Accelerating system integration by enhancing hardware, firmware, and co-simulation

K.-D. Schubert
E. C. McCain
H. Pape
K. Rebmann
P. M. West
R. Winkelmann

System integration of an IBM eServer™ z990 begins when a z990 book, which houses the main processors, memory, and I/O adapters, is installed in a z990 frame, Licensed Internal Code is "booted" in the service element (SE), and power is turned on. This initial system "bringup," also referred to as post-silicon integration, is composed of three major steps: initializing the chips, loading embedded code (firmware) into the system, and starting an initial program load (IPL) of an operating system. These processes are serialized, and verification of the majority of the system components cannot begin until they are complete. Therefore, it is important to shorten this critical time period by improving the quality of the integrated components through more comprehensive verification prior to manufacturing. This enhanced coverage is focused on verifying the interaction between the hardware components and firmware (often referred to as hardware and software co-simulation). Verification of the activities of these components first occurs independently and culminates in a pre-silicon system integration process, or virtual power-on (VPO). This paper focuses primarily on the hardware subsystem verification of the CLK chip [which is the interface between the central electronic complex (CEC) and the service element (SE)] and on enhanced co-simulation. It also considers the various environments (collections of hardware simulation models, firmware, execution time control code, and test cases to stimulate model behavior), with their advantages and disadvantages. Finally, it discusses the results of the improved comprehensive simulation effort with respect to system integration for the z990.

Introduction

Success in the server industry is directly related to the features, quality, and development costs of a product, and the time it takes to deliver that product to the marketplace. To that end, IBM implements a very efficient yet comprehensive test strategy to reach a high level of quality. Specifically, in the eServer* systems this validation process begins very early in the development process with the verification of the hardware subsystem and software designs, ending with hardware and software co-simulation. Subsequently, with the delivery of the first engineering hardware, the focus is shifted to post-silicon system integration and system test. System test then uses operating systems such as System Assurance Kernel

0018-8646/04/$5.00 © 2004 IBM

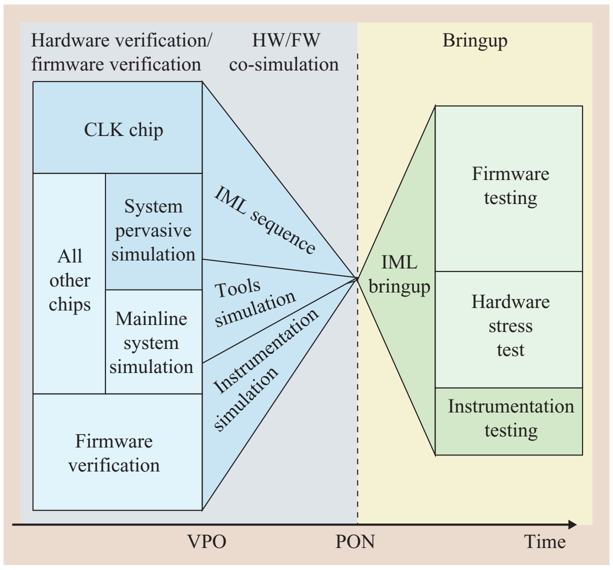


Figure 1

System integration bottleneck. (PON: power-on; VPO: virtual power-on.)

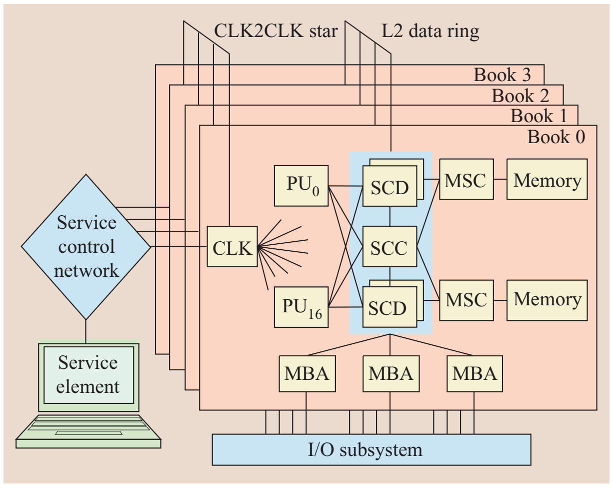


Figure 2

Overview of eServer system. (PU: processor unit; MSC: memory storage controller; MBA: memory bus adapter.)

(SAK), an internal IBM system exerciser, z/OS*, z/VM*, and Linux** to verify the architecture and complex system functions.

After review of postmortems on server products and analysis of each phase of the development cycle, one basic conclusion can be reached. The time from design start to the first hardware delivery is determined primarily by the complexity of the design, and that time cannot be reduced significantly. However, the time from first engineering hardware delivery to general customer availability is driven by testing activities and can be optimized. One way to optimize would be to completely overlap the system integration and system test phases. At first this approach appears very attractive, but several issues must be considered. Parallelization increases the development cost significantly because there is a cost premium on early hardware. Also, this is difficult because of the serial nature of the system integration phase, as stated earlier. For example, one problem involving either hardware or software this early in the system integration phase could gate all test activities, thus reducing testing efficiency to unacceptable levels. In the case of a hardware problem, it could take weeks or months to fix the problem and fabricate enough hardware to move past it. Finally, human resources are a constraint in conducting many parallel test activities, because more experts are needed than are available.

Despite these inherent limitations, parallelization is still used for most system integration and system test verification work because there are no alternatives at this time. However, one activity, initial machine load (IML), sometimes referred to as initial microcode load, is affected to the greatest extent by the problems mentioned above, and so cannot be overlapped because IML is the first step in system integration. Therefore, IBM strategy is to improve the quality of the system components at power-on time in order to reduce the amount of time needed to reach the parallel phase of system testing.

Another way to optimize would be to understand the nature of certain post-silicon environments and "move" the verification platform to a less expensive, more efficient and user-friendly platform. If this is done correctly, the negative consequences are minimal. This paper describes the efforts that have been made to significantly increase simulation coverage and verify function on the cheapest, simplest platform possible (blue areas in **Figure 1**) to reduce the time needed for system integration and test (green areas in Figure 1).

The next section provides an introduction to the system structure to help put subsequent explanations of the simulation environment in perspective. The following sections focus on the IML sequence and post-IML simulation environments, including their scope and possibilities as well as their limitations. Finally, a review of lessons learned and an outlook on future opportunities for improvement are presented.

System structure

The eServer system consists of hardware (HW) and firmware (FW) elements. The hardware components are the central electronic complex (CEC), consisting of a shared multiprocessor system (SMP) with a memory subsystem, an IBM ThinkPad* used as a service processor to control the system, the I/O subsystem, and a system control network consisting of various control chips and cables as shown in **Figure 2**. The power subsystem, which is currently not in the scope of the functional simulation activities, is not shown. This subsystem is pre-verified with special bringup vehicles (BUVs) prior to the full system power-on (PON).

