IBM Technology

IBM Enterprise X-Architecture™ Technology

Reaching the summit

First Edition 2002

Foreward

IBM Enterprise X-Architecture™ Technology is a collection of articles that describe the chipsets and systems that make up the **@server** xSeries products. This book is written by technical professionals to communicate their ideas to other technical professionals. The main focus is on the new, different and unique aspects of the initial design and design methods of the referenced products.

This book is not intended to be used as a product specification reference or user's guide. It does not contain detailed descriptions of the product features, nor does it explain how the products are to be used. Rather it is an attempt to tell the reader something about the new ideas and techniques used in the current and future design of the products. Product features described in this publication may be different when the products are announced.

Much work was put into trying to maintain consistency throughout the book but because the articles were written by many different authors, this was sometimes difficult to achieve. Major revisions of the original articles are minimized to preserve the authenticity and vitality of the information.

This book demanded a significant amount of time from the authors when they were busy developing a product. It required repeated critiques of their writing and caused frequent interruptions in their work. This editor is grateful to the authors for their cooperation in this effort. Special thanks are due to Paul Keeling, Production Manager, InStudio, IBM Canada and his talented team. Also, special thanks are due to Sudhir Dhawan, Senior Technical Staff Member, whose dedication and hard work allowed this project to complete in a timely manner. Jennifer Vargus and Jim Hanna deserve special thanks for their help in critiquing several articles. Also, this editor is grateful to Harry Schultze, System Manager, xSeries Server Development for "pulling" me out of retirement and providing me this opportunity for an enjoyable and unusual experience.

George Mirabella, Editor

Preface

Over the last five years we have witnessed major changes in the PC server environment, including the need for high availability, scalability, performance and overall reliability and operational costs. As business leaders wrestle with the challenges of market globalization, or attempt to improve Information Technology asset management to continue growth, they are becoming increasingly aware of the limitations posed by internal infrastructure and boundaries. Against the backdrop of growing e-business operations, the tradeoffs between the low-cost advantages of PC based servers vs. the robustness of mainframe systems come to the forefront. Business and economic factors are driving the need to bridge the gap between these high-end, high maintenance systems and the low-cost systems.

Where once Intel® architecture server usage was limited to providing only simple file-and-print services, we have seen its role expand into providing business-critical resources for e-business applications, Customer Resource Management (CRM), Enterprise Resource Planning (ERP), business intelligence, supply chain management, collaboration and other traditional "big iron" arenas.

To drive the design and development of server solutions that address the growing market needs for mainframe robustness and reliability, IBM assembled a team of highly experienced and talented engineers and architects and chartered them with exploiting upcoming technological advances to bring large systems advantages to the PC based servers.

The team addressed these business needs through Enterprise X-Architecture™ (EXA) Technology. The IBM X-Architecture strategy embodies the team's belief that the capabilities IBM has delivered on larger systems can be applied to industry-standard servers. EXA, building on the IBM X-Architecture blueprint, paves the way for unprecedented scalability, flexibility, availability, performance and operational efficiencies in deploying server resources to meet dynamic e-business needs.

EXA Technology addresses both IA-32 and 64 bit systems and provides a road map from single node to multiple nodes (16-way) Symmetric Multiprocessor systems. This innovative architecture enables customers to start small and easily upgrade to a larger system by simply connecting nodes with cables. Similar technology is applied to provide expansion of Input/Output capabilities. Enterprise X-Architecture provides users of Intel-based server's functionality not previously available on that platform, including partitioning and serviceability (without having to bring the system down) and mainframe-like reliability – at a much lower cost.

These innovations demonstrate that evolutionary thinking can lead to revolutionary advances in technology. It brings to industry-standard servers the kinds of capabilities formerly only available to users of mainframes and other high-end systems. These new capabilities, combined with existing X-Architecture technologies, result in revolutionary "economies of scalability," unmatched flexibility and new levels of server availability and performance.

The architecture and development of Enterprise X-Architecture has been a multi-location effort of hundreds of individuals, both within and outside IBM. EXA incorporates the architectural advances, technology innovations, engineering and programming developments from Watson Research, Burlington, Rochester, Raleigh, Austin, Kirkland and Poughkeepsie. The IBM team partnered with Intel® to validate the chipset and the processor together. The articles in this book describe some of their contributions. My thanks to the entire team that worked so hard to bring this technology to market.

Bill Colton
General Manager
@server xSeries Group
Raleigh, NC

IBM Enterprise X-Architecture™ Technology

Contents

Enterprise X-Architecture™ Technology Overview

– Jeffrey D. Brown, Sudhir Dhawan

Introduction

IBM is revolutionizing the world of Intel® compatible system development. With its recent announcement of the Enterprise X-Architecture (EXA) Technology, the foundation of the IBM @server xSeries products, IBM has captured the opportunity to bring mainframe features to Intel-based servers. This article provides an overview of the innovative technology and focuses on the motivation behind its conception and the unique value that systems built from EXA bring to the Intel-based server market.

In 1998, a number of trends emerged that influenced the direction of the server market. The operating systems of choice were in transition. One to four-way Symmetric MultiProcessor (SMP) servers using Windows NT™ and UNIX™ were gaining market share. At that same time Intel was promoting 64-bit processors and it was clear that they would play a major role in changing the market.

There were several established chipset vendors whose products were very niche-focused resulting in a chaotic market and development environment. Vendors entered and exited the market quickly. Mergers and buyouts were common. Chipset features were diverse and no single vendor provided a consistent set of features that could be applied across the spectrum of products that the market required.

As the Windows operating system was gaining popularity, Microsoft® was pushing the use of its technology in mission-critical Online Transaction Processing environments. Recognizing the need to provide this high-end functionality at a low cost, IBM began to investigate how to leverage its proven mainframe technology to bring the scalability, reliability, and performance to Intel processor based servers. Soon after, IBM designed a blueprint of the Enterprise X-Architecture™ which leveraged existing, innovative IBM technologies with the intent to build the most powerful and robust Intel-based servers suitable for businesses of all sizes. While the Intel® microprocessors used in xSeries products provided industry-leading processor performance and features, critical enhancements to the performance, scalability, reliability, availability, and serviceability of the overall system could only be provided through the innovation of a new chipset.

Enterprise X-Architecture Conception

During the conception phase of EXA, one of the fundamental tasks was to balance schedule, development expense and function. A steering committee was formed to drive the definition of the Enterprise X-Architecture Technology and to ensure high functionality, aggressive time to market and low development cost.

Experts from xSeries (then Netfinity) product engineering joined with members of IBM's Research Division and IBM's Server Group, including experts from iSeries (AS/400), pSeries (RS/6000), and zSeries (S/390). These experts represented IBM's significant inventory of server technology and extensive experience in system design.

Given the competitive intensity and the emerging customer demand, emphasis on "time-to-market" was as crucial for success as the technology features and cost/performance characteristics.

An important aspect of reducing development costs was to focus on an optimum number of chips that were flexible enough to support products across the entire span of low to high end servers. After careful research, the team of experts defined two chipsets - IBM XA-32™ and the IBM XA-64™. The IBM XA-32™ chipset was designed to implement systems using the Intel Xeon™ Processor MP while the IBM XA-64™ chipset was designed to implement systems using the Intel Itanium™ family of processors.

To achieve the economies of scale, two chips were designed to work across both platforms with the third being unique to either the 32-bit or the 64-bit architectures. In total, four new chips were designed as part of this undertaking.

1. The Memory and I/O Controller[1] is the first common chip that implements the system bus interface, memory controller and RXE Expansion Port link that provides remote I/O capability to the system.
2. The I/O Bridge[2] is the other common chip that connects to the Memory I/O Controller (MIOC) using the RXE Expansion Port link and generates three PCI-X busses. Multiple I/O Bridges (IOBs) may be connected using cables to provide the system remote I/O expansion capability.

The third chip in each chipset is the Cache/Scalability Controller (CSC32)[3] which implements the XceL4™ Server Accelerator Cache (L4 Cache) system coherency logic and the SMP Expansion Port Link.

3. CSC32 enables the IA-32 systems to scale from 4-way SMP to 16-way SMP.
4. CSC64 enables the 64 bit systems to scale from 4-way SMP to 16-way SMP.

To meet the needs of the low end market, systems can be designed with just the two common chips without the additional cost of the CSC32 chip. A low end Basic 4-way SMP can be constructed using Intel's Xeon™ Processor MP. However, to expand beyond the 4-way configuration, and to implement what is known as an Enhanced Node, the Cache/Scalability Controller is required. For the high end Intel Itanium™ processor based 4-way SMP systems the CSC64 chip is required.

This innovative partitioning of functionality of the common underlying technology resulted in high levels of development effectiveness and enabled the construction of a comprehensive line of systems.

Customer Value

IBM @server xSeries systems support IBM's X-Architecture which introduces many innovative mission critical system functions and attributes. These important features of the X-Architecture have roots in IBM's mainframe and high end server system architectures. The new Enterprise X-Architecture Technology continues to expand the higher level X-Architecture by implementing support directly in the hardware. Some examples of these capabilities are:

- XpandOnDemand™ Scalability
  - The ability to expand to 16-way SMP by adding other 4-way / 8-way systems
  - The ability to expand the I/O capacity by adding remote I/O which includes loops for fault tolerance
  - Partitioning system resources for optimum performance
- Active Memory™
  - The ability to hot swap and hot add memory
- Memory ProteXion™
  - The ability to tolerate a DRAM chip failure using redundant pathways within the controller. These include chipkill™ tolerant Error Correcting Codes (ECC), redundant bit steering, and dynamic scrubbing.

The leadership in Intel-based servers enabled by the Enterprise X-Architecture Technology is the result of the functional and architectural innovation supporting the chipset implementation, the speed-to-market availability of the chipset, and the overall performance, scalability and the price/performance obtained. EXA supports leadership performance in TPCC and other benchmarks.

Leadership And Feature Overview

The Enterprise X-Architecture Technology embodies "leadership" for Intel-based servers and brings mainframe class features to the price/performance server marketplace. The features define a new standard for the Intel Xeon Processor MP and Itanium Processor family based server offerings.

Scalability is the most obvious feature providing mainframe class performance and high volume server cost structures. Systems built from the IBM XA-32 and IBM XA-64 chipsets support scalability up to 16-way SMP systems for both Intel Xeon™ MP and for Intel Itanium™ family processors. Built-in clustering hardware support enables even larger systems.

Configurability and partitioning[4] is another key mainframe class capability. The Enterprise X-Architecture Technology embodied in the IBM XA-32 and XA-64 chipsets support physical partitioning that allows a 16-way system to have all the management advantages of a 16-way system but functionally be configured as four 4-ways, two 8-ways, a 4-way and a 12-way, etc.

Leadership performance is another key attribute of the systems built from these chipsets. Incorporating an innovative Level 4 Cache design, referred to as the XceL4™ Server Accelerator Cache, enables these systems to scale to 16-way and exhibit significant performance advantages over competitor offerings.

When users hear "mainframe" features the primary attribute considered is Reliability, Availability, and Serviceability (RAS). IBM XA-32 and XA-64 chipsets implement features designed to support enhanced RAS. Error detection and correction is pervasive throughout the XA-32 and XA-64 chipsets and covers processor system bus, XceL4 Server Accelerator Cache data and control directory, SMP Expansion Port Link retry and fail over, Active Memory™ and Memory ProteXion™, and finally RXE Expansion Port link retry and fail over.

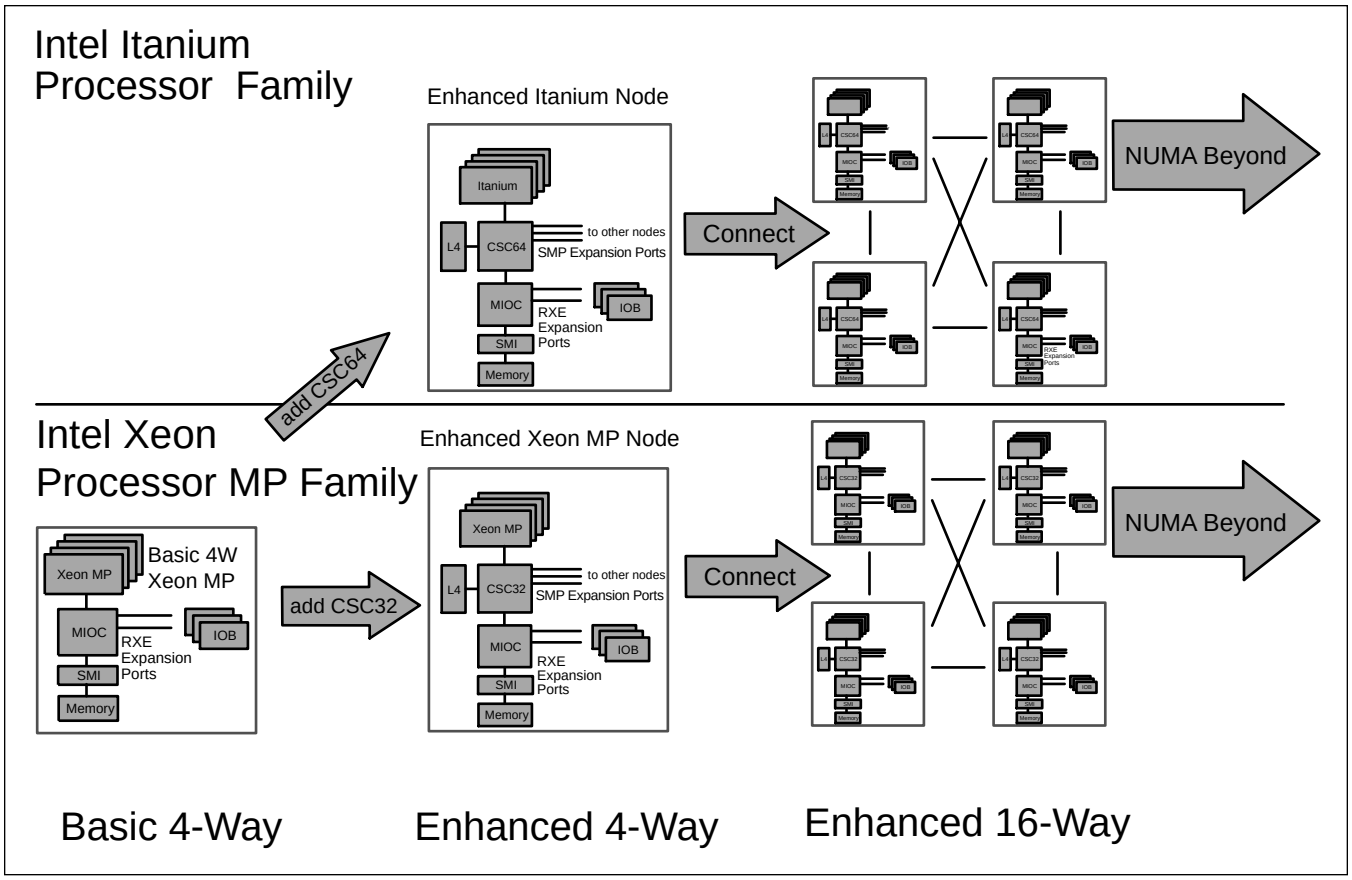


Figure 1 Enterprise X-Architecture Implementation Overview

The IBM XA-32 and XA-64 chipsets leverage IBM's CMOS 7SF foundry technology. CMOS 7SF brings innovative technology features to the foundry ASIC market including copper interconnects and embedded DRAM (eDRAM) technology. It supports Single Data Rate (SDR) DRAM and Double Data Rate (DDR) DRAM for system flexibility. The data arrays for the XceL4 Server Accelerator Cache are implemented using high performance DRAMs initially offered as graphics memory.

System Architecture

The Enterprise X-Architecture Technology embodied in the IBM XA-32 and XA-64 chipsets initially supports a family of four possible rack-optimized products.

1. xSeries 360, IA-32 Basic Node[5]
   – 3-EIA (1 EIA = 1.75 inches) Basic 4-way Intel Xeon Processor MP server
2. xSeries 440, IA-32 Enhanced Node[6]
   – 4-EIA 8-way Intel Xeon Processor MP server
3. IPF Enhanced Node Prototype (IPF Node)[7]
   – 4-EIA 4-way Intel Itanium Processor Server
4. xSeries RXE-100™ Remote Expansion Enclosure[8]
   – 3-EIA Remote Expansion Enclosure with up to 12 PCI-X slots.

Figure 1 shows the range of platforms that can be built from the IBM XA-32 and IBM XA-64 chipsets. The most basic of these systems is an IA-32 server, shown in the lower left of the figure. Using this as a base, IA-32 and IPF Enhanced Nodes can be built by adding CSC32 and CSC64 chips. The Enhanced Nodes can then be connected to implement up to 16-way SMP systems. This shows the degree of flexibility built into the chipsets such that a given chip is used in some, if not all, of the platforms.

xSeries 360

The entry product of the Enterprise X-Architecture family, as shown in Figure 2, is the xSeries 360 Platform. xSeries 360 is a 3-EIA rack-optimized product with price/performance as its main goal. The system includes

- 4 Intel Xeon™ Processors MP
- 8 DIMM slots for system memory
- 6 PCI-X slots
- RXE Expansion Port for connection to IBM RXE-100 Remote Expansion Enclosures.

xSeries 360 is a standalone system supporting between one and four Intel Xeon™ Processor MP family processors. The chipset connects the processors with eight DIMMs of DDR DRAM memory and a complete internal I/O subsystem supporting six PCI-X slots.

To support building a system with more than six PCI-X slots, xSeries 360 externalizes an RXE Expansion Port. The RXE Expansion Port can be attached to an IBM RXE-100 Remote Expansion Drawer containing up to 12 additional PCI-X slots. The architecture allows cascading RXE-100 drawers for a total of 48 PCI-X slots.

xSeries 360 uses the Memory and I/O Controller (MIOC) and PCI-X I/O Bridge (IOB) chips to configure a basic one to 4-way Intel Xeon™ Processor MP SMP system.

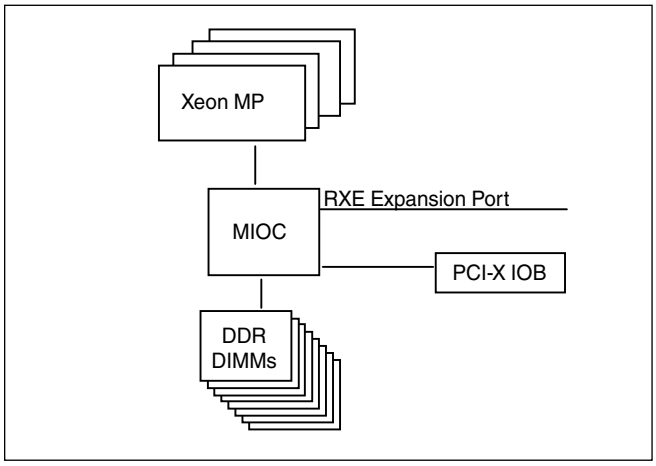


Figure 2 xSeries 360 XA-32 System

The system bus supports four processors. The memory bus bandwidth is matched to the system bus for data and address transfers. The MIOC chip implements the system bus as defined by Intel®.

The MIOC has two RXE Expansion Port interfaces. The high speed RXE Port is IBM's I/O expansion innovation that allows customers to add separate I/O drawers filled with PCI-X slots. In the xSeries 360 platform one of the Remote I/O ports drives a PCI-X IOB chip in the same drawer supporting six PCI-X slots. The second Remote I/O port goes directly to a cable connection at the rear of the xSeries 360 drawer for I/O expandability via a IBM RXE-100 Remote Expansion Enclosure.

In xSeries 360, the MIOC chip supports an 8-DIMM DDR DRAM system memory interface. This allows xSeries 360 to configure up to 8GB of DDR DRAM memory using eight, 1GB DIMMs, or up to 16GB using 2GB DIMMs.

xSeries 440

The xSeries 440 system embodies the largest set of EXA features in a highly-scalable platform. xSeries 440 is a rack-optimized product with performance and scalability as its main goals. The base xSeries 440 package is a dense 4-EIA containing

- 8 Intel Xeon Processors MP
- Up to 32 DIMMs for system memory
- 6 PCI-X slots
- 64MB of XceL4 Server Accelerator Cache
- SMP Expansion Ports for interconnect to another xSeries 440 supporting scaling up to 16-way SMP
- RXE Expansion ports allowing connection of IBM RXE-100 Remote Expansion Enclosures.

The xSeries 440 system is a standalone system with up to eight Intel Xeon MP processors. The processors are internally arranged on two 4-way nodes referred to as SMP Expansion Modules. Each SMP Expansion Module contains four processors and several components of the IBM XA-32 chipset. The chipset connects the four processors with 32MB of XceL4 Server Accelerator Cache for optimal performance and 16 DIMMs of SDR DRAM.

Each SMP Expansion Module has SMP Expansion Port connections to attach to other SMP Expansion Modules. By interconnecting SMP Expansion Modules, the IBM XA-32 chipset can scale from 4-way to 16-way. The xSeries 440 package itself can hold two SMP Expansion Modules, or eight processors.

The package externalizes SMP expansion port connections such that a xSeries 440 system can be connected to another 8-way xSeries 440 system to form a 16-way. The xSeries 440 system can also be configured with a single SMP Expansion Module. In this case, up to four xSeries 440 systems can be interconnected to form a 16-way.

Regardless of whether there are one or two SMP Expansion Modules installed, there is a complete I/O subsystem supporting six PCI-X slots in the xSeries 440 system drawer. The xSeries 440 system drawer also externalizes two RXE Expansion Ports for attachment of IBM RXE-100 Remote Expansion Enclosures. Each RXE Expansion Port connection can be attached to a RXE-100 drawer containing up to 12 more PCI-X slots per drawer. For further expansion, any RXE-100 drawer can cascade to other RXE-100 drawers.

To further take advantage of the XpandOnDemand scalability of interconnected xSeries 440 systems, it is possible to configure multiple xSeries 440 systems into one SMP server or to configure them as unique SMP servers. The IBM XA-32 chipset supports the concurrent addition or removal of xSeries 440 SMP Expansion Modules.

xSeries 440 builds upon the Memory and I/O Controller (MIOC chip). However, instead of the eight DIMMs of memory supported in xSeries 360, xSeries 440 supports attaching up to 16 DIMMs of DRAM memory per MIOC. Each xSeries 440 system contains two MIOC chips, as shown in Figure 3, for a total of 32 DIMMs in the 8-way xSeries 440. Also shown in the figure is the Cache / Scalability Controller 32 (CSC32) chip connected to MIOC. CSC32 connects to the QuadT bus interface of MIOC.

CSC32 has a full system bus interface to connect to four Intel Xeon™ Processors MP, a XceL4 Server Accelerator Cache interface and three SMP Expansion Port interfaces for connection of up to three SMP Expansion Modules. Finally, the CSC32 interfaces to Memory and I/O Controller through the MIOC's QuadT bus.

The XceL4 Server Accelerator Cache on CSC32 supports a 32MB cache built from three DDR DRAM chips and supports 3.2GB/s bandwidth of the Intel Xeon™ MP system bus.

The three SMP Expansion Ports can be connected to three other CSC32's. Each SMP Expansion Port can transfer data at 1.6GB/s in each direction simultaneously. A novel circuit driver technique, simultaneous bi-directional signaling, allows data to travel in both directions on the same wire at the same time at very high speed.

The SMP Expansion Ports not only connect CSC32s to form larger SMPs, but can also be used when multiple xSeries 440s are interconnected and partitioned into smaller systems. The SMP Expansion Ports support a new messaging technique called Inter-Process Communications (IPC).

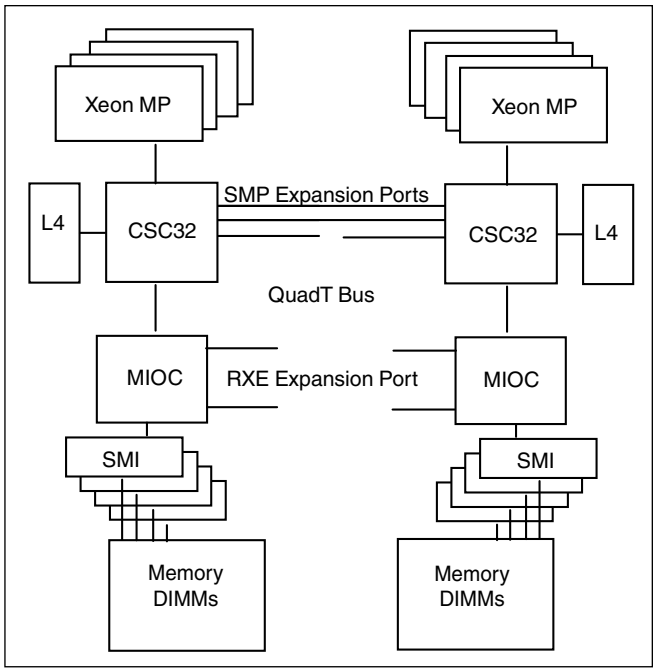


Figure 3 xSeries 440 IBM XA-32 Chip System

IPF Enhanced Node Prototype

The IBM Enterprise X-Architecture family product which supports the Intel Itanium™ processor family is the IPF Enhanced Node Prototype (IPF Node) platform.

IPF Node is a 4-EIA rack-optimized product with performance and scalability as its main goal. The system includes:

- 4 Intel Itanium™ family processors
- 64MB of XceL4 Server Accelerator Cache
- Up to 28 DIMMs for system memory
- 6 PCI-X slots
- SMP Expansion Port connections to interconnect to other IPF Node systems to scale to 16-way
- RXE Expansion Ports to connect to remote I/O drawers (IBM RXE-100 Remote Expansion Enclosure drawers)

Alone, the IPF Node is a complete system with up to four Intel Itanium™ family processors. The XA-64 components in IPF Node connect the four Intel Itanium™ family processors with 64 MB of XceL4 Server Accelerator Cache for optimal performance, and 28 DIMMs of DDR DRAM.

The IPF Node SMP Expansion Module has SMP Expansion Port connections to attach to other IPF Node SMP Expansion Modules. By interconnecting SMP Expansion Modules, the XA-64 chipset can scale from a 4-way to a 16-way system. The IPF Node package

externalizes the SMP Expansion Port connections such that it can be connected up to three other IPF Node systems.

There is a complete I/O subsystem supporting six PCI-X slots in each IPF Node system drawer. To configure more than six PCI-X slots, IPF Node also externalizes two RXE Expansion Port connections. Each RXE Expansion Port connection can be attached to an IBM RXE-100 Remote Expansion Enclosure drawer containing up to 12 PCI-X slots. For further expansion, any IBM RXE-100 Remote Expansion Enclosure drawer can cascade to other IBM RXE-100 Remote Expansion Enclosure drawers.

To further take advantage of the XpandOnDemand scalability of the interconnected IPF Node SMP Expansion Modules, it is possible to configure multiple IPF Node SMP Expansion Modules into one SMP server or to configure them as unique SMP servers. The XA-64 chipset supports the concurrent addition or removal of the SMP Expansion Modules.

Like xSeries 440, IPF Node builds upon the MIOC chip. Unlike xSeries 440, which supports 16 DIMMs on MIOC, IPF Node configures up to 28 DIMMs for the increased performance capability of Intel Itanium™ processor family systems. As shown in Figure 1 the Cache / Scalability Controller 64 (CSC64) chip is connected to MIOC. CSC64 connects to the QuadT interface of MIOC.

In addition to the QuadT bus interface to MIOC, CSC64 has a full system bus interface to connect to four Intel Itanium™ family processors, a XceL4 Server Accelerator Cache interface and three SMP expansion port interfaces.

The XceL4 cache on CSC64 supports a 64MB cache built from five DDR DRAM chips and supports the full 6.4GB/s bandwidth of the processor system bus.

The three SMP expansion ports can be connected to three other CSC64 chips. Each SMP expansion port can transfer data at 1.6GB/s in each direction simultaneously. A novel circuit driver technique, simultaneous bi-directional signaling, allows data to travel in both directions on the same wire at the same time at very high speed.

The SMP expansion ports not only interconnect CSC64s to form larger SMPs, but can also be used when multiple IPF Nodes are interconnected but partitioned into smaller systems. The SMP expansion ports support a new messaging technique called Inter-Process Communications (IPC).

RXE-100 Remote Expansion Enclosure

One of the most innovative features of the Enterprise X-Architecture XA-32 and XA-64 chipsets is Remote Expansion Port based I/O. With this technology, PCI-X adapters can be added to a system via a separate drawer. Not only does this allow for a more cost-effective system drawer, it further offers the capability to add more PCI-X slots to a system than would normally be put in the base system drawer.

IBM RXE-100 Remote Expansion Enclosure is a 3 EIA rack-optimized product with I/O scalability and expansion as its main goal. Each RXE-100 Remote Expansion drawer contains:

- Up to 12 PCI-X slots, 6 slots standard
- Remote I/O connections to connect to system drawers and to other IBM RXE-100 Remote Expansion Drawers.

IBM RXE-100 Remote Expansion Enclosure takes advantage of the flexibility designed into the PCI-X IOB chip used in each of the XA-32 or XA-64 based systems. The PCI-X IOB chip can be used (without a MIOC chip) standalone in a drawer to create an I/O-only drawer.

Figure 4 shows a block diagram of the IBM RXE-100 Remote Expansion Enclosure drawer. The RXE Expansion Port cables externalized by xSeries 360, xSeries 440 and IPF Node are brought out for the purpose of expanding the I/O capability of each of these systems.

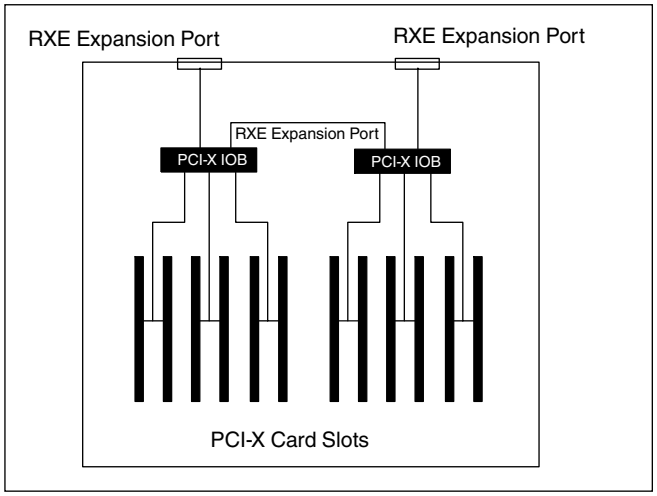


Figure 4 RXE-100 Structure

IBM RXE-100 Remote Expansion Enclosure brings in one Remote I/O cable and routes it first to one PCI-X IOB then another PCI-X IOB, which then externalizes to a second

Remote I/O cable for connection to more IBM RXE-100 Remote Expansion Enclosure drawers. The second cable can be looped back to the system to provide redundancy if a cable fault were to occur.

Each PCI-X IOB chip has two RXE I/O interfaces and three PCI-X busses. In IBM RXE-100 Remote Expansion Enclosure, each PCI-X IOB supports a "6-pack" of six PCI-X slots. Each of the three PCI-X busses is dedicated to two PCI-X slots.

Conclusion

IBM has succeeded in bringing mainframe features to the Intel® server marketplace. Several patent-pending, revolutionary innovations propelled the XA-32 and XA-64 chipsets to provide groundbreaking scalability, partitioning, performance, and RAS features. These features are offered on a line of products that support both the Intel Xeon Processor MP and Itanium processors, thereby enabling a smooth transition from 32-bit to 64-bit architectures.

References

1. L. Blackmon, B. Drehmel, T. Greenfield, J. Kirscht, J. Marcella, D. Shedivy, *EXA Memory/IO Controller*, IBM Enterprise X-Architecture Technology, p.9
2. T. Moe, C. Paynton, R. Shearer, S. Willenborg, C. Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31
3. J. M. Borkenhagen, R. D. Hoover, K. M. Valk, *EXA Cache/Scalability Controllers*, IBM Enterprise X-Architecture Technology, p.37
4. J. Bozek, *Partitioning in the EXA Technology*, IBM Enterprise X-Architecture Technology, p.121
5. D. Hurlimann, Y. Ku, H. Schultze, T. Tam, C. Walter, L. Wilson, *xSeries 360, IA-32 Basic Node*, IBM Enterprise X-Architecture Technology, p.71
6. M. Bland, R. Kolvick, *xSeries 440, IA-32 Enhanced Node*, IBM Enterprise X-Architecture Technology, p.77
7. J. Hanna, *IPF Enhanced Node Prototype*, IBM Enterprise X-Architecture Technology, p.85
8. J. Haidinyak, N. Sherali, *xSeries RXE-100 Remote Expansion Enclosure*, IBM Enterprise X-Architecture Technology, p.93

EXA Memory/IO Controller

– Lee Blackmon, Bob Drehmel, Todd Greenfield, Joe Kirscht, Jim Marcella, Dave Shedivy

Introduction

The Memory/IO Controller (MIOC) is a key component of the Enterprise X-Architecture™ building block structure, enabling multiple system configurations, processor scalability options, and enhanced memory reliability, availability, and serviceability (RAS) features. By itself, the MIOC enables a basic one to 4-way Intel Xeon™ Processor MP Family system. Enhanced one to 16-way Intel Xeon™ Processor MP Family and Intel Itanium™ Processor Family systems are enabled when MIOC is configured with Cache/Scalability Controller 32 (CSC32) or Cache/Scalability Controller 64 (CSC64) chips. System I/O is attached via I/O Bridge[1] (IOB) chips with up to two high speed RXE Expansion Ports. Refer to the Enterprise X-Architecture™ Technology Overview[2] article for a summary of system configurations. MIOC supports both single data rate (SDR) and double data rate (DDR) SDRAM memory and integrates several enhanced main memory RAS features, such as chipkill ECC, memory scrubbing, memory mirroring, and redundant bit steering. This paper describes MIOC's functional characteristics and provides detail on the specific implementation.

System Interfaces

MIOC has three main system interfaces, Processor, Memory, and RXE Expansion Port, as shown in Figure 1.

Processor Interface

The Processor Interface provides a dual function depending on the system topology. For Basic systems, with MIOC directly connected to Intel Xeon™ Processor MP Family processors, this interface provides full support for the system bus protocol as defined by Intel®. MIOC provides full multiprocessor support for up to four Intel Xeon™ Processor MP Family processors.

MIOC performs the role of the central and responding agents. The Intel Xeon™ Processor MP Family system address bus is a 36-bit double-pumped source synchronous bus with parity protection. The maximum bus rate is one transaction every two bus cycles. The Intel Xeon™ Processor MP Family system data bus is a 64-bit quad-pumped source synchronous bus also with parity protection. The cache line size is 64 bytes.

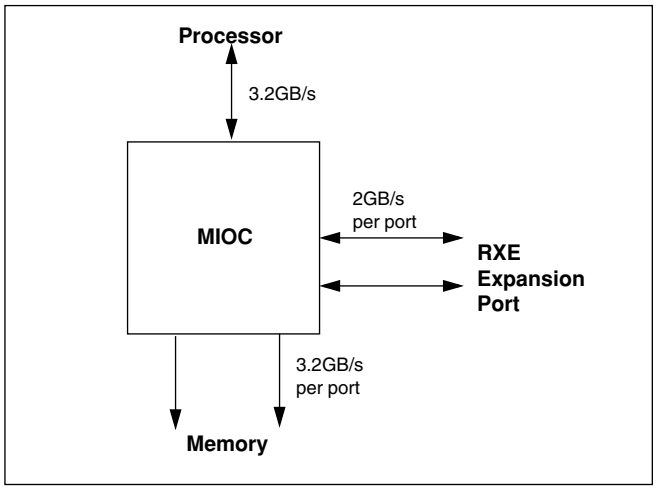


Figure 1 MIOC Interfaces

For Enhanced systems, with MIOC connected to CSC32 or CSC64 chips, this interface is referred to as the QuadT bus. The QuadT bus interfaces to CSC32 or CSC64 chips. In general, MIOC maintains the role of the central agent and CSC32 or CSC64 act as a symmetric processor. The QuadT bus extends the address bus to 44 bits and provides support for both 64 and 128 byte cache lines.

Memory Interface

The memory interface provides a dual function depending on the system topology. DDR SDRAM DIMMs may be directly connected to MIOC. Both SDR and DDR SDRAM DIMMs may be connected to MIOC via Synchronous Memory Interface (SMI) chips. For direct attached DDR DIMMs, the control interface operates at 100MHz. The 16-byte wide data bus operates at a frequency of 200MHz. For configurations with SMI, the control bus operates at 200MHz and the 8-byte data bus operates at 400MHz. For SMI configurations, MIOC supports two independent 8-byte data busses.

RXE Expansion Port Interface

The RXE Expansion Port is a scalable high speed point-to-point interface intended for low latency high bandwidth coupling of I/O buses. The RXE Expansion Port link architecture operates to yield a maximum bandwidth of 2GB/s. Switching bridge nodes may be used to provide connection to a larger number of bridges. The RXE Expansion Port operates in two modes. The first consists

of two unidirectional differential 10 bit busses. Each bus consists of eight data bits,one clock bit and one flag bit. The second mode operates as a 16-bit bidirectional interface with 16 data bidirectional data bits, two unidirectional clock bits and two unidirectional flag bits.

Technology

The MIOC chip is built with IBM's CMOS 7SF SA27E copper technology. Table 1 lists the technology attributes of MIOC.

Table 1 MIOC Technology Attributes

Attribute	MIOC
Technology	IBM CMOS7SF 0.18 micron
Chip Size	11.1mm x 11.1mm
# Signal I/O	692
# Transistors	25M
Substrate	Ceramic
Substrate Size	42mm x 42mm
I/O Pitch	1.27mm

The MIOC chip is composed of four major logic components: Memory, Coherency Unit (CU), RXE Expansion Port, and Interrupt. The following sections provide detailed descriptions of the function, RAS, configurability, and scalability features of each of these components.

MIOC Memory Subsystem

Table 2 shows a summary of the memory features supported by the MIOC. MIOC's memory control function consists of several partitions within the chip. Two major memory control logic components are the Memory Command Buffer and the SDRAM Array Controller as shown in Figure 2. The Command Buffer receives commands from other logic units, reorders these commands for efficiency, and dispatches them to the Array Controller. The Array Controller, which consists of several state machines per memory port, generates the control signals to the SDRAMs and the interface to the data flow logic. The Scrub logic generates and paces memory scrub commands to the Command Buffer. The Refresh logic controls the frequency and sequencing of the SDRAM refresh. The memory data flow contains partitions responsible for checking/generating ECC to/from memory, performing redundant bit steering, and generating statistics on memory scrubbing as shown in Figure 3. The data flow is also responsible for reading/writing the data buffer.

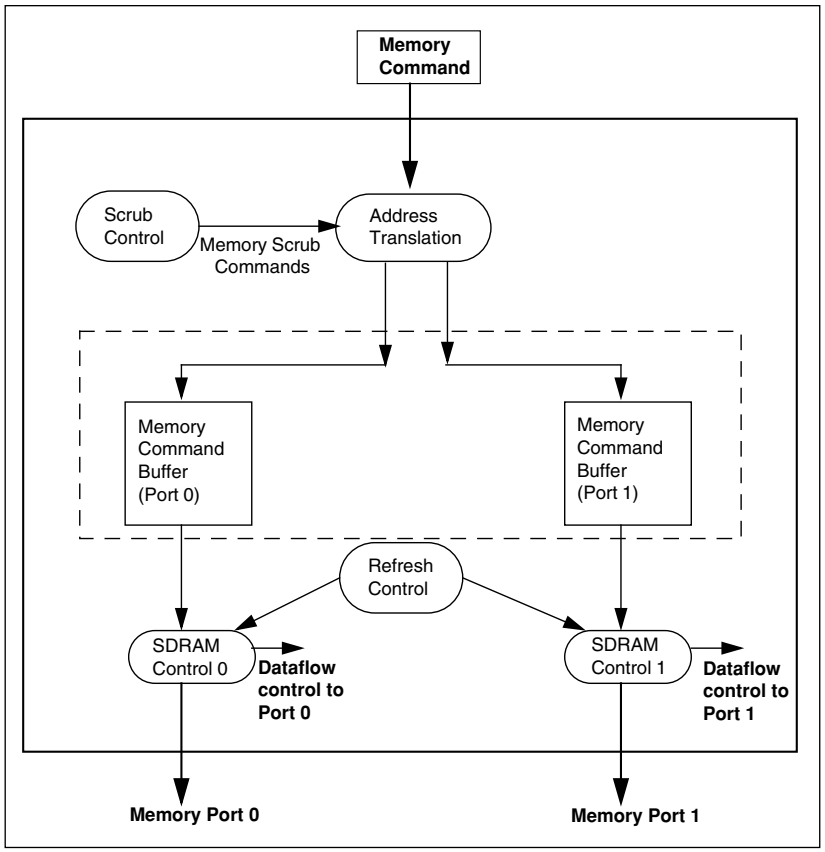


Figure 2 Memory Control High Level Flow

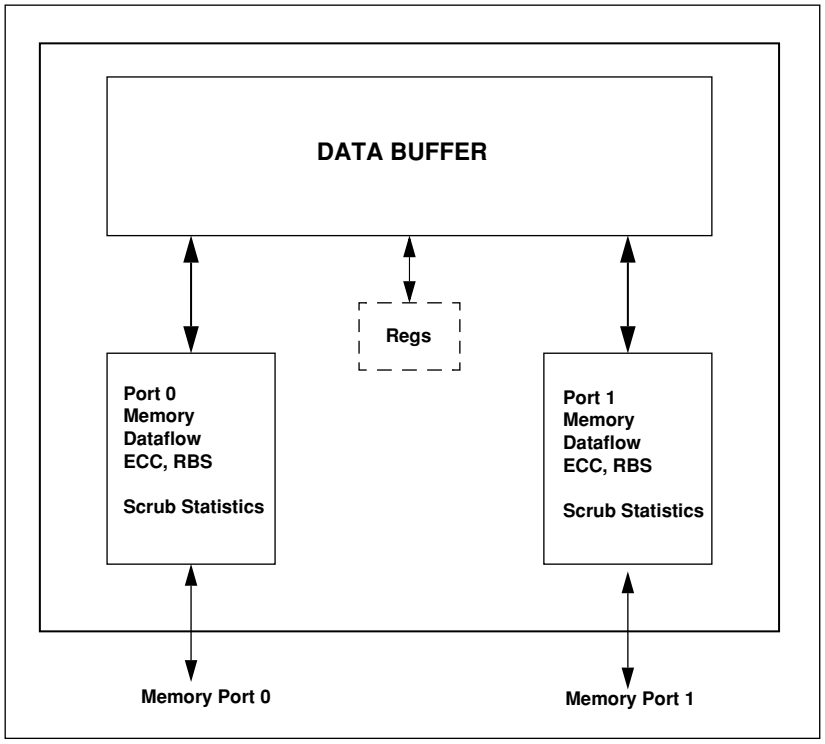


Figure 3 Memory Data High Level Flow

Table 2 Memory Features Summary

Command Queues	• 16 Read and 32 Write entries per port
Data Lengths	• 1-8, 16, 32, 64, and 128 bytes
System Topologies with SMI attached Memory	
# of Memory Ports	• 1 or 2-8 byte SDRAM ports
DIMM Type	• Industry Standard DIMMs
DIMM Plug Rules (SDR SDRAM)	• DIMMs plugged in groups of four
DIMM Plug Rules (DDR SDRAM)	• DIMMs plugged in groups of two
MIOC Control Interface Frequency	• 200MHz
MIOC Data Interface Frequency	• 400MHz
Control/data frequency between SMI and SDR SDRAM	• 100MHz
Control/data frequency between SMI and DDR SDRAM	• 100/200MHz
Port Interleave method	• cache line
System Topologies with Direct-Attach DDR Memory	
# of Memory Ports	• 1-16B DDR SDRAM port
DIMM Type	• Industry Standard DIMMs
DIMM Plug Rules	• DIMMs plugged in groups of two
MIOC Control Interface Frequency	• 100MHz
MIOC Data Interface Frequency	• 200MHz
SDRAMs	
Type	• SDR or DDR
CAS Latency	• 2 or 3(2.5 for DDR)
Trod	• 2 or 3
Timing Modes Supported	• 4/2, 5/2, or 6/3
Configuration	• X4 planar or stack
Technology	• 64, 128, and 256Mb
Technology con't.	• 512Mb and 1Gb based on availability
RAS Features	• Single ECC matrix with a 2 bit packet mode for SMI with SDR SDRAM, 4 bit packet mode for Direct Attach DDR, and 8 bit packet mode for SMI with DDR SDRAM • Single packet correct/double packet detect (SPC/DPD) ECC • RAS/CAS address parity embedded in ECC ChipKill coverage • Dynamic bit scrubbing with statistic gathering • Support for Memory Mirroring • Support for Hardware Initialization and Diagnostics
Memory ECC 2 bit Packet Mode Summary	• 144/132 – 2 bit packet ECC • 64-2 bit data packets • 6-2 bit check packets • 2-2 bit packets free for sparing
Memory ECC 4 bit Packet Mode Summary	• 144/132 – 4 bit packet ECC • 32-4 bit data packets • 4-4 bit check packets
Memory ECC 8 bit Packet Mode Summary	• 288 – 8 bit packet ECC • 32-8 bit data packets • 3-8 bit check packets • 1-8 bit packet free for sparing

Configurations and Modes of Operation

MIOC's memory controller supports two different interfaces: Direct Attach and SMI (Synchronous Memory Interface). In the Direct Attach topology, industry standard Double Data Rate (DDR) DIMMs attach directly to MIOC, plugged in groups of two. In the SMI topology, MIOC drives command/data to and receives data from SMI chips on a high speed Source Synchronous bus. Industry standard Single Data Rate (SDR) DIMMs, which are plugged in groups of four, or DDR DIMMs, which are plugged in pairs, attach to the SMI chips.

