilities to a feature called micropartitioning. In the POWER5 design, a single processor can service multiple partitions. Using this capability, each LPAR is now allocated a certain amount of time on each processor, and this allocation can be dynamically adjusted to satisfy changing workload demands. A single POWER5 processor is capable of servicing up to ten partitions with this new capability. The hardware functions to enable such a feature are distributed throughout the processor, requiring strong expertise in the verification community to ensure a comprehensive test plan. This function is implemented in the POWER5 design by a combination of hardware and hypervisor functions which minimize changes required to the various operating systems in order to support micropartitioning and several other unique hardware configuration aspects. A detailed explanation of the enhanced partitioning architecture is given by Armstrong et al. [4] in this journal issue.

Simultaneous multithreading and dynamic thread switching

In contrast to the POWER4 processor, which was capable of executing only a single instruction stream per processor core, each POWER5 processor core supports the execution of two instruction streams, or *threads*. This capability is referred to as *simultaneous multithreading* (SMT). This design point provides the ability to fetch instructions from one or two threads per processor core and schedule instructions for execution from both threads concurrently. Each processor core dynamically adjusts to the environment, allowing for possible execution of instructions from both threads and for preferential treatment of one thread if the other thread encounters long-latency events. A significant challenge in the verification of such a design is to ensure effective stressing of contention for shared resources between each thread. Additionally, verification of dynamic thread switching, which is the transition from simultaneous multithreading to single-threading (or the reverse), introduced new and unique challenges into the verification environment.

Power savings

Chip power has become an extremely important design parameter with current CMOS technologies. Simultaneous multithreading leads to better execution unit utilization and thus further magnifies the problem of keeping the chips cool. POWER5 chips use fine-grained dynamic clock-gating mechanisms extensively throughout the chip. In addition, the POWER5 design also has a low-power mode that significantly reduces the switching power on the chip. A new methodology was brought to bear in the verification environment to verify this dynamic power management.

New tools and methodologies

The POWER5 verification effort benefited from improvements made to existing tools and methodologies as well as the introduction of new, powerful tools. The architectural test-case generator used in the POWER5 chip verification, Genesys-Pro [3], was significantly upgraded from that used in the POWER4 timeframe. A full programming language was introduced to allow a more intuitive construction of test cases to be generated by the verification engineer. These tests were developed with portability in mind: Not only could they be used in multiple levels of the verification hierarchy on a given program, but they could be transported to other Power Architecture* designs under test. Coverage metrics were used to measure the completeness of the verification effort as well as to ensure effective use of simulation resources. A new paradigm for test plan coverage was used in the POWER5 verification effort. Whereas the POWER4 verification effort used coverage analysis to gauge effectiveness of a fairly generic test plan, the POWER5 effort employed a coverage-directed test plan constructed jointly by the design and verification lead engineers. Additionally, the internally developed IBM semiformal verification tool, SixthSense [5], was used to assist in the coverage analysis. SixthSense was also deployed as a powerful bug finder in the POWER5 microprocessor verification effort [6].

This paper describes how the team built on the success of the POWER4/POWER4+ processor and enhanced the verification arsenal to deal with these unique design additions to the POWER5 processor and system. The paper is organized into four major sections that follow. Section 2 addresses the verification challenges that are specific to the POWER5 design points, such as dynamic power management. Section 3 describes the system simulation configuration and methodology. Section 4 details the coverage analysis deployed to gauge the completeness of the testing. Finally, Section 5 describes the overall results of the verification effort and offers concluding remarks.

2. Unique POWER5 verification challenges

Shared resource pool management

Verification of processor resources shared between threads
One of the main challenges of verifying a simultaneous multithreaded processor such as the POWER5 processor is to effectively stress the contention for shared resources between threads. There are two types of such shared resources: those which involve the memory hierarchy and those which functionally are completely internal to the processor core. Resources involving the memory hierarchy include the following:

- L2 cache.
- Memory.
- Page (translation) tables.
- Semaphores (software lock reservations).
- Effective to real address translations (ERATs) (first-level translation caches).
- L1 data and instruction caches.

Resources that are functionally internal to the processor core include the following:

- Rename register pools for general-purpose registers (GPRs) and floating-point registers (FPRs).
- Issue queues.
- Branch instruction queue (BIQ).
- Load miss queue (LMQ).
- Load reorder queue (LRQ).
- Store reorder queue (SRQ).
- Global completion table (GCT).

Verification of shared resources involving the memory hierarchy in an SMT processor such as the POWER5 processor is quite similar to true multiprocessor (MP) verification, with one primary difference: The creation of contentions for shared resources can now occur with only a single processor core running in SMT mode. The typical method of verifying these resources involves testing contentions involving the same cache lines, cache congruence classes, or page-table entries. This is accomplished by constraining the memory addresses that are requested by each processor to be the same or to have similar properties (i.e., to target the same congruence class). While this approach lends itself to verifying L1 instruction and data caches and the ERATs, it is not directly applicable to verifying shared resources internal to the processor core, since the means for addressing these resources are, in general, less straightforward. Verification of these internal resources at the processor core level requires extra measures.

For example, in the POWER5 design, there are 120 physical GPRs and another 120 physical FPRs that are respectively renamed, or mapped, to one of 32 architected GPRs or FPRs on a per-thread basis. Since this register pool is completely shared between the threads (and is not directly addressable), once all of these registers are in use, neither thread can dispatch additional instructions until a group of instructions on one of the threads "completes," thereby freeing up one or more physical registers for the next dispatch group of instructions (which could be for the same or the other thread). Running randomly generated instruction sequences, as was frequently done on both the POWER4 and POWER4+ designs, would typically not stress these resource pools to the point at which all 120 registers were fully mapped. Therefore, in order to exhaust these registers, special test templates were constructed using the Genesys-Pro test generator to create sequences consisting of a long-latency instruction followed by several instructions requiring one or more GPRs or FPRs. In some cases, the long-latency instruction would delay completion of a group by a programmable number of simulation cycles (i.e., longer than would normally occur). Employing this approach, it is possible to create tests in which both threads are able to exhaust these resources on a much more frequent basis, thereby verifying the design integrity under otherwise unusual circumstances.

