Multiple-logical-channel subsystems: Increasing zSeries I/O scalability and connectivity

L. W. Wyman
H. M. Yudenfriend
J. S. Trotter
K. J. Oakes

With the advent of the z990 multi-book multiprocessor family of server offerings, significant increases in total system capacity and scalability can be realized. Essential to an increased processing capacity is the corresponding need for significant increases in total I/O scalability and connectivity. With the z990, increased I/O capacity is provided by increasing the number of physical I/O channels that can be configured to the system and by restructuring the physical channel subsystem (CSS) into logically distinct channel subsystems. This restructuring is commonly called the multiple-channel subsystem (MCSS) facility. Each logical CSS is then assigned to one or more logical partitions as necessary in order to provide the total I/O connectivity necessary to accommodate the increased processing capacity of the system. An overview of the z990 MCSS architecture is presented with respect to how it is structured, the channel-subsystem constraints that have been removed, and how MCSS functions are provided to the operating systems executing in each of the system's logical partitions (LPARs) in a predominantly transparent manner. Also discussed is the channel-subsystem hardware and firmware (embedded software) design necessary to accommodate the MCSS architecture, as well as overviews of the MCSS I/O configuration process and the z/OS® programming support necessary to accommodate the MCSS facility. Finally, enhancements to the MCSS I/O measurement facility necessary to facilitate autonomic computing are discussed.

Introduction

In contemporary zSeries* computing systems [1] and their predecessor systems dating back to the S/370* XA architecture systems of the late 1970s [2], each system footprint, called a central processing complex (CPC), was limited to a maximum of 256 I/O channels, called channel paths, that could be configured to the CPC. This 256-channel-path maximum did not significantly limit the overall growth, in terms of total system capacity, of previous S/370 XA, S/390* [3], and z/Architecture* class systems. However, with the advent of the z990 multibook system [4] and its significantly increased maximum processing capacities, the need to provide more than 256 channel paths becomes integral to achieving a balanced system environment (processor, memory, and I/O). More than 256 I/O channel paths are fundamental to the larger

Note: For the reader's convenience, acronyms as used in this paper are expanded in an appendix at the end of the paper.

0018-8646/04/$5.00 © 2004 IBM

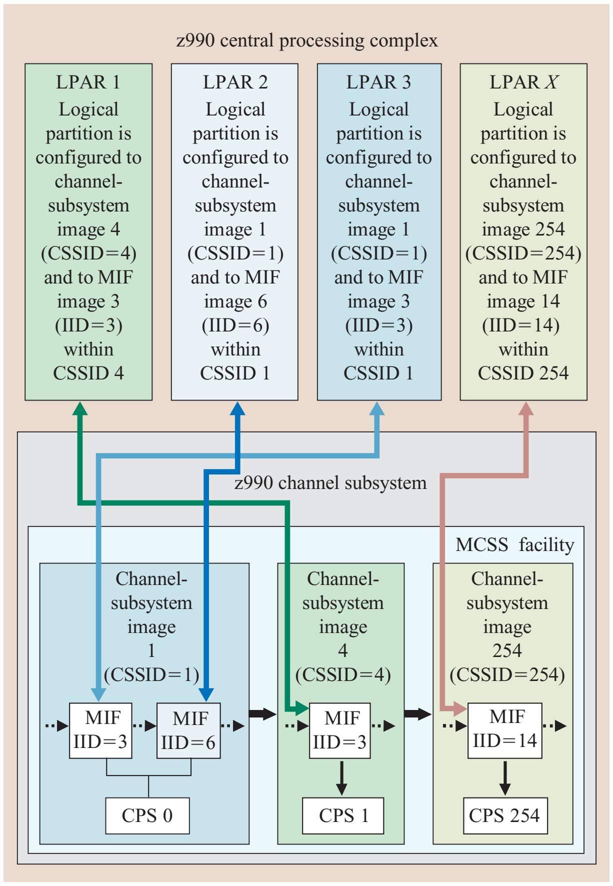

z990 central processing complex
LPAR 1 Logical partition is configured to channel-subsystem image 4 (CSSID=4) and to MIF image 3 (IID=3) within CSSID 4
LPAR 2 Logical partition is configured to channel-subsystem image 1 (CSSID=1) and to MIF image 6 (IID=6) within CSSID 1
LPAR 3 Logical partition is configured to channel-subsystem image 1 (CSSID=1) and to MIF image 3 (IID=3) within CSSID 1
LPAR X Logical partition is configured to channel-subsystem image 254 (CSSID=254) and to MIF image 14 (IID=14) within CSSID 254
z990 channel subsystem
MCSS facility
Channel-subsystem image 1 (CSSID=1)
Channel-subsystem image 4 (CSSID=4)
Channel-subsystem image 254 (CSSID=254)
MIF IID=3
MIF IID=6
MIF IID=3
MIF IID=14
CPS 0
CPS 1
CPS 254

Figure 1

Overview of the z990 MCSS facility.

z990 offerings in order to service the increased processing capacity of these systems. The increased I/O connectivity allows for the consolidation of multiple smaller server platforms without requiring redesign of the existing I/O topologies of these smaller servers.

In addition to the requirement for increased I/O capacity and connectivity, a basic programming requirement existed; namely, that of providing the increased I/O processing capacity in a manner that minimizes the changes necessary to the zSeries operating systems, supporting vendor products, applications, their associated I/O-configuration control information, and user documentation, in order to support more than 256 channel paths.

Finally, the multiple-channel-subsystem (MCSS) architecture, design, and support efforts focused on expanding upon the zSeries existing multiple-image facility (MIF) in order to 1) minimize the changes necessary to provide greater I/O capacity, 2) build upon and increase the MIF channel-sharing capabilities, and 3) ensure backward compatibility with previous zSeries computing environments. The z990 MCSS facility achieves these fundamental requirements as discussed below.

Constraints to increased I/O capacity

In order to achieve the necessary increase in the total I/O capacity of the z990 system, several z/Architecture constraints had to be addressed and redefined in a manner that minimizes their impact on providing more than 256 channels and associated I/O devices to the zSeries operating systems, such as z/OS*, z/VM*, z/Linux, the zSeries Transaction Processing Facility (TPF), and the VSE/ESA* operating systems, that execute in the logical partitions configured to the z990 system. Specifically, the architecturally defined channel-path identification number, called the channel-path identifier (CHPID), had to be maintained without change. The CHPID value is defined as an 8-bit binary number resulting in a range of unique CHPID values from 0 to 255; therefore, a maximum of 256 channel paths were possible on previous S/370, S/390, and z/Architecture-class systems. Since the inception of the precursor S/370 XA channel-subsystem architecture in the late 1970s, this 8-bit CHPID has been maintained without change because of its pervasive use in the z/OS and z/VM operating systems. For example, the CHPID value is maintained in many internal programming control blocks, is displayed in various operator messages, and is the object of various system commands, programming interfaces, etc., all of which would have to be redesigned if the CHPID value was increased to more than an 8-bit number in order to accommodate more than 256 channel paths.

In addition to increasing the total number of channel paths that may be configured to a z990-class system, configuring a corresponding increase in the number of I/O devices to a large z990 system was necessary. In z/Architecture, each I/O device is represented by a separate set of controls, called subchannels, which are used by the channel subsystem to activate, monitor, and report the progress of I/O operations for their associated I/O device. Prior to MCSS, the z/Architecture provided a maximum of 64K subchannels and an equal maximum number of I/O devices. As was true with the limited number of channel paths on previous S/390 and zSeries systems, this 64K maximum I/O device limitation had to be removed in a manner that caused minimal disruption to the zSeries operating systems and their associated application programs.

Eliminating the I/O constraints

The MCSS facility, as depicted in **Figure 1**, extends the z/Architecture CSS to provide up to 65,280 channel paths.

A corresponding increase in the total number of attaching control units and I/O devices is also provided in a manner that is predominantly transparent to the programs (i.e., the operating systems and their associated applications) executing in each of the configured LPARs on the z990 system. This was accomplished in the following manner.

1. An additional level of channel-path-addressing indirection is created that allows more than 256 physical channel paths to be installed and uniquely identified without changing the legacy 8-bit CHPID value and the corresponding programming dependencies on the CHPID. This new channel-path-identification value, called the physical-channel identifier (PCHID), is a 16-bit binary number ranging from 0 to 65,279, which uniquely identifies each physically installed channel path. With the z990, a current maximum of 1024 external channel paths (e.g., ESCON*, FICON*, OSA) and 48 internal channel paths (e.g., Internal Coupling and IQDIO hyperlink) are each assigned a unique PCHID value in the range of 0 to 2,047. With the exception of the z990 I/O configuration program (IOCP) and the dynamic I/O configuration programming, both of which are used to create and modify the z990 channel subsystem I/O configurations, the PCHID value is transparent to the programs operating in each LPAR. Correspondingly, both of these I/O configuration management programs are enhanced to provide the controls necessary to associate the PCHID value of each channel path with its corresponding CHPID values.
2. The physical CSS is restructured into multiple "logical" channel subsystems. Each logical CSS is called a channel-subsystem image, and each image is identified by a unique 8-bit binary number ranging from 0 to 254, called the channel-subsystem-image identifier (CSSID), resulting in an architecture maximum of 256 channel-subsystem images per central processor complex (CPC) footprint. Additionally, each CSS image may be configured with a maximum of 256 unique channel paths, called a channel-path set (CPS). This results in an architecture maximum of 64K physical channel paths for a given CPC footprint.