SDRAM Technology and Configurability

MIOC supports 64Mb, 128Mb, 256Mb, 512Mb, and 1Gb SDRAM technologies with x4 organization (i.e. four data pins). Different SDRAM technologies may be installed with the restriction that DIMMs within a plugging group (pair or quad, depending upon topology) must be of the same technology. When operating in mirrored mode, the DIMM plug/technology configuration must match identically between the two memory ports for the portion of memory that is mirrored.

Direct Attach Topology and Configurations

In the Direct Attach topology, MIOC supports a single 16 Byte DDR SDRAM memory port with capability to add up to 16 memory extents. The current system implementation supports up to four DDR DIMM pairs and, with stacked DRAM technology, utilizes eight of the 16 available memory extents. The minimum capacity is 256MB with 64Mb SDRAMs (2 DIMMs). With currently available 256Mb stacked SDRAM technology, the maximum capacity is 8GB (8 stacked DIMMs) with the capability to expand to 32GB when 1Gb SDRAM technology is available.

SMI Topology and Configurations

In the SMI topology, MIOC supports one or two 8 byte memory ports with the capability to add up to sixteen memory extents on each port. The current system implementation that supports SDR SDRAM technology supports up to two SDR DIMM quads per port. With stacked DRAM technology, four of the 16 available memory extents are used on each port. The minimum capacity for this system is 512MB with 64Mb SDR SDRAMs (4 DIMMs). With currently available 256Mb stacked SDRAM technology, the maximum capacity is 16GB (16 stacked DIMMs) with the capability to expand to 64GB when 1Gb SDRAM technology is available.

The current system implementation with SMI that supports up to sixteen DDR DIMM pairs per port. With stacked DRAM technology, all 16 available memory extents are used on each port. The minimum capacity for this system is 256MB with 64Mb DDR SDRAMs (2 DIMMs). With currently available 256Mb stacked SDRAM technology, the maximum capacity is 32GB (32 stacked DIMMs) with the capability to expand to 128GB when 1Gb SDRAM technology is available.

SDRAM Modes Supported

MIOC supports both Cas Latency 2 and 3 with SDR SDRAMs and Cas Latency 2 and 2.5 with DDR SDRAMs. It also supports registered DIMMs operating in either registered or pass-through modes. Depending upon cache line size, SDRAM type, and system topology, MIOC supports data burst lengths of 2, 4, or 8.

Memory Hole

MIOC supports a single address hole in memory so as to utilize main memory that is mapped to I/O. Both hole ending address and hole size are configurable. The memory hole size is subtracted from any incoming address that is above the hole ending address to normalize the address to a contiguous DRAM memory address space.

Port Address Translation

With the countless combinations of plug configurations and SDRAM technologies, there exists three general scenarios. These scenarios are given below with a brief description as to how they are handled by the address translation logic.

- Single Port of Memory: The single port address maps are defined in Figure 4. It shows the Row, Column, and Internal Bank selects that are asserted. The chip selects are comprised of the next four consecutive address bits above the row address.
- Dual Port of Memory, Equal Port Sizes: The dual port address maps are also defined in Figure 4. It shows the Row, Column, and Internal Bank Selects that are asserted. As in the single port case, the chip selects are comprised of the next four consecutive address bits above the row address.
- Dual Port of Memory, Unequal Port Sizes: This uses a combination of both single and dual port addressing. The dual port memory map is used in the address space below the address boundary equal to two

times the amount of memory in the smaller port. The single port memory map is used for the address space above this boundary.

Memory Extent Configuration

The Port configuration register indicates the amount of memory installed in each port. This is a count of the number of 128MB units of memory for a 32-byte memory bus. When memory is configured, the largest extents of memory are configured to be located at the lowest addresses, the next largest extents above those, etc. This method guarantees that there will be no holes in the memory address map due to usage of non-homogeneous combinations of memory technology.

Single Port Address Map Example

SBANK	MS ADDRESS BUS						
	37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08			07	06	05	04 03 02 01 00
64x4 - SDrams	0	R11 - R0	C9 - C1	B1	B0	C0	
128x4 - SDrams	0	R11 - R0	C11,C9 - C1	B1	B0	C0	
256x4 - SDrams	0	R12 - R0	C11,C9 - C1	B1	B0	C0	
512x4 - SDrams	0	R12 - R0	C12,C11,C9 - C1	B1	B0	C0	
1024x4 - SDrams	0	R13 - R0	C12,C11,C9 - C1	B1	B0	C0	

Dual Port Address Map Example

DBANK	MS ADDRESS BUS						
	37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08			07	06	05	04 03 02 01 00
64x4 - SDrams	0	R11 - R0	C9 - C1	B1	B0	C0	
128x4 - SDrams	0	R11 - R0	C11,C9 - C1	B1	B0	C0	
256x4 - SDrams	0	R12 - R0	C11,C9 - C1	B1	B0	C0	
512x4 - SDrams	0	R12 - R0	C12,C11,C9 - C1	B1	B0	C0	
1024x4 - SDrams	0	R13 - R0	C12,C11,C9 - C1	B1	B0	C0	

Figure 4 Memory Address Map Examples

Memory Mirroring

MIOC supports memory mirroring when configured with two memory ports. When enabled, mirroring has two modes of operation:

1. Full mirroring in which the entire memory space is mirrored, or
2. Partial mirroring in which only a subset of memory is mirrored.

In either case, the mirrored region starts at address zero and extends upwards based on a programmable region size (128MB, 256 MB, 512MB, 1024MB, 2048MB, 4096MB, 8192MB, or all of memory).

To operate in mirrored mode, all memory in the mirrored region must be of equivalent SDRAM technology. The memory hole, if any, is subtracted from the address first. Mirroring is then performed on the translated address.

All stores are performed to both SDRAM ports for addresses in the mirrored region. Reads from the mirrored region can be configured in one of three ways:

1. All reads from the mirror region will come from either Port 0 or Port 1 as specified by a configuration bit which may be set by hardware. This is the default for mirrored mode and is used during a hot plug replace of memory on a given port.
2. Reads interleave between ports, independent of cache line. This improves performance by spreading accesses across two independent memory ports and minimizes command/data bus scheduling conflicts. This mode can be enabled by software, but must be disabled prior to doing a hot plug replace of memory on a given port.
3. In the mirror fail over mode, reads are issued synchronously to both ports. In this mode, if an uncorrectable

error occurs on one memory port, the valid data is used from the other memory port and the uncorrectable error is reported to software as a recoverable error. This mode can be enabled by software.

After performing a hot plug replace of memory on a given memory port, MIOC implements a copy function to copy data from the primary port to the replace port concurrently with processor and I/O commands. The interval between copy commands is configurable to allow trade off between copy time and impact to system performance during the copy.

When operating in partial mirrored mode, all operations which are in the non-mirrored region interleave across both ports.

Memory Command Buffering Logic

One of the major elements in the MIOC chip architecture is the Command Buffer. This logic resides in the control path for memory commands between the Memory Control Multiplexor (MCMUX) and the Array Control logic (ARCT). The Command Buffer uses the following techniques to maximize system performance and memory interface utilization:

- Dual Independent State Sequencers and Command Flow
- Deep Queuing Structure with Dynamic Update
- High Speed Coherency Management
- Command FastPath
- Speculative Read and Cancel
- High Priority Store
- Command Reordering
- Arbitration Orders and Watermark Mode

Dual Independent State Sequencers And Command Flow

The Command Buffer is architected as two completely independent State Sequencers and Command Flows to take full advantage of the dual memory interface capabilities of ARCT. Depending on memory addresses, commands are reordered before entering the Command Buffer. To increase efficiency the cache lines are interleaved between the two memory ports.

The Command Buffer takes advantage of the address mapping and provides a set of state sequencers and command flow for even address memory commands for port 0 and a separate set of sequencers and command flow for odd address memory commands for port 1. This reduces latency by allowing maximum overlap while processing the command stream.

Deep Queuing Structure With Dynamic Update

The Command Buffer provides a deep queuing structure to handle bursts of memory commands without throttling incoming processor and I/O commands. The Read Queue can accommodate up to 16 commands, while the Store Queue can manage up to 32 commands.

The queue structure is architected to provide streamlined management of other functions such as Address Dependency, Snoop Cancel, Store Data Availability, and Command Reordering. The handling of these functions within such a deep queue has been optimized to yield minimum latency.

A technique called Dynamic Update was developed to manage specialized function in logic separate from the queue. The resultant information is then matched to the associated queue entry and permitted to enter the queue at that entry location. The data is dynamically updated into the queue entry by being blended with the entry's feedback path, sometimes as the entry is being shifted in the queue. By providing certain information within the queue itself, management of the queue, in terms of identifying valid arbitration candidates, can be made with optimum performance.

High Speed Coherency Management

The Command Buffer maintains coherency by presenting commands to ARCT in a certain order relative to how they were received from MCMUX. The ordering requirement is required when an incoming command has an address dependency between any of store commands held in the 32-deep store queue. An address dependency exists when two commands have the same cache line address.

This address dependency is detected and managed using a specially designed Contents Addressable Memory (CAM) operating at 200MHz. This customized CAM design is highly tuned for performance, which allows the coherency to be managed while minimizing latency of the command flow path. The Command Buffer coherency logic is also architected to minimize the strictness of ordering rules such that opportunities are created to reorder memory commands. Reordering memory commands allows efficient use of the memory interface, increases memory throughput, and improves system performance. The coherency management logic implements the following reduced set of the Address Dependency Rules to reorder commands:

- Two read commands to the same cache line can be

presented to ARCT in any order relative to when they were received.

- Two store commands to the same cache line must be presented to ARCT in the order they were received into Command Buffer.
- A read command to the same cache line as a previously received store command must be presented to ARCT after that store command is presented.
- A store command to the same cache line as a previously received read command can be presented to ARCT in any order relative to that read command. No dependency exists between them.

Command Fastpath

The Command Buffer has a Command FastPath function that allows a read command to be received from the MCMUX and bypass the Command Buffer's deep queuing structure and coherency management logic and be presented to the ARCT interface in the next cycle. For example, if MCMUX presents a read to Command Buffer during cycle #1, then during cycle #2, Command Buffer will present this read to the Array Control logic.

The condition for allowing a FastPath case occurs when the Command Buffer does not have any valid commands (for that port) that need to be presented to ARCT. While the deep queuing structure allows the handling of command bursts efficiently, the FastPath function reduces read latency.

Speculative Read And Cancel

The Command Buffer receives, manages, and speculatively propagates processor reads to memory before a response has been received from processors on the system bus. If a response indicates that the read data is cached within a processor, the Command Buffer searches and aborts the read command if found. This cancel technique improves performance by reducing the number of speculative reads and improving the SDRAM bus utilization. By speculatively forwarding reads to memory, the latency is improved.

High Priority Store

The Command Buffer implements a High Priority Store function. This function is designed to improve the latency of any read command that has an Address Dependency on a previously received store command pending in the Command Buffer. A dependency exists when both commands have the same cache line address. When this condition is detected, the Command Buffer promotes the priority of the store to High Priority. The arbitration logic within the Command Buffer selects the store command from this virtual High Priority queue during any open arbitration cycle. High Priority stores win arbitration over any other class of memory command.

Any of the 32 store commands can be marked High Priority, arbitrated for, and pulled from the queue. By executing the High Priority store as soon as possible, the dependency on that store is removed, and the subsequent read to that cache line can then be considered valid as an arbitration candidate.

Command Reordering

When accessing an SDRAM, delay penalties can be incurred based on the previous access. A RAS precharge penalty can be incurred if the previous access was to the same internal bank select of the same SDRAM. A driver switching penalty can be incurred if the previous access was to a different SDRAM on the same data bus.

The Command Buffer analyzes the command addresses (chip selects and internal bank selects) and then reorders the commands to minimize these penalties. Reordering improves the SDRAM data bus utilization and subsequently improves system performance.

Command reordering is performed on both the read and store command queues. For a given command class, the reordering is performed on the oldest eight commands. If there is no advantage to reordering the commands, then the oldest valid command is selected.

The Command Buffer compares the command being presented (or the last presented) to the ARCT interface with the list of valid candidates within each command type. The following shows the order that is sought within a command type:

1. Oldest command that has a different chip select on a different data bus, or else has the same chip select and a different internal bank select.
2. Oldest command that has a different chip select on the same data bus.
3. Oldest command.

The Command Buffer reevaluates which command is the best candidate for each queue every cycle to maximize choices as commands are received.

Arbitration Ordering And Watermark Mode

The Command Buffer decides which command is to be presented next to the ARCT interface. The arbitration logic inside of the Command Buffer selects which command class will win the arbitration based on a priority order. There are three classes of commands: Read, Store, and High Priority store. The default priority order is: 1) High Priority Store, 2) Read, and 3) Store. When there are valid arbitration candidates from two or three of these command types, the above priority order is used.

To maximize performance, reads are given priority over stores unless a read has an address dependency on a previously received store. The store is promoted to a High Priority store. Since these cases are relatively rare, and because there is significant opportunity to reduce the latency of such reads, the Command Buffer offers the ability to place High Priority stores at the beginning of the priority order.

Besides the Default Priority Order, there is another Priority Order that can be used. This second Order is provided to allow performance tuning of the system. This tuning may vary due to the command mix affected by the system hardware configuration, the operating system, or the application. This second Order is intended to bias arbitration in favor of store commands. The Secondary Priority Order is: 1) High Priority Store, 2) Store, and 3) Read.

To provide the ability to tune performance of the memory interface, the arbitration logic can be put in a mode where it will alternate between the default Order and the secondary Order. This allows the biasing of arbitration to favor reads for X number of wins, and then switch the Order and favor stores for Y number of wins. This type of alternating Order enables the tuning of the arbitration to better match the command mix and thus improve system performance. The Command Buffer provides both static and dynamic enables for controlling the arbitration so that it alternates between the default and the secondary Orders.

The static enable forces the biasing to alternate continually, biasing toward reads for X wins, then biasing toward stores for Y wins, and repeating indefinitely.

The dynamic enable allows the arbitration Orders to alternate while certain conditions are met and then return to the default Order until the conditions are met to return to alternating arbitration mode. The dynamic enable is handled by using the Watermark function. This function monitors the number of stores currently held inside the Command Buffer for a given port. When this level reaches a programmable value (High Watermark), the alternating arbitration mode is entered. When the number of stores drops to another programmable level (Low Watermark), then the arbitration returns to the default Order.

The dynamic enable can also be configured to activate by the presence of any High Priority Store commands.

Together, the many ways of biasing the arbitration allows the Command Buffer to be tuned to provide the best system performance.

SDRAM Array Controller Logic

Another major element within the MIOC Chip architecture is the Array Controller (ARCT). This logic resides in the control path for memory commands between the Memory Command Buffer and the external interface to the SDRAMs. The Array Controller uses the following techniques to maximize system performance and memory interface utilization:

- Dual Independent Memory Ports (memory interleaving)
- Multiple State Sequencers (memory command overlap)
- Dynamic Array Command/Data bus scheduling
- Command FastPath
- Bursting Refreshes
- Fast Memory Initialization
- Fast Hardware Diagnostics
- Memory Display/Alter

Dual Independent Memory Ports

MIOC implements two independent memory ports to support memory interleaving. Each port has its own set of state sequencers that are sourced by commands from port dedicated command buffering logic. Two ports improve sustained latency by allowing overlap of even and odd cache line address commands with no command or data bus scheduling conflicts.

Multiple State Sequencers

MIOC has also implemented five state sequencers for each memory port. These sequencers allow overlapping of commands on the same port by communicating state information between them. They operate in a round robin fashion. If a command is presented from the Command Buffer to the Array Controller and the next sequencer is available, the Array Controller immediately processes the

command. The Array Controller then looks for the next command from Command Buffer and assigns it to the next sequencer, and so on.

An option exists to configure a single sequencer per port, thus single threading all commands. This is useful for laboratory debug as well as stress testing the command queuing logic during hardware verification.

Dynamic Array Command /Data Bus Scheduling

After receipt of a command from Command Buffering logic, the Array controller determines if there are any DRAM bank address conflicts with up to four other commands that are currently in progress. If such conflicts exist, the command must wait until the previous conflicting command has completed. (Command Buffer tries to minimize such penalties by command reordering prior to presenting commands to the array controller). If no conflict exists with a prior command in progress, the array controller, knowing the state information from the other four sequencers, determines when the command bus is available to present the row address command to the DRAMs to open the bank. Based upon the fixed relationship between the column address command and data for reads and writes and the state information from the other four sequencers, ARCT determines when both the command and data bus are available to issue the column read or column write command. This is done dynamically for any combination of commands currently in progress.

Command Fastpath

In the event that there are no commands queued within the command buffering logic, then commands fast path around the command buffer directly to the array controller on a dedicated interface. This reduces latency to commands that go through the command buffering logic. This fast path can be enabled/disabled by software.

If there are no commands queued within the command buffering logic and there are no commands pending within the coherency unit, memory read commands fast path directly from the system bus interface logic to the array controller on a dedicated interface, bypassing both the coherency unit logic and the command buffering logic. This fast path can be enabled/disabled by software. These two fast paths are mutually exclusive. Only one can be enabled.

Bursting Refreshes

The DRAM refresh function is required to maintain charge storage within the DRAM chips. All commands active to a given SDRAM are completed and all banks closed before refresh is issued. MIOC completes all commands currently in progress before allowing a refresh command to be issued. There are up to 16 memory extents per port, all of which are refreshed back to back. MIOC also supports bursting of up to 16 refreshes for a given memory extent, thus refreshing on much greater intervals and reducing the overhead of having to flush the Array Control sequencers. The number of consecutive refreshes performed is configurable by software. Most DRAM manufacturers now support bursting of up to eight consecutive refreshes.

Fast Memory Initialization

MIOC also provides a means for the hardware to initialize the SDRAMs with a data pattern and good ECC. Depending upon configuration, this function overlaps commands in such a way as to keep the SDRAM data busses as efficiently utilized as possible. Initialization starts in memory extent zero and increments through all memory extents until all addresses have been written. After all addresses have been written, a special interrupt is issued to software to indicate that the memory is ready for use. Upon completion of the fast card initialization, an option exists to perform a fast scrub which reads all memory addresses and checks for good ECC. This option is great for performing memory diagnostics quickly during the initial program load procedure.

Fast Hardware Diagnostics

An enhanced mode for fast memory initialization exists to allow isolation of Uncorrectable Errors. Instead of writing a fixed pattern into memory, a pseudo random shifting data pattern with good ECC is written into the SDRAMs. The same registers used to specify the data pattern for Fast Memory Initialization are used as a starting seed. After having initialized the memory with the pseudo random shifting pattern, the data pattern registers are restored to the original starting seed. A Fast Scrub is then performed. In this enhanced mode, the hardware determines the expect data from the pseudo random pattern generator and compares it with data received from the SDRAMs bit by bit. This allows better failure isolation than using ECC.

Memory Display/alter

MIOC supports a register interface to display and alter contents of a memory cache line. This feature is very useful in the laboratory and a performance enhancement for performing memory diagnostics. Current methods use the processor to perform write/read/compares four bytes at a time. With larger memory spaces, the length of time to do this is not tolerable. Also, if ECC is reduced to parity coverage on the system bus, the ECC bits do not get checked with this method. With Display/Alter, memory can be diagnosed cache line at a time using the hardware to flag ECC errors. The processor streams a sequence of register write commands, one for each address location.

Memory RAS Features

Advances in computer systems and memory technology have resulted in very large main storage capacities. Along with increased size, main storage must be more reliable, operate for longer intervals between initial program loads (IPLs), and take less time at IPL to test and initialize. One of the key contributors to main storage subsystem reliability is failing memory modules. To improve reliability, the system needs to be designed to tolerate these module failures. One method to improve main storage reliability is to use an error correction code (ECC) on the data in conjunction with a diagnostic process or processes to remove failing memory modules from use before the failure progresses to the point of an uncorrectable error (UE). An uncorrectable memory failure is catastrophic and results in system down time and lost data. Due to the increased interval between IPLs, the memory diagnostic process must execute during runtime and not interfere with normal activity. It must also place as small a burden as possible on the system so as to not interfere with the system's main function, executing user programs.

The comprehensive main storage diagnostic process implemented by the MIOC chip consists of the following major components:

- ECC
- Scrubbing
- Scrub Statistics Gathering
- Real Time Redundant Symbol Steering
- IPL Diagnostics

The following sections describe these techniques and how they build upon one another and work together in the MIOC chip to increase the main storage subsystem reliability.

ECC

Error correction codes (ECC) are used to detect and/or correct data errors. They are simply parity schemes where the pattern of parity errors indicate an error and point to the bit or bits that are incorrect. The simplest form of ECC is parity. Parity is a single error detect (SED) ECC scheme. As the requirements for the ECC algorithm increase to double error detect (DED) or single error correct (SEC) or further, the number of parity bits required also increases. When the scheme is greater than parity, these bits are called check bits. In general, these schemes correct and detect a certain number of errors in an ECC word. An example is a 'single error correct/double error detect' (SEC/DED) scheme. ECC schemes can be expanded to cover groups of bits. This group or packet of bits is called a symbol. This type of ECC is capable of correcting as many errors as are in a symbol, but they must be contained within only one symbol. An example is a 'single packet correct/double packet detect' scheme. Another way of looking at this is that the common SEC/DED code has a symbol size of one.

The number of ECC algorithms and their complexity is infinite. The MIOC chip design team made simplifying assumptions to come up with a scheme that was practical. Tradeoffs were made in picking the ECC algorithm best suited for the memory configurations to be supported. In general, the cost of ECC goes up with increased symbol size and goes down with increased ECC word data width. It is the most economical to lower the number of bits required to be corrected in as wide of an ECC word as possible.

In the discussion that follows, there are references to the ability to correct a chipkill failure. This simply means that if a DRAM fails, all the data received from that DRAM can be corrected and the system continues to run with no loss of data.

The MIOC chip supports several memory configurations and has implemented multiple ECC schemes. The direct attach interface, which is designed to interface with Double Data Rate (DDR) Synchronous DRAMs, is a 16 byte interface. Each 64-byte cache line access consists of a burst of four 16-byte data transfers, each flows through a 16 Byte wide Single Packet Correct/ Double Packet Detect ECC tree. Since x4 DRAMs are used (i.e. four data bits per DRAM), the packet size for the ECC code chosen is a 4-bit symbol. This allows for data correction even in the event of a full DRAM chipkill. The number of check bits needed for a 16-byte data width

and 4-bit symbol size is 16. Two DDR DIMMs supplies 144 bits for 128 data plus 16 check bits, leaving no extras for support of redundant bit steering.

The elastic memory interface, which is a source synchronous interface for high speed data transfer between MIOC and SMI re-power chips, is an 8-byte interface. MIOC has implemented two such interfaces. The SMI chips may receive a cache line of data (64 or 128 Bytes) from either four Single Data Rate (SDR) DIMMs running at 100MHz (i.e. 32 Bytes every 10 nanoseconds) or two Double Data Rate (DDR) DIMMs running at 200MHz (i.e. 16 Bytes every five nanoseconds). The data is then transferred back to MIOC on the elastic memory interface at a rate of eight bytes every 2.5 nanoseconds.

If using SDR x4 DRAM technology, each 32-byte SDR SDRAM access results in four 8-byte transfers on the elastic memory interface. The first 8-byte transfer consists of data from only one data (DQ) pin from each DRAM. The second 8-byte transfer consists of data from another data pin from each DRAM. The two transfers are stitched together such that data originating from two pins of the same DRAM and transferred across the same physical board wire reside within the same two bit symbol. This allows for data correction even in the event of a full DRAM chipkill or defect in the board wire between MIOC and SMI. The data flows through a 16-byte wide Single Packet Correct/ Double Packet Detect ECC tree. Each subsequent two transfers get stitched together and flow through the same ECC tree. The number of check bits needed for a 16-byte data width and 2-bit symbol size is only 12, leaving four bits for support of redundant bit steering.

If using DDR x4 DRAM technology, each 16-byte DDR SDRAM access results in two 8-byte transfers on the elastic memory interface. Since all four data pins of every DRAM end up in the same 16-byte ECC word, a 4-bit symbol would be needed to cover the chipkill scenario. As stated in the Direct Attach DDR scenario, this requires 16 check bits, leaving no additional bits for support of redundant bit steering. Because this implementation is directed at larger systems which would require higher reliability, a 32-byte ECC tree was also implemented at the expense of one additional cycle of latency. Every two 16-byte DDR SDRAM accesses now result in four 8-byte transfers on the elastic memory interface. Within this 32-bytes of data, each DRAM has sourced eight bits of the data. The eight bits of data are grouped into the same symbol and flow through the 32-byte ECC tree. The number of check bits needed for a 32-byte data width and 8-bit symbol size is only 24, leaving eight bits (one symbol) for support of redundant bit steering.

MIOC also makes use of 'virtual' ECC bits to add coverage to control lines, such as in DRAM address parity. For many ECC codes, the number of check bits required for a power of two data width actually covers more bits. Such is the case for the ECC codes implemented in MIOC. For each store, the MIOC chip determines the address parity of both the row and column addresses of the DRAM address. Those addresses are treated as data bits when generating ECC, but only the data and ECC bits are stored to memory. For subsequent access of the same address, the row and column address parity is again calculated (address is known). Upon receipt of the data, the calculated address parity bits are appended to the data prior to being fed through the ECC tree. If there is a stuck or open fault on one of the address lines, there is 75% chance that the data returned will be from an address with different address parity and the error will be reported.

Scrubbing

Main storage scrubbing is a method to retain the integrity of data in dynamic random access memory (DRAM) modules. Alpha radiation or other failure mechanisms can cause a DRAM cell to have a value different than that which was originally written. Scrubbing consists of a read to the DRAM, error correction if appropriate, and store back to the DRAM. If an uncorrectable error is detected by the scrub read, the data is written back unaltered.

Scrubbing is primarily used to fix intermittent or soft errors. If the DRAM cell is permanently damaged or the error is hard, the failure will always exist. Soft errors that accumulate over time have the potential to line up with a hard error in an ECC word to cause a uncorrectable error (UE). Scrubbing provides a preventative function for UE's. The soft errors are fixed before they contribute to forming multiple bit errors that cannot be corrected. The entire memory space could be scrubbed once a day or as seldom as once a week. The key is to scrub often enough to fix the soft errors before they accumulate.

In MIOC, main storage scrubbing is implemented in the hardware and enabled by software. In this implementation, two scrub controllers exist in the memory subsystem, one per memory port. A counter within each scrub controller counts internal clock cycles. When the software defined interval is reached, the controller initiates a request for a scrub. The scrub command is injected into

the normal flow of commands such that it may utilize command reordering to determine when the best possible time is to execute (i.e. minimize penalties due to conflicts with normal commands and preserve priority for read commands). The interval value is variable and chosen such that the entire memory space is scrubbed once a day for a given memory size. Once a scrub request has been issued, the counter is reset to its initial value and counting starts over.

The scrub operation itself is performed on a cache line. The following procedure describes the general methodology of how scrubbing works:

1. Read a cache line from memory
2. Correct any correctable errors
3. Write the corrected data back to memory (if no UE)
4. If a UE is detected, write the data back unaltered
5. Increment the scrub address offset and wait for defined scrub interval
6. Repeat the process until all addresses in the extent are scrubbed
7. Continue scrubbing in next memory extent.

Several options exist in the scrub controller for flexibility and debug/verification capability. These options allow for more efficient testing of other implemented RAS features. Scrubbing may be started at any cache line offset within any memory extent. Options exist to scrub a single cache line, scrub all cache lines up to the end of a memory extent, or to continuously scrub all extents of memory in a round robin fashion. An additional option exists to scrub all of memory as fast as possible, throttled only by the hardware ability to execute the scrub commands. The option also exists to disable the store back of corrected data. This is useful in testing other RAS features.

MIOC supports a low power mode. This means that the DRAMs only have power and operate in a self refresh mode. Because the DRAMs are in self refresh mode, scrubbing does not resume until exit from low power mode. Upon exit of low power mode, scrubbing can be used to quickly remove any soft errors that have accumulated while in low power mode. This function is a key component of a system that has a low power mode and provides an integrity check on the data. This process is referred to as a “fast” scrub and is completed as rapidly as possible before any reads or stores are sent from the processor to MIOC.

Scrub Statistic Gathering

Scrubbing can be used as a stand-alone feature to improve the reliability of the system’s main memory. It can also be used to predict uncorrectable memory failures before they happen. In general, uncorrectable errors are created when a group of correctable errors are present within the same ECC word. By keeping track of correctable errors, the ability exists to take action before another error accumulates that may align with it.

In MIOC, scrub statistics gathering is implemented in the hardware and enabled by software. While scrubbing an extent of memory, all single symbol errors are logged in an internal low power register array. This gives an accurate representation of total errors within the extent. If the error count for any symbol exceeds programmable threshold, an interrupt is generated.

The hardware maintains the captured information about the memory port, memory extent within the port, and symbol within the extent that exceeded the threshold. Software can access this information and take the appropriate action. If the error counts for all symbols remain below the threshold, the counts within the array are cleared and scrubbing continues in the next memory extent.

MIOC keeps track of single symbol errors for scrub commands only to ensure that the counts within an extent are accurate. If statistics were gathered for all memory accesses, misleading information could accrue if the software gets into a tight loop that repeatedly reads from a location that has an error.

A hardware block diagram of Scrub Statistics Gathering is shown in Figure 5.

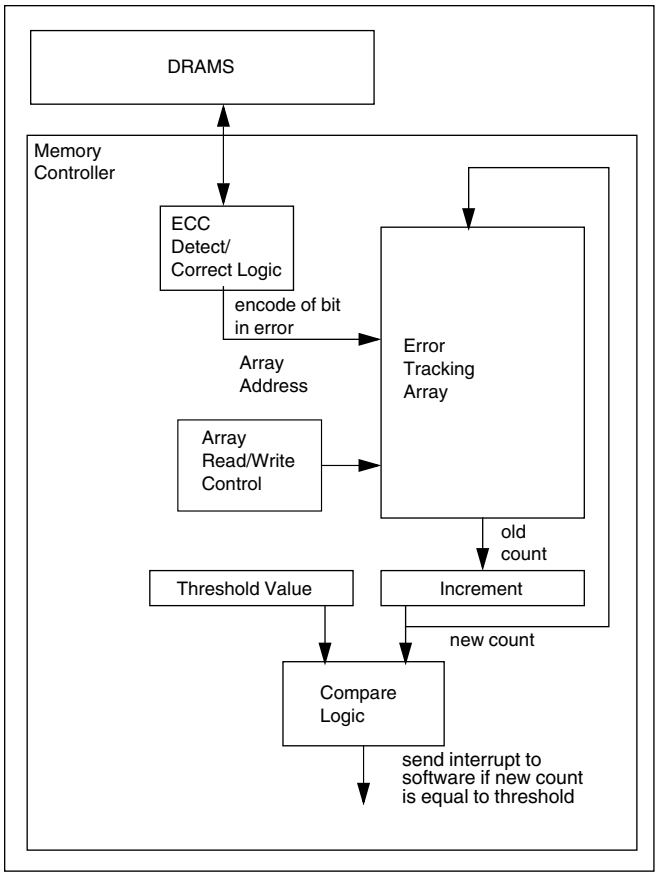


Figure 5 Scrub Statistics Gathering Overview/Hardware

Real Time Redundant Symbol Steering

With scrubbing and scrub statistics working together, the reliability is improved and situations can by identified where an uncorrectable error is likely. The next question is what action needs to be taken when an uncorrectable error is likely. In some systems, a portion of the memory address space can be marked by software so it is not used by the operating system. For example, the address space may be unallocated in units of 4K pages. A second level threshold is usually enforced such that when enough pages are unallocated, then the entire memory extent is unallocated and called out for replacement. The software sends the appropriate messages to the system operator. The main objective of this action is to prevent an uncorrectable error from occurring.

MIOC has implemented a method by which the logic may switch in unused DRAM bits real time to replace the symbol that has been flagged by scrub statistics gathering. This function is enabled by software by writing to a control register that specifies the symbol to be switched/steered. There exists one control register for each extent of memory in each port such that any symbol of any extent, and any combination thereof, may be steered. This action prolongs the need to unallocate or replace a memory extent.

There are two major ways of implementing a redundant symbol steering function. The first is a “split and shift” scheme by which the ECC word is “split” at the defective symbol position and, for writes, all bits to the right of that symbol are shifted by the width of one symbol. The rightmost symbol bits shift into the spare DRAM bits. For reads, all data to the right of the defective symbol are shifted left prior to forwarding to the ECC logic. This is easier to implement in the hardware from a chip timing perspective (uses “2 to 1” multiplexors), but is more difficult to transition to the steered state real time because an address boundary must be maintained to know which addresses have or have not already been steered.

MIOC has implemented a second scheme that provides large multiplexors to allow the spare symbol to be substituted for any symbol rather than just the adjacent symbol as done in the “split and shift” method. This is more difficult to implement in the hardware from a chip timing perspective, but lends itself to an easier, faster way of steering. During the process of changing over to the steered DRAMs (referred to as “Clean Up”), the scrub function is used with the “clean up” option. During this time frame, the memory controller will perform all reads from the non steered location and perform all writes to the steered location. By performing the reads from the non steered location, unless there is a full chipkill, there is a higher probability that the data for any given ECC word will be correct and minimize exposure to alignment to an error in a different symbol. Also, since only one symbol is steered, the ECC logic can be used to correct the data for stores to the steered location rather than having to shift or move data for each access. Upon completion of the “clean up”, all reads and writes for that memory extent will be performed to the steered location. This process is transparent to processor and I/O activity.

Hardware block diagrams for an example 80-bit data width are provided in Figures 6-9 for the read and write paths for each method described above.

IPL Diagnostics

Hardware-based Initial Program Load (IPL) Diagnostics are another key factor to guarantee a reliable memory subsystem and reduce boot time. Since there are connectors between the memory DIMMs and the memory

control logic, testing is required to ensure that this path is sound between IPLs. The scope of these tests is limited to diagnose the most obvious failure mechanisms.

MIOC provides a methodology that uses unique data patterns for each 16 bytes of data. Under software control, the hardware writes, reads and compares these data patterns to memory. This scheme covers failures in all address and control bits and will detect most stuck or open data lines. The hardware implementation detects memory failures and is several orders of magnitude faster than any software implementation.

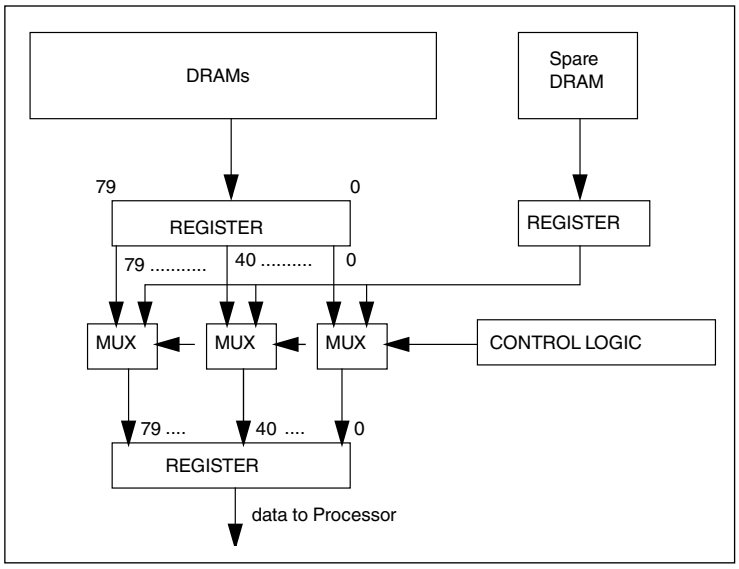


Figure 6 Steer Method Read Path/Hardware Block Diagram

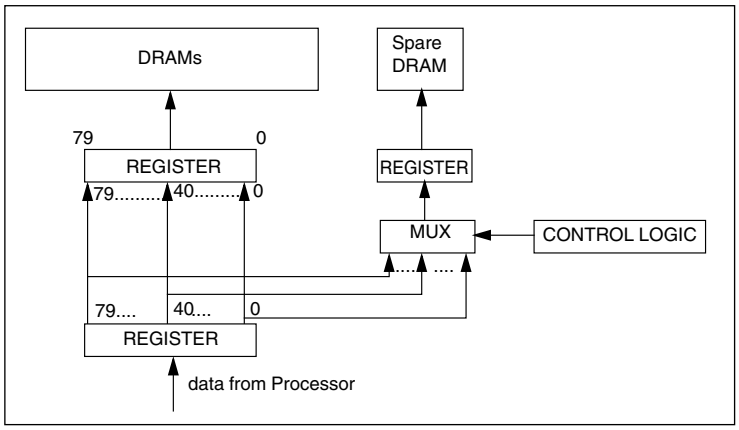


Figure 7 Steer Method Store Path/Hardware Block Diagram

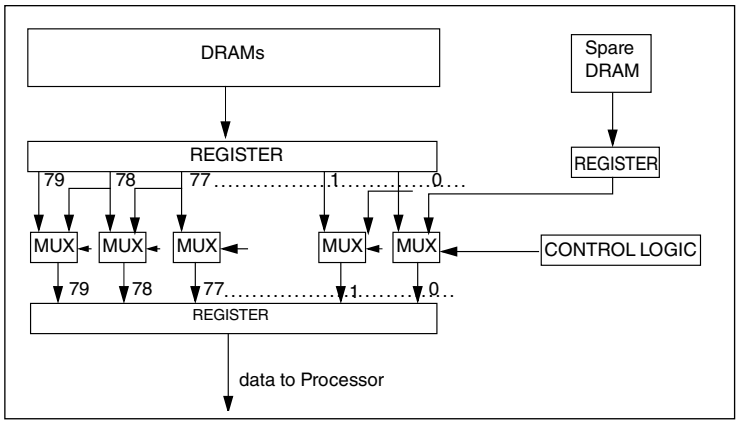


Figure 8 Shift Method Read Path/Hardware Block Diagram

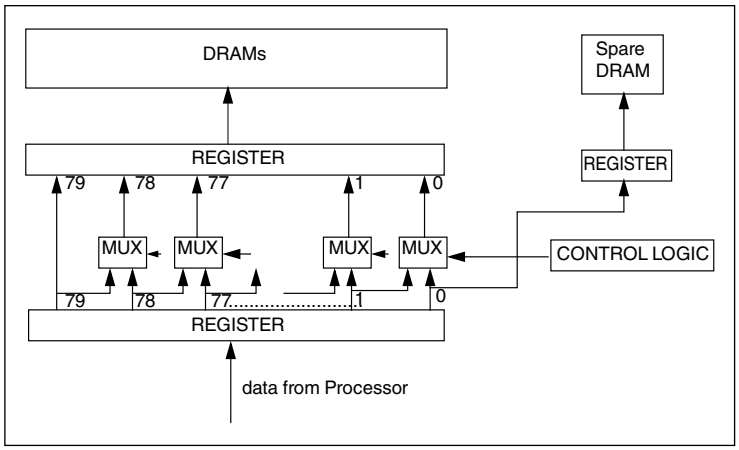


Figure 9 Shift Method Store Path/Hardware Block Diagram

Coherency Unit

The Coherency Unit (CU) in MIOC is responsible for accepting commands from the system bus, the RXE Expansion Port interface, and the internal interface (Interrupts) and transferring the commands to the appropriate unit to be processed. Commands received by the CU can be transferred to the Memory Unit, the RXE Expansion Port Interface, the Register Unit, the system bus, or the Interrupt Unit. The destination of the new command is determined by a set of registers containing routing information.

The CU is comprised of the Request Handler, the Pending Queue (PQ), the Request Generators, the Response Generators, the Response Handlers, and the Buffer Handler as shown in Figure 10.

The Request Handler accepts all new commands from the system bus, the RXE Expansion Port interface, and the internal Interrupt interface. System bus commands have the highest priority and are always accepted by the Request Handler as soon as they are presented. No acknowledge is sent back to the system bus logic. RXE Expansion Port commands and internal interrupts alternate priority with each other. These commands do not have to be accepted as soon as they are presented, so the Request Handler sends an acknowledge when it has accepted a command from one of these two interfaces.

Once a command has been accepted by the Request Handler, it moves through a series of stages. The first stage passes the command information to the PQ to check for address collisions with commands already in the PQ and to the I/O Directory to check for address collisions with DMA Read data that is cached in IOB. The second stage receives the address collision information from the PQ and uses it along with its own internal checking to determine if the command should be retried back to the presenting interface. This stage also uses the routing information registers to determine where the command needs to be sent in the chip. The third stage takes all the information from the original command and the previous stages and determines the initial values for all the fields in the PQ entry. This information is passed to the PQ in this stage.

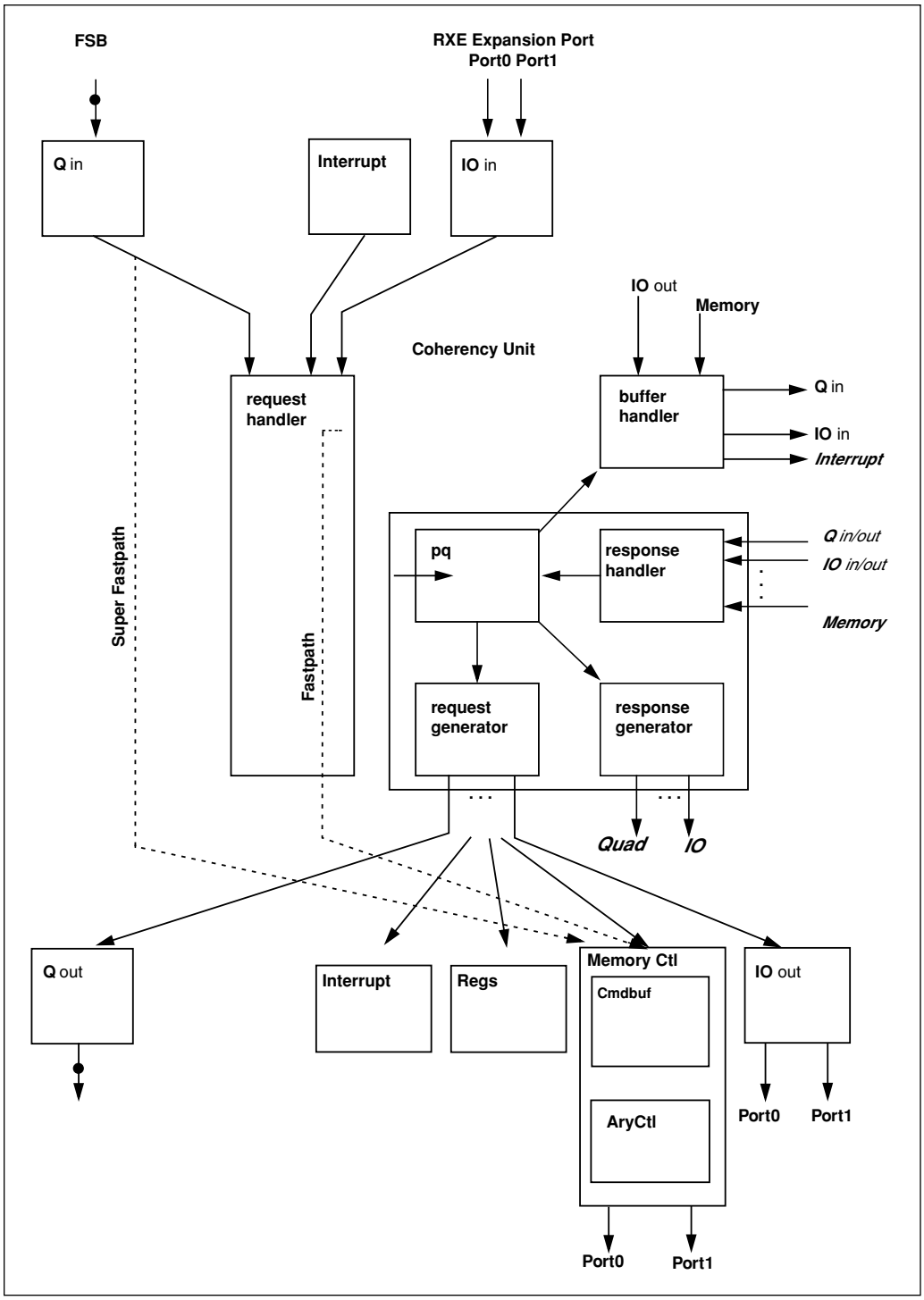


Figure 10 Coherency Unit Transaction Flow

The CU contains a 32-entry Pending Queue (PQ) that will hold all the command information until MIOC has finished execution of the command. Ordering rules are maintained by the CU by doing address comparisons between an incoming command in the Request Handler and the commands already in the PQ.

The PQ follows the command through all the stages of execution. It receives its initial command information from the Request Handler. The PQ entry interfaces with the Request Generators, Response Handlers, and the Response Generators.

If the command is to be sent to another functional unit on the chip, the PQ entry will inform the appropriate Request Generator that a new request needs to be sent. There is a unique Request Generator for each functional unit that is to receive a command from the PQ. These units include the Memory Unit, the RXE Expansion Port Interface, the Register Unit, the System Bus, and the Interrupt Unit. The Request Generator arbitrates between the 32 PQ entries, picks a winner, and sends the command information to the functional unit. The functional unit will return an acknowledge when it accepts the command.

When the functional unit is finished with the request, a response from the functional unit is handled by a Response Handler. There is a unique Response Handler for each functional unit. There are also Response Handlers that process information from the system bus phases such as the Snoop phase and the Response phase. The Response Handler accepts the response information and forwards it to the appropriate PQ entry. No acknowledge is returned for the response since responses are always accepted on the cycle received.

Once the command has been executed by the functional unit and the response has been received, the PQ entry will inform the appropriate Response Generator that a response can be sent back to the originating interface to indicate that the command has been completed. There is a unique Response Generator for each interface that can send commands to the CU.

These Response Generators behave like the Request Generators except that they do not require an acknowledge to be returned since the responses are always accepted by the originating interface.