The register rename pools described above and the GCT are examples of dynamically shared resources whose thread allocation is determined by the requirements of each thread and is indirectly controlled by thread priority logic. Several other resources, however, are always shared equally between the two threads (in SMT mode). To stress equally shared resources, special tests were constructed to target the particular resource in both single-threaded and multithreaded modes by simply changing parameters in the test templates on the basis of the number of threads in the output test case. (This was accomplished automatically by the test generator.) Again, using long-latency instructions to delay completion of a particular thread was an approach commonly used to target "queue full" conditions for both threads separately or simultaneously. An added advantage of being able to utilize the same test templates for both single-threaded and multithreaded modes was that the same set of directives which targets the shared resource on one thread in single-threaded mode was utilized to target the same resource on both threads in SMT mode. Additionally, a "smart asynchronous flush" irritator[1] was used to delay injection of random flushes or interrupts into the design until key times such as the above "queue full" conditions. This was necessary in order to allow the full conditions

[1]The term *irritator* refers to the stimulus applied to the interface; this term is used interchangeably with the term *driver*.

to occur near or on the same cycle in which the flush was injected.

Special tests were also developed that targeted the L1 instruction cache (I-cache) and ERAT in SMT mode. These tests targeted scenarios in which both threads access the same instruction stream with either the same or different "effective address" to "real address" translation mappings, instruction streams that use the same effective address on both threads but mapped to different instruction streams, and instruction streams in which both threads access the same congruence classes in either the L1 I-cache or the I-ERATs to cause thrashing to occur.

All of the checking which was utilized for the POWER4 design was enhanced to handle the simultaneous multithreading mode in the POWER5 design. This included the instruction-by-instruction checking[2] of register results and instruction execution on a per-thread basis. Additionally, since SMT essentially turns each processor core into two logical processors, it was necessary to bring the portable coherency monitor (SPECTOR) into the processor core level of simulation. SPECTOR provides a framework for architectural-level checking of rules such as coherence and sequential load execution; its use is described in detail in Section 3. In the POWER4 verification effort, this type of coherency checking was employed for only the chip-level models (i.e., two-processor cores) and above (*N*-way system-level models). SPECTOR combined with other runtime and end-of-test checking code uncovered all memory-consistency and atomicity violations that resulted from collisions between loads and stores from the two threads prior to first-pass hardware fabrication.

Dynamic thread switching in the POWER5 processor

Simultaneous multithreading

As stated previously, the POWER5 processor contains a maximum of two threads per core, with dedicated program counters per thread, pools of rename registers including 120 general-purpose registers (GPRs) and 120 floating-point registers (FPRs), and support for two modes of operation: simultaneous multithreaded (SMT) and single-threaded (ST) [2]. In SMT mode, instruction fetches alternate cycles for each thread, and physical register files, both GPR and FPR, are dynamically shared by both threads. SMT mode increases the efficiency of applications that regularly encounter long-latency events such as cache misses and I/O accesses; however, by its very nature, SMT mode can be less efficient for applications that utilize the higher instruction-level parallelism offered by POWER5. For example, some technical computing applications are structured to effectively utilize the data prefetching capabilities of POWER4 and POWER5 to minimize cache misses and fully consume the execution pipelines, especially the floating-point units. These applications perform better when running in ST mode, since they have complete access to the entire queue depth and all 120 physical FPRs. In SMT mode, since the depth of most queues and the number of rename registers available to each thread are reduced, the overall throughput of these applications can in some cases also be reduced when running in SMT mode. Therefore, POWER5 supports a second mode of operation with only one active thread in which it behaves more like its predecessor, the POWER4 processor, while still taking advantage of the additional physical registers added for POWER5 to support SMT mode.

Dynamic thread switching

A processor is typically configured for either SMT or ST mode, but the POWER5 system gives software the flexibility of dynamically switching between modes in order to enhance performance. Dynamic thread switching has two fundamental transitions: SMT → ST and ST → SMT. The SMT → ST transition is by far the more straightforward. This transition essentially puts one of the two active threads into an inactive state and can be initiated only by hypervisor software [4] running on the thread in transition. More specifically, the hypervisor software executes an instruction sequence which resets the appropriate "thread enabled" bit in the control register (CTRL). Once the CTRL is updated, the POWER5 processor shuts down instruction fetching for the thread in transition and reallocates all of its internal resources to the remaining active thread. The reallocation of resources results in the loss of the architected state except for state common to both threads and, if appropriately enabled, the decrementer of the inactive thread. Hypervisor software must save all required state information prior to initiating the transition. Only one thread can be inactive at a time, and the remaining active thread is not allowed to make this transition until both threads are active again and back in SMT mode.

The processor remains in ST mode until an external event dynamically forces it to enter the ST → SMT transition. The dormant thread can be reactivated by one of the following events:

- Hypervisor software running on the remaining active thread of the same core.
- System reset interrupt (SRI).
- Decrementer interrupt.
- External interrupt.

[2]Instruction-by-instruction results checking is achieved by comparing the architected results of each instruction as they appear in the Architectural Verification Program (AVP or test case) against the VHDL model as each instruction executes. (VHDL = VHSIC Hardware Description Language, where VHSIC = Very High Speed Integrated Circuits.)

Once the transition is initiated by any one of these events, the inactive thread begins an initialization sequence that reallocates the shared internal resources to the thread in transition and then initiates an instruction fetch from the system reset interrupt vector located at effective address 0x0000000000000100. In addition, the machine state register (MSR) and the save restore register 1 (SRR1) are loaded with the state information required for the hypervisor software to restore the thread to its proper state.

Scope of verification

The capability of dynamically switching between ST and SMT modes introduced new challenges to the POWER5 verification team beyond those already addressed for standalone SMT testing. The first challenge was the integration of these scenarios into a random environment based on implementation verification programs (IVPs) with predetermined architectural results. In contrast to the random testing of asynchronous interrupts, which is generally transparent to the IVP, these scenarios required the test case to be consistent with the new transitions. Additionally, since the architected state is lost during the transition, a method of saving and then restoring state information was required.

The second challenge was to establish methods of detecting problems that can occur during the transition. For instance, we added checking to detect problems that occur when a thread is deactivated and subtle problems that can occur during the initialization of the instruction stream after a thread is restarted. We also enhanced our capabilities to introduce asynchronous events into the simulation environment for targeting interesting windows during the transition between modes.