In contrast to previous zSeries* systems, the structure of the CEC comprises four books. This packaging was necessary to combine up to 48 processor cores in the one system. This multibook structure presents new challenges because of the increase in complexity in the system control structure and the CLK chip. The CLK chip is the interface between the service control network [1] and all of the chips in a book. The main tasks of the CLK chip are starting the clocks, shifting the level-sensitive scan design (LSSD) chains of all chips, and providing a fast data path to the processors. To support the multibook structure, additional functions have been added to the z990 system. These functions include communication paths among all CLK chips in the system and the means of unfencing (logically separating books for simultaneous operations) the book-to-book interfaces.

To support all of these functions, the CLK chip has four interfaces, as shown in **Figure 3**, establishing the connections to the various targets:

- Service support interface (SSI) to the cage controller (CC).
- Clock service element (CSE) interface to each processor core in a book.
- Clock-to-clock interface (CLK2CLK) for multibook support.
- Serial interface (SIF) to the system control chip (SCC) and storage controller data (SCD) chip in a book.

In addition to the hardware pieces described above, the system also consists of a significant amount of firmware. One part of the firmware operates within the CEC and is responsible for providing microprocessor and I/O subsystem control. Another part of the firmware operates in the service processor and is used for functions such as system maintenance, error recovery, and multibook structure support.

IML sequence overview

An important function in a zSeries system is the IML. During IML the hardware is initialized and system clocks are started; then millicode and i390 code (henceforth referred to as CEC code) are loaded into the CEC. After the CEC code establishes an S/390*-architected reset state, it is ready to load an operating system and start applications. The IML control firmware operates on the service processor; since this firmware is in the critical path of all other testing activities, any problems and delays in debugging it during system integration have a direct effect on the overall time to market.

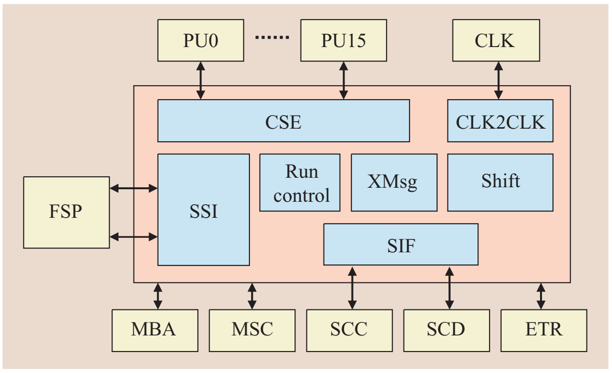
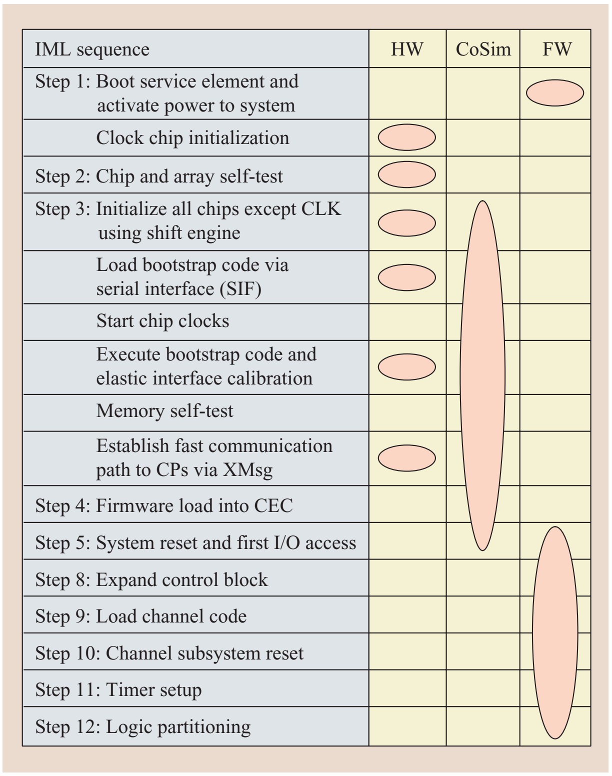


Figure 3

CLK chip overview showing all interfaces. (FSP: flexible service processor.)



IML sequence	HW	CoSim	FW
Step 1: Boot service element and activate power to system			●
Clock chip initialization	●		
Step 2: Chip and array self-test	●		
Step 3: Initialize all chips except CLK using shift engine	●	●	
Load bootstrap code via serial interface (SIF)	●	●	
Start chip clocks		●	
Execute bootstrap code and elastic interface calibration	●	●	
Memory self-test		●	
Establish fast communication path to CPs via XMsg	●	●	
Step 4: Firmware load into CEC		●	
Step 5: System reset and first I/O access		●	●
Step 8: Expand control block			●
Step 9: Load channel code			●
Step 10: Channel subsystem reset			●
Step 11: Timer setup			●
Step 12: Logic partitioning			●

Figure 4

Overview of IML sequence steps.

IML is a complex process that is broken down into multiple steps; for historical reasons, the numbering of the steps is not consecutive. **Figure 4** provides an overview of the different steps that make up IML. Before any interaction with the service element can be executed, the CLK chip must execute a power-on reset (POR) to initialize itself. This is a pure hardware function which is operated under the control of the run control engine[1] of the CLK chip. This engine retains information about the internal chip state and controls the clocks of the other chips in the system. After the CLK chip has reached the reset state, the connection to the service control network is established via the SSI.

At this point the service element code can access the CLK chip[2] and communicate with the other hardware macros (engines) on the chip. To verify that the CLK chip and the firmware in the service element are working in lockstep, the sense control engine allows direct access to the internal control registers of the CLK chip. When a stable connection is established, the service element then initializes the other chips by using the shift engine to access the internal shift latches (chains) of these chips and scan in the appropriate data. After the chips are in their initial state, the first piece of code, the bootstrap code, is loaded into the Level 2 (L2) cache [2] via the serial interface engine on the CLK chip. Thus far, the CLK chip has communicated only with the service element. With the start of the clocks to the other chips and the execution of the bootstrap code, a new communication path between the processor and the CLK chip is established; this is called the clock service element (CSE) interface. It allows a higher data rate between the service element and the processor chips when the special XMessage (XMsg) engine of the CSE on the CLK chip is enabled. With the end of IML step 3, this fast communication path is available; all of the megabytes of CEC code that have to be transferred from the service element to the CEC in steps 4 and 5 use this path.