The last chiplet in the CU is the Buffer Handler. The Buffer Handler decides how to allocate PQ entries and Data Buffers among the interfaces that send commands to the CU: the system bus, the RXE Expansion Port interface, and the Internal Interrupt interface. Each interface has a low PQ entry number that is used in determining who has priority for the next PQ entry to be handed out, and a high PQ entry number that indicates the maximum number allowed for that interface.

Performance

The CU contains a Fast path that allows memory read commands to bypass the PQ and be sent directly to the Memory Unit. This reduces latency for memory reads.

Memory reads received from the system bus can be accepted in order by the CU, or deferred depending on a programmable bit in the CU. If the command is deferred, it causes a deferred reply transaction to be sent on the system bus to return the data. The deferred reply uses up Address Bus bandwidth and increases the latency for reads. An in-order command does not require a deferred reply transaction. This also improves read latency. Memory reads that collide with a command already in the PQ are not able to execute in-order. These commands are deferred.

RAS

The CU checks all incoming commands from the system bus, the RXE Expansion Port interface, and the Internal Interrupt interface to see if the address maps to a valid destination. The three valid destinations for memory-type commands are the Memory Unit, the RXE Expansion Port interface, or the Register Unit. A set of registers that contain routing information are used to determine if the command maps to a valid destination. If the command does not map to a valid destination, an error will occur. The type of error and the action taken is programmable.

If the error is programmed to be a recoverable error, then a recoverable error bit is set, the command information is saved away in a register, and an interrupt is sent. If the command was a read, ones will be returned to the originator. If it was a write command, it will be discarded.

If the error is programmed to be a machine check error, then a machine check bit is set, the command information is frozen in a PQ entry, and an attention is sent to the service processor. If the command was a read, ones will be returned to the originator. If a write command, it will be discarded.

The PQ logic inside the CU has protocol checks that make sure that undefined events that occur to a given

entry will result in a machine check error. These checks include such things as a response from an interface to an entry that is not valid, a response from an interface that was not involved with a valid entry, or an invalid response from an interface that was involved with a valid entry. These errors set the machine check bit, the command information is frozen in the PQ entry, the specific type of protocol error is frozen in the PQ entry, and an attention is sent to the service processor. The PQ logic will record which entry was the first to report a protocol error so that the first error can be analyzed.

The CU contains logic that prevents a processor Bus Read Invalidate Line (BRIL) command from not making forward progress. A live lock condition may occur when several processors want to own the same cache line. Each processor sends a BRIL command on the system bus to gain ownership of the cache line. It is possible that one of the processors gets continually retried due to bus timings. To prevent this from happening, the CU logic will save away the first BRIL command that gets retried along with the processor that sent it. All accesses to that cache line will be retried until the original retried processor sends the command again, and the command is no longer retried. Once this happens, the BRIL live lock information is discarded, and commands to that cache line can flow again.

Configurability/Scalability

The CU can be set up to run in one of two main configurations.

The first configuration is called the Basic System. This system has MIOC connected to 1-4 Intel Xeon™ Processor MP Family Processor(s) via the system bus. In this mode, the CU handles all coherency and ordering rules for commands received from the system bus, the RXE Expansion Port interface, and the Internal Interrupt interface. The I/O Directory in MIOC will handle coherency for DMA Read data that is cached in IOB.

The second configuration is called the Enhanced System. This system has MIOC connected to either the CSC32 or CSC64 chip via the QuadT Bus. In this mode, the CU does not handle coherency and ordering rules for commands received from the QuadT Bus, the RXE Expansion Port interface, and the Internal Interrupt interface. The I/O Directory in MIOC does not handle coherency for DMA Read data that is cached in the IOB. These coherency functions are handled by either CSC32 or CSC64.

The CU can be configured to handle cache line sizes of 64 or 128 bytes. A cache line size of 64 bytes is used for the Basic System and the Enhanced System with CSC32. These systems use the Intel Xeon™ Processor MP Family processor. A cache line size of 128 bytes is used for the Enhanced System with CSC64. This system uses the Intel Itanium™ Processor Family processor.

There are three interfaces that send commands to the CU and its internal PQ. These are the system bus/QuadT bus, the RXE Expansion Port interface, and the Internal Interrupt interface. There are 32 PQ entries that must be shared by these three interfaces. The CU contains a register that specifies the maximum number of PQ entries that each interface can have. Each interface has its own maximum value so that the distribution can be tailored to get the best system performance. This register also contains a minimum value for each interface which is used in deciding which interface will be handed out the next PQ entry.

RXE Expansion Port

The RXE Expansion Port Interface in MIOC is comprised of two parts: the internal I/O logic specific to MIOC and the I/O Macro logic[1] that is common for all I/O interfaces. This section describes the internal I/O logic specific to MIOC.

The I/O Interface supports two RXE Expansion ports. The two ports may be connected in a ring fashion to provide a redundant path, or they may be left as two separate buses.

The I/O Interface is responsible for accepting I/O packets sent from IOB. These packets include DMA Reads/Writes, Register Reads/Writes, Interrupts, Ordered Responses, and Responses. These packets are stored in the I/O In Queue. Packets from the same I/O Node and PCI-X Bus are ordered with each other to follow PCI-X ordering rules. DMA Read/Write, Register Read/Write, and Interrupt packets are sent by the I/O Interface to the Coherency Unit over the command interface to be processed. DMA Read/Write packets may be as large as 128 bytes. If the system cache line size is 64 bytes, these DMA packets will be broken up into two 64-byte DMA commands when presented to the Coherency Unit. Ordered Response and Response packets are sent by the I/O Interface to the Coherency Unit over the response interface to indicate that a previous command sent by MIOC to IOB has finished.

The I/O Interface is also responsible for sending I/O packets to IOB. These packets include MMIO Reads/Writes, IO Reads/Writes, DKills, Ordered Responses, and Responses. These packets are stored in the RXE Out Queue. All packets in the I/O Out Queue are ordered with each other in a FIFO fashion. MMIO read/write and IO read/write packets are received by the I/O Interface from the Coherency Unit over the command interface. DKill packets are received from either the IO Directory in a Basic System or the Coherency Unit in an Enhanced System. Ordered Response and Response packets are received by the I/O Interface from the Coherency Unit over the response interface to indicate that a previous command sent by IOB to MIOC has finished.

The IO Directory contained in the I/O Interface logic maintains a record of all I/O bridges which may have pre fetched any byte within a 4K byte block of the system memory. IOBs do not own modified data or share data with a processor. If a processor reads data that an IOB has a copy of, then the IOB's copy will be invalidated. Therefore, no snoop responses (ie. HIT#) to the Intel Xeon™ Processor MP Family bus are necessary.

MIOC supports the IO Directory for Basic systems only. CSC32/CSC64 support this function for Enhanced systems. The coherency block size used by MIOC and I/O bridges is 4K bytes.

IOB read operations of any length are cached by the IOB and create an entry in the IO Directory. The IO Directory is 4-way set associative with 512 entries, and supports 4K byte blocks. This is independent of the system page size. IOB operations are the only requests that cause an entry to be created in the I/O directory.

All writes to memory from a processor or from an IOB are snooped against the IO Directory. If a Hit is detected then a Cache Invalidate (DKILL) is sent to the IOB identified in the directory, and the directory entry is invalidated. All processor reads to memory are snooped against the IO Directory. If a Hit is detected then a Cache Invalidate (DKILL) is sent to the IOB identified in the directory, and the directory entry is invalidated. (Bridges do not share data with processors).

Performance

The I/O Interface can support different link speeds. Selectable link clock rates can obtain a peak bandwidth in each direction of 1000/500/250MB/s.

DMA Write commands can pass DMA Read commands and MMIO responses that are from the same PCI-X bus. This allows DMA Writes to be sent to the Coherency Unit faster, thus generating higher DMA write bandwidth.

The 128-byte packet received from the RXE Expansion Port is split into two 64-byte operations on the system bus. The second 64-byte DMA Write command can be presented to the Coherency Unit as soon as the first has been accepted by it. This performance enhancement doubles the DMA Write bandwidth.

RAS

All packets sent across the I/O Interface contain CRC bytes that are used by the receiving device to ensure that the packet data has not been corrupted. If a CRC error does occur, the Link Acknowledge packet will not be returned to the originating chip. The originating chip will time out waiting for the Link Acknowledge packet and will send the original packet again.

The I/O Interface contains an Agent function that allows packets to be sent down the other RXE expansion port if the normal RXE Expansion port goes down. This allows a redundant path between MIOC and all IOB chips when they are all connected in a loop.

The I/O Out Queue will time all packets in its queue to detect packets that do not complete within the time-out period. The time-out period is programmable, and can also be disabled. If a time-out does occur, it can be programmed to set a recoverable error bit or a machine check bit. The command information will be saved away in a register.

Configurability and Scalability

The I/O Interface supports two RXE Expansion ports. Only one port is required to be connected to an IOB. If both ports are used, all the IOBs may be connected in a loop to provide redundancy.

The I/O Interface supports Hot plug of a RXE Expansion port. This allows the second RXE Expansion port to be connected to an IOB(s) with the associated PCI-X buses while the system is running.

There are several mode bits contained in the I/O Interface that allow variations on how commands are handled.

Interrupts

The Interrupt Unit (IU) in MIOC handles routing and redirecting interrupts to the processor, and generation of interrupts for internal chip set functions. The Interrupt unit supports Intel's Extended Advanced Programmable Interrupt Controller specification (xAPIC).

The destination for an interrupt is always one or more processors. There are several sources of interrupts. A processor may send an interrupt to another processor, called Inter-Processor Interrupt (IPI). Interrupts may be sourced by IO devices supporting the IOAPIC specification. In the Enterprise X-Architecture™ chipset, the IOB chip implements support for IOAPIC. Interrupts from IOB's IOAPIC are sent to the IU via MIOC's Coherency Unit. The Interrupt Unit also includes an IOAPIC, which is used to report recoverable error conditions and to support the CSC32/CSC64 IPC functions.

Interrupts are either edge or level triggered. Edge triggered interrupts do not require an acknowledge from the processor before another interrupt from the same source is sent. Level triggered interrupts require an End of Interrupt (EOI) transaction before the source is resampled and another interrupt sent. MIOC is responsible for routing the EOI transaction from the processor to all IOAPICs on the same node.

The EOI transaction is normally broadcast to all IOAPICs in the system. In order to reduce IO traffic from MIOC to IOB, the IU tracks up bound level triggered interrupts. The source and vector of each interrupt is recorded. When an EOI is received, its vector is used to determine which IOAPICs should receive the EOI.

The Interrupt Unit includes features to improve reliability. Parity coverage is used for the EOI routing array. If a parity error is detected, the EOI transaction is broadcast rather than routed. Input signals from CSC32 or CSC64 are also covered by parity. All interrupt transactions are checked for valid destinations. For example, an interrupt with a remote destination in a single node system will cause a machine check.

Conclusion

MIOC's several functional modes and multifunction interfaces provide the basis for the Enterprise X-Architecture™ building block structure. This flexibility enables a wide variety of system configurations and processor scalability options. Supported Intel Xeon™ Processor MP Family and Intel Itanium™ Processor MP Family processor topologies range from a basic one to 4-way to an enhanced high performance 16-way. Additional system I/O, if needed, is supported via a second RXE Expansion Port. Both SDR and DDR SDRAM memory, from 256MB up to 256GB, are supported. Multiple state-of-the-art memory RAS features make the memory subsystem extremely reliable.

References

1. T. Moe, C. Paynton, R. Shearer, S. Willenborg, C. Wollbrink, *EXA I/O Bridge*, p.31
2. J.D. Brown, S. Dhawan, *Enterprise X-Architecture™ Technology Overview*, p.1

EXA I/O Bridge

– Timothy Moe, Calvin Paynton, Robert Shearer, Scott Willenborg, Curt Wollbrink

Introduction

The I/O Bridge (IOB) is a high performance, scalable and reliable bridge that generates three PCI-X[1,2] busses and connects to the Memory IO Controller (MIOC) via a scalable, high-speed, reliable, point-to-point, low-latency, high-bandwidth link. This RXE link is intended for board-to-board and box-to-box coupling of system busses and/or I/O busses and scales to meters rather than centimeters. It supports switched topologies for high connectivity and incorporates a signaling methodology that scales with technology. The PCI-X interfaces are composed of three independent PCI-X Host Bridges (PHBs) each generating a 64-bit PCI-X bus that is PCI2.2 compliant. The IOB supports up to 133MHz PCI-X bus frequency. A logical view of the IOB is shown in Figure 1.

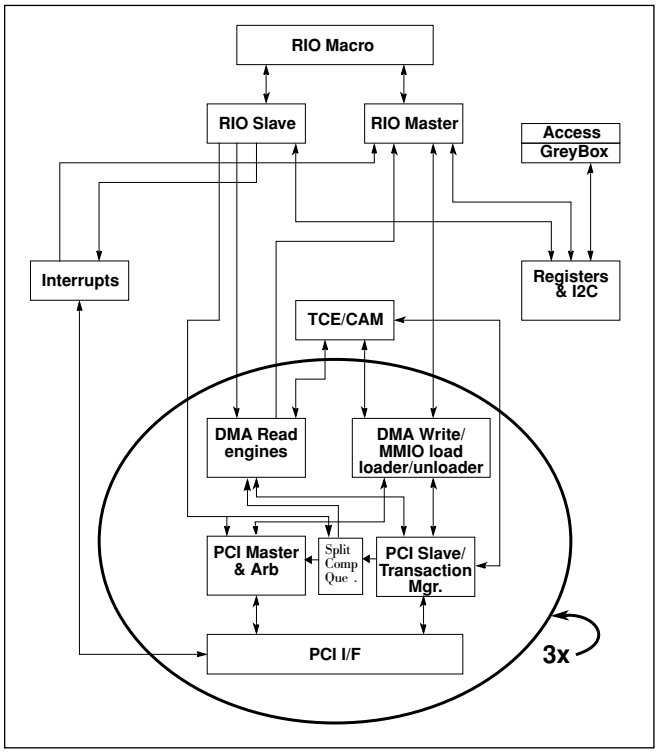


Figure 1 A Logical View of the IOB

RXE Interface

IOB contains two high speed proprietary RXE links, each with a maximum bidirectional bandwidth of 2GB/s. These two interfaces can be configured to operate independent of each other in a 500MB/s, 1GB/s, or 2GB/s modes.

The RXE physical interface is derived from the IEEE 1596-1992 (SCI) Standard. Significant enhancements have been made to improve the performance, increase fault tolerance, and provide the capability of runnng in more hostile (ESD) environments.

The functional characteristics of RXE are summarized below:

- Enhancement of the IEEE 1596-1992 (SCI) link protocol for data integrity
  - 32-bit CRC
  - Hardware packet retry on primary link
  - Hardware alternate path retry on redundant link.
- Support of switch-based topologies
- Agent function that relays packets to adjacent IOBs to support loop topologies
- Low latency
- Selectable 1000/500/250MB/s peak bandwidth in each direction with a 500/250/125MHz link clock rate.
- Physical interface (similar to IEEE P1596.3 SCI LVDS extension)
  - 500/250MHz modes:
    - 1-byte: 8-bit data, one flag bit and one clock bit
    - All signals are unidirectional and differential
- 500MHz Simultaneous Bidirectional Mode:
  - 2-byte: 16-bit data, one flag bit and one clock bit
  - All data signals are bidirectional and differential, flag and clock are unidirectional and differential.

An RXE subsystem consists of one or more IOBs connected to a MIOC. For increased connectivity, switch nodes or switching bridge nodes are used to provide connection to additional IOBs. For example eight IOBs can be connected to the MIOC for a total of 24 PCI-X busses in a system.

Three distinct high-level layers exist within each RXE device; an Application Specific Integrated Circuit (ASIC) Interface Layer, an End-to-End Layer, and a Link Layer. These layers make up the RXE macro which is common to all RXE ASIC devices.

ASIC Interface Layer

The ASIC Interface Layer is the boundary between the ASIC and the RXE interconnect. It operates at a trans-

action level, mapping user requests and responses into RXE commands, and vice versa. It is the application layer which must be tailored to the particular external bus or I/O adapter; the End-to-End and Link layer definitions remain common over all RXE designs.

End-to-End Layer

The End-to-End Layer of the interconnect is responsible for ensuring reliable transmission and reception of packets across multiple RXE links. It can optionally use any available alternate paths and may initiate end-to-end packet re-transmissions as necessary.

Link Layer

The Link Layer ensures that packets are delivered successfully across a link. MIOC and IOBs with multiple links also provide some switching function among the links (e.g., switches).

PCI-X Interface

The IOB also contains three full 64-bit capable PCI 2.2/PCI-X 1.0 compatible buses. Each bus is independently configurable to operate in either mode with a plethora of choices of bus speeds. A bus configured in PCI mode supports operating frequencies of 33MHz or 66MHz; while in PCI-X mode, a bus supports operating frequencies of 66MHz, 100 MHz, or 133MHz. Peak bandwidth on these buses is ~1.1 GB/s.

Table 1 shows the combination of bus slots/bus speed while the IOB is operating in PCI and PCIX mode.

Table 1 PCI-X Bus Speed vs. Slot Tradeoff

Bus Speeds	PCI Mode	PCI-X Mode
133 MHz	–	One Slot
100 MHz	–	Two Slots
66 MHz	Two Slots	Four Slots
33 MHz	Four Slots	–

Arbitration

Each bus contains arbitration support for up to four slots and a Hot Plug Controller which can be used to dynamically add devices during run time. Access to the bus is controlled via a multilevel arbitration prioritization scheme allowing certain slots to operate at a higher priority with regard to obtaining access to the bus. The IOB supports legacy I/O on one PHB with inclusion of ISA bridge arbitration, allowing ISA and up to four PCI devices to be attached on a single PCI bus. An arbiter fairness counter controls how long grant is held to an individual device. Using the arbiter fairness counter in conjunction with the PCI Configuration Space Latency Timer provides flexibility to control bus accesses. If a PHB device has high bandwidth devices attached, larger values can be loaded into the arbiter fairness counter and latency timer to maximize bandwidth with large, efficient PCI transfers. If latency sensitive devices are attached, the arbiter fairness counter and latency timer can be loaded with smaller values, minimizing the delay between requests and grants.

The IOB parks the PCI-X bus on itself during periods of bus inactivity. This minimizes the time delay between when the IOB receives a command destined for the PCI-X bus and when it is able to drive the command to the target device.

Addressing Support

The IOB supports full 64-bit addressing. When attaching 32-bit PCI devices, the IOB supports Dual Address cycles on the PCI bus as both a master and a slave. This allows access to above 4GB address space in both directions, negating the need for address translation when attached devices are capable of addressing above 4GB.

The IOB supports multiple memory ranges per PHB to allow for maximum software configurability in communicating with attached devices.

Bus Lock Support

The IOB supports exclusive access on the PCI bus as a master via the PCI LOCK# signal. By asserting LOCK#, the IOB can gain exclusive access to a target device, preventing the device from accepting new requests for either of its interfaces that are not originated by the IOB. Lock accesses are initiated by the process via loads and stores to the PCI bus. To initiate a lock, a load is issued from a processor to the IOB, destined for the PCI bus with the lock bit active. The IOB asserts LOCK#, performs the load and leaves LOCK# active until the processor follows the load with a store to the same PCI bus with the lock bit inactive.

PCI-X Support

The IOB can support up to three PCI-X busses. PCI-X provides significant performance improvements over PCI 2.2. Since each bus can operate at 133MHz, 8-byte wide, the IOB has an input bandwidth capacity up to 3

GB/s. PCI-X supports twice the bandwidth of PCI, just based on the enhanced frequency.

Bus efficiency has also been improved with PCI-X. On a PCI bus, a device requesting data would be retried by a PHB until the PHB was able to gain access to the data in main memory. In cases where the access time to main memory is large, a PCI device would continue to retry its operation on the PCI bus, consuming bandwidth that could be used by other devices. In PCI-X, the PHB is allowed to give a split response to the read request and provide data after main memory has been accessed via a split completion. Additionally, PCI-X includes the length of the transfer with the read request, allowing the target to fetch the entire byte count of an operation. The IOB utilizes these performance enhancements to improve PCI-X bandwidth. All reads on the PCI-X bus are given split response by the IOB. The IOB is capable of handling up to sixteen read requests per PHB concurrently, allowing devices to have multiple reads outstanding. The IOB fills all read requests approximately in the order they were received, but does allow out of order completion. The IOB monitors the sixteen read requests outstanding and initiates split completions once the entire byte count, or 512B, whichever is less, has been received from main memory.

For reads of greater than 512B, the IOB kicks off the additional fetches as soon as it starts the split completion. If the latency to main memory is small enough and the additional load data is returned before the IOB has sent all 512B of the split completion, the IOB extends the split completion to include the additional data. In this manner, it is possible for the entire PCI-X read to be returned in one burst, provided main memory bandwidth can provide data and the IOB retains grant ownership of the PCI-X bus.

The IOB supports single data phase disconnect as described in the PCI-X specification when mastering commands. When IOB is a target it will not single phase disconnect. It will accept operations at least up to the next Allowable Disconnect Boundary (ADB) of 128 bytes.

Interrupt Controller

The IOB implements Intel's I/O xAPIC interrupt controller. The IOB also supports an external I/O xAPIC interrupt controller. The choice of an internal or external interrupt controller is system dependent. In either case an interrupt is delivered to the processors by issuing a Store to an address within the Interrupt Delivery Block.

An interrupt can be initiated by a PCI-X device or by an internal condition within the IOB. PCI-X devices can be wired to any of the 48 external interrupt pins on the IOB independent of which PCI-X bus the device is resident on.

If an External Interrupt Controller is being used then the IOB signals the External Interrupt Controller of an internal interrupt condition by driving an output pin low. This in turn causes the External Interrupt Controller to issue a Store on PHB0 to an address within the Interrupt Delivery Block. External Interrupt Controllers must be resident on PHB0.

The 48 external interrupt pins on the IOB can be programmed to be active high or low, as well as level or edge triggered.

Technology

The IOB was one of the first ASICs developed that incorporated IBM's advanced SA27E technology. IBM's SA27E technology is a 0.18 micron foundry offering with copper wiring.

Basics

Shown in Table 2 are some of the unique technological aspects of the IOB design.

Table 2 IOB Key Technology Points

Die Size	8.5 mm
Package	32 mm x 32 mm
I/O Pitch	1.27 mm
Supply Voltages	1.8V, 2.5V, 3.3V
Signal I/O	518
Cell Count Total	14.5 million
% Cell Count I/Os	~20%
% Cell Count Arrays	~35%
% Cell Count Logic	~45%

In addition to the above, the IOB design also took advantage of a couple of "new" features of the base SA27E technology:

- Power Quadrants: The IOB design required support for three different voltage domains (1.8V, 2.5V, & 3.3V). In previous technologies this would have required a custom image, but now was a part of the base offering.
- Analog Ground for PLL: Allowed for higher PLL performance over previous technology offerings.

Image

Shown in Figure 2 is the full chip schematic view of the IOB. An interesting point to take from this is that the IOB is a very full ASIC. In fact, the IOB consumed ~80% of all available cells. The black rectangles shown are data buffers.

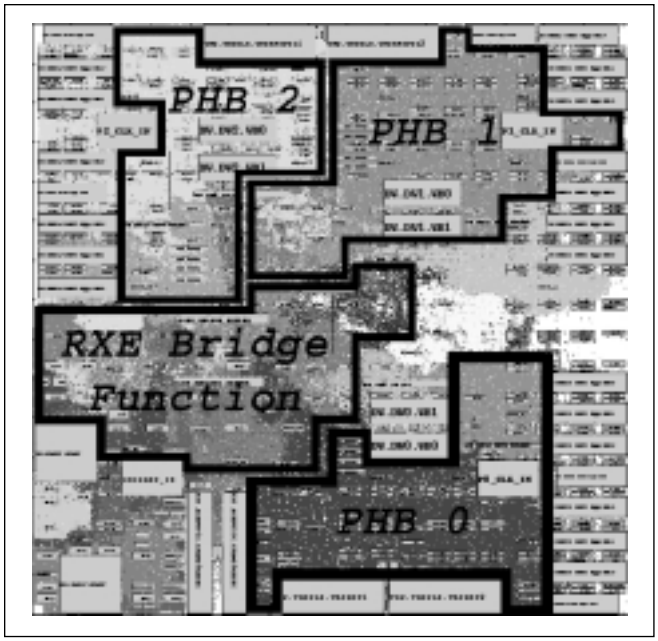


Figure 2 Chip Image

Transaction Flow

In the IOB, transactions flow between the RXE interface and the three PCI-X busses. Processor initiated commands are presented at the RXE interface and are targeted to the PCI-X bus or to the IOB's internal registers. PCI-X initiated commands are presented at one of the three PHBs and are targeted to the RXE interface, bound for processor or memory. The logic handling the transaction flow consists of Control Flow/Queuing logic and Data flow logic.

Control Flow

Figure 3 shows the major IOB control partitions and interface paths. Inbound queues are for holding commands received from RXE, and outbound queues are for holding commands to be sent to RXE.

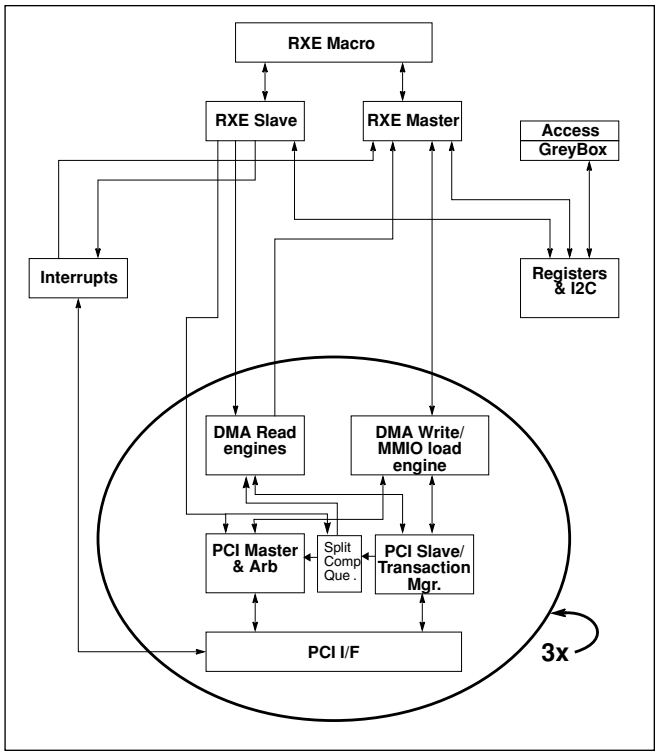


Figure 3 IOB Control Flow

Inbound Store Request Queue

Each PCI-X Host Bridge (PHB) in the IOB contains a Store Request Queue located in the RXE Slave partition. Each queue is capable of holding up to four store commands received from RXE. These commands can be Memory Mapped I/O (MMIO) stores to PCI memory space or I/O stores to PCI I/O space.

Inbound Load Request Queue

Each PHB in the IOB contains a Load Request Queue located in the RXE Slave partition. Each queue is capable of holding up to three load commands received from RXE. These commands can be MMIO loads to PCI memory space or I/O loads to PCI I/O space.

Inbound Internal Request Queue

The GreyBox partition contains an MMIO Request Queue for holding MMIO Load and Store commands to internal IOB registers. This queue is capable of holding up to three commands.

Outbound DMA Write Engine/queue

Each PHB in the IOB contains a DMA Write Engine. The DMA Write Engine accepts memory bound stores from the PCI Slave partition. It then breaks them into valid packet sizes for the RXE link and puts each individual

command onto a queue. This command queue can hold up to 32 entries targeted for RXE comprising of DMA writes, MMIO load replies, I/O load replies and interrupts. Requests are sent from this queue to the RXE Master partition.

Outbound DMA Read Engine

There is a DMA Read Engine for each DMA read channel in a PHB. The read engine does not contain a command queue. Instead, each engine contains a sequencer which dispatches DMA read requests to the RXE Master partition.

Outbound RXE Request Queue

The RXE Master partition contains an Outbound RXE Request Queue capable of holding up to 32 commands from any of the three PHBs. This queue is used for sending requests and ordered responses to the MIOC chip. This queue holds DMA writes, DMA reads, ordered MMIO load replies, I/O load replies and interrupts. A round robin arbitration scheme is used to place commands from the three PHBs on this queue. Commands are received from the DMA Write Queues, DMA Read Engines and the GreyBox.

Outbound RXE Response Queue

The RXE Master partition also contains an Outbound RXE Response Queue capable of holding up to eight commands from any of the three PHBs. This queue is used for sending responses that have no ordering requirements with other requests. This queue holds unordered MMIO load replies. A round robin arbitration scheme is used to place commands from the three PHBs on this queue. Commands are received from the DMA Write Queues and GreyBox.

Data Flow Logic

The IOB has four different sets of data buffers as shown in Figure 4. They are the DMA Read Reply Buffer, DMA Write Buffer, MMIO/IO Load Reply Buffer and MMIO/IO Store Buffer. Each PHB owns its data buffers.

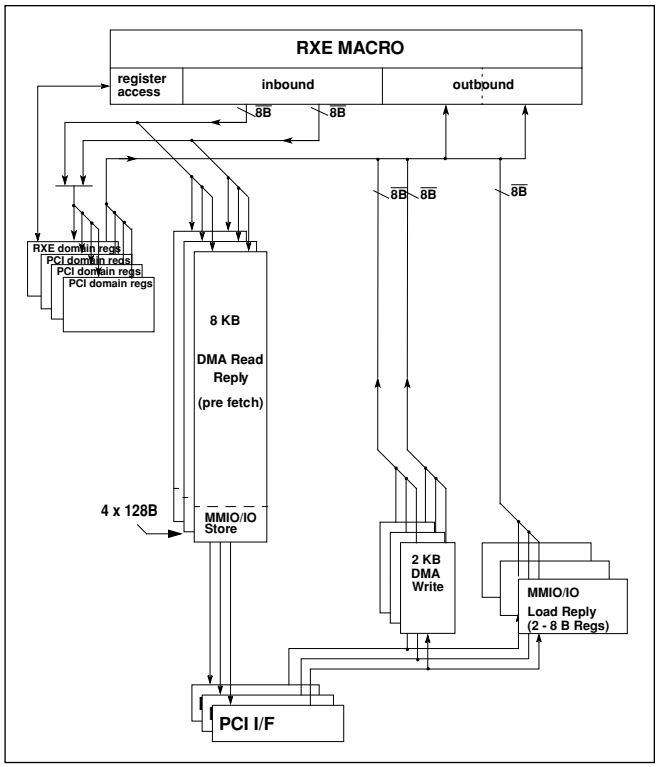


Figure 4 IOB Data Buffering

DMA Read Reply Buffer

The IOB implements an 8KB buffer per PHB for storing DMA Read Reply data. This buffer is divided into channels. The number of DMA read channels is configurable per PHB as follows:

1. Fifteen channels of 512 bytes, each channel containing four 128 byte sectors.
2. Seven channels of 1024 bytes, each channel containing eight 128 byte sectors.

These options provide the capability to tune a PCI-X bus for the type and number of devices on that bus.

Each channel is divided into four or eight sectors. These sectors define the data regions within the buffer where access control is passed back and forth between the RXE and PCI clock domains. A channel containing DMA read data is also considered a cache from a coherency perspective due to the prefetching that occurs on DMA reads.

DMA Write Buffer

The IOB implements a 2KB buffer per PHB for storing DMA Write data. Each buffer is divided into sixteen 128 byte sectors. These sectors are used to manage data access across the RXE and PCI clock domains.

Each entry in the DMA Write queue owns one of these 128 byte sectors. Note that one DMA Memory Write received from the PCI bus can occupy multiple 128 byte sectors. The DMA Write Engine breaks the PCI write into separate commands for RXE. Each RXE command has a maximum data length of 128 bytes.

MMIO/IO Load Reply Buffer

MMIO load reply data and I/O load reply data normally flow through two 8-byte registers. The dataflow supports two loads in progress of up to eight bytes in length. Load replies that are greater than eight bytes use the first sector of the DMA Write Buffer.

MMIO/IO Store Buffer

MMIO and I/O store data flows through four reserved 128-byte sectors in the DMA Read Buffer.

DMA Read Engine Operation

The DMA Read Engines are designed to prefetch data from memory whenever a read multiple is received from the PCI bus. A read channel is allocated and the engine sends enough 128-byte DMA reads to RXE to fill all sectores of that channel. This could be four or eight 128-byte reads depending on the channel configuration. Response packets to these DMA reads are later received on RXE and fill the DMA Read Buffer from the RXE domain. The first 128 byte sector becomes available for reading in the PCI clock domain after all 128 bytes has been received. The PCI device is now allowed to begin the data transfer. When the full 128 bytes has been read from the sector, data transfer on the PCI bus is allowed to continue if all the data has been received for the next sector from the RXE. The first 128-byte sector is now free to receive data. Access is transferred back to the RXE domain as a new DMA read packet is generated to a free sector. Prefetching continues as the engine tries to keep the entire channel filled with data.

Prefetching stops at a 4K address boundary. Read channels are allocated on 4K boundaries. If the device crossed a 4K boundary, a new read channel is allocated. Prefetching greatly improves the performance of a page out operation.

Reliability/Serviceability

The RXE macro contains hardware packet retry capabilities in the case of a corrupted packet. It also contains an alternate path retry mechanism if a redundant RXE link is installed. The three PCI-X Host Bridges operate independently so an error on one of the busses does not affect operations on the other busses. If an error is detected, the bus is put into a freeze state and subsequent operations on the bus are not allowed until the error condition is cleared. Upon detection of a recoverable or non-recoverable error, the IOB can be configured to either send an interrupt to the processor or notify a service processor.

Debug

Debug of the IOB is accomplished through JTAG, I2C, external debug outputs and two internal debug trace arrays. All architected registers can be read or written through JTAG or I2C. Sequencers and other internal logic signals from the various IOB partitions can also be routed to the 48 debug I/O pins for analysis. Alternatively, these signals can be routed to an internal trace array for easier analysis.

Conclusion

The IOB allows Enterprise X-Architecture to expand and scale its I/O capabilities. By using an RXE Expansion Port in the MIOC and by cascading multiple IOBs together on a RXE link, I/O expansion is easily obtained. The three PCI-X Host Bridges in the IOB allow a wide range of configuration options for system designers. They can be independently configured for PCI to support legacy hardware or for high speed PCI-X to support new high performance I/O adapters.

References

1. *PCI Local Bus Specifications*, Rev. 2.2, December 1998.
2. *PCI-X Addendum to the PCI Local Bus Specifications*, Rev. 1.0a, July 2000.

EXA Cache/Scalability Controllers

– John M. Borkenhagen, Russell D. Hoover, Kenneth M. Valk

Introduction

The Enterprise X-Architecture Cache/Scalability Controllers deliver high performance and scalability to xSeries Servers by enabling a large-scale, shared-memory, multi-processor system with hardware cache coherence. The Cache/Scalability Controller 32 (CSC32) and Cache/Scalability Controller 64 (CSC64) chips began as one design and both implement the same fundamental set of features. The major distinction is CSC32 supports the Intel Xeon™ Processor MP Family while CSC64 supports the Intel Itanium™ Processor Family. CSC32 and CSC64 each integrate a state of the art 4-way Symmetric Multi-processor (SMP), XceL4™ Server Accelerator Cache (L4 cache), three high bandwidth SMP Expansion ports with a chipset expansion bus into a fully scalable 16-way SMP. This paper describes these high end server features and their basic operation. Major architecture differences that lead to a unique CSC32 and CSC64 implementation are noted.

System Interfaces

Each chip has four major system interfaces as shown in Figure 1 and Figure 2. Internally, each chip runs at 200MHz or twice the system bus frequency.

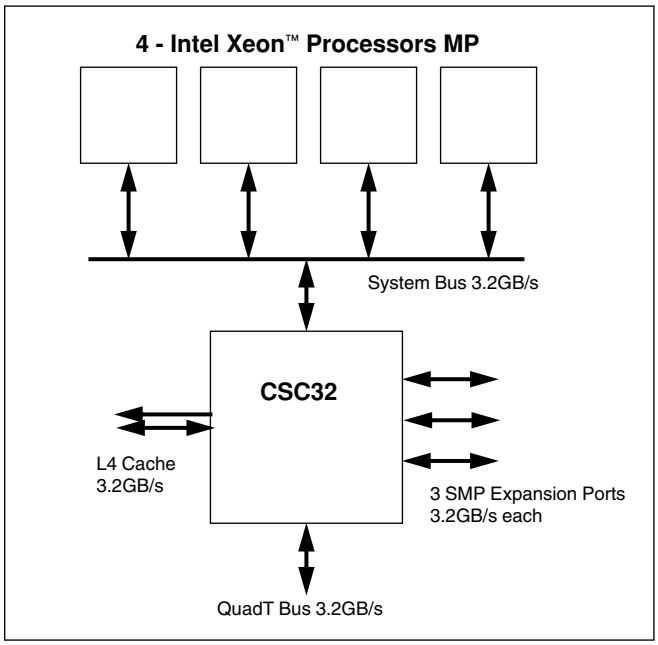


Figure 1 CSC32 Interfaces

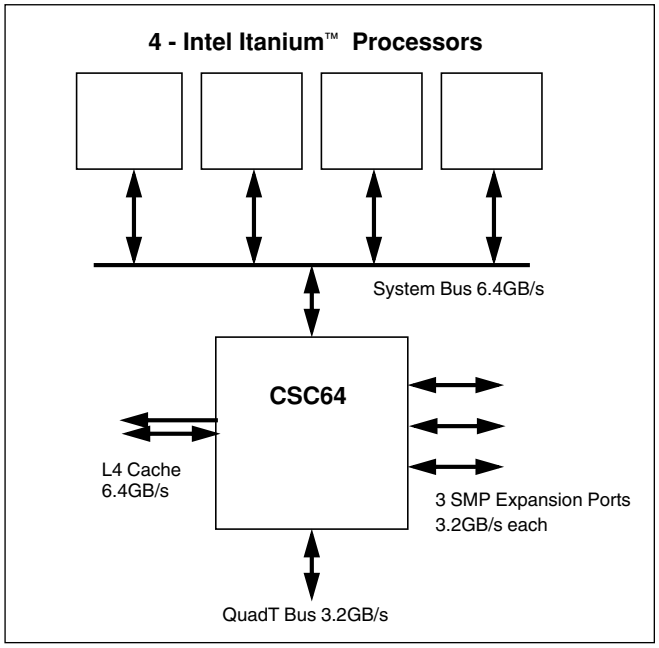


Figure 2 CSC64 Interfaces

System Bus Interface

Both CSC32 and CSC64 fully implement the system bus as defined by their respective processors. The bus features that are most critical to system performance are fully leveraged such as split transactions, source synchronous transfers, multiple outstanding transactions and pipelining all transaction phases. All system bus transactions to the L4 cache are completed in order to achieve the minimum memory latency. Transfers of both address and data are done source synchronously to increase bandwidth and improve performance. The maximum data rate of 3.2GB/s on CSC32 and 6.4GB/s on CSC64 can be realized from the L4 cache to the system bus. The maximum number of outstanding transactions and pipeline depth is limited by the processors. Each chip fully supports the processor bus MESI cache coherence protocol and integrates it into a fully scalable shared memory multi-bus cache coherent system. Parity protection is implemented for data integrity as defined by the system bus specifications.

XceL4 Cache Interface

The XceL4 cache (L4 cache) interface directly attaches CSC32/CSC64 to industry standard, high performance Double Data Rate (DDR) Synchronous Dynamic Random Access Memory (SDRAM). The maximum data rate at this interface equals the system bus interface at 3.2GB/s for CSC32 and 6.4GB/s for CSC64. A higher degree of data integrity protection is maintained with a single bit error correction code (ECC) used on the data transferred, and parity protection for address and control signals. Read transaction latency is the primary focus and has been optimized for minimum latency and maximum throughput. An access can start immediately when a command is received from the system bus. This speculative access reduces latency and improves performance. If the access is not needed it is terminated before throughput is impacted. Multiple transactions to multiple banks are interleaved and reordered to optimize both latency and throughput. Data can be transferred from the L4 cache interface directly to the system bus without buffering.

SMP Expansion Port Interface

The SMP Expansion Port is a scalable, high-speed, point-to-point interface optimized for high bandwidth, board-to-board and box-to-box coupling. Also referred to as a scalability port, it scales to meters rather than centimeters and incorporates the same signaling technology as the RXE Expansion Port. The physical interface consists of a simultaneous, bidirectional 16-bit data path, and unidirectional clock and flag bits. The clock rate is up to 400MHz and data is transferred on every clock edge leading to a maximum data transfer rate of 3.2GB/s per port. Data integrity is a primary focus and features include a full 32-bit CRC, hardware packet retry and alternate path retry. While the scalability port reuses the End-to-End Layer and Link Layer from RXE Expansion Port, the application layer is unique. This layer operates at the transaction level, mapping bus transactions into packets. The transaction set and packet formats are similar to the system bus transactions but include many optimizations for a directory-based, scalable coherence protocol. The maximum data payload within a packet is one cacheline and therefore is twice as large in CSC64 than CSC32.

QuadT Interface

The interface between CSC32/CSC64 and the Memory/IO Controller (MIOC) is optimized for performance. Arbitration is optimized for a point-to-point connection to reduce latency for processor accesses. An early data indication for memory data reads improves latency. Data transfer size doubles for CSC64 while the bus width remains the same.

Technology

The CSC32 and CSC64 chips are built with IBM's CMOS 7SF SA27E 0.18micron copper technology. Table 1 lists the technology characteristics of the CSC32 and CSC64 chips.

Table 1 Technology Summary

Attribute	CSC32	CSC64
Chip Size	13.8x13.8mm	14.7x14.7mm
# Signal I/O	670	872
# Transistors	59M	63M
On-Chip EDRAM	4 MByte	4 MByte
Substrate	Ceramic	Ceramic
Substrate Size	42 mm	42 mm
I/O pitch	1.27 mm	1.00 mm
Proc Bus BW	3.2 GB/s	6.4 GB/s
L4 Brandwidth	3.2 GB/s	6.4 GB/s
Scalability Port Bandwidth	9.6 GB/s	9.6 GB/s
Scalability Memory Bandwidth	3.2 GB/s	3.2 GB/s

The die image of the CSC32 chip shown in Figure 3 is superimposed with logic partition boundaries to show the die area consumed by each function.

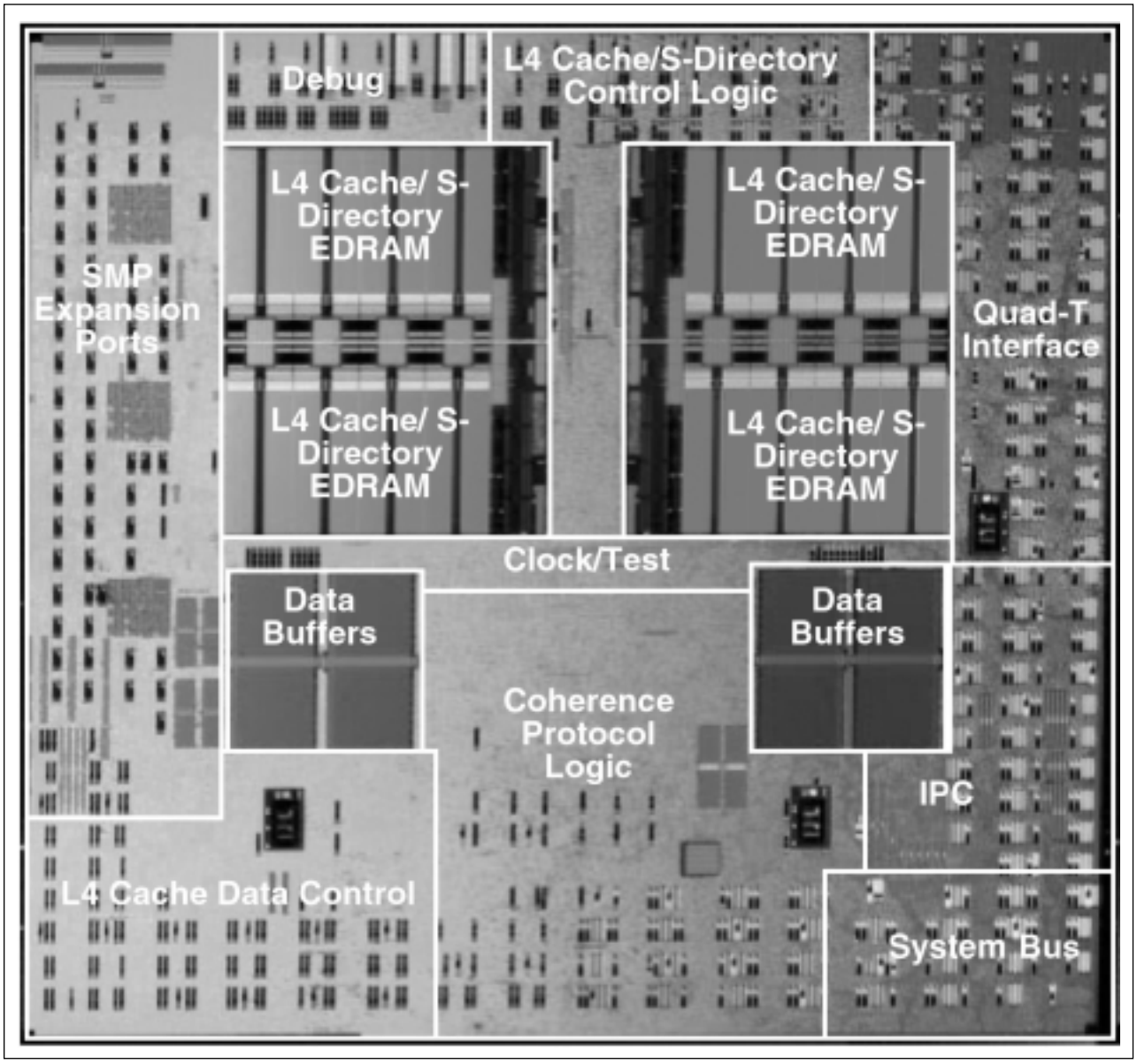


Figure 3 CSC32 Image

System Configurations

CSC32 and CSC64 support a very flexible set of system configurations. A node is the fundamental building block around which many system configurations may be built. A node is a self-contained, fully functional 4-way SMP, capable of executing a single image operating system image. A partition is an aggregation of one or more nodes, each containing a full complement of processors, memory and I/O. A multiple node partition is joined through a scalable shared memory architecture. The dual node configuration requires a single scalability port connection but the second scalability port adds a redundant path as well as doubles the bandwidth. The hardware balances the load across both ports. A primary and a backup, or redundant, path can be configured between any two nodes in all multi-node configurations.

