z/CECSIM: An efficient and comprehensive microcode simulator for the IBM eServer z900

by J. von Buttlar
H. Böhm
R. Ernst
A. Horsch
A. Kohler
H. Schein
M. Stetter
K. Theurich

An IBM eServer zSeries™ system uses various types of microcode (firmware) that implement functions such as the execution of complex instructions in the CPUs, I/O operations performed by the system assist processors (SAPs), the management of logical partitions (LPARs), and control by the support element (SE). Each microcode component must be verified by itself and in conjunction with the others. Tight development schedules and a very limited supply of expensive engineering hardware make it desirable to perform this verification in a simulation environment. For the development of the z900, a new microcode simulator, the z/CECSIM (Central Electronic Complex Simulator), was successfully implemented. Several microcode components are connected in a single simulation environment, thereby allowing an unprecedented amount of development, integration, and testing without the use of engineering hardware. z/CECSIM creates a virtual zSeries CEC on VM/ESA® or z/VM™ that allows the simulation of zSeries microcode. It executes the instruction stream as completely as possible on the underlying hardware. Only instructions that are newly introduced with the system being developed or that perform a microcode-internal function are simulated. Additional software models mimic the behavior of I/O and coupling channels. An optional SE connection allows verification of interactions between the CEC and its support element.

Introduction

In the IBM S/390* and zSeries* CECs there are two levels of microcode. The lowest level is designated as millicode; it implements functions that either are performance-critical or require direct control of the hardware [1, 2]. Among other tasks, millicode implements complex instructions and interrupt handling, and participates in system functions

0018-8646/02/$5.00 © 2002 IBM

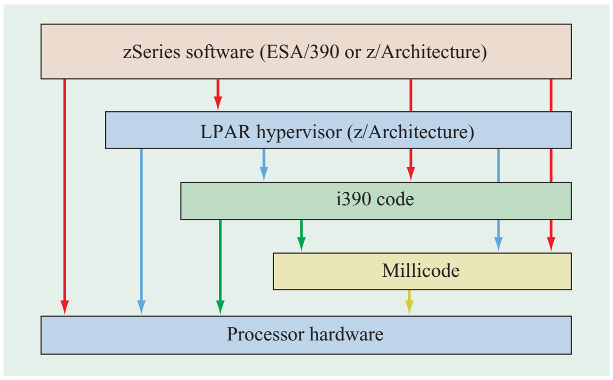
zSeries software (ESA/390 or z/Architecture)
LPAR hypervisor (z/Architecture)
i390 code
Millicode
Processor hardware

Figure 1

Structure of z900 code layers.

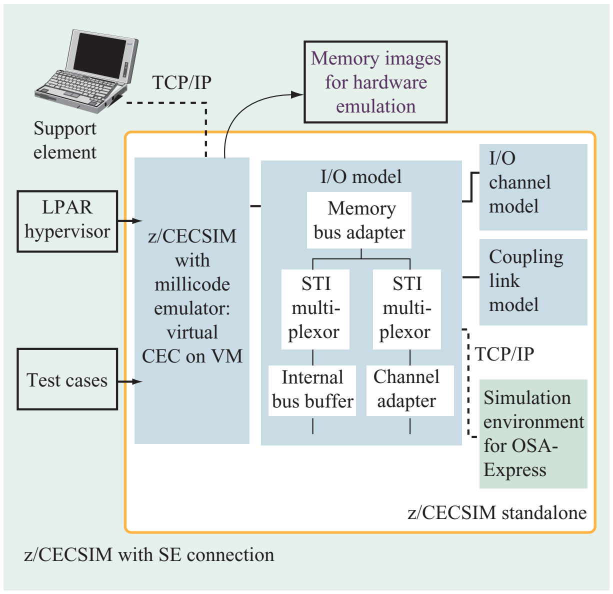

Figure 2

Structure of z/CECSIM.

such as reset and recovery. The millicode instruction set consists of a subset of those z900 instructions that are implemented directly in hardware, plus some special instructions that are valid only in millicode mode. Millicode is written in High-Level Assembler language.

The other level of microcode is designated as i390 (internal 390) [3]. It implements functions such as the channel subsystem or power-on reset. The i390 instruction set consists of all instructions implemented in hardware and by millicode; i390 code is written in PL.8 (a dialect of PL/I), C, and High-Level Assembler language.