The team agreed that the leading consideration was to adapt the substantial verification building blocks already in place. The synergy of the instruction sequencing unit (ISU) unit-level and the core-level simulation efforts was instrumental in developing new techniques for testing dynamic thread switching. The team developed new unit-level checkers, new unit-level irritators, and new IVPs for use with the model-based test generator, Genesys-Pro. The new unit checkers and irritators and the Genesys-Pro IVPs were installed and activated on both the ISU unit model and the core model. Once we had dynamic thread switching running consistently at the core level, we installed and activated the new checkers, irritators, and IVPs on the chip level to confirm that the behavior remained consistent in the chip environment.

Random test-case generation

The POWER5 project is the first project to use Genesys-Pro exclusively. Genesys-Pro, the "next-generation" model-based test generator, contains capabilities above and beyond those of legacy test-generation tools [3]. More specifically, Genesys-Pro provides the capability to write advanced test-case scenarios that would previously have been impossible to write and maintain. For dynamic thread switching, two new prototype IVPs using Genesys-Pro were developed. The first prototype, known as the Dynamic Thread Switching Irritator (DTSI), provided an effective vehicle for initially verifying both thread-switching transitions. The basic premise was to generate single-threaded IVPs that contained a built-in thread-switching irritator. The irritator is a relatively simple subroutine appended to the IVP that is invoked to handle the thread-switching transitions. **Figure 1** depicts the flowchart for the DTSI prototype. As shown in the figure, the initially inactive thread has passed through multiple transitions by the time the test case finishes.

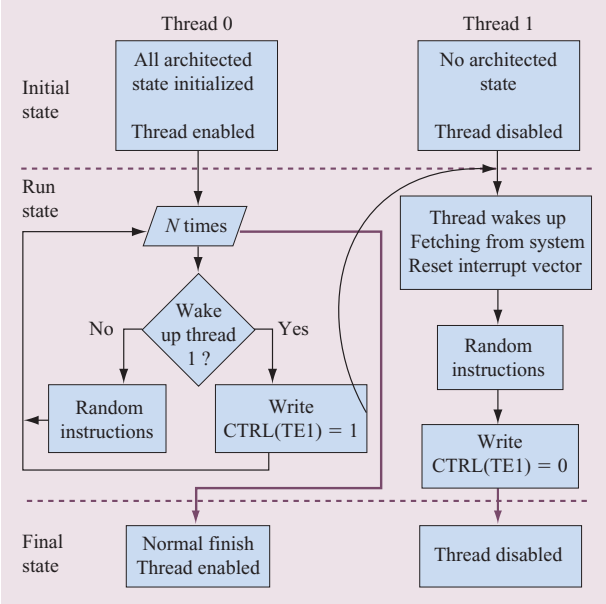

Figure 1

Example of the Dynamic Thread Switching Irritator (DTSI) prototype.

The second prototype, known as Dynamic Thread Switching (DTS), was complex to develop but more closely resembled the scenarios that actual hypervisor software would require in order to invoke a dynamic thread switch. To make this prototype work, routines that could be randomly inserted into the IVP, including reentrant save and restore algorithms that could be used by both threads simultaneously, had to be developed. The initial SMT → ST transition was incorporated into the system call interrupt handler at 0x0000000000000C00. The system call interrupt can be randomly invoked by

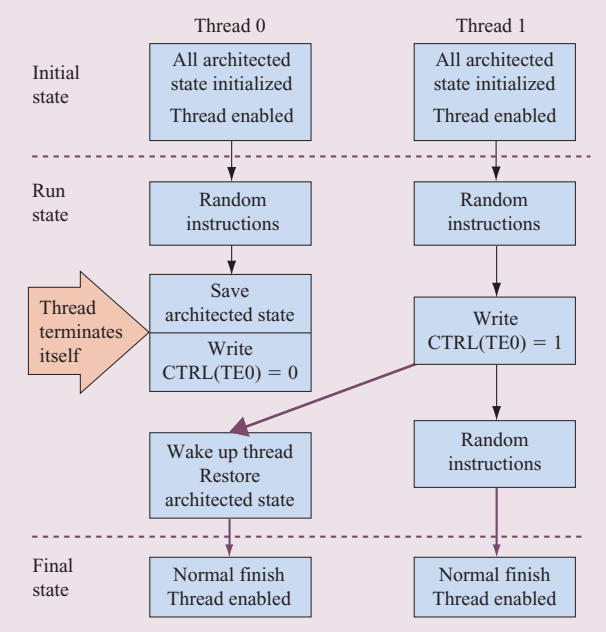

Figure 2

Example of the Dynamic Thread Switching (DTS) prototype.

any IVP running in any mode using the system call (sc) instruction. Multiple entry points into the system call handler to support the two transitions were added, namely "SC_Wake_Thread" and "SC_Thread_Kill." The wake routine reactivates a thread if it is currently inactive, similarly to the sequence used in the DTSI prototype. The kill routine deactivates a thread after executing the save algorithm. Once all of the necessary state information is safely stored in memory, the "kill" routine deactivates the current thread.

RTX[3] drivers and checkers

As illustrated in Figure 1 and **Figure 2**, verification of thread-switching transitions requires additional checkers to confirm that the transitions have correctly allocated resources and placed threads in their proper states. The RTX unit, which received significant updates to support the new SMT mode, was also enhanced to handle dynamic thread switching. New checkers were added to confirm that the entire register rename mapper state information is correct when a thread is deactivated and correctly reassigned when a thread is reactivated. Many existing checkers were also enhanced to make them aware of dynamic thread switching. For instance, updates to the asynchronous interrupt checkers were required for system reset interrupts to detect problems unique to these transitions, such as detecting the correct type of SRI and verifying that the reset type posted to software matches. Unique updates were also necessary for detecting hang and end-of-test conditions.

Taking the scenarios illustrated by the previous examples to the next level also required enhancements for the irritators used to inject asynchronous interrupts into the simulation environment. Recall that asynchronous events (including system reset interrupts, external interrupts, and decrementer interrupts) can be used to initiate the ST → SMT-mode transition, in addition to the software initiation of a dynamic thread switch. Therefore, the irritators were enhanced to detect thread-switch transitions followed by the injection of multiple random asynchronous events into critical timing windows associated with the transition.