Figure 4 illustrate examples of multiple partition configurations within a multi-node complex.

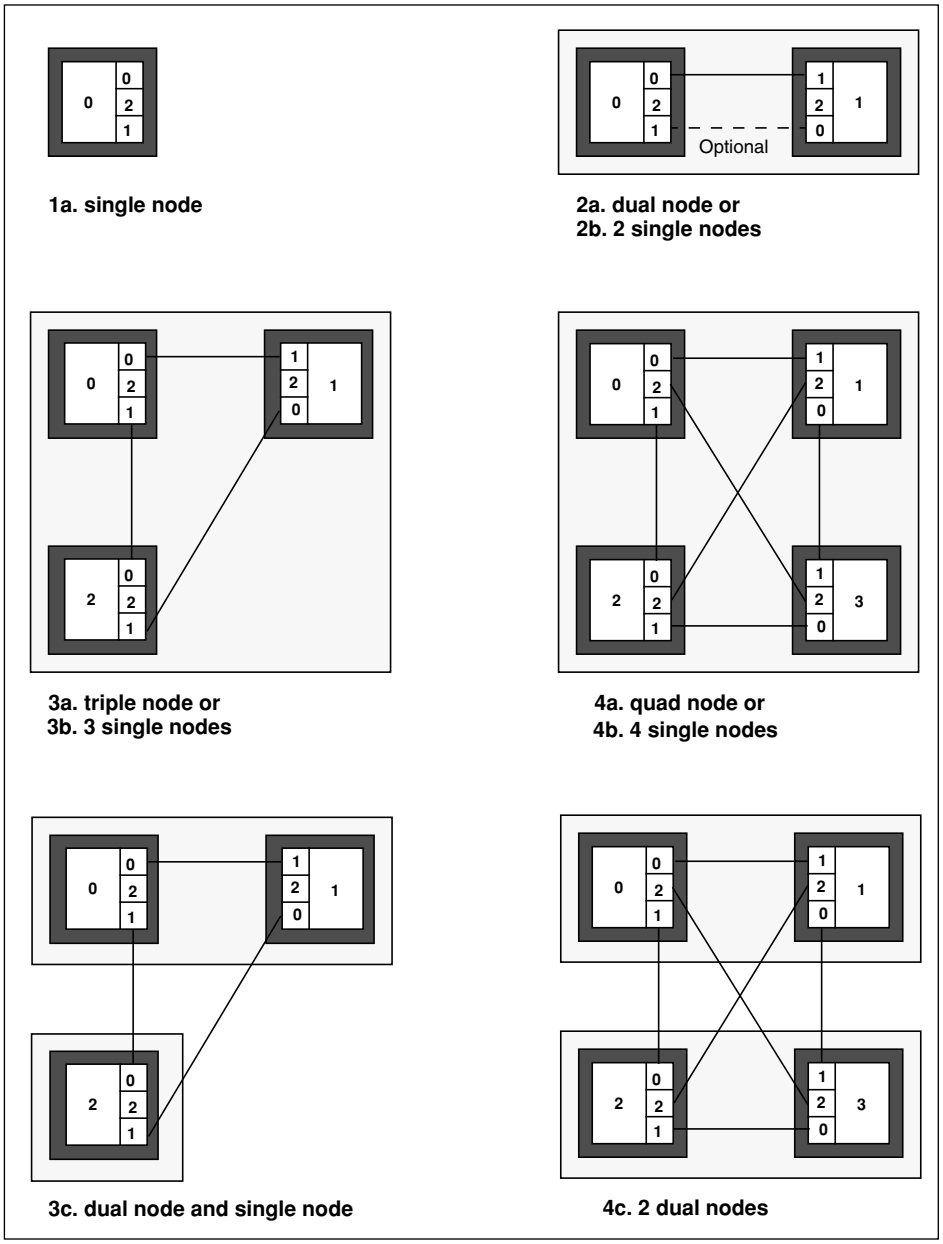


Figure 4 Partition Configurations within a Multi-node Complex

L4 Cache

A key attribute of the Enterprise X-Architecture is that a customer only pays for the hardware he needs. This is not achievable with most SMP switch architectures because the cost of the switch hardware is incorporated in the smallest systems. The Enterprise X-Architecture is able to achieve XpandOnDemand™ scalability by implementing a building block approach for growing SMP size. When a customer encounters the need to grow beyond a 4-way SMP, he can retain his original SMP Expansion Module and simply add additional SMP Expansion Modules. The additional SMP Expansion Modules are connected together with cables to create a larger SMP.

While the building block approach provides a great growth path for customers, it results in additional main memory access latency when one node is accessing the main memory on another node. Incorporating an L4 cache in each node allows local storage of frequently used memory locations physically on a different node. The access latency of the L4 cache is almost 4X better than a local system memory access and 9X better than a remote system memory access. The reduced latency of the combination of frequently used remote and local system memory accesses results in an average system memory access latency that is less than a switch topology with no L4 cache. The L4 cache is key to both enabling a building block architecture and providing a performance advantage over systems with L4 cache.

L4 cache sizes are 32MB for CSC32 and 64MB for CSC64. The L4 cache is 4-way set associative and operates on a 64-byte cache line size on CSC32 and a 128-byte cache line size on CSC64. The L4 cache holds both instructions and data. The L4 access latency measured from request on processor bus to first data returned on processor bus is 80ns on CSC32 and 70ns on CSC64. The L4 interface was designed with generous data bandwidth to ensure it did not become a bottleneck. The L4 bandwidth matches the processor bus with 3.2GB/s on CSC32 and 6.4GB/s on CSC64.

L4 Cache Implementation Specifics

The L4 cache directory is physically implemented together with the S-Directory (Scalability Directory) using an on-chip EDRAM (Embedded Dynamic Random Access Memory) shown in Figure 5. The reason for pairing the two directories is that both the L4 cache directory and S-Directory need to be accessed together for each coherent access. The coherency functions of the L4 cache directory and S-Directory are described in the next section. The L4 cache directory is 4-way set associative. The S-Directory is 7-way set associative. The EDRAM is 256 bits wide, supporting a read of four L4 cache directory classes with ECC/LRU (Least Recently Used), in combination with seven S-Directory classes with ECC/LRU on each access. To minimize the impact of EDRAM precharge time, the directories were built with four EDRAM macros to allow interleaving. The directory logic has a snoop request queue to optimize EDRAM bank interleaving. There is also a directory update buffer to allow EDRAM updates to be held off during snoop read activity, resulting in improved performance. The EDRAM update buffer is snooped in parallel with the EDRAM. The contents in the update buffer override the EDRAM contents on an update buffer hit.

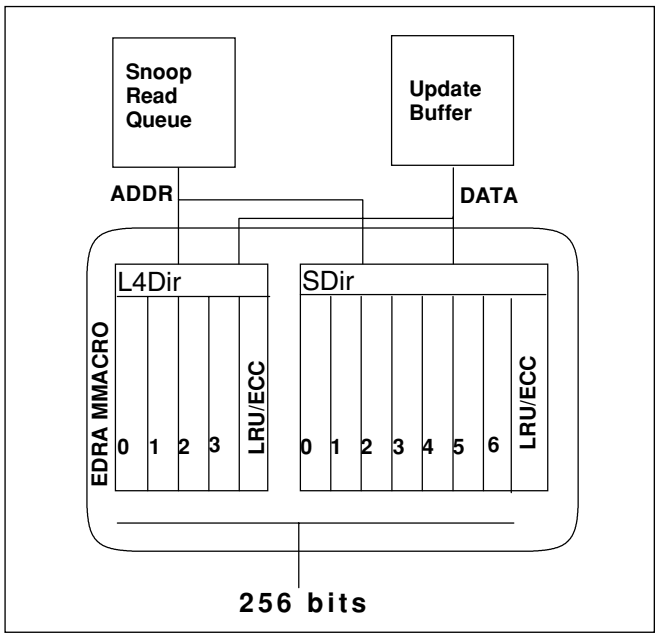


Figure 5 EDRAM usage

The L4 cache data and controller (Figure 6) consists of on-chip load and store command buffers along with the DRAM control logic used to control the external L4 cache data DDR SDRAMs. The load and store buffers hold the address and control information for up to 32 commands each. The data is stored in the chip's internal buffer before being written to the SDRAMs. L4 cache load data comes from the SDRAMs and goes either directly to the system bus or is temporarily stored in the chip's internal buffer until the requester is ready for it.

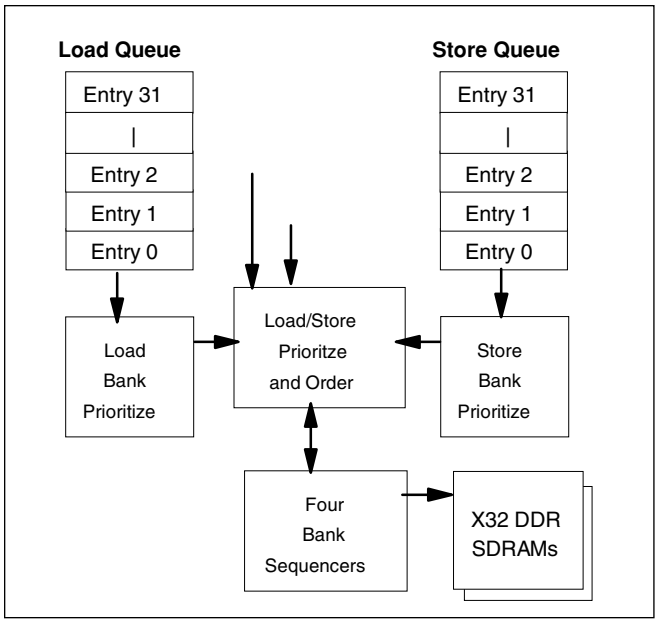


Figure 6 L4 Cache Data Control

The L4 cache data is stored in 4M X 32 bit wide DDR SDRAMs. All load/store accesses to the SDRAMs are a full cache line and require a data burst length of eight. To achieve a full cache line transfer with a burst length of eight, CSC32 uses 2-X32 SDRAMs for data and 1-X32 SDRAM for ECC for its 64-byte cache line size. CSC64 uses 4-X32 SDRAMs for data and 1-X32 SDRAM for ECC for its 128-byte cache line size. The SDRAMs are controlled in lockstep to provide a unified 72-bit interface for CSC32 and a 144-bit interface for CSC64.

To optimize L4 cache access latency, load commands default to higher priority than store commands. The L4 logic detects incoming load commands that have an address match with a command in the store buffer. Any store buffer entries with an address match are prioritized ahead of the incoming load to maintain memory ordering. The DDR SDRAMs have four banks that are controlled with four on-chip bank sequencers. Central prioritization logic keeps track of which banks are idle and what commands are buffered for each bank. The central prioritization logic schedules the bank sequencers to optimize SDRAM data bus bandwidth. Commands are not queued and the bank sequencers are not issued a command until the exact time the SDRAMs can provide data in an open data bus slot using absolute minimum SDRAM timings. With this scheme, if a high priority command arrives while other commands are waiting for an open data bus slot or a bank sequencer to go idle, the high priority command will execute ahead of any lower priority commands that were waiting.

CSC32 and CSC64 use a standard SEC/DED (Single Error Correct, Double Error Detect) ECC (Error Correction Code) scheme for L4 data cache. In applications where the processor bus also uses a SEC/DED ECC scheme, data is sent to the processor without going through correction to reduce L4 load latency by one cycle. L4 SDRAM address and command have their own unique error protection logic.

Operating the DDR SDRAM interface at 400Mb/s per pin required special logic for DQS read skew control. DDR reads require incoming DQS clock strobes to be delayed by 1/2 cycle relative to data. Process, temperature, and voltage variations make it difficult to maintain the required tolerances for the 1/2 cycle (1.25ns) delay. CSC32 and CSC64 solved this problem with DQS delay calibration circuitry that continuously compensates for process, temperature, and voltage variations. The DQS calibration is hidden under the SDRAM refreshes so that calibration does not impact performance.

Another challenge at 400Mb/s is to ensure a glitch free, source synchronous DQS clock. A DQS clock glitch will corrupt data in the source synchronous latches used to receive data. The DQS signal goes into tri-state when neither the SDRAM nor the control chip are driving. The standard methods to prevent DQS glitches are to have the controller chip drive DQS low during inactivity or use a pulldown resistor to keep DQS low when it is not being driven. These methods interfere when trying to achieve 400Mb/s operation. CSC32 and CSC64 solved this problem with DQS clock gating. The DQS clock gate logic generates the enable clock gate edge with latches clocked by the synchronous internal clock. The disable clock gate edge is generated with DQS edge counting latches clocked by the source synchronous DQS.

L4 Cache Technology

The L4 cache directory needs to be large enough to hold a tag for each cache line in the L4 data cache. The L4 cache is too large to implement the cache directory with on-chip SRAM. Implementing the cache directory external to the CSC32/CSC64 chip and maintaining associativity would significantly impact L4 cache access latency and require additional chip signal pins. CSC32/CSC64 leveraged IBM's leading edge SA27E EDRAM technology to implement an on-chip cache directory. EDRAM technology provides the DRAM storage cell style density that is required to fit the directory on the ASIC chip.

The L4 cache data was originally defined to be built with RAMBUS™ DRAM technology. As high level definition proceeded, the graphics card industry started pushing the envelope of high end, wide I/O DDR SDRAM technology. The high end graphics cards were driving towards 400Mb/s X32 128Mb DDR SDRAM. The attributes of RAMBUS DRAM were compared to the high end X32 DDR SDRAM. A decision was made to switch the L4 cache data technology to the X32 DDR SDRAM. The biggest factor in making the switch was that the X32 DDR SDRAM reduced the L4 cache access latency by six cycles. The six cycle improvement was achieved by not going through the chip RAMBUS command logic on the outbound path (one cycle), not going through the chip RAMBUS data logic on the inbound path (one cycle), not serializing the command on the command bus (one cycle), not serializing the critical data and the data bus (one cycle), and using the improved DRAM timings of the X32 DDR SDRAM (two cycles).

Scalable Coherence Protocol

The coherence protocol used by CSC32/CSC64 between nodes is directory based. The Scalability Directory or S-Directory is a type of limited partitioned memory directory.

The Scalability Directory is limited because this structure maintains a 'cache' of memory address tags which allows only a portion of local memory to be cached remotely. The goal is to provide enough entries in the cache to facilitate an acceptable level of remote caching without incurring the added cost of a fully allocated structure.

The Scalability Directory is partitioned because memory is physically distributed (partitioned) into nodes. A node maintains the portion of the S-Directory that corresponds to the portion of system memory physically resident on that node.

With other directory schemes, a transaction must be sent to the home memory node of the requested line in order to check the state. If the state of the line is not sufficient to satisfy the request, a second request must be sent to the node caching the line if it is different from the home node. If the line is indeed modified, this causes the situation where the request for the modified data is delayed by the time it takes to consult the home node. Processor to processor cache transfer rates are high and increasing with larger processor caches, especially for On-Line Transaction Processing workloads. This extra delay has a significant impact on performance.

CSC32/CSC64 Scalable Coherence Protocol eliminates consulting the home node. All internode data transfers are accomplished with direct transactions between the requester and the data owner. The Scalable Coherence Protocol separates the 'point of coherency' from the home node. The point of coherency is known as the Serializer.

Within a partition the serializer is always a CSC32/CSC64 chip. Prior to any requests or responses being sent by a CSC32/CSC64 chip, they must unanimously agree on which chip is the serializer.

- The chip that is the serializer must know it is the serializer.
- Chips that are not the serializer must know they are not the serializer.

The chip that is the serializer is said to hold the serializer "token".

- A chip that has Exclusive 'E' access always has the serializer token.
- A chip that has dirty data (with respect to memory) always has the serializer token.
- A transfer or return of data is always accompanied by the serializer token (the token always moves from current serializer (responder) to the requester).

The serializer token is either:

- In exactly one CSC32/CSC64 chip.
- In exactly one packet (in transit) between CSC32/CSC64 chips.

The scalability directory indicates whether or not the Home Node is the serializer for this cache line.

In order to keep memory coherent at any point in time for each cache line there must be exactly one piece of hardware that is the serializer for operations to that cache line. A transaction may not be allowed to complete until it has been accepted by the serializer.

The scalability directory improves performance by:

- Reducing unnecessary reads to the memory subsystem.
- Reducing unnecessary traffic on the scalability ports.

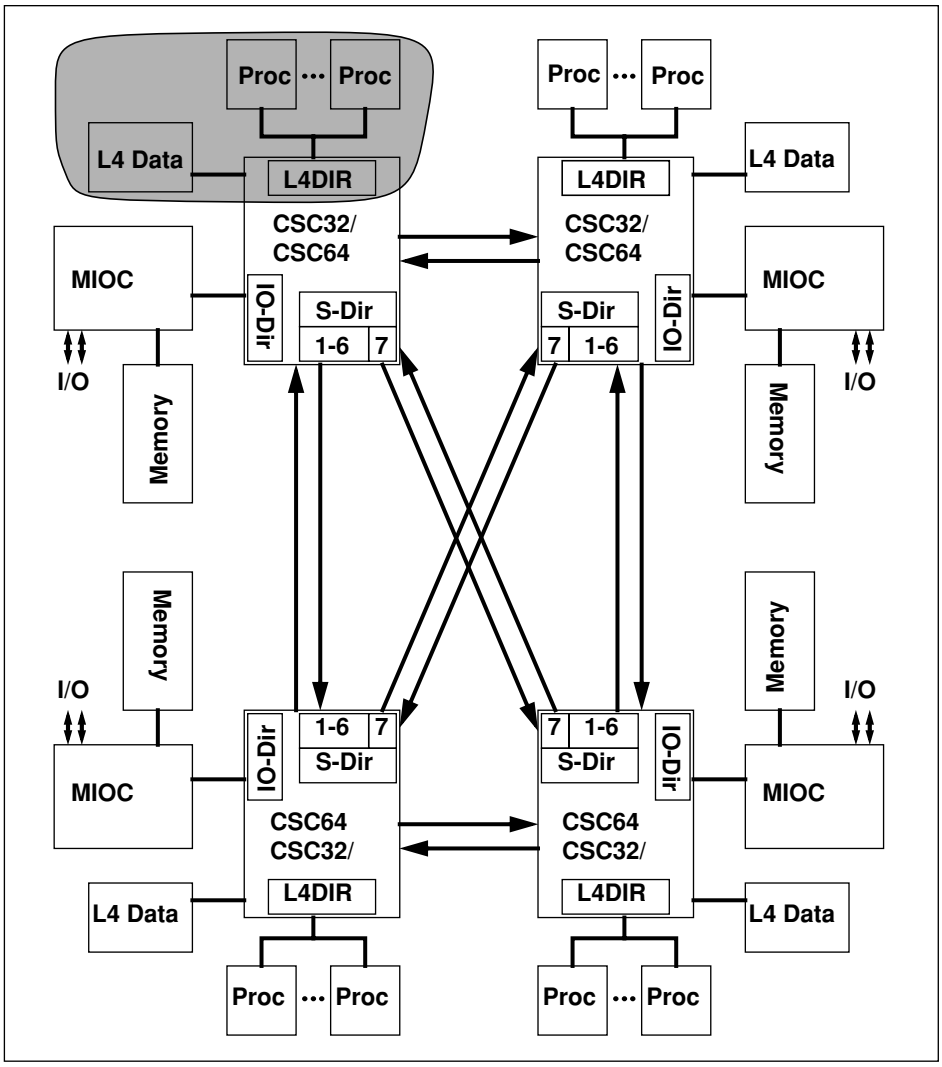


Figure 7 L4 Directory coverage

L4 Directory States

Figure 7 illustrates that the L4 Cache Directory (L4Dir) is inclusive of all Processor caches on this node. The L4 directory contains the state of the data cached in the L4 itself and the state of the data cache in all the processors. The cache state in the processors is called the Quad State. In addition to the traditional MESI states the chipset adds three new states: the T, U, and F states.

I: The L4 does not have a copy of the data for this cache line.

S: The L4 has a copy, has Shared access, this chip is not the serializer for this cache line[1], and the L4 data is "not dirty"[2] with respect to memory.

T: The L4 has a copy, has Shared access, this chip is the serializer for this cache line, and the L4 data is not dirty with respect to memory.

U: The L4 has a copy, has Shared access, this chip is the serializer for this cache line, and the L4 data is dirty with respect to memory.

E: The L4 has a copy, has Exclusive access, this chip is the serializer for this cache line, and the L4 data is not dirty with respect to memory.

F: The L4 does not have a copy, has Exclusive access, and this chip is the serializer for this cache line[3].

M: The L4 has a copy, has Exclusive access, this chip is the serializer for this cache line, and the L4 data is dirty with respect to memory.

[1] If this is the home node for this cache line and the Scalability State is not R (see section "Scalability Directory States") then this chip is the serializer for this cache line.

[2] The data in the L4 may in fact be dirty with respect to memory but this chip is not responsible for remembering this fact. If the data is dirty then some other chip is in the 'U' state and that chip is the one responsible for remembering to write the data to memory before the data is discarded

[3] The 'F' state is mainly used to improve performance of DMA store processing. When a DMA store needs to allocate a L4 directory entry in order to bring the line into the 'F' state the entry is allocated in a special "5th" class in the L4 directory that neither causes a L4 castout nor gets stored in the L4 directory EDRAMs (the entry exists only in the Directory Buffer(s) and is held there until the line returns to the 'I' state).

Quad States (L4 Directory)

The L4 directory also keeps track of the "Quad" state. The Quad state represents the maximum level of access to the cacheline that the processors on the local processor bus have. This state is used to filter commands coming from remote nodes or local I/O devices onto the local processor bus, reducing processor bus utilization.

I: No processors on this quad bus have a copy of the cache line.

S: Processors on this quad bus may be in either the S or I state. The processors cannot have a later copy of the cache line than the L4.

E: Processors on this quad bus may be in M, E, S, or I state. The Processor (if M) may have a later copy of the cache line than either the L4 or memory.

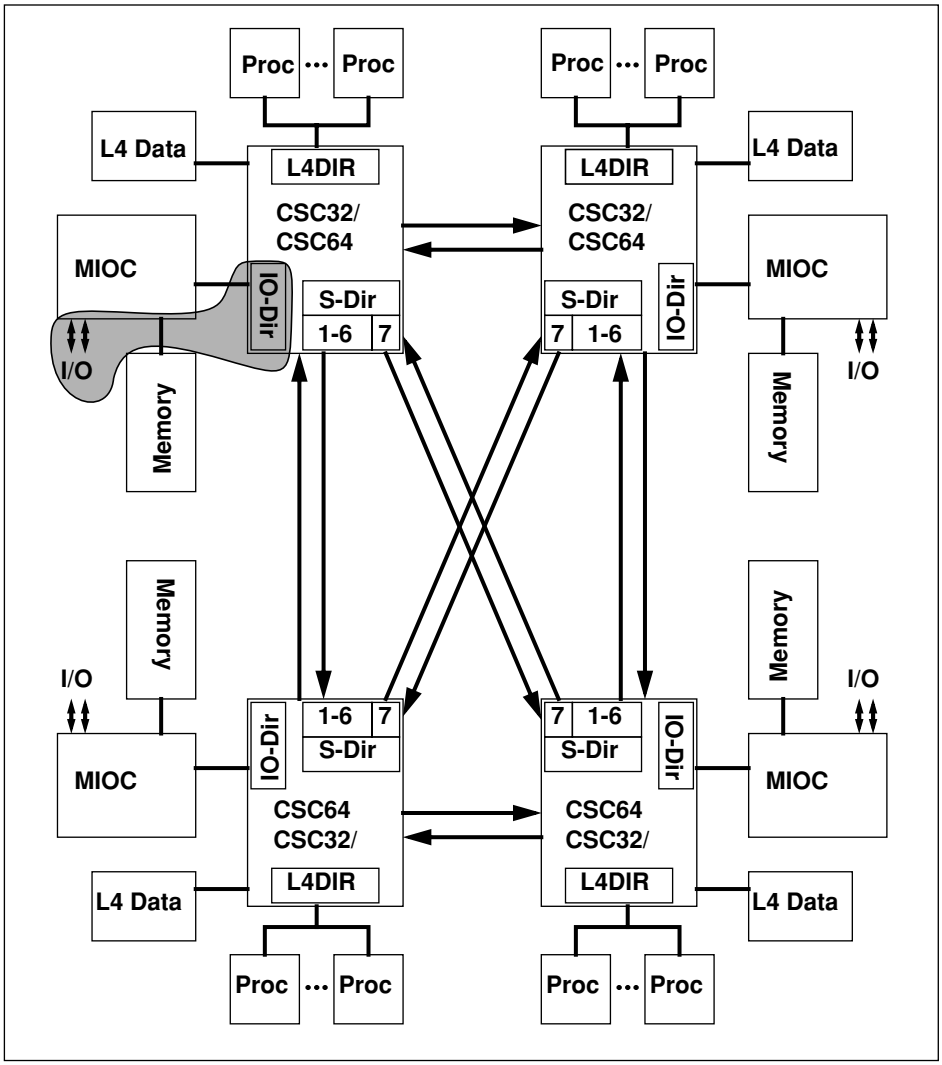


Figure 8 I/O (Page) Directory coverage

I/O Page Directory States

Figure 8 illustrates that the IO Page Directory (IO-Dir) is inclusive of all IO device caches on this node. The IO Page Directory states indicates the level of access the IO devices below this CSC32/CSC64 have.

I: None of the I/O devices below this CSC32/CSC64 have any of this page cached.

S: Zero or more of the I/O devices below this CSC32/CSC64 may have part of this page cached.

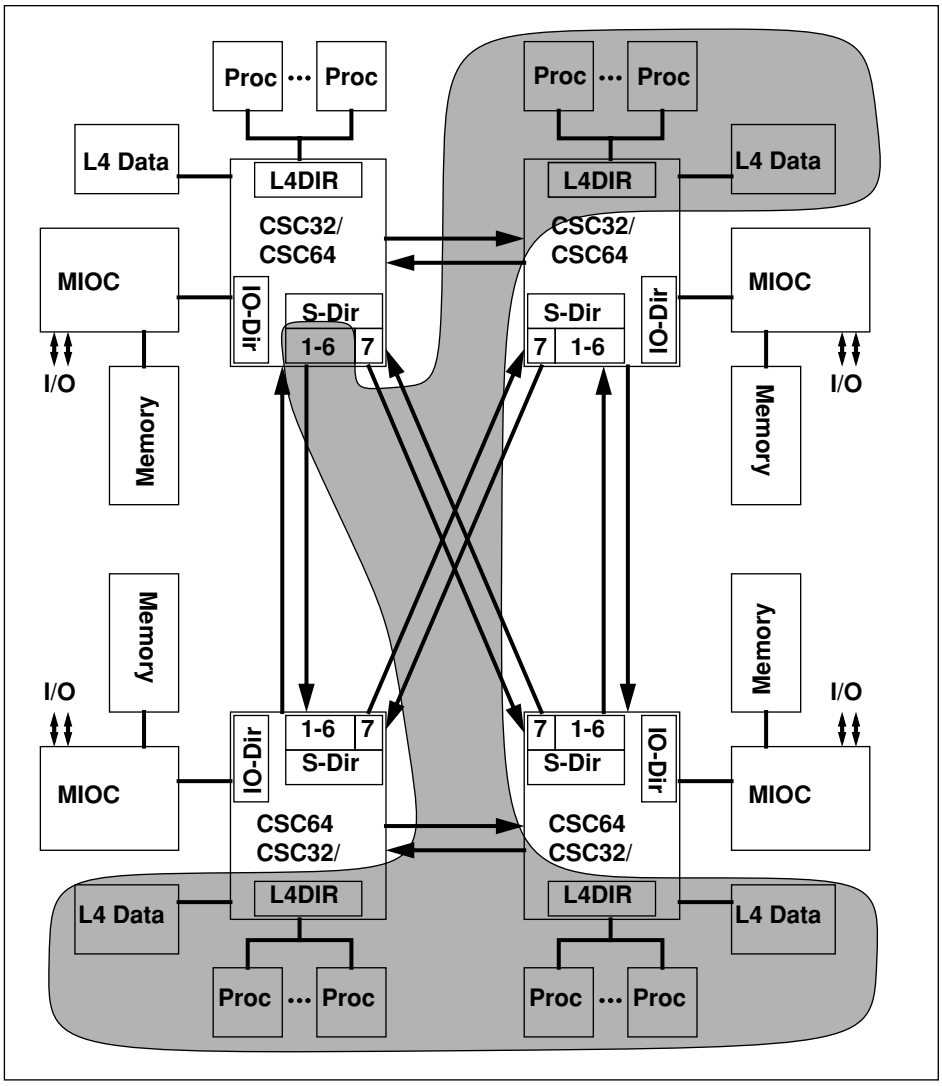


Figure 9 Scalability Directory (classes 1-6) coverage

Scalability Directory States

Figure 9 illustrates that the first six associativity sets of the Scalability Directory (S-Dir) are inclusive of all remote L4 cache and remote Processor cache. When this chip is the Home node (contains the memory for this cache line) the Scalability directory indicates whether or not a remote chip is currently the serializer and/or whether or not remote chip(s) may have shared copies of the line.

I: None of the remote chips are the serializer and no remote chips have any access (all are I).

S: None of the remote chips are the serializer but may have 'S' access.

R: One of the remote chips is the serializer and one or more remote chips are not in the invalid (I) state.

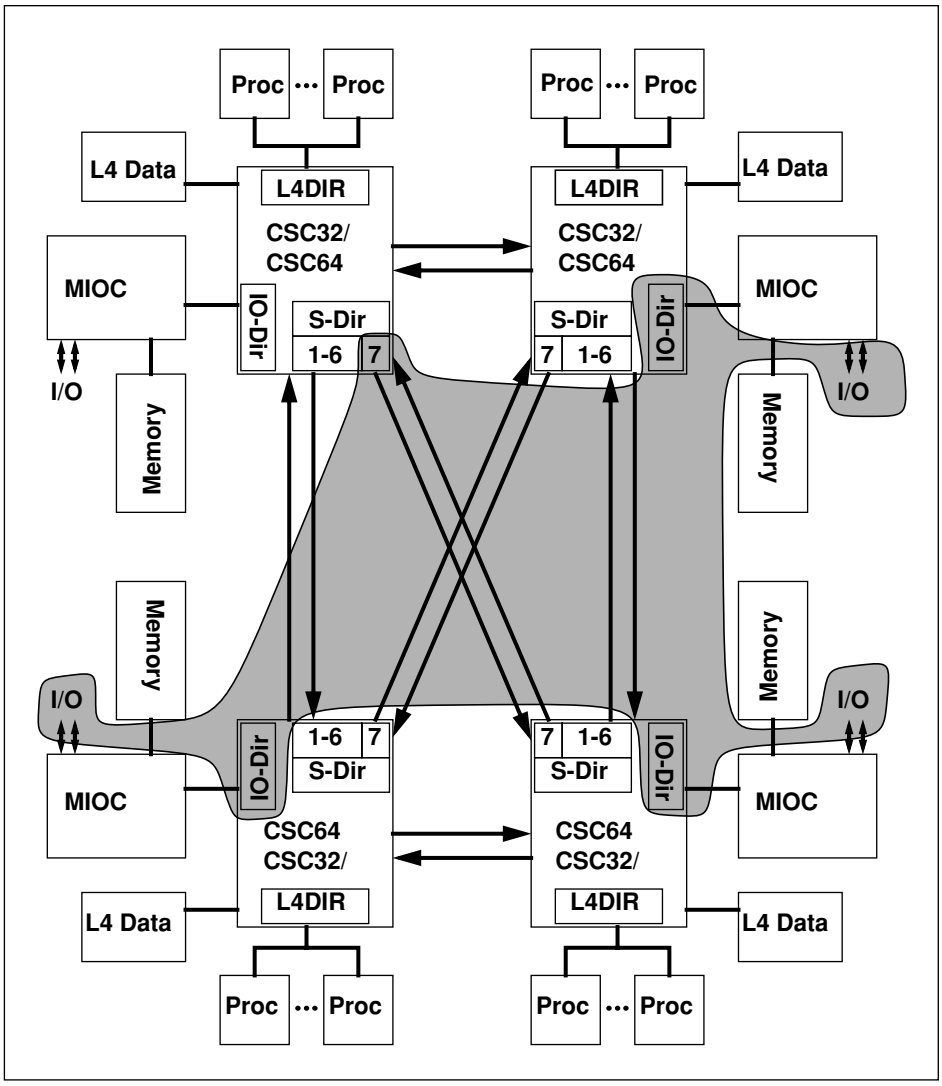


Figure 10 Scalability Directory (class 7) coverage

Figure 10 illustrates that the 7th associativity set of the Scalability Directory (S-Dir) is inclusive of all remote I/O Directories and all remote I/O Device cache. The state definitions for this special purpose associativity set are the same as all other Scalability Directory entries.

Example 1 - Data From Remote Processor Cache

Figures 11-13 show a processor on one node fetching a cache line which is initially modified in a processor cache on another (remote) node.

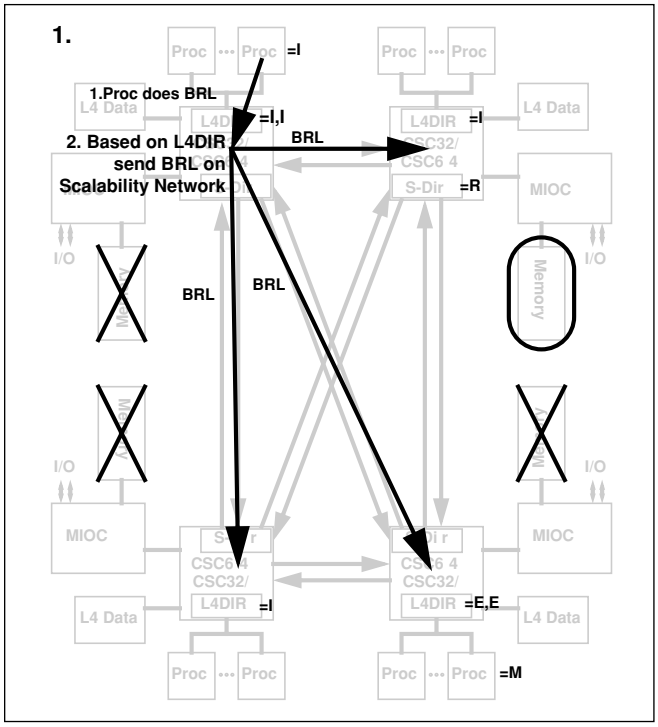


Figure 11 Remote Processor Cache Read – Part 1

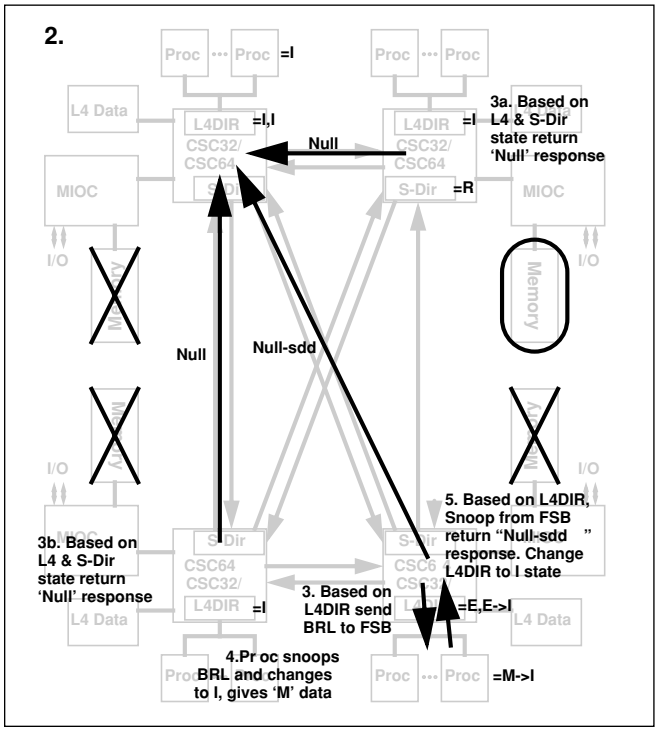


Figure 12 Remote Processor Cache Read – Part 2

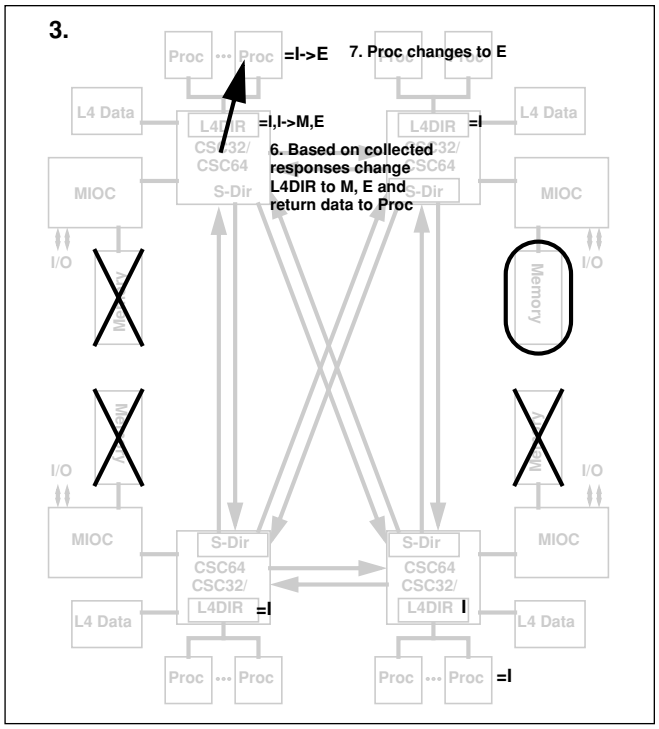


Figure 13 Remote Processor Cache Read – Part 3

Example 2 - Data From Remote Memory

Figures 14-16 show a processor on one node fetching a cache line that is not currently in any cache and therefore must be fetched from the memory on the “home” node.

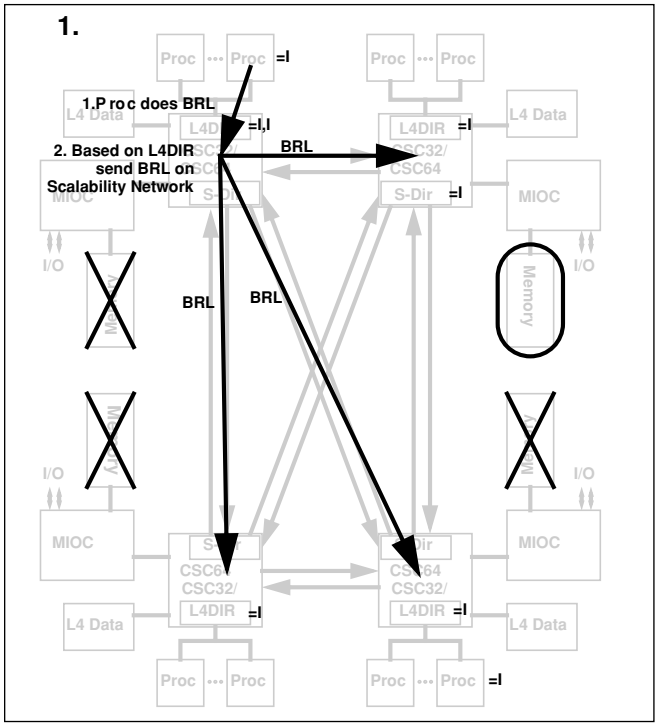


Figure 14 Remote Memory Read – Part 1

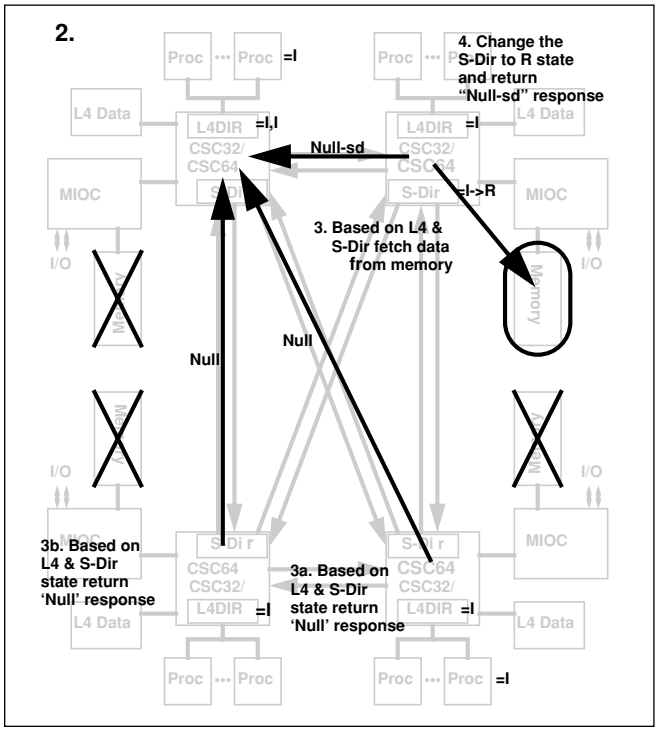


Figure 15 Remote Memory Read – Part 2

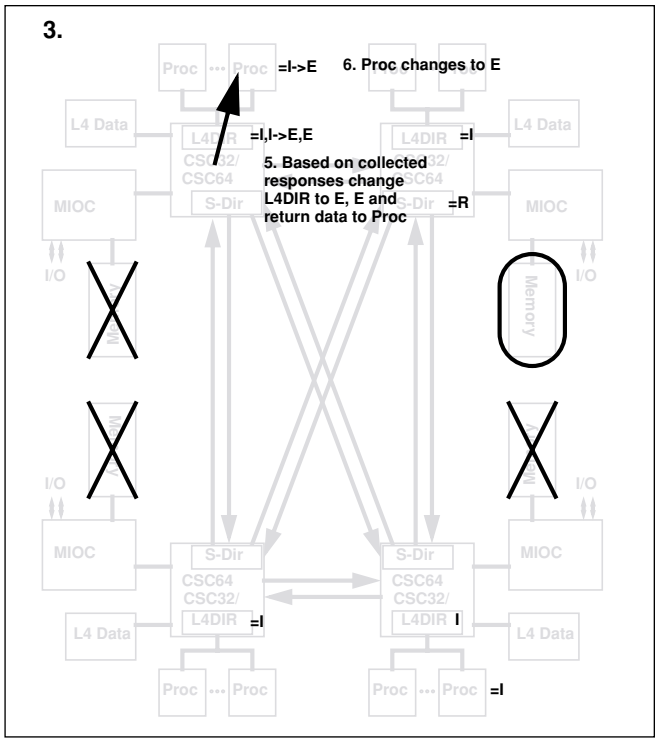


Figure 16 Remote Memory Read – Part 3

Inter-process Communication

Inter-process communication (IPC) is a term used to describe the communication between two or more processes in independent address spaces. IPC can also refer to inter-partition communication, which is a low latency inter-node communication mechanism used to provide support for a System Area Network. CSC32 and CSC64 implement a set of features that support high performance network functions. These features are:

- User Level Remote DMA
- User Level Inter-partition Messaging
- Memory Mapped Remote Load/Store

User Level Remote DMA is provided to transfer the contents of an arbitrary sized block of memory in Partition A to or from memory in Partition B. This function is optimized for large block data transfers between nodes.

User level Inter-partition Messaging provides the ability to transport a short message to a remote partition followed by an interrupt.

Memory mapped remote load/store is used to address memory on another partition but within a defined address range. A portion of the local address space is mapped via CSC32/CSC64 hardware translation onto a physical address in a remote partition.

All these features enable a low latency communication path between two partitions enabling a high performance network.

Reliability And Serviceability

The CSC32/CSC64 chips were designed with mainframe level RAS (Reliability and Serviceability) as a primary focus from the bottom up. The CSC32/CSC64 engineering team's background is in IBM mainframe design, making focus on RAS a natural fit.

The on-chip EDRAM directories are protected by SEC/DED (Single Error Detect, Double Error Detect) ECC (Error Correction Code). The external SDRAMs used for L4 data cache are also covered by SEC/DEC ECC. Additionally, the L4 SDRAM address and command paths are protected with a new error checking algorithm. The SDRAM source synchronous strobes have checking to detect missing or extra clock strobes.

The SMP Expansion port interface design incorporated eLiza self-healing technology. Using this technology, the SMP Expansion port data paths are continually checked. If a transaction cannot complete when sent on a particular scalability port, the hardware automatically reroutes the transaction on another port. The design supports hot plugging of cables to allow on-the-fly repair and expandability. The SMP Expansion port uses CRC (Cyclic Redundancy Code) to protect against intermittent errors.

The processor and memory busses are covered by either ECC or parity. Source synchronous busses have strobe error checking. Parity is used throughout the chip on both logic and arrays. ABIST (Array Built In Self Test) and LBIST (Logic Built In Self Test) were designed into the chips.

