Processor subsystem interconnect architecture for a large symmetric multiprocessing system

P. Mak
G. E. Strait
M. A. Blake
K. W. Kark
V. K. Papazova
A. E. Seigler
G. A. Van Huben
L. Wang
G. C. Wellwood

Integral to the significant capacity growth of the IBM eServer™ z990 (the eighth-generation zSeries® CMOS-based server) from its predecessor z900 system is the interconnect architecture, which tightly couples 48 customer CPUs in the system. A major attribute of this architecture is a new "hot swap" feature which improves zSeries system availability for customers by permitting the substitution or addition of a field-replaceable unit (FRU) in the processor subsystem without requiring the system to be powered down. The novel two-level interconnect architecture contains a distributed switch which connects up to four processor-memory nodes in book packages. The book packages, which are also FRUs, are connected in a dual concentric ring topology at the second-level (L2) interconnect. This architecture also contains an integrated 32-MB L2 cache and central switch connecting up to eight dual-core processor chips in a star topology at the first-level interconnect inside one of these nodes. This paper describes the bus protocol on the second-level interconnect, the cache coherency management throughout the storage hierarchy, and the ring topology reconfiguration for hot swap. Also described is a memory power management scheme to support the power demand from the 48 CPUs and up to 256 GB of memory.

Introduction

The z990 is the latest in the zSeries* line of enterprise servers. The z990 shares much with its predecessors, but also introduces significant advances. Some of the most significant advances are made possible by a new system package.

The z990 system comprises one to four book packages; each book package is a pluggable unit containing up to 12 processors and up to 64 GB of memory, I/O adapters, and a centralized switch and coherency manager known as the system control element (SCE), through which the processors and I/O connect. The SCE includes a second-level cache (L2 cache) and a pipelined switch that manages data routing and maintains strong storage coherency across the multiprocessing system. The shared L2 cache is interposed in the storage hierarchy between a private L1 cache dedicated to each processor and the fully shared, fully coherent memory of the z990. The minimum system configuration consists of one book; additional books may be plugged into a system and configured online without stopping the previously installed books, until the system reaches its maximum capacity of four books.

0018-8646/04/$5.00 © 2004 IBM

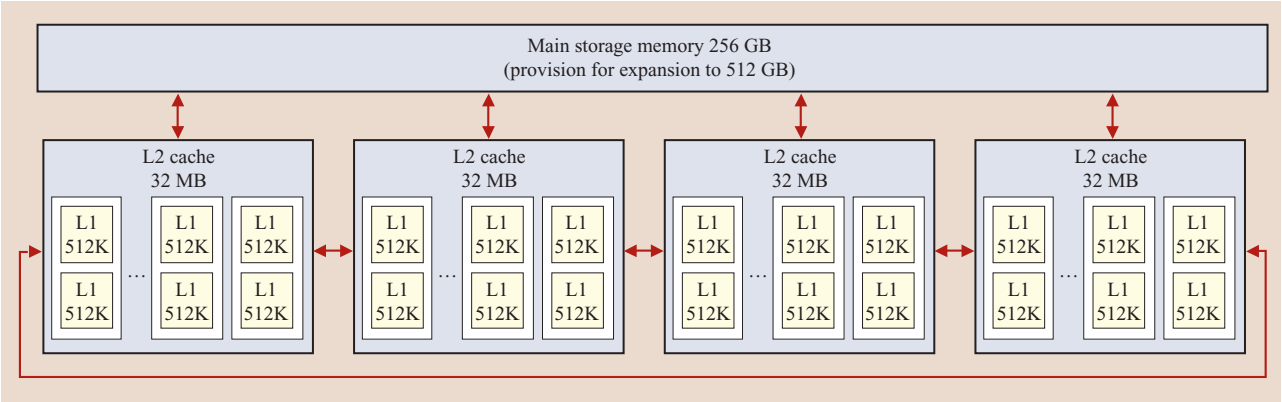

Figure 1

IBM zSeries 990 storage hierarchy.

Figure 1 illustrates the system storage hierarchy. Main storage memory consists of one contiguous address space, physically located in processor memory arrays (PMAs) spread across the installed books. Each storage address corresponds to one physical location in a PMA on one of the books. Each book contains a shared 32-MB L2 cache that is used by all of the processors and I/O on the book. The contents of memory may be cached in one or multiple books. The processors on each node contain their own 512-KB L1 caches, represented in the figure in groups of two corresponding to the packaging of two processor cores on one chip. Each book contains up to twelve processors, though only six are shown in the figure; the ellipsis represents additional instances. The contents of the L2 cache may also be held in one or more L1 caches on the book. The L2 cache maintains a full subset rule for all of the L1s; i.e., it contains a copy of the data stored in each of the L1s on the book. The L2 cache may contain additional data that is not in any L1 cache because of the removal of data from the L1 cache by the least recently used (LRU) replacement algorithm. This may happen because of pre-fetching of additional data from memory beyond what is requested by a processor or use of the L2 cache by I/O devices (data is not cached within z990 I/O devices, but may be held in the L2 cache for use by I/O devices). This L2 full subset rule is represented in the figure by showing the L1 caches contained within the L2 space. All changes made by processors to their respective L1 caches are immediately duplicated to the shared L2 cache. This cache structure continues the evolution of prior binodal zSeries systems which utilized dual shared L2 caches [1, 2] to up to four shared L2 caches in the z990 that define new intervention master (IM) and multicopy (MC) states extending the previous modified, exclusive, shared, invalid (MESI) cache management scheme.

The single largest advance in the z990 is the increased capacity made possible by the four-book ring-connected system structure, as compared with the binodal structure of the predecessor z900 and G6 [3]. Not only does this structure allow for more processors, memory, and I/O than before; it also, for the first time, allows additional books, containing additional processors, cache, memory, and I/O ports, to be hot-plugged and configured online into an operating system, a capability previously supported only for hot-plugging of I/O cards and for enabling processors and memory that were already installed but not configured online [4]. **Table 1** summarizes some of the z990 advances in comparison with prior zSeries models. This paper describes design features that make the z990 system possible.

Ring topology

The z990 system comprises one to four books. Each book is a pluggable unit containing up to 12 processors with up to 64 GB of memory (with provisions for future expansion to 16 processors and 128 GB of memory), I/O adapters, and a system control element (SCE) which connects these other elements. The SCE within each book contains a 32-MB L2 cache which serves as the central coherency point for that particular book. Both the L2 cache and the main memory are accessible by a processor or I/O adapter within that book or any of the other three books in the system.

Figure 2 illustrates the contents of the z990 book package. The heart of the book package is a multichip ceramic module (MCM). This module contains up to eight dual-core processor chips; it also holds the SCC and SCD chips that include the controls (on SCC) and data (on

Table 1 Comparison of IBM eServer* zSeries models.

Feature	*G6*	*z900*	*z990*
System structure	Two-node (point-to-point connected)	Two-node (point-to-point connected)	Four-book† (ring connected)
Processors	Seven per node (14 total)	Ten per node (20 total)	Twelve per book (48 total)
Design target clock speed	Processor: 667 MHz; Level 2 cache and central switch: 333 MHz	Processor: 1 GHz; Level 2 cache and central switch: 500 MHz	Processor: 1.25 GHz; Level 2 cache and central switch: 625 MHz
Concurrent book upgradable	No	No	Yes
L2 cache management	L2 directory	L2 directory	L2 directory Memory-coherent directory
Memory DRAM speed	Synchronous 1/4 processor (6 ns)	Synchronous 1/8 processor (8 ns)	Asynchronous (8 ns)
Memory technology	SDRAM (synchronous dynamic random-access memory)	SDRAM	DDR SDRAM (double-data-rate SDRAM)
Key cache	No	For I/O accesses only	For I/O and processor accesses
Average L2 cache capacity per processor core	1.15 MB	1.6 MB	2.0 MB
Memory buses	4 × 16 bytes per node	4 × 16 bytes per node	2 × 32 bytes per book

†A book is the pluggable equivalent of the node found in earlier systems.