With the basic testing complete, the team continued to introduce new elements into the simulation environment to increase the robustness of the testing. Both unit and core simulation environments enabled existing unit irritators to cause these new tests to run under more extreme conditions. For instance, we continued to enable all of the various types of asynchronous events as well as combinations of internal modes.

Dynamic power management (DPM)

Overview

Power consumption and the resulting system cooling requirements are becoming a more significant limit to the performance and maximum configuration size of each successive generation of computer systems. The increased utilization of processing resources in the multithreaded POWER5 processor places additional stress on power and cooling requirements. To reduce the impact of power consumption, POWER5 systems implement a DPM scheme not present in the previous POWER4 series of processors.

The POWER5 DPM design approach relied on dynamically gating the functional clocks to logical subsections of the design that were not currently being used by the executing instruction streams. To simplify the implementation, the clock gating was implemented for functional subunits rather than focusing on independently gating clocks of every individual group of latches in the design. For example, the translation table-walk logic has to be clocked only when a translation lookaside buffer (TLB) miss occurs; at all other times, the clocks to this logical subunit can be stopped without affecting function or performance. With this scheme, clock gating and power management are controlled by hardware, with no assistance required from the system firmware or software.

[3]The acronym RTX stands for "runtime executable." It stands for the simulation code (C/C++) written by the user for a particular testbench.

Verification

The goal for functional verification of DPM was to ensure that the clock gating logic did not introduce errors into the function or the performance of the POWER5 processor. The basic verification methodology was applied to all levels of POWER5 verification. DPM functional verification focused on logical correctness, not the actual power-saving results. DPM logic was enabled and verified at the unit, core, nest, chip, and system levels of the verification hierarchy. Functional and performance verification tests were performed while testing the clock-gating functions. In addition to typical verification techniques, several tasks were focused specifically on power management clock gating.

An effort was made to provide a simulation check for each local clock-gating control. The goal was to write an abstract model of each clock-gating control and check the actual simulation result against the abstract model on each simulation cycle.

The design goal of providing cycle-level performance accuracy between clock-gating-enabled and clock-gating-disabled conditions helped to reduce the potential increase in state space introduced by clock gating. To verify this design goal, a subset of the directed random simulation was run twice: In the first run, all clock gating was enabled; in the second run, some or all of the clock gating was disabled. The simulation execution time and event time of key events could be compared to verify that cycle-level performance equivalence was maintained between the two runs.

As noted, DPM functional verification did not verify the actual power-saving results. However, data collected from functional simulation was fed into the tools used for power estimation to measure clock-gating effectiveness. The methods and results of that analysis are beyond the scope of this paper.

Elastic interface

Given the bus transfer speeds and the length of wires between chips, certain areas exist in which information cannot travel from one logical block to another in one cycle even if there are no logical gates on that path. In some cases it takes several clock cycles for the signal to be received. Thus, there can be more than one logical value in transit at any given time. In a cycle simulator, wires present no delay. Using a time-based simulator, one can specify a delay on a wire, but only one logical value can be on the wire at a time. The problem is additionally compounded by the fact that the bus speed is not constant, thus changing the number of logical values that can be in flight at any given point in time.[4] To accurately model the variable number of logical values in transit, the simulation model was modified by inserting a programmable delay block on the long wires. This delay block was then configured by the test case to provide the required number of bits in flight to accurately model the wire. Wires that are long in relation to clock frequency present the phenomenon of differential arrival times. This is because two adjacent wires may be slightly different in length or molecular composition. IBM POWER4 systems introduced a method of realigning the common bits on an interface using the elastic interface (EI), depicted in **Figure 3**. The basic function of the elastic interface is to sample each data bit on a bus at the optimal time and then delay the presentation of the data until all of the data on the bus has been captured, so that the logic that interprets the data sees a fully synchronized bus transfer image.

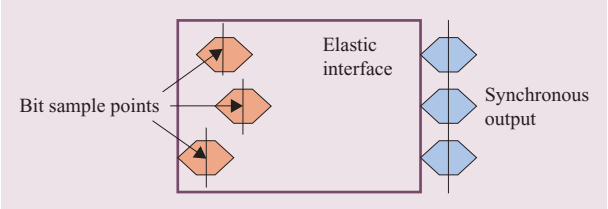

Figure 3

Illustration of elastic interface.

Since the electrical characteristics of a bus change with temperature and average workload, the POWER5 system architecture had to improve upon the original POWER4 EI implementation. The POWER5 EI implementation not only determines the optimal sample points at system boot time, but also recalculates the sample points during system operation. The recalibration process can be initiated by direct service processor operations, at periodic timer intervals, and when an error is detected on the interface. In verifying the elastic interface, we were able to show that by selectively configuring the programmable wire delay blocks, the elastic interface selects the optimal sampling point and realigns the data as designed. Additionally, the selected delays can be modified during simulation, and the act of recalibrating the interface again adjusts the sample points and properly aligns the data output.

Memory tracing

Overview

The enhanced memory-tracing feature was implemented for performance-tuning reasons. On every POWER5 chip, there exists a tracing macro that collects instruction

[4]In POWER5 systems (as in POWER4 systems), buses operate at a sub-multiple of the processor speed. Because POWER5 systems ship at multiple frequencies, the buses must also operate at corresponding multiple frequencies.

trace information and stores it directly in memory. When this tracing macro is enabled to capture data, the entire address space mapped to that chip's memory controller is devoted to either core or fabric tracing. The trace provides a means for the performance group to determine how code streams actually present themselves on the fabric or in the core.

In the core trace mode, the memory tracer generates a trace for every instruction executed and stores information such as effective instruction address, instruction image, and effective data address. When the memory tracer nears the end of its allocated memory buffers, it sends a signal to the core that stalls the core until the memory tracer can allocate more write buffers. This mechanism ensures that trace packets are not dropped in this mode.

When the tracer is running in fabric trace mode, packets can be dropped, because there is no feasible way to halt system traffic. The Tracer counts the number of dropped packets and records this information along with fabric trace information as soon as write buffers are allocated. In this trace mode, the tracer can be programmed to collect snoop address and response information on all/even/odd fabric cycles. Selection of snoop traffic sources is also configurable. The data collected in this mode has already been used in the laboratory during bring-up to help with debugging, since it gives users the ability to look at a snapshot of what is happening inside the chip, much as a logic analyzer would.