   Each channel-subsystem image is also structured to provide its own z/Architecture MIF. The multiple-image facility (previously supported by some S/390 and all previous z/Architecture models) that is associated with each channel-subsystem image provides for the replication of both channel-path and subchannel controls. This is necessary to allow each of the logical partitions that are configured to a given channel-subsystem image to have its own set of I/O controls in order to dynamically access and share up to 256 physical channel paths and up to 64K physical I/O devices that may be attached to the channel paths configured to the channel-subsystem image.
3. The multiple-image facility is also extended in order to allow physical channel paths to be defined and concurrently configured to multiple channel-subsystem images. Such shared channel paths are called "spanned" channel paths because they allow the channel paths and their attached I/O devices to be dynamically and transparently shared by programs operating in LPARs which are configured to different channel-subsystem images; that is, they span multiple channel-subsystem images.

Correspondingly, each configured z990 LPAR is assigned to an appropriately defined CSS image in order to accommodate the I/O connectivity requirements of the operating system and associated application programs that are executed in each of the configured LPARs, as depicted in Figure 1.

Overview of MCSS architecture

Channel-subsystem images

Each z990 CSS image, depicted in **Figure 2**, consists of a single unique set of MIF subchannel controls called subchannel images, an associated set of MIF logical channel-path controls called channel-path images (CPIs), and a related set of physical channel paths and associated controls called a CPS, containing from 1 to 256 physical channel paths. Collectively, these I/O controls are called a channel-subsystem image. The channel-subsystem image extends both the MIF and associated physical-channel-path controls in order to provide a means by which the LPAR hypervisor[1] [5] can assign subchannels and channel paths (either nonshared or shared channel paths) to each of the configured LPARs. The MCSS architecture provides from 1 to 255 channel-subsystem images that may be configured either by z990 model-dependent means or by use of the appropriate z/Architecture dynamic I/O configuration commands. As previously mentioned, each channel-subsystem image is specified by a unique 8-bit binary integer called the CSSID. Either the LPAR hypervisor or the z990 IOCP can assign each configured channel-subsystem image to a maximum of 15 LPARs.

Additionally, the MCSS facility extends the z/Architecture Start Interpretive Execution (SIE) I/O-instruction interpretation controls used by the LPAR hypervisor to associate the correct channel-subsystem resources with each of the logical processors associated with each LPAR. When the program operating in an LPAR executes an I/O instruction [e.g., a Start

[1] Hypervisor: A software layer to manage multiple operating systems running in a single central processing complex.

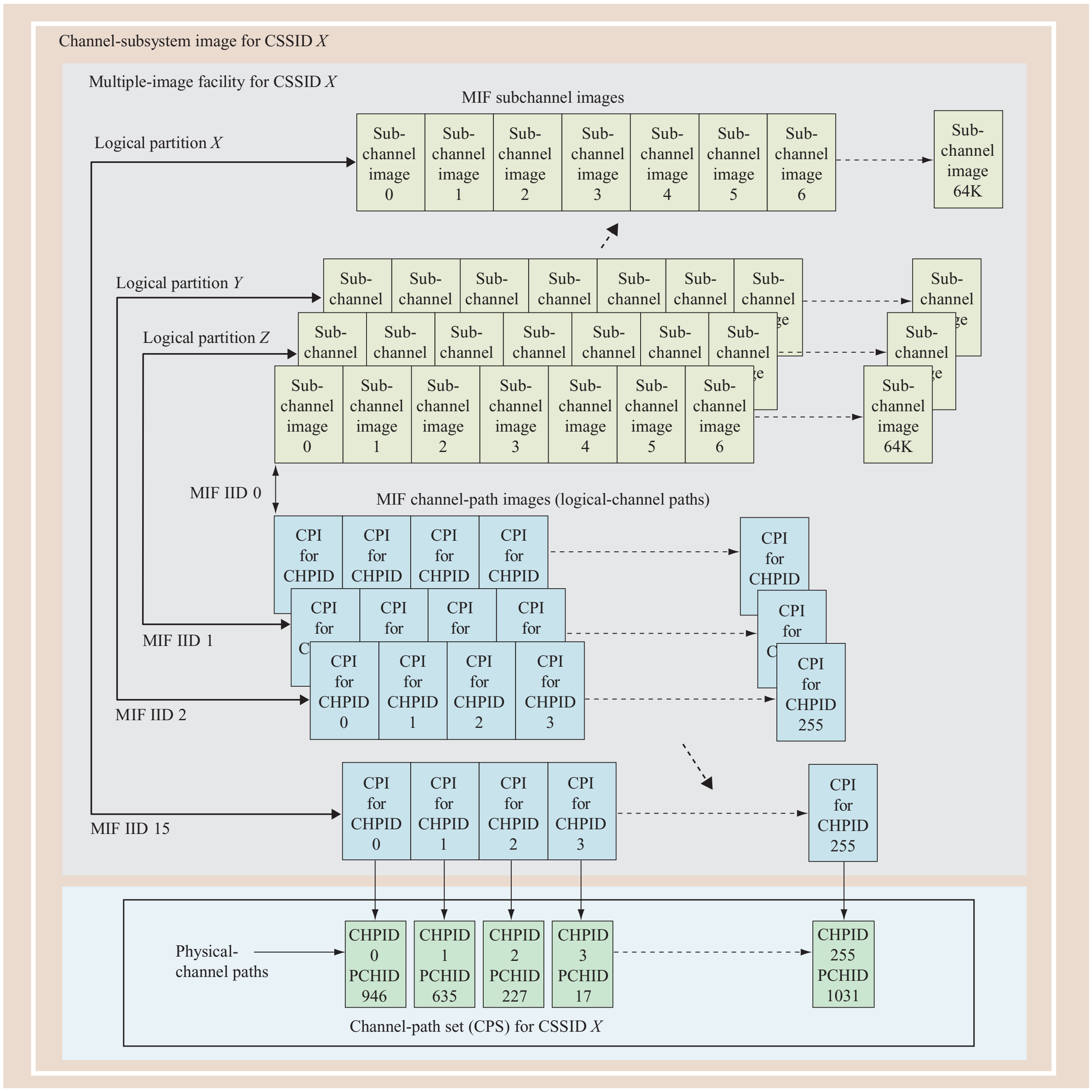


Figure 2

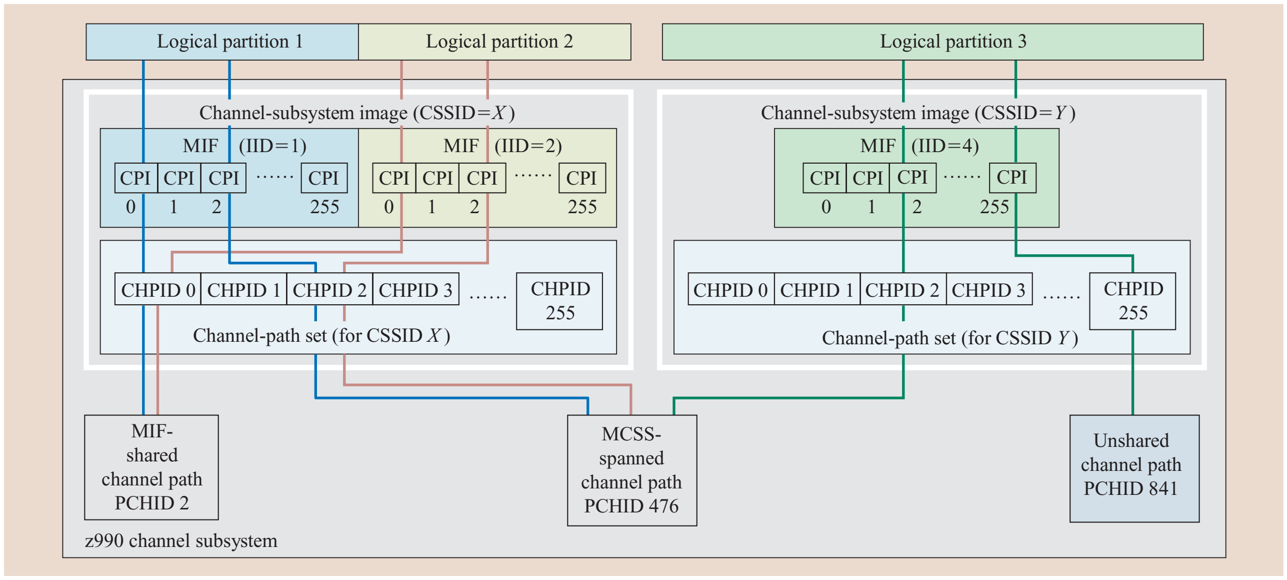
MCSS channel-subsystem image.

Subchannel (SSCH) instruction], the SIE extensions for the MCSS facility transmit both the CSSID and the MIF IID to the CSS in a manner that is transparent to the programs operating in the LPAR. In turn, the CSS uses the transmitted CSSID and image identification (IID) to access the correct set of channel-subsystem image I/O controls as described later in the section entitled "Subsystem-identification word (SID)."

MIF extensions for the MCSS facility

The MCSS facility extends the MIF as follows:

1. The MCSS architecture provides from 1 to 255 unique MIFs and their associated I/O controls. One MIF is implicitly created and configured to each configured channel-subsystem image.
2. All of the pre-MCSS channel path and attached I/O-



Logical partition 1
Logical partition 2
Logical partition 3
Channel-subsystem image (CSSID=X)
MIF (IID=1)
CPI CPI CPI ...... CPI
0 1 2 255
MIF (IID=2)
CPI CPI CPI ...... CPI
0 1 2 255
Channel-subsystem image (CSSID=Y)
MIF (IID=4)
CPI CPI CPI ...... CPI
0 1 2 255
CHPID 0 CHPID 1 CHPID 2 CHPID 3 ...... CHPID 255
Channel-path set (for CSSID X)
CHPID 0 CHPID 1 CHPID 2 CHPID 3 ...... CHPID 255
Channel-path set (for CSSID Y)
MIF-shared channel path PCHID 2
MCSS-spanned channel path PCHID 476
Unshared channel path PCHID 841
z990 channel subsystem

Figure 3

MCSS-spanned channel path.

device dynamic sharing functions provided by the MIF are maintained in MCSS-class systems. Additionally, each MIF function that requires the specification of the MIF IID is extended to require the CSSID of the corresponding channel-subsystem image with which the specified MIF image is associated. For example, I/O instructions that require a MIF IID to be specified in the instruction operand subsystem-identification word (SID) are extended to also require the specification of the CSSID of the associated channel-subsystem image. As previously mentioned, both the MIF IID and the CSSID are implicitly provided to the channel subsystem, on behalf of the program executing in the LPAR, by the MCSS extensions to SIE I/O-instruction interpretation.