The z900 CEC can run in two modes, basic and LPAR. In basic mode, z/Architecture* software runs on top of the microcode layers. In LPAR mode, a hypervisor program manages logical partitions, each of which runs z/Architecture software. From the millicode/i390 perspective, the hypervisor looks like a z/Architecture program. **Figure 1** provides an overview of the code layers.

During z900 development, z/CECSIM was used to verify i390 code and millicode in conjunction with test programs at the z/Architecture level. The environment can also be configured to run the LPAR hypervisor program and to connect to a support element (a regular laptop computer) via TCP/IP, thereby verifying these code components as well. Interactive debug and trace facilities allowed very efficient code development and test. **Figure 2** shows the components of z/CECSIM.

Execution environment

Since i390 architecture consists of zSeries instructions, VM/ESA* and z/VM* are ideal environments in which to execute such code. This provides a tremendous performance advantage over hardware simulation models that interpret each instruction cycle by cycle [4, 5]. However, running i390 code on plain VM/CP is not possible, since i390 uses many additional instructions that are not provided by underlying hardware or VM/CP. Moreover, some rules for storage access and interrupts differ from those described in the *z/Architecture Principles of Operation* [6]. z/CECSIM creates a virtual zSeries CEC with all properties visible not only at the software level, but also at the i390 and millicode levels. It is implemented as a multitasking CMS application with each simulated processor running in a separate thread [7]. To execute the instruction stream at the software and i390 levels, z/CECSIM uses the *start interpretive execution* (SIE) instruction [8]. This instruction is normally used by VM/CP and LPAR to run a virtual machine and a logical partition, respectively. It gives z/CECSIM full control over the simulated code, while, on the other hand, the simulated code has no access to resources used by the simulator. During z900 development, z/CECSIM was installed and executed on 9672 G3–G6 systems.

Instruction execution

The execution of the instruction stream by SIE ends when an instruction or condition is encountered that requires intervention by z/CECSIM. This is usually an instruction unknown to the underlying hardware, or it may be an instruction that enables a pending interrupt. I/O instructions always require interpretation with respect to the simulated I/O configuration. The instruction that is to be simulated is implemented in either z900 hardware or millicode. In either case, z/CECSIM invokes a millicode emulator (MCE) to perform the requested function.

Table 1 Instruction execution.

z/Architecture program		*Millicode program*	*Mode of execution*
. . .			
LG	R0, COUNTER		Instructions
AGHI	R0, 1		interpreted
STG	R0, COUNTER		by MCE
LA	R2, ORB		Executed
LH	R1, SCHNUM		on underlying
ICM	R1, B'1100', = X'0001'		hardware
SSCH	0(R2)	RSR MR12, MWORK@	Millicode of SSCH
		. . .	interpreted by MCE
		MCEND	
BZ	IOSTARTED		Executed on
BO	OFFLINE		underlying hardware
. . .			

If the instruction is implemented in z900 hardware, it is first interpreted by MCE. Then, z/CECSIM resumes simulation of the processor using SIE right after the interpreted instruction. An example of this is the LG instruction (*load 64-bit*), which is not available on the 9672 on which z/CECSIM was run. However, z/CECSIM maintains all 16 64-bit general registers such that a succeeding STG instruction (*store 64-bit*) would deliver the value previously loaded by LG. A 32-bit store instruction (ST) would store the low-order part of the general register without intervention by the simulator. Once z/CECSIM itself runs on a z900, LG and STG are executed directly on the underlying hardware, together with other instructions that were introduced with the z900.

If the instruction to be simulated is implemented in z900 millicode (e.g., *start subchannel*, *signal processor*), MCE switches the state of the simulated processor to millicode mode. It sets up the millicode instruction address on the basis of the operation code. From then on, millicode is interpreted until an MCEND (*millicode end*) instruction is encountered. The processor state is switched back to i390 or software mode, and z/CECSIM resumes execution of the instruction stream using SIE. (See **Table 1**.)

The z900 storage model in z/CECSIM

The storage of the z900 CEC is allocated from CMS free storage. The portion of storage that contains i390 code and millicode is known as the hardware system area (HSA); the other portion is available to the z/Architecture software. While a processor runs in i390 mode, it may not access software storage directly, but only through special instructions implemented in millicode. On the other hand, software must not be able to access the HSA. In the case of an access violation, an addressing exception is presented to either the software or the i390 code.