Triggering on an event during trace collection is also available and configurable. The tracer can store a centered post-trigger image, a full post-trigger image, or an image that traces until the trigger event is reached. The size of the image that is stored depends on the size of memory configured behind the memory controller. The tracer supports memory sizes ranging from 256 MB to 2,048 GB.

Verification

Testing of this logic was done at the FBC unit level and the chip level. In any given test case, one or more POWER5 chips in the system would be randomly chosen to enable memory tracing on chip and would be set in either core or fabric trace mode. The memory spaces on these selected chips were designated as memory trace addresses, so random address generation for mainline-cacheable operations did not include any lines in these address ranges. Checkers were written to monitor the interactions from the core to the tracer and from the fabric to the tracer. Trace packets written to memory were compared to what was expected by the checkers in order to verify proper recording of trace events.

To test the core stall feedback mechanism, a core-to-tracer driver was written to flood the tracer with data for collection. We also ran heavy system traffic in fabric trace mode to verify the proper recording of drop stamps. In both modes, the tracer could be set to store timestamps marking the number of idle cycles between events. When this mode was enabled, checkers verified that these timestamps were written to memory with the correct value.

The biggest challenge faced in verifying the memory tracing logic was testing the wraparound conditions when the tracer reached the end of memory space. Because of the massive number of cache-line writes that would have to be executed before reaching the end of memory, the design team implemented a special simulation mode in which the memory size could be set to just 256 cache lines. Without this special mode, it would take millions of simulation cycles in a single test case to see the overlap occur. We ran predominantly in this special mode, but eventually tested the other memory sizes by overriding internal address counters to effectively skip to the end of memory in manual test cases.

3. System verification

System simulation is hierarchically the highest level of pre-chip hardware verification. The main objective for system simulation is to verify interactions between chips using actual chip VHDL for the processor, memory, and I/O chips in system configurations similar to the ones that will be shipped. For interfaces to the real world, such as I/O, drivers are used to generate traffic into the system. Building a full 64-way POWER5 system using VHDL is neither practical, because of model size, nor the best approach to verify a system, because of relatively slow simulation throughput. Thus, the system simulation challenge was to define several different smaller model configurations that effectively represented the 64-way structure without the complete 64-way model. The 64-way system structure consists of four books connected using a book-to-book bus interconnect. Internally, each book contains two MCMs connected by a vertical bus interconnect. Each MCM contains four POWER5 chips interconnected with a chip-to-chip bus structure. With the addition of SMT, a 64-way system effectively becomes a 128-way system. Each POWER5 chip also contains memory and I/O buses providing connections to L3 cache, memory redrive chips, and I/O chips. Three categories of models were used, each containing different subsets of the system bus structure.

The majority of system simulation cycles were run on eight-way models (with and without I/O). As in the POWER4 system effort, the RTX checkers written for lower levels of verification were moved up and used in the system verification environment. Additional code was

developed to support the I/O chips and memory redrive chips at the system level.

System verification test generation and checking

Test-case generators and system-level checkers are the primary components in system-level verification. Two internally developed IBM test generators were used in the POWER5 system simulation: the MultiProcessor Test Generator (MPTG) [3] and X-Gen [7]. MPTG was the primary test generator for the POWER4 system-level verification, and it was enhanced for POWER5 systems to support SMT mode. X-Gen, developed by the IBM Haifa Research Laboratory, is capable of generating comprehensive system-level test cases, including I/O. The X-Gen framework includes sophisticated testing knowledge of the system and complex constraint-solving algorithms. X-Gen was the only test-case generator in system simulation that was used to test some of the new POWER5 design features, including hardware locks and the barrier synchronization register (BSR).

In the area of system-level checking, the original portable coherency monitor used during POWER4 verification was rearchitected to make it easier for users to write new system-level checkers. This new version of the coherency monitor, called SPECTOR (for System-wide Portable Environment for Comprehensive Testing of Operational Rules), also integrated coverage modeling and event calculation. SPECTOR, an internally developed IBM tool, provides a verification framework targeted specifically at the system level of simulation. It consists of an application programming interface (API) for development and execution of checkers, a graphical simulation debugger, a coverage data-generation engine, and several hundred portable system-level checkers. SPECTOR relies on external interface monitors for input simulation data. These tracers typically monitor well-defined interfaces throughout the system and report transaction-level events in a generic format defined by the SPECTOR framework. Additionally, SPECTOR was used to dynamically check the X-Gen test results.

Several hundred system-level checkers, implemented in terms of the above verification API, are included in the SPECTOR framework. The SPECTOR framework provides efficient checkers for architectural-level checking of rules such as coherence, sequential load execution, locking, DMA observation, and cache consistency. Architectural checkers have a number of advantages over the more common implementation checks. One important advantage is that it is possible for these checkers to catch errors in design specification as well as implementation. Additionally, this class of checkers is not subject to the possibility of overlooking subtle but important implementation details. Such advantages have allowed SPECTOR to be useful in the core and chip simulation environments as well as at the system level. SPECTOR proved to be effective in finding design problems that all other methods had missed.

N-way system simulation testing

N-way testing was performed on multiprocessor (MP) models without I/O. Many of the MPTG tests written for previous POWER *N*-way systems were ported so that they could be used to verify the POWER5 system design. Owing to the new system architecture, a large number of MPTG and X-Gen tests were created. Tests were written to stress new areas such as the L2/L3 cache hierarchy, on-chip memory controller, SMT/ST, instruction and data prefetching, the instruction cache block invalidate (icbi), lockless TLB invalidate entry (tlbie), crossing of cache-line and page boundaries, LPAR, BSR, and hardware locks.