Whenever a recoverable or non-recoverable error is detected, information pertaining to the error is recorded by the hardware and made available to system diagnostics to isolate the failing circuits. If the number of recoverable errors reaches a programmable threshold, the operator is provided with a warning. This allows for corrective action to be taken before a catastrophic error occurs.

Debug

Debug of the chipset presents a challenge due to the number of interfaces and the complexity of these chips. There are actually thousands of bits of signals that would be useful to have available at the 64-bit debug port on each chip. Bringing all these signals to a central point where, under control of a configuration register, they would be multiplexed out the 64-bit debug port is not possible due to wiring problems. Two solutions are used to solve this problem.

First, instead of bringing all signals to a central point from each logical unit in the chip, a chain is constructed. Each unit first multiplexes the signals internal to its logic and then multiplexes either its own signals or the output of the previous logical unit in the chain to the next logical unit in the chain. In this way instead of hundreds or thousands of signals being routed to the debug port, only a 64-bit bus need be routed. The order of the logical units in the chain is arranged to match the floor plan of the chip.

Second, each interface will not be active on every single cycle. For example, a new command can be driven on the processor bus at most once every two bus cycles. At the central queue in each chip that coordinates the interaction between the various interfaces, a different mechanism is used. Instead of simply multiplexing the signals onto the debug chain, each interface to the central queue is tagged with an interface ID, and events from the interfaces are fed into a FIFO (First in First Out). The output of the FIFO is then multiplexed onto the debug chain. In this way, interfaces totaling over 320 bits are able to be sent out the 64-bit debug port, giving a much more detailed picture of what is happening inside the chip during debug.

Conclusion

This article has described the CSC32 and CSC64 designs for high end xSeries servers, large-scale, shared-memory multi-processor systems with hardware cache coherence. The key means to this scalability are a large, high performance XceL4 Server Accelerator Cache, a partitioned memory directory cache coherence protocol and a scalable interconnection network. First, the design focused on reducing memory latency to keep the system performance high. Second, the design focused on providing a high degree of error detection and correction to support server class RAS capabilities.

EXA Chip Design Methodology

– John M. Borkenhagen, Anthony D. Drumm

Introduction

The continual increase in processor frequencies has made memory access latencies a larger impact to overall system performance. As a result, more focus has been put on reducing memory access latencies. One component of memory access latency is the access delay through the "chipset" that connects the processor to the memory DRAMs (Dynamic Random Access Memory). The Enterprise X-Architecture™ (EXA) chipset minimizes memory access latency with advanced silicon technology, aggressive chip clock cycle times, and design optimization of critical timing paths.

At the same time that chipset delay optimization is becoming more important, on-chip wire delays are becoming more of a problem. Physical dimensions of transistors on silicon chips are getting smaller and smaller each year. This is resulting in the number of transistors on a chip doubling approximately every 18 months (Moore's law). The width of the wire used to interconnect the transistors on chips is decreasing to support the higher wire density required by increased transistor density. The decreased wire width results in higher wire resistance. Spacing between the wires is also being decreased to increase wire density. The decreased wire spacing results in higher wire capacitance.

Silicon design implementations have long on-chip connections between groups of logic (partitions). These interconnects are composed of lengths of wire driven by repeaters. The repeaters are transistor circuits used to maintain signal integrity and provide desired timing. The delay value for the interconnect wire can be approximated as RC/2, where R is the resistance of the wire and C is the capacitance of the wire. As discussed previously, both resistance and capacitance of silicon chip wires is increasing. This means the RC wire delay is increasing. In early technologies, large transistor delays made wire delays almost negligible. In today's technologies, wire delay often dominates the delay in critical paths. Therefore, management of wire delays is critical to achieving high frequencies.

The EXA chipset was designed with an advanced IBM chip technology called SA27E. SA27E is a 0.18 micron technology that helps minimize interconnect delays by using copper instead of traditional aluminum to reduce resistance. SA27E also uses new insulator technology with a low dielectric constant to reduce capacitance. Even with this advanced technology, a new design methodology was required to account for interconnect delays so that the desired chipset delays could be achieved. The importance of this new design methodology was proportional to the chip size. The Cache/Scalability Controller 32 (13.9mm X 13.9mm) and Cache/Scalability Controller 64 (14.7mm X 14.7mm) chips were most dependent on it.

This paper describes a novel design methodology used on the EXA chipset to minimize interconnect delays. The design methodology reduced the XceL4™ Server Accelerator Cache and main store memory access latencies, resulting in improved system performance.

Floor Planning For A Flat Layout Methodology

The EXA chipset design can be broken into three phases. The Pre-PD (Pre-Physical Design) phase involves coding a design in VHDL (Very high speed integrated circuit Hardware Description Language)[1] and synthesizing the language into a transistor level design. In the PD phase, transistors are "placed" on the chip by assigning physical locations to each transistor. Routing of interconnect wires is also done in the PD phase. In the Post-PD phase, Post-PD timing is run to determine if timing objectives are met. The Post-PD timing run output is used to alter transistor placement and wire routing in the Post-PD design.

Actual interconnect delay information is not available until the Post-PD design phase. A chip design may appear to meet timing objectives using average wire delays until more accurate Post-PD delay information is available. This standard design methodology results in identifying global wire timing problems so late in the design cycle that they may not be resolved without either slowing the chip operating frequency or iterating on the design cycle and delaying the design schedule.

One way to keep interconnect delays controlled between Pre- and Post-PD is by floor planning which normally requires doing PD hierarchically instead of flat. A flat layout methodology provides many benefits. Generally speaking, optimization tools are more effective when

they see the full picture and have minimal arbitrary constraints. This is true for placement and routing tools which must construct the puzzle of what gates and macros go where on the silicon and how they are interconnected. Unhindered by hard boundaries, the placer is free to move objects into positions where wires may be more easily routed. On large, complex chips with high cell usage, this can make the difference between a design which can be routed and one which cannot.

There are costs for using a flat methodology involving both the layout tools and the logic design. The layout tools must cope with a very large problem which drives up the memory and run time consumed by the tools. These costs are mitigated somewhat by the continuous improvement in compute power available to run the tools. The logic design cost consists mainly of performance uncertainty.

In a hierarchical layout, partitions are floor planned to specific, hard-bounded regions of the chip. Signals which remain within a partition (local) can be assumed to fall into a delay range depending on the size of the partition, while interpartition (global) signals are identified early and can be modeled fairly accurately. This information is very important for high-speed designs; it becomes increasingly important as feature sizes decrease and wires account for a larger percentage of delay.

However, partitions in a flat layout are free to spread out. In the extreme case, elements of one partition may span the full chip. This behavior makes it difficult to determine which signals can be considered local and which can be considered global. Signals which remain entirely within a partition may reach from one corner of the chip to another, while signals interconnecting partitions might be connecting adjacent gates. Thus, the delay characteristics of the signals is unknown until placement is complete, and the possible variation in delay is extreme.

The use of conservative wire delay estimates before placement and logic optimization techniques such as buffer insertion and deletion after placement can reduce this effect, but it presents a difficult problem to designers of high-speed logic. Designers often trade the improved optimization and routability of a flat layout for the early delay knowledge provided by a hierarchical layout.

Rather than make this tradeoff, the EXA chipset designers chose a hybrid methodology in which floor planning was used, but the layout tools operated on the full flat design keeping important partitions within overlapping regions. The placement tool had considerable freedom, yet the partitions were left substantially intact and in regions approximating the floor plan. The problem then shifted to finding an efficient way to account for global wire delays, especially during full, flat chip timing analysis prior to placement. The solution to this problem will be detailed in the next section, entitled “Automated Accounting for Global Wire Delays”.

Pre-PD logic synthesis was done at the logic partition level. The individually synthesized logic partitions were then stitched together in preparation for entering the PD stage. Synthesis was done using a standard cell library augmented with custom macros for arrays and timing critical function such as ECC (Error Correction Code) logic. Chip floor planning was done to place chip I/Os and large macros including arrays, EDRAM (Embedded Dynamic Random Access Memory), CAMs (Content Addressable Cache), PLL (Phase Locked Loop) circuitry, and custom macros.

The chip micro architecture was designed to minimize interconnect delays. For example, architected register read/write logic was implemented with a daisy chain approach. The daisy chain started in the partition that controlled register read/writes and passed through each partition using forwarding. The daisy chain approach resulted in fewer global interconnect wires than the more traditional approach of implementing the connections as a star configuration. In a star configuration, each partition has connections to and from the controlling partition. The concept of minimizing interconnects was carried through the mirco architecture phase into the high level definition phase of the design. Before the start of VHDL writing, logic partitions were manually defined in the high level definition phase with a focus on minimizing interconnects between partitions.

Partitions needed to be defined to fit within a specified maximum area limit. A maximum partition area limit was required to prevent large wire delays from occurring within a partition. The design methodology did not provide a way to account for long wire delays within a partition. The maximum partition size limit was set at $3mm^2$. The $3mm^2$ value was somewhat arbitrary, but the 5ns chip cycle time and the wire delay of SA27E technology were taken into consideration to arrive at that value. Some logic partitions exceeded the $3mm^2$ area limit. Those partitions had the most difficulty with post PD timing closure.

Figure 1 shows an example of a floor plan that was used to help guide physical design for the Cache/Scalability Controller 32 chip.

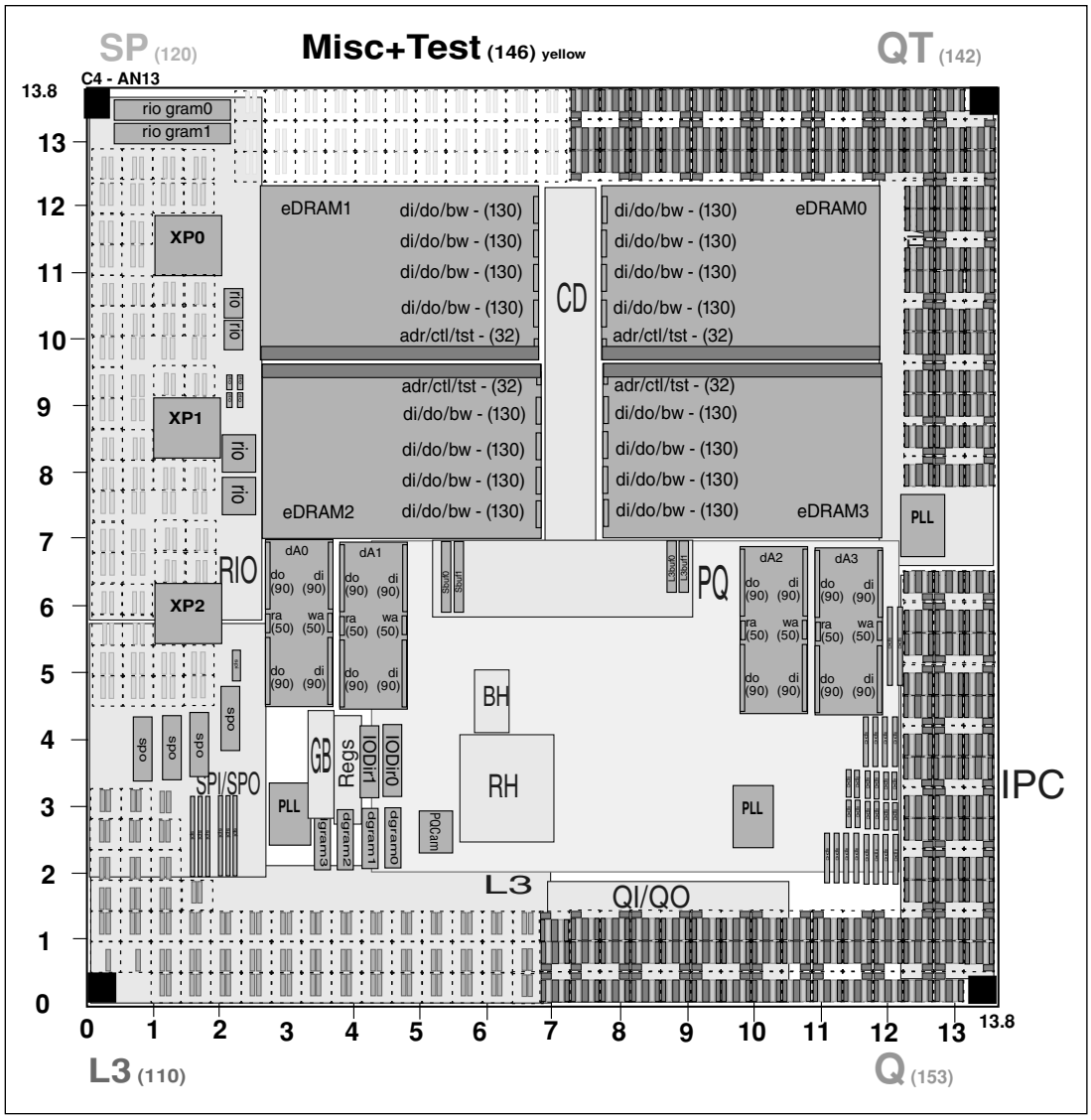


Figure 1 Cache/Scalability Controller 32 Chip Floor plan

Automated Accounting For Global Wire Delays

The partition to partition distance information from the floor plan was documented as a distance grid in a text file. The same text document was used to hold the technology unit wire delay. This information was used in the automatic generation of interpartition delays as described later in this section. Figure 2 is an example of a text file that held the partition interconnect distances and unit wire delay for the Cache/Scalability Controller 32 chip.

Twister Partition to P artition Distances (Center to Center)

	BH	CD	DA01	DA23	DB	IPC	L3	PQ	QI	QO	QTI	QTO	REG	RIO	RH	SPI	SPO	TIO
BH	0																	
CD	6	1																
DA01	8	8	0															
DA23	8	8	7	0														
DB	8	8	6	8	0													
IPC	6	11	12	5	10	1												
L3	6	9	5	11	6	12	1											
PQ	3	3	8	8	6	5	6	4										
QI	6	9	10	7	8	6	6	6	0									
QO	6	9	10	7	8	6	6	6	1	0								
QTI	10	7	12	8	14	8	16	10	12	12	0							
QTO	10	7	12	8	14	8	16	10	12	12	1	0						
REG	3	7	9	7	6	5	6	6	4	4	12	12	0					
RIO	12	10	8	14	8	14	12	10	16	16	16	16	14	1				
RH	3	6	6	8	6	5	5	3	3	3	9	9	4	12	0			
SPI	8	12	5	12	4	12	6	6	10	10	14	14	8	4	7	0		
SPO	8	12	5	12	4	12	6	6	10	10	14	14	8	4	7	1	0	
TIO	8	8	8	8	8	8	6	8	6	6	6	6	8	6	8	8	8	0

Technology delay per mm, including wire and redrive circuit

pS/mm	110

Figure 2 Distance Grid and ps/mm

The typical method to design a hierarchy of partitions uses what might be described as a block-based methodology: each block is described along with its ports, then the blocks are connected together, one-by-one. However, this view of the design is not necessarily natural for designers, especially at the higher levels of the design. Designers, instead, tend to think in terms of the signals. They communicate to one another about the signals each is either providing or using. The attributes of a partition someone else is designing are of less importance than the attributes of the signals that person is providing to the subject partition.

A signal-based methodology was constructed in the following way. Text files were created, called I/O files, which each contain information about related signals. The I/O files served two purposes. First, they were the source used for automatic generation of VHDL entities for each logic partition. A program reads the I/O files and builds a VHDL entity for each partition by generating VHDL port statements and timing attribute statements (e.g. arrival times and required arrival times) for each of the partitions connected by a signal. Secondly, the I/O files were the holding place for the official logical definition of each inter-partition signal that the owners of

each partition designed to. The connectivity of the design was managed from these I/O files. If the characteristics of a signal changed, such as its connectivity, size, or timing, the change was made to the appropriate I/O file and automatically propagated to the affected partitions.

Partition input and output timing attributes were included in the I/O files. I/O file timing attributes were critical in helping direct partition-level logic synthesis to appropriately optimize paths with global wire delays. The timing attributes were also used for performing timing analysis at the partition level. Output timing assertions were coded as a fixed time for the driving partition. The fixed output times were determined by the designer of the driving partition based on the amount of logic between a clocked latch and the output. The base design convention was to drive all partition outputs directly from a latch, but there were cases that did not follow this convention. The input times to the receiving partitions were coded as the output time along with special syntax to symbolically represent inter-partition delay.

A script was used to extract the length table from the documentation (Figure 2) and convert it to a text file. As VHDL timing attributes were generated during automatic entity creation using the I/O files, the text file was examined and used to replace the symbols in the I/O file timing assertions with the actual partition to partition delays. The script took the distance between the partitions (in mm) and multiplied it by the unit wire delay (in ps/mm) found in the same text file. The product of this multiplication was the global interconnect delay. The global interconnect delay was added to the output time of the signal to generate the input time at the receiving partition. Coding the delay between partitions in this manner instead of using a constant allowed floor plan changes to be updated in the distance grid and automatically be reflected in the VHDL without having to manually change the timing assertions for each interpartition signal in the I/O files.

Figure 3 is an example of signal connection and timing information coded in an I/O file. The timing information is coded with the syntax “9.0+@CD_L3” for a signal named “HITL3” that leaves the CD partition at time 9.0ns. “@” is a keyword used by the entity generation code to create an input arrival time to the L3 partition. The “@” symbol tells the code to go the distance grid and ps/mm table shown in Figure 2, find the distance in mm between CD and L3 partitions, multiply that distance by the ps/mm value in the table, add the result to 9.0ns, and use the final sum as the HITL3 arrival time to the L3 partition.

```
CD:   ?Port HITL3          buffer;
L3:   ?Port HITL3          in;

CD:   ?timechk HITL3       out(9.0);
L3:   ?timechk HITL3       in(9.0+@CD_L3);
```

Figure 3 I/O File Attribute coding

Timing analysis of the full chip required an additional step. Assertions at the partition boundaries were not used during full chip timing analysis. Instead, the actual times propagated through intrapartition logic were used since they represent more accurate computed data rather than the estimates used to generate the assertions. However, the interpartition delay information is lost. Another script was used to overcome this problem. The script read the I/O files and the text file generated from the partition table, as before. It output a file of timing adjusts for each global signal which the timer uses to adjust propagated times traversing those signals. These timing adjusts added delay to the global signals corresponding to the interpartition distances.

Optimizing Technology Wire Delay

In order to achieve the lowest possible memory access latencies on the EXA chips, it was very important to achieve the optimal delay per unit distance on global wires. Sample signals were analyzed for various lengths to derive the delay/length value. Inverters were added to the signals at intervals needed to keep the slew time within desired limits. These delays were found to be excessive due to the gate delays of the inverters. The standard large inverters in the technology had been designed to minimize input pin capacitance; they were actually three stages rather than a single stage. Two new single-stage inverters were designed specifically to serve as long wire repeaters, one for standard width wires and one for wide wires. Once the sample signals were reanalyzed, an acceptable delay/length was produced. Table 1 is a comparison of the ps/mm delays for between the inverter offered in the standard technology library and the inverters optimized for redrive.

Table 1 Redrive Inverter Delay Comparison

Inverter Type	Wire Width	Opt. Inv Spacing	Delay with Inv.
Library offering	1X	1.6mm	160ps/mm
ROInv 60/30	1X	1mm	133ps/mm
ROInv 90/45	2X	1.3mm	105ps/mm

Physical Layout

Several methods were used to close timing during physical layout of the chips. As stated earlier, placement and routing were run on the flat chip design. This provides these tools with the flexibility they need to find a workable solution. The partition floor plan was used, though, to guide placement which was restricted in some cases to keeping certain partitions within prescribed (but overlapping) regions. This helped to ensure the global signal delay estimates used before placement would be reasonably close to the actual, post-placement delays. Capacitance targets for placement were generated from a full chip timing analysis. As stated earlier, global signal delays were considered during timing analysis. About 20% of the nets were given capacitance targets. Placement was done using MCPLACE, a simulated annealing placer[2].

Finding an ideal floor plan is difficult for high performance designs. In addition to the logic partitions, large memory macros such as EDRAMs must be placed carefully to keep connected wires short while allowing sufficient space for other wires to cross. Various floor plans were tried by running placement for each and comparing the results.

Timing correction during logic synthesis uses estimates for wire delays since no physical information is yet available. Although the method used to account for global wire delays and the floor plan-based region constraints helped avoid making gross estimation errors for long wires, many of the partitions were large and some were permitted to spread over a large area. Also, adding repeaters to the long global wires could not be done properly prior to placing the chip.

Two IBM optimization tools were used to resolve these problems by analyzing the timing after placement and re-optimizing the design. TDCopt (Timing Driven Control Optimization) has been used in this capacity since 1995. It uses timing correction transforms from the BooleDozer logic synthesis system[3] (which is integrated with the EinsTimer static timer) along with a special subsystem to compute wire delays based on placement and to incrementally place new logic cells. PDS (Placement Driven Synthesis)[4] is a newer tool using many of the TDCopt components but integrating components of ChipBench[5] including the CPLACE placer. CPLACE, unlike MCPLACE, uses a partition-based placement method rather than simulated annealing. Partitioning allows for closer integration and interaction with the optimization functions. PDS enables a larger suite of optimization techniques and generally has more physically-aware transformations.

An important optimization technique is buffer insertion to reduce capacitance and slew and to repeat signals on long wires[6]. A key optimization method used for the EXA chips is BuffOpt[7] which finds an optimal buffering solution given a set of inverting and noninverting buffers, a net, and a prescribed route. We found that BuffOpt consistently met the predetermined delay/length value used for all global signals. This was extremely important in closing timing for these chips. BuffOpt was run in both TDCopt and PDS.

Manual Optimizations

Some manual changes were used to complete timing closure. These changes include buffering with wide wires on selected critical nets as well as moving some latches physically or temporally to balance slacks. Use of wide wires has generally been specified manually to maintain control over the potential routing problems the wide wires may cause. BuffOpt and PDS will incorporate automatic wire sizing in the near future.

There are several ways to balance timing around memory elements in a latched design. Two were used for the EXA chips. Some latches were physically moved to reduce wire delay on one side of the latch while increasing it on the other side. In other cases, the clock signals to the latches were adjusted to arrive either before or after the nominal clock arrival time.

Congestion And Other Problems

Wiring congestion became a problem in certain regions. In some cases, this was related to the buffers inserted to reduce slew and RC delay. Buffers inserted by BuffOpt were sometimes at fault due to particular behavior which became known as the L problem.

BuffOpt finds an optimal buffering solution along a given path, usually a Steiner tree. Consider a large bus of

two-pin nets crossing a large distance from one region of the chip to another. Those wires may be routed anywhere within the rectangle enclosing the sources and sinks with no change to the lengths of the wires. This results in a good distribution of the routes and frees the router to find viable paths. However, if those nets require buffering, BuffOpt will be given a set of nearly identical Steiner routes (in the shape of an L) upon which it will add buffers. The existence of these buffers will force routing to follow the L to route through the buffers. Thus all wires are forced through the same small regions.

Local area congestion was also a concern. The areas around large arrays, such as the EDRAMs, were densely populated by buffers and inverters used to repeat signals which are routed across or around the arrays. This congestion prevented later insertion of cells needed to fix hold violations or logic bugs.

Buffers were moved manually to avoid the wire congestion caused by the L problem or to free up space around arrays. Placement halos were added around large arrays forcing placement to avoid those areas and leaving space for the buffers added later by TDCopt and PDS.

Manual mitigation of congestion problems is time consuming. For the second pass of the Cache/Scalability Controller 64 chip, it was decided that all changes would be done using Engineering Change Orders (ECOs) applied to the pass one net list rather than restarting from scratch. The changes were fortunately contained enough to allow this approach to work, saving a lot of effort and time from the schedule. Meanwhile, enhancements have been made to PDS to avoid many of these problems in the future.

Clock Distribution

The physical layout of the EXA chipset required a unique approach to implementing the clock tree. This was mainly driven by the large EDRAMs which not only prevented placement of an H tree, but also made it impossible to route the clocks across them without causing slew problems. High frequency and low skew requirements further complicated the clock tree distribution. Floor planning and preplacement had to be done on all of the large macros.

Once this was done, a custom clock distribution was placed around them. Several iterations were required to determine how many stages of the clock tree were required and where the optimal locations for the stages would be. Once determined, the first n stages of the clock tree were fixed in place and manually wired (due to the nature of the large clock driver books) to produce minimal skew. These stages were implemented with large clock driver books tapered down in drive strength from one stage to the next.

Strong books could drive fatter wires, and ultimately cover larger distances. This ultimately enabled the clock tree to be implemented with fewer stages, reducing the overall latency of the tree. Use of inverting books also helped to offset pulse-width distortion. The final stages of the clock tree were repowered and placed automatically using Clock Designer[8] (a parallel clock repowering tool) and then routed using Scorpion (a balance router). Many iterations were required, and the process was repeated for multiple clock domains. Ultimately a minimal latency, low skew clock tree was obtained.

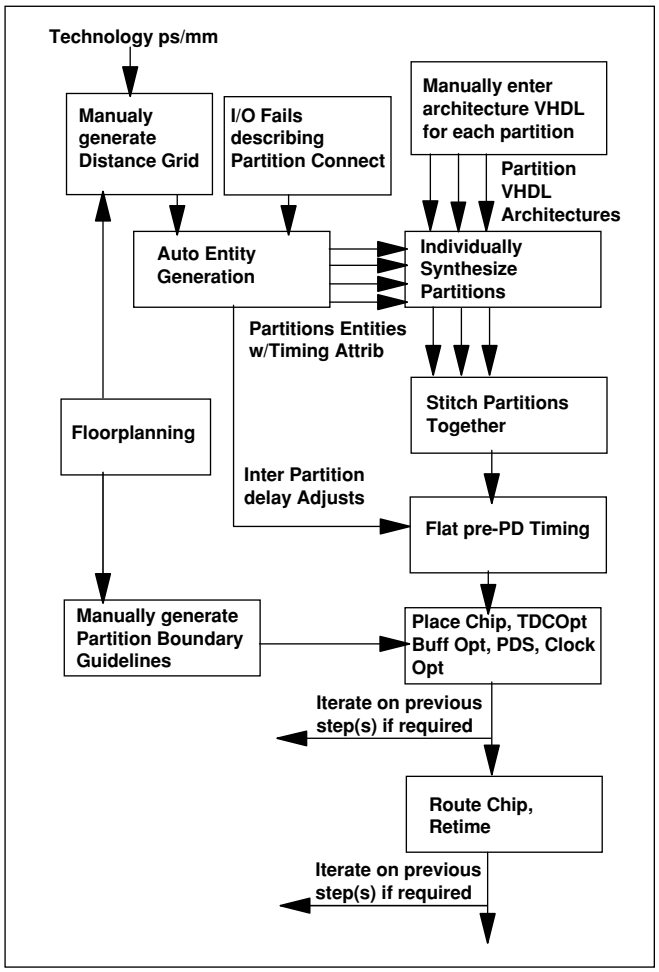


Figure 4 EXA Chip Design Methodology

Conclusion

In today's technologies, management of global wire delays is critical to achieving high frequencies. Global wire delays need to be accounted throughout the design methodology. The EXA chipset used a hybrid methodology that combined both flat and hierarchical attributes to manage global wire delays. In this methodology, global wire delays were managed from the very start of the design all the way through the final stages of the design. The end result was the EXA chipset achieving desired memory access latencies on a compressed schedule. Figure 4 is a flowchart of the EXA chip design methodology.

Acknowledgements

The design of the Enterprise X-Architecture chipset involved an extensive team of engineers and programmers. The list is too long to acknowledge all names in this paper. The authors would like to give special thanks to the following people for their support with the design methodology. Bob Lembach, Don Eisenmenger, Bruce Winter, Sean Evans, Bruce Rudolph, Kerry Pfarr, Nghia Phan, Scott Frei, and Matt Ellavsky from the physical design and clock teams. Jeremy Leitzen, Brian Wilson, John Zack, Charles Alpert, Stephen Quay, and Andrew Sullivan from the tools group. Phil Hillier, Todd Greenfield, Ken Valk, Russ Hoover, and Jeff Brown from the chip design team.

References

1. IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993, 1994.
2. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi, *Optimization by Simulated Annealing*, Science, Volume 220, Number 4598. May 13, 1983.
3. L. Stok, D.S. Kung, D. Brand, A.D. Drumm, A.J. Sullivan, L.N. Reddy, N. Hieter, D.J. Geiger, H.H. Chao, and P.J. Osler, *BooleDozer: Logic Synthesis for ASICs*, IBM Journal of Research and Development, vol. 40, no. 4, p. 407-430, July 1996.
4. P. Kudva, W. Donath, L.M Reddy, L. Stok, P Villarubia, *Transformational Placement and Synthesis*, Proc of Design Automation and Test in Europe 2000, March 2000.
5. John Sayah et al, *Design Planning for high performance ASICs*, IBM Journal of Research and Development, vol 40, no. 4, July 1996.
6. L.P.P.P. van Ginneken, *Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay*, 1990 International Symposium on Circuits and Systems, p. 865-868.
7. C.J. Alpert, A. Devgan, S.T. Quay, *Buffer insertion with accurate gate and interconnect delay computation*, 36th Design Automation Conference, 1999, Proceedings p. 479-484.
8. D.J. Hathaway, R.R. Habra, E.C. Schanzenbach, and S.J. Rothman, *Circuit Placement, Chip Optimization, and Wire Routing for IBM IC Technology*, IBM Journal of Research and Development, vol. 40, no. 4, p. 453-460, 4 July 1996.

EXA Chipset Verification

– Wayne Barrett, Kyle Nelson, Pete Thomsen, Kenneth Valk, David Wood

Introduction

The verification effort for the Enterprise X-Architecture (EXA) chipset focused on four chips: PCI-X I/O Bridge (IOB), Memory/IO Controller (MIOC), Cache/Scalability Controller 32 (CSC32) and Cache/Scalability Controller 64 (CSC64). This paper explores pre-silicon verification and post-silicon validation of these four chips.

Pre-silicon verification was completed in three different phases: formal verification, partition verification and chip verification. The first phase, formal verification, was used to verify the high-level design of the system's complex coherency protocol design. This verification effort early in the design cycle made it a useful aid in design decisions. The next phase, partition verification, verified that each of the various logic partitions in the chip would function correctly in the chip as a whole. The final phase, chip verification, included both single and multi-chip verification environments. Cycle simulation engines were used exclusively for this effort.

The chips in this chipset were designed to be modular, flexible, and scalable. This paper shows how chip level verification environments took advantage of the forethought put into the EXA system.

The final verification effort is post-silicon lab validation. The post-silicon validation effort tapped IBM's @server validation technologies that have been used to produce many of the most reliable systems in the industry.

Formal Verification

For this project, formal methods were used primarily to verify the high-level design of the system's coherency protocol due to the complex nature of its algorithms. Maintaining coherency is a difficult task for a system in which data can be cached simultaneously across multiple nodes and at various levels in the memory hierarchy. The coherency protocol must simultaneously process a number of transactions over multiple interfaces. The goal of formal verification at this high level of design was to generate a complete and verified specification which could be used as a basis for the Very high speed integrated circuit Hardware Description Language (VHDL) implementation.

Formal Methodology

The methodology consisted of three steps. First, the specification was formally described in an algorithmic manner. Next, the specification was modeled and formally verified. Finally, the specification was automatically translated into VHDL, yielding hardware correct by construction. Each of these three steps is discussed in detail below.

Algorithmic Specification Of The Hardware

Typically, hardware specifications inadequately describe the algorithms the hardware intends to implement. The functionality of the hardware is described in detail, but the underlying algorithms are often oversimplified or viewed as implementation details. This results in algorithmic bugs being found much later in the design cycle than necessary. Writing a hardware, or algorithmic, specification that describes the underlying algorithms with enough detail to enable formal modeling required a new approach. In defining an algorithmic specification, a methodology was developed for describing the specification despite the inherent complexity associated with coherency protocols.

The complexity of coherency protocols, and control code in general, is due to the simultaneous processing of concurrent transactions across several interfaces. Moreover, the processing of a given transaction often consists of several intermediate steps which may be processed concurrently. The methodology employed to define the specification leveraged this insight and modeled the hardware's behavior as a set of communicating processes ("sequencers"), each of which dealt with the transactions at a particular interface. Each sequencer, in turn, was comprised of smaller intermediate processes which corresponded to the steps of the algorithm that the design was to implement. Sequencers and processes interacted with each other via shared variables on a cycle-by-cycle basis. At the beginning of each cycle, zero or more processes within an active sequencer could be triggered.

As part of the modeling process, the control path and data path are distinguished from each other. Representing data in an abstracted manner is a standard

technique used in formal modeling to reduce model size. The modeling process also distinguished between "core" algorithmic function and "bookkeeping" function. Some examples of bookkeeping function are queue entry selection, buffering, address matching, and buffer allocation.

Formal Verification Of The Specification

The first step in the verification process was to translate the specification's sequencers into Environment Description Language (EDL), the language used by the formal verification tool RuleBase. Translation was an automated process made possible by the fact that the specification was written in a structured language. Next, the sequencers and their processes were tied together to create a model of the complete system. This required code that allowed the individual processes of the sequencers to communicate with each other via shared variables. Shared variables were also used to facilitate communication between the sequencers. External signals which triggered the processes were also defined. The final step was to define the interface logic and the environment.

The verification cycle started after the definition of the first system model. The model was formally verified against the expected functionality of the control code. Inconsistencies resulted in either corrections to the specification or the model. A new model was constructed and the process repeated. The final result was a formally verified algorithmic specification of the hardware.

Translating The Specification To VHDL

The translation of the specification to VHDL was similar to the EDL translation process described above. This resulted in VHDL that was, by construction, equivalent to the high-level specification. The term "by construction" is used loosely since the translation path was not proven correct. This accounted for the algorithmic function of the hardware. The bookkeeping function (address matching, buffer allocation, and the external triggers) was manually coded into VHDL.

Formal Verification Models

Formal models are typically heavily abstracted in order to avoid size problems. From the above description of the specification and the EDL model, it is evident that the model is quite detailed. Indeed, it is cycle accurate and the specification contains enough information for the VHDL code to be derived directly and automatically from the specification.

Despite the model's detail, there are two important abstractions which allow it to be model checked. First, the specification does not deal with addresses explicitly. Rather, the specification describes the behavior of the algorithm, ignoring the address. Also, the specification abstracts the data by completely ignoring the width of the data bus. Thus, the abstractions are not obtained by abstracting out the behavior of the control logic. Instead, they are obtained by reducing to a minimum the size of the address and the data. The abstraction level marks the line between the specification and the implementation.

Despite the fact that the models were cycle accurate, it was possible to create system models with multiple processes in them. That is, these models were able to process multiple commands across several different interfaces. System models which were verified using the model checker consisted of multiple instances of individual processes along with a model of the system's environment. For the coherency protocol, the processes which execute system bus commands, scalability port commands, I/O commands, and castouts were modeled. The environment modeled the processors, the system bus, I/O units, and the system's memory and L4 cache.

Execution of Formal Verification

The system models were verified with RuleBase, IBM's model checker. In model checking, system properties are written and the model checker proves whether the property holds. If the property does not hold, then a counter example which demonstrates the failure is produced. Properties that were checked were safety rules which verified:

- an unexpected state (an error state) was not reached in any of the individual process models
- correct data was written and received

Cache coherency was not validated. This takes into account the L3 state, the L4 states, and the scalability state.

Typically, hundreds of properties are checked, most of them being simple checks for error states. The remainder of the properties checked coherency and data rules.

Formal Verification Results

Over the course of the high-level design phase, 47 bugs were found in the coherency protocol's specification. Many of these bugs were corner cases involving unusual interactions between two or more of the processes. A majority of the bugs were violations of data consistency, despite the fact that cache coherency was properly maintained.

The formal design methodology had a significant impact on the design process. It drove rigor, completeness, and correctness of the high-level design. It found many errors and helped to direct changes early in the design process. It facilitated and accelerated the design teams understanding of the coherency protocol. Using this methodology lengthened the high-level design process but because the VHDL was derived from a complete algorithmic oriented specification, this time was more than made up in the VHDL coding process.

Partition Verification

Partition level verification takes on many forms since it is left up to the partition designers to implement. The level of sophistication found in this effort largely depends on the complexity of the partition. Partition verification was done with event simulators. Most efforts were basic environments that exercised a single partition. In the more complex cases several partitions were combined and simulated as a whole.

Since partition verification environments are numerous and vary greatly, only two of the more complex and innovative efforts at this verification level are explored. The first outlines the verification of the Coherence Unit (CU) found in CSC32 and CSC64. The second effort described is the QuadT bus, a critical interface between MIOC and CSCS32 (or CSC64).

CU Partition Verification

Partition verification of the CU in CSC32/CSC64 chips consisted of two stages:

1. High-level simulation of the coherence protocol
2. Simulation of the CU as a subset of the actual chip hardware design.

Simulation of Coherence Protocol

During the architecture and high-level design phases of the EXA project, a coherency protocol and coherence tables were defined. The coherency protocol, which defines how to process commands and use the coherency tables, was written in Java. By the end of high level design there were 23 tables with 3000 total rows and the coherency protocol code contained 4500 lines of code. A small portion of one of these tables is seen in Table 1.

Table 1 Example of Coherence Table

#	SB Command (Received)		SB Facilities			Directory Next State		
	Command	Variant	Hit#	HitM#	Defer#	L4 Cache	SB	Scala-bility
1	BRL	–	0	0	1	–	I	–
2	BRL	–	0	0	1	–	S	–
3	BRL	–	–	1	–	M	E	–

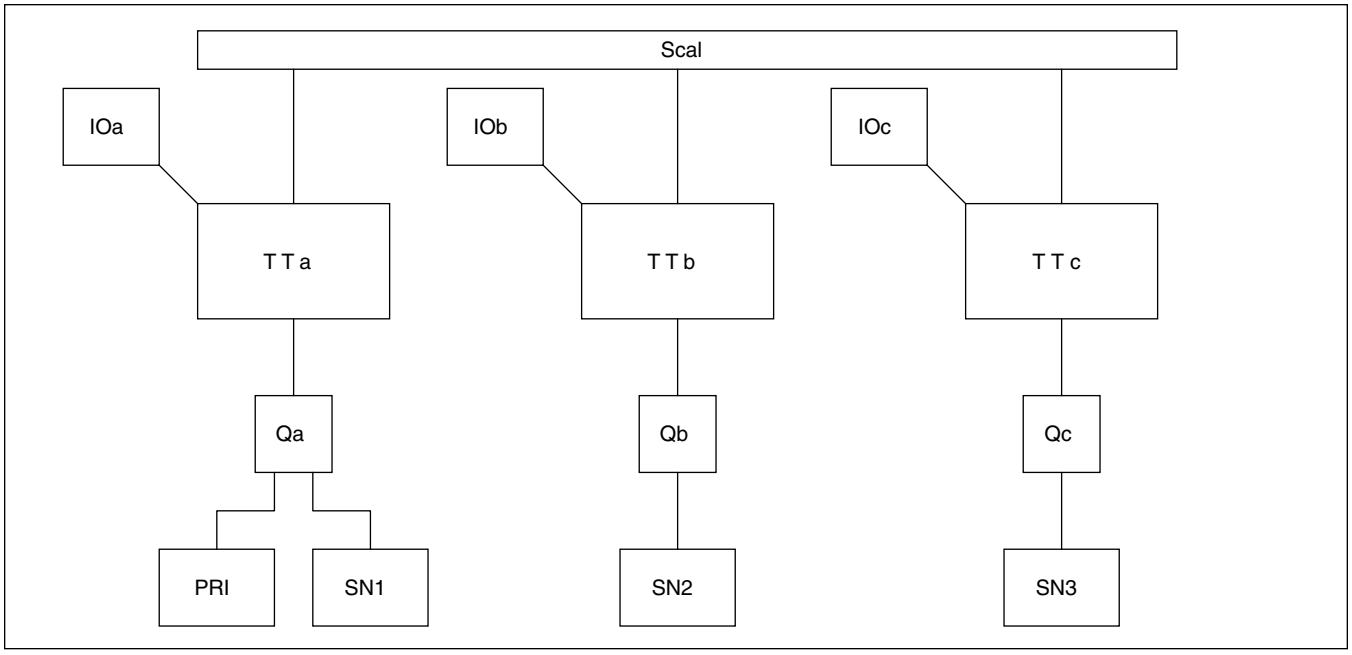


Figure 1 CU High Level simulation structure

The high level simulation model consists of a primary processor (PRI, the only processor to "master" commands), three snooper processors (SN1, SN2, SN3), three system busses (Qa, Qb, Qc), three CSC32/CSC64 models (TTa, TTb, TTc which contain the coherency model under test and glue code to interface the model to the rest of the verification environment), three IO devices (IOa, IOb, IOc), and a scalability model (Scal) used to connect nodes together. Figure 1 shows a graphic of the CU simulation structure just described.

For a single simulation scenario the simulation model is loaded with a valid combination of cache states + cache data + memory data and a single command is run from PRI (processor commands), IOa (DMA loads & stores), or TTa (L4 and S-Dir castouts). The command is run to completion then the return data (for loads) is checked to the expected data and the cache states + cache data + memory data are checked for a valid combination of state + data.

The verification environment then cycles through the following to verify the coherency protocol:

- All possible (valid) cache states for the system
- Memory "Home" location either A or B
- Memory type of WB, WT, UC (for processor transaction)
- All possible memory transactions

When completed, the high level verification environment ran 346,000 tests from the above combinations and verified that the coherency protocol and coherence tables were correct for single-command scenarios.

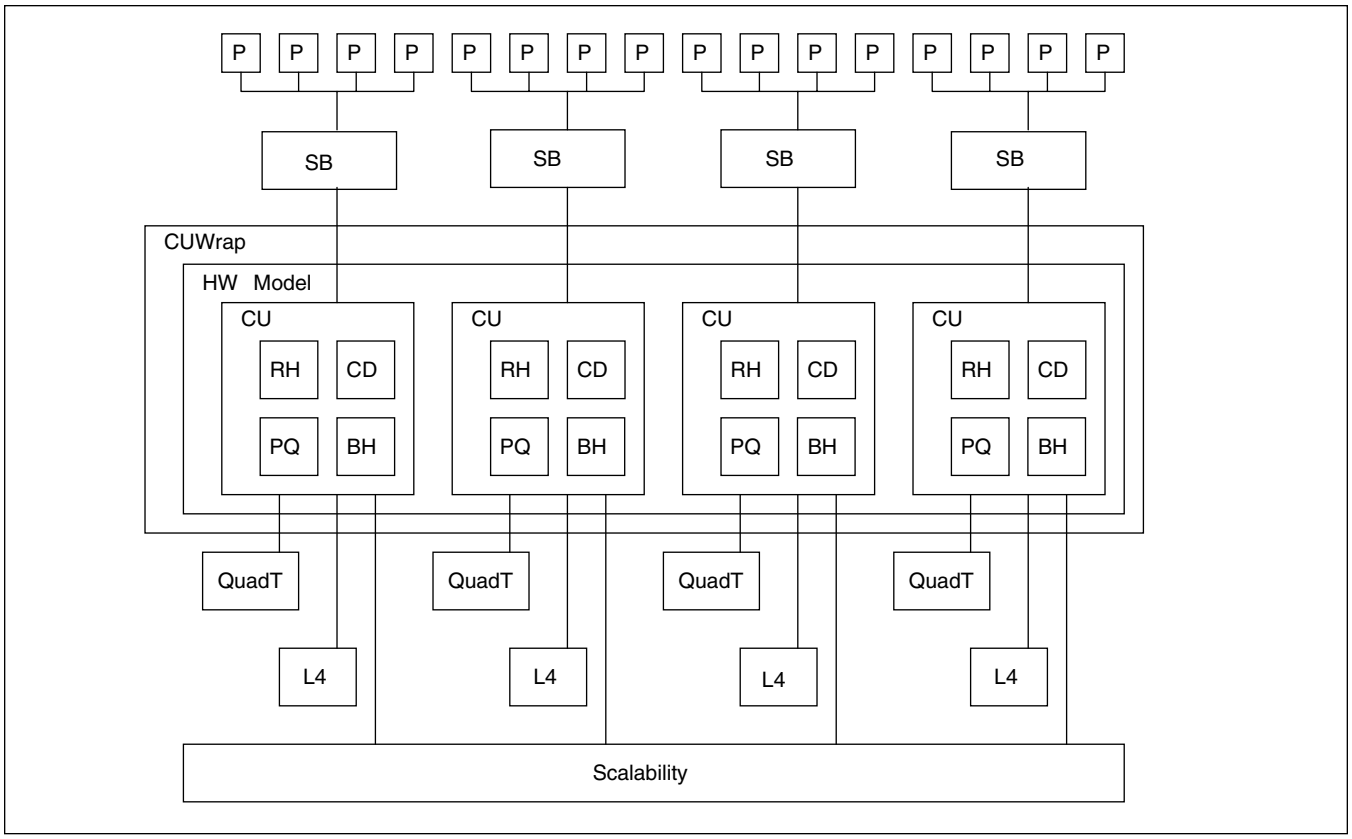


Figure 2 "CU" Partition verification structure

Simulation of CU Partition

After low level design was completed, a simulation model of the Request Handler (RH), Coherence Directory (CD), Buffer Handler (BH), and Pending Queue (PQ) units (collectively called the "CU") was created and used to verify the implemented coherency protocol for multiple commands, including commands to the same cache line and commands to the same directory row.

As shown in Figure 2, the CU partition verification environment consists of:

- Processors (P) that model either Intel Xeon™ Processor MP Family or Intel Itanium™ Processor Family processors
- System Busses (SB) that model the processor busses
- CU "Wrapper" code (CUWrap) that handles the details of interfacing between the HW Model and the rest of the verification environment
- The compiled HW model (HW Model) which contains four instances of a "CU" container which itself contains instances of the actual RH, CD, PQ, and BH partitions
- QuadT (QuadT) units that model the connection to main memory and I/O
- L4 (L4) units that model the CSC32/CSC64 L4 cache interface logic and external L4 DRAM chips
- Scalability (Scalability) units that model the connections between nodes.

