IBM eServer z990 improvements in firmware simulation

M. Stetter
J. von Buttlar
P. T. Chan
D. Decker
H. Elfering
P. M. Gioquindo
T. Hess
S. Koerner
A. Kohler
H. Lindner
K. Petri
M. Zee

With the IBM eServer™ z900, simulation methods and tools for verification of code that is to be embedded in the memory of the system (firmware) were introduced. Since that time, firmware developers have simulated their code prior to the availability of new system hardware components, thereby reducing the time required to bring a large computer system to market. With the z990 system, code simulation efficiency has been improved. The simulation coverage for host and service firmware has been increased from approximately 60% in the z900 to 85% in the z990 by introducing new concepts and extensions. For the first time, the central electronic complex (CEC) firmware simulator, CECSIM, has been enabled to run code in a logical partition (LPAR). This was a prerequisite for code verification of the intra-CEC connectivity, HiperSockets™. For verification of HiperSockets, a Linux® operating system is loaded into an LPAR. Code verification is accomplished more easily, more effectively, and with better coverage using Linux debugging features because of the ease of performing functional tests with Linux. Another major improvement was the connection of the channel code simulator for the networking I/O adapter OSA-Express to the CECSIM environment to provide a comprehensive verification that covers the entire path of firmware interaction between the CEC and the I/O channels. For the simulation of card control code, a combined software and hardware verification approach was introduced. The overall functionality was verified with a system simulation model, and the base hardware accesses were verified by attaching real hardware. In addition, the cage controller code was integrated into the simulation environment. As a result, the firmware interfaces between the support element (SE) and the cage controller as well as between the cage controller and the hardware have been tested.

Introduction

The application of simulation techniques for the z900 system, the predecessor of the z990, significantly reduced the time required for development of the z900 [1–3]. On the basis of an intensive analysis of the z900 system microcode verification by simulation, an improved firmware simulation concept was implemented for the z990 system. This resulted in an increase in verification coverage of the z990 system code, additional cost savings, and a shorter system development cycle when compared with the z900 system. The new key elements of the z990 firmware simulation concept are described in this paper. Besides enhancements to existing components, such as the firmware simulator, CECSIM, completely new firmware simulation environments have been established. **Figure 1** shows an overview of the new components.

CECSIM enhancements for the z990

The use of the firmware simulator CECSIM during the development of the IBM eServer* z900 was very successful [3]. It was used by about a hundred users at four IBM locations. At the end of the project, CECSIM users

0018-8646/04/$5.00 © 2004 IBM

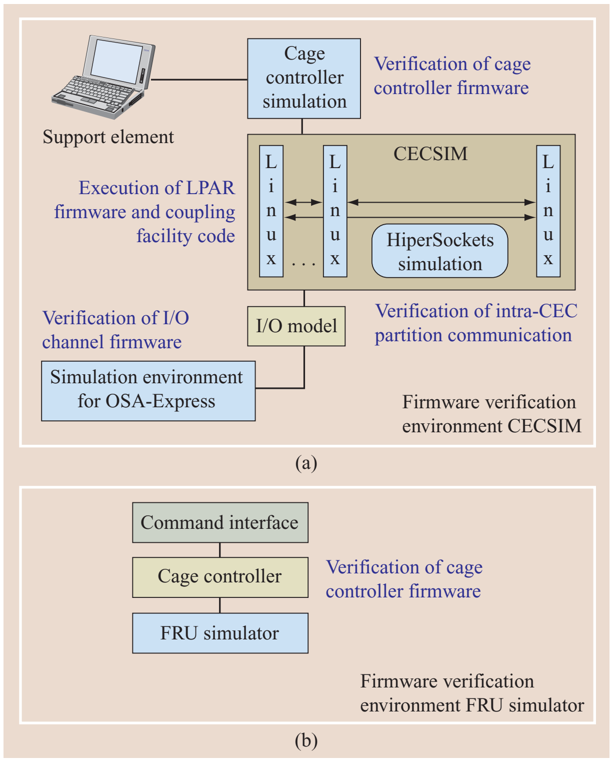
Cage controller simulation
Verification of cage controller firmware
Support element
CECSIM
Execution of LPAR firmware and coupling facility code
Linux
Linux
Linux
HiperSockets simulation
Verification of I/O channel firmware
I/O model
Verification of intra-CEC partition communication
Simulation environment for OSA-Express
Firmware verification environment CECSIM
(a)
Command interface
Cage controller
Verification of cage controller firmware
FRU simulator
Firmware verification environment FRU simulator
(b)

Figure 1

Generic view of two new concepts in z990 firmware simulation: (a) New concepts and (b) new facilities (shown in blue).

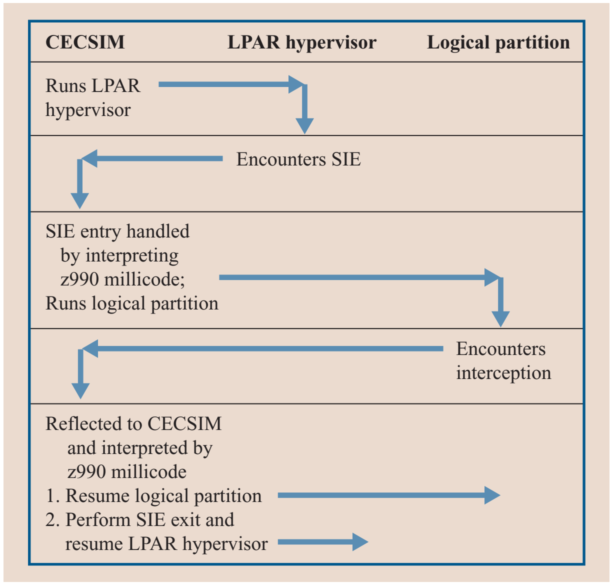
CECSIM
LPAR hypervisor
Logical partition
Runs LPAR hypervisor
Encounters SIE
SIE entry handled by interpreting z990 millicode; Runs logical partition
Encounters interception
Reflected to CECSIM and interpreted by z990 millicode
1. Resume logical partition
2. Perform SIE exit and resume LPAR hypervisor

Figure 2

CECSIM support to run a simulated logical partition (LPAR).

requested a number of enhancements to the simulator in order to further increase productivity. One request was to simulate code that runs in a logical partition (LPAR). The logical partition concept is equivalent to running several independent systems, each with its own system resources and operating systems, within a single physical z990 system. The capability to simulate code in LPARs would become very important, since the z990 is the first zSeries* system that runs only in LPAR mode.