The basic philosophy for verifying an *N*-way MP system is to have two or more processors access the same memory word or cache line during a test. The POWER5 *N*-way tests were created to select the memory addresses in a controlled manner so that all interesting combinations of processors and targeted memory controllers could be exercised. Processor transactions were chosen randomly on the basis of weights in MPTG test cases. Specific tests had to be written for each system configuration, since the mapping of address to memory varied depending on the physical topology of number of chips, MCMs, and books with the system model. Corresponding tests were also written with X-Gen to validate the new test generator. Since X-Gen has knowledge of the physical topology of the chip, including the memory map and cache structure of the system, only one set of X-Gen request files were written, and the test cases were effectively run on each of the system models. X-Gen was the only test-case generator used to test timebase, LPAR, and I/O clustering in system simulation.

I/O system simulation testing

As with POWER4 systems, simulation of the I/O chips was done in a system model. Several *N*-way models were built, each containing different system bus connections, and I/O chips were added to the models. Four remote I/O hub chips were added to the *N*-way models connected to the POWER5 chips through the GX+ bus. A varying number (one to four) of RIO ports were used to attach PCI-X bus bridge chips, limiting the number of these chips in the system to eight in order to keep the model size down and still stress remote I/O hub chips. Attached to the PCI-X bus bridge chips were one to four PCI-X bus driver/behaviorals on each of the PCI-X ports.

The PCI-X bus bridge chip uses the same RIO-bus-to-PCI-X-bus bridge chip as in POWER4+ systems. The GX+ to RIO I/O hub chip was a new chip, but the

design was derived from the comparable chip used on POWER4+ systems. It was modified in several ways for the POWER5 system. The GX bus interface was replaced with a GX+ bus interface. The major changes to the GX+ bus were in the areas of reliability, availability, and serviceability, performance, and function.

4. Coverage analysis

Methodology changes

As hardware designs become increasingly complex, it is essential to have a mechanism to evaluate the thoroughness of the testing. Establishing methods to measure the completeness of a verification effort for such complex systems is an industry-wide challenge [8]. Meaningful coverage metrics help to gauge the completeness of the verification effort and to ensure effective use of simulation resources. Further, they indicate which portions of the design may have to be tested more rigorously. The coverage approach for the POWER4 microprocessor relied on a large number of automatically generated coverage events. In the POWER5 verification effort, the methodology shifted to deliberate consideration and implementation of coverage events. The coverage events used in the POWER4 verification effort remained applicable to the POWER5 design, and the fact that there was working POWER4 hardware gave higher confidence to the common points between the two designs. Thus, while there remained value in the POWER4 coverage events, the team focused on coverage events to target function unique to the POWER5 design. A priority-naming convention was developed to mark the level of importance of the coverage event. The label "Priority1" indicated that the event was being tracked for completion for release to manufacturing. If a Priority1 event was not hit, a risk assessment was completed. In general, the new events developed for the POWER5 design were given Priority1 status, and the bulk of POWER4 events that were reused in the POWER5 effort were given Priority2 status. The priority-naming convention allowed the team to monitor the old events while concentrating on new ones.

Implementation

Reviews attended by verification engineers, design engineers, and the chief architect were held for each unit to establish which areas to cover within the new POWER5 functions. Additionally, a few areas within the POWER4 function were identified for improved coverage. Both the new POWER5 functions and the POWER4 functions identified for increased coverage were given Priority1 status. The POWER5 strategy was to create a manageable set of well-crafted test templates that targeted the events pseudo-randomly. The team reasoned that in hitting this set of meaningful events a significant number of times in a random environment, the surrounding logic was inherently covered. That is, points that had to be traversed to reach the coverage events did not have to be specified as separate events.

After the new events were added to the simulation model, monitoring began. All 35,000 test-case templates (consisting of a Genesys-Pro definition file and a set of simulation environment runtime parameters) were allowed to run for approximately one month. During this time a trend began to emerge that led to eliminating all but approximately 2,000 test templates. To eliminate the large number of test templates, a history was kept of the templates that contributed most to covering "hard-to-hit" events (i.e., the list of Priority1 events that were hit at least once but not more than ten times). After this coverage data had been collected for six weeks, the test templates that were not contributing were eliminated. Removal of the noncontributing test templates allowed the overall coverage to increase and significantly reduced the time required to hit the same number of Priority1 events.

There were approximately 4,500 Priority1 events, of which 4,000 were covered very quickly. The remaining 500 events proved to be a challenge. All Priority1 events were constantly monitored to ensure that they could be hit consistently. All events were given an age-out[5] time period of 25 days. If an event "aged out," a report was automatically generated that showed not only which test templates hit the event, but also how many times that template hit the event. Using this information, another report was generated that indicated when the test template was last run and how many test cases it took to hit the event for the first time. This information was valuable in determining simulation job submission weights for each of the 2,000 test templates to maintain continuous Priority1 event coverage.

In covering the last 500 events, deficiencies in both the Genesys-Pro generator and the test plan were revealed. More significantly, several logic bugs were discovered in covering these remaining events, and the overall size of the test suite was significantly reduced by improving test quality [9]. A novel use of the semiformal tool SixthSense was deployed to assist in analyzing a number of these remaining coverage events, which proved difficult to hit in a random or directed simulation environment. The coverage team passed the events that had never been hit in simulation over to the formal verification team for analysis. SixthSense was used to prove the coverage events reachable or unreachable, providing valuable feedback for the coverage team. For events that were

[5]The term *age-out* is defined as the time period set by the user over which the coverage event is monitored. If a coverage event is not hit within its age-out period, it is flagged. If the event is hit, the age-out counter is reset.

proven unreachable, the coverage team evaluated a portion of the checks to be invalid, while the remainder were evaluated to be "not written as intended." The invalid checks were eliminated, and the incorrect checks were appropriately modified. For coverage events that did prove reachable by SixthSense, the coverage team gained the knowledge that these were valid checks, worthy of their continued attention. In these cases, the SixthSense-produced traces demonstrating a path to hit the coverage events were additionally used to provide insight into how to tune the simulation environments to expose such rare conditions.

The initial coverage reviews yielded a set of very difficult events that required targeting of multiple random resistant scenarios in combination with one another to achieve the desired coverage goals. The suite of test templates derived from this coverage analysis now forms the basis of a portable test library which is employed by all PowerPC* processor projects currently in development within IBM, ranging from game systems and desktop computers to follow-on server products.