3. Each MIF within a channel-subsystem image provides for a maximum of 16 images. Image IDs range from 0 to 15, with image 0 reserved for use by the LPAR hypervisor. Consequently, each channel-subsystem image may be configured to a maximum of 15 LPARs.
4. Each provided MIF is configured to and associated with the physical channel paths contained in the channel-path set associated with the channel-subsystem image.
5. Spanned channel paths are provided. Depending on the type of channel path, the channel path and its associated I/O devices may be configured to multiple channel-subsystem images. Spanned channel paths extend the MIF sharing capability to z990 configurations that require shared channel-path access by more than 15 LPARs or shared access by LPARs configured to different channel-subsystem images. See also the section entitled *Sharing beyond 15 logical partitions—spanning* later in this paper. **Figure 3** depicts a spanned channel path. MCSS-spanned channel paths extend the MIF channel path and I/O device-sharing capabilities for z990 configurations that require shared access to one or more common physical-channel paths by more than 15 LPARs or by LPARs assigned to different channel-subsystem images.

Channel-path sets (CPSs) for the MCSS facility

The MCSS architecture provides from 1 to 255 groups of physical channel paths, called channel-path sets (CPSs). Each channel-subsystem image is configured to a single unique CPS. With the exception of spanned channel paths, each physical-channel path configured to each CPS is unique among all other physical-channel paths configured to other channel-subsystem images.

Because as many as 255 channel-subsystem images may be provided, the CHPID values assigned to each physical-channel path in each of the channel-path sets may not be unique. In order to accommodate these CHPID synonyms for each physical-channel path, the following channel-path identifications are provided by the MCSS architecture:

1. As observed by the programs executing in each LPAR, each channel path configured to each LPAR is specified by the CHPID value assigned to the physical-channel path, just as when the MCSS facility is not provided.

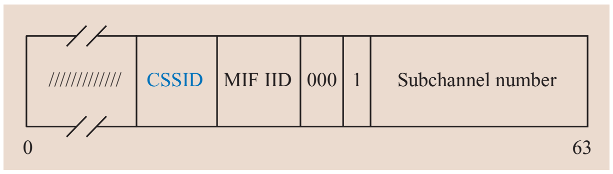


Figure 4

Subsystem-identification word.

For example, the reset-channel-path (RCHP) instruction still provides a CHPID operand value [1] as the means for the program to identify and initiate a resetting operation for the CHPID-specified channel path. In general, the only programming exceptions to the continued use of the CHPID value to specify a channel path exist within the dynamic I/O configuration commands used to create and modify the z990 I/O configuration definitions and within the z990 I/O configuration program (IOCP).

2. As observed by the LPAR hypervisor and the channel subsystem, the CHPID, CSSID, MIF IID, and PCHID values may all be used either collectively or in part to access and control a given channel path as a function of the specific hypervisor, channel subsystem, or dynamic I/O configuration operation being performed.

Subsystem-identification word (SID)

As with pre-z990 z/Architecture servers, the SID operand is used by all z/Architecture I/O instructions that require a subchannel specification as the means of identifying the target subchannel associated with a specific I/O device. For example, the Start Subchannel (SSCH) instruction that is used to initiate an I/O operation with a specific I/O device requires an SID operand specification as the means for identifying the subchannel that is used to access the associated device. With the advent of the MCSS facility, the SID is extended to provide the CSSID of the channel-subsystem image to which the subchannel is configured, as shown in **Figure 4**. The channel-subsystem identifier (CSSID) field specifies the binary number of the channel-subsystem image containing the referenced subchannel. The MIF IID field specifies the binary number of the specific MIF image within the channel-subsystem image containing the referenced subchannel. The subchannel number field specifies the binary number of the specific subchannel image within the specified MIF image and within the specified channel-subsystem image of the target subchannel to be accessed.

For all programs operating in the z990 LPARs (and therefore under control of the LPAR hypervisor via SIE as provided by the z/Architecture SIE facility), both the CSSID value and the MIF IID value are transparent and are not specified by the program operating in the LPAR. With MCSS, the LPAR hypervisor and the SIE architecture controls are extended to allow the hypervisor to implicitly specify the CSSID and MIF IID configured to the target LPAR. For High-frequency I/O instructions that execute interpretively, such as SSCH, the LPAR hypervisor places the CSSID value in the SIE logical processor state description (SD) for each logical processor configured to the LPAR. When the instruction is then interpretively executed by central processor I/O instruction firmware, the CSSID and MIF IID values are implicitly passed to the channel subsystem without hypervisor involvement just as if the program in the LPAR had explicitly specified these values in the SID operand.

For infrequently executed I/O instructions [such as the Modify Subchannel (MSCH) instruction] that are not intended to execute interpretively, execution of the instruction by the program operating in the LPAR causes the LPAR hypervisor to implicitly gain control. The LPAR hypervisor then loads the proper CSSID value (as well as the proper MIF IID value) into an internal copy of the SID operand and re-executes the I/O instruction on behalf of the program operating in the LPAR in a manner that is transparent to the program.

These LPAR hypervisor and MCSS SIE firmware extensions are applied to all of the appropriate z/Architecture I/O instructions as well as to the z/Architecture coupling facility instructions used to access coupling facility channel paths.

Figure 5 depicts a z990 system configuration showing four logical-channel subsystems. The hardware and firmware controls for each logical-channel subsystem are contained in individual channel-subsystem images described previously.

Overview of z990 channel-subsystem design

The z990 CSS comprises all of the hardware and firmware required to implement the zSeries CSS architecture, including all of the different types of channel paths provided by the z990 system. The firmware in the system-assist processors (SAPs) and I/O channel paths performs the bulk of the I/O instructions and I/O interrupt processing. There is also firmware in the CPs that initiates the I/O instructions and participates in the handling of I/O interruptions. The CSS directs the flow of information between I/O devices and main storage. The CSS uses one or more channel paths as the communication links in managing the flow of this information. As part of I/O processing, the CSS also performs channel-path selection and channel-path management functions, such as testing for channel-path availability, selecting an available channel path, and initiating execution of I/O operations over the selected channel path with the attached I/O devices. When

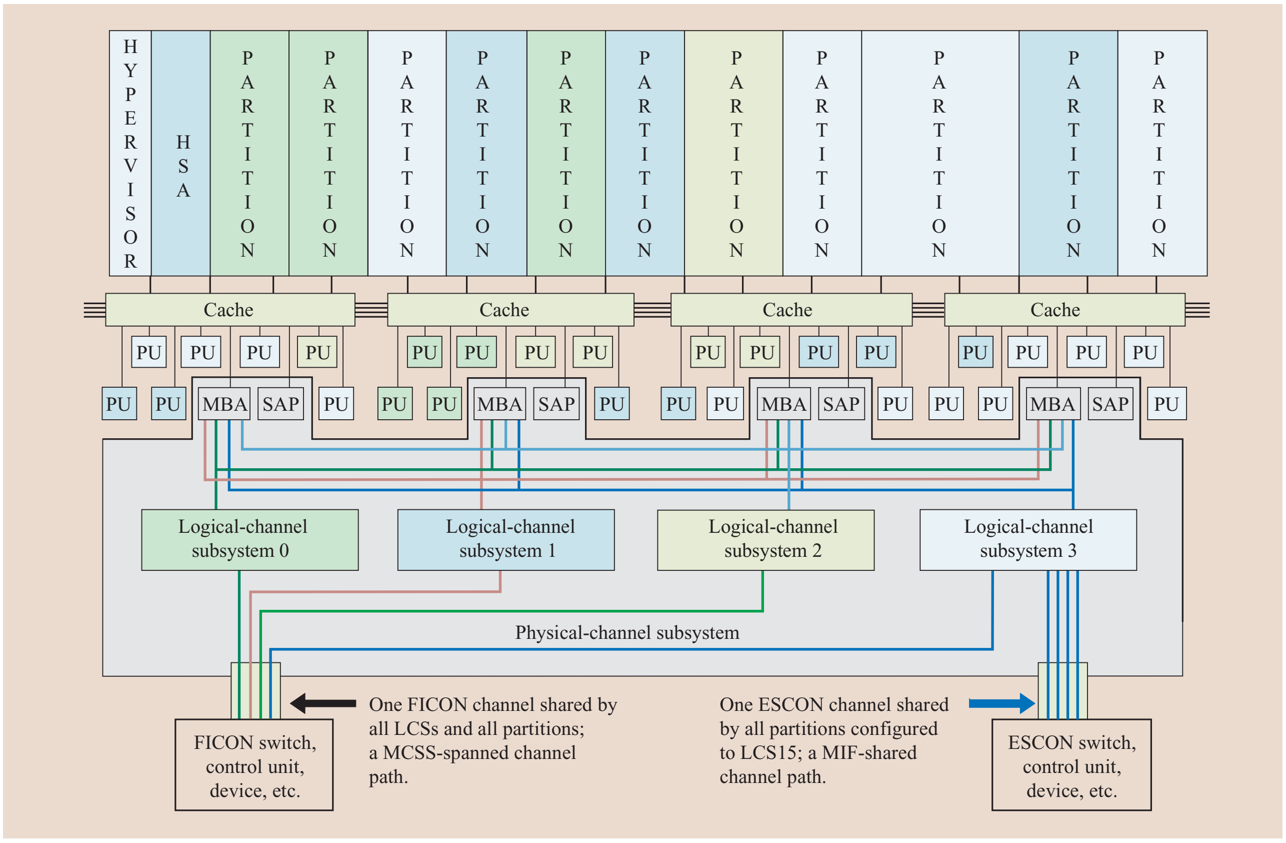

HYPERVISOR
HSA
PARTITION
Cache
PU
MBA
SAP
Logical-channel subsystem 0
Logical-channel subsystem 1
Logical-channel subsystem 2
Logical-channel subsystem 3
Physical-channel subsystem
One FICON channel shared by all LCSs and all partitions; a MCSS-spanned channel path.
One ESCON channel shared by all partitions configured to LCS15; a MIF-shared channel path.
FICON switch, control unit, device, etc.
ESCON switch, control unit, device, etc.