During the z900 project we could simulate the LPAR hypervisor (the control program that manages LPARs in a zSeries CEC) in CECSIM together with the underlying firmware. However, the simulator did not support running code *within* a logical partition. Therefore, a shortcut was implemented: Whenever the hypervisor attempted to run a logical partition, CECSIM returned a disabled wait program status word (PSW) in the partition to the hypervisor. Thus, at least this code path in the hypervisor could be simulated, but simulation was not available for the many other conditions within a partition that required assistance by the hypervisor.

When the z990 was developed, CECSIM was enhanced to actually run a logical partition controlled by the LPAR hypervisor. This capability allowed another level of complexity to be tested with firmware simulation: First, a variety of assembler programs were used to trigger many code paths in the LPAR hypervisor. Second, the complete coupling facility control code was run in simulation. Third, a RAM disk image of Linux** on zSeries systems was used to test HiperSockets*, which is a technology that provides high-speed communication between LPARs within a CEC. Each of these environments increased the test coverage for millicode and i390 code, which are always an integral part of the simulation environment.

The LPAR hypervisor runs a logical partition by issuing the instruction *start interpretive execution* (SIE) [4]. This instruction is implemented in millicode, which is interpreted by CECSIM, as are many other instructions (e.g., *start subchannel*). The simulator now supports the hardware (HW) facilities used by SIE, such as the relocation of partition storage and partition timers. When SIE millicode is complete, all information relevant to describing the partition is known to the simulator. In particular, the PSW now points to the next instruction to be executed within the partition. CECSIM itself uses SIE to run the simulated processor, so there are now three logical levels: CECSIM, the LPAR hypervisor, and the LPAR. However, CECSIM sets up its own SIE state description such that it *directly* runs the simulated LPAR, as outlined in **Figure 2**.

When running a processor using SIE, CECSIM has to set up interception controls to ensure that it is notified of certain instructions or events in the simulated processor [3]. When the simulated processor *itself* uses SIE to run a logical partition, information from the hypervisor state description must be merged with the CECSIM state description. For instance, the interception controls of the processor state description are merged into the CECSIM state description. This ensures that CECSIM receives control if either it or the simulated processor requests this interception. If the interception is requested by the processor, the event is handled by interpreting the millicode that switches the processor from the LPAR back to the LPAR hypervisor.

HiperSockets testing with Linux

Before the z900 system generation, the benefits of the LPAR concept could not be exploited until the early "bringup" hardware (system prototype) was available. Now, operating systems can be loaded into partitions in the CECSIM environment, and developers are thereby able to use operating system functionality to verify their code. During z990 system development, this approach was applied for the verification of HiperSockets firmware, which is described in this section.[1] The general concept may be reused for the simulation of other kinds of firmware.

In the following paragraphs, we describe how Linux was "booted" on CECSIM as a simulated system and the tests performed to demonstrate the capabilities of the CECSIM/Linux approach; the HiperSockets project for z990 is used as a concrete example.

Figure 3 shows the configuration of the CECSIM/Linux system. CECSIM is controlled by the support element (SE), which is used on zSeries servers for system configuration and control (for additional details about the role of the SE in various code-simulation environments, see [1]).

The HiperSockets firmware that is to be tested is transferred from the SE to CECSIM during the system "power-on reset" step. Afterward, the Linux kernel, the root-file system, and a parameter file are transferred to each system partition, and the Linux "boot up" processes are initiated.

The Linux program mounts the root-file system from z990 memory and redirects its console input/output to the operating system message console on the SE. The activation and configuration of the HiperSockets firmware to be tested are done with Linux. **Figure 4** shows an example of a configured HiperSockets setup for zSeries as it appears to the operator: There are *n* logical zSeries partitions, LPAR1 . . . LPAR*n*, each of them running Linux as the operating system with HiperSockets interfaces labeled hsi[*n*]:[*m*] with assigned Internet protocol (IP) addresses. The logical view of the setup is *identical* to the one on a real system if the same configuration procedures are applied in both cases.

Firmware testing can now be done efficiently using standard Linux commands and programming techniques. The following examples demonstrate the ease of performing functional tests under Linux:

- Power-on reset (firmware load) tests.
  - Basic initialization, i.e., channel emulator/control unit emulator.
  - Basic initialization with multiple channel subsystems (MCSS) [7].
  - Shared/dedicated HiperSockets channels.
- Linux initial program load (IPL): Linux booting tests.
  - HiperSockets firmware interaction with Linux booting.
- From the command line: *ifconfig hsi0:0 10.1.1.1* up or *ifconfig hsi0:0* down
  - Tests multipath channel (MPC) handshake.
  - Tests the system of control tables.

[1] zSeries HiperSockets is a technology that provides high-speed transfer control protocol/Internet protocol (TCP/IP) connectivity between virtual servers running within different LPARs of a zSeries server. It obviates the need for any physical cabling or external networking connections between these virtual servers [5, 6].

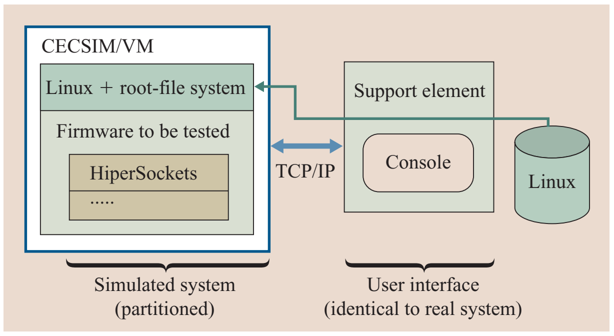

Figure 3

CECSIM/Linux system configuration (VM, root-file system, TCP/IP).