The SIE instruction requires dynamic address translation (DAT) tables to describe CEC storage. A segment table is used to implement the access protection mechanism described above. For each processor thread, a separate segment table is constructed. The segment tables of all processor threads point to the same page table. Since each processor maintains its own mode (software mode, i390 mode), only the storage segments that are accessible in the current mode are marked valid. Whenever a processor changes its mode, the entries in the segment table are updated to reflect the correct valid/invalid state.

Millicode emulation

In addition to the 16 general registers defined in the *z/Architecture Principles of Operation* [6], there is a set of 16 millicode general registers (MGRs). While the processor runs in millicode mode, instructions operate on the MGRs instead of the software general registers. There is also an extra condition code for millicode mode.

The millicode architecture defines special rules for storage accesses in millicode mode. Depending on the base register, an address is treated either as a program logical address or as a hardware system area address. Other address types such as a program real address and special access modes such as "test for store exceptions without actually storing" can be specified using operand access control registers (OACRs) that are associated with four of the available base registers.

There are about 70 millicode-only instructions that facilitate bit string manipulation, access to the software general registers, access to internal processor registers, and communication with other components of the system. These are executed by hardware and are not available to zSeries software. Thirty percent of the instructions in the current millicode load are such special millicode instructions.

The special handling of storage operands and the high number of millicode-only instructions make it undesirable to execute millicode in a virtual machine environment; therefore, the millicode emulator was developed as an independent component. It interprets all machine instructions instead of executing them. This also has the advantage that MCE can be used as a standalone tool for verification of millicode. In this mode it is not tied to the VM environment, but can be used on various kinds of workstations. MCE was developed with both standalone and z/CECSIM usage in mind.

MCE emulates the behavior of one or more z900 processors as seen by software, i390 programs, and millicode. The software and i390 view of the state of a processor is the set of architected registers as described in the *z/Architecture Principles of Operation* [6]. The millicode view of the processor state is primarily a set of 256 64-bit registers, designated as the R-unit. The architected z900 registers are a subset of the R-unit. Since MCE is designed for the verification of microcode, it consists internally of components such as instruction fetch, decode, execution, address translation, and storage access. MCE does not implement a cache and has no translation lookaside buffer (TLB). For performance reasons, however, instruction buffers are implemented. The external interfaces of MCE consist of access to the R-unit data within MCE, functions for storage access that must be provided outside MCE, and a function to step a processor instruction by instruction.

In the z/CECSIM environment, the z/CECSIM kernel controls MCE. When an instruction or an interrupt is encountered that is implemented in millicode, the R-unit is initialized with the current state of the processor, including the instruction address. Then MCE is called to execute instruction steps. The first step performs millicode entry for the instruction or interrupt to be executed. Stepping the processor ends as soon as it is no longer in millicode mode. Storage access requests are handled by z/CECSIM. Since each processor is implemented as a separate thread, there are also multiple instances of MCE. Communication between the processors and signaling of asynchronous events are accomplished by queues that are similar to POSIX** message queues.

Standalone MCE has a storage model and an attached monitor for architecture verification programs (AVPs). AVP is a test-case format that has been used for hardware verification in simulation and on real hardware [9]. The AVP specifies the initial state and the expected final state of processor and storage. By convention, a program status word (PSW) of all zeros ends an AVP. This is normally achieved by a supervisor call (SVC) interrupt or a program interrupt. Starting from the initial state, MCE executes instructions until the ending condition is met. Then MCE compares the actual state of processor and storage with the expected state. There exist libraries of AVPs that are carried over from previous projects. In addition, there is an AVPGEN program that generates AVPs for many architected functions. The syntax for AVPs was extended for MCE to allow testing of multiple processors and the initialization of hardware registers outside the processor, e.g., in the memory bus adapter (MBA) or clock chip. An error-inject feature was also added. It allows the alteration of registers or storage when specified conditions are met during the run of a test case. Standalone MCE runs as a single thread. The processors are stepped in a round-robin fashion. After one instruction is executed on each processor, interprocessor communication and asynchronous events are handled.