Figure 5

z990 system configuration showing four logical-channel subsystems.

I/O operations are completed, the CSS analyzes the resulting status and transmits it back to the program by use of I/O interruptions and I/O status information.

To provide communication among the various elements in the CSS and to maintain information about the I/O configuration, a set of control blocks are allocated in the hardware system area (HSA)—storage that is accessible only to the embedded software (firmware). One class of control block in HSA is the subchannel control block (SCB). Each SCB contains much of the information used to represent the architected subchannel. As with the architected subchannel, one HSA subchannel for each device is associated with an LPAR; it contains most of the information required to communicate with the associated I/O device. An SCB contains information such as the channel program address, path selection controls, architected path masks, addressing information such as the device address and device number, and subchannel and device status. In short, this is the major control block used to pass information among the elements in the CSS. Some additional control blocks are used to manage I/O operations with the channels, while others allow the queuing of work or interruptions.

Multiple-image facility

As previously stated, the MIF architecture, upon which the MCSS facility is structured, provides the necessary set of controls and facilities that make it possible for multiple LPARs to share the same physical external channels and firmware-implemented internal channels. The MIF architecture makes use of I/O device addressing features, such as the ESCON* [6] and FICON* [7] native I/O interface architecture source logical address (SLA) and the companion destination logical address (DLA) to accomplish this. For ESCON, these fields are each 4 bits. For FICON native, the fields are each 8 bits. They are located in the frames that are transferred on the I/O interface. These fields are used by the MIF implementation as the source and destination LPAR identifiers in the I/O frames. These frames are used to transmit data between the sharing LPARs and the shared I/O devices on the same shared physical channel. The SLA and DLA keep these frames both segregated and unique as observed by the control unit (CU), the I/O device, and the CSS from the frames for other sharing LPARs.

Adding a new dimension—the CSS image

To maintain compatibility with the large installed base of control units and zSeries systems already in the field, the existing ESCON I/O interface architecture [6] had to be preserved in the design and implementation of MCSS. With the 4-bit SLA/DLA limitation for ESCON, the highest logical partition number (PN) that can be set into the ESCON serial I/O frames is 15. Since one of the goals for developing the z990 was to increase the number of LPARs to more than 15, setting the PN into the SLA/DLA was no longer possible. This limitation, along with the goal for preserving the 8-bit CHPID number, resulted in the decision to provide one MIF with one companion channel-path set for each configured CSS image.

This, literally and figuratively, added a new dimension to the CSS. Each MIF can still support up to 15 LPARs. However, instead of having the IID equal to the PN, each CSS image now has a MIF with up to 15 usable IIDs in the range from 1 to 15. With MCSS, the IID is now used in the ESCON frames for the SLA/DLA instead of the PN. This allows ESCON channels, within a channel-path set for a single CSS image, to be shared. Since the z990 supports up to four CSS images, each with 15 IIDs, the CSS could theoretically support up to 4 × 15 LPARs, which is more than sufficient to support the 30-LPAR machine limit on the z990.

Sharing beyond 15 logical partitions—spanning

Prior to the introduction of the MCSS facility, the FICON native architecture [7] had expanded the SLA and DLA to 8 bits. To be consistent with MIF and the way in which ESCON is handled, the rightmost 4 bits of the SLA/DLA have been set to the IID. To expand the sharing capability across more than 15 LPARs, the leftmost 4 bits now contain the CSSID of the logical CSS configured to the LPAR. This concatenation of the CSSID and IID (CSSID.IID) enables the channel path to be shared not only across LPARs configured to a given CSS image, but also shared or spanned between LPARs configured to different CSS images. The z990 supports channel-path spanning for FICON channel adapters, for its Fibre Channel protocol (FCP)/SCSI adapters, networking adapters, coupling facility channel paths, and internal firmware channel paths. Only the ESCON and FICON bridge channel paths do not support spanning, because they must adhere to the ESCON I/O interface architecture at the attached switch and its associated control units.

More I/O devices and subchannels possible with the z990

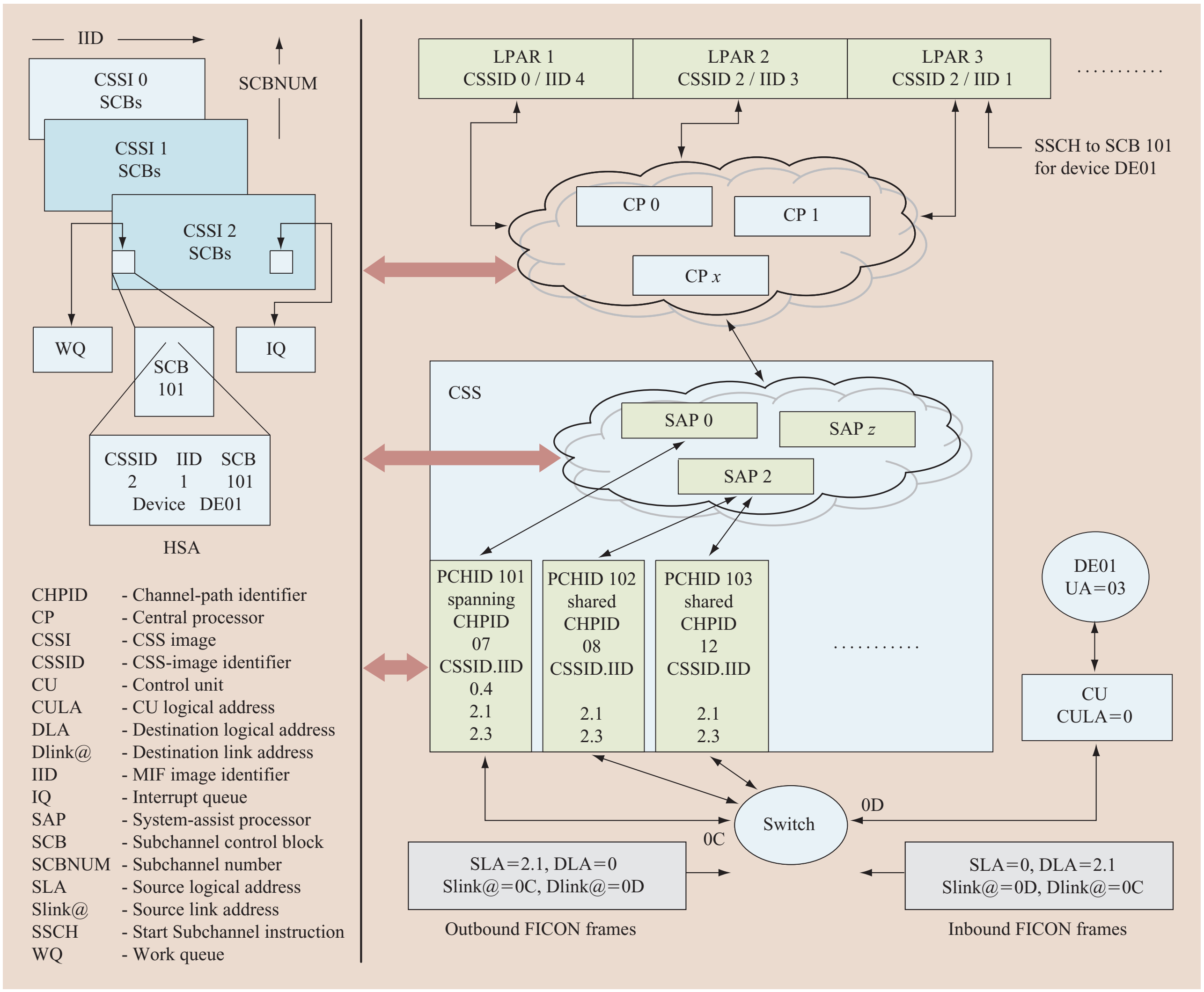
The MCSS facility allows each CSS image and its associated LPARs to be defined with their own unique set of I/O devices. The I/O devices are shared by multiple LPARs configured to a single CSS image, by spanned channel paths configured to multiple CSS images, or by combinations of these. On the z900 zSeries system, up to 63K devices are supported. For the z990 system, each CSS image supports up to 63K devices. When four images are configured, a maximum of 252K devices may be configured. Because each shared device requires a unique subchannel image for each sharing LPAR, each CSS image may have as many as 945K subchannels, and a four-image configuration may have as many as 3.7M subchannels that are concurrently managed by the CSS firmware.