System-level coverage analysis

The SPECTOR framework, described previously, was used for system-level coverage analysis. Users of the framework can specify temporal cross-product coverage models. The coverage-generation engine analyzes the input trace files and reports coverage events for all tuples of events that correspond to valid cross products for each model. The coverage events are sent to the internally developed IBM coverage database tool, Meteor, for analysis and tracking. SPECTOR/Meteor were used to evaluate whether the system-level tests were hitting all of the desired sequences. New coverage models were written for the eServer p5 system to ensure that the SMT function was being adequately tested. Additional tests were written to hit the scenarios that did not occur with the initial tests.

5. Results and concluding remarks

If the quality of verification is measured by the number of defects found during hardware bring-up, the verification of the POWER5 processor was quite successful. The team delivered first-pass hardware capable of booting all three of the supported operating systems (AIX*, Linux**, and i5/OS*) in simultaneous multithreading mode with dynamic power management enabled, and this first-pass hardware was capable of running all three operating systems simultaneously on a single POWER5 chip.

There were, of course, some design defects found in the hardware laboratory running test exercisers. However, the POWER5 team identified approximately 95% of all design problems prior to the release of the chip to manufacturing, and less than 1% of the problems would have a serious impact on the bring-up of hardware in the laboratory. The high-impact design defects were of such a nature that they did not impede progress in the laboratory. The hardware bring-up team was able to circumvent all of the high-impact problems with temporary hardware or, in some cases, with firmware workarounds.

Many of the enhancements to the POWER5 design were extremely difficult to verify. The design and verification teams were aware of most problematic areas from the early stages of the project on the basis of their previous experience with the POWER3* and POWER4 projects. To minimize hardware defect transfer to the bring-up laboratory, tradeoffs and design enhancements were made that would minimize the likelihood of transfer of design defects and enhance our ability to work around potential problems encountered in the laboratory. The emphasis on hitting high-priority coverage events for all of our new mainline features so that we could gauge the effectiveness of test-case development and target simulation resource was to be critical to our success. We also added well-thought-out programmable switches to the design that would allow us to either reduce complexity (and thus performance) to work around problems or stress components more heavily than normal to aid in more rapid discovery of defects in areas of concern. Finally, we added a new workaround capability in the POWER5 design, called workaround triggering, or WAT, which allowed us to detect trigger conditions and act on them by manipulating the behavior of the logic to avoid a problem.

With the support of these new debugging and workaround capabilities on the POWER5 chip, we were able to sustain our progress in the laboratory and work around difficult situations that would otherwise have caused delays. Ultimately, the actions that we took in verification as well as the new and improved debugging and workaround features in the POWER5 design resulted in a product with excellent quality which took less time to deliver.

In examining the trends of current microprocessor designs, we fully expect that future designs will embrace more of the same features exemplified in the POWER5 processor:

- Out-of-order execution.
- Multithreading for each core.
- Dynamic power management.
- Increasing virtualization and partitioning capabilities.
- Multiple processor cores for each chip.
- Enhanced reliability, availability, and serviceability.

The methodology employed by the POWER4 team and the enhancements added by the POWER5 team have

proven to be solid. They will provide a sound foundation for building verification methodologies for future processors within IBM and likely for the industry in general. Any changes to this methodology will probably be evolutionary, applying innovation on an as-needed basis to meet new design and architecture requirements.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark of registered trademark of Linus Torvalds or Intel Corporation.

References

1. J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM J. Res. & Dev.* **46**, No. 1, 5–25 (January 2002).
2. R. Kalla, B. Sinharoy, and J. Tendler, "IBM POWER5 Chip: A Dual Core Multithreaded Processor," *IEEE Micro* **24**, 40–47 (March–April 2004).
3. 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).
4. W. J. Armstrong, R. L. Arndt, D. C. Boutcher, R. G. Kovacs, D. Larson, K. A. Lucke, N. Nayar, and R. C. Swanberg, "Advanced Virtualization Capabilities of POWER5 Systems," *IBM J. Res. & Dev.* **49**, No. 4/5, 523–532 (2005, this issue).
5. H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann, "Scalable Automated Verification via Expert-System Guided Transformations," Conference on Formal Methods in Computer-Aided Design, November 2004; *Lecture Notes in Computer Science* **3372**, 159–173 (2004).
6. R. M. Gott, J. R. Baumgartner, P. Roessler, and S. I. Joe, "Functional Formal Verification on pSeries Microprocessors and Communication Subsystems," *IBM J. Res. & Dev.* **49**, No. 4/5, 565–580 (2005, this issue).
7. R. Emek, I. Jaeger, Y. Naveh, G. Bergman, G. Aloni, Y. Katz, M. Farkash, I. Dozoretz, and A. Goldin, "X-Gen: A Random Test-Case Generator for Systems and SoCs," *Proceedings of the IEEE International High Level Design Validation and Test Workship*, Cannes, France, October 2002, pp. 145–150.
8. S. Tasiran and K. Keutzer, "Coverage Metrics for Functional Validation of Hardware Designs," *IEEE Design & Test of Computers* **18**, 36–45 (July–August 2001).
9. M. Behm, Y. Lichtenstein, J. Ludden, M. Rimon, and M. Vinov, "Industrial Experience with Test Generation Languages for Processor Verification," *Proceedings of the 41st Design Automation Conference*, Session 4.1, 2004, pp. 36–40.

Received September 29, 2004; accepted for publication April 15, 2005; Internet publication August 11, 2005

David W. Victor *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (dvictor@us.ibm.com)*. Mr. Victor currently manages the IBM Systems and Technology Group Global Verification Team, a 200-engineer organization responsible for the pre-silicon verification of microprocessors and ASICs that are utilized in IBM iSeries, pSeries, and zSeries* systems. He has more than 16 years of experience developing and managing the logic design, microarchitecture, verification, and hardware test of various processor, memory, I/O, and graphics chips for IBM computing systems. Mr. Victor received his B.S. degree in electrical engineering and applied physics from Case Western Reserve University in 1988 and his M.S. degree in electrical engineering from Syracuse University in 1990. He holds 11 patents, one of which was recognized with a supplemental patent award. He has authored six technical publications, including two papers for the *IBM Journal of Research and Development* and a recent article in the *IBM MicroNews*. Mr. Victor has received multiple awards for his work in design and verification, including an IBM Outstanding Technical Achievement Award for SMP memory controller design leadership, an IBM Outstanding Technical Achievement Award for POWER4 development, and a recent IBM Outstanding Technical Achievement Award for POWER5 development.