SCD) for the L2 cache, and ports for connecting to the processors, main storage controller (MSC) chips, and the ring interface with other books. The MCM also contains a clock chip to provide clock signals to the book. In addition, the book contains three I/O memory bus adapter (MBA) chips that provide I/O connectivity through self-timed interface (STI) ports, and two memory cards that are described in greater detail in a later section.

The SCE within each book contains two ring ports, each of which comprises one incoming and one outgoing interface, allowing the book to be connected to corresponding ports in up to two other books via a dual concentric ring. Each ring port consists of an incoming and an outgoing address/command/response bus, and incoming and outgoing data buses. This is illustrated in **Figure 3(a)**, which shows a system populated with four books. The books are numbered from 0 to 3 in the order in which they are installed, from the smallest to the largest configuration. In this figure, showing a fully configured system, book 0 is directly connected via unidirectional buses to two other books (2 and 3). Books 2 and 3 are in turn connected to Book 1, completing two separate rings running in opposite directions, as illustrated by the arrows. The ring that flows in the clockwise direction is referred to as Ring 0, while the ring that flows in the counterclockwise direction is referred to as Ring 1. The SCEs exchange address, command, and response information and data over these ring interfaces.

Since each book connects directly to up to two other books without having to go through an external switch element, the book becomes the only pluggable component with switching silicon devices. A passive backplane with only embedded copper wiring for the ring buses is far less prone to fail than a backplane with active switching silicon devices.

As shown in **Figures 3(b)** and **3(c)**, a system consisting of fewer than four books can also have its books connected via a dual concentric ring topology using the same backplane designed for four-book systems. This is done by replacing the missing book(s) with passive jumper card(s) to electrically bridge the two sets of embedded wires on the backplane in order to maintain the flow of data through Ring 0 and Ring 1.

One of the advantages of this topology over other ring-based or switch-based topologies is improved data intervention latency in cases in which a processor fetch operation is targeting data located in the L2 cache or memory of another book. The required coherency interrogation command and address are broadcast from the home, or local, book of the processor on both rings simultaneously in order to expedite the L2 cache coherency interrogation required on each of the other, or remote, books. As each book performs the interrogation of its own L2 directory, it compares each locally generated response with an incoming response that it received on the same ring bus as the interrogation command and address.

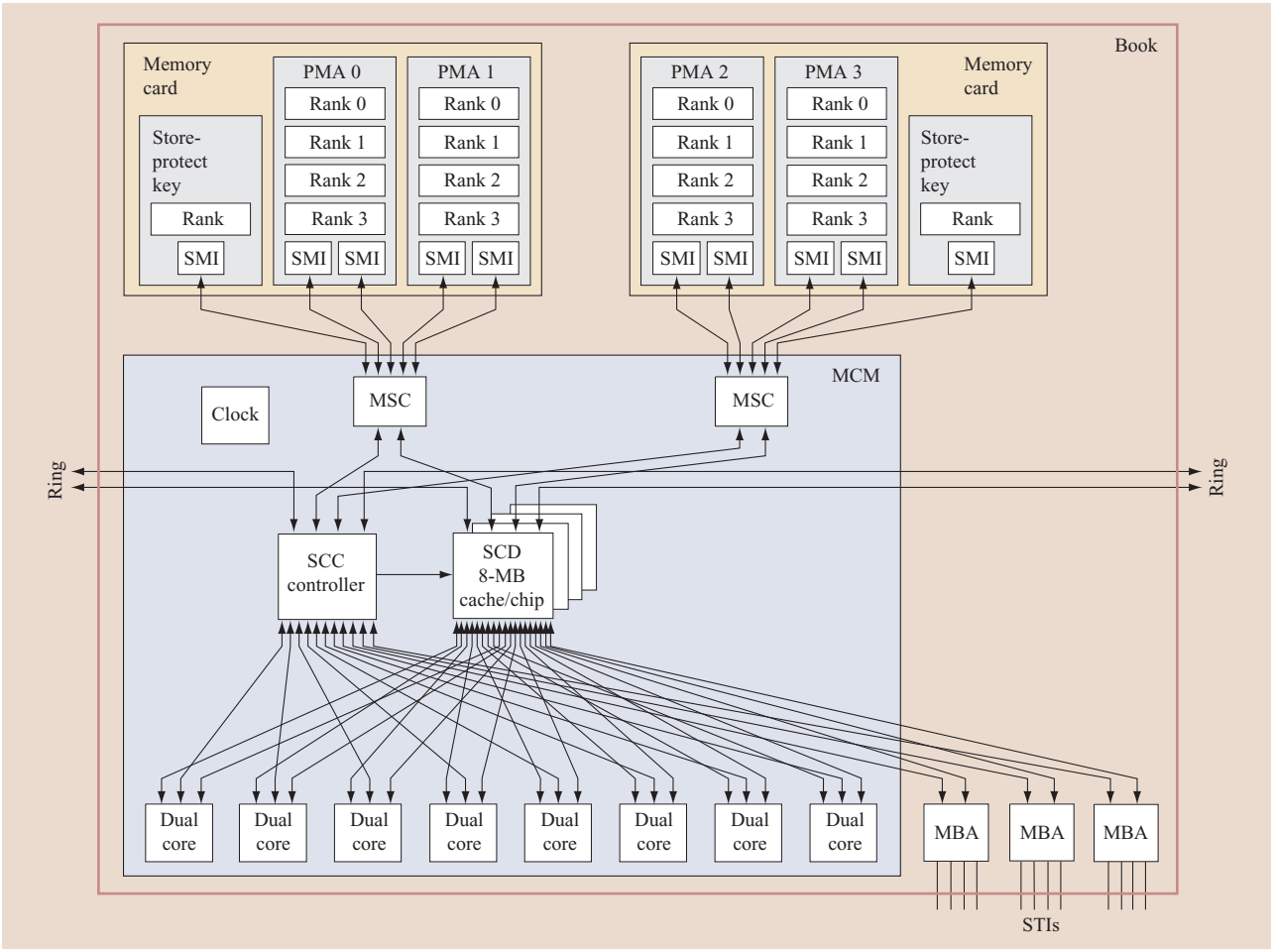

Figure 2

Book structure of the z990. (Rank: a segment of memory.)

The results of the comparison are used to determine what response to forward on the outgoing ring bus. Note that this response comparison and generation are performed twice by each book, once per ring. The response forwarded by a book on the first ring on which it receives the interrogation command contains only status from coherency interrogations on books that have already received the coherency interrogation command on this ring. This is referred to as an early or first response. The response forwarded by the book on the second ring on which it receives the interrogation command contains status regarding coherency interrogations performed by all books in the system and is referred to as a final or second response. The returning fetch data is transmitted over only one of the rings, which is selected on the basis of the relative positions of the source and destination books so that the shortest path is chosen. In the case of data being returned from a remote L2 cache, the data is returned as quickly as possible as part of a special intermediate data response. For the remote L2 data access case, the access latency for this topology from coherency interrogation launch to data return is an average of 2.67 book-to-book crossings, or "book hops." In addition to helping speed up data returns, having two sets of data buses also improves overall data bandwidth.

The other significant advantage of the dual-ring topology is the relative ease with which it supports concurrent book repair (e.g., the ability to replace a defective book in the system while the other books continue to process normal workloads) and concurrent update (e.g., the ability to add a new book to the system to upgrade its total capacity while the other books continue to process normal workloads with degraded system resources).