Flow of an I/O operation within MCSS firmware

In **Figure 6**, we see the flow of a Start Subchannel (SSCH) instruction that is executed by a program running in LPAR 3 to initiate an I/O operation. The SSCH is issued to device DE01 represented by a subchannel whose SCB number (SCBNUM) is 101. LPAR 3 is configured to CSSID 2, IID 1. Once the LPAR hypervisor dispatches the LPAR containing the program executing the SSCH instruction to one of the CPs configured to LPAR 3, the CP executing the SSCH instruction determines the address of SCB 101 maintained in HSA by using the SCBNUM, the CSSID, and the IID. As previously stated, the program that executes the SSCH instruction specifies only the SCBNUM in the architected SID operand of the SSCH instruction, without awareness of the CSSID or IID to which the LPAR is configured. The SSCH instruction uses SCB lookup tables maintained in the HSA to locate the associated CSS image SCB that represents the architected subchannel for the target I/O device. Since the SCBs maintained in HSA are organized and "tagged" by CSSID, IID, and SCBNUM, access to the correct SCB is achieved with very little overhead.

Once state controls in the SCB are set to reflect the beginning of an SSCH instruction as was done prior to MCSS, the CP enqueues the SCB on one of the SAP work queues (WQ), signals the SAP to begin processing the SSCH request, sets the appropriate condition code to inform the program that the request has been initiated, and proceeds to end-of-instruction, releasing the processor to execute the next z990 instruction.

Once signaled, the SAP, operating within the CSS, dequeues an SCB from the top of the WQ. At this point, the SAP performs I/O path selection from among the channel paths connecting the device that is in the same CSS image and MIF image as the subchannel selected at the beginning of the I/O operation. In this case, channel paths identified by CHPIDs 07, 08, or 12 are candidates



CHPID	- Channel-path identifier
CP	- Central processor
CSSI	- CSS image
CSSID	- CSS-image identifier
CU	- Control unit
CULA	- CU logical address
DLA	- Destination logical address
Dlink@	- Destination link address
IID	- MIF image identifier
IQ	- Interrupt queue
SAP	- System-assist processor
SCB	- Subchannel control block
SCBNUM	- Subchannel number
SLA	- Source logical address
Slink@	- Source link address
SSCH	- Start Subchannel instruction
WQ	- Work queue

Figure 6

Flow of an I/O operation with MCSS.

that match these criteria. Once a channel path is selected on the basis of algorithms for optimizing channel and SAP utilization, the channel is signaled with information about the SCB associated with the target device. In this example, PCHID 101, which is spanned to CSSID 0 and 2 and defined as CHPID 07, is selected to perform the SSCH-specified I/O operations. The selected channel-path firmware uses the destination link address (Dlink@=0D), control unit logical address (CULA=0), and unit address (UA=03) to determine the SCBNUM from lookup tables built during HSA initialization. The CSSID and IID (2 and 1) are also passed to the channel when it is signaled by the SAP. The channel-path firmware accesses the SCB in HSA to determine and update the state of the I/O operation using CSSID 2, IID 1, and SCBNUM 101. To communicate with device DE01, the channel sets the CSSID.IID 2.1 into the SLA as previously described, along with the Slink@=0C, the DLA=CULA=0, and the Dlink@=0D in the link outbound frames sent on the interface.

When the control unit sends inbound link frames to the channel, it sets the parameters SLA=CULA=0, Slink@=0D, DLA=2.1, and Dlink@=0C. The channel-path firmware uses the DLA from the control unit to determine the CSSID.IID in addition to other information in the inbound frames to determine the SCB associated with the I/O operation. The channel path can then set the state of the I/O operation into the correct SCB. Once the

SCB is updated in HSA, the channel-path firmware signals the SAP and passes the related CSSID, IID, and SCBNUM to the SAP.

Once the SAP services the interrupt from the channel, it uses the CSSID, IID, and SCBNUM to locate the correct SCB. The CSS then verifies and updates the status of the I/O operation in the SCB and enqueues the SCB on an I/O interruption queue (IQ) for the particular interruption zone and subclass as indicated in the SCB. One of the CPs interrupts the program operating in the LPAR associated with the interruption zone when the program is enabled for that particular interrupt subclass. In this case, the program in LPAR 3 is interrupted.

The actual I/O processing within the CSS beyond the management of which SCB to pick from and which set of channel paths to choose is minimally affected with MCSS. This is a considerable advantage in that we have built MCSS on top of a proven, robust zSeries design.

Software support for the MCSS facility

I/O configuration definition methodology

There are two basic methodologies by which a server can obtain configuration information about the I/O resources that it can manage. The server operating system can either use a definition methodology, in which the I/O resources are explicitly defined, or it can use a discovery methodology to locate the resources that are either integrated into the host computer or attached via some type of transport technology (e.g., Fibre Channel). Many systems employ a hybrid approach in which the physical resources are discovered and certain policy information for usage is defined and applied to the discovered resources. For example, user-friendly names may be assigned such that they are predictable and repeatable, or when a choice of different device drivers is available, the installation may have to specify the one to use.

Defining the I/O configuration

zSeries systems with ESCON [6], FICON [7], or other I/O technologies employ a “definition methodology” for describing the I/O configuration to the processor and the operating system. This process allows the administrator to control and customize various features. User-friendly names can be associated with I/O resources for the purpose of access control, resource-based job scheduling, resource monitoring, and event reporting. Security policies can be enforced by limiting the resources accessible by the machine, the LPAR, and the operating system. Bandwidth can be managed by designating which host I/O adapters (channel paths) may be used to access which control units and I/O devices.

Configuration-definition methodology background

IBM zSeries servers and the control programs that operate on them require definitions of the system I/O configuration and interconnection topology in order to effectively and dynamically manage the execution of application-initiated I/O operations. Since the advent of the IBM 308X processor family in the early 1980s, the hardware system I/O configuration-definition process has been performed through the use of the I/O Configuration Program (IOCP). The corresponding software I/O configuration-definition process has evolved over the last 25 years. For the IBM multiple virtual storage (MVS) operating system, the I/O configuration-definition process required that the systems programmer use System/370* assembler macros to define the I/O resources for the software. These macros generated object code that was link-edited with the base control program of the operating system for subsequent loading into storage when the system was IPLed. This process, known as system generation (SYSGEN), became unacceptable over a period of time for a number of reasons. For example, whenever an I/O configuration change was required, the SYSGEN process had to be repeated. Correspondingly, large I/O configuration definitions took hours of processing power to build all of the object code needed to represent the I/O configuration and to link-edit this code with the operating system code. Any changes to the device support code and associated data structures required this entire cumbersome SYSGEN process to be repeated.

In the mid-1980s, the MVS operating system provided a new function, called the MVS Configuration Program (MVSCP), in order to define and segregate the software I/O configuration process from the base MVS control program code. This process separated the I/O configuration data structures and device support code into separate load modules. With the advent of the MVSCP, I/O device definitions were still described via S/370 XA assembler macros; however, these macros simply generated tables that were processed by device-specific exits that were called and executed from within the MVSCP. The output of the MVSCP program was a separate load module that provided separation of the operating system code from the data structures that represented the I/O configuration. This restructuring allowed customers to create multiple different MVS I/O configuration definitions and to select any one of these definitions during the IPL process. These configuration-definition enhancements eliminated the requirement to rebuild the entire operating system and reduced the configuration-creation process from hours to seconds. However, changes to the I/O configuration could still not be made dynamically while workloads were executing. All I/O configuration changes still required a power-on-reset (POR) of the entire system and thus affected all active work.

Although, for MVS, the hardware configuration-definition process was separate from the software configuration-definition process, both processes originally made use of a common input source of configuration statements. Accommodations were made in both the MVSCP and the IOCP programs to allow the inclusion of configuration information required by both. However, over time, this common hardware/software input source became more and more difficult to maintain. For example, the introduction of LPARing in the late 1980s and the functionality it introduced, such as the use of duplicate device numbers across LPARs, created the potential for configuration mismatches between the corresponding hardware and software configuration definitions. Further, such errors and mismatches were often not detected until there was an attempt to initiate an I/O operation to an incorrectly specified device, which might then fail. Recovery from such failures often required both the MVSCP and IOCP to be executed, a subsequent POR of the hardware, and an initial program load (IPL) of the operating system.

In 1990, an improvement to MVS called the Hardware Configuration Definition (HCD) [8] function was introduced. Its objective was to consolidate the hardware and software I/O configuration definition (IOCD) process into a single interactive end-user interface, and to address the problem of late detection of inconsistencies between the hardware and software configuration definitions. HCD provides an interactive panel-driven capability that supports both the hardware and software I/O configuration-definition functions, as shown in **Figure 7**. HCD validates all input against both hardware and software "rules" (unit information modules, or UIMs, are device-dependent exits that validate the user input) and detects any inconsistencies and errors. This allows an interactive user to make immediate corrections. Even with HCD, however, changes to the current hardware and software I/O configuration definitions still required a POR and a subsequent IPL.