In addition to the functions mentioned thus far, functions are available on the CLK chip that are required only in a multibook system; these have additional verification requirements. The CLK2CLK engine connects all CLK chips with one another. It is used to synchronize the status of the different books and exchange information between them. Finally, the serial interface engine and the clock service element interface are reused to fence and unfence the SCC cache ring [3].

Of course, the CLK verification does not cover the complete verification of the IML sequence. Each of the remaining chips has some logic incorporated to support the IML. These functions are related primarily to clocking and scanning. Since the functions interface with the CLK chip, additional environments[3] are used to verify each of these chips together with the CLK chip.

[1] Hardware macros on the chip are referred to as *engines*.
[2] The service element code interfaces with the books via the service control network and the CLK chip.
[3] These consist of collections of hardware simulation models, firmware, execution time control code, and test cases to stimulate model behavior.

In addition to the hardware functions mentioned so far, the IML sequence relies on firmware. The two main firmware components described in the previous section, the service element and CEC code, are verified separately. After the verification of these elements in a standalone environment, they are combined into the CEC simulator, CECSIM [4, 5]; however, since this environment contains no hardware model, some of the IML code must be bypassed.

The environments described so far already cover a significant portion of the IML sequence. As shown in Figure 4, the hardware verification focuses on aspects that are used mainly in steps 2 and 3, while the firmware verification covers primarily the service control network boot and steps 5 to 12. The third environment to be discussed is the one that focuses on the interaction between hardware and firmware. History has proven that when two separate groups design the firmware and hardware, there is a high probability of miscommunication. Therefore, hardware/firmware co-simulation (CoSim) was used and enhanced to verify the consistency of design assumptions and interfaces between hardware and firmware. This activity focuses on the IML steps in which the likelihood of such interface problems is relatively high, such as IML steps 3 and 4, in which firmware is performing low-level hardware commands. To ensure valuable overlap among the various activities, the co-simulation also covers parts of steps 2 and 5. In the following sections, we describe these activities in more detail and highlight points at which they have been skipped in one environment because they have been verified in other environments.

Hardware verification of the CLK chip

This section focuses on the comprehensive simulation strategy for the CLK chip using a multilevel verification approach. The scope of verification for each level is determined by solving the following optimization problem: Minimize the effort required to verify a set of properties of a system, including constraints such as start and finish dates.

The first step is to decide how the logic is to be partitioned so that the resultant hierarchical tree structure reduces the complexity within each level. The partitions are typically determined along the boundaries of macros, units, chips, or groups of chips. Each level can be considered a single problem in verification, and a detailed test plan can be constructed by analyzing the characteristics of the design and its interfaces. However, complete coverage of all test items on each level contradicts the goal of minimizing the resources generally—a target of the optimization problem mentioned above. This results in decisions such as skipping aspects of the test plan on one level in favor of integrated testing one level up in the hierarchy.

The design of the CLK chip is characterized by a large set of functions with many dependencies among them [6] which are difficult to address on a unit level. Therefore, it was decided to skip unit-level verification, leaving the following three verification levels for the CLK chip: macro-level verification, chip-level verification, and multichip-level verification.

Macro-level verification

Since macros are considered the smallest design entity, it is common practice to assign a single designer to implement their design. Each macro provides basic functionality that is accessible through stimuli on its interfaces. Since there is typically no detailed documentation on all of the internals of a macro, the most efficient way to verify it is to have the logic designer carry out the macro-level verification. This allows the designer to detect and correct many simple problems such as typographic errors on this level.

Chip/multichip-level verification

The chip-level verification is split into two phases, deterministic verification and random verification. Both environments run using the same set of tools and share common components such as drivers and monitors. In chip verification every chip interface is typically verified using such components. The multibook feature of the z990 eServer resulted in a design in which CLK chips are connected to one another; thus, instead of creating software for the CLK2CLK interface, models for two- or four-CLK chips were built to cover this interface. Using the CLK chip models in this way reduced the workload significantly for the price of maintaining multiple software behaviorals and models.

The majority of verification tests ran on these models. Before these tests were run, however, a chip initialization file was applied that is common to single-book and multibook configurations (i.e., each CLK chip was initialized with the same data). The model-build process of the CLK chip was extended to generate the initialization file by executing a partial POR sequence and storing the latch values in a file. It should be noted that the same initialization file is also applied in other simulation environments such as hardware/firmware co-simulation.

Deterministic environment

The deterministic verification phase (environment) was implemented before the random verification phase. It is based on a macro language that is built on top of the e programming language.[4] These macros define stimuli on

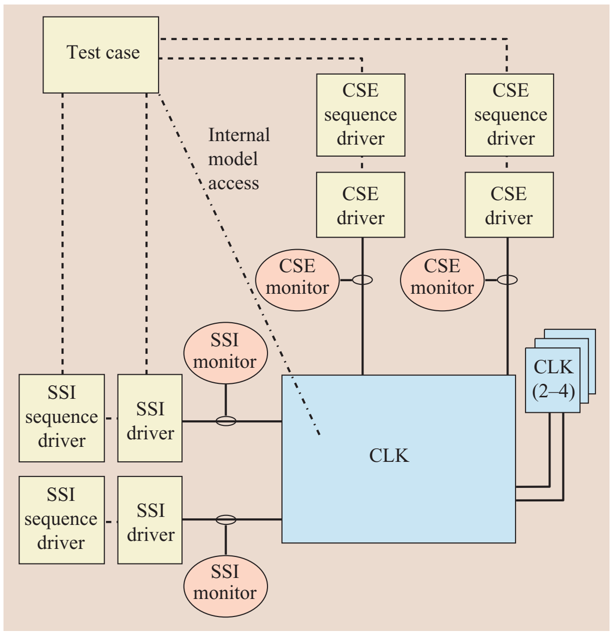


Figure 5

Deterministic CLK chip simulation environment.

the service support interface (SSI) and the clock service element (CSE) interface. The SSI is a command-based interface that provides read/write access to the internal registers of the CLK chip. The CSE interface connects the CLK chip to all processor cores in a book. Like the SSI, it provides read/write access to the CLK chip internal registers in the same book or via the CLK2CLK interface to the CLK chips in remote books.

All registers accessible via SSI and CSE are located in a set of engines: the shift engine, dealing with the shifting of internal and external chains; the sense control engine, with direct access to the internal clock control registers and run control functions; the XMsg engine,[5] with a fast data communication path between the service element and the processing units or cores; the clock-to-clock engine; and the serial interface engine.

Figure 5 shows the simulation environment for deterministic stimuli. Test cases are defined by a sequence of commands, with each command based on the macro language. The test case is then interpreted by the SSI or CSE sequence drivers, which use the lower-level interface drivers to execute individual commands. As a result, the interface drivers stimulate the model on the interface. The CLK model is checked for correct behavior, either by comparing expected values to the data returned on the

[4] Verification language provided together with the tool SPECMAN from Verisity, Inc., Mountain View, CA.