To perform concurrent book upgrade, the two rings that are fully intact, or closed, by means of a jumper card or

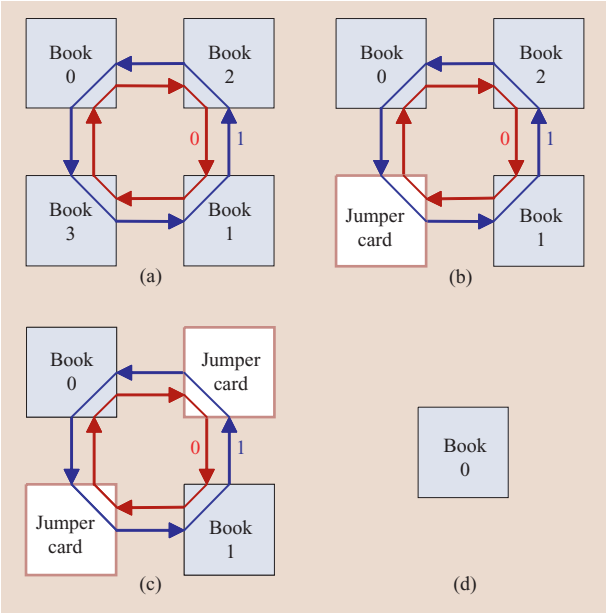

Figure 3

z990 system ring configurations: (a) Four books; (b) three books; (c) two books; (d) one book.

cards, as in Figures 3(b) and 3(c), are temporarily "opened" while a jumper card is being replaced, and then reclosed upon completion of a book substitution. While a jumper card is being replaced, the two open ring ends are dynamically bridged across those books with the severed connection; this forms one folded unidirectional ring to provide the pathways required for interbook communication. This upgrade sequence is illustrated in **Figure 4**. In the case of Figure 4, which initially contains two jumper cards as in Figure 3(c), the remaining jumper card continues to carry traffic between the two active nodes while the other jumper card is being replaced.

Coherency and ring protocol

The ring topology embodies a novel bus protocol and storage coherency management system which allows a common design to support a multitude of system configurations ranging from two to four books (one-book configurations are also supported, with no ring connections used). All ring operations begin with a simultaneous request launch on both rings of the fabric in closed-ring configurations. The end result is a pair of ring responses returned to the requesting book containing the combined coherency information for the entire system. As ring messages are propagated asynchronously through each remote book back to the requesting book, request and response information is shipped in a single packet,

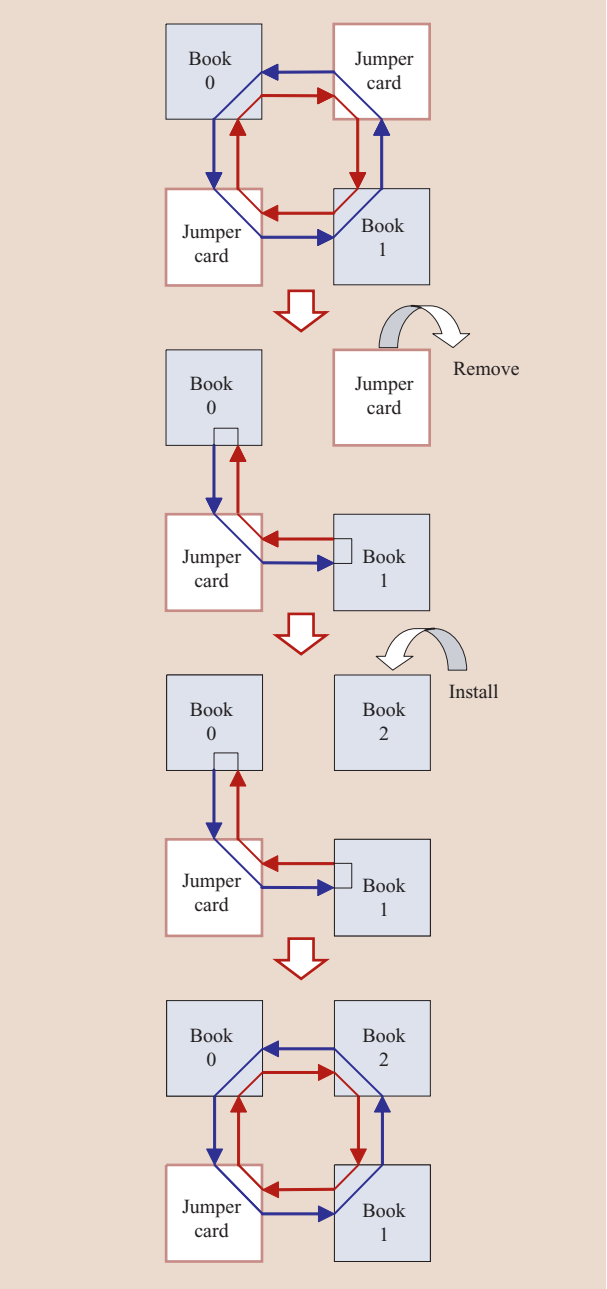

Figure 4

Concurrent book upgrade.

minimizing ring bus utilization. An intermediate response may possibly precede the pair of final responses returned. If sent, this intermediate response serves to expedite the return of remote L2 cache data to the requesting book as quickly as possible. The transfer of data via ring buses takes place concurrently as multiple command/response ring transactions are executed.

Table 2 L2 cache ownership states in the z990 system.

Invalid
MC 1, IM = 0, Read-Only to one CP[†]
MC 1, IM = 1, Read-Only to one CP, changed
MC 1, IM = 1, Read-Only to one CP, unchanged
MC 1, IM = 0, Read-Only to all CPs
MC 1, IM = 1, Read-Only to all CPs, changed
MC 1, IM = 1, Read-Only to all CPs, unchanged
MC 0, IM = 1, Read-Only to one CP, changed
MC 0, IM = 1, Read-Only to one CP, unchanged
MC 0, IM = 1, Read-Only to all CPs, changed
MC 0, IM = 1, Read-Only to all CPs, unchanged
MC 0, IM = 1, Exclusive to one CP, changed
MC 0, IM = 1, Exclusive to one CP, unchanged
MC 0, IM = 1, Unowned by CPs, changed
MC 0, IM = 1, Unowned by CPs, unchanged

[†]CP: central processor.

The IBM eServer z990 ring-protocol and cache-management schemes work in concert to allow contending remote operations to circulate as ring bus transactions. The z990 cache-management scheme adds essential extensions to the modified, exclusive, shared, invalid (MESI) cache ownership states employed in previous S/390* generations. These extensions exist in the form of intervention master (IM) and multicopy (MC) states.

When the IM bit is active for a particular address on a book, it indicates that this book was the most recent to receive ownership of that address and write the data into its cache. By definition, there can be no more than one book with IM = 1 for a given address. Furthermore, if the cache ownership state for a given address indicates that the data is changed, it follows that the IM bit for that address must be active, because changed data is allowed to be held in cache on only one book. When the MC bit is active for a particular address on a book, it indicates that read-only (unchanged) copies of the data for this address may exist in multiple books.

Table 2 lists each of the possible L2 cache ownership states in a z990 system. If the cache ownership state for a particular address on a book is invalid, the data for this address does not exist in the L2 cache on that book. Note that for an address to be valid, either of the conditions IM = 1 or MC = 1 must exist. Also, if an address is valid, it must be either read-only, exclusive to a CP, or unowned. All lines held exclusive must be IM = 1. Also note that valid lines with IM = 0 are always unchanged.