Dynamic Reconfiguration Management (DRM), introduced with System/390* [9], built on the HCD functionality. With DRM, an HCD-created I/O definition file (IODF) could be used to change the I/O configuration definition without requiring the system outages previously associated with the configuration process (i.e., without a POR or IPL). At control program initialization, architected interfaces between the control program and the hardware allow the control program to determine whether its representation of the I/O configuration is consistent with the hardware I/O configuration definition initialized and active within the channel subsystem of the server. Once consistency has been verified, the current I/O configuration definition can be updated with a new

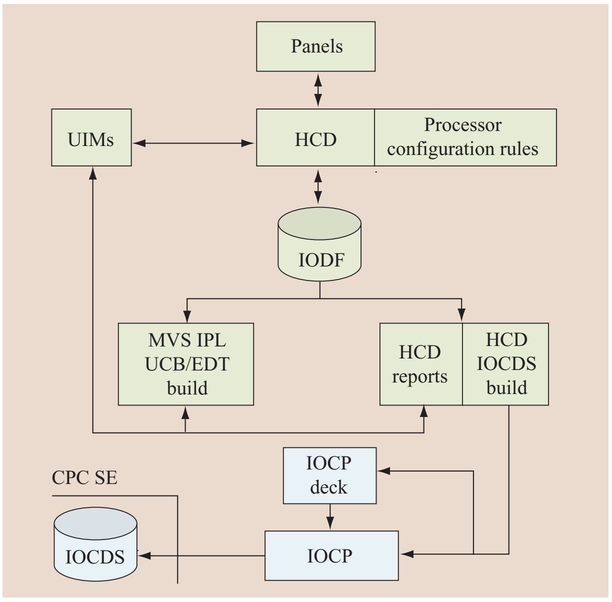


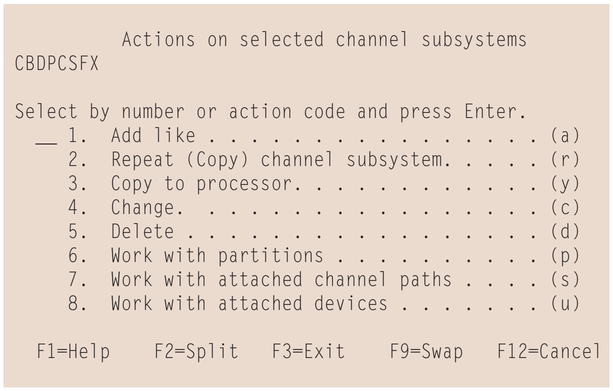
Figure 7

Output I/O definition file is used by IPL to build the software configuration. The IODF is also used to create the processor configuration. (UIM: unit information model.)

definition by the use of an HCD interactive panel, or by an MVS operator command, which invokes the DRM function. The control program determines the changes required to the existing definition (i.e., additions, deletions, and modifications) and makes the necessary changes to the software and, through architected interfaces, to the hardware-channel subsystem. Changes are synchronized with existing I/O activity to minimize and/or eliminate disruption. Additionally, the resultant hardware definition can be optionally written to a hardware I/O configuration data set (IOCDS) for use during subsequent system PORs.

The control program provides services that allow installation application programs and vendor products to be notified of a planned or completed configuration change. Such services are vital to applications that are sensitive to the I/O configuration (e.g., system automation for OS/390* I/O operations, or SAFOS IOOPs).

The DRM function became a single control point for defining the I/O configuration for the entire server and its associated LPARs. DRM, invoked in an LPAR running z/OS or z/VM, changes the hardware definition of I/O resources across all affected LPARs. It permits a single point of control for DRM related to hardware I/O definitions. In order to ensure that deleted resources do not affect currently executing applications within other LPARs, functions were provided to easily allow the



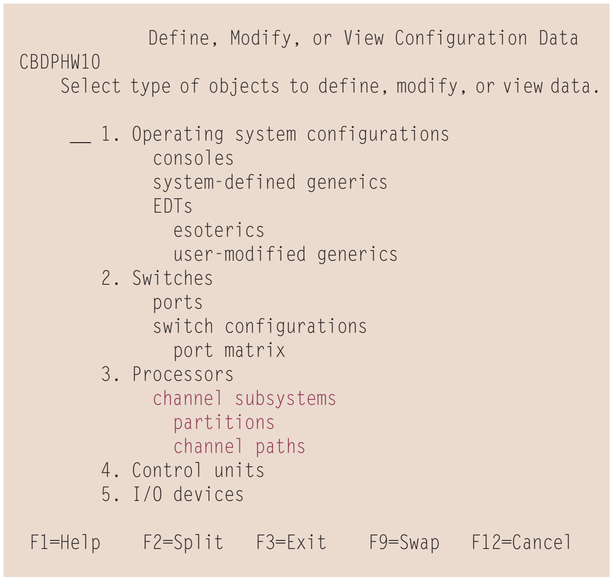
```
        Actions on selected channel subsystems
CBDPCSFX

Select by number or action code and press Enter.
 __ 1.  Add like . . . . . . . . . . . . . . . . (a)
    2.  Repeat (Copy) channel subsystem. . . . . (r)
    3.  Copy to processor. . . . . . . . . . . . (y)
    4.  Change.  . . . . . . . . . . . . . . . . (c)
    5.  Delete . . . . . . . . . . . . . . . . . (d)
    6.  Work with partitions . . . . . . . . . . (p)
    7.  Work with attached channel paths . . . . (s)
    8.  Work with attached devices . . . . . . . (u)

  F1=Help    F2=Split    F3=Exit    F9=Swap    F12=Cancel
```

Figure 8

HCD functions provide the ability to add, delete, modify, and copy the definition of a logical-channel subsystem.



```
          Define, Modify, or View Configuration Data
CBDPHW10
    Select type of objects to define, modify, or view data.

     __ 1. Operating system configurations
             consoles
             system-defined generics
             EDTs
               esoterics
               user-modified generics
        2. Switches
             ports
             switch configurations
               port matrix
        3. Processors
             channel subsystems
               partitions
               channel paths
        4. Control units
        5. I/O devices

 F1=Help    F2=Split    F3=Exit    F9=Swap    F12=Cancel
```

Figure 9

The definition of processors is extended with MCSS to include the definition of one or more logical-channel subsystems. Each of the logical-channel subsystems can have up to fifteen LPARs and 256 CHPIDs.

installation to coordinate the planned I/O configuration change across all affected LPARs before making the change.

Information systems perform a critical function within business enterprises. For many businesses, operation twenty-four hours a day, seven days a week (24×7) is a requirement. Time zone differences across international operations are often an important factor. Outages and the duration of outages can easily be correlated to loss of productivity and/or to lost revenue. The DRM function allowed customers to make significant improvements in continuous availability by eliminating planned outages for I/O configuration changes.

Elements of a zSeries I/O configuration

The zSeries I/O configuration consists of the following elements:

1. *Processor* – Each processor definition has an implicit channel subsystem definition with an architected limit of 256 channels. The specific processor type determines the limits and capabilities of the channel subsystem, such as the number of total channels that can be supported and the features that are supported (e.g., HiperSockets* [10]).
2. *Logical partitions* – Each processor can be divided into up to fifteen local LPARs. Certain physical channels can be actively shared by the different partitions at the same time, increasing the effective utilization of the shared resources.
3. *Channel paths* – These are either the physical hardware adapters used to communicate with external peripheral devices or logical entities used for communication between LPARs for networking or clustering. Since the physical ESCON and FICON channels can be used for multiple purposes (i.e., ESCON CTC vs. ESCON channel, or FICON native channel vs. FICON bridge channel vs. FCP), customization information is needed to define the behavior that is required.
4. *Control units* – Control unit definitions were originally required in order for the channel subsystem to understand the scope of devices attached to the control units for the purpose of managing busy/no-longer-busy status indications. With ESCON and FICON, the control unit definitions are necessary for the purpose of managing addressing and the establishment of logical paths [6, 7].
5. *Devices* – The I/O device is the target of the z/Architecture I/O commands.
6. *Switches* – Switches have been introduced into the defined I/O configuration in order to integrate the path-management and reconfiguration functions of the sharing hosts with the switch-management functions required to manage the I/O fabric.

Figures 8 and **9** show the HCD user interface for managing the I/O configuration definition.

z990 configuration definition extensions

The definition methodology and system management for host configurations has been extended for the z990 with a set of tools consistent with the pre-existing zSeries configuration tools and concepts. This allows customers to exploit the new MCSS features of the z990 with minimal additional work and to maintain continuous availability by extending the dynamic I/O reconfiguration management [9] function of zSeries to include the MCSS features of the z990.

HCD support for MCSS consists of the following functions:

1. HCD provides the ability to define multiple-logical-channel subsystems for a processor. For simplicity of migration, this includes the ability to copy a physical-channel-subsystem definition from another processor definition into a logical-channel-subsystem definition and the ability to copy a logical-channel-subsystem definition into another logical-channel subsystem. Each logical-channel subsystem includes up to fifteen logical LPARs. Certain channel types, such as FICON and HiperSockets, are sharable across logical-channel subsystems. Channels defined this way are denoted as spanned.
2. HCD participates with the channel mapping tool (CMT) to automatically assign the best physical channel for a given control unit configuration. The CMT is a function that assigns a PCHID to each CHPID definition in the processor configuration.
3. HCD automatically reassigns the definition of FICON CTC control units when migrating a configuration into an MCSS configuration. This is needed because each FICON CTC control unit definition must include the channel subsystem in which it resides. Those processor definitions in the same IODF using FICON CTC must have the definitions updated if the target control unit becomes part of an MCSS definition.
4. HCD automatically detects the definition of channel types that are not supported when copying a channel subsystem definition. These unsupported CHPID types are not included in the new configuration.

Autonomic computing

Over the years, zSeries architecture has been enhanced with a number of features that have allowed customers to efficiently manage their systems and exploit the availability, scaling, and workload-management capabilities of z/OS [11, 12] to efficiently run multiple workloads at the same time. For I/O, these zSeries features include the extended channel-path-measurement facility (CPMF), for gathering performance data on channel-path resources [13], the channel monitoring mode, which allows the creation of channel measurement blocks (CMBs) that gather I/O resource usage and contention statistics to individual device granularity [1], plus a number of other zSeries machine facilities that allow resource monitoring products such as the IBM resource monitoring facility (RMF) product to provide detailed reporting on other I/O statistics and resource contention for capacity planning and problem analysis. These I/O facilities also provide the customer with the ability to do accurate accounting and billing of applications for the consumption of I/O resources.