Standalone MCE is used early in the development phase of a new processor. A significant part of the millicode is needed for hardware verification in simulation. Most of this millicode is pre-tested in unit simulation on the standalone MCE. The performance of MCE is about 10 000 times better than the performance of a hardware simulator. The result of a typical AVP test case is available almost instantaneously. Moreover, the traces generated by MCE are more readable for a code developer than traces from a hardware simulator.

The focus of the tests for which MCE is used within z/CECSIM is different. From a millicode point of view, the interfaces between millicode and other code—i390, software, and LPAR hypervisor—are the test target. In this environment, only instructions that can be intercepted in the VM environment are passed to MCE. These are primarily the special i390 instructions that serve as an interface between i390 code and millicode, and complex z/Architecture instructions such as I/O instructions. All of the general ESA/390 instructions are directly executed on the hardware of the VM system, and the millicode for these instructions is not used (e.g., *translate and test*, *move long extended*).

Connection to the support element (SE)

The support element of the z900 is a laptop computer running an OS/2* application that controls and monitors the CEC [10]. Two modes of operation are available with z/CECSIM.

The standalone mode simulates millicode, i390, and zSeries programs. SE functions are performed by z/CECSIM itself. For example, microcode loads that are normally sent by the SE are read from CMS files. To simulate a power-on reset, z/CECSIM generates all required data packets (service words) and passes them to i390 code.

In the other mode, z/CECSIM is connected to an SE via TCP/IP. Instead of simulating and interpreting service words itself, z/CECSIM acts as a transport layer for communication between the SE and processors. Microcode loads are read from the SE hard disk, as on a real system, and sent to the simulated CEC. Power-on reset simulation includes loading and initialization of the LPAR hypervisor, thereby testing all of its interactions with the SE, i390 code, and millicode.

Most SE functions that involve the CEC can be tested; these include alter-and-display operations, debug functions, channel service and diagnostic operations, concurrent application of microcode fixes, and system event traces.

I/O models

To simulate I/O operations, z/CECSIM provides high-level software models that mimic the operations of memory bus adapters (MBAs) and coupling links (intersystem channels, or ISCs). I/O channels such as ESCON* and FICON* are simulated to the extent that status information defined by test cases is returned. This approach satisfies the needs of CEC microcode verification. A model that is based on the actual hardware description (e.g., VHDL) would execute the real logic, but it would be far too slow for an efficient microcode simulation environment. The I/O models implemented in z/CECSIM allow execution of all affected code paths in i390 and millicode and verification at the z/Architecture level. The components of the I/O software model are described below.

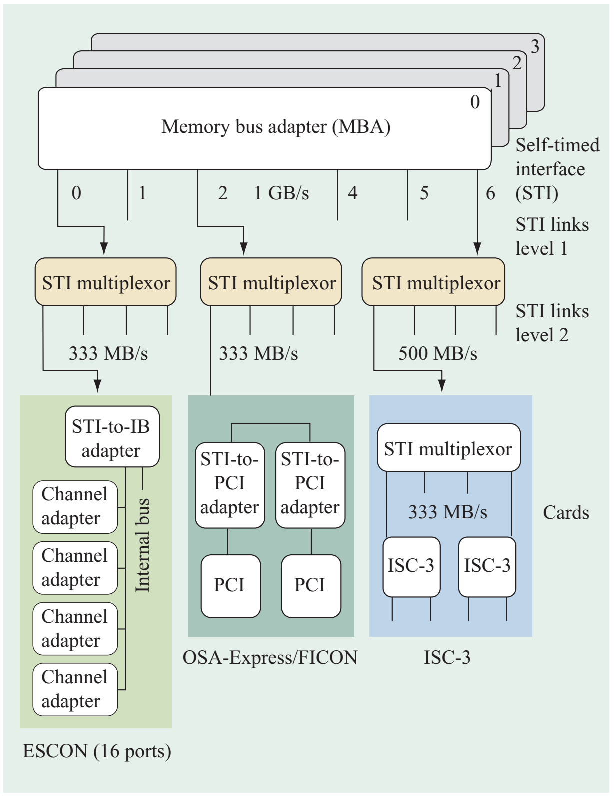

Figure 3

IBM eServer z900 I/O structure.