The following is a summary of the cache-coherent ring commands that may be launched from any given book, and conditions specific to the launch of these commands:

- *Fetch Read-Only* (CP instruction fetch which misses local L2).
- *Fetch Conditional Exclusive* (CP operand fetch which misses local L2). Grants exclusive ownership to requesting CP only when L2 cache management state is IM = 0 and MC = 0 on all remote books.
- *Fetch Exclusive* (CP operand fetch for subsequent store). Fetch which misses local L2.
- *Read-Only Invalidate.* Spawned from local CP Fetch Exclusive when L2 cache ownership state in requesting book is MC = 1. No data transfer performed, since requesting book already has a copy of the data.
- *Least Recently Used (LRU) Castout.* Stores aged, changed L2 data to memory on remote book.

With the exception of the Read-Only Invalidate command (which never transfers data), data for remote-fetch-type operations is sourced from a remote L2 cache when IM = 1 on a remote book. If IM = 0 on all remote books, data is sourced from the target memory book (i.e., the book which contains the memory for the specified address). Note that the target memory book (also known as the memory master, or MM, book) may exist on either the local book or one of the remote books, but there is one and only one target memory book for any given address.

For all remote operations in closed-ring configurations, an incoming ring message is received on each ring, and an outgoing ring message is generated on the same ring. Following receipt of the first incoming ring message, a coherency interrogation of the local L2 directory is performed, yielding a local response. To formulate the outgoing ring response, the local response is merged with the incoming ring response using the response coherency ordering shown in **Table 3**.

Before giving an example of how this response coherency ordering is used, we provide a brief description of each of the responses listed in Table 3:

- IM Hit – This local response is generated if L2 cache ownership state is IM = 1 and no local IM Reject or MM Reject conditions (described below) are indicated.
- IM Reject – This local response is generated if another remote operation contending for the same address has already indicated an IM hit response.
- MM Reject – This local response is generated if another remote operation contending for the same address could potentially access data at the target memory address.
- Memory Data – This local response is generated as a second response accompanied by data from the target

memory book if the local and incoming ring responses at the target memory book are any combination of Read Only Hit, Miss, or No Status (described below).

- Read Only Hit – This local response is generated if the L2 cache ownership state is Read-Only, IM = 0, and no local IM Reject or MM Reject conditions are indicated.
- Normal Completion – This local response is generated whenever processing for an LRU Castout operation has been completed at the target memory book.
- Miss – This local response is generated if the L2 cache ownership state is found to be Invalid and no local IM Reject or MM Reject conditions are indicated.
- No Status – This response is launched on both rings of all closed-ring configurations whenever a new remote fetch or remote store command is launched.

As an example of how the coherency-response-merging scheme in Table 3 is applied, if the incoming ring response is No Status and the local response is IM Hit, the outgoing ring response will be IM Hit (i.e., the higher-order response takes precedence). Note that the response forwarded as the first outgoing ring response (i.e., the early or first response) reflects the merged status from coherency interrogations on those books that have received an incoming first ring message on this ring. The response forwarded as the second outgoing ring message (i.e., the final or second response) reflects the merged status from coherency interrogations on all books.

Table 3 shows that the IM Hit response is the highest-ordered ring response. This means that for any remote book which observes a local IM Hit response, the final responses returned to the requesting book will be IM Hit responses. For remote fetch operations requiring data transfer (e.g., Fetch Read-Only), a local IM Hit response also ensures the generation of a special intermediate data response on one of the rings. The intermediate data response is always transmitted on the ring which results in the shortest path between the book sourcing the data and the requesting book. Intermediate data responses may precede or follow an IM Hit first response, but always precede an IM Hit second response. The intermediate data response does not factor into the coherency-response-merging scheme shown in Table 3.

For diagonally opposite books (which exist only in four-book closed-ring configurations), the incoming first response and the incoming second response are merged with the local response to formulate the outgoing message on both rings. This process is called *diagonal book accumulation*. Diagonal book accumulation ensures that all incoming second responses reflect the merged status of coherency interrogations on all books in four-book closed-ring configurations. Intermediate data responses sourced by any adjacent book in a four-book closed-ring configuration are returned to the requesting book in the minimal number of hops. (That is, the direction of the intermediate data response at an adjacent book will always be on the ring opposite to the first incoming ring message.) Note that an intermediate data response sourced by a diagonal book can be sent on either ring, since the paths between the data source book and the requesting book on the two rings are equidistant.

Table 3 Response coherency ordering for combined ring responses.

Order	*Response*
1	IM Hit
2	IM Reject
3	MM Reject
4	Memory Data
5	Read-Only Hit
6	Normal Completion
7	Miss
8	No Status

A critical component of the IBM eServer z990 ring coherency management system is the establishment of two system coherence points known as the intervention master (IM) and memory master (MM) coherence points.

As described earlier, an IM = 1 L2 cache ownership state indicates that this book was the most recent to receive cache ownership for that address, and that there can be only one book with IM = 1 for a given address. Remote operations which generate an IM Hit local response simultaneously set a token called IM Pending.

Activation of this cache-based token permits the operation which set the token to proceed while any subsequent contending requests for the same address are rejected via the IM Reject response described earlier. In general, the IM Pending token is set if

- L2 cache or memory data is to be sourced from a book.
- L2 cache or memory data or an IM Hit ring response is in the process of passing through a book (as a result of being sourced from another book).
- The cache ownership states for a Read-Only Invalidate operation are currently in the process of being updated.
- An LRU Castout operation is being processed at the target memory book, or store data for an LRU Castout operation is passing through a book (en route to the target memory book).

For the MM coherency point, there is only one MM book in the system. More specifically, this book is the target memory book. Remote operations which are the first to arrive at the MM book and which do not observe an IM = 1 L2 cache ownership state set a token called MM Pending. Activation of this memory-based token permits

the operation which set the token to proceed while any subsequent contending requests for the same address are rejected via the MM Reject response described earlier. Remote fetch operations which have set the MM Pending token return memory fetch data only when both incoming ring responses on the target memory book are Read-Only Hit, Miss, or No Status and the L2 cache ownership state is IM = 0.

In conjunction with the IM and MM coherence points, a third type of address interlock is used which ensures that incoming first-ring messages targeting the same address always leave the book in the same order in which they arrived. This first-in first-out (FIFO) ordering of first responses is maintained regardless of the order in which the incoming ring operations are actually processed. Maintaining the order of first responses helps to ensure that all outgoing ring responses always reflect a merged status that is consistent with directory coherency results.

Memory-coherent directory

The zSeries architecture provides a fully coherent memory for the entire multibook z990 system. This is a single fully shared memory address space, with all processors and I/O devices on all books having the capability of updating the entire address space and having access to updates made by any other processor or I/O device in the shared address space anywhere in the system. This coherency is managed entirely by the hardware. The coherency requirement can produce considerable traffic on the interconnections for maintaining coherency, in particular on the ring connecting the books.

To maximize efficiency and performance in a z990 ring-connected multibook system, techniques are employed to minimize the number of storage requests that require coherency broadcast requests on the ring. Ring capacity for handling broadcast requests is limited by the number of interconnecting wires and signaling rate the package technology can provide. Coherency broadcast requests on the ring also increase storage access latency owing to the distance and additional logic these requests must traverse. The z990 storage subsystem contains unique hardware features to minimize memory-coherency-checking traffic on the ring, thereby reducing the average storage access latency while maintaining the fully coherent memory required by the zSeries architecture. This is accomplished without requiring any memory management by software.