Many of the models in CU partition verification are models from the high level coherency simulation that have been enhanced to handle multiple simultaneous commands and the greater level of detail present in the CU partition verification environment.

Coherence Protocol Checking

There are three main checks used to verify that a test case passes.

1. Test case time limits – Each test case specifies a limit on the # of cycles to run. Thus if the hardware "hangs" and cannot complete all commands and return to an idle state, the test case fails.
2. Hardware error checkers – The hardware has a number of built-in checkers that detect incorrect or inconsistent states. The test case fails if the hardware error checkers go off.
3. Ideal Memory – The verification environment maintains a global "Ideal Memory" that is updated on the completion of any store command and checked on the completion of any load command. If the current contents of Ideal Memory do not match the data returned to a load command then the test case fails.

High level coherency simulation verified that a testcase ended with a valid combination of states + data. CU partition verification did not perform this end-of-test-case due to the complexity required to perform the check. Any failure that would show up in end-of-test case checking would eventually happen in the middle of a test case and get caught by one of the three checks listed above.

A typical test case would first generate a set of addresses that were synonyms of each other and then generate hundreds of random commands using these addresses.

CU Partition Verification Results

At the end of CU partition verification, 56 bugs were found and corrected in the CU logic, preventing the bugs from appearing in full chip verification.

QuadT Partition Verification

The QuadT Bus is the name given to the interface between MIOC and CSC32 (or CSC64). In a multi-chip verification environment, the QuadT Bus looks like an "internal interface" of the EXA chipset. The ability to extensively stress the QuadT Bus in a multi-chip verification environment was limited. Because this interface was "removed" from the other major interfaces (System bus, I/O, etc.) a way was needed to validate (and stress) not only the chip logic connected to the QuadT bus, but also a way to validate the unique architectural functions of the QuadT bus.

To solve this, a special partition verification environment was created. This effort took into account four chip partitions. Two partitions from MIOC that constituted the QuadT bus interface logic and the two complimentary partitions from CSC32/CSC64. A "testbench" environment was then created that called in these four partitions as components. This testbench had no inputs/outputs and was a self-contained entity. The testbench controlled all of the internal chip interfaces that were exposed to these four partitions, and it also emulated the boundary-scan latches and drivers/receivers so that the virtual QuadT Bus traffic could be monitored.

QuadT Partition Verification Methodology

The only "input" to the verification environment was a random seed. With that, the testbench sourced random transactions on both ends (MIOC's and CSC32/CSC64's) with random data. As transactions passed through to the opposite side of the partition, all contents (address, data, etc.) were checked for accuracy. These transactions were "blasted" from both sides for approximately 1500 clock cycles, at which time all new transactions would stop and all pending transactions which were queued would be allowed to complete. Once the logic had settled down, the various configuration/mode settings would be randomly altered and the sequence would start over. Other "irritators", like retries and hold-off conditions, were periodically inserted. With this random, self-checking, self-transformation nature, this testbench could run continuously as long as needed.

Execution

The labor to create this testbench and to perform the desired testing was about 2-3 man-months of work, spread out over four months.

Logic simulation tools were used to run this testbench. Even though the testbench could run for a very long time, for practical reasons, the longest run times were for approximately 156,000 clock cycles (or about 100 iterations of the testing loop). This would take about 24 hours to run.

QuadT Partition Verification Results

There was a total of 42 hardware bugs found by this effort, excluding the bugs found within the testbench itself. More importantly, it was estimated that six (of the 42) bugs were, or would have been, "escapes" from the multi-chip verification work.

The QuadT partition verification produced a good "return on investment". With approximately three man-months of effort, multiple bugs that would have otherwise escaped into the laboratory were discovered. The robustness of the QuadT bus has been strong since the time this special verification effort was completed. These results validated the initial concern that the "internal" nature of the QuadT bus (with its new definition) posed a high risk of having bugs escape the multi-chip verification.

Partition Verification Results

Combined with design reviews and VHDL code reviews, 30% of all known problems were discovered through partition verification efforts.

Chip Verification

Chip level verification typically focuses on a single chip or subset of chips which is referred to as the device under test (DUT). The goal of chip verification is to verify that the DUT as a whole functions correctly. For the EXA chipset this would be the last level of pre-silicon verification.

The dynamic nature of the design was a challenge. The verification environment needed to be flexible and scalable to keep up with design changes from IBM and Intel®. The building block chip verification structure used took into account the similar structure of the EXA chipset that needed verification which saved on the number of environments that needed to be created.

Changes to Chip Verification Methodology

Several major changes were undertaken to meet the challenges presented by the EXA chipset. The first big change was to move exclusively to a cycle simulation engine. Cycle simulation was adopted over event simulation due to its ability to take on larger models and provide greater performance. The cycle simulator used was IBM's own multi-value simulator (MvlSim).

With the change to cycle simulation came the change to coding the environments in C++. Before this point the verification environments had been coded in sequential VHDL. Object oriented code was a natural fit for the scalable and modular design that was needed. To jump start this effort, an internally developed set of C++ base code, called USimBase, was picked up. USimBase provided the following benefits:

- A building block methodology that divided the environments into reusable parts that could have multiple instantiations. Strict rules for building block creation and use kept dependencies to a minimum so that each build block could be plugged in and out with ease.
- The basics for a test case parser which allowed a single text file to be contained in the test case. Syntax checking and test case command routing were built into this code.
- An optimized verification environment interaction with MvlSim to minimize DUT evaluations.
- Various utility functions.

While USimBase provided much of what was needed, it lacked the ability to synchronize and coordinate drive stimulus. To provide the level of control required to verify the EXA chipset something else was needed. The solution was to design a set of base classes over USimBase to create the verification framework needed. This verification framework was called Touchstone and provided the following capabilities.

Transaction Manager

Transaction packets are the basic communication unit in Touchstone. Various classes of transactions exist for control and configuration, instructions to drive an interface and instructions to check an interface. This communication was managed in Touchstone by a transaction manager. Almost all test case commands resulted in one or more transaction packets getting created.

Test Case Monitor

Touchstone provided a test case monitor that allowed the execution of the test case to be monitored and controlled centrally. It controlled test case delays, synchronization, and test case flow. It provided the foundation to create test case command and transaction packet interactions. Two independent operations could be forced to interact, overlap or collide with the control provided by the test case monitor. This was critical for writing complex test cases such as those needed by the CSC32 and CSC64 chips in a four node environment.

Programmable Clock Oscillators

Special clock building blocks were provided by Touchstone for clock oscillators. These oscillators drove both the clock tree in the DUT and were used to clock the various building blocks.

Exception Handling

Touchstone provided a central controller for all building blocks through which to feed exceptions. The central controller could then determine if this error was to be ignored and would need to halt the simulation. This was important for bad machine path testing when exceptions need to be ignored.

The Building Blocks

The basic building blocks included Touchstone, environment behaviors (EBs) and physical interface behaviors (PIBs). PIBs were further broken down into unit drivers (UD) and interface monitors (IM). These basic blocks were used to build the various environments required for EXA chip verification. Figure 3 shows a generic example, including Touchstone, which was previously explained. It also shows the flow of information through the various communication channels that exist in an environment.

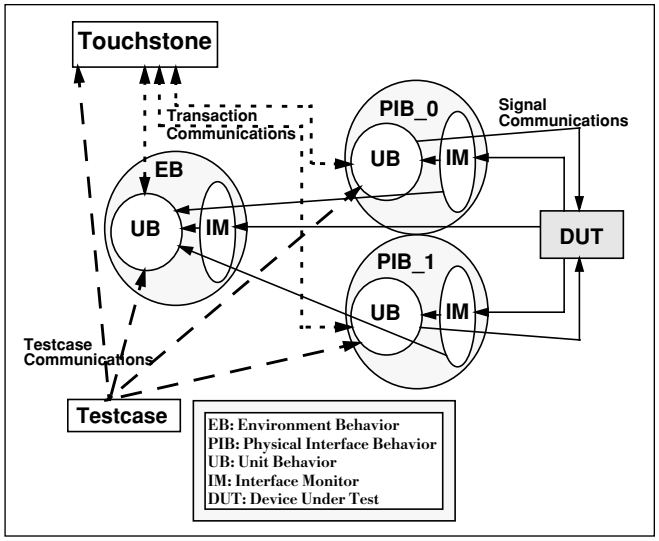


Figure 3 Generic Environment showing building blocks.

The PIB

Each external interface is given a PIB. The PIB is comprised of two blocks, UB and IM, for flexibility and reuse. The IM is used to gather information from the interface. It will never stimulate the interface. The IM is also responsible for performing all interface protocol checks. The UD is the block that stimulates the interface. This can be done directly through the test case or through transaction packets from an EB. The UD also performs checks that are not considered protocol checks. An example is data returned on a read initiated by the UD. All check data is contained in the test case command or communicated from the EB in a transaction packet.

The EB

The environment behavior (EB) coordinates PIBs in the environment to properly drive and check all interactions during a test. Since the EB has a view of the entire system contained in the DUT, testing can be abstracted to a higher level. Complex commands involving multiple interfaces can be coded up as a single command. The EB breaks these commands into multiple transaction packets that the PIBs must drive and check. Touchstone allows these transaction packets to be properly routed and

released to the various PIBs as appropriate. Touchstone allows the EB to place very complex dependencies on the transactions to create the needed interaction necessary to complete a given test case.

The Test case

The final piece is the test case. This text file can be either manually or randomly generated. It contains the commands that will be used to accomplish a given test. These commands are either targeted for the EB and/or a PIB. Each environment used to verify the EXA chipset had several thousand test cases that were run against it. Several tools were created to randomly generate test cases. Combined with continuous submission tools, test cases could be run around the clock, keeping the simulation machine farm continuously busy.

EXA PIBs

The EXA verification effort had nine interfaces requiring PIBs. The nine interfaces were the Intel Xeon™ Processor MP Family system bus (Xeon SB), Intel Itanium™ Processor Family system bus (Itanium SB), QuadT bus, I2C bus, PCI/PCI-X bus, RXE Expansion Port, JTAG bus, SDRAM interface (Memory), and the XceL4 Cache interface. Each PIB took two to three months to initially develop and test independent of the environment in which they would be used. Independent testing was an important fact since it eliminated a majority of the bugs in a PIB, allowing more focus on EB and DUT bugs.

The SB PIBs, which included the Intel Xeon™ Processor MP and Intel Itanium™ Processor Family system bus, required additional innovation to make up for the fact that the processors they represented would not be part of pre-silicon verification. Normally, a PIB is developed to be independent of the chip to which its interface is attached, allowing the PIB to be reused on any chip having that interface. This model could not be followed thus, some of the specific function of these two processors needed to be placed into the PIBs.

A flexible solution was needed that could quickly change as new information was learned about the processors. One solution was to take advantage of the coherence tables (see Table 1) that were done during partition verification of the coherency unit. These tables were used in conjunction with the L3 cache that was modeled in each PIB for coherency purposes. Since these details were not explicitly written into the C++ code, a quick table update was all that was required when changes were needed.

Another time saver was the fact that two SBs were very similar. This allowed sharing base classes that incorporated the common behavior of each PIB. Keeping common function in common files meant making only one update when common functions changed.

These techniques helped, but there were still significant differences and details that could not be abstracted out. In the end, significant work was required to keep these two PIBs working with the latest knowledge of the processors.

EXA EBs

The EXA verification effort required the creation of four EBs. These four EBs could handle more than four DUTs. Simply stated, the four EBs were as follows:

- MIOC EB: Capable of handling any variant of a single MIOC chip.
- PCI-X IOB EB: Capable of handling any variant of a single IOB chip.
- MIOC/PCI-X IOB EB: Capable of handling a DUT containing both a single MIOC and two PCI-X IOB chips.
- CSC32/CSC64 EB: Capable of handling a DUT containing any variant of MIOC matched up with a CSC32 or CSC64. It could also scale up from one node to multiple nodes (see Figure 4).

The Environments

Numerous environments had to be configured and supported to test the various DUTs. Figure 4 shows four of the many possible configurations. It shows the four node, 16-way environment for verification of either the CSC32 or the CSC64 chips with MIOC chips. It also shows what a single node case would look like if the other three nodes were removed for both CSC32 & CSC64. A single CSC32/CSC64 EB can handle all the environment configurations for this case. Likewise, the other EBs can handle numerous environment configurations. This is just one example of how the environments building block structure was designed to encompass the flexible, scalable, and modular nature of the EXA chipset.

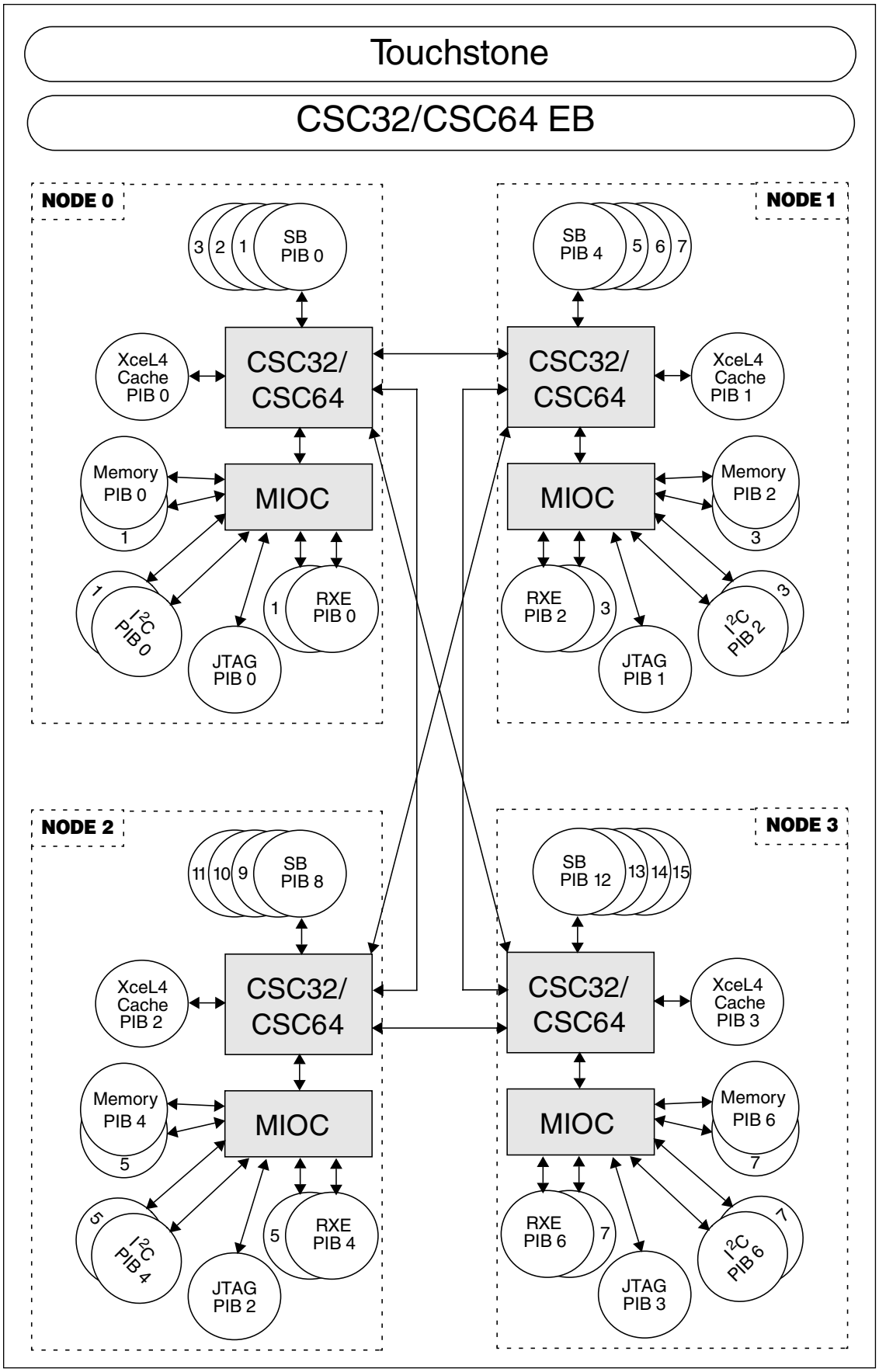


Figure 4 CSC32/CSC64 Chip Verification Environment.

Chip Verification Results

Chip verification constituted the bulk of the pre-silicon effort for EXA. Over 60% of all known problems were discovered during this phase of testing. Combined with the other pre-silicon efforts, over 91% of all known problems were removed from the system. The fact that first pass hardware booted up in the laboratory and made significant progress shows that the pre-silicon effort was a success.

Post-silicon Validation

The EXA chipset represents a new foray of IBM technology into the Intel-based server arena. One of the key challenges in this arena was to improve system reliability beyond industry expectations. EXA chipset based platforms take advantage of IBM @server validation technologies to ensure a robust and resilient platform. Many of these validation tools and techniques have been employed within IBM to produce many of the most reliable platforms in the industry.

IBM has invested a great deal in custom test environments to provide the most rugged and thorough validation environment possible. IBM also invests heavily in multi-OS testing and fully tests every supported operating system to ensure compatibility and robustness.

Development Tools and Exercisers

Many different development tools were utilized in the validation of the EXA chipset. Many of these tools were developed within IBM specifically for EXA-based platforms.

Grub

Grub is a proprietary test case environment used to test system cache coherency and multi-processor behavior. Grub is run very early in the validation cycle to flush out any potential coherency or multi-processor related issues. Grub is a low-level environment and is used for testing both 32-bit and 64-bit processor based systems.

NK32/64

NK32/64 is another proprietary test case environment that is capable of testing a wide variety of system functions including cache, spin locks, coherency and memory, as well as I/O, such as RXE link, PCI and PCI-X. NK32/64 stands for Nano-Kernel 32-bit (or 64-bit) and is based on a custom Minix-based operating system. NK32/64's operating system is completely customizable to create targeted test cases to stress all aspects of EXA-based systems.

AE2000 Memory, Disk, Cache, And Lan Exercisers

AE2000 is an IBM proprietary test case environment that runs under Microsoft® Windows 2000. AE2000 includes OS based exercisers that thoroughly test memory, disk, cache and LAN functions. AE2000 allows for platform stability testing and is used by the system development and classical test teams to stress the system in areas such as thermal, EMC, and power.

PCI/PCI-X Exercisers

IBM has a comprehensive test case environment for validating PCI/PCI-X functions. This environment utilizes advanced PCI/PCI-X adapters and test cases specifically developed to test all functions. These test PCI/PCI-X corner cases as well as stress the PCI/PCI-X bus. The test cases are also utilized to test error handling and recovery functions within the system.

Cronus

Cronus is at the heart of the validation tool suite for EXA-based platforms. Cronus is a powerful low-level validation environment that is capable of controlling and monitoring every bit within the EXA chipset. Cronus takes advantage of the Joint Test Access Group (JTAG) ports on the EXA chipset. Cronus enabled the following capabilities during the validation of the EXA chipset:

- Scripting to create powerful test cases and exercisers. Test cases were written to validate the functional behavior of the chipset and exercisers were written to stress each of the interfaces.
- Tuning of processor, memory and I/O interface timings
- Debug trace arrays which allow for storing of, and triggering on, internal logic conditions
- Clock and PLL tuning

Test Environments

Building Block Functional Verification

The first test phase for EXA-based platforms is the Building Block Functional Verification (BBFV). This testing is done on initial prototypes of the system when development engineers thoroughly evaluate the design and make improvements to ensure a stable platform for the test phases that follow. During this phase of testing all functions of the design are verified and exercised. Ensuring the quality of the design is paramount in this phase of testing.

The following are some of the tests that are performed on EXA-based platforms during this phase of test.

— *Custom Interface Tuning To Achieve Maximum System Design Margins*

Extensive tuning is done on all electrical interfaces of the EXA chipset to provide a level of design margin required in demanding environments. Tuning involves I/O driver analysis to determine the optimum I/O driver type for each electrical interface. Tuning also involves setting receiver input voltage references to optimum values which are unique for each interface such as processor, memory and I/O.

— *Full Parametric Sweeps – Voltage, Temperature, Frequency*

EXA chipset-based platforms are fully characterized using full parametric sweeps to ensure that they are the most robust platforms in the industry. Parametric sweeps include all AC/DC voltages, temperature, frequency and voltage references.

— *Signal Characterization*

Part of the design validation activities for EXA-based platforms includes full signal analysis and measurement to evaluate and confirm specification adherence for all signal parameters including quality, timings and noise.

— *Printed Circuit Board Quality Characterization*

Printed circuit boards are evaluated for adherence to IBM specifications (e.g. impedance, cross talk, dielectric), and measurements are made of all interfaces using Time Domain Reflectometry (TDR) to ensure design compliance.

System Design Verification and Integration Testing

System Design Verification (SDV) is the first phase of formal testing for xSeries servers. SDV testing is done on pre-production level hardware and firmware and is used to flush out problems prior to production level hardware and firmware.

System Integration Testing (SIT) is the production level hardware and firmware formal test phase.

The following summarizes some of the tests performed in SDV and SIT.

Integrated Functional Testing

Tests in this environment include network loading and services, IP services, advanced system management services, server subsystem stress, file services and data integrity tests.

Platform Compliance Testing

Platform compliance testing includes guard band, thermal, electromagnetic compatibility, susceptibility, electrostatic discharge immunity, acoustics and vibration.

Configuration Testing

Platforms utilizing EXA chipset are subjected to vast arrays of differing configurations to verify multi-vendor compatibility as well as validate processor, memory and PCI-X plug ordering.

Maintenance Testing

Extensive testing is done to verify error handling and notification so that the failing system component can be serviced.

Network Operating System Testing

This test certifies each supported operating system on the platform and verifies system robustness under each operating system.

Automated Tests

Automated AC and DC power cycling as well as reboot tests are run for weeks.

Conclusion

The various levels of pre-silicon verification helped minimize the number of bugs found during post-silicon validation, decreasing development cost and accelerating time to market. Post-silicon validation ensured error free operation of the chipset and resulted in robust and reliable systems with maximum margins.

xSeries 360, IA-32 Basic Node

– Dan Hurlimann, Yiming Ku, Harry Schultze, Tommy Tam, Cindy Walter, Lee Wilson

Introduction

The xSeries 360 is a fault tolerant price performance scalable server with high availability attributes that meets the requirements of enterprise servers running mission critical applications. The xSeries 360 is the first of many systems to use recently announced Enterprise X-Architecture™ [1] and the IBM XA-32™ chipset, bringing mainframe technology, features and functions to the Intel-based server environment.

The xSeries 360 supports up to four Intel Xeon™ Processor MPs in a Symmetric Multiprocessor (SMP) configuration. It also supports up to 8GB (using 512Mb memory technology) of industry standard PC1600 ECC Double Data Rate (DDR) system memory. The system incorporates six full length 64-bit ActivePCI-X slots, one of which can support an ultra fast 133MHz adapter. The xSeries 360 is packaged in a dense 3-EIA (total height of 5.25") chassis, ideal for optimizing rack utilization.

Product Description

At the center of each xSeries 360 is the IBM XA-32 chip-set, consisting of the Memory/IO Controller (MIOC)[2] and the PCI-X I/O Bridge (PCI-X IOB)[3] chips. The MIOC communicates with the processors via the system bus, the memory via the memory bus and the PCI-X I/O Bridge chip via the Remote Expansion Port. The PCI-X I/O Bridge converts the Remote Expansion Link to three PCI-X busses.

The memory used in the system is the industry standard PC1600 ECC DDR memory modules in 256MB, 512MB and 1GB sizes for a maximum system memory of 8GB.

The I/O subsystem of the xSeries 360 has six full length industry standard 3.3 volt 64-bit PCI-X slots. ActivePCI-X is the Enterprise X-Architecture extension to IBM's ActivePCI technology, bringing hot swap, hot add and fail over capabilities to PCI-X. One bus has two full size PCI-X slots that run up to 100MHz. If one slot on this bus is not populated, then the bus can run up to 133MHz. The second bus has four full size PCI-X slots that can run up to 66MHz.

Integrated functions in the system are attached to a 64-bit, 33MHz PCI bus. They consist of a single channel Ultra160 SCSI controller that provides up to three hot swap disk drives, a video controller and a 10/100 Ethernet chip. Three Universal Serial Bus (USB) ports (one in front and two exiting from the rear of the system) allow native attachment of low speed I/O devices. The xSeries 360 also provides one slim 24x-10x CD-ROM drive and one slim 1.44 MB diskette drive.

The xSeries 360 transitions to a reduced legacy I/O system. In addition to USB ports, the system includes legacy keyboard and mouse ports. An internal serial port connector is available for software debug.

This server provides the capability to optionally attach one RXE-100 Remote Expansion Enclosure[4]. The RXE-100 provides an additional 12 PCI-X slots, increasing the total number of PCI-X slots to 18, thus greatly increasing the I/O capabilities of the system.

The xSeries 360 is packaged in a 3-EIA (5.25") by 28" deep drawer in a 19" rack. It is powered by three 370 watt hot swap power supplies. Two power supplies are required for a fully loaded system, the third providing redundancy.

Reliability, Availability and Serviceability (RAS) are key features of the xSeries 360. These are provided through ServerGuide, IBM Director, Remote Supervisor Adapter (RSA), Drawer Management Controller (DMC), Lightpath, and hot swap redundant cooling fans.

Several operating systems are supported by the xSeries 360 including, Windows® 2000 Advanced Server and Server, Windows NT®4.0 Enterprise Edition, Novell Netware®, Linux (RedHat, Caldera, SuSE) and SCO Unixware®(trademarks, registered and content).

System Architecture

The xSeries 360 system diagram is shown in Figure 1.

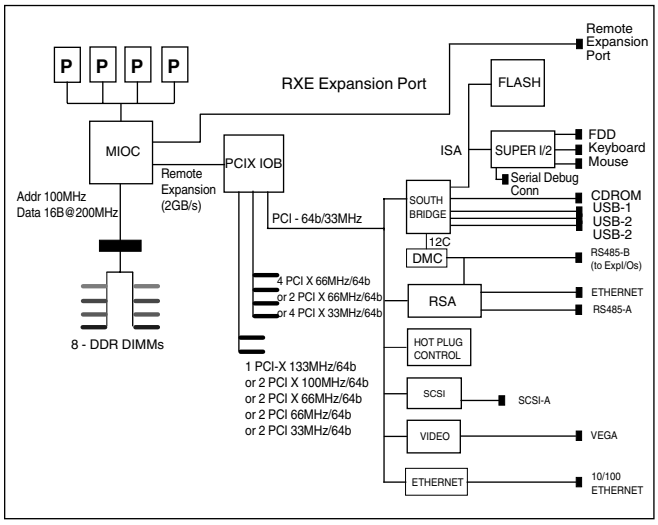


Figure 1 xSeries 360 Block Diagram

The Memory/IO Controller (MIOC) has three interfaces; the system bus, the memory bus and two RXE Expansion Ports. The MIOC communicates with the processors via the system bus, with the memory via the memory bus and with the PCI-X I/O Bridge (PCI-X IOB) via the Remote Expansion Port.

The MIOC chip has full SMP support for up to four Intel® Xeon™ Processor MPs. The system bus control signals run at 100MHz, the address signals at 200MHz and data signals at 400MHz. The address bus is 36 bits, double pumped source synchronous with a maximum transaction rate of one every two 100MHz bus clocks. The data bus is 64-bit quad pumped source synchronous, providing 3.2GB/s peak bandwidth. Both address and data busses incorporate parity to insure data integrity.

The MIOC chip has a 16-byte memory interface that provides 3.2GB/s bandwidth, matching the system bus. The control signals to the direct attached DDR memory modules operate at 100MHz and the data bus to and from the memory modules at 200MHz. This interface supports eight industry standard DDR memory modules. These modules are populated in pairs of like sizes. The maximum memory capacity is 8GB using 1GB memory modules. The MIOC provides Chipkill Error Correcting Codes (ECC), allowing system operation to continue even when there is a multi-bit failure in a single memory chip.

The MIOC chip communicates with the PCI-X I/O Bridge chip and the external RXE-100 Remote Expansion Enclosure via the two Remote Expansion Ports. These ports are bidirectional, differential, two byte wide links that operate up to 500MHz resulting in up to 2GB/s peak bandwidth.

The PCI-X IOB chip is attached to the MIOC via one of the Remote Expansion Port. The IOB generates three independent 3.3 volt, 64-bit PCI-X bus segments. It also implements an I/O xAPIC interrupt controller. Peer to peer data transfers are supported between devices connected to the same PCI-X bus segment.

One of the three busses is dedicated to native I/O devices and the Remote Supervisor Adapter. The other two 64-bit PCI/PCI-X busses are exclusively for PCI or PCI-X feature cards.

The integrated native I/O devices are accessed through the PCI-X IOB PCI bus 0 which runs at 33MHz. Bus 0 supports ISA bus operations like the special PCI bus arbitration signals PHOLD, PHOLDA for ISA DMA operations and PCI interrupt acknowledge transactions for PIC mode interrupt operations.

A South Bridge chip attached to PCI bus 0 generates an ISA bus, one IDE bus and four USB ports. The ISA bus devices include a 4MB flash memory for POST/BIOS and Diagnostics code and a Super I/O chip. The Super I/O chip converts the ISA bus to a floppy drive bus, a mouse port, a keyboard port and also a serial port for internal use. The IDE bus supports both Programmed I/O (PIO) and Ultra DMA (33MHz) mode for the slim CD-ROM drive. Three of the four USB ports are available in the system.

Three other PCI bus 0 devices are integrated on the system planar. A SCSI controller with 64-bit PCI data bus supports one Ultra160 SCSI bus allowing the attachment of three internal hot-swap disk drives. A video controller converts the PCI bus to the SVGA CRT port signals. An ethernet controller supports 10BASE-T/100BASE-TX.

The two busses dedicated to feature cards are capable of running in either PCI or PCI-X mode. In the past, reconfiguring bus speed and mode required rebooting the system. In the xSeries 360, these two busses are dynamically reconfigurable for both speed and PCI/PCI-X mode operation, improving the capabilities of hot plug operations. With these new capabilities, the xSeries 360 begins a new generation of systems with enhanced I/O performance and new ability to dynamically reconfigure PCI/PCI-X busses.

The system configures the I/O slots to make maximum possible use of the installed feature cards. Since PCI-X feature cards are also downward compatible to PCI, it is not necessary to separate PCI-X and PCI feature cards onto separate busses. Configuration of each of the two busses is the lowest common denominator for all of the feature cards installed on that bus. Care should be taken when installing feature cards into slots so that the performance of each bus is maximized.

The MIOC has a second Remote Expansion Port, which is routed to a connector on the rear bulkhead of the system. An RXE-100 Remote Expansion Enclosure may be connected to this port, increasing system I/O capability by adding up to two additional PCI-X IOB chips, six additional PCI/PCI-X bus segments and 12 additional PCI/PCI-X feature card slots. The total of 18 available feature card slots far exceeds the capabilities of competitive systems.

Performance

System performance is established by the processor, the processor bus bandwidth, the total bandwidth to memory and I/O, as well as latency in the system. A major focus was placed on these performance factors.

The goal of the IBM XA-32 chipset was to match the processor requirements such that the processor would not wait for the system. The Intel® Xeon™ Processor MP processor bus has an effective bandwidth of 3.2GB/s. The interface from the MIOC to the DDR memory has a data bandwidth of 3.2GB/s. Each Remote Expansion Port from the MIOC has a bandwidth of up to 2GB/s for a total available I/O bandwidth into the MIOC of up to 4GB/s. Thus, from a system view, the bandwidth from the processor to memory is matched and the I/O bandwidth is sufficient to keep the processor data bus saturated.

Latency to memory is the other significant factor in system performance. Latency in the MIOC and the system was modeled and simulated for the higher end systems in the family of Enterprise X-Architecture systems. This work factored into the xSeries 360 system to make the overall system performance on initial offering better than any of the previous systems in its class

The electrical characteristics of the memory bus[5], PCI-X busses[6] and the RXE Expansion port[7] were a major focus. Each bus was modeled with several different tools. HSPICE models were used as much as possible. Within the custom tools, detailed circuit models were used to do in depth analysis of the busses. Without these tools, it would have been difficult to achieve the performance without adding significant cost to the raw boards or adding voltage levels to the system. Rules were generated from this modeling to be used during board layout.

Correlation between the model results and the initial hardware was done when the hardware became available. The models were updated and the results compared with the processor manufacturer for the processor bus. All the results were fed into the final pass of the board design, yielding a system performance that is not gated by the board or overall system implementation.

Scalability

One of the major attributes in the Enterprise X-Architecture technology is the ability to increase system capacity with XpandOnDemand scalability. The xSeries 360 is the first Intel Architecture system to provide optional, external I/O scalability. I/O scaling is provided through the Remote Expansion Port to an RXE-100 Remote Expansion Enclosure. The RXE-100 comes with six PCI-X slots and has the ability to grow to 12 slots in the same 3 EIA rack drawer. This provides each xSeries 360 with the ability to scale to a maximum of 18 PCI-X feature cards installed in the system.

POST/BIOS

POST/BIOS is stored in a programmable flash module in the system. It is executed once the system is powered up. POST/BIOS provides the following key features.

The first key function is to test and configure the components of the system. The processor subsystem, memory subsystem, host I/O subsystem and optional RXE-100 Remote Expansion Enclosure are optimally configured, and the configuration information is made available for use by the operating system or other software.

POST/BIOS also provides industry standard BIOS run time interfaces to operating systems and applications. Standard interfaces supported on the xSeries 360 include the standard INTx BIOS calls, SMBIOS 2.3, ACPI BIOS 1.0b, and MP 1.4 BIOS.

Finally, POST/BIOS provides a boot time user setup interface. This allows users to view or customize certain system configuration parameters.

POST/BIOS can be easily updated locally by a flash update diskette or remotely through LAN or Internet.

Reliability/Availability/Serviceability

At a system level, xSeries 360 handles both recoverable and unrecoverable errors. Recoverable errors are detected and handled by the System Management Interrupt (SMI) handler. These errors are single bit memory errors and other recoverable errors detected by the MIOC and the PCI-X IOB.

The IBM xSeries 360 utilizes the extensive RAS features of the XA-32 chipset to deliver a highly reliable and easily serviceable system. Traditionally the SMI handler dealt with all unrecoverable errors. In the xSeries 360, unrecoverable MIOC detected errors are detected and the error state logged out by the Remote Supervisor Adapter. This allows accurate error logging even for cases where the SMI handler is unable to run. Recoverable MIOC detected errors are reported to and handled by the SMI handler.

MIOC error handling includes Chipkill ECC on system memory. Single bit memory error and multiple bit errors where all failing bits are from a single memory chip are corrected on the fly and reported as recoverable errors. Multiple bit errors where the failing bits are not from a single memory device are detected and reported as unrecoverable errors. In order to further improve the recoverability of system memory errors, the MIOC also supports hardware memory scrubbing. The scrubbing hardware in the MIOC continually accesses system memory locations. When a correctable soft error is detected, the location is rewritten to correct the error. This prevents the buildup of correctable errors and reduces the probability for unrecoverable errors. Chipkill ECC along with scrubbing significantly improve the reliability of the memory subsystem, reducing the instances of unrecoverable system errors due to the memory.

The MIOC also detects and reports errors on the processor system bus.

Error handling in the PCI-X IOB chip is built on enhancements from previous generations of IBM Remote Expansion Port Bridge chips and IBM PCI Bridge chips used in IBM iSeries and pSeries systems. RXE Expansion Ports are protected using a Cyclic Redundancy Check (CRC) mechanism to identify errors in data movement between the MIOC and PCI-X IOB.

Both recoverable and unrecoverable errors detected by the PCI-X IOB chip generate a System Management Interrupt and are logged by the SMI handler. For unrecoverable errors, including PCI bus parity error (PERR) or PCI bus system error (SERR), the SMI handler will generate a non maskable interrupt (NMI) after logging the error to the Remote Supervisor Adapter. The error log can then be reviewed to pinpoint the failing component.

Another feature of the xSeries 360 is that if an Intel® Xeon™ Processor MP detects an internal error (IERR), system logic will detect this condition, disable the failing processor and reboot the system, permitting continued operation in degraded mode.

Several major components in the xSeries 360 are hot pluggable, allowing their removal/addition without the need to halt system operation. These components include power supplies, cooling fans, disk drives and PCI feature cards.

Power/Packaging

Packaging the xSeries 360 was challenging due to the requirements of supporting up to four Intel Xeon Processor MP microprocessors, a memory card with eight DIMM slots, six full length PCI-X feature cards, a Remote Supervisor adapter (service processor), on board functions (SCSI, Video, Ethernet, PCI Hot Plug Control, reduced IO legacy and flash EEPROM) in a 3-EIA (5.25") by 28" rack drawer. The majority of this function is placed on either the system motherboard or the memory card with a minimal amount of function on ancillary cards.

The system motherboard (Figure 2) contains the vast majority of the system components, including the XA-32 MIOC and PCI-X IOB chips. Up to four Intel Xeon Processor MPs may be installed in the processor sockets located in the front right quadrant of the motherboard. The memory card connector is oriented front to rear at the middle of the motherboard. The six 64-bit PCI-X feature card slots are located at the left rear of the motherboard, along with the connector for the Remote Supervisor Adapter. Integrated I/O components are populated across the remainder of the motherboard, with the integrated I/O connectors exiting at the right rear.

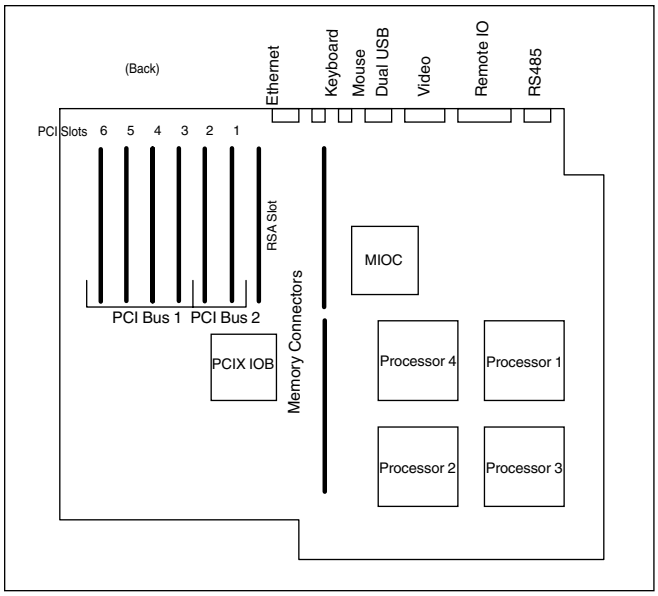


Figure 2 System Motherboard

The memory card (Figure 3) contains eight sockets for the PC1600 DDR Memory Modules.

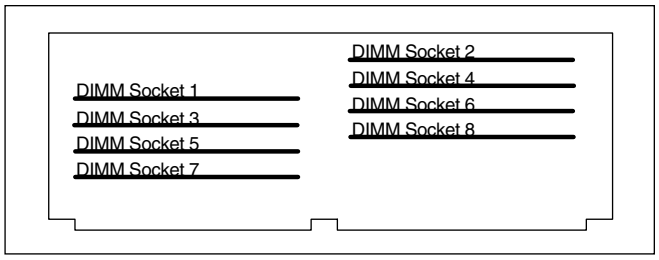


Figure 3 Memory Card

The remaining cards are the Power Backplane, SCSI Backplane, Operator Panel card, Level 2 Lightpath card, Media Interposer card and PCI Hot Plug Switch card.

The mechanical package for the xSeries 360, shown in Figure 4, is an industry standard 19" rack drawer, 3-EIA units (5.25") high by 28" deep.

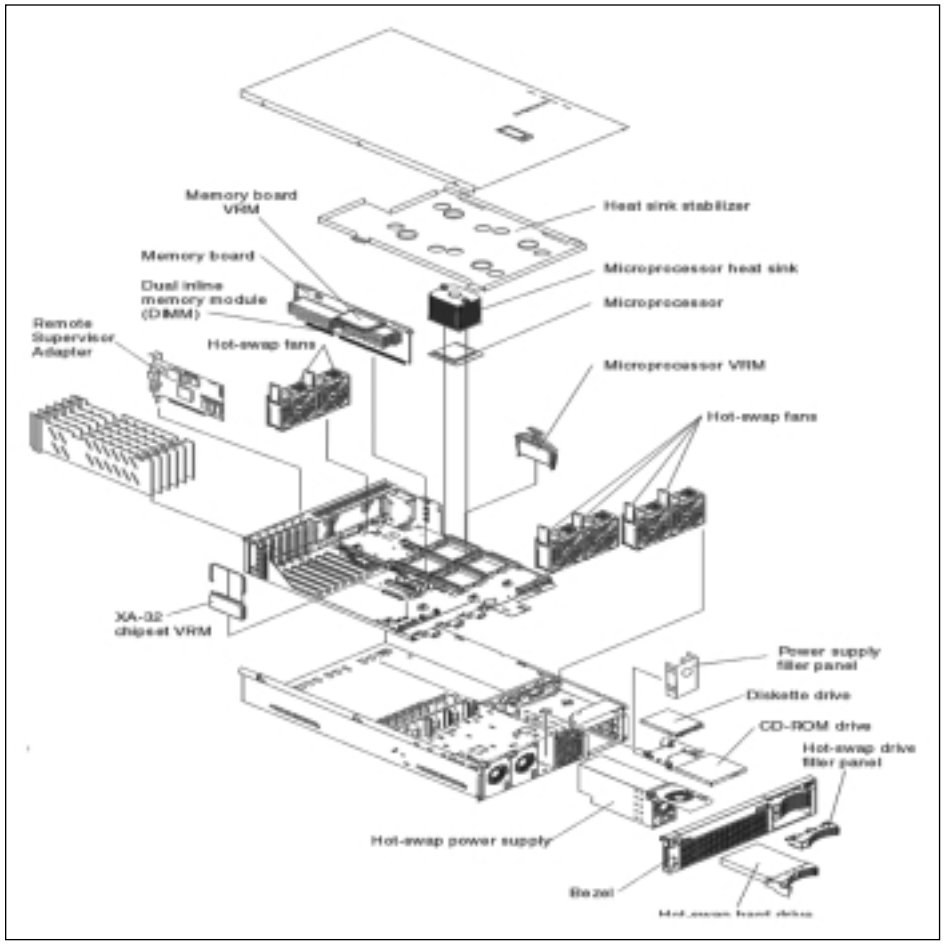


Figure 4 Mechanical Package

System cooling consists of six redundant and hot swappable fans. The processor area is cooled by two mid system fans and two rear fans. The memory and I/O area is cooled by two mid system fans. Fan redundancy was designed so that if any one of the six fans fail, the remaining five will operate at full speed to maintain the proper cooling.

The xSeries 360 includes three hot swap 3.5" bays for the up to three hot swap hardfiles. A hot swap backplane is part of the drive bay assembly. This backplane uses the industry standard 68-pin Single Connector Attach (SCA) connector to attach to the Ultra Fast and Wide SCSI hard drives.

The system is supplied by grounded, single-phase (3-wire), five output level power supplies with auto sensing input voltage ability. The power supplies can provide up to 370 watts each at 110V / 220V. Systems are shipped with one to three power supplies. Redundant power supply operation is supported with N+1=3 configurations. A fully loaded system requires two power supplies to operate, with the third one being redundant. Hot plug is supported by allowing a power supply to be inserted or removed without affecting system function. Each power supply provides 5V@20A, 12V@27A and -12V@ 0.6A. The power supply also provides two standby voltages, 5VSB@2A and 3.3VSB@4A, which are available whenever the power supply has AC power. Combined power for all five outputs is limited to 370 watts.

In order to provide distributed power throughout the system, DC to DC converters (VRMs) are used to provide the on demand current when needed. A VRM 9.0 is required to supply 1.5V core voltage to each Intel Xeon Processor MP. A VRM40 is used to provide the 1.8V core voltage for the XA-32 chipset. Another VRM40 provides the 2.5V power required by the DDR memory modules. Each of these VRMs is powered by the 12V level from the power supply. Finally, two 3.3V VRMs are used to provide PCI-X slot power. These VRMs are supplies from the 5V power supply level.

Conclusion

The xSeries 360 use of IBM's XA-32 chipset along with the Intel Xeon Processor MP gives the user the latest generation of price for performance PC Server. The compactness allows the most performance available in the smallest amount of space, optimizing the rack usage.

Acknowledgments

Numerous people contributed to the success of the xSeries 360. We would like to specifically acknowledge the system architect, Sudhir Dhawan, for his insight and tireless effort.

References

1. J. D. Brown, S. Dhawan, *Enterprise X-Architecture Technology Overview*, IBM Enterprise X-Architecture Technology, p.1
2. L. Blackmon, B. Drehmel, T. Greenfield, J. Kirscht, J. Marcella, D. Shedivy, *EXA Memory/IO Controller*, IBM Enterprise X-Architecture Technology, p.9
3. T. Moe, C. Paynton, R. Shearer, S. Willenborg, C. Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31
4. J. Haidinyak, N. Sherali, *xSeries RXE-100 Remote Expansion Enclosure*, IBM Enterprise X-Architecture Technology, p.93
5. M. Cases, D.N. De Araujo, D. Guertin, N. Pham, *EXA Source Synchronous Memory Design*, IBM Enterprise X-Architecture Technology, p.105
6. M. Cases, D.N. De Araujo, N. Pham, *EXA PCI-X Subsystem Design*, IBM Enterprise X-Architecture Technology, p.113
7. M. Cases, D.N. De Araujo, N. Pham, *EXA Simultaneous Bidirectional Interface Design*, IBM Enterprise X-Architecture Technology, p.99

xSeries 440, IA-32 Enhanced Node

– Maurice Bland, Randy Kolvick

Introduction

The xSeries 440 is IBM's premier IA-32 Symmetric Multiprocessor (SMP) system based on Enterprise X-Architecture™ (EXA) technology[1]. EXA scalability, performance and Reliability, Availability and Serviceability (RAS) features, along with unmatched package density, make the x440 a unique industry standard server.