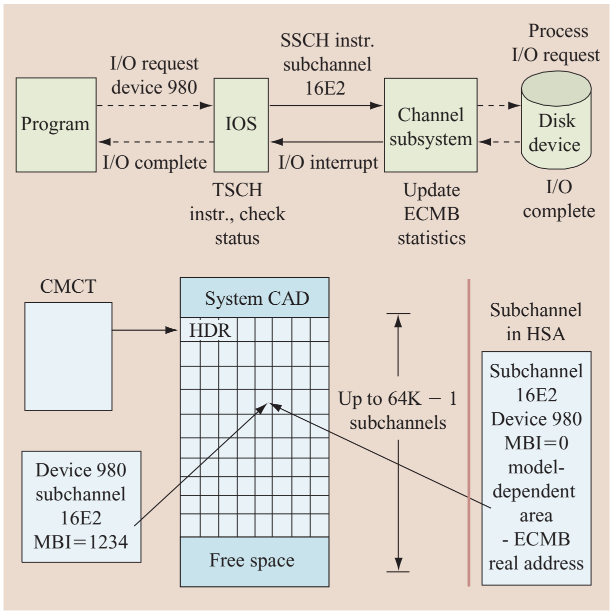
The z/OS workload manager (WLM) component is constantly evolving to exploit the zSeries I/O instrumentation data. It is the objective of z/OS to create systems that are self-tuning and require fewer specialized skills in order to plan and configure, and to maximize the efficient utilization of all of the I/O resources to provide the maximum business value to the customer.

With WLM, the z/OS operating system provides the capability to prioritize I/O requests. The objective of this support was to introduce sysplex-wide, goal-oriented management of I/O priorities, driven by WLM knowledge of goals for work and the business importance of those goals [14]. The I/O priority for each request is divorced from the dispatching priority associated with the requesting dispatchable unit of work in order to allow independent algorithmic adjustments. This requires that

1. I/O priorities be dynamically determined on the basis of the business value of each I/O to the customer.
2. I/O priorities be maintained at a WLM service class period level and be synchronized across the systems in a Parallel Sysplex*.
3. The I/O priority associated with each I/O request be passed to the disk storage systems (for example, the enterprise storage server, or "Shark") for efficient management of the resources it controls outboard.
4. Monitoring and reporting extensions be provided to externalize relevant performance data.

In OS/390 Version 2 Release 7, while operating in WLM goal mode, the systems programmer can choose to let WLM dynamically manage the parallel access volume (PAV) aliases. Dynamic management of PAVs means that WLM determines which workloads in the sysplex are not meeting their goals because of IOS queue time delays. With this information, WLM optimizes which PAV-based UCB devices should have PAV-alias devices added to them in order to increase the I/O parallelism and eliminate IOS queuing delays [15].

Having WLM manage the PAV-alias assignments provides a number of benefits. The systems programmer does not have to perform detailed analysis of where to place data sets and where to assign aliases. Workloads are constantly changing, and the systems programmer can at best only statically configure for an average access pattern for the control unit. However, the average access pattern and I/O rate are almost always suboptimal. With WLM dynamic alias tuning, data does not have to be moved



I/O request device 980
SSCH instr. subchannel 16E2
Process I/O request
Program
IOS
Channel subsystem
Disk device
I/O complete
I/O interrupt
TSCH instr., check status
Update ECMB statistics
I/O complete
CMCT
System CAD
HDR
Free space
Up to 64K − 1 subchannels
Subchannel in HSA
Subchannel 16E2 Device 980 MBI=0 model-dependent area - ECMB real address
Device 980 subchannel 16E2 MBI=1234

Figure 10

z/OS support for extended channel-measurement blocks.

around to avoid hot spots. This improves application availability and reduces management costs. All that the systems programmer need do is to assign enough PAV aliases in the control unit to meet the overall aggregate workload requirements. WLM can adjust the PAV alias requirements as the workload changes.

WLM dynamic alias tuning helps to reduce hardware costs because dynamically moving PAV aliases around decreases the average response time to the control unit. This has an effect equivalent to increasing the cache size of the control unit, without the added hardware cost. In other words, this has the same effect as spreading the data across several devices, without taking the time and resources to move the data, finding a means to accurately predict how to spread the data for best response time, or obtaining enough devices to distribute the I/O.

Allowing OS/390 to manage the PAVs facilitates larger volume sizes. Multiple I/O requests arriving for the same logical volume do not necessarily block one another from executing. Having larger volume sizes has the added benefit of reducing virtual storage constraints below the 16-MB storage line. Instead of running with three 3390-3 DASD volumes, each with a UCB below the line, the installation can choose to define a single 3390-9 volume with the PAV base still below the 16-MB line and the PAV-alias devices in 31-bit storage. The PAV aliases are invisible to the applications but continue to provide the I/O parallelism achieved by three separate devices.

Finally, with WLM dynamic alias tuning, the installation does not have to dedicate as many unit addresses (subchannels) in the LSS for use as PAV aliases. Thus, more data can be addressed in a single LSS by using more of the available unit addresses to define PAV-based devices. The smaller number of PAV aliases can be adjusted as required by the workload.

I/O measurements

The zSeries I/O measurements constitute a critical function that allows the z/OS WLM component to monitor the I/O delays in order to determine whether or not I/O contention is causing work to miss its goals. A number of enhancements were made to MCSS to improve the facilities that provide measurements in order to increase the ability to provide autonomic computing capabilities.

Prior to MCSS, the original S/370 XA I/O architecture [2] defined measurement blocks to allow the operating system to gather detailed measurements about the execution of I/O requests. This architecture had several deficiencies. First, it required that all of the measurement blocks be located in contiguous real memory. A typical customer configuration required thousands of devices, and each measurement block was 32 bytes in length. It was not practical for the operating system to allocate huge amounts of contiguous real storage after program initialization. Thus, the customer had to pre-allocate all of the measurement blocks that could possibly be wanted over the life of the IPL. This would waste system resources until the time at which the blocks were needed. If the customer did not plan correctly, new devices that were dynamically added to the system could not have measurements gathered.

Second, the measurement architecture was optimized for monitoring programs to sample the measurement blocks directly at fixed intervals in order to report average utilization and delays. If the program had to determine all of the measurements for each specific I/O request, it would have to take a snapshot of the measurement blocks before and after each I/O request to determine the contributions of each operation. Since DASD devices typically have many datasets and I/O activity for many service classes, WLM requires an efficient way to gather the measurements for each individual I/O operation so that it can be aggregated back to the service class.

New technologies placed new requirements on the measurements. As the I/O capabilities increased, some of the 16-bit counters were no longer sufficient to prevent wrapping twice in a typical measurement interval. Timer granularity down to 128 microseconds was no longer sufficient to maintain precise totals. New transport architectures such as FICON required additional information to better provide for capacity planning and problem determination.

To address these problems, two new features were added to the z/Architecture for measurements: the extended channel-measurement blocks (ECMBs) and the extended measurement word (EMW).

Extended channel-measurement blocks

Figure 10 shows the operating system data structures in support of the ECMBs. The ECMBs are created in a data space in contiguous virtual storage. The software control block for the device (UCB) contains an index into the contiguous virtual storage for identifying the device. Real storage is allocated to back the virtual storage as it is needed. Using a data space to hold ECMBs, instead of using common fixed storage as with the original CMB support, provides important constraint relief for common fixed storage.

The extended channel-measurement blocks double the size of the space available for device measurements from 32 to 64 bytes, allowing room for new functions and measures. However, the most significant advantage of the new architecture is to provide for a full doubleword pointer in the subchannel to its corresponding measurement block. This 64-bit pointer is set to contain the real address of the measurement block for the device allocated by the operating system. With a full doubleword pointer, the operating system no longer has to guarantee that the measurement blocks reside in contiguous real storage. Instead, z/OS allocates them in a common area data space. This means that the measurement blocks can reside in contiguous virtual storage and are backed by 4K real pages as needed. Placing the measurement blocks in a data space instead of common storage frees up to two megabytes of common storage.

Figure 11 shows the format of each ECMB entry. ECMBs are 64 bytes in length. They expand the 16-bit counters for counting I/O operations from S/370 XA I/O architecture into 32-bit counters. Two new fields are added in support of multiple-channel subsystems, a 32-bit aggregate time for device busy and a 32-bit time for aggregating the total command response time for FICON operations. The subchannel is modified to eliminate the device-model-dependent times and instead provide a 64-bit pointer to the measurement block in real storage.

Extended measurement word (EMW)

In z/Architecture, I/O interrupts are reported to the operating system via an interruption response block (IRB) (**Figure 12**). The IRB contains the status for each I/O operation. When the I/O operation is complete, the IRB contains the contribution of that operation to the ECMBs shown above. This eliminates the overhead for having to store the ECMB measurements from the last I/O operation(s) and to compute the incremental difference with the ECMB measurements of the currently completed I/O operation(s). This calculation would be even more significant with the ECMBs described above as opposed to the CMBs because of the significant additional overhead to locate ECMB blocks in memory.

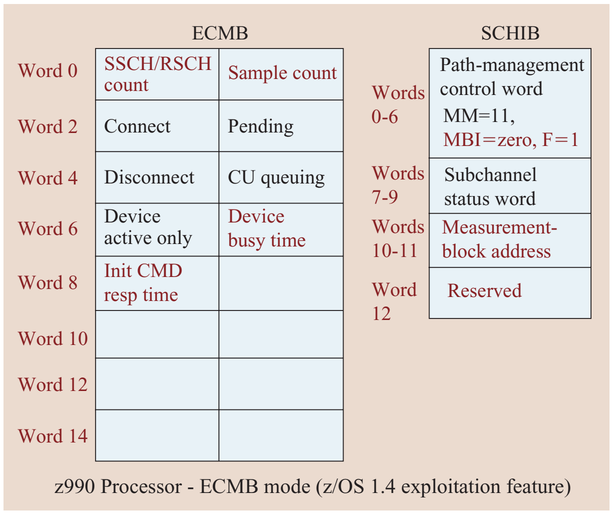


Figure 11

Extended channel-measurement block: CMB/SCHIB changes.