To accomplish this, the z990 server provides a memory-coherent directory (MCD) that tracks storage references in order to control which references require coherency checking on the full system (across all books) and which do not. Each memory location is physically backed by storage on one book (the home book), but the data corresponding to this storage location may be cached or used on any book. The MCD contains remote access tag (RAT) bits that indicate whether or not regions of storage have been referenced and are potentially cached on books other than the home book. When a storage access for a storage location located on the home book is initiated and data is not already cached locally in a favorable state, the RAT bit for the corresponding region of memory is referenced to determine whether memory-coherency checking for the referenced storage must take place across all books or may remain within the referencing book. Storage references for addresses with the backing storage on another book must always be broadcast to all books unless the data is already cached in a favorable state on the local book. When the contents of storage are removed from a book to be potentially cached elsewhere, the corresponding RAT bit must be set. This is done automatically as part of the data access.

The design of the MCD allows for some imprecision in the RAT bits to lessen the cost and performance impact of implementation. The MCD must always provide a positive indication when data has been cached outside the owning book, but may provide a false positive indication in some instances in which data is not actually cached outside the owning book. With this imprecision, one RAT bit can represent a range of storage addresses rather than a single address, which minimizes ring traffic overhead for maintaining the state of the directory. This avoids negating the effect of reducing memory-coherency ring traffic by adding a large amount of other traffic for maintaining the state of the MCD. Each RAT bit in the z990 represents 16 MB of storage, a size compatible with unit sizes of storage regions typically assigned by software to individual applications. This MCD design consumes a minimum of chip area and can be placed in a favorable location on the chip where it can be accessed with minimal delay.

The success of the MCD depends on the fact that a large multiprocessor system often runs a multitude of smaller workloads simultaneously, each utilizing a fraction of the available processors and memory. If software assigns tasks to processors on the same book as the physical storage they are utilizing, the RAT bits (which are initially off) remain off, and storage accesses remain within one book and do not drive any memory-coherency checking on the ring to other books. For workloads where processors are working with storage that is resident on other books, the RAT bits are set as a result of storage requests from the processors; this causes future memory-coherency checking for the affected address range by processors on the book containing the physical backing storage to be broadcast on the ring to the full system. RAT bits are reset by a Power-On Reset [5], and when storage is de-allocated by clearing the corresponding entry in the storage configuration array, a process normally associated with dynamic storage reconfiguration [6]. By

monitoring storage references and updating the RAT bits automatically, the hardware is able to dynamically modify memory-coherency checking for different applications to the appropriate scope of a single book or the whole system. This guarantees full adherence to architected memory-coherency requirements while providing improved performance for applications that are either intentionally or by chance contained within a single book.

Hardware manages the scope of memory-coherency checking automatically. The only role played by system software (e.g., the operating system) in this is to assign processors and storage to obtain the best effect of the known system structure and characteristics, and this role is not a requirement.

When large modular systems as disclosed here run workloads that do not require the resources of multiple books, it is advantageous to assign workloads to processors and memory that are contained entirely within one book where possible. When workloads cannot satisfy this requirement, the system automatically accommodates them by expanding the scope of memory-coherency checking as required.

Memory asynchronous interface

During the architecture and design phases of a chip, physical limitations often dictate certain aspects of the design point. For example, these limitations might include performance, power, cooling, space, size, available technologies, and cost.

In the prior IBM eServer z900 machine, the physical realities allowed an operating frequency relationship among the chips in the system chip set that satisfied all requirements. The central processor (CP) chips operated at a frequency f. The CP chips communicated with the SCC, SCD, MSC, and other central infrastructure chips, collectively called *nest chips* because they provide the common connection point for processors, memory, and I/O. The nest chips, including the main storage controller (MSC) chip, operated at $f/2$. The MSC chips communicated with the synchronous memory interface (SMI) chips, which operated at $f/4$. The SMI chips communicated with the DRAM chips, which operated at $f/8$.

To satisfy the data bandwidth requirements of each faster chip, the slower chip would supply data over a double-width bus at half the signaling rate of the faster chip, and this would be converted to the width and signaling rate required by the faster chip. All interchip communications remained synchronous because of the integral speed ratios and a latch in each faster chip to align its clock cycles to those of the slower chip.

For the z990, the available SMI technology could not meet the requirement to operate the SMI chips at half the speed of the nest chips. One option would have been to operate the chips as follows: f (CP), $f/2$ (nest), $f/8$ (SMI), $f/16$ (DRAM), with the SMI chips operating at one-eighth the speed of the CP chips rather than one-fourth, and with the DRAM chips operating at half the speed of the SMI chips.

The option of operating the SMI and DRAM chips at synchronously slower integral speeds would have resulted in system performance degradation, because the effects of the slower speeds could not be overcome with wider interfaces. It was decided that the best approach to maintaining the needed data bandwidth was to operate the SMI and DRAM chips at their maximum operating frequency, utilizing a fully asynchronous interface rather than the integer ratios used in the past. The MSC chips would have two clock domains: one to interface with the SMI chips at their frequency relationship (SMI domain), and one to interface with the nest chips at their frequency (nest domain).

To provide communications among the facilities in each clock domain, a robust yet flexible operating protocol had to be established so that the frequency ratio between the two domains could be varied.

The frequency ratio between the two clock domains is referred to in the design specifications as *gear ratio*; in this case, the ratio involves time rather than frequency. The fastest speed at which the nest domain can run with respect to the SMI domain is a 3/8 ratio. The slowest speed at which the nest domain can run with respect to the SMI domain is a 9/8 ratio. The normal operating ratio is 4/5 (i.e., the nest domain is running at 1.6 ns while the SMI domain is running at 2 ns).

Before we examine the internal communications protocol between two asynchronous clock domains within the MSC, it would be helpful to understand the external interfaces to the MSC and their associated communication protocols.

There is a synchronous interface for the command and address from the cache control chip (SCC) to the MSC chip. The command and address are delivered over two consecutive cycles: Cycle 1 contains the command and part 1 of the address; cycle 2 contains a null command and part 2 of the address. No gap is needed between commands. A "command accept" status, with a fixed time relationship to each command, is sent back to the SCC. Commands are sent on the basis of available MSC resources monitored by the SCC. The final status for a command has no fixed time relationship to the receipt of the command by the MSC. Commands can be executed out of order on the basis of MSC priorities and resources.

The MSC also has a synchronous interface to and from the SCD chips. Data flows to the MSC in a fixed relationship to its associated command. Data flows to the SCD chips in a fixed relationship to the "alert" status sent to the SCC.

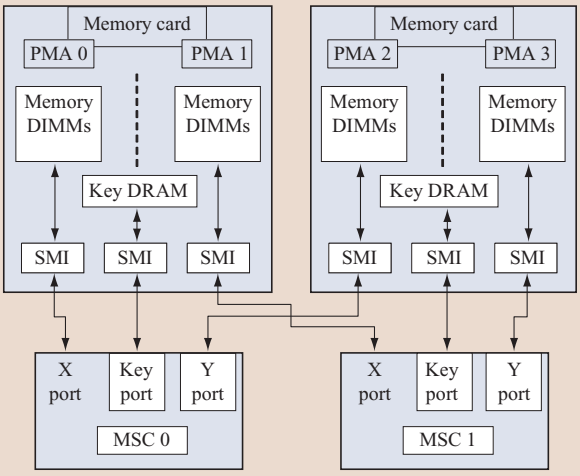

Figure 5

Memory structure.