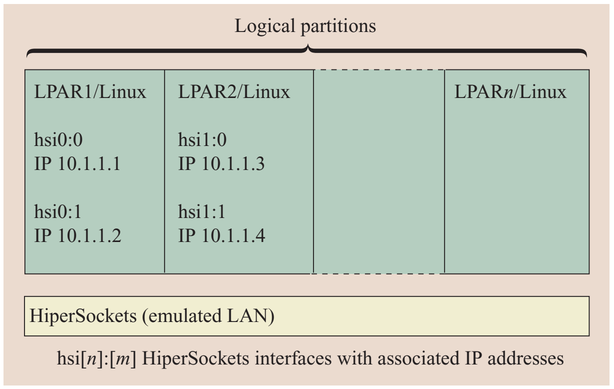

Figure 4

Logical view of a CECSIM/Linux system.

- With a simple shell script program, the "up" and "down" commands such as those in the preceding example can be "chained" to exercise multiple control table changes.
- A simple ping command such as *ping –c 100 –I hsi:0 10.1.1.4* tests data transfer and its control by configuration tables held in the background, including MCSS tests.
- *ping –b –I hsi0 10.1.1.255* tests broadcast data transfer, *ping –I hsi0 224.0.0.1* tests multicast data transfer, and *ping –f –b –I hsi0 –s 4096 10.1.1.255* is a broadcast transfer of large packets (4096 bytes).
- More tests can be created by programming to the standard socket interface.
- Interaction tests: While one logical partition is sending out data packets for transmission, another partition is adding or deleting devices. This tests the locking mechanisms that are implemented to avoid race conditions.
- Dynamic configuration changes: E.g., rebooting Linux while other partitions are still transmitting data.
- Error recovery: Critical entries in control tables are intentionally changed to invalid, invoking system recovery.
  - Testing HiperSockets first error data capture (FEDC) [8]; i.e., gathering the HiperSockets data in case of an error.
  - Testing the full FEDC path, including FEDC data analysis tools.
- Special functional features such as multicast routing and Virtual Local Area Network (VLAN).
- HiperSockets firmware interaction with surrounding firmware:
  - Configure on/off.
  - Re-IPL of Linux.
- Shut down Linux.

Since Linux standard functions and components can be used, these tests are easy to program and do not require a special test environment or test-case syntax; thanks to CECSIM, they can be performed before a real system is available. After testing with Linux on CECSIM, only subtle errors such as critical timing conditions or errors in the interaction with other operating systems (z/VM*, z/OS*) remained to be solved by tests on the real system (estimated to be 20% of the overall problem count), since most of the basic kinds of errors had previously been eliminated. The advantages for faster system qualification and enhanced quality are evident.

OSA-Express simulation

Open Systems Adapter-Express (OSA-Express, [9]) is one type of zSeries I/O channel that provides a network interface to the system. The firmware is executed in an embedded environment running on a RISC-based processor. It utilizes a real-time operating system (OS Open) to support various services such as queuing, threading, and timers. OS Open provides a scalable runtime and development environment for embedded systems running on PowerPC* processors.

Prior to the development of the OSA-Express firmware simulator OSASIM (described below), testing firmware changes to OSA-Express was complex and cumbersome because of the nature of its environment. Testing required access to system hardware for which expense was high and availability was limited. The development of the simulator allowed for ease of testing and removed the hardware dependency. It provided a controlled environment for debugging purposes and verification of code paths. During z990 system development, one of the improvements made to the CECSIM firmware verification tool was attaching OSASIM to CECSIM. The integration of CECSIM and OSASIM allows for full execution of code components in a seamless environment. This was the first time a functional I/O microcode was ever used as part of system functional microcode verification in CECSIM.

The OSASIM is written in C and runs on an AIX* platform with a RISC-based processor. The decision to develop the simulator in this environment was based on portability of the code. Since OSA and OSASIM processors are both RISC-based, source-code changes are not required. The implementation of the simulator utilizes the AIX services as opposed to the OS Open services. In this environment, when using a source-level debugger, debugging becomes easier than on the physical hardware.

The OSASIM can be run in a standalone or integrated mode. Standalone mode allows tests specific to the OSA-Express component. The developers control test conditions via one or more input files utilized by the simulator. This allows stabilization of the component before it is used for testing with other firmware components. Integrated mode attaches OSASIM and CECSIM using a pair of TCP/IP sockets to test interactions between the firmware components.

OSASIM creates a virtual common I/O platform with network adapter [10] using several software models that emulate all hardware components found on the physical card (STI links, STI–PCI[2] bridge ASIC, PCI network adapter, memory). Each modeling component runs its own thread independently.

Various test cases were written to verify the network functions of the OSA-Express code. For basic verification, a test program that runs in CECSIM generates a network frame to the OSASIM and expects it to be sent back. A more versatile method of verifying OSA-Express code uses the NetSim Test Vehicle (see the section on network adapter modeling and the NetSim test vehicle).

[2] PCI: Peripheral component interconnect.

Multifunction PCI–STI bridge chip hardware model

The bridge chip hardware is modeled as one of many threads that are created during the initialization of OSASIM. It supports both the standalone and CECSIM environments. In the OSASIM environment it was necessary to emulate only the STI function of the bridge chip. The actual hardware has three major functional entities: the PCI interface, the memory management unit (MMU) high-speed memory access, and the STI logic that provides high-speed data movement to and from zSeries memory. Since the firmware code does not directly access the MMU and PCI interfaces, there was no need to add these functions to the hardware model.

The STI function emulation has three major areas: the channel communications area (CCA), register operations, and data mover queues (DMQs). The CCA is used as a command-passing interface between OSASIM and CECSIM. The CCA hardware emulation consists of a 64-bit CCA register to pass commands and a 32-bit CCA status register that contains both a busy bit and write status bits. In conjunction with the CCA hardware, there is a 32-bit channel interrupt register that contains a bit (called a *tap channel bit*) indicating that a new CCA has been written. When this bit is enabled, the firmware under test is prompted to read the CCA register and act upon the command.

The DMQ and register operation functions are used to move data and control information between OSASIM and CECSIM, including data sent to and received from the test programs that may be running in CECSIM. The hardware registers for these functions are allocated in AIX memory during the initialization of OSASIM.

After initialization, the model polls for an indication of a new CCA in the channel interrupt register, any register operations to be done, and any outstanding DMQs that have to be processed. Depending on the function to be performed, the bridge chip model either sends or receives an STI packet from the TCP/IP interface when running with CECSIM, or from a test-case input file when running standalone. The STI packet is decoded, the appropriate memory-mapped hardware registers (e.g., CCA) are updated, data is moved to or from the specified address space of the firmware, and responses are sent back to either CECSIM or the standalone test case.