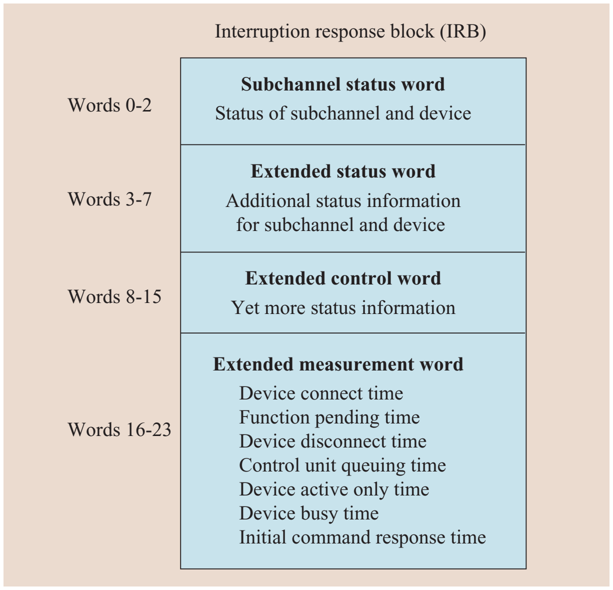

Interruption response block (IRB)
Words 0-2
Subchannel status word
Status of subchannel and device
Words 3-7
Extended status word
Additional status information for subchannel and device
Words 8-15
Extended control word
Yet more status information
Words 16-23
Extended measurement word
Device connect time
Function pending time
Device disconnect time
Control unit queuing time
Device active only time
Device busy time
Initial command response time

Figure 12

The extended measurement word (EMW) of the interruption response block (IRB) provides the measurement data for the I/O operation that has completed.

Conclusions

The z990 MCSS facility increases I/O scalability and compatibility of the zSeries I/O topology and infrastructure necessary for continued growth of the zSeries server platform in the high-end server marketplace. While the nature of the MCSS facility described in this paper is, of necessity, specific to z/Architecture-based systems and their existing I/O technology, the concepts of addressing indirection, associated replication of control information, and virtualization transparency used to achieve the necessary I/O scaling and sharing of the various channel, adapter, and I/O device components that comprise the z990 I/O system can be applied to any I/O model regardless of whether such a model is implemented in hardware, firmware, software, or combinations of these system components.

Appendix: Acronyms as used in this paper

CAD	Computer-Aided Design
CP	Central Processor
CPC	Central Processing Complex
CPI	Channel-Path Image
CPS	Channel-Path Set
CHPID	Channel-Path Identifier
CMB	Channel Measurement Block
CMCT	Channel Measurement Control Table
CMT	Channel Management Tool
CPMT	Channel-Path Measurement Tool
CSS	Channel Subsystem
CSSID	Channel-Subsystem Identifier
CTC	Channel-to-Channel
CULA	Control Unit Logical Address
DASD	Direct Access Storage Device
DLA	Destination Logical Address
DLINK	Destination Link Address
DRM	Dynamic Reconfiguration Management
ECMB	Extended Channel Measurement Block
EMW	Extended Measurement Word
ESCON	Enterprise Serial Connection
FCP	Fibre Channel Protocol
FICON	Fiber Channel Connection
MIF	Multiple-Image Facility
HCD	Hardware Configuration Definition
HSA	Hardware System Area
IID	Multiple-Image Facility Identifier
IOCDS	I/O Configuration Definition Data Set
IOCP	I/O Configuration Program
IODF	I/O Definition File
IOS	I/O Supervisor
IPL	Initial Program Load
IQ	Interruption Queue
LPAR	Logical Partition
LSS	Local Storage Subsystem
MBA	Memory Bus Adapter
MBI	Measurement Block Index
MCSS	Multiple-Logical-Channel Subsystems
MM	Monitoring Mode
MVS	Multiple Virtual Storage
MVSCP	Multiple Virtual Storage Configuration Program
OS/390	Operating System for S/390 Architecture Systems
PAV	Parallel Access Volume
PCHID	Physical Channel Path Identifier
PN	Partition Number
RMF	Resource Monitoring Facility
POR	Power-On Reset
RCHP	Reset Channel Path Instruction
RSCH	Resume Subchannel Instruction
SAFOS	System Automation For OS/390
SAP	System Assist Processor
SCB	Subchannel Control Block
SCBNUM	Subchannel Control Block Number
SCHIB	Subchannel Information Block
SE	Service Element
SID	Subsystem-Identification Word
SIE	Start Interpretive Execution
SLA	Source Logical Address
SLINK	Source Link Address
SSCH	Start Subchannel
SYSGEN	System Generation
S/370 XA	System/370 Extended Architecture
TPF	Transaction Processing Facility
TSCH	Test Subchannel
UA	Unit Address
UCB	Unit Control Block
UIM	Unit Information Module
WLM	Work Load Manager
WQ	Work Queue
VSE/ESA	Virtual System Extensions for the Extended Systems Architecture Operating System
z/OS	zSeries Operating System
z/VM	zSeries Virtual Machine Operating System

* Trademark or registered trademark of International Business Machines Corporation.

References

1. IBM Corporation, *z/Architecture Principles of Operation* (SA22-7832); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
2. IBM Corporation, *System/370 Extended Architecture: Principles of Operation* (SA22-7085); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
3. IBM Corporation, *IBM eServer zSeries 990 Technical Introduction* (SG24-6863), 2003; see *http://*

www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/.
4. IBM Corporation, *Enterprise Systems Architecture/390 Principles of Operation* (SA22-7201); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
5. IBM Corporation, *eServer zSeries Processor Resource/System Manager Planning Guide* (SB10-7033); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
6. IBM Corporation, *IBM Enterprise Systems Architecture/390 ESCON I/O Interface* (SA22-7202); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
7. INCITS, *Fibre Channel Single Byte Command Code Sets-2 Mapping Protocol* (FC-SB-2), T11/Project 1357-D/Rev 2.1 (December 2000); see *http://t11.org/index.htm* (drafts).
8. IBM Corporation, *z/OS HCD User's Guide* (SC33-7988); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
9. R. Cwiakala, J. D. Haggar, and H. M. Yudenfriend, "MVS Dynamic Reconfiguration Management," *IBM J. Res. & Dev.* **36,** No. 4, 633–646 (July 1992).
10. M. E. Baskey, M. Eder, D. A. Elko, B. H. Ratcliff, and D. W. Schmidt, "zSeries Features for Optimized Sockets-Based Messaging: HiperSockets and OSA-Express," *IBM J. Res. & Dev.* **46,** No. 4/5, 475–485 (July/September 2002).
11. J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive Algorithms for Managing a Distributed Data Processing Workload," *IBM Syst. J.* **36,** No. 2, 242–283 (1997).
12. IBM Corporation, *Workload Manager/System Resources Manager for OS/390*; see *http://www.s390.ibm.com/wlm/*.
13. IBM Corporation, *z/OS RMF User's Guide* (SC33-7990); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
14. W. J. Rooney, J. P. Kubala, J. Maergner, and P. B. Yocom, "Intelligent Resource Director," *IBM J. Res. & Dev.* **46,** No. 4/5, 567–586 (July/September 2002).
15. A. S. Meritt, J. A. Staubi, K. M. Trowell, G. Whistance, and H. M. Yudenfriend, "z/OS Support for the IBM TotalStorage Enterprise Storage Server," *IBM Syst. J.* **42,** No. 2, 280–301 (2003).

Received August 25, 2003; accepted for publication February 2, 2004; Internet publication May 27, 2004

Les W. Wyman *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (lwyman@us.ibm.com).* Mr. Wyman joined IBM in 1965, retired in 1993 as a Senior Technical Staff Member, and rejoined IBM in 1999. He has held numerous technical and technical leadership positions in programming systems: control program design, development, and testing of the SSS, MVT, and MVS operating systems for the IBM S/360, S/370, and S/370 XA systems; channel engineering: design and development of the 308X channel subsystem; and systems architecture: the S/370 XA channel-subsystem architecture, S/390 channel-subsystem architecture including the ESCON Multiple Image Facility, the initial S/390 logical partitioning, and VM high-performance virtual machine architectures, the zSeries QDIO and MCSS architectures, and others. Mr. Wyman has achieved the seventh invention plateau and has received a number of technical achievement awards.

Harry M. Yudenfriend *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (harryy@us.ibm.com).* Mr. Yudenfriend is a Distinguished Engineer. He joined IBM in 1980 after receiving his B.S. degree in computer science from the Columbia University School of Engineering and Applied Science. He has worked in the z/OS software and design area for his entire IBM career, involved in many programming projects as a developer, tester, development team leader, and designer. Mr. Yudenfriend has achieved his sixteenth invention plateau and was named an IBM Master Inventor in December 2001.

John S. Trotter *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (trotterj@us.ibm.com).* Mr. Trotter is a Senior Software Engineer. He joined IBM in 1977 after receiving his B.S. degree in electrical engineering from the Polytechnic University. He has worked in both the hardware and software laboratories on many projects as a microcode and software developer, tester, development team leader, and designer. Mr. Trotter has received several awards and patents, including an IBM Outstanding Technical Achievement Award for the development and testing of MIF.

Kenneth J. Oakes *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (oakes@us.ibm.com).* Mr. Oakes is a Senior Technical Staff Member in the eServer I/O Development Group. He joined IBM in 1977 after receiving his B.S. degree in electrical engineering from the University of New Haven. Mr. Oakes has held various technical positions in the eServer I/O design area. He holds numerous patents relating to channel and I/O subsystem design and has achieved the fourth invention plateau. He has received seven other formal awards, including four IBM Outstanding Technical Achievement Awards and three IBM Outstanding Innovation Awards. Mr. Oakes is currently involved in I/O subsystem design for the next-generation eServer.