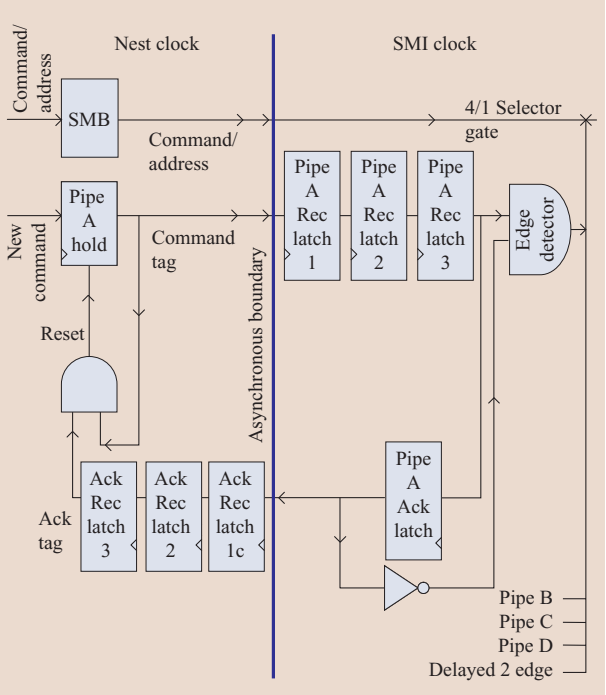

Figure 6

Asynchronous communications protocol. (SMB: speed-matching buffer; SMI: synchronous memory interface; Ack: acknowledge; Rec: receive.)

On the asynchronous interface from the MSC to the memory cards, the MSC has three independent operating partitions (data X, data Y, and key) with three independent physical interface ports, as shown in **Figure 5**. The X and Y ports for memory access are selected by address. The SCC encodes this selection in the command. The X port is connected to a processor memory array (PMA) on one memory card, and the Y port is connected to a PMA on the other memory card (the port with the shorter connection in the figure is considered local and the other port is considered remote: The X port is local for the first MSC chip, and the Y port is local on the second MSC chip). The key port, for accessing storage protection keys required by the z/Architecture* [7], is connected to the local memory card.

The X port, Y port, and key ports are all asynchronous interfaces utilizing the logic shown in **Figure 6**. Each X and Y port interface is further partitioned for fetch, store, and special operations. The separation of commands to the X, Y, and key ports is done in the nest domain. The gathering of status and data from the X, Y, and key ports is also done in the nest domain. There are a total of 48 tag signals and 48 acknowledge signals between the clock domains. Only one port is described in the following, although the description applies to all X, Y, and key port operations.

When information such as commands, addresses, data, or status has a destination in the other clock domain, the information is first placed in a speed-matching buffer (SMB) array, as shown in Figure 6. The write clocks and write enable controls for the SMB are generated in the same clock domain as the source information. No clocks are required to read the information from the SMB, only a signal to control a selector which gates the output of one of the array elements.

At the time the information is written into the SMB, a single control signal (Command tag in Figure 6) is sent through a series of three metastability latches. Given that the clock from the launching latch and the clock for the receiving latches are totally asynchronous [i.e., they have different cycle times and different clock phases with respect to each other from two separate phase-locked loops (PLLs)], the receiving latches may exhibit metastability; that is, their behavior may be unpredictable in that during level transitions the receiving latch may capture a one or a zero, or may exhibit a poorly timed, slow, or delayed level transition. A series of latches are inserted to reestablish stability in the receiving domain with properly timed and shaped transitions on the correct clock edge before the result can be used. Three latches are placed in series for metastability correction on the basis of timing analysis for the z990 application frequency. The clocks for the metastability latches are generated in the clock domain where the receiver latches for the information are located. The number of clock cycles it takes to pass through the three latches varies because the

first latch may act as a flush path with no delay rather than as a one-clock-cycle delay device. In every case, enough time elapses for the information in the SMB to become stable before the tag signal emerging from the metastability latches is used to gate the SMB array output.

The number of operations the nest sends to the X port, Y port, or key port is determined by the number of controllers available to handle the various requests. For example, there are four fetch operation controllers available on the X port. The nest does not send more than four fetch operations to the X port until an X port fetch operation completes and the controller becomes available for a new fetch operation. This allows the nest to send four back-to-back fetch operations. The SMB therefore is designed with enough elements to handle four concurrent X side fetch requests.

In Figure 6, one command tag signal is associated with each element of an SMB. As an SMB element is being written in the sending domain (SMB in Figure 6), a tag hold latch (Pipe A hold in the figure) is also being set on. Similar Pipe B, C, and D command tag logic exists for each of the other SMB elements, not shown in the figure except for their inputs to the 4/1 selector gate. The output of the Pipe A hold latch drives the input of the first metastability latch in the receiving domain. This first receiving latch (Rec latch 1) drives the input of a second receiving latch (Rec latch 2), the second latch drives the input of a third receiving latch (Rec latch 3), and the third latch (now stabilized) drives the input of a fourth latch (Ack latch) whose purpose is to return acknowledgment of the receipt to the sender. This acknowledgment latch output is sent back to the original clock domain as an acknowledgment tag (Ack tag) and eventually becomes the reset for the Pipe A hold latch. However, the acknowledgment signal must first pass through three receiving metastability latches (Ack Rec latches) in the original sending domain before resetting the Pipe A hold latch.

In the SMI clock (receiving) domain, an edge detector looks for receiving latch 3 to go to a set state while the acknowledgment latch is still in a not-set state. This detection becomes the basis for reading an element from the SMB. Owing to the asynchronous interface and the frequency application, two command tags (corresponding to different elements within the SMB) may emerge simultaneously from their respective metastability latch chains in the SMI clock domain. The logic has a provision to serialize these two events by delaying one (via the delayed 2 edge event to the edge detection in Figure 6) to prevent an attempt to simultaneously read two different elements of the SMB, since there is only one SMB bus to the SMI clock domain.

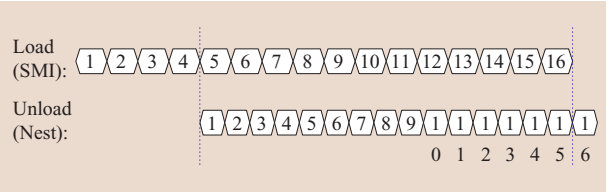

Figure 7

Speed-matching sequence in a 4/5 gear ratio.

Along with the edge-detection logic on the receiving domain, logic exists to check receiving latch 3 to ensure that it goes to a set state and holds. A condition in which receiving latch 3 would set for one cycle and return to a non-set state sets an error checker that would stop the machine. This latch should never set for only one cycle, and it should always be stable.

Because of the different clock rates in the nest and SMI domains, the data transfer rates within the two domains do not match each other. Special care is needed to achieve seamless data transfers on both sides and to maximize system performance. The z990 system places the SMB between the two clock domains to solve this problem.

For fetch operations, data flows from the SMI side to the nest side. Data flows in the opposite direction for stores. **Figure 7** is an example of data transfer for a fetch operation.

The z990 nest clock runs faster than the SMI clock. In order to maintain a seamless data transfer for fetches, once the data starts to load in the SMI side, the nest-side logic must wait for a certain number of cycles before it starts unloading the data. The number of cycles depends on the clock ratio between the two clock domains. Figure 7 illustrates the speed-matching behavior in the SMB. The clock ratio between the unload (destination) side and the load (source) side in the example is 4/5. Thus, in order to maintain seamless data transfer for 16 cycles of data without buffer data overrun, the unload side must wait until the load side preloads four or more cycles of data transfer.