z900 I/O structure

Figure 3 shows a simplified view of the treelike z900 I/O structure [11]. The root of this tree consists of four MBAs that perform direct memory access (DMA) operations. At one end, these chips are connected to the z900 cache logic and its processors. At the other end, each MBA is connected via six high-speed self-timed interface links (STI, 1 GB/s) to multiplexor chips. These, in turn, split one high-speed STI link into four STI links with lower speed (500/333 MB/s). The lower-speed links, which are compatible with the STI links used in 9672 G5 and G6 systems [12], connect to various I/O cards:

- OSA-Express (two high-speed network interfaces, e.g., Ethernet, ATM).
- FICON (two Fibre Channel interfaces).
- ESCON (16-port legacy fiber optic connections to external control units).
- ISC-3 (fiber optic coupling connection to other zSeries systems).

On each of these I/O cards, some further multiplexing (ESCON) or daisy-chaining (OSA-Express, FICON) takes place. Each of the channels consists of some interface logic (channel adapter) and a channel engine (a processor that executes channel microcode). In addition to the elements shown in Figure 3, STI links themselves can be used as very fast coupling connections (integrated cluster bus, or ICB) and connections to a cage containing legacy I/O (internal-bus-based I/O cards, e.g., parallel channel adapters).

Models for MBA and multiplexor chips

The I/O model uses a register-based approach; i.e., each I/O unit type is described by its register contents. This approach is sufficient for code verification, since from a microcode point of view the only means for interaction with I/O hardware is reading and writing registers. The characteristics of each type of hardware unit, such as register layout and register manipulation during command execution, are described in lookup tables. This concept offers a convenient way to set up an I/O configuration and also to implement new I/O unit types. The I/O configuration used in simulation is set up during z/CECSIM start by examining the internal configuration control blocks that are used also with zSeries hardware.

All accesses to I/O components are initiated by millicode instructions. Therefore, in simulation, the

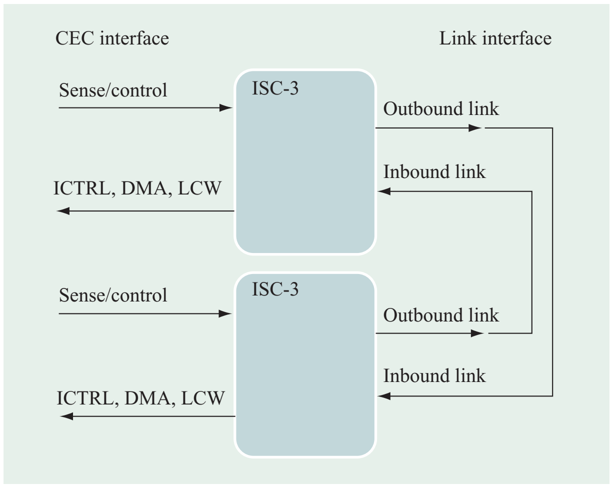

Figure 4

Connection between CEC and ISC-3.

millicode emulator invokes the I/O model and passes the hardware address and command information. The command is then routed to the specified MBA [i.e., the ReceiveCommand( ) method of the addressed MBA object is called; even though the model is implemented in ANSI C, the terminology of object-oriented programming is well suited to describe its structure].

If the command is not addressed to the MBA itself, it is transferred to the attached STI multiplexor by calling its receive method. To address STI multiplexor objects, the MBA contains a list of pointers to all objects attached to its links. A null pointer here indicates that the addressed object (i.e., the adapter) does not exist. Before finally calling the multiplexor receive method, the MBA object checks the link status (operational vs. not operational) using the register contents of the simulated MBA hardware. If the link to the addressed adapter is not operational or the addressed adapter does not exist at all, the command is rejected, and an error message is logged. Assuming that the adapter works without any error, the same procedure is performed for the next levels until the control block reaches the correct I/O model object.

Next the command is executed (i.e., the ExecuteCommand method of the addressed object is called), and register data is read or modified. These modifications trigger further actions in the I/O model. For example, setting a bit indicating an error results in summary bits being set in other registers in the simulated hardware. Finally, an interrupt condition is signaled to a processor thread and recognized by millicode.