[5] The XMsg engine is a hardware interface on the clock chip that connects the cage controller to the CEC.

interfaces or by checking model internal register values. The checking code is implemented in the test case itself.

The main purpose of deterministic test cases is to ensure that every logic function is covered at least once. In addition, testing of these logic functions in a sequence that mimics zSeries applications such as IML is more complex and must also be verified. Although the development of these test cases is a manual, labor-intensive process, they are an important foundation for the verification of the CLK chip. These deterministic tests are self-checking; they are part of an extensive regression package that is extremely valuable in ensuring that design fixes do not impair the functionality of the chip. The following two examples show how deterministic tests are used and how they enhance the whole simulation process.

The first example describes the creation of the CLK chip reset file. This test case is a sequence of SSI commands to access internal model facilities; it comprises the following steps: First, the CLK chip is forced into the *Power-On-Reset* state. Second, stimuli from CLK chip neighbors are emulated. (Neighbors are the chips that are physically connected to the CLK chip in the real system.) Third, the sequence is completed by running the CLK chip into a well-defined *Power-On-Reset-complete* state. At this time, a dump of all latch values is written to a file and used in all subsequent environments that include the CLK chip, i.e., the CLK chip verification itself.

The second example deals with the scan/shift function of the CLK chip. This is a good example of a simulation which begins with simple test cases and increases its complexity over time. The primary contributor to the scan function is the shift engine of the CLK chip and its registers. The engine is accessible via the SSI and the CSE interface. The verification begins with the checking of the base functions, i.e., access of all registers in the shift engine and shifting of internal scan chains such as the CSE chain. It continues by providing test cases for complete and partial shifting, the locking mechanism for concurrent access via SSI and CSE, and the partial shift of external chains. Since the shift engine can be accessed from all processor cores in the system via a remote CLK chip, the next step is to verify the shift mechanism via the CLK2CLK interface, which includes the sequence for the unfencing of the interface. Finally, all of the simulation steps described above are the base for the simulation of the processor sparing sequence. In this case all steps which are initiated by millicode later in the real system must be modeled by a comprehensive simulation sequence.

Random environment

In the deterministic simulation environment illustrated above, scenarios are defined by test cases. These test cases are generally not fully deterministic, but support parameterization (i.e., choose to execute an SSI command on SSI A instead of SSI B). This type of parameterization improves the coverage, but from a more abstract point of view, these test cases keep their deterministic characteristics.

In the random environment we extend the idea of parameterization significantly and thereby cover a different state space. The basic idea is to concurrently stimulate the interfaces of the CLK chip with randomly selected sequences or just a single command. These sequences are divided into those that leave the state of the logic unchanged or at least in a known state after completion (i.e., nondestructive sequences such as a simple read command) and those that result in an unknown state due to triggered internal activity. The term *unknown state* refers in this context to a state that results from a transition that is too expensive to model and therefore leaves the checking code in an *unknown state.* Thus, these sequences are called destructive.

These destructive sequences are not desirable for the following two reasons:

1. Checking of these scenarios and their side effects is extremely complex and often adds no significant coverage.
2. Some of these sequences are not likely to happen in hardware because they are prevented by code.

However, it is desirable to run permutations of sets of sequences that leave the CLK chip in a known state at all times. These sequences are supposed to run concurrently. For instance, a typical scenario is a shift operation over the SSI interface in combination with some processors reading or writing to the sense control engine in the local book, and some accessing engines in remote books with recoverable error injection. Another scenario would be to have SSI and multiple processors concurrently[6] request the lock to the shift engine.

All sequences applied during a random simulation have variation built in for variables such as node number, register number, or engine number. The test-case manager maintains the status of all concurrently running sequences, and the resource manager addresses constraints between sequences (i.e., determines whether two sequences are allowed to be executed concurrently). On the basis of this infrastructure, the constraint-driven generation works as follows. After a randomly selected number of cycles, a new sequence is generated by the sequence generator. At this point all degrees of freedom (i.e., engine number) have been eliminated. The list of required resources

[6] The shift engine has a lock mechanism that ensures exclusive access to it by a single source.

for this sequence is then computed. A constraint solver evaluates whether it is legal to start the new sequence while the current ones are still being executed, according to an algorithm based only on resources. If there is a conflict, the sequence is rejected; otherwise a new thread is started, and the sequence is executed within it. The execution of sequences is handled with no interaction of the test-case manager. Sequences have built-in checking implemented and are driven by SSI and CSE sequence drivers. The sequence drivers utilize the SSI and CSE drivers. This process is similar to deterministic test-case execution, and some functionality is reused.

Thus far the test-case manager, sequence drivers, and interface drivers have been discussed. The other components shown in **Figure 6**, such as the monitors and the reference model, make up the checking code for the CLK chip[7] environment. Checking is based on self-checking commands and is implemented via message passing.

The following example illustrates this checking strategy: for the case "Write command to register #1 of the sense control engine in the local book via processor #4," it begins with the CSE driver #4 sending the command over its CSE bus. The corresponding CSE monitor reads the data and starts the protocol checking, which will finish on completion of the command or in case of an error. While the CSE bus is being monitored, a message is assembled that comprises information on the source processor, target engine, target book, data, etc.; this message is sent forward to the CSE interface. The CSE interface checks its content, i.e., for a valid target engine. If this check is successful, it forwards the message to the reference model. Within the reference model, the message is associated with the corresponding engine, register #1 is updated in the reference model, and further actions are triggered if needed. On completion of these actions, a message is sent back to the CSE monitor via the same path. The CSE monitor then compares the predicted result to the response from the hardware model.

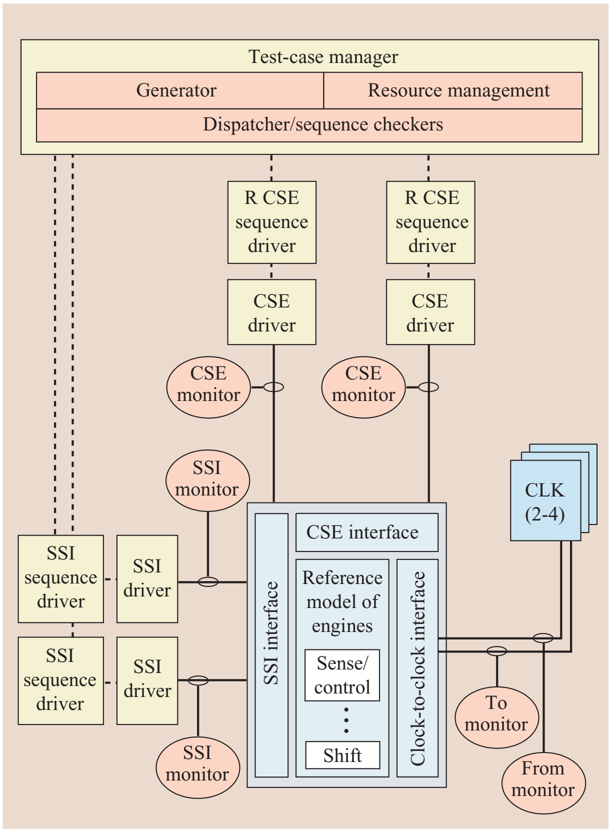


Figure 6

Random CLK chip simulation environment.