Product Description

The xSeries 440 comes standard with one SMP Expansion Module containing one or two of Intel's Xeon™ processors MP and 1GB of system memory. Four total processors may be installed in the standard SMP Expansion Module. The SMP Expansion Module also provides 16 DIMM slots. 256MB, 512MB and 1GB SDRAM DIMMs may be installed for a maximum of 16GB system memory using 512Mb memory technology.

For larger configurations, a second SMP Expansion Module may be installed, doubling capacity to eight Intel Xeon processors MP and 32GB of system memory in the 4-EIA (1-EIA=1.75 inches) unit enclosure.

I/O is provided by six full length ActivePCI-X slots. Integrated I/O includes a dual Ultra160 SCSI controller and a Gigabit Ethernet controller. Traditional legacy ports such as keyboard and mouse are provided, along with three Universal Serial Bus (USB) ports, to allow attachment of either legacy or new I/O devices.

The 4-EIA unit enclosure supports two Hot Swap Hard Drive Bays along with two ThinkPad Media Device bays. Redundant power and cooling are standard.

XpandOnDemand™ scalability is a unique capability of EXA implemented in the xSeries 440. It is the capability to add additional system or I/O enclosures to increase capacity as needed instead of having to purchase unused headroom within each system unit enclosure. Two 8-way or four 4-way xSeries 440 nodes may be cabled together within a rack for a maximum 16-way SMP configuration with up to 64GB of system memory. Interconnected systems support physical partitioning[2], allowing multiple operating system images to run on the interconnected system node enclosures if desired. As on the xSeries 360[3], XpandOnDemand also provides I/O scalability. One 3-EIA Unit RXE-100 Remote Expansion Enclosure[4] may be cabled to each enhanced node in either the same rack or an adjacent rack, providing up to 12 additional full length ActivePCI-X slots each. This provides a maximum capability of 72 full length PCI-X slots for a maximum system incorporating four 4-EIA Unit system node enclosures and four 3-EIA Unit RXE 100 enclosures.

The Remote Supervisor Adapter (RSA) is standard, providing advanced System Management capabilities with IBM Director software. IBM Director now includes a System Partition Manager to support XpandOnDemand multinode configurations. Real Time diagnostics, Lightpath diagnostics and traditional standalone diagnostics are also supported.

The xSeries 440 may be used with many standard operating systems, including Microsoft® Windows, multiple Linux distributions and Novell Netware.

Node Architecture & Performance

The xSeries 440 Enhanced Node system structure is shown in Figure 1. As mentioned, the processor and memory subsystem can consist of one or two SMP Expansion Modules which are then connected to a single I/O subsystem.

The processor subsystem is based on the IBM XA-32 Cache/Scalability Controller 32 (CSC32) chip[5]. This chip interconnects the processors, XceL4™ Server Accelerator Cache, SMP Expansion Ports for multinode interconnect and the IBM XA-32 Memory/IO Controller (MIOC) chip[6].

The processors are Intel's most advanced Xeon processors MP, including Hyper Threading technology. They are attached to the CSC32 with an enhanced system bus. The system bus meets all Intel architectural and electrical requirements for attachment of up to four Xeon processors MP. Address and Control signals operate in common clock mode at 100MHz. The data signals are source synchronous quad pumped to 400MHz, providing a peak data transfer capacity of 3.2GB/s.

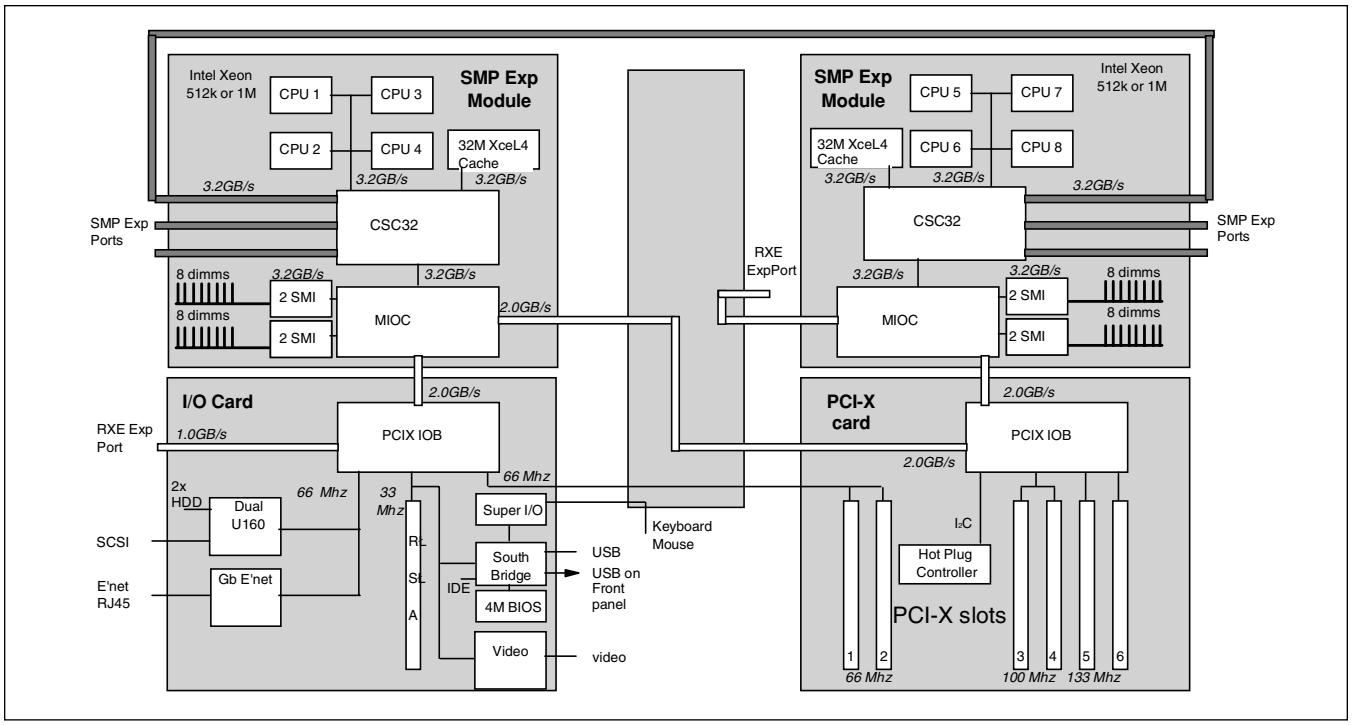


Figure 1 System Structure

The CSC32 chip provides a large 32MB XceL4 Server Accelerator Cache (L4 cache). This cache is 4-way set associative and inclusive of the individual processor caches. The cache directory is implemented internally to the CSC32, while the data array is implemented externally with 128Mb DDR SDRAM modules originally developed for high performance graphics applications. These chips are run at 200MHz for address/control with the data double pumped to 400MHz. The wide bus width graphics SDRAMs allow implementation of an 8-byte wide L4 cache data bus, matching the processor system bus bandwidth at 3.2GB/s.

The CSC32 also provides three SMP Expansion Ports. These allow the interconnection of up to two 8-way xSeries 440 nodes or four 4-way nodes. These ports are also used to connect together the two SMP Expansion Modules in a single node for 8-way configurations. Multiple xSeries 440 nodes located within a single rack may be interconnected using external cables. The CSC32 chip contains a directory that allows coherent operation of multiple nodes by tracking cache lines from local memory that are cached in another node. This directory is 7-way set associative. A non-cache coherent method of SMP Expansion Port data transfer, known as Inter-Process Communications (IPC), is also supported by the CSC32 to support multinode cluster configurations. SMP Expansion Ports are 2 byte wide, simultaneous bidirectional, double pumped, source synchronous links operating at a base frequency of up to 400MHz. This provides a total bandwidth of up to 3.2GB/s per port for a aggregate bandwidth of up to 9.6GB/s.

Finally, the CSC32 provides the path to the MIOC chip. It operates at a 100MHz base frequency quad pumped data, providing a bandwidth of 3.2GB/s.

The MIOC provides two independent memory ports to a PC133 SDRAM memory array running at 100MHz. Each memory port is eight bytes wide, operating at 400MHz. This provides a memory bandwidth of 3.2GB/s per port, or a total of 6.4GB/s. The Intel Xeon processors MP operate on a cache line of 64 bytes. Cache lines are usually interleaved between memory ports for maximum performance. However, if additional reliability is needed, the memory ports may be mirrored to provide redundancy. In this case, memory is read from one of the two memory ports and written to both of the memory ports simultaneously to keep memory data in both ports consistent.

Each memory port is connected to two System Memory Interface (SMI) chips. These are used to speed match and multiplex an aggregate 32-byte wide bus, encompassing four industry standard PC133 SDRAM memory modules operating at 100MHz, on to the 8-byte memory port running at 400MHz. This bus structure allows a maximum of 16 memory modules to be installed on each of the

two SMP Expansion modules, providing the large 16GB per node memory capacity using 512Mb memory technology. The minimum supported memory configuration is four memory modules attached to a single memory port.

Chipkill Error Correcting Codes (ECC) is included in the MIOC, enhancing system availability compared to standard memory ECC schemes. Chipkill is an enhanced ECC algorithm providing error correction not only for single bit errors, but also for clusters of 4-bit errors where the bits are all contained in a single memory chip. This capability allows for continued system operation in the presence of any single memory chip failure mechanism instead of only the bit line failure mechanisms of standard ECC algorithms. This advanced Chipkill design is used with industry standard PC133 memory modules; special 'Chipkill' memory modules are not needed.

EXA Active Memory technology implemented in the MIOC provides memory scrubbing, memory ProteXion, memory mirroring and memory HotSwap/HotAdd capabilities for the system.

Scrubbing hardware in the MIOC continually accesses memory locations. Upon discovery of a correctable error, the location is rewritten to correct the error and the error is logged for Predictive Failure Analysis (PFA). This prevents the buildup of correctable errors to the point where random distribution of correctable errors could result in an uncorrectable error.

Memory ProteXion is an EXA technology providing an additional level of protection from memory subsystem errors compared to ECC with Chipkill. Memory ProteXion provides redundant memory data paths with an ability to reroute memory data bits. When a significant number of correctable memory errors are detected, the failing bits are replaced by redundant bits, eliminating the errors. This greatly reduces the probability of encountering uncorrectable errors, improving overall system availability.

Memory mirroring is the complete redundancy of the memory arrays, providing the maximum level of system availability. It allows system operation to continue even when there are errors that are uncorrectable even with the enhanced ECC algorithms. This is done by maintaining a separate copy of the contents in memory attached to each of the two memory ports. When one of the memory ports starts to encounter an excessive number of correctable ECC errors, memory access are switched to the alternate port, allowing continued operation with a healthy memory array without needing to reboot the system.

Memory mirroring also provides the technical foundation for the Memory Hot Swap and Memory Hot Add features of Active Memory. Memory Hot Swap is the capability of replacing a failed memory DIMM while the system remains operational. When the system detects an excessive number of errors in mirrored mode, memory accesses are switched to the non-failing memory port and the failing memory module is identified with using Lightpath Diagnostics. Service personnel may then locate the failing memory module, disable and remove power to memory modules on the failing port, replace the failing memory module and re-enable mirroring without taking the system down.

Memory Hot Add is the capability to add memory to the memory subsystem without taking down the system. This feature requires an operating system capable of recognizing increased memory capability without requiring a reboot, and can be accomplished in one of two ways depending on whether mirroring is enabled. When mirrored mode is not enabled, Hot Add is supported if one of the two memory ports is completely empty of memory DIMMs. Memory DIMMs can be added to the empty port and then enabled for use by the operating system. While in mirrored mode, each memory port in turn is disabled and powered off then memory module(s) added and re-enabled. The final result is more memory in both ports of the mirrored memory which can then be reported to the operating system for its use.

The MIOC is also the connection point for the I/O subsystem. Each MIOC provides two RXE Expansion ports. These ports are routed to PCI-X I/O bridge (PCI-X IOB) chips[7] which can be attached locally (within the same enclosure) or remotely via external RXE Expansion Ports as shown in Figure 1. RXE Expansion Port technology is similar to SMP Expansion Port technology, but operates at a lower base frequency. The 250MHz base speed yields a bandwidth of 2GB/s per locally attached RXE Expansion Port segment for a maximum I/O bandwidth of up to 4GB/s into the memory subsystem of each SMP Expansion Module. Connection of the external RXE Expansion Port yields a half bandwidth of 1GB/s per remotely attached interface.

The I/O subsystem includes two PCI-X I/O bridge (PCI-X IOB) chips. Each PCI-X IOB has two RXE Expansion Port segments connected to MIOCs or external connectors as shown in Figure 1. The external RXE Expansion Ports are designed to attach to separate RXE-100 enclosures. Additional PCI-X IOB chips located in the separate

RXE-100 enclosures are connected to the RXE Expansion Ports to expand the I/O subsystem capability if needed. The RXE-100 may be physically located in a different rack than the xSeries 440 system node that it is connected to, providing a great deal of system configuration flexibility.

Each PCI-X IOB generates three PCI-X bus segments. As shown in Figure 1, two of these chips are used in the xSeries 440 node for a total of six 64-bit PCI-X bus segments. These six segments are distributed to provide two 133MHz ActivePCI-X slots, two 100MHz ActivePCI-X slots and two 66MHz ActivePCI-X slots, along with connections to Integrated I/O such as the SCSI controller and gigabit ethernet controller.

Scalability

The xSeries 440 provides the standard hardware scalability mechanisms for high performance servers, supporting up to eight processors, 32GB of memory, ServeRAID Adapters attached to external Storage Enclosures and PCI-X based Network Adapters. In addition to these standard forms of scalability, the xSeries 440 supports XpandOnDemand capabilities for both SMP and I/O scalability which are unique in the industry. For example, one 8-way or two 4-ways can be double connected for scalability port interleaving and cable fail over.

XpandOnDemand SMP scalability is the ability to interconnect multiple enhanced nodes through high speed SMP Expansion Ports. The xSeries 440 provides three SMP Expansion Ports, allowing up to two 8-way or four 4-way enhanced nodes to be connected together. These configurations enable scaling of up to 16 processors and 64GB of system memory in a single SMP system and are illustrated in Figures 2 and 3. Also, cable configurations shown here, using six cables for the attachment of the four SMP Expansion Modules, will provide cable fail over.

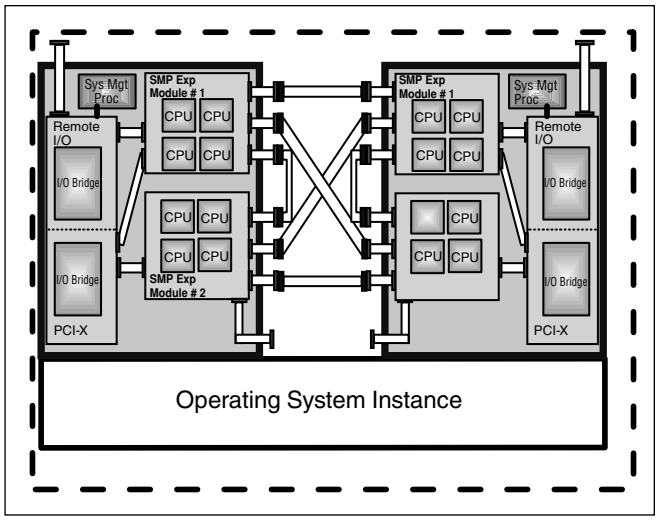


Figure 2 Two 8-way Nodes, Single SMP System

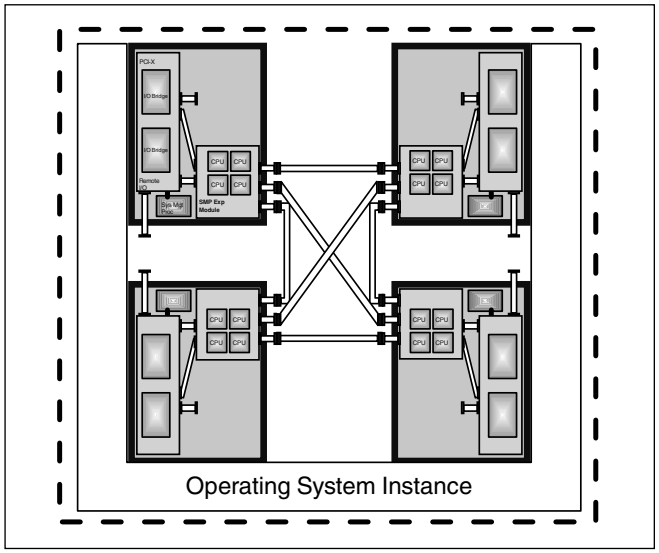


Figure 3 Four 4-way Nodes, Single SMP System

Communications over SMP Expansion Ports can be either cache coherent for SMP configurations or Inter-Process Communications (IPC) for clustered configurations. System partitioning provides the ability to run independent operating systems on different nodes of a server connected with SMP Expansion Ports. Partitions can consist of one, two, three or four nodes, allowing as many as four operating system images to be running concurrently. Systems can be repartitioned using the System Partition Manager, providing a great deal of flexibility to manage system resources. Examples of partitioned systems are shown in Figures 4 and 5.

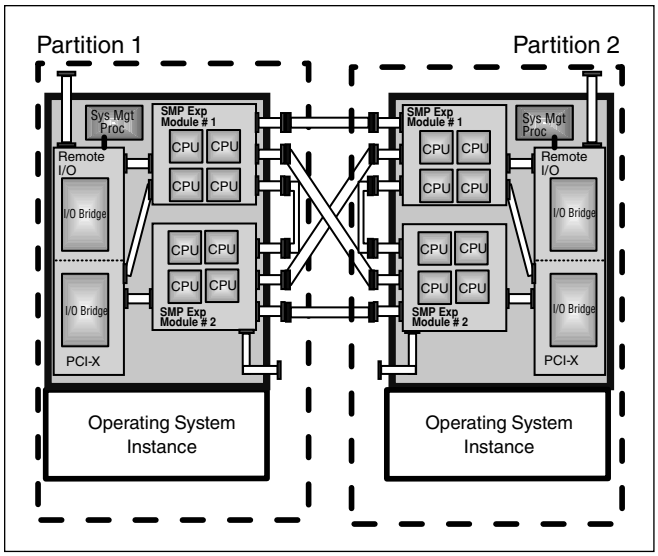


Figure 4 Two 8-way Partitions

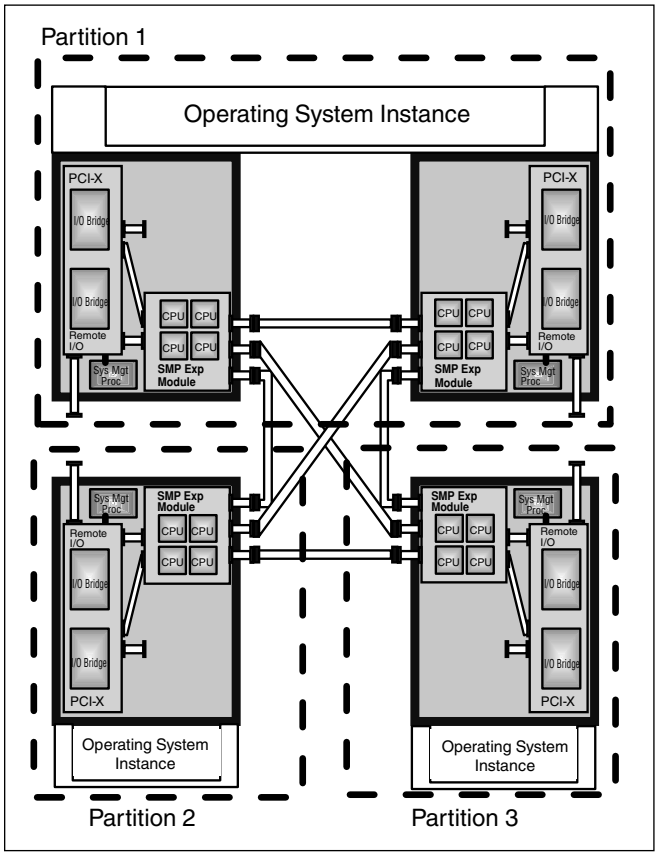


Figure 5 One 8-way, Two 4-way Partitions

Nodes in separate partitions can use the SMP Expansion Ports as a high speed cluster connection. The IPC facilities built into IBM XA-32 chipset provide a very high bandwidth (3.2GB/s) communications link using standard communications protocols. The ability to provide both cache coherent and non-cache coherent data transfer over SMP Expansion Ports provides tremendous flexibility to repartition systems without requiring manual hardware reconfiguration.

XpandOnDemand I/O scalability resolves the traditional problem with system I/O of insufficient PCI-X slots. It allows optional connection of an IBM RXE-100 Remote Expansion Enclosure to each xSeries 440, each one providing up to twelve additional ActivePCI-X slots. Figure 6 shows an example of an RXE-100 connected to an xSeries 440. RXE Expansion Port connections can be long enough to locate the IBM RXE-100 Remote Expansion Enclosure in an adjacent rack to the one containing the xSeries 440, providing tremendous rack configuration flexibility. Redundant RXE Expansion Port interconnect enhances availability for these multiple enclosure configurations. With both SMP Expansion Modules installed in an xSeries 440, a second external cable can be added to a twelve slot RXE-100 for improved performance and cable fail over.

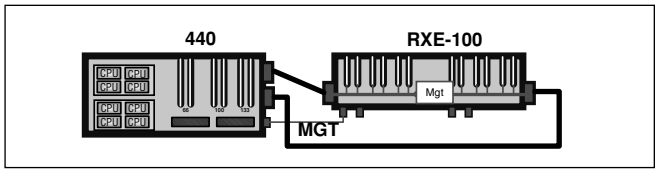


Figure 6 8-way xSeries 440 with RXE-100

BIOS

BIOS is responsible for configuring and initializing the hardware platform prior to loading an operating system and for providing the low level hardware interfaces required by the operating system. The BIOS implementation in the xSeries 440 Enhanced Node is compliant with SMBIOS 2.3 and ACPI 1.0b architecture specifications.

In addition to traditional functions, BIOS supports multinode capabilities of xSeries 440 Enhanced Nodes. When the operating system is loaded, a multinode SMP system will present a single image to the operating system, including any associated system tables. One of the system tables that the BIOS builds will describe static resource affinity which provides processor and memory locality information to the operating system to enable improved Non-Uniform Memory Access (NUMA) performance.

System Management

System Management is a very important consideration for the mission critical, multinode capable systems. IBM Director has added two new capabilities to support multinode configurations, the System Partition Manager and the ActivePCI-X Slot Manager. Figure 7 illustrates a managed, multinode system consisting of three partitions.

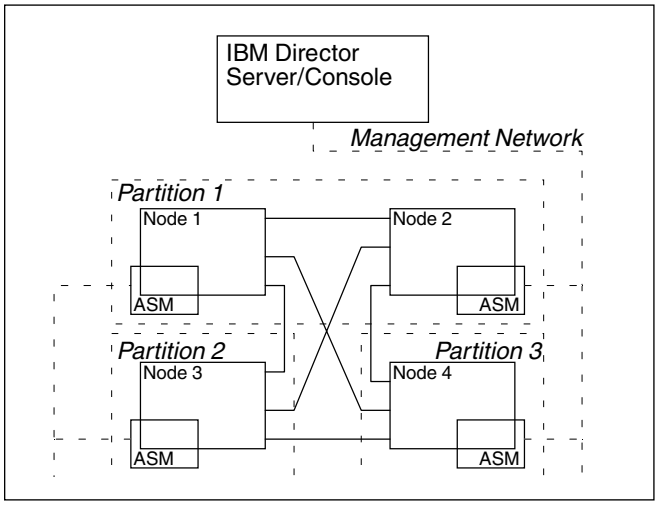


Figure 7 System Management & Partitioning

The xSeries 440 supports static physical partitioning with currently available operating systems. Two to four xSeries 440 Enhanced Nodes can be interconnected through SMP Expansion Ports. Physical partitioning allows the collection of these nodes into SMP systems with a granularity of a single node. Static partitioning provides the ability to reconfigure the nodes to any of the supported configurations during a reboot without the need for any physical reconfiguration. In addition to partition configuration, the System Partition Manager supports power on/off, boot and system shutdown control on an individual partition basis.

The ActivePCI-X Slot manager provides a graphical, hierarchical tool to manage I/O adapter configurations. A system based on xSeries 440 Enhanced Nodes can have four nodes, each attached to an RXE-100 Remote Expansion Enclosure for a total of 72 ActivePCI-X slots. The benefit of a tool to manage a system of this complexity cannot be overestimated.

Each xSeries 440 Enhanced Node includes an IBM Remote Supervisor Adapter (RSA) as standard equipment. This adapter, available in many xSeries products, provides system control and monitoring, system problem alerts, the system error log and out of band communications resources enabling remote system management. RSA operation is independent of system operational state and is available for remote management operations even when the system itself is powered off or inoperable.

In addition to the standard features provided by the RSA, RSA is an integral component in the management of multinode systems. In a multinode server configuration, all nodes are connected through RSA to the IBM Director Server and Console. The RSA maintains local node partition configuration data and controls the state of both the node and any attached RXE-100s. An important standard feature of the RSA included with the xSeries 440 is that it supports graphics redirection and “headless” operation, allowing complete remote management.

Reliability/Availability/Serviceability

The IBM XA-32 chipset has an extensive set of EXA RAS features, such as memory scrubbing, memory mirroring, Memory Hot Add/Hot Replace, memory chipkill, Memory ProteXion and ActivePCI-X, which have been mentioned previously.

IBM XA-32 also provides an exhaustive set of error detection capabilities. ECC is used on the XceL4 Server Accelerator Cache on both the data and directory arrays. SMP Expansion Ports and RXE Expansion Ports have robust error recovery capabilities based on CRC and cable fail over. All major busses are protected with ECC or parity as well as bus protocol checkers.

IBM XA-32 error reporting is based on the principle of first error capture. First error capture is a method where the platform collects sufficient information at the time an error occurs to allow identification of the failing component. This method reduces dependence on recreating the error with diagnostic programs and reducing diagnostic escapes. IBM XA-32 supports first error capture with an extensive set of run time error detection capabilities and error logging registers, as well as an error reporting methodology that ensures all applicable error data is available for analysis.

Unrecoverable platform errors halt the system and report the error to the Remote Supervisor Adapter. Halting the system allows RSA to accurately log error information from the IBM XA-32 chipset error registers into a non-volatile error log. This ensures that error information is not lost for severe errors while guaranteeing data integrity.

The xSeries 440 includes Lightpath Diagnostics which quickly leads to the failing component. Enhancements include a pop-out second level lightpath indicator that is visible without moving the server, and indicators on

the SMP Expansion Module that can indicate failed components even when it is removed from the server.

Power/Packaging

The xSeries 440 enhanced node is packaged in a compact 4-EIA Unit x 27.5" enclosure that is designed to fit into a 19" rack. Service access is primarily from the top, with power supplies and disk drives serviced from the front. Media devices, operator panel and lightpath indicators are also accessed from the front.

Layout for the major components is shown in Figure 8.

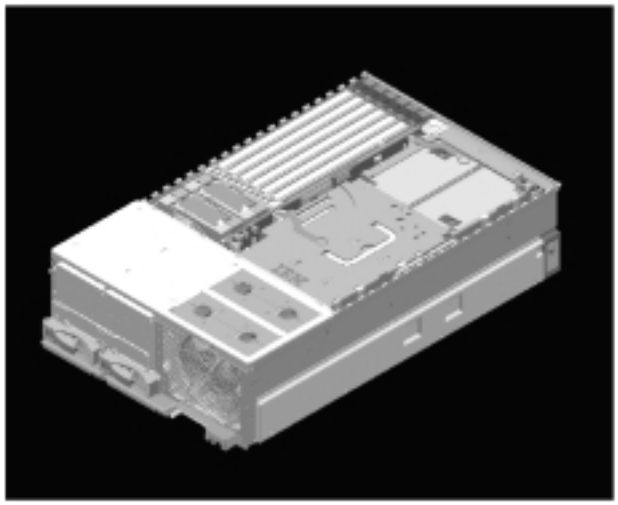


Figure 8 Mechanical Layout

At the left-front of the system are the two power supplies, providing redundant power. Immediately underneath the power supplies are the two Hot Swap Disk Drives and below them are the CD-ROM and Diskette Drive devices. Immediately to the rear of the power supplies are two redundant fans that cool the power and I/O components. Continuing to the rear are the six ActivePCI-X slots. Integrated I/O devices are located beneath the PCI-X slots at the rear of the system.

The two redundant fans for cooling the processors and memory are at the right-front of the system. The SMP Expansion Modules, each containing up to four processors, up to 16 memory modules, the CSC32 chip and the MIOC chip, are located behind these fans. The standard SMP Expansion Module is located in the bottom of the system and the optional SMP Expansion Module is at the top, providing incredible packaging density for an 8-way system.

Cooling is front to rear with independent airflow channels on the left and right sides of the enclosure. The enclosure was carefully designed to eliminate blockages in the airflow path in the right side of the system in order to adequately cool the high power processors and memory. All fans are redundant and hot swap capable with service access from the top.

The power supplies are 1050 watts each with a power density that is greater than nine watts per cubic inch. One power supply can power the system and the second provides redundancy. The power supplies are hot swap capable from the front. Two AC line cords are included with the system. The power subsystem is implemented as distributed 12V and the system's power on/off control is implemented on a power back plane. This allows for a simplified design that supports robust continuous power for the RSA and system management components.

Internal power distribution is by multiple 12 volt branch lines, each with protection from hazardous energy exposure. Voltage regulators are distributed throughout the enclosure to convert 12 volts to the various voltages required by the system electronics.

Conclusion

The xSeries 440's implementation of EXA architecture and IBM's XA-32 chipset provides a powerful new platform for high performance servers. The packaging of eight processors and 32GB of memory in a single 4-EIA unit enclosure is unmatched in industry standard servers. Added to the system and I/O scalability capabilities of EXA, the xSeries 440 is a very flexible and capable mission critical industry standard server.

Acknowledgments

The development of the IBM XA-32 chipset and the xSeries 440 is the result of many people working together across IBM. Sudhir Dhawan was instrumental in defining the base chipset architecture and its application to the xSeries 440. The BIOS team lead by Shel Sigrist and Scott Dunham jumped many hurdles, incorporating new technology into industry standard structures. Dave Jensen and the mechanical and thermal team did a tremendous job fitting eight processors and 32 memory modules into a space efficient, 4-EIA unit rack package. Finally, the Kirkland team, led by Bob Stephens, was key to development of the System Partition Manager, a key component of the x440 system.

References

1. J. D. Brown, S. Dhawan, *Enterprise X-Architecture Technology Overview*, IBM Enterprise X-Architecture Technology, p.1
2. J. Bozek, *Partitioning in the EXA Technology*, IBM Enterprise X-Architecture Technology, p.121
3. D. Hurlimann, Y. Ku, H. Schultze, T. Tam, C. Walter, L. Wilson, *xSeries 360, IA-32 Basic Node*, IBM Enterprise X-Architecture Technology, p.71
4. J. Haidinyak, N. Sherali, *xSeries RXE-100 Remote Expansion Enclosure*, IBM Enterprise X-Architecture Technology, p.93
5. J. M. Borkenhagen, R. D. Hoover, K. M. Valk, *EXA Cache/Scalability Controllers*, IBM Enterprise X-Architecture Technology, p.37
6. L. Blackmon, B. Drehmel, T. Greenfield, J. Kirscht, J. Marcella, D. Shedivy, *EXA Memory/IO Controller*, IBM Enterprise X-Architecture Technology, p.9
7. T. Moe, C. Paynton, R. Shearer, S. Willenborg, C. Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31

IPF Enhanced Node Prototype

– Jim Hanna

Introduction

The IPF Enhanced Node Prototype (IPF Node) marries IBM's Enterprise X-Architecture™ (EXA) technology[1] with Intel's new Itanium™ Processor Family (IPF) processor architecture. This unique combination leverages the strengths of both technologies to provide a new class of high performance, highly scalable, mission critical servers.

Product Description

Each IPF Node supports four IPF processors in a 4-EIA (1-EIA=1.75inches) unit rack optimized enclosure. Each comes standard with 1GB of system memory based on industry standard DDR SDRAM memory module technology. 256MB, 512MB, 1GB and 2GB memory modules may be used. Up to 28 memory modules may be installed for a maximum system memory configuration of 56GB per node.

I/O is provided by six full length ActivePCI-X™ slots. Integrated I/O includes a RAID controller and dual Gigabit Ethernet. Legacy ports such as serial and parallel have been replaced with Universal Serial Bus (USB) connections.

The 4-EIA enclosure supports two Hot Swap Hard Drive Bays along with two ThinkPad Media Device bays. Redundant power and cooling are standard.

XpandOnDemand™ scalability is a unique capability of EXA implemented in the IPF Node. It is the capability to add additional system or I/O enclosures to increase capacity as needed, instead of having to purchase unused headroom within each system unit enclosure. Up to four IPF Nodes may be cabled within a rack for a maximum 16-way Symmetric Multiprocessor (SMP) configuration with up to 224GB of system memory. Interconnected systems support physical partitioning[2], allowing multiple operating system (OS) images to run on the interconnected IPF Nodes if desired. XpandOnDemand also provides I/O scalability. One 3-EIA unit RXE-100 Remote Expansion Enclosure may be cabled to each IPF Node providing up to 12 additional full length ActivePCI-X slots each. This provides a maximum capability of 72 full length PCI-X slots for a maximum system incorporating four 4-EIA IPF Node enclosures and four 3-EIA RXE-100 enclosures.

The IBM Remote Supervisor Adapter is standard. Software support includes standalone Lightpath Diagnostics, Concurrent Diagnostics and IBM Director software that includes an Active Partition Manager supporting multinode configuration.

The IPF Node requires 64-bit operating systems. 64-bit versions of Windows.NET and Linux™ are planned. Both 64-bit and 32-bit middle ware and applications are supported, with 64-bit preferred for any performance critical software.

Node Architecture & Performance

The IPF Node system structure is shown in Figure 1. The top portion of the figure shows the processor/memory subsystem and the bottom part shows the I/O subsystem.

The processor subsystem is based on the IBM XA-64 Cache/Scalability Controller 64 (CSC64) chip[3]. This chip interconnects the processors, XceL4™ Server Accelerator Cache, SMP Expansion Ports for multinode interconnect and the IBM XA-64™ Memory/IO Controller (MIOC) chip[4].

The processors are Intel second generation IPF. These processors have been significantly enhanced over Intel's first generation IPF.

The processor system bus meets all Intel architectural and electrical requirements for attachment of up to four IPF processors. Address and control signals operate in common clock mode at 200MHz, double the frequency of IA-32 systems. The data signals are source synchronous double pumped, providing a transfer rate of 400M transfers per second. The data bus width for IPF has been doubled from IA-32 to 16 bytes, providing a peak data transfer capacity of 6.4GB/s.

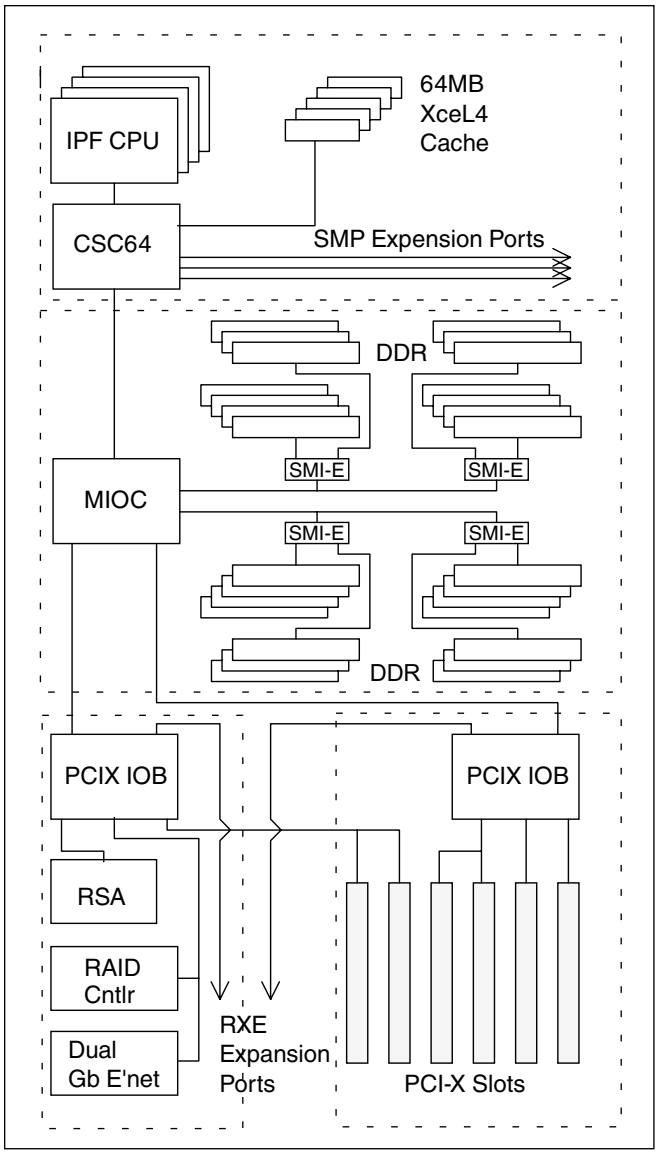


Figure 1 System Structure

The CSC64 chip provides a large 64MB XceL4 Server Accelerator Cache. This cache is 4-way set associative and inclusive of the individual processor caches. The cache directory is implemented internally to the CSC64, while the data array is implemented externally with 128Mb DDR SDRAM modules originally developed for high performance graphics applications. Again, these chips operate at 200MHz for address/control with the data double pumped for 400M transfers per second. The wide bus width graphics SDRAMs allow implementation of a 16 byte wide cache data bus, matching the processor system bus bandwidth at 6.4GB/s.

The CSC64 also provides three SMP Expansion Ports that allow up to four identical IPF Nodes to be connected together using cables up to four meters long to make larger systems. The chip includes a directory that allows coherent operation of multiple nodes by tracking cache lines from local memory that are cached in another node. This directory is 7-way set associative. A non-cache coherent method of scalability port data transfer, known as Inter-Process Communication (IPC), is also supported by the CSC64 to support multinode cluster configurations. SMP Expansion Ports are two bytes wide, simultaneous bidirectional, double pumped, source synchronous links operating at a base frequency up to 400MHz. This provides a total bandwidth up to 3.2GB/s per port for a aggregate bandwidth up to 9.6GB/s.

Finally, the CSC64 provides the path to the MIOC chip. Since the MIOC is common with IBM XA-32 chipset based systems, the interface provided is the eight byte wide QuadT bus. This bus has a 100MHz base frequency, with double pumped address/control and quad pumped data providing a bandwidth of 3.2GB/s.

The MIOC provides two independent memory ports to a DDR SDRAM memory array. Each memory port is eight bytes wide, operating at 400M transfers per second. This provides a memory bandwidth of 3.2GB/s per port, or a total of 6.4GB/s. Generally, the 128 byte cache lines are interleaved between ports.

Each memory port is connected to two System Memory Interface-Enhanced (SMI-E) chips, each of which provides two memory module busses each capable of supporting four industry standard DDR SDRAM memory modules. This bus structure allows a maximum of 28 memory modules to be installed, allowing the large 56GB per node memory capacity using 2GB DIMMs. The minimum supported memory configuration is two memory modules attached to a single memory port.

Chipkill ECC is included in the MIOC, enhancing system availability compared to standard memory ECC schemes. Chipkill is an enhanced ECC algorithm providing error correction not only for single bit errors, but also for clusters of 4-bit errors where the bits are all contained in a single memory chip. This capability allows for continued system operation in the presence of any single memory chip failure mechanism instead of only the bit line failure mechanisms of standard ECC algorithms.

EXA Active Memory technology implemented in the MIOC provides Memory Scrubbing, Memory ProteXion™, Memory Mirroring and Memory HotSwap/HotAdd capabilities for the system.

Scrubbing hardware in the MIOC continually accesses memory locations. Upon discovery of a correctable

error, the location is rewritten to correct the error and the error is logged for Predictive Failure Analysis (PFA). This prevents the buildup of correctable errors to the point where random distribution of correctable errors could result in an uncorrectable error.

Memory ProteXion is an EXA technology providing an additional level of protection from memory subsystem errors compared to ECC with Chipkill. Memory ProteXion provides redundant memory data paths with an ability to reroute memory data bits. When a significant number of correctable memory errors are detected, the failing bits are replaced by redundant bits, eliminating the errors. This greatly reduces the probability of encountering uncorrectable errors, improving overall system availability.

Memory mirroring is the complete redundancy of the memory arrays, providing the maximum level of system availability. It allows system operation to continue even when there are errors that are uncorrectable even with the enhanced ECC algorithms. This is done by maintaining a separate copy of the memory contents in memory attached to each of the two memory ports. When an uncorrectable memory error is detected, the access automatically switches to the alternate port to obtain a correct copy of the memory data.

Memory mirroring also provides the technical foundation for the Memory Hot Swap and Memory Hot Add features of Active Memory. Memory Hot Swap is the capability of replacing a failed memory DIMM while the system remains operational. When the system detects an uncorrectable error in mirrored mode, memory accesses are switched to the non-failing memory port and the failing memory module is identified. Service personnel may then locate the failing memory module, disable and remove power to memory modules on the failing port, replace the failing memory module and re-enable mirroring without taking the system down.

Memory Hot Add is the capability to add memory to the memory subsystem without taking down the system. This feature requires an operating system capable of recognizing increased memory capability without requiring a reboot, and is based on mirroring and the memory port disable and power controls mentioned previously. While in mirrored mode, each memory port in turn is disabled and powered off, memory module(s) added and re-enabled. The final result is more memory in both ports of the mirrored memory, which can then be reported to the operating system for its use.

The MIOC is also the connection point for the I/O subsystem. It provides two RXE Expansion ports. Each of these ports is routed to an IBM XA-64 PCI-X I/O Bridge (PCI-X IOB) chip[5]. RXE Expansion Port technology is similar to SMP Expansion Port technology, but operates at a lower base frequency. The 250MHz base speed yields a bandwidth of 2GB/s per RXE Expansion Port for a total I/O bandwidth of up to 4GB/s into the memory subsystem.

The I/O subsystem has much in common with that of the xSeries 440 Enhanced Node[6]. It includes two PCI-X IOB chips. Each PCI-X IOB has two RXE Expansion Ports. One of these connects to the MIOC while the other provides the external RXE Expansion Port connection to additional PCI-X IOB chips located in separate RXE-100 enclosures.

Each PCI-X IOB generates three PCI-X bus segments. Two of these chips are used in the enhanced node for a total of six 64-bit PCI-X bus segments. These six segments are distributed to provide two 133MHz ActivePCI-X slots, two 100MHz ActivePCI-X slots and two 66MHz ActivePCI-X slots, along with connections to integrated I/O such as the RAID controller and Dual Gigabit Ethernet.

Scalability

The IPF Node provides the standard hardware scalability mechanisms for high performance servers, supporting up to four processors, 56GB of memory, ServeRAID Adapters attached to external Storage Enclosures and PCI-X based Network Adapters. In addition to these standard forms of scalability, the IPF Node supports XpandOnDemand capabilities for both SMP and I/O scalability which are unique in the industry.

XpandOnDemand SMP scalability is the ability to interconnect multiple IPF Nodes through high speed SMP Expansion Ports. Each IPF Node provides three SMP Expansion Ports, allowing up to four IPF Nodes to be connected together. Communications over these links can be either cache coherent for symmetric multiprocessing (SMP) configurations or Inter-Process Communications (IPC) for clustered configurations. Supported configurations of IPF Nodes are illustrated in Figure 2.

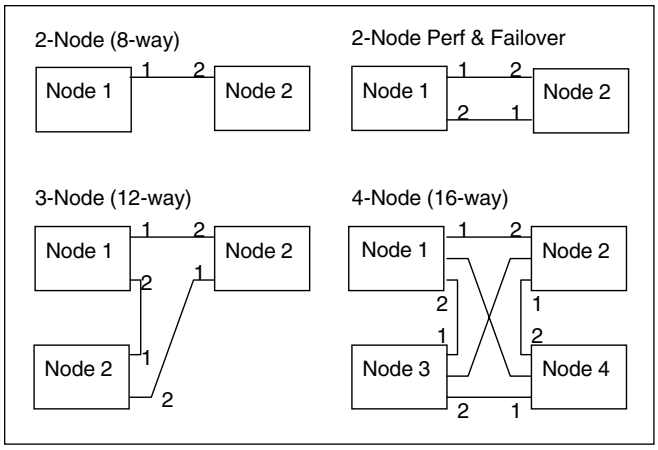


Figure 2 Multinode Configurations

These configurations enable scaling of up to 16 processors and 224GB of system memory in a single SMP system.

System partitioning provides the ability to run independent operating systems on different nodes of a server connected with SMP Expansion Ports. Partitions can consist of one, two, three or four nodes, allowing as many as four operating system images to be running concurrently. Systems can be re-partitioned using the Active Partition Manager, providing a great deal of flexibility to manage system resources.

IPF Nodes in separate partitions can use the SMP Expansion Ports as a high speed cluster connection. The IPC facilities built into IBM XA-64 chipset provide a very high bandwidth (3.2GB/s) communications link using standard communications protocols. The ability to provide both cache coherent and non-cache coherent data transfer over SMP Expansion Ports provides tremendous flexibility to re-partition systems without requiring manual hardware reconfiguration.

XpandOnDemand I/O Scalability resolves the traditional problem with system I/O of insufficient PCI-X slots. It allows optional connection of an IBM RXE-100 Remote Expansion Enclosure[7] to each IPF Node, each one providing up to twelve additional ActivePCI-X slots. Redundant RXE Expansion Port interconnect enhances availability for these multiple enclosure configurations.