Both the register operation and DMQ emulation code are driven by firmware. With the hardware registers being mapped to an AIX address space, the firmware under test performs the same function as if it was running on the actual hardware. The emulation code polls the memory-mapped hardware register that enables these functions, translates the software structure passed in by the firmware, breaks up the request into the STI packet format, and sets the appropriate status fields pertinent to the operation. In the case of a DMQ operation, a back-end function in the firmware processes the data being fetched, or sends up ending status in the case of a read command on data that is stored to and by the test case. The DMQ emulation code has support for two DMQ engines. In our OSA firmware code, DMQ engine 0 typically is used to fetch data, and DMQ engine 1 is used to store data into zSeries memory.

STI link emulation

OSASIM and CECSIM are connected via TCP/IP. The two simulation environments run on different operating systems (OSASIM on AIX, CECSIM on VM/CMS) and could not be integrated into a single application. The use of a network-based protocol was especially useful during bringup of this environment.

The protocol used on the TCP/IP link is basically a simplified version of the protocol used on the actual STI links; this way, the protocol was well-documented and known to both development teams right from the start. On the TCP/IP link, a five-word header was added to each STI message. This header contains the following material:

- A key to detect transmission errors.
- The message size (number of bytes).
- Information about the type and source of STI information contained in the message.
- The address of the OSA-Express card to which the message is directed.

The OSASIM–CECSIM connection uses two sockets. One is used for processor-initiated STI traffic (usually read and write operations on registers of the bridge chip). Such transfers are triggered by synchronous calls from CECSIM. The other socket is used for I/O-initiated STI traffic (read and write main storage, present interrupts, read timer information from the CEC). A separate thread in CECSIM "listens" on this socket to handle these asynchronous I/O requests.

To run the whole simulation, one or more OSASIM instances are started first. After initialization they listen for connections on agreed-upon TCP/IP ports. Then CECSIM is started and connects to the OSASIM servers as specified in a configuration file. The first message sent to OSASIM contains information which defines the IP address and port number of the second socket to open (for I/O requests). This connection is then initiated by OSASIM.

This scheme allows the use of two sockets with a requirement for only a single port number to agree upon on the AIX system and no fixed port number at all on the VM system. This latter point was important, since all users of the VM system share the same range of

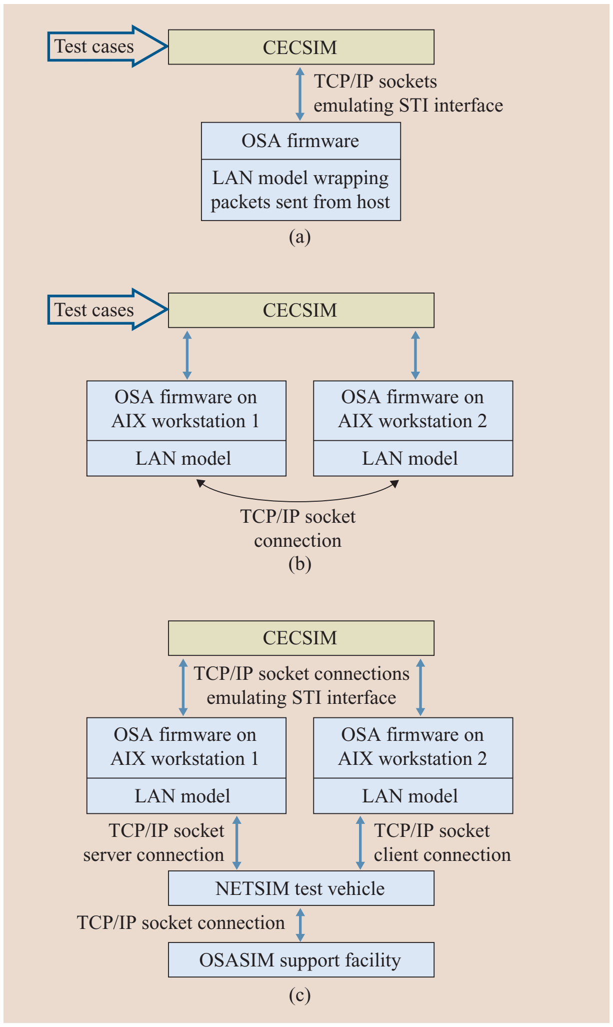
Test cases
CECSIM
TCP/IP sockets emulating STI interface
OSA firmware
LAN model wrapping packets sent from host
(a)
Test cases
CECSIM
OSA firmware on AIX workstation 1
LAN model
OSA firmware on AIX workstation 2
LAN model
TCP/IP socket connection
(b)
CECSIM
TCP/IP socket connections emulating STI interface
OSA firmware on AIX workstation 1
LAN model
OSA firmware on AIX workstation 2
LAN model
TCP/IP socket server connection
TCP/IP socket client connection
NETSIM test vehicle
TCP/IP socket connection
OSASIM support facility
(c)

Figure 5

(a) Basic OSASIM configuration. (b) Enhanced OSASIM with CECSIM. (c) Enhanced OSASIM with NetSim vehicle.

port numbers, and this scheme avoided the administrative effort of assigning port number ranges to each user.

For the CECSIM connection, the OSASIM bridge chip model was extended by some TCP/IP interface code, STI packet format support, simulation of some bridge chip facilities used by CEC firmware only ("maintenance registers"), and code to replace the boot loader used in a real system to transfer the OSA-Express firmware into the memory of the OSA-Express card. This boot loader is missing in OSASIM, of course. However, in order to satisfy CEC firmware during the initial microcode load process, OSASIM has to mimic its behavior before the real OSA-Express firmware takes over communication with CEC firmware.

Network adapter modeling and the NetSim test vehicle

In the basic OSASIM environment [**Figure 5(a)**], a test program written in high-level assembly language drives I/O programs through CECSIM to the OSASIM. Various types of I/O programs with arbitrary numbers of bytes are transferred to OSASIM. This is one step forward compared with the standalone simulator. However, the local area network (LAN) driver cannot be fully exercised because there is no outbound connection with which the LAN driver can communicate. It takes the outbound packet and routes it back as an inbound packet.