Hardware/firmware co-simulation

Acceleration environment

The terms hardware/firmware (HW/FW) co-simulation and VPO imply that this activity begins before real hardware is available [7, 8]. To be successful, the VPO concept requires all firmware for the initial bringup to be available and simulated by the time at which hardware design is fixed. This concept was enhanced for the z990 system design cycle by moving the VPO start date earlier, before hardware design is fixed, making available more time for "pre-silicon" co-simulation test activities. The difficulty in following this strategy is that during this time models traditionally change very frequently and are too unstable for meaningful co-simulation. Therefore, a VPO model snapshot process was invented to closely control the model definition and its associated data files that describe clocking and initialization. Further, the accelerator of choice continues to be the Cadence CoBALT** Ultra system, because it has the performance and capacity necessary to handle very large zSeries models. However, since a fully configured 12-processor book cannot be built and loaded into the CoBALT Ultra accelerator, tradeoffs were made by deleting noncritical chips for the mainline IML path. The model most frequently used during the VPO of the latest zSeries 990 system comprised one processor chip with two cores, the memory subsystem for one book, one I/O adapter chip, and the CLK chip.

To execute firmware against that hardware model, some way of modeling the service control network is needed. The simplest solution is to strip the service control network down to just the service element, executing the firmware code. The network itself is replaced by a simulation-only software layer that connects the output

[7] Some checking is done by the test-case manager, as was previously explained.

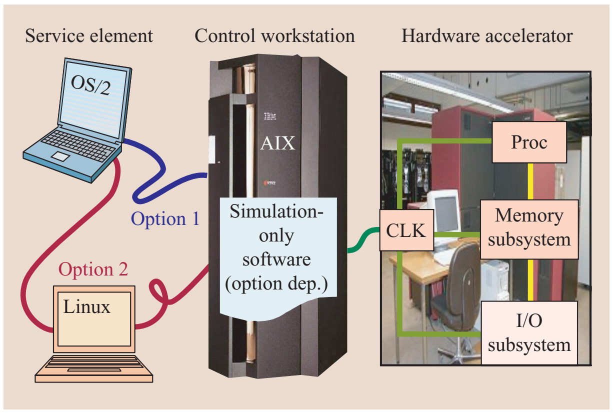


Figure 7

Co-simulation environment: two options.

from the laptop to the correct interfaces in the hardware model on the accelerator. **Figure 7** shows the simulation environment, which consists of the laptop running the firmware, a workstation to host the special software layer reusing components from other hardware simulation activities, and the accelerator into which the hardware model is loaded. The software layer establishes a socket connection to the laptop, and any data traffic from the laptop is translated into commands for a CLK chip internal parallel bus [7, 9] by mimicking the function of the replaced service control network. The serial SSI interface that has been verified extensively during the CLK chip verification is bypassed in this environment to gain additional simulation speed.

With this setup, the service element code executes as though it were targeting a real hardware z990 system. The major differences between the service element running in simulation and a connection to a real hardware system are twofold. Many of the timeout settings must be modified for simulation, since the responses in simulation take significantly longer. In addition, all communication between the service element and the power subsystem must be handled within the firmware, emulating real behavior to some extent.

The setup described thus far has been used for most of the co-simulation activities. The environment is capable of complete firmware-to-hardware model interaction, and it has no constraints that would prevent it from running through the complete IML sequence from step 2 to step 12. However, the IML is inherently a sequential process, and simulating the whole sequence would take at least a month of pure runtime, even on a hardware accelerator capable of executing some 200,000 model cycles per second.

Simulation of IML tasks

For all practical purposes it is not acceptable to have a turnaround time of more than a couple of hours, since debugging of problems becomes impossible otherwise. Consequently, this environment is used only for those parts of the IML process that cannot be verified adequately by other means. This restricts the co-simulation to IML steps 2 through 5, in which most of the hardware/firmware interaction takes place. In addition, some elements which require many simulation cycles can be broken out of this process and verified in a standalone manner. This allows us to break the sequential process of IML into tasks that can be attacked in parallel, avoiding delays when problems are gating progress. This is important because the simulation has to be completed by the time real hardware arrives on the test floor.

One of the first tasks to be performed is to prepare a consistent system model state obtained from preceding verification activities such as the CLK chip simulation and CEC subsystem simulation. This shortcut approach does not guarantee that the same initial state can be reached by executing the code on the service element. However, it is close enough to the real machine setup that all subsequent activities behave in the same way as on a real system. This initial state is the starting point for IML step 3 bypassing the requirement to run IML step 2 and the complex array reset process (ABIST reset) usually executed at the beginning of step 3. Thus, using this shortcut method, the verification of all IML step 3 activities can be started immediately, without waiting for debugging of bypassed functions, in effect parallelizing step 2 and step 3 debugging.

Since most of the individual steps have already been verified in previous hardware or firmware verification activities, this co-simulation environment finds the problems at the interfaces or in transitions between IML steps. For instance, a class of problems is found when hardware initialization is completed and fast communication between the firmware and the hardware via the XMsg engine must be established. Another hot spot is the engineering data, which describes the initial values for all hardware registers. The data is provided by the hardware design team and then processed and to some extent interpreted by the firmware group. During the initial scan operations, this data is shifted into the registers. Since the shift process has been verified during the CLK hardware verification, this typically exposes no new problems. However, being able to shift data into registers does not guarantee that the data values in the registers are correct. As soon as the chip clocks are started, any errors in this data will likely appear as errors in the hardware, which will detect these inconsistencies within the first few hundred cycles. Other areas in which problems are found are the firmware sequences that control the calibration of the elastic interfaces and also the section that is responsible for initializing the memory and performing the memory self-test. With the execution of the code reset at the end of step 5, this activity is completed.