Model for standard I/O channel paths

From the i390 perspective, an I/O channel is triggered by writing to its channel communication area (CCA), which is a register located in the channel adapter logic. The channel reads the incoming command, which is typically a start function, initiated by a *start subchannel* instruction. Processing includes DMA operations to fetch the channel program and to fetch and/or store data. Also, the channel updates the control blocks of the channel subsystem to present status, such as channel end/device end. Finally, the channel indicates completion of the operation by writing to the CCA register. This condition is recognized by millicode.

Within the I/O model, there is a separate component that takes the role of the channel. It receives control when a command is written to the CCA register. This channel model reads the CCA and generates status information by updating control blocks in CEC storage. It responds to the CEC by writing to the CCA register. No channel program is fetched or interpreted, and no data transfer occurs. The model allows arbitrary responses to be predefined (e.g., unit check, channel control check) that cause i390 and/or millicode to execute any desired code path.

To verify the real channel microcode in cooperation with i390 code, the I/O model contains a TCP/IP network interface to attach channel code simulators. The protocol used on this interface is the message protocol actually used on the STI links, with an additional wrapper around the STI information packets. Today, this interface is exploited by OSASIM, a simulation environment for OSA-Express microcode.

Models for coupling channels

With the z900, a new type of coupling channel called ISC-3 was introduced [13]. It did not follow former designs, in which the channel consisted of an optical module, a processor, control logic, and memory. With the new design, logic was moved to i390 code on the SAP, which eliminated the need for an extra processor. As a result, the former ISC code had to be completely redesigned to fit the new ISC-3 environment.

Figure 4 shows the interfaces of an ISC-3. The ISC-3 receives sense/control commands through the CEC interface. On the inbound optical link, it receives frames from the ISC-3 on the other end of the link. The ISC-3 sends DMA and internal control requests to the CEC. In addition, it sends frames on the outbound optical link to the ISC-3 on the other end of the link.

z/CECSIM implements a software model of the ISC-3 hardware. This model is triggered by z/Architecture coupling instructions, e.g., *send message*. It simulates the entire ISC-3 code, including the transfer of all status information and data. Moreover, the ISC-3 model not only mimics the functional behavior of ISC-3 hardware, but it can also be set up to inject all kinds of hardware errors that are reflected to ISC-3 code.

Numerous test cases were written to verify ISC-3 code. For basic verification, the test cases consisted of z/Architecture programs that issued coupling instructions and verified the results at the software level. However, a number of complex scenarios had to be tested as well:

- Valid states that are difficult to generate on the real hardware, e.g., due to timing behavior.
- Storage errors that occur when ISC-3 access to CEC storage fails. The ISC-3 code implements complex code flows to deal with such situations.
- Errors on the optical link that can occur when the ISC-3 receives data.
- Hard-stop errors, where ISC-3 hardware is frozen and recovery by ISC-3 code is required.
- Bugs in the hardware that are circumvented by microcode.

All of the above situations have to be covered by ISC-3 code, but occur only under very rare circumstances. For these cases, the ISC-3 model provides a REXX interface. A REXX program is associated with a test case. The ISC-3 model invokes this program at a specified event (e.g., when an outbound frame is about to be sent, or when sense/control operation occurs). The REXX program may then modify incoming or outgoing data. This unusual behavior must be handled correctly by ISC-3 code. In addition to its use with z/CECSIM, the ISC-3 model can be run on a workstation by itself. This environment, which provides a source-level debugger, was used for basic tests of ISC-3 code.

Several test programs have been used with z/CECSIM. In most cases, an S/390 test program capable of running on the hardware was used to execute test cases for I/O and coupling operations at the z/Architecture level. A variety of standalone assembler programs have been run, as well as the I/O configuration program (IOCP). To ensure continuous quality of the delivered code, test-case packages were run as a regression test.

Concluding remarks

With z/CECSIM, the eServer z900 CEC microcode community made a major step forward in the area of code testing, integration, and verification. For the first time it was possible to run millicode, i390, LPAR hypervisor, and support-element code in a common environment, independently of the availability of target hardware. This environment was available months before the first z900 hardware was installed on the test floor. There is still a need to verify microcode on engineering hardware, especially for I/O device operations, performance measurements, complex multiprocessing, and hardware error recovery scenarios, but the majority of the code paths can now be run and debugged in simulation. The trace facilities allowed a detailed path-length analysis for performance-critical operations (namely, the execution of *start subchannel*). An investigation of the code problems after completion of the project showed that in those areas where simulation test-case packages were run regularly, 80% of the problems were detected before the code reached the test floor. Only 20% of the problems had to be debugged on the real machine. This result also supports the need to extend test-case packages to further increase simulation coverage in the future.