In order to verify the LAN driver as well as the core OSA firmware, an additional copy of OSASIM was introduced into this simulation environment [**Figure 5(b)**]. While one copy of the OSASIM sends out packets to the simulated network environment through a TCP/IP socket connection, the other end of the socket connection is the other copy of OSASIM waiting to receive the packet from the network. Once packets are received, data is sent up to the driver for data comparison and verification. With this setup, the main line paths are verified from i390 code down to the OSA firmware LAN driver.

To simulate the networking environment (packets coming from neighbor networks, the link between networks going up and down, etc.), the network simulator (NETSIM) was created along with the simulator support facility (SIMSF) and added to the OSASIM/CECSIM environment [**Figure 5(c)**].

A NETSIM model is connected to each OSASIM model via a TCP/IP socket connection. The SIMSF model is also attached to the NETSIM model via another TCP/IP connection. SIMSF collects statistics from the NETSIM as well as making adjustments to the settings of the simulator. The NETSIM model can generate many test variations, including IP frames with and without acknowledgment, multicast frames, and broadcast frames in various frame sizes. In addition, address-resolution protocol caching, IP Version 6, and link up/down conditions are simulated by the NETSIM vehicle.

NETSIM generates datagrams and sends them to the test program through OSASIM copy number 1. The test program passes the data received to OSASIM copy number 2 and further on to NETSIM. While NETSIM executes test cases and sends frames up to OSASIM, SIMSF comes in and alters the configuration, causing NETSIM to behave differently. This simulates a continuously changing network environment.

In summary, the attachment of OSASIM to CECSIM has proven to be effective for I/O firmware development. The simulation approach enables the development process to proceed at a more rapid pace aided by the ease of setting up test scenarios. As a result, initial bringup time is reduced, and code delivered to system test is more stable. During the z990 project, a similar simulation environment for PCI crypto cards had already been realized on the basis of the OSASIM code. The implementation of the CECSIM infrastructure is independent of OSA-specific items; therefore, it may also be used for the connection of standalone test facilities of other channels.

Simulation of control code using office hardware

The coverage of the firmware simulation has been increased significantly by inventing verification mechanisms for the system control firmware, which have been included in the z990 firmware verification process for the first time. System control code is referred to as in-band system management whenever it is implemented within the scope of an operating system. The innovations of in-band control code verification are described in the previous sections of this paper. This section focuses on the verification improvements of the out-of-band firmware, i.e., control code that uses functions beyond the scope of the operating system.

System environment

System control firmware is deeply embedded in the system structure. This makes it difficult to design firmware test cases running in the system environment that make results visible but do not affect the system operation. This also applies to the out-of-band (OOB) firmware. The processor executing this OOB code is a controller card called the flexible support processor (FSP); it is located in each CEC, I/O, and power cage, as indicated in [11].

This property of the system firmware makes the testing and debugging extremely difficult and time-consuming if done in a real system environment, since the system often requires special test equipment to monitor the firmware output. For the predecessor system z900, this was the only way to verify FSP code. However, for the z990 system a new concept has been invented which integrates an FSP firmware verification into the system simulation to achieve 90% simulation coverage. This section describes the simulation approach for the system configuration (cage configuration object model, or CCOM), which includes I/O and power configuration. The new verification environment for the CEC initialization and communication code is described in the section on cage controller simulation.

One main focus was to be as consistent as possible with the overall z990 system simulation efforts. This is essential in testing the correctness of the interfaces and protocols between the different firmware parts. However, timing issues must be tested under the same conditions as those existing in the real system. For the z990 system, an interbalanced approach between a software simulation and a hardware simulation was chosen, and a special hardware was built to simulate the timing conditions exactly as they

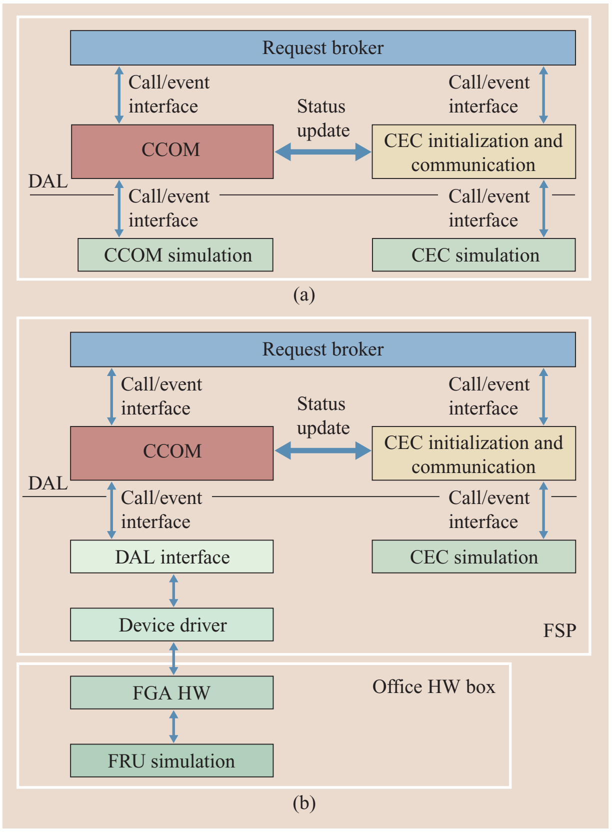

Figure 6

(a) Software simulation setup. (b) Simulation setup with office hardware.

would be seen in the real system. This environment, called office hardware or field-replaceable unit (FRU) simulator, was used for additional out-of-band firmware testing. This allowed us to participate in the overall system simulation while achieving extensive coverage of time-critical functions.

In the system control firmware, hardware access is via a device abstraction layer (DAL), as described in [11]. This defined interface is used to access real hardware via device drivers, and also in the firmware simulation environments to access simulation code.