In parallel with the simulation activity described thus far, initialization shortcuts were used. However, processes that were bypassed must be verified as well; one of these is the array reset function. Starting from a state in which the array state is all zeros and only a minimal amount of surrounding control logic is set up, array built-in self-test (ABIST) [10] reset procedures are executed. After this process is completed, the array state is verified; it should yield the same state as the shortcut procedures started within IML step 3, proving those assumptions that led to the shortcut in the original task.

Another activity is the verification of IML step 3 to step 5 with I/O chips. These I/O chips were initially left out of the model in order to make the model smaller, thereby improving the simulation performance. Also, since much of the IML sequence is independent of these chips, any problems with the engineering data of the I/O chips can be corrected while making progress on the IML without I/O. It is critical to have I/O chips in the model in IML step 5, where the I/O hardware reset is executed and the STI links are initialized.

The environment used for the IML co-simulation can also be used to verify tools that are used later during bringup as debugging aids. The tools are executed on the service element and allow different types of accesses to the hardware. They typically use scan operations to read or write certain registers or complete arrays. Tools requiring millicode routines to be executed on their behalf cannot be started before the IML simulation has reached a certain point in IML step 4. While tools verification is often viewed as a side activity, its impact on bringup and integration can be huge. Also, after they have been successfully simulated, some of these tools have even been used for debugging the IML co-simulation process itself.

Additional simulation environments

The requirements of multibook simulation are greater than those of the tasks discussed previously. The models are by nature at least twice as big, assuming a two-book structure, which is an issue if acceleration capacity and acceleration time are limited. In addition, the simulation environment is more complex, since the simulation-only software layer must handle two or more independent communication streams. After setting up the environment and finding some simple setup problems in the service element, we terminated that activity, primarily because almost all activities in IML steps 2 to 5 for multibook are identical to those for a single-book IML. In addition, all of the differences have already been verified in the CLK multibook environments. Owing to limited accelerator access, we stopped early in favor of the other activities described in this paper. In hindsight, this was the correct decision, because during system integration there were no problems that could have been found in this environment.

The processes described thus far have used an environment in which the complete service control network between the service element and the CLK chip has been replaced by a software layer for simulation (Figure 7, option 1). However, the service control network is not trivial, since it contains a few chips and executes a significant amount of code as well. Therefore, in order to increase the simulation coverage of the co-simulation environment as discussed thus far, we decided to include at least the code that is executed within the service control network. Because the code can be executed in a Linux environment, we added another component in our environment, as pointed out in option 2 of Figure 7. On one side, the Linux system is connected to the service element, and on the other side to the simulation-only software layer, which had to be slightly modified to support this configuration. As a test case, the same sequence of IML steps 3 to 5 was executed, focusing only on problems in the firmware code that is executed on the Linux system, since everything else had previously been verified in the simpler environment.

Results

The first attempts to establish a VPO-like process and to use hardware-firmware co-simulation for “virtual system integration” or “pre-silicon” were made in 1998. Since then, the process has been improved continuously in zSeries systems. With the zSeries 990 eServer, a major breakthrough in VPO simulation has been achieved. Most of the activities planned up front have been successfully co-simulated prior to real PON, many of them for the first time ever (i.e., IML steps 4 and 5 with I/O chip and bringup tools). The overlap among hardware verification, firmware verification, and co-simulation has enabled a very fast bringup of the simulated functions.

During the co-simulation activities after VPO, as shown in **Figure 8**, a significant number of problems were eliminated from the system. A total of 120 code problems, seven hardware problems, and 91 problems in the simulation environment were found. Hardware problems found at this point, which is after tape-out (a point in time at which the physical design layout is ready for chip fabrication) cannot be fixed until the next tape-out; however, it is important to state that the problems that would have had a severe impact during system integration were circumvented via firmware changes, and, more significantly, the circumventions were verified prior to PON. Thus, by identifying these verification problems and

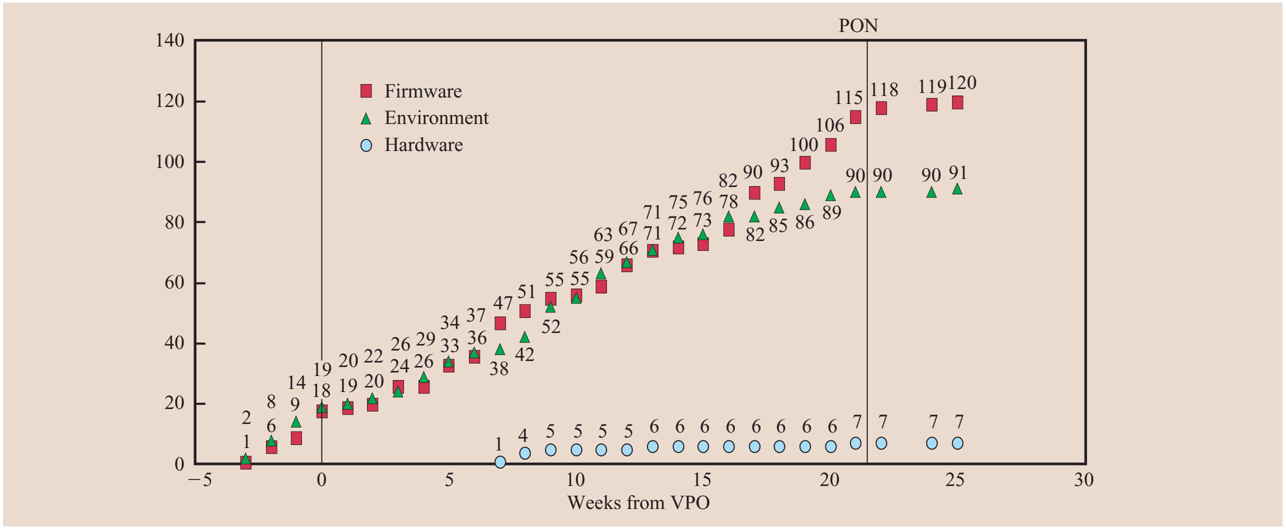


Figure 8

Co-simulation results.

installing firmware changes in the system driver, the VPO process has significantly shortened the time required for real system bringup. Also, if a severe gating problem was found, weeks would be saved by immediately releasing an emergency tape-out.

Instrumentation

PICOSIM environment

Thus far, this paper has described simulation activities aimed at minimizing the IML bringup time. Since other activities could benefit from similar work, it was logical to extend the co-simulation environment discussed previously in order to achieve additional simulation coverage. This initiative has led to the development of the post-IML co-simulation (PICOSIM) [11] environment, which encompasses all CEC code (service element code, i390 code, and millicode) and the hardware models that have already been used for the co-simulation. PICOSIM allows the verification of post-IML activities that would normally be executed for the first time during bringup and system integration.