John M. Ludden *IBM Systems and Technology Group, 1000 River Street, Essex Junction, Vermont 05452 (ludden@us.ibm.com)*. Mr. Ludden is a Senior Engineer; he has been involved in design verification since joining IBM in 1990 after receiving a B.S. degree in electrical engineering from the Rochester Institute of Technology. He was initially involved in the verification of I/O and storage control subsystems for IBM S/390* mainframes. Since 1993, Mr. Ludden has been the processor-level test plan developer for the POWER3*, POWER4, and POWER5 microprocessors, specializing in the application of pseudo-random test-case generation for verifying out-of-order execution, simultaneous multithreading, and symmetric multiprocessing designs. He has worked closely with the IBM Haifa Research Laboratory in Israel for more than eleven years to improve test-generation capabilities and with the PowerPC architecture community to advise them on the verification implications of proposed architecture changes. Mr. Ludden received an IBM Outstanding Technical Achievement Award for the POWER4-based eServer systems; he has two patents pending. He has served as a consultant responsible for reviewing the verification plans for several other microprocessor designs within IBM, and spearheaded the development of a portable library of PowerPC verification test templates used by all current 64-bit PowerPC designs within IBM.

Richard D. Peterson *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (petersn@us.ibm.com)*. Mr. Peterson joined IBM in 1983; he is currently a Senior Engineer. Since 1992, he has worked on processor and system verification for IBM eServer, iSeries, and pSeries machines. He received a B.S. degree in electrical engineering from the University of Texas in 1982 and an M.S. degree in computer engineering from the National Technological University in 1996. In 2001, Mr. Peterson received an IBM Division Award for his contributions to AS/400* CEC verification.

Bradley S. Nelson *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (bsnelson@us.ibm.com)*. Mr. Nelson received his B.S. degree in computer engineering from Texas A&M University in 1997. After graduation he joined the RS/6000* processor memory subsystem group. He worked as a verification team leader for the POWER4 memory controller and

L3 data cache, and during that time pioneered new methodologies to verify asynchronous interfaces. Mr. Nelson worked as a verification project manager for the POWER5 memory subsystem. He is currently the verification team leader for the load store unit for POWER6*.

W. Keith Sharp *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (sharp2@us.ibm.com)*. Mr. Sharp is a Senior Technical Staff Member. He was the core verification leader for the POWER5 program and is currently a verification leader on the POWER6 project. He received a B.S. degree in electrical engineering in 1984 from the University of Missouri at Rolla. Mr. Sharp has worked in POWER processor design and verification since 1987. He worked on memory subsystem designs for early POWER processor systems, and has focused on functional verification for five generations of POWER/PowerPC processors. In addition to his functional verification efforts, Mr. Sharp was the processor hardware verification leader for the POWER3 processor.

James K. Hsu *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (hsuj@us.ibm.com)*. Mr. Hsu is currently a Staff Engineer at IBM, working as a storage subsystem unit verification leader (fabric bus controller) in POWER5 and follow-on projects. He joined IBM in 1999 and worked on memory controller verification in POWER4 after receiving a B.S. degree from Duke University with a double major in electrical engineering and biomedical engineering. In 2001, Mr. Hsu received an M.S. degree in electrical engineering from Stanford University with a focus on computer hardware.

Bing-Lun Chu *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (bchu@us.ibm.com)*. Mr. Chu received a B.S. degree in engineering science from the State University of New York at Stony Brook in 1973 and an M.S. degree in electrical engineering from Cornell University in 1974. He joined IBM at Poughkeepsie in 1978, working on storage subsystem design for System/370*. He later focused on SMP storage subsystem functional verification. Mr. Chu pioneered the use of a random-driven/concurrent checking methodology that proved to be extremely successful in several IBM products, including the POWER4 storage subsystem and the zSeries server and its predecessors. He served as the team leader for the POWER4 and POWER5 storage subsystems. Mr. Chu is currently the chief architect for the zSeries memory subsystem verification.

Michael L. Behm *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (behm@us.ibm.com)*. Mr. Behm is currently a Senior Engineer; he joined IBM in 1978. He spent five years in S/390 manufacturing test, three years in the S/370* engineering laboratory, eight in S/390 core/vector processor verification, and the past ten years in POWER core/chip verification. Mr. Behm's current assignment is chip/core/system architectural and microarchitectural coverage verification.

Rebecca M. Gott *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (gott@us.ibm.com)*. Dr. Gott joined IBM in 1999; she is currently an Advisory Engineer with the IBM Systems and Technology Group. For the last four years, she has worked in the area of functional formal verification, and for the last three years has served as the team leader for functional formal verification within the IBM Advanced Processor Development organization. Most recently she has worked on the formal verification of the POWER6 microprocessor. In 1995, she received her Ph.D. degree in electrical engineering from Tulane University. Following her graduation, she served on active duty in the U.S. Air Force at Holloman AFV, New Mexico, working in the area of GPS development and test. Dr. Gott has recently taken a new position at IBM as the simulation leader for the zSeries processor subsystem.

Audrey D. Romonosky *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (romonosk@us.ibm.com)*. Ms. Romonosky is currently a Senior Engineer; she joined IBM in 1978. She received a B.S. degree in electrical engineering from Pennsylvania State University in 1978 and an M.S. degree in computer engineering from Syracuse University in 1985. She has worked in system verification since 1996 on POWER3-, POWER4-, and POWER5-based systems and is currently the pSeries system verification leader. Most recently, she received an IBM Outstanding Technical Achievement Award for her work on POWER4-based systems.

Steven R. Farago *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (sfarago@us.ibm.com)*. Mr. Farago joined IBM in 1998 and is currently a Staff Engineer. He received bachelor's degrees in computer engineering, mathematics, and statistics from the University of Florida in 1998, and an M.S. degree in computer science from the University of Texas in 2002. He has spent the majority of his career working on logic verification tools. Much of this work has focused on the coherency monitor, a system-level checking and coverage tool used in the verification of several processors, chips, and systems across the corporation. Mr. Farago is currently working on a new tool, the Hardware Developer's Workbench (HDWB).