Data flows in the opposite direction for store operations. Since the load side generally runs faster in this direction, there is no need for unload side to wait more than one count of data transfer before beginning to unload the buffer. If the clock ratio is defined as m/n, the formula for the number of data transfers to wait before beginning unloading is

$$\left\lceil 16 - 15\frac{n}{m} \right\rceil \qquad \text{if } m > n\text{; otherwise 1.}$$

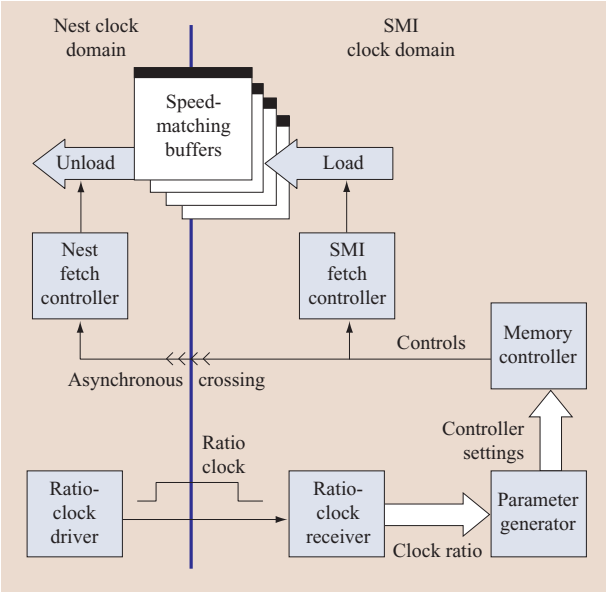

Figure 8

Clock ratio detector in z990 applications.

Table 4 shows the number of data transfers that must complete before unloading begins for various clock ratios in order to seamlessly transfer 16 cycles of data transfer. From the table, we can see that the hardware must behave differently for different clock ratios. A hardware clock ratio detector is included to adjust the wait time automatically on the basis of clock ratios. The clock ratio detector automatically calibrates the clock ratio between the two clock domains. According to the ratio detected, the detector logic informs other logic in the MSC chip to adjust its behavior to achieve maximum efficiency.

The clock ratio detector shown in **Figure 8** consists of two physical parts: a ratio-clock driver in the nest domain and a ratio-clock receiver in the SMI domain. The ratio-clock driver constantly outputs ratio-clock pulses. The pulse width equals 128 local nest clock cycles. The receiver in the SMI domain counts the pulse width N in term of its local SMI clock cycles. For each clock pulse, the receiver can determine the clock ratio between the two domains by the following formula:

$$\text{Ratio } r = N/128.$$

The r here is usually a real number, but the clock ratio output must be an integer in a digital machine. The z990 server uses a predefined resolution of 16. The output integer R in a real machine can be represented as

$$\text{Integer ratio } R = N/8.$$

Table 4 Speed-matching delay for 16-cycle fetch data transfers on the z990 SMI-to-nest interface.

Clock ratio	*Data transfer wait count*
1	1
7/8	3
4/5	4
3/4	5
5/8	7
1/2	9
3/8	11

In addition to the basic ratio-counting ability, the clock ratio detector has the ability to adjust the output resolution and output range of each step. It also prevents possible oscillation of the output ratio if the ratio is right at the boundary where output is about to change. For error reporting, the detector continuously monitors for any change of clock ratio and will report an error if the ratio changes in a wrong direction.

Once the ratio detector determines the system clock ratio, the ratio is used to generate parameters to adjust memory controller timing behavior. This behavior ultimately changes the relative load and unload times between the two clock domains so that the system can automatically maintain a seamless data transfer with minimum wait.

Memory power governor

Designing a power supply and cooling system for the maximum possible memory utilization would have been unnecessarily expensive on the z990 system, since the majority of the time the memory utilization is low owing to high cache-hit ratios. The alternative was to design a power supply for a more typical case with some margin of cooling capacity, and then add power governor logic to prevent overheating in the rare cases in which the memory utilization in the system comes close to reaching its maximum. The solution offered is based on capping the average DRAM current consumption by limiting the number of memory transfers during each fixed refresh interval. This can guarantee that the total memory card power consumption will never exceed the limit. The power management is transparent to the operating system and is done concurrently with normal machine activity.

Each DRAM must be refreshed within a certain period of time, which is fixed for a given DRAM technology. This makes it an excellent reference point for observing the memory transfers. Any command (Activate, Read, or Write) that has to be issued to fetch or store the data in the DRAM is considered to be a memory transfer. The DRAM memory power consumption depends strongly

on the number of received Activate, Read, and Write commands. The power governor indirectly measures the current by counting these commands during each refresh interval. If the current exceeds the limit, the power governor changes the timing parameters to reduce the number of memory transfers during the next refresh interval.

The power governor implementation shown in **Figure 9** includes a governor counter which counts the number of energy-consuming commands sent to the memory within one refresh interval. Once the refresh controller completes a memory refresh interval, the value of the governor counter is latched into the governor hold register and the governor counter is reset. Once the refresh request receives a refresh grant from the DRAM control logic, the power governor decides whether to change the memory timing parameter. At this time, all memory controller state machines with the exception of the refresh controller are idle. Changing parameters at this time guarantees that the memory timing restrictions are not violated. If the value in the governor hold register exceeds the threshold value in the governor ON threshold, the timing behavior for mainline functions (ordinary memory fetches and stores) is changed for the next refresh interval. The time for executing a single fetch or store operation remains the same, but the time between two consecutive operations is changed by adding idle cycles. At the end of the next refresh interval, if the governor hold register is less than the governor OFF threshold, the 'governor restores the initial timing parameters.

A worst-case analysis shows that for an IMS[1] workload a memory access would be initiated once every 38.4 ns in a 16-processor, two-book configuration. On the basis of the current threshold, the power governor would be activated if the rate of memory requests was greater than 1 every 33 ns. These calculations are based on an IMS workload because it has one of the highest memory interface utilizations.

The operating parameters are programmable and may be dynamically updated by firmware to adapt to operating conditions. The power governor implementation provides a simple and flexible means of balancing system performance against system power and cooling limits.

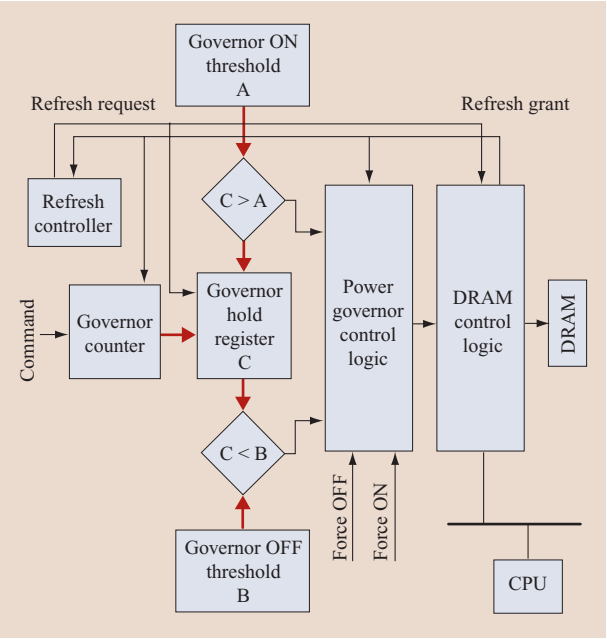

Figure 9

z990 power governor.

Concluding remarks

In this paper we have described a few of the advanced design features of the z990, including

- A modular package that supports a variable number of processing books for each system.
- A ring-connected structure and communication protocol that provides high bandwidth and fast response connections between books to sustain high performance in a system with fully shared, fully coherent memory, and that supports dynamic reconfiguration for adding books while the system is running.
- Additional cache states, and a memory-coherent directory, to minimize memory-coherency interrogation requests between books for enhanced performance and efficiency.
- An asynchronous memory interface to allow processors and memory to each operate at its optimal frequency, with seamless transfers, for maximum performance.
- A memory power governor to obtain maximum performance consistent with system power and cooling limits.

These features are included to improve the performance of the IBM eServer zSeries family of processors.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. P. Mak, M. A. Blake, C. C. Jones, G. E. Strait, and P. R. Turgeon, "Shared-Cache Clusters in a System with a Fully Shared Memory," *IBM J. Res. & Dev.* **41,** No. 4/5, 429–448 (July/September 1997).
2. K. M. Jackson and K. N. Langston, "IBM S/390 Storage Hierarchy—G5 and G6 Performance Considerations," *IBM J. Res. & Dev.* **43,** No. 5/6, 847–854 (September/November 1999).