Software simulation

Like the CEC simulation (described in the section on cage controller simulation), the configuration code employs a well-defined DAL interface. This interface allows easy replacement of the device driver code, which is needed to access real hardware with a simulation model [**Figure 6(a)**]. Since the CEC simulation and the configuration code use different devices and thus different DALs, the two simulation environments are independent and may

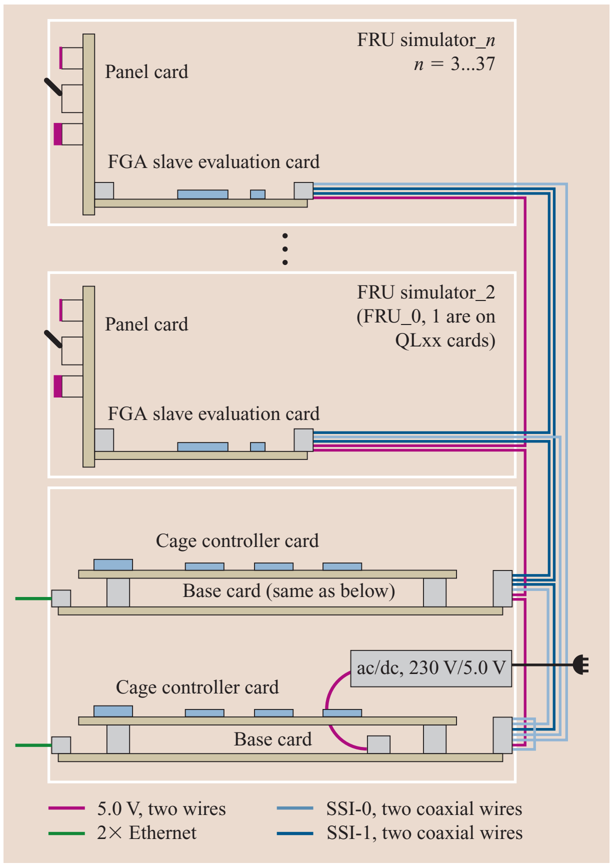

Figure 7

Office hardware overview.

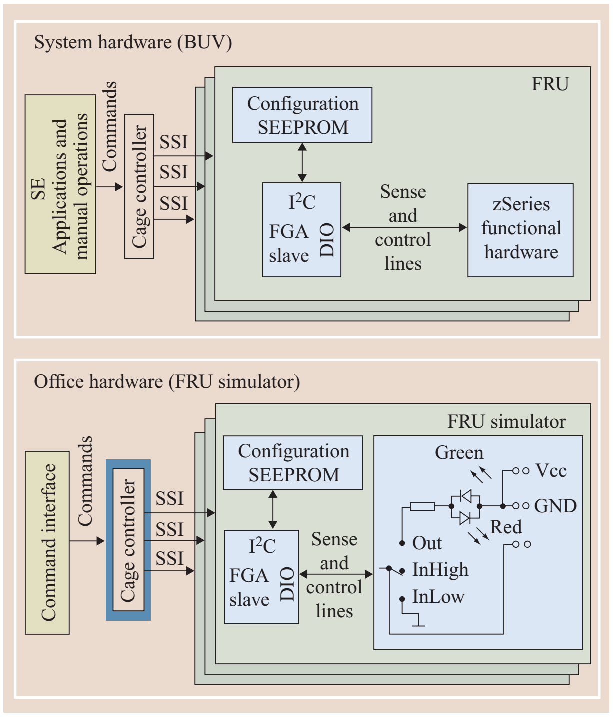

Figure 8

Comparison of real system and office hardware.

(though they need not) run together. Furthermore, for each device type the DAL code (simulation vs. real hardware access) may be selected independently or mixed. The first extreme option is an independent code simulation model which uses shortcuts for hardware accesses whenever commands are sent to the DAL. The other extreme configuration is a comprehensive model in which the entire simulation environment executes hardware accesses in FRU simulators. This FRU simulator [**Figure 6(b)**] is equipped with a specially designed piece of hardware which, as an assembly, is called office hardware and is described in more detail in the next section.

Office hardware

To be able to verify the cage controller code on a PowerPC platform which uses the devices, the device drivers, and the operating system OS Open as the real system does, an office hardware environment was defined. We used a system that comprises one or more FRU simulator boxes and a box containing one or more FSP cards [Figure 6(b)]. The plugged FSP cards are responsible for power, CEC, and I/O control. The following system configurations are possible:

- One power cage (with FSP master/slave), one CEC cage (master only), and one I/O (master only) cage.
- One power cage (with FSP master/slave) and two CEC nodes (master only) without an I/O cage.
- One power cage (with FSP master/slave) and one CEC or I/O cage (master and slave).

The primary components of an FRU simulator are the FRU gate array (FGA) slave, providing the card access, and a SEEPROM, as on a real system. The SEEPROM content (also referred to as vital product data, or VPD) defines which type of card (e.g., OSC, ETR, STI, ESCON, and ISC-3) should be generated by the cage controller object model. This allows the same FRU simulator hardware to be used for different card types by modifying the SEEPROM data. The FGA slave which is connected to the FGA master on the FSP card provides several standard interfaces such as UART, JTAG, I^{2}C, general-purpose input/output (GPIO), and the proprietary service bus adapter interface. All of these interfaces are necessary for the out-of-band control of a real system.

Figure 7 is an overview of the office hardware environment, in which chip control signals are verified under real-time conditions, voltage violations can be detected, interrupt mechanisms are verified, and cable traffic is sensed. **Figure 8** shows a generic view of the differences between a real system and the office hardware.

Cage controller simulation

In another major improvement in the z990 firmware verification, the cage controller code was simulated for the first time. Approximately 75% of this firmware was verified with the simulation approach. The cage controllers, which are embedded in the CEC cage, initialize and maintain system operations. Each cage in a zSeries system contains one pair of cage controllers, of which one is defined as a master, the other as a slave. The slave works as a backup that is ready to take over all tasks when the master fails (for details, see [11]). The cage controller has interfaces to other firmware components [**Figure 9(a)** shows a very generic view of the z990 cage controller architecture]:

- The cage controller uses a system support interface (SSI) to communicate with the CEC.
- The TCP/IP connection with the support element is used for the reporting and controlling of system operations.

To be able to verify the cage controller code prior to the availability of the hardware, it was decided to build connections to the existing simulation environments ET4 [2] and CECSIM [**Figure 9(b)**]. To use the cage controller code in the existing simulation environments, some adaptations have been necessary. Both CECSIM and ET4 have TCP/IP interfaces but do not offer SSI support as used for the communication between the cage controller code and the CEC on a real system. Therefore, a special simulation SSI driver has been implemented which translates SSI commands. This driver provides the same application interface as the real SSI device driver (e.g., initializations, command execution, interrupt handling).