It is very difficult to set up an environment that is capable of modeling system functions after IML is complete. This had been accomplished once, with the simulation effort for the IBM Enterprise System/9000*, but since the shift in technology to CMOS, this environment has been dismantled [12]. One way to create such an environment would be to run a "full" IML on the system model; however, as already mentioned, this would take an extremely long time even on a CoBALT Ultra accelerator system. To overcome this problem, a process has been developed by which z/CECSIM [5], a microcode simulator that runs at zSeries speed, can be configured to match the model in PICOSIM. To synchronize both environments, the following four steps must be executed:

- *Execute the IML.*
  The first step is to execute IML in z/CECSIM with a configuration that corresponds to the one in the PICOSIM environment. During the IML sequence, millicode and i390 code verify the hardware configuration by assigning processor units (PUs), system assist processors (SAPs), and channel path IDs (CHPIDs), and allocating a section of memory to allow communication among processors, i390 code, and millicode subsystems (also known as the hardware system area, or HSA). The target configuration for post-IML verification for the z990 was a 1+1, with one PU and one SAP, which was the XSAP (Master). This configuration would match the one-cycle model used that includes one PU chip (two cores), one system control chip (SCC), four SCDs (L2 cache chips), two memory storage controllers (MSCs), a memory macro, a two-cycle MBA, an STI-M, and a CLK chip.
- *Create a snapshot.*
  Second, a snapshot of all microarchitected facilities and the associated data areas in memory is created from z/CECSIM. In this particular application, after IML step 11 all of the programming model data for millicode and i390 code is saved to a file in z/CECSIM. This includes general-purpose registers, the processor recovery hardware state, millicode registers, and timing facilities.

The data that resides in memory HSA is also saved to a file.

- *Load the hardware model.*
  The model is pre-initialized with the data from the CEC subsystem hardware simulation, ensuring a post-IML clock running state for the hardware-only environment. Then the hardware model initialization is updated with this z/CECSIM snapshot of the associated ECC and parity information saved in the earlier step.
- *Transfer the memory image into the hardware model.*
  By using the Memmove program (an IBM internal tool used in verification environments to load and manage the storage hierarchy), the large binary file, which contains 120–150 MB of data, can be loaded into the memory macro.

Upon completion of these steps, the clocks are started and the instruction-fetch process begins, with each hierarchy of cache requesting data from the level above it until the data is found in memory. The instructions and operand data are fetched and placed in the pipeline, and instruction execution continues as usual. The millicode and i390 code go to the idle loop routines until they receive an outside stimulus such as a system restart [13]; the restart program status word (PSW) is then set up and points to a small ESAME program or operating system bootstrap routine. For the z990 system, the instrumentation millicode has been chosen as the first test case for this new environment.

Verification of instrumentation

For the z990, instrumentation millicode verification was selected because it had tremendous potential for savings by exploiting the PICOSIM environment. Instrumentation is the mechanism used to measure the performance of the IBM eServer z990 system. This is achieved by iterative execution of instruction streams targeted to stress particular hardware functions and to collect its performance characteristics. The collection is done by capturing selected signals and storing them in hardware arrays within the processor. Each time the arrays are filled, millicode routines are invoked to move the data from the arrays to main memory. The data is later used for the analysis of important metrics such as CPI (cycles per instruction), cache misses, and pipeline stalls.

Since instrumentation is for internal use only, it has little dedicated hardware. Because the hardware arrays that are used to collect the instrumentation data require significant real estate on the chip, they are shared with another function. Since the arrays are normally used to collect debugging trace data, no debugging traces can be obtained while in instrumentation mode. For this reason, instrumentation is extremely difficult to debug.

Historically, instrumentation has suffered from a lack of conventional debugging and verification tools, exacerbated by complex code and hardware interactions. Therefore, on previous projects this function could be tested only by using real hardware, because conventional firmware verification environments do not include support for the instrumentation hardware.

The comprehensive PICOSIM environment currently provides the capability to stress the complex interactions among hardware, i390 code, and millicode, since the model contains the logic design, including the special instrumentation hardware as well as the post-IML checkpoint from z/CECSIM. In addition to being comprehensive, the PICOSIM environment, like any simulation environment, is dynamic and flexible. It allows hardware facilities and storage locations to be viewed and altered during test-case execution. In contrast to execution on real hardware, new test cases can be loaded into storage without bringing the environment down, and local fixes in the millicode can be applied, using z/CECSIM, without generating a new image.

While debugging of the environment has been performed on a software simulator, the environment can easily be moved to hardware accelerators to gain the required speed as soon as extensive simulation runs are required. Verification using the PICOSIM environment uncovered numerous bugs in millicode, i390 code, and the hardware for this system. As a result, the bringup time for the instrumentation function has been reduced from nearly three months on the predecessor system to only two and a half weeks.

Future work

The environments that have been described thus far have pushed the limits of simulation further out. The approach of moving activities that have traditionally been executed during system integration into simulation and, within simulation, into the smallest possible environment has proven to be successful. As the complexity of future systems increases, more work must be moved into simulation just to maintain the project cycle at current levels. Targets for simulation, such as IML and instrumentation, which have a high potential for savings in bringup, have already been covered. However, the investment required for this effort can now be directed to address other scenarios with much less effort but sufficient potential.

I/O-related operations are certainly a good example of the requirement for future enhancements. Another area that is already under investigation involves bringup tools of all types. While a few bringup tools have already been verified, the verification of additional ones would make a difference during bringup as well. To mention another example, error path testing has been under examination for some time now, and may be feasible with current environments.

To free the required resources for addressing the items mentioned above, it is critical to improve the handling and efficiency of the environments. This can be achieved by tool improvements such as more automation and faster turnaround time, by verifying certain functions in simpler environments, or even by changing the design to minimize simulation requirements. This would be the ultimate extension of the strategy presented here.

Conclusion

In this paper we present a new strategy that bridges hardware and firmware verification with the goal of optimizing system integration. The effort was driven by the need to save time, reduce development cost, and enhance product quality. Through analysis of our existing simulation environments, enhancements were identified and implemented. All hardware and firmware components can now be covered in the same environment; to achieve a high overall efficiency, certain verification pieces have been moved into the smallest and therefore least expensive environment possible. Only minimal overlap has been retained to ensure coverage of the boundaries between simulation environments.

This new strategy has been successfully implemented and executed for the eServer z990. The combined effort of the entire development team has resulted in a significant improvement with regard to system integration time. A reduction of about eight weeks was achieved compared with the original system integration plan based on data and experience from previous projects. Simulation efforts using PICOSIM have resulted in similar time savings, especially when subsequent chip tape-outs are required, because sufficient feedback from performance measurements is available much earlier, so that these tape-outs can take place earlier and with greater confidence of reaching customer quality.