[1] IMS is an IBM transactional and hierarchical database management software application.

3. P. R. Turgeon, P. Mak, M. A. Blake, M. F. Fee, C. B. Ford III, P. J. Meaney, R. Seigler, and W. W. Shen, "The S/390 G5/G6 Binodal Cache," *IBM J. Res. & Dev.* **43,** No. 5/6, 661–670 (September/November 1999).
4. J. Probst, B. D. Valentine, C. Axnix, and K. Kuehl, "Flexible Configuration and Concurrent Upgrade for the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 551–558 (July/September 2002).
5. IBM Corporation, *z/Architecture Principles of Operation* (SA22-7832-01), October 2001, pp. 4–47; see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
6. S. G. Glassen, R. S. Capowski, N. T. Christensen, T. O. Curlee III, R. F. Hill, M. J. Kim, M. A. Krygowski, A. H. Preston, D. E. Stucki, and F. J. Cox, "Method and Apparatus for Dynamic Storage Reconfiguration in a Partitioned Environment," U.S. Patent 5,819,061, October 6, 1998.
7. IBM Corporation, *z/Architecture Principles of Operation* (SA22-7832-01), October 2001, pp. 3–9; see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.

Received October 15, 2003; accepted for publication April 12, 2004; Internet publication May 28, 2004

Pak-kin Mak *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (pmak@us.ibm.com).* Mr. Mak is a Distinguished Engineer in zSeries custom hardware design. He received his B.S.E.E. degree from the Polytechnic Institute of New York and his M.B.A. degree from Union College. Mr. Mak joined IBM Poughkeepsie in 1981, working on the ES/3090 BCE cache design. He has designed high-end system controllers and L2 caches for ES/9021 bipolar-based systems, and he was the lead architect for the S/390 G4, G5, G6, z900, and z990 processor storage/cache subsystem designs. Mr. Mak currently holds 14 patents; he has received six IBM Invention Achievement Awards, three IBM Outstanding Innovation Awards, three IBM Outstanding Technical Achievement Awards, and two IBM Division Awards.

Gary E. Strait *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (strait@us.ibm.com).* Mr. Strait is an Advisory Engineer in eServer Hardware Development. He was the logic team leader for the z990 SCE I/O interface. He joined IBM in 1980 after receiving his B.S. and M. Eng. degrees in electrical engineering from Rensselaer Polytechnic Institute. Mr. Strait previously held design positions on the storage subsystem of the IBM ES/3090, ES/9021 subsystems, and the I/O interface of the S/390 G4, G5, G6, and z900 systems. He has received four IBM formal awards, holds four U.S. patents, and has three patents pending.

Michael A. Blake *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (mablake@us.ibm.com).* Mr. Blake joined IBM in 1981 after receiving his B.S. degree in electrical engineering from Rensselaer Polytechnic Institute. He is a Senior Technical Staff Member in eServer Hardware Development, and was the overall logic team leader for the z990 SCE, as well as lead ring unit designer. He previously held design and team leader positions on the IBM ES/3090, ES/9021, S/390 G4, G5, G6, and z900 systems. Mr. Blake has received several IBM formal awards, including two IBM Outstanding Technical Achievement Awards, two IBM Outstanding Innovation Awards, and an IBM Corporate Award for his work on the z900 SCE. Mr. Blake currently holds four U.S. patents and has seven patents pending.

Kevin W. Kark *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (kark@us.ibm.com).* Mr. Kark received his B.E.E.E. degree from the City College of New York in 1983 and his M.S.C.E. degree from Syracuse University in 1989. He joined IBM in 1983 at the IBM Product Development Laboratory in Poughkeepsie in the storage subsystem organization. Mr. Kark is a Senior Engineer in eServer Hardware Development and has held various design and team leader positions on the IBM ES/9000 and IBM S/390 CMOS G3, G4, G5, G6, z900, and z990 systems. He was the logic team leader for the z990 main storage controller. Mr. Kark has received several IBM formal awards, including three IBM Outstanding Technical Achievement Awards, an IBM Outstanding Innovation Award, an IBM Division Award, and an IBM President Award (Europe). He currently holds three U.S. patents and has two patents pending and several publications.

Vesselina K. Papazova *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (papazova@us.ibm.com).* Mrs. Papazova is an Advisory Engineer in eServer Hardware Development. She was responsible for the memory interface logic design for the z990 SCE and was a designer on the ring unit. She received her M.S. degree in electrical engineering from the Technical University, Sofia, Bulgaria, in 1995. From 1995 to 1998 Mrs. Papazova worked as a microprocessor system designer for ROCON, LCC. In 1998 she joined the Sigma-Delta Corporation in Bulgaria, where she was responsible for I/O unit logic design for single-chip microcontrollers. In 2000, she joined IBM upon her arrival in the United States. Mrs. Papazova has one pending patent.

A. E. (Rick) Seigler *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (seigler@us.ibm.com).* Mr. Seigler joined IBM Poughkeepsie in 1980 after receiving B.S.E.E. and M.S.E.E. degrees from Rensselaer Polytechnic Institute. He worked as a logic designer and systems test engineer on ES/3090 systems, and as a recovery/serviceability systems test manager for ES/3090 and ES/9021 systems. In 1992 he joined S/390 Custom Hardware Design, where he now works as an Advisory Engineer; prior to that, he served one year on an IBM Faculty Loan assignment at the Georgia Institute of Technology in Atlanta. He was the z990 SCE test floor leader following his assignment as a logic designer on the ring unit. Mr. Seigler holds seven patents and has four patents pending; he has received IBM Outstanding Technical Achievement Awards for his work on the G4, G5, G6, and z900 systems.

Gary A. Van Huben *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (vanhuben@us.ibm.com).* Mr. Van Huben joined IBM in 1986 and is currently an SCE design team leader in eServer Hardware Development. He has held various design assignments involving the S/390 storage controller, central processor, and I/O subsystem. He served as the dataflow chip team leader for the z990 SCE in addition to his assignments as a logic designer on the ring unit. Mr. Van Huben also architected and supervised the data management methodology and processes used to develop the z990 CEC. In 1986, he graduated from Clarkson University with a B.S. degree in electrical and computer engineering; he currently holds 14 U.S. patents and has received five IBM Invention Plateau Awards, two IBM Outstanding Technical Achievement Awards, and an IBM Outstanding Innovation Award for his work on the z900.

Liyong Wang *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (liyongw@us.ibm.com).* Mr. Wang is a Staff Engineer in eServer Hardware Development. He received an M.S. degree in electrical engineering from Rensselaer Polytechnic Institute and a B. S. degree in engineering physics from Tsinghua University, China. He joined IBM in 2000 after working for Pitney Bowes, Inc., as an ASIC engineer. Mr. Wang is a logic team member for the z990 MSC; he currently has one patent pending.

George C. Wellwood *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (wellwd1@us.ibm.com).* Mr. Wellwood joined IBM in 1963 in the thin-film memory development area. In 1969 he joined the S/360 Model 195 development team and has since remained in mainframe development. He was a designer on the S/370 Models 3032 and 3033. On the H series machines, he was a designer on the memory controller. On the G4, G5, and G6 CMOS machines, he was a designer on the L2 cache and memory subsystem. For the z900 and z990 machines, he was a designer on the memory controller. Mr. Wellwood was an Advisory Engineer in S/390 Custom SCE Design and is now a consultant to that team. He received IBM Outstanding Technical Achievement Awards for his work on S/390 G4 L2 cache development in 1997 and S/390 G5 cache and memory subsystem development in 1998. He holds two U.S. patents and has five publications.