On a real z990 system, the cage controller code runs on a PowerPC controller with the OS Open operating system. Since it is very difficult and complex to establish a communication between OS Open and CECSIM or ET4, it was decided to port the cage controller code from OS Open to an Intel**-based Linux. In addition, some hardware accesses were replaced with simulation-specific code. With this approach, each developer is able to verify cage controller code on his Intel-based workstation by connecting to the simulation environment.

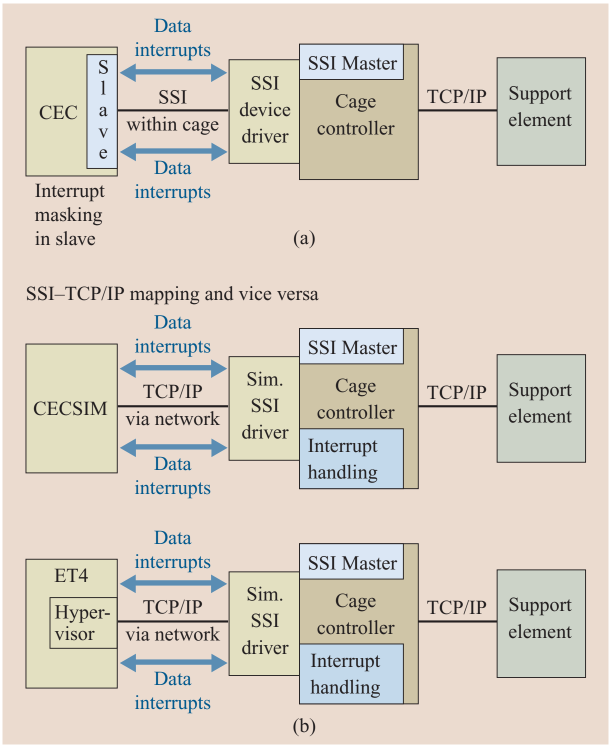

Figure 9

(a) Basic cage controller configuration in a real z990 system. (b) Cage controller simulation configurations with CECSIM and ET4.

On the real hardware, the services on the cage controller run on fixed ports; each cage controller is identified via a unique IP address. This concept of fixed ports is not applicable for the simulation environment, since in the simulation environment there is only one IP address associated with the ThinkPad* on which the cage controller code is executed. In order to handle more than one instance of the cage controller code, the port number is used as an identifier for the cage controller. This is needed to simulate systems with more than one node, since each node has its own cage controller, and to allow different users to access the same ThinkPad. Port numbers are therefore defined for each user and simulated node.

For the cage controller firmware simulation, minor adaptations on the support element and in the simulation environment were required. New configuration files were needed to make the support element work together with the cage controller simulation. Hypervisor (ET4 verification) and CECSIM must translate the commands that come in via TCP/IP to a format that the simulator is able to understand.

The benefits of the new cage controller firmware simulation are multilateral:

- The GCC (GNU compiler collection) compiler which is used in the simulation environment has a significantly lower fault tolerance and a better warning concept than the standard AIX* compiler, XLC, which was used for OS Open. The GCC cannot be used for code generation in a real system because it does not support OS Open.
- In the simulation environment there is sufficient hard disk space to save debugging data on the attached support element. On the controller there are only very limited resources to save this data during runtime.
- The use of a source code debugger adds extra comfort for the developers for debugging their code.
- Bringup time on real systems is very expensive and limited. With the combination of the office SE, which is a standalone support element used for code verification, and the CECSIM environment, every developer is able to create his own simulation environment, because neither of these environments relies on special hardware.

Concluding remarks

With the new simulation concepts, the duration of the first system integration phase was reduced significantly. For the z900 system it took eight weeks from the power-on of the system until the first test operating system (system assurance kernel, or SAK) was running. In conjunction with a powerful emulation engine [12] for the z990 system, this milestone was reached after three weeks, although the estimated duration was twelve weeks because of its significantly higher complexity compared with the preceding z900 system. In addition, most of the concepts described in this paper have been implemented in such a way that they can be reused, applied, or extended for the verification of other firmware parts of future systems.

*Trademark or registered trademark of International Business Machines Corporation.

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

References

1. 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).
2. 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).
3. 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).
4. IBM Corporation, *IBM 370-XA Interpretive Execution* (SA22-7095); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
5. B. White, J. Nesbit, and I. Neville, "IBM eServer zSeries Connectivity Handbook," IBM Redbooks (SG24-5444); see *http://www.redbooks.ibm.com/pdfs/sg245444.pdf*, pp. 133-141 (2003).
6. B. White, R. Ayvar, and V. Uskokovic, "zSeries HiperSockets," IBM Redbooks (SG24-6816); see *http://www.redbooks.ibm.com/redbooks/pdfs/sg246816.pdf* (2002).
7. L. W. Wyman, H. M. Yudenfriend, J. S. Trotter, and K. J. Oakes, "Multiple Logical Channel Subsystems: Increasing zSeries I/O Scalability and Connectivity," *IBM J. Res. & Dev.* **48,** No. 3/4, 489–505 (May/July 2004, this issue).
8. S. Koerner, R. Bawidamann, W. Fischer, U. Helmich, D. Klodt, B. K. Tolan, and P. Wojciak, "The z990 First Error Data Capture Concept," *IBM J. Res. & Dev.* **48,** No. 3/4, 557–567 (May/July 2004, this issue).
9. B. White, V. Braga, and J. van Dellen, "OSA-Express Implementation Guide," IBM Redbooks (SG24-5948-02); see *http://www.redbooks.ibm.com/redbooks/pdfs/sg245948.pdf* (2003).
10. D. J. Stigliani, T. E. Bubb, D. F. Casper, J. H. Chin, S. G. Glassen, J. M. Hoke, V. A. Minassian, J. H. Quick, and C. H. Whitehead, "IBM eServer z900 I/O Subsystem," *IBM J. Res. & Dev.* **46,** No. 4/5, 421–445 (July/September 2002).
11. 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).
12. K.-D. Schubert, E. C. McCain, H. Pape, K. Rebmann, P. M. West, and R. Winkelmann, "Accelerating System Integration by Enhancing Hardware, Firmware, and Co-Simulation," *IBM J. Res. & Dev.* **48,** No. 3/4, 569–581 (May/July 2004, this issue).