Because z/CECSIM is a VM application, it is available to all code developers and testers. It was used by all z900 development locations: Endicott, New York, Poughkeepsie, New York, Boeblingen, Germany, and Austin, Texas. Microcode development could deliver code to system integration at a very high quality level that effectively reduced the bringup and testing times. It was estimated that this approach of system-wide microcode simulation gained about three months in the bringup and integration phase compared to previous systems.

Since the major part of the i390 instruction stream is executed within the native performance capabilities of the underlying VM system and does not require interpretation, z/CECSIM is a highly efficient debugging environment. The simulation of a power-on-reset on z/CECSIM with an SE connection takes less than twice the time needed on actual hardware. It is even faster if no SE connection is required. A hardware model would run for many hours, and thus is not suitable for general-purpose debugging. For users running small test programs, z/CECSIM appears to be nearly as fast as the real system. Many code developers were able to complete their work by using only the simulation environment, without the need for expensive and limited engineering hardware. This has resulted in a reduction in the number of engineering systems needed for z900 development.

The modular concept allows various user groups to tailor the environment to their specific needs. While SE code developers need the SE connection of z/CECSIM all the time, i390 developers can often use the standalone setup of z/CECSIM for simplicity. Millicode developers normally use MCE standalone before testing their code on z/CECSIM. In addition, z/CECSIM is used to create memory images after simulating power-on reset. This machine state is then loaded into hardware simulation models for further tests. This concept is highly efficient in early unit test phases as well as during later stages of system integration. Furthermore, automatic regression runs allow verification of the latest code levels during night shifts. Results are available for analysis in the morning.

In summary, z/CECSIM has proven to be an effective, comprehensive, and powerful simulation environment for the microcode development of the IBM z900. For the development of future zSeries systems, z/CECSIM itself runs on 64-bit z/VM on a z900. It uses an IBM-internal version of CMS that includes very basic support for z/Architecture and 64-bit addressing. In this environment, z/CECSIM uses the z/Architecture version of SIE, and the 64-bit instructions introduced with the z900 can be executed directly on the underlying hardware. Instructions such as LG, AG, and STG no longer have to be simulated, which is very important once they are heavily exploited by i390 code. Also, very large CECs can now be simulated because the 2GB storage limit no longer applies. These and other enhancements further improve and extend the simulation capabilities for follow-on systems of the IBM z900.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of The Institute of Electrical and Electronics Engineers, Inc. (IEEE).

References

1. C. F. Webb and J. S. Liptay, "A High-Frequency Custom CMOS S/390 Microprocessor," *IBM J. Res. & Dev.* **41,** No. 4/5, 463–473 (1997).
2. M. A. Check and T. J. Slegel, "Custom S/390 G5 and G6 Microprocessors," *IBM J. Res. & Dev.* **43,** No. 5/6, 671–680 (1999).
3. J. Maergner and H. R. Schwermer, "High Level Microprogramming in I370," *The Design of a Microprocessor*, W. G. Spruth et al., Eds., Springer, New York, 1989, pp. 303–316.
4. B. Wile, M. P. Mullen, C. Hanson, D. G. Bair, K. M. Lasko, P. J. Duffy, E. J. Kaminski, Jr., T. E. Gilbert, S. M. Licker, R. G. Sheldon, W. D. Wollyung, W. J. Lewis, and R. J. Adkins, "Functional Verification of the CMOS S/390 Parallel Enterprise Server G4 System," *IBM J. Res. & Dev.* **41,** No. 4/5, 549–566 (1997).
5. 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 (2002, this issue).
6. IBM Corporation, *z/Architecture Principles of Operation*, Order No. SA22-7832; available through IBM branch offices.
7. IBM Corporation, *z/VM CMS Application Multitasking*, Order No. SC24-5961; available through IBM branch offices.
8. IBM Corporation, *IBM 370-XA Interpretive Execution*, Order No. SA22-7095; available through IBM branch offices.
9. A. Chandra, V. Iyengar, D. Jameson, R. Jawalekar, I. Nair, B. Rosen, M. Mullen, J. Yoon, R. Armoni, D. Geist, and Y. Wolfsthal, "AVPGEN—A Test Generator for Architecture Verification," *IEEE Trans. Very Large Scale Integration* (*VLSI*) Syst. **3,** No. 2, 188–200 (June 1995).
10. IBM Corporation, *2064 zSeries Support Element Operations Guide 1.7.1*, Order No. SC28-6811, available through IBM branch offices.
11. D. J. Stigliani, Jr., 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 (2002, this issue).
12. J. M. Hoke, P. W. Bond, T. Lo, F. S. Pidala, and G. Steinbrueck, "Self-Timed Interface for S/390 I/O Subsystem Interconnection," *IBM J. Res. & Dev.* **43,** No. 5/6, 829–846 (1999).
13. T. A. Gregg and R. K. Errickson, "Coupling I/O Channels for the IBM eServer z900: Reengineering Required," *IBM J. Res. & Dev.* **46,** No. 4/5, 461–474 (2002, this issue).