IBM's investment in hardware/firmware co-simulation is significant, but well compensated by the result of time saving and reduction in engineering hardware required for bringup. Also, a substantial portion of the investment was spent on accelerator hardware that will be reused in future projects.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Linus Torvalds or Cadence Design Systems, Inc.

References

1. F. Baitinger, H. Elfering, G. Kreissig, D. Metz, J. Saalmueller, and F. Scholz, "System Control Structure of the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 523–535 (July/September 2002).
2. P. Mak, M. A. Blake, C. C. Jones, G. E. Strait, and P. R. Turgeon, "Shared-Cache Clusters in a System with a Fully Shared Memory," *IBM J. Res. & Dev.* **41,** No. 4/5, 429–448 (July/September 1997).
3. P. Mak, G. E. Strait, M. A. Blake, K. W. Kark, V. K. Papazova, A. E. Seigler, G. A. Van Huben, L. Wang, and G. C. Wellwood, "Processor Subsystem Interconnect Architecture for a Large Symmetric Multiprocessing System," *IBM J. Res. & Dev.* **48,** No. 3/4, 323–337 (May/July 2004, this issue).
4. M. Stetter, J. von Buttlar, D. Chan, D. Decker, H. Elfering, P. Gioquindo, T. Hess, S. Koerner, A. Kohler, H. Lindner, K. Petri, and M. Zee, "IBM eServer z990 Improvements in Firmware Simulation," *IBM J. Res. & Dev.* **48,** No. 3/4, 583–594 (May/July 2004, this issue).
5. J. von Buttlar, H. Böhm, R. Ernst, A. Horsch, A. Kohler, H. Schein, M. Stetter, and K. Theurich, "z/CECSIM: An Efficient and Comprehensive Microcode Simulator for the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 607–615 (July/September 2002).
6. G. Doettling, K. J. Getzlaff, B. Leppla, W. Lipponer, T. Pflueger, T. Schlipf, D. Schmunkamp, and U. Wille, "S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure," *IBM J. Res. & Dev.* **41,** No. 4/5, 405–428 (July/September 1997).
7. S. Koerner, M. Kuenzel, and E. C. McCain, "IBM eServer z900 System Microcode Verification by Simulation: The Virtual Power-On Process," *IBM J. Res. & Dev.* **46,** No. 4/5, 587–595 (July/September 2002).
8. J. Kayser, S. Koerner, and K.-D. Schubert, "Hyper-Acceleration and HW/SW Co-Verification as an Essential Part of IBM eServer z900 Verification," *IBM J. Res. & Dev.* **46,** No. 4/5, 597–605 (July/September 2002).
9. S. Koerner and S. M. Licker, "Run-Control and Service Element Code Simulation for the S/390 Microprocessor," *IBM J. Res. & Dev.* **41,** No. 4/5, 577–580 (July/September 1997).
10. Gary A. Van Huben, "The Role of Two-Cycle Simulation in the S/390 Verification Process," *IBM J. Res. & Dev.* **41,** No. 4/5, 593–599 (July/September 1997).
11. E. C. McCain, "Post Initial Microcode Load Co-Simulation Method System, and Program Product," U.S. Patent Reference No. POU920040001US1, filed May 11, 2004.
12. D. F. Ackerman, M. H. Decker, J. J. Gosselin, K. M. Lasko, M. P. Mullen, R. E. Rosa, E. V. Valera, and B. Wile, "Simulation of IBM Enterprise System/9000 Models 820 and 900," *IBM J. Res. & Dev.* **36,** No. 4, 751–764 (July 1992).
13. IBM Corporation, *z/Architecture Principles of Operation* (SA22-7832); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.

Received September 18, 2003; accepted for publication March 2, 2004; Internet publication May 27, 2004

Klaus-Dieter Schubert *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (kdschube@de.ibm.com).* Mr. Schubert is a Senior Technical Staff Member in the Hardware Development organization in the Boeblingen laboratories. He received his M.S. degree in electrical engineering in 1990 from Stuttgart University (Germany). He subsequently joined IBM in Boeblingen and has been responsible for hardware verification of multiple S/390 systems. He was the technical leader for hardware verification of the z900 2064 system and is currently responsible for system verification, including the VPO activities for all eServers including the z990 system. Mr. Schubert is the author of three patents; he has received two IBM Outstanding Technical Achievement Awards for his work on zSeries verification.

Edward C. McCain *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (mccain@us.ibm.com).* Mr. McCain is currently a Senior Verification Engineer, team leader for the design of PICOSIM, z990 bringup and development function verification leader, and team leader for the S/390 emulation program. He joined IBM in 1982 and has worked on engineering systems testing for the IBM 308X, 3090, ES/9000, and the S/390 G3, G4, G5, and G6. He has received a Leadership Award for his work on PR/SM and MPG, a Division Award for his work on the ES/9000, Excellence Awards for his work on S/390 Parallel Sysplex EDVT testing, S/390 G4 functional test leadership, and S/390 G6 EST project leadership, and two IBM Outstanding Technical Achievement Awards for his work on zSeries 900 and 990 verification, virtual power-on, and bringup.

Hermann Pape *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (hpape@de.ibm.com).* Mr. Pape received his M.S. degree in electrical engineering in 1983 from the University of Siegen (Germany). In 1984, as a microcode development engineer, he joined IBM Finance Systems, where he was the microcode project leader for the IBM 4725 Statement Printer. In 1996, he joined the zSeries hardware development group. Mr. Pape was certified as a Project Management Professional (PMP) by the Project Management Institute (PMI®) in 1999; he has since worked with the zSeries system simulation group and is currently the team leader for hardware/firmware co-simulation efforts during the VPO phase.

Karin Rebmann *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (krebmann@de.ibm.com).* Mrs. Rebmann is a Staff verification engineer in the zSeries hardware development group. She received an M.S. degree in medical computer science from the University of Heidelberg in 1982, joining the IBM Development Laboratories that same year. Mrs. Rebmann holds one verification patent and has been the team leader for zSeries CLK chip verification since 1998.

Patrick M. West *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (pmwest@us.ibm.com).* Mr. West is currently a millicode design engineer for zSeries servers. He joined IBM in June 2001 after completing his B.S. degree in electrical and computer engineering at Rutgers University.

Ralf Winkelmann *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (rwinkel@de.ibm.com).* Dr. Winkelmann is a verification engineer in the Hardware Development organization in the Boeblingen laboratory. He studied computer science at the University of Applied Sciences Braunschweig/Wolfenbüttel, Germany, and received his diploma in 1994. In 1999 he received his Ph.D. degree from the University of Greenwich, London, UK. Since 2001 he has worked for IBM in hardware verification.