Received September 15, 2003; accepted for publication April 20, 2004; Internet publication May 27, 2004

Michael Stetter *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (stetter@de.ibm.com).* Mr. Stetter graduated from the University of Ulm, Germany, in 1997 with a diploma degree in mathematics and economics. In January 1998 he joined the IBM development laboratories in Boeblingen, Germany. He worked on a variety of firmware simulation assignments to design and implement CECSIM extensions and simulation tools. Mr. Stetter was CECSIM team leader and firmware simulation representative for the z990 program. After working as firmware Project Manager for the next zSeries system generation, Mr. Stetter currently manages the Support Element Infrastructure Department.

Joachim von Buttlar *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (joachim_von_buttlar@de.ibm.com).* Mr. von Buttlar is an Advisory Engineer in the zSeries I/O Microcode Development group. He received an M.S. degree in computer science (Dipl.-Inform.) from the Technical University of Berlin in 1983. In 1984, he joined the IBM development laboratories in Boeblingen, Germany, to work on microcode development for the IBM 3092, 9221, and 9672 systems. From 1990 to 1991 he worked as Liaison Engineer on international assignment in Endicott, New York. In 1997 he initiated the CECSIM project, developed its concepts, and implemented the simulator kernel. He has been the technical leader of this project during the development of the z900 and z990 systems. Mr. von Buttlar received an IBM Corporate Award in 2002.

Ping T. (Danny) Chan *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (ping@us.ibm.com).* Mr. Chan is an Advisory Engineer working in the OSA Development group. He graduated with a B.E.E.E. degree from City College of New York in 1984 and joined IBM at Poughkeepsie, New York, that same year. He has held various technical positions in the eServer diagnostics and I/O areas. Mr. Chan has received several IBM Outstanding Technical Achievement Awards for his contributions in OSA development.

Dietmar Decker *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (ddecker@de.ibm.com).* Mr. Decker received a diploma in electrical engineering from the Universitaet Karlsruhe (TH). In 1994 he joined the IBM development laboratories in Boeblingen. He currently works on HiperSockets microcode and InfiniBand Linux device driver development.

Herwig Elfering *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (hge@de.ibm.com).* Mr. Elfering is an Advisory Engineer currently working as a team leader on the zSeries service processor application code for cage configuration and control. He received an M.S. degree (Dipl.-Ing.) in electrical engineering from the faculty for Digital Information Processing of the University of Paderborn, Germany, in 1994. He joined the IBM development laboratories in Boeblingen that same year in the Power Control Department, where he worked in different areas of S/390 power and system control applications for the G2 to G6 systems. Since 1997 Mr. Elfering has worked on the design and implementation of the new zSeries power and system control network, focusing on configuration, high reliability, availability, and serviceability of the IBM eServer.

Paul M. Gioquindo *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (gioquind@us.ibm.com).* Mr. Gioquindo is an Advisory Engineer in the Networking Communications Development Department. He received his A.S. degree in electronics from the PennCo Technical School, subsequently joining IBM at Poughkeepsie, New York. He has held various technical positions in the eServer manufacturing, assurance, engineering test, and networking areas. Mr. Gioquindo has received several IBM Outstanding Technical Achievement Awards for his contributions in S/390 Parallel Sysplex bringup, OSA hardware bringup, and OSA development. He is a coauthor of three patents.

Thomas Hess *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (hessth@de.ibm.com).* Mr. Hess is a Software Development Engineer currently working on zSeries system sensing and control. He received a certified engineer degree from the Technical School for Data Electronics in Pforzheim, Germany, in 1990. That same year Mr. Hess joined IBM in Boeblingen, Germany, where he was responsible for the manufacturing logistic system. In 1993 Mr. Hess worked on hardware and software development for several customer projects. In 1998 he began designing and developing software for the zSeries service processor. He is currently working on a new object-oriented design which allows rule-based access to the system control hardware.

Stefan Koerner *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (koerners@de.ibm.com).* Mr. Koerner is a Senior Technical Staff Member in the IBM eServer z990 Hardware Development Group in the Boeblingen laboratories. He joined IBM in Boeblingen in 1981 after receiving an M.S. degree in electrical engineering from the Technical University of Furtwangen. He has held a number of positions in logic design, firmware development, and verification. He holds three patents and received an IBM Outstanding Innovation Award in 2001. Mr. Koerner is currently the technical leader for firmware verification in the IBM Systems and Technology Group.

Andreas Kohler *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (akohler@de.ibm.com).* Dr. Kohler received M.S. and Ph.D. degrees in physics from the University of Stuttgart, Germany, in 1993 and 1999, respectively. In 1999 he joined the IBM development laboratories in Boeblingen, Germany. His current responsibilities in zSeries I/O microcode development include test tools, simulation, and error-recovery code.

Heinrich Lindner *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (hlindner@de.ibm.com).* Mr. Lindner received Dipl.-Ing. and Dr.-Ing. degrees in physics from the Technical University Berlin. In 1967 he joined IBM and was involved in semiconductor technology development (primarily memory devices) until 1982. He subsequently worked in processor development on five generations of S/370, S/390, and zSeries processors, starting with the first single-chip microprocessor in CMOS technology; other fields of work were memory development, new I/O concepts, and logic simulation. Since 1998, he has been an engineer in zSeries firmware development.

Karlo Petri *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (xxpetri@de.ibm.com).* Mr. Petri is a Staff Engineer currently working on zSeries flexible support processor applications. He studied computer science at the Berufsakademie Mannheim and graduated in 1996, receiving a Diplom-Ingenieur (B.A.) degree. He joined the IBM development laboratories in Boeblingen that same year, working on the support element for three years. Since 1999, he has worked on the cage controller for zSeries and contributed to making the cage controller software part of the system simulation. Mr. Petri is currently working on the transition of the cage controller software from OS Open to Linux.

Mooheng Zee *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (mzee@us.ibm.com).* Mr. Zee received a B.S. degree in electrical engineering from Polytechnic University in 1988, joining IBM that same year in Poughkeepsie, New York. He started in the ESCON channel diagnostic group, then moved to the networking I/O team to develop the ATM networking products for the eServer. During ten years in networking development, Mr. Zee has received several IBM Outstanding Technical Achievement Awards and filed two patents. He is currently involved in the development and continued enhancement of the networking channel functions and OSA code simulation.