Received September 8, 2001; accepted for publication February 12, 2002

Joachim von Buttlar *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. During the 9221 project from 1990 to 1991, he worked as Liaison Engineer on international assignment in Endicott, New York. In 1997 he initiated the z/CECSIM project, developed its concepts, and implemented the simulator's kernel. His current responsibility is z/CECSIM for the next generation of zSeries systems.

Harald Böhm *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (hboehm@de.ibm.com).* Dr. Böhm received a Ph.D. degree in mathematics from the University of Karlsruhe in 1984, with research experience in numerical analysis. He subsequently joined the IBM development laboratories in Boeblingen to develop software for numerical computations. Since 1987 his activities have involved the verification of processor hardware and the development of microcode for S/370, S/390, and zSeries servers.

Reinhard Ernst *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (rernst@de.ibm.com).* Dr. Ernst received M.S. and Ph.D. degrees in physics from the University of Bonn, Germany, in 1997 and 2000, respectively. In 2000 he joined the IBM development laboratories in Boeblingen. His current responsibilities are i390 code development in the I/O microcode group and the I/O model for z/CECSIM.

Axel Horsch *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (ahorsch@de.ibm.com).* Mr. Horsch received a Dipl.-Ing. degree in electrical engineering from the Ingenieurschule in Augsburg, Germany, in 1965. That same year he joined the IBM development laboratories in Boeblingen as a test engineer to work on system testing of IBM 370/125, 4331, 4361, and 9370. He was responsible for system I/O attachment testing, system microcode testing, and system microcode error projections. In 1989 he joined the S/390 system microcode development group. He is currently responsible for the design and development of i390 code for channel service functions and for the i390 code simulation of all S/390- and z/Architecture-based systems.

Andreas Kohler *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.

Herbert Schein *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (schein@de.ibm.com).* Dr. Schein studied at the Technical University of Vienna, Austria, where he received a Dipl. Ing. degree in mathematics in 1972 and a Dr. Ing. degree in computer science in 1973. In 1974 he joined the IBM development laboratories in Boeblingen, Germany. He initially developed microcode for S/370 disk attachments, and subsequently worked in various tool projects. He is currently involved in the development and continued enhancement of components for z/CECSIM.

Michael Stetter *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (stetter@de.ibm.com).* Mr. Stetter is a Staff Software Engineer in the zSeries system simulation group. He 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 has worked in a variety of system simulation assignments, and is currently I/O microcode simulation team leader and z/CECSIM representative for the next generation of zSeries systems. Mr. Stetter is a PMI®-certified project management professional and received the master certificate in project management from George Washington University, Washington, D.C.

Klaus Theurich *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (theurich@de.ibm.com).* Mr. Theurich studied electrical engineering at the Fachhochschule Esslingen, graduating in 1987. He joined the IBM development laboratories in Boeblingen that same year to work on S/370 system testing and development for the Parallel Processing Compute Server (PPCS), and on hardware development for the first intersystem channel (ISC). In 1994, he joined the S/390 I/O microcode development effort. During an international assignment from 1998 to 2000 in Poughkeepsie, New York, he worked on a new concept for simulation of coupling microcode. He is currently responsible for investigating the Infiniband simulation concept.