BIOS

BIOS is responsible for configuring and initializing the hardware platform prior to loading an operating system and for providing the low level hardware interfaces required by the operating system. The IPF Node includes a new BIOS architecture developed for IPF systems. The new architecture is based on the Processor Abstraction Layer (PAL) & System Abstraction Layer (SAL) abstractions defined by Intel for IPF based systems, along with the Extensible Firmware Interface (EFI) specification. The BIOS implementation in the IPF Node is compliant with SMBIOS 2.3, SAL 2.6, EFI 1.1 and ACPI 2.0 architecture specifications. This new architecture is highly structured and provides an improved level of independence between the operating systems and the hardware platform.

The new IPF architecture BIOS introduces a new, more robust error reporting architecture called Machine Check Abort. This architecture abstracts error reporting through the BIOS PAL and SAL firmware, isolating the operating system from processor and platform hardware implementation differences. This architecture gives the operating system the ability to participate in error logging and recovery to a much greater degree than IA-32 systems. Depending on the severity of error, the operating system may log the error, kill the failing task or reboot the system.

In addition to traditional functions, BIOS supports multinode capabilities of IPF Nodes. When the operating system is loaded, a multinode SMP system will present a single image to the operating system, including a single instance of EFI & SAL abstractions and any associated system tables. The memory map created for these systems is slotted to allow the addition of memory and I/O resources to any IPF Node. As multinode configurations are non-uniform memory access (NUMA) architecture, BIOS utilizes the capabilities of the ACPI 2.0 specification to pass hardware topology information to the operating system. The operating system may then use this information to optimize resource allocation and task management for improved performance.

The Extensible Firmware Interface (EFI) is another key architectural element of BIOS. As a part of the EFI implementation, an EFI partition for hardware platform use of at least 100MB will be created. An EFI shell will be created, allowing standalone diagnostics and utility software previously running on DOS to work in an IPF environment. EFI compliance is required of operating systems to boot and function properly on the IPF Node.

System Management

System Management is a very important consideration for the mission critical, multinode capable systems. IBM Director has added two new capabilities to support multinode configurations, the Active Partition Manager and the ActivePCI-X Slot Manager. Figure 3 illustrates a managed, multinode system consisting of three partitions.

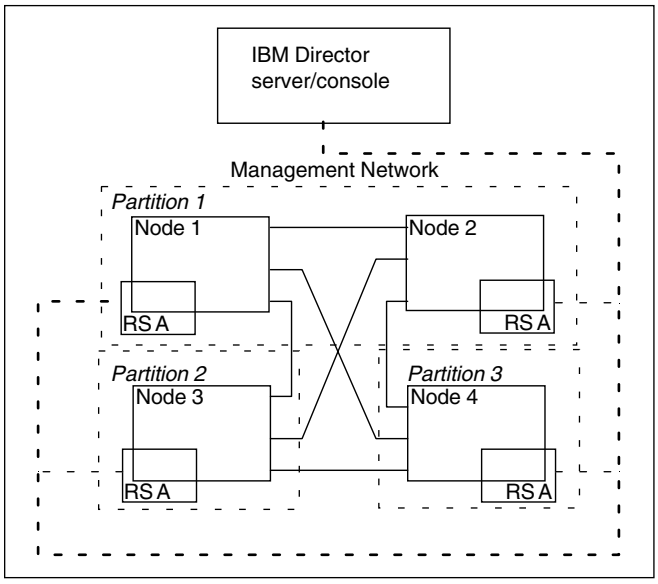


Figure 3 System Management & Partitioning Example

The IPF Node supports physical, static partitioning with currently available operating systems. From two to four IPF Nodes can be interconnected through SMP Expansion Ports. Physical partitioning allows the collection of these nodes into SMP systems with a granularity of a single node. A four node system, for example, can be configured by the Active Partition Manager to provide 4 x 4-way SMP systems, 2 x 8-way SMP systems, 2 x 4-way & 1 x 8-way SMP systems or
1 x 16-way SMP system. Static partitioning provides the ability to reconfigure the nodes to any of the supported configurations during a reboot without the need for any physical reconfiguration. In addition to partition configuration, the Active Partition Manager supports power on/off, boot and system shutdown control on an individual partition basis.

The Active PCI Slot manager provides a graphical, hierarchical tool to manage I/O Adapter configurations. A system based on IPF Nodes can have four nodes, each attached to a RXE-100 Remote Expansion Enclosure for a total of 72 ActivePCI-X Slots. The benefit of a tool to manage a system of this complexity cannot be overestimated.

Each IPF Node includes an IBM Remote Supervisor Adapter (RSA) as standard equipment. This adapter, available in many xSeries products, provides system control and monitoring, system problem alerts, the system error log and out of band communications resources enabling remote system management. RSA operation is independent of system operational state and is available for remote management operations even when the system itself is powered off or inoperable.

In addition to the standard features provided by the RSA, it is an integral component in the management of multinode systems. In a multinode server configuration, all nodes are connected through RSA to the IBM Director Server and Console. The RSA maintains local node partition configuration data and controls the state of both the node and the attached RXE-100s.

Reliability/Availability/Serviceability

The IPF Node combines features of both the Intel IPF processor architecture and Enterprise X-Architecture technology implemented in the IBM XA-64 chipset to provide a system with superior RAS features.

The IBM XA-64 chipset has an extensive set of EXA RAS features, such as Memory Scrubbing, Memory Mirroring, Memory Hot Add/Hot Replace, Memory Chipkill, Memory ProteXion and ActivePCI-X, which have been mentioned previously.

IBM XA-64 also provides an exhaustive set of error detection capabilities. ECC is used on the XceL4 Server Accelerator Cache on both the data and directory arrays. SMP Expansion Ports and RXE Expansion Ports have robust error recovery capabilities based on CRC. All major busses are protected with ECC or parity as well as bus protocol checkers.

IBM XA-64 error reporting is based on the principle of first error capture, leveraging the IPF reporting structure where appropriate. First error capture is a method where the platform collects sufficient information at the time an error occurs to allow identification of the failing component. This method reduces dependence on recreating the error with diagnostic programs and reducing diagnostic escapes. IBM XA-64 supports first error capture with an extensive set of run time error detection capabilities and error logging registers, as well as an error reporting methodology that ensures all applicable error data is available for analysis.

Error reporting in the IPF Node takes advantage of both IBM XA-64 chipset technology and IPF architecture enhancements. Processor errors and I/O subsystem errors are reported through the IPF Machine Check Abort mechanism, as are recoverable memory subsystem errors. Unrecoverable memory subsystem errors, which are likely to prevent execution of Machine Check Abort software, halt the system and report the error to the Remote Supervisor Adapter. Halting the system allows RSA to accurately log error information from the IBM XA-64 chipset error registers into a nonvolatile error log. This combination of error reporting methods maximizes the ability of an operating system to recover and contain errors while insuring that error information is not lost for the most severe errors.

Power/Packaging

The IPF Node is packaged in a compact 4-EIA x 19" x 27.5" enclosure. Service access is primarily from the top with power supplies and hard drives serviced from the front. Media devices, operator panel and lightpath indicators are also accessed from the front.

Layout for the major components is shown in Figure 4.

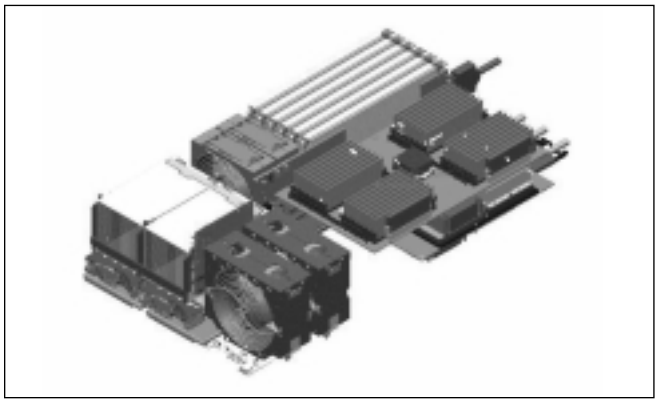


Figure 4 Mechanical Layout

At the left-front of the system are the two power supplies, providing redundant power. Immediately underneath the power supplies are the two hot swap disk drives and below them are the CD-ROM and LS240 high capacity diskette media devices. Immediately to the rear of the power supplies are two redundant fans that cool the power and I/O components. Continuing to the rear are the six ActivePCI-X slots. Integrated I/O devices are located beneath the PCI Slots at the rear of the system.

The two redundant fans for cooling the processors and memory are at the right-front of the system. A single memory card containing locations for the 28 DDR SDRAM memory modules is to the rear of the fans and located at the top of the system to provide convenient access for memory Hot Add and Memory Hot Replace. The IBM XA-64 MIOC is also located on the memory card. The processor card containing the four Intel IPF processors and the IBM XA-64 CSC64 chip is located directly beneath the memory card. Two processors are mounted on each side of this card to maximize the electrical performance of the processor system bus. The SMP Expansion Ports exit the enclosure at the rear from the processor card.

Cooling is front to rear, with independent airflow channels on the left and right sides of the enclosure. The enclosure was carefully designed to eliminate blockages in the airflow path in the right side of the system in order to adequately cool the high power processors and memory. All fans are redundant and hot swap capable with service access from the top.

The power supplies are 1050 watts each. One power supply can power the system and the second provides redundancy. The power supplies are hot swap capable from the front.

Internal power distribution is by multiple 12 volt branch lines, each with protection from hazardous energy exposure. Regulators are distributed throughout the enclosure to convert 12 volts to the various voltages required by the system electronics.

Conclusion

The IPF Node's use of Intel's IPF processor architecture and IBM's XA-64 chipset illustrates a powerful new platform for high performance servers. The large memory space architected into IPF is fully realized with the 224GB capability available with a four node SMP configuration. I/O is equally scalable using IBM RXE-100 Remote Expansion Enclosures to support the vast amounts of storage and network connections required with such a powerful system. This combination drives industry standard servers well into markets previously populated only with proprietary solutions.

Acknowledgments

Many people are involved bringing EXA and IBM XA-64 based products to market. I would particularly like to thank Sudhir Dhawan, who as system architect defined XA-64 to be applicable to IPF as well as IA-32, John Borkenhagen, who lead the design for the CSC64 chip

for IPF based systems, and Randy Kolvick who ensured maximum compatibility between the xSeries 440 and the IPF Enhanced Node Prototype.

References

1. J. D. Brown, S. Dhawan, *Enterprise X-Architecture Technology Overview*, IBM Enterprise X-Architecture Technology, p.1
2. J. Bozek, *Partitioning in the EXA Technology*, IBM Enterprise X-Architecture Technology, p.121
3. J. M. Borkenhagen, R. D. Hoover, K. M. Valk, *EXA Cache/Scalability Controllers*, IBM Enterprise X-Architecture Technology, p.37
4. L. Blackmon, B. Drehmel, T. Greenfield, J. Kirscht, J. Marcella, D. Shedivy, *EXA Memory/IO Controller*, IBM Enterprise X-Architecture Technology, p.9
5. T. Moe, C. Paynton, R. Shearer, S. Willenborg, C. Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31
6. M. Bland, R. Kolvick, *xSeries 440, IA-32 Enhanced Node*, IBM Enterprise X-Architecture Technology, p.77
7. J. Haidinyak, N. Sherali, *xSeries RXE-100 Remote Expansion Enclosure*, IBM Enterprise X-Architecture Technology, p.93

xSeries RXE-100 Remote Expansion Enclosure

– Jim Haidinyak, Nusrat Sherali

Introduction

The RXE-100 Remote Expansion Enclosure, when coupled with xSeries 360[1], xSeries 440[2] or IPF Enhanced Node Prototype (IPF Node)[3] host systems, creates systems of unparalleled performance and I/O capability. The RXE-100 enhances the ActivePCI-X capabilities of Enterprise X-Architecture™[4] based systems by providing up to twelve additional 64-bit PCI-X slots. The RXE-100 Remote Expansion Enclosure is packaged in a 3-EIA unit rack drawer.

Product Description

The base RXE-100 includes one backplane providing six 64-bit PCI-X slots on three busses. If both adapters are populated, the PCI-X bus can run at 100MHz. With a single adapter, the bus can operate at 133MHz. An optional second backplane may be added, yielding an additional six 64-bit PCI-X slots.

The RXE-100 is cabled to the RXE Expansion Ports on the host system with a cable. The cable length is in meters and the RXE-100 may be located in either the same rack as the host system or an adjacent rack, allowing tremendous configuration flexibility.

The RXE-100 is packaged in a standard 19" rack drawer, 3-EIA units high and 26" deep. Redundant power and cooling are standard.

System Management for the RXE-100 is provided by a standard Drawer Management Controller (DMC). This controller is attached to the host system using a RS-485 cable. The RXE-100 is managed by the Remote Supervisor Adapter (RSA) that is standard in the Enterprise X-Architecture (EXA) host systems.

The RXE-100 can be attached to the EXA host systems and run many standard operating systems, including Microsoft® Windows, Novell Netware® and multiple Linux® distributions.

System Architecture

The xSeries RXE-100 Remote Expansion Enclosure system structure is shown in Figure 1.

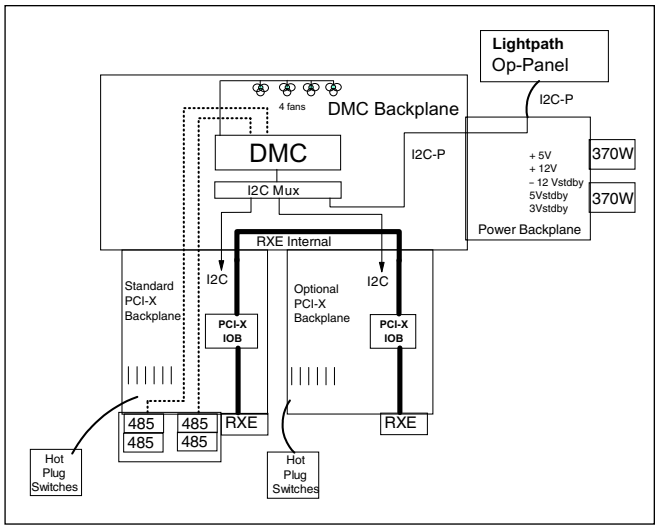


Figure 1 RXE-100 Structure

The RXE-100 includes one standard backplane. A second backplane is optional. Each backplane provides six 64-bit PCI-X slots, one PCI-X I/O Bridge (PCI-X IOB) and a Hot Plug Controller. The PCI-X IOB[5] chip has two RXE Expansion Ports. One of these is routed internally to the optional PCI-X backplane, while the second is routed to an external connector as shown in Figure 2. If a single backplane is installed, the RXE-100 presents one external RXE Expansion Port. When both standard and optional backplanes are installed, the RXE-100 presents two external RXE Expansion Ports out the back of the enclosure. This routing enables host systems connected to an RXE-100 to access PCI-X slots on both PCI-X backplanes with one cable connection. Alternatively, a separate host system may be attached to each PCI-X backplane so that each host system uses six of the twelve PCI-X slots in a single RXE-100 enclosure. This provides tremendous system configuration flexibility.

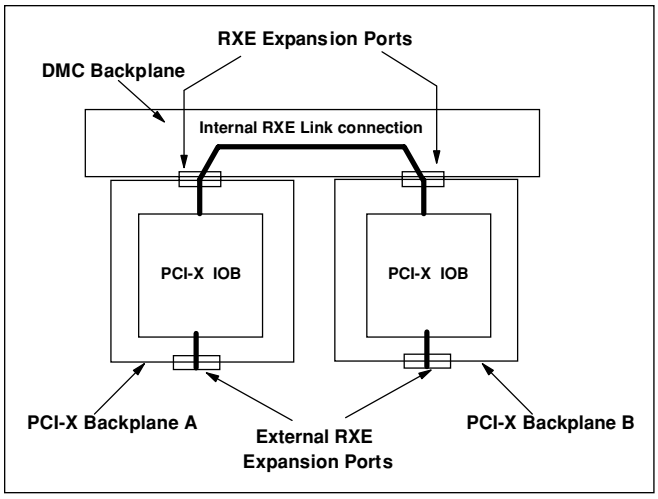


Figure 2 RXE Expansion Port Routing

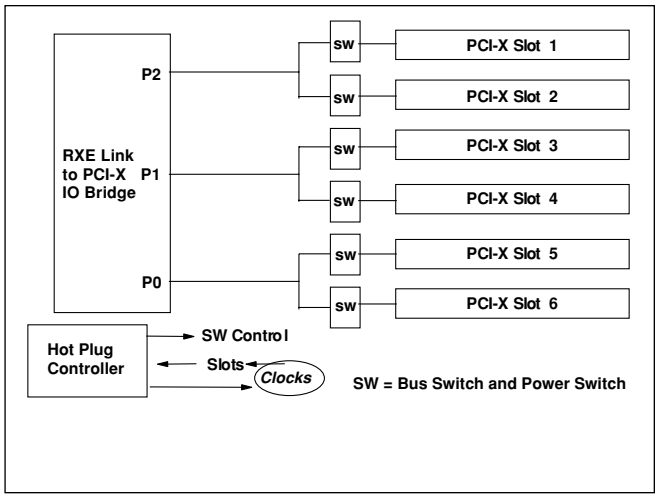


Figure 3 PCI-X Bus Structure

Each PCI-X IOB chip generates three PCI-X busses (P0, P1, P2) as shown in Figure 3. Each PCI-X bus is wired to two slots. If only one slot on each bus is populated with an adapter, the second slot is electrically removed from the bus and the populated slot will support 133MHz operation. When both slots on a bus are populated the slots will support a maximum of 100MHz operation. Hot Plug Controller configures installed adapters for maximum performance. It identifies the adapter type, sets up the PCI/PCI-X bus mode and bus speed, and powers the slot on.

On each PCI-X backplane there is a private I2C bus which connects the PCI-X IOB chip to the Hot Plug Controller. PCI hot plug control commands are passed from the PCI-X IOB to the Hot Plug Controller over this I2C bus. The Hot Plug Controller provides proper sequencing for the bus switches and power controls to each Hot Plug PCI-X slot. A PCI-X Hot Plug Switch Card is cable attached to each PCI-X backplane. This card communicates with the Hot Plug Controller that a slot is being accessed for a hot plug operation.

Drawer management support is provided through the Drawer Management Controller (DMC) located on the DMC backplane. Connection to the host system Remote Supervisor Adapter is via the host system's DMC RS-485 port. The RXE-100 has two separate RS-485 DMC links. Two links are provided so that a single RXE-100 can be shared between two host systems. Each host system can obtain drawer management status over its own RS-485 link. Four chassis fans dock to the DMC backplane. Fan tachometer monitoring and speed control are maintained by the Hot Plug Controller. Lightpath diagnostics capability for the RXE-100 is provided by the DMC.

Power is provided by two 370 watt power supplies. One power supply is sufficient to operate a fully populated RXE-100. The second power supply provides redundancy. These power supplies are plugged into a power backplane which provides power controls, DC voltage distribution and hazardous energy protection. Fuel gauge is supported.

Performance And Scalability

The RXE-100 Remote Expansion Enclosure interfaces directly with a host system using the RXE Expansion Port. The RXE Expansion Port consists of a 16-bit bidirectional, differential link operating at 500MHz. This interface provides a peak data transfer rate of 2GB/s between the host system and the RXE-100.

Up to twelve additional PCI-X adapter slots can be added to the EXA-based host system with the addition of an RXE-100 Remote Expansion Enclosure. This unprecedented I/O expansion capability can be used to enable massive high speed data storage by adding ServeRAID SCSI or high speed Fibre Channel adapters, or to increase network bandwidth by adding multiple Local Area Network adapters to obtain faster response times and support more users.

Reliability/Availability/Serviceability

The RXE-100 Remote Expansion Enclosure is designed for use in mission-critical applications. Redundant hot swap power and cooling is standard with two auto ranging 370watt power supplies and four cooling fans.

All PCI-X slots support hot swap and hot add operations, allowing replacement of failed adapters or improved I/O configurations without bringing down the host system or the RXE-100.

The Drawer Management Controller (DMC) in the RXE-100 is attached directly to the standard Remote Supervisor Adapter in an EXA host system for system management monitoring of vital components, including power supplies, fans, PCI-X backplanes and the Power Backplane. This monitoring also includes Predictive Failure Analysis (PFA) for these components, allowing identification of potential problems before they actually cause the system to go down. The DMC also enables Light Path Diagnostics that simplify diagnosis of a failing component.

Power/Packaging

The xSeries RXE-100 is packaged in a compact 3 EIA unit x 26" deep 19" rack drawer. Service access is primarily from the top, with power supplies, operator panel and lightpath indicators accessed from the front. Layout for the major components is shown in Figure 4.

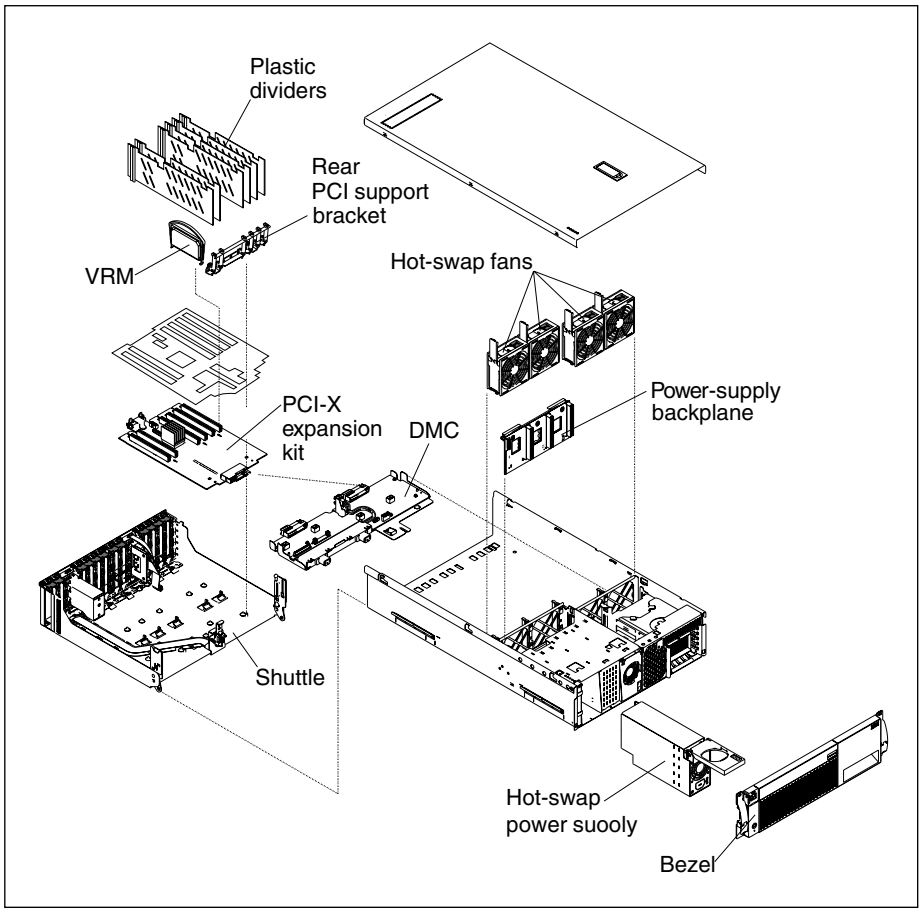


Figure 4 Mechanical Layout

At the left-front of the system are the two 370 watt power supplies. The second power supply is redundant. The power supplies are hot swap capable from the front.

Four fans are located above the DMC backplane to cool the enclosure. Fan speed is controlled in pairs, with two fans per six PCI-X slots. The cooling system is designed to meet thermal specifications even when any single fan fails. If a fan fails the speed on all remaining fans is set to maximum. All fans are hot swap capable from the top.

The DMC backplane is located under the fans and is assembled to the power shuttle assembly. The standard and optional backplanes are located at the rear of the enclosure and are attached to I/O shuttle assembly. The two backplanes are connected using a shuttle latch mechanism. Six PCI-X slots are located on each of the backplanes.

The orange color on components and labels in the enclosure identifies hot-swap components. Customers can install or remove these components while the system remains operational.

Configuration

Until recently, industry specifications for PCI and PCI-X required that all adapter slots be contained in the main system cabinet. This resulted in a tradeoff in server implementation. On one hand, designers wanted to maximize the number of adapter slots in a box, but at the same time, they wanted to shrink the size of the server as much as possible to minimize the rack space required. However, reducing the size of the system beyond a certain point meant sacrificing adapter slots.

One obvious solution was to limit the number of slots inside the server chassis while extending additional bus segments to external I/O expansion units to hold adapters. By moving adapter slots outside of the main system cabinet, the base server can be made much smaller. No industry-standard server vendor has had this capability.

IBM Enterprise X-Architecture technology's XpandOnDemand™ scalability allows users to start out small and inexpensively with a host EXA server such as the xSeries 360. An IBM RXE-100 Remote Expansion Enclosure may be added later when needed as shown in Figure 5.

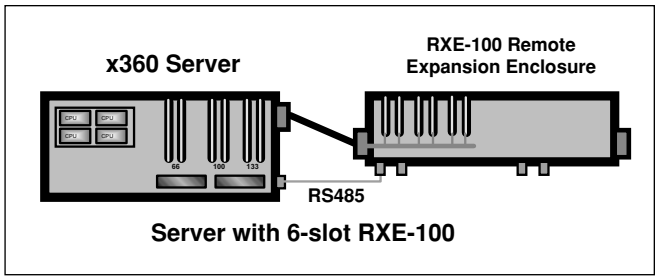


Figure 5 RXE-100 Configuration Examples

The xSeries 360 server is the first 4-way system offered in a slim 3-EIA unit rack optimized form factor, and the RXE-100 is an equally rack-dense 3-EIA unit. Because it is possible to start small with just a single xSeries 360 server, there is the flexibility to expand considerably in a single 42-EIA unit rack. For example, fourteen xSeries 360 servers providing up to 56 processors and 84 PCI-X slots may be installed in a single 42-EIA unit rack. Alternatively, if fewer systems are required, seven xSeries 360 systems and seven RXE-100 Remote Expansion Enclosures providing up to 28 processors and 126PCI-X slots may be installed in the same 42-EIA unit rack.

Topologies

A host system with two RXE Expansion Ports can be looped as shown in Figure 6. This topology allows data to be rerouted if an RXE Expansion Port cable is accidentally disconnected or fails.

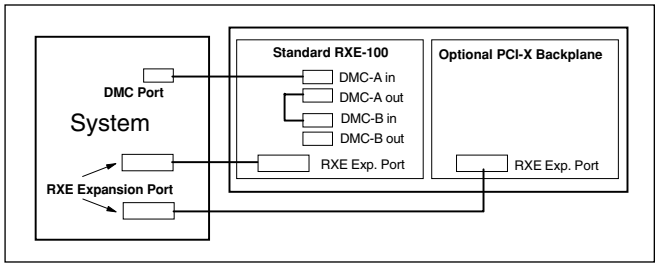


Figure 6 Dual Port System with RXE-100

Additionally, an RXE-100 can be shared between two host systems. PCI-X slots on the standard PCI-X backplane can be owned by one host system and PCI-X slots on the optional PCI-X backplane can be owned by another as shown in Figure 7.

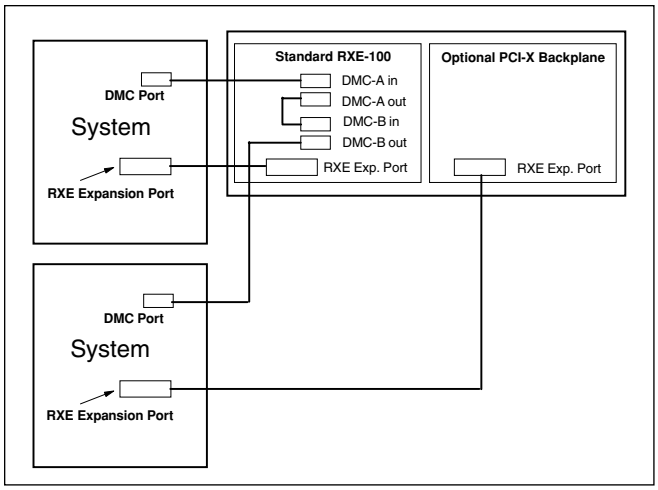


Figure 7 Two Host Systems with RXE-100

Conclusion

The RXE-100 provides and supports several configurations within the EXA technology family. In its simplest form, one drawer with six PCI-X slots can be added to one host. The RXE-100 in turn can support numerous I/O adapters, including RAID technology, that can support a large variety of storage configurations. Over the life of this technology, numerous configurations will be supported, driven by the users unique requirements, fulfilled by the flexibility of the EXA technology.

References

1. D. Hurlimann, Y. Ku, H. Schultze, T. Tam, C. Walter, L. Wilson, *xSeries 360, IA-32 Basic Node*, IBM Enterprise X-Architecture Technology, p.71
2. M. Bland, R. Kolvick, *xSeries 440, IA-32 Enhanced Node*, IBM Enterprise X-Architecture Technology, p.77
3. J. Hanna, *IPF Enhanced Node Prototype*, IBM Enterprise X-Architecture Technology, p.85
4. J. D. Brown, S. Dhawan, *Enterprise X-Architecture Technology Overview*, IBM Enterprise X-Architecture Technology, p.1
5. T. Moe, C. Paynton, R. Shearer, S. Willenborg, C. Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31

EXA Simultaneous Bidirectional Interface Design

– Moises Cases, Daniel N. de Araujo, Nam Pham

Introduction

This paper describes an electrical design optimization methodology for a high-speed point-to-point source-synchronous simultaneous bidirectional interface. These physical links are used to interconnect Enterprise X-Architecture nodes to build symmetric multi-processor (SMP) systems[1], as well as to connect input/output (I/O) subsystems[2] across relative long distance. Major design issues such as attenuation, crosstalk, delay skew, impedance control and inter-symbol interference (ISI) are discussed for long and parallel external interconnections.

Modern CMOS technologies have yielded FET devices with less than 0.1 um gate length, allowing microprocessor to operate at clock frequencies greater than 1 GHz. Similarly, such increase in device density has resulted on memory densities greater than 1 Gbit/chip. This creates a huge amount of data processing capability, but system memory latency and system interface bandwidths are not keeping pace with processor speeds and memory capacities. Furthermore, the I/O and memory capacities required for servers continue to increase the physical distance required between the processors, memory and I/O subsystems. Effective cycles/instruction (CPI) rates have not changed significantly in the past decade. Most realized digital processing power has come from processor frequency, wider busses, cache subsystems and increased memory capacity. Advanced internet and symmetric multi-processor switch systems are already approaching an aggregate data bandwidth of 1Tbit/s.

Optimization of bandwidth, power, pin count or number of wires and cost are the goals for high-speed interconnect design. The electrical performance of the package, board and cable interconnects are the major limiting factors for high-speed and data transfer bandwidth. Today, increased data rates can be achieved using circuit design techniques, such as differential drivers/receivers, and system timing techniques, such as source synchronous clock signaling. Innovative signaling schemes, like simultaneous bidirectional data transmission over one wire, double the effective bandwidth per wire over a point-to-point unidirectional scheme[3].

This paper describes an optimization methodology where circuit design techniques are coupled to board/cable design schemes to minimize undesirable electrical effects such as attenuation, crosstalk, signal/clock jitter and inter-symbol interference.

System Architecture

A differential point-to-point 400MHz double-data rate (DDR) I/O hub structure consisting of two bytes of data is used to illustrate typical attenuation and jitter related design issues for the scalability interface, giving a 3.2 GB/s throughput. The I/O node interface operates at 250MHz DDR simultaneous bidirectional to yield a throughput of 2GB/s over much longer cables. The bi-directional signaling scheme uses current source drivers on both ends of the connecting cable and on-chip termination resistors across the differential pin pairs. The receiver located on either end of the net subtracts the voltage due to the near driver from the total voltage across the termination resistor to obtain the voltage due to the far driver. In order to do the subtraction, an on-chip replica driver creates a scaled copy of the line driver output current. Figure 1 shows a block diagram for the simultaneous bidirectional driver/receiver circuitry.

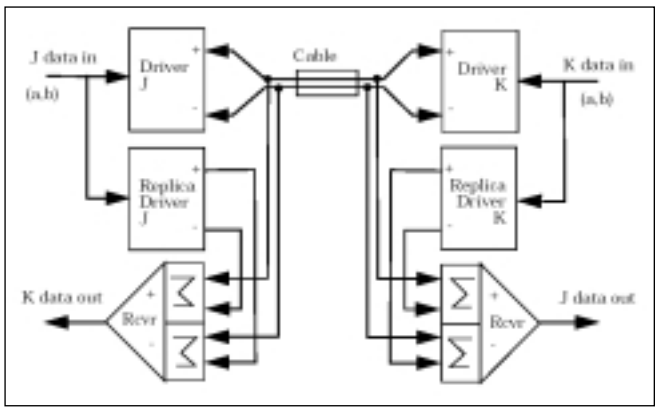


Figure 1 Block Diagram for I/O Circuit

This scheme results in increased overall performance at lower cost since both terminations are on-chip and all the wires can be used for sending and receiving data simultaneously. These links are typically used to connect processor subsystems to build symmetric multi-processor systems, as well as to connect I/O subsystems which are usually daisy-chained in a loop configuration to provide path redundancy in case of a cable or connector failure.

Design Issues

There are five basic electrical design issues for this type of high-speed interface:

1. Skin-effect attenuation in cable and board interconnects and associated data-dependent jitter;
2. Static skew between data and clock for long cable lengths;
3. Dynamic skew due to jitter caused by process, power supply and temperature variations, impedance mismatches and interconnect discontinuities;
4. Near-end and far-end crosstalk and crosstalk induced jitter caused by board, cable and connector assemblies; and
5. Accurate testing and measurement of interface performance. The impact of each issue is evaluated and controlled through careful design guidelines in order to achieve the overall budget and hence timing closure. Simulations were carefully analyzed to confirm the assumptions as well as to understand the simultaneous bidirectional signaling sensitivities.

Design Optimization

Optimization of digital interconnect design is a balance and tradeoff of various factors in order to meet timing. Due to the different nature of challenges, gaussian and non-gaussian, the design goals were specified separately in terms of jitter and skew.

Skin Effect Attenuation Control

The most obvious limitation of copper interconnects is loss[4]. Frequency dependent attenuation such as skin effect, causes data dependent jitter when the frequency of operation of the interface increases to the point where the voltage at the receiver is no longer able to reach the full swing value. This affects timing because the signal will start transitions at different levels and hence cross the threshold faster or slower depending on the previous data pattern.

Precompensation/equalization is commonly used to minimize attenuation data pattern dependent variations in jitter through high pass filters, transmitter and/or receiver equalization. An equalizer high-pass filter cascaded with the cable to reduce the pulse distortion is commonly used, however, this technique wastes board space. The delay skew between different data paths as well as static variations of the reference clock have to be tracked and continuously adjusted. Discrete equalizers work best for only particular cable/board trace lengths and sizes[4]. Precompensation or transmitter equalization circuit is used in this design to reduce ISI jitter. The driver circuitry implements a discrete precompensation logic where the driver current is a function of the present and previous data bit. Additional current is sourced into the channel to reduce the rise time degradation of the signal which will counter the attenuation of the channel and enable the signal to make the full transition at the receiver. With a consistent starting point, the transitions are more predictable and the inter-symbol interference is reduced. Since frequency dependent attenuation is closely related to length of the interconnect, the additional energy needs to be tuned to a certain amount of loss and hence tuned to a particular length of cable. Time domain simulations are needed to quantify jitter due to signal loss for the total interconnect network. This includes cable size, length and type, and board trace width, spacing and length. Given the attenuation guidelines, the designers are free to balance the choices from the various combinations of these factors using attenuation charts such as Figure 2 to meet the budgets and stay within cost.

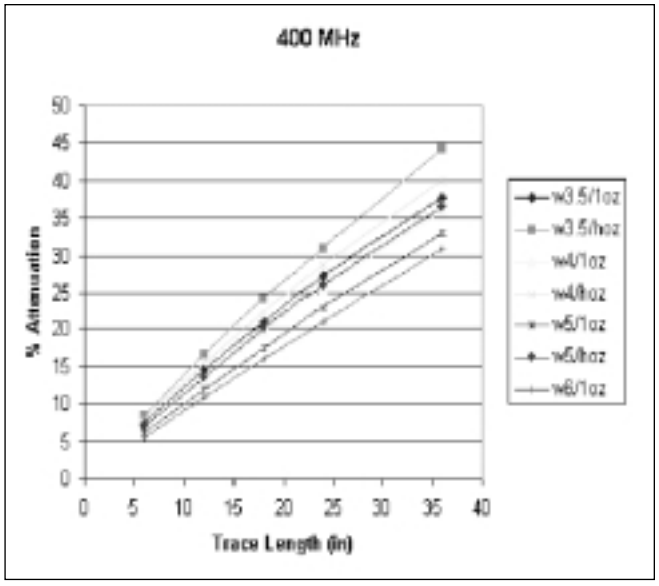


Figure 2 Attenuation vs. Length @ 400 MHz

Static Skew in Long Cable Lengths

Controlling the static delay skew for both data and clock in a source-synchronous bidirectional signaling technique falls into two categories: skew among differential pairs of the bus and skew within a signal's differential pair. For this technique, the design optimization problem reduces to the minimization of skew between the data bits and the link clock and data. Skew among differential pairs can be corrected with the automatic deskew circuit. The receiver circuitry implements deskew circuits to compensate for static skew on the incoming data for up to 2ns so that the link clock samples the data in the middle of the data window, allowing for optimum timing margin for both setup and hold timing requirements. Programmable delay lines are used on-chip to add the appropriate amount of correction delay to each bit. The driver sends synchronization packets with unique patterns to determine how much delay needs to be added to each bit during initialization. The clock is then delayed to the center of the aggregate data eye. A static delay skew budget for each interconnect component in the link (modules, boards, connectors and cable) is allocated within the allowable delay skew correction in the design.

Skew within a pair cannot be corrected by the receiver so every picosecond of in-pair skew seen at the receiver results in a picosecond less in margin. To control the static skew within a pair, strict routing rules were implemented.

Dynamic Skew

Dynamic skew and jitter due to process and power supply are budgeted within the component and board timing allocations while the skew variations due to temperature are compensated by synchronization packets. Periodically, they track delay changes due to temperature changes and adjust the deskew circuitry accordingly.

Near-end and Far-end Crosstalk

Various simulations of board/cable impedance corners and stackup/spacing crosstalk characterization indicate that if crosstalk is kept within 5%, its impact will be within the allocated budget of ± 50 ps. Although differential signaling crosstalk should be minimal due to cancellation of return currents, even-mode effect is present since there is certain skew allowed within a pair. For simultaneous bidirectional interface, both near-end and far-end of the line are being probed and decoded by the corresponding receiver circuit at both ends of the link. Hence, any near-end noise creates a difference voltage in the receiver, as well as crosstalk induced jitter. Crosstalk control can be maintained for different components of the link by proper selection of board trace width/space, cable construction and connector choice (see Table 1). The use of Quad cable construction is very common for parallel interfaces due to size savings, but Twinax cable construction offers better performance and reduced crosstalk.

Table 1 Connector Crostalk Comparison

Rise Time Differential Xtk	MDR (S:G) 2.5:1	LFH (S:G) 3:1	GignaCN Gnd Shield	VHDCI (S:G) 2:1
500 ps				
Near End	3.86%	8.05%	1.17%	13.16%
Far End	4.82%	2.20%	0.33%	4.89%
300 ps				
Near End	4.44%	12.6%	1.72%	17.6%
Far End	5.55%	3.79%	0.39%	8.28%

Dynamic skew jitter is determined by accurate time-domain simulations including process corners, power supply and temperature variations, impedance mismatch (modules, cards, connectors and cable) and data-dependent effects (ISI). Table 2 summarizes a typical dynamic delay skew allocation for a data rate of 800 Mb/s.

Table 2 Summary of Typical Timing Allocation:

Parameter	**Skew(ps)**
ISI (incl. replication error)	520
Crosstalk (boards+connectors)	100
Wiring tolerance	120
Clock jitter	100
Component noise induced delay	100
Receiver setup/hold	90
De-skewing circuit tollerance	200
Total	1250

Interface Testing and Measurement

Testing and measurement were taken at two levels: link level and physical level tests. Link layer level testing is helpful in determining the margin on all deskewed data bits with respect to the automatic clock centering circuitry and the error rate seen at the system level. The ping test is a link level test feature of the macro in which packets

with randomly generated payloads are transmitted along with the cyclic redundancy check (CRC) and acknowledgements are received for each error free packet. Using counters on each port, the error rates can be measured. The bit error rate (BER) is estimated by counting the number of errors at the receiver output and dividing by the total number of transmitted bits over a specified period of transmission. An accurate BER at the link level is not possible since errors are detected through CRC validation which only indicates the presence of an error, but not where or how many bits were the culprit. The error free window (EFW) clock margin test is an extension of the ping test where the automatically centered clock can be manually skewed providing an insight to how well the clock centering circuit is performing in both setup and hold margins. Data deskew relies on synchronization packets that are sent with specific data patterns used to align the data at the receiver and automatically center the clock. The error rate is plotted against deskew values to measure the aggregate data eye (Figure 3).

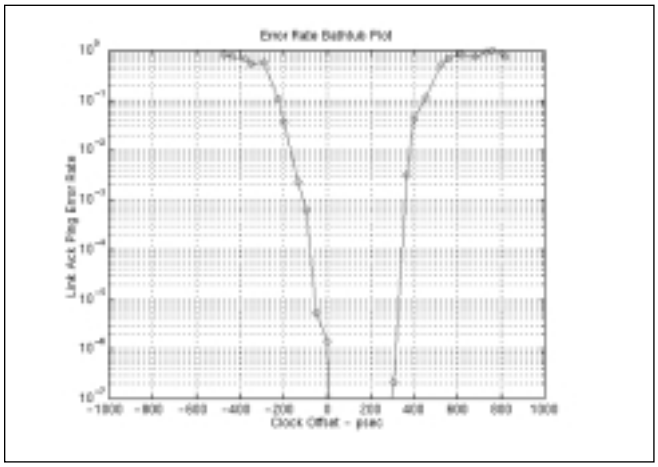


Figure 3 EFW test “bathtub” curve

An oscilloscope was used to characterize the data eye waveforms with the transmitter in unidirectional mode. The advantage of time-domain measurement is its ease of understanding and its coverage of voltage amplitude, signal distortion and time eye closure. The pre-compensation effects can be seen in Figure 4 where the signal will have a larger swing when switching values to compensate for the channel attenuation. In Figure 5, the received signal does not display the different voltage levels at each logic value because the high frequency energy injected by the pre-compensation circuit is attenuated by the interconnect.

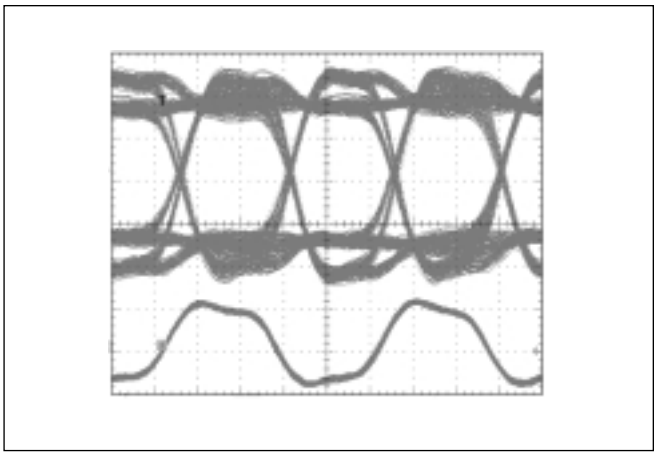


Figure 4 Unidirectional Signal at driver output

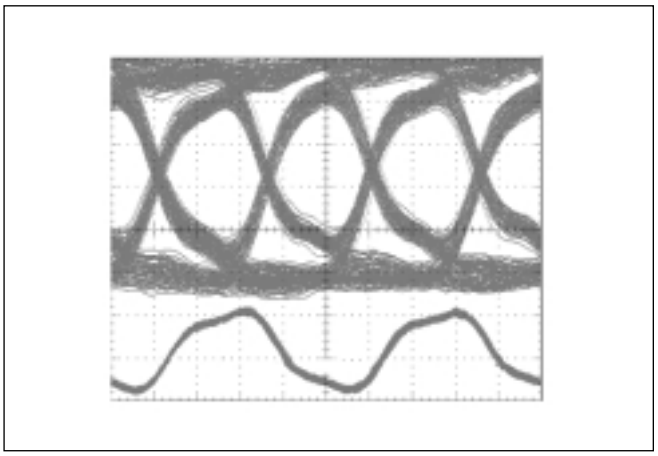


Figure 5 Unidirectional signal at receiver input

In addition, time domain reflectometry (TDR) characterization of channel impedance was used to verify simulation models. Simulation and lab testing of this simultaneous bidirectional interface has shown a BER dependency on the clock phase of the outgoing and incoming data stream. This is due to the noise susceptibility of the driver replica and the summing circuit and, if not carefully taken into consideration when performing simulation, can lead to misleading conclusions. The clock phase effect has been simulated and shown to increase the jitter as much as +/- 80 ps. Due to the asynchronous nature of the multi-system clocks, this jitter would be lower since the phase would vary over time. Simulation can take this effect into account by slightly increasing the period of one transmitter which would then result in a change in phase as the simulation progresses.

Conclusion

A design, modeling and simulation methodology is presented that allows for optimization of the simultaneous bidirectional source synchronous interface operating at data rates of 1.0 - 1.6Gb/s. For system timing closure, the critical design parameters are attenuation induced jitter from inter-symbol interference, impedance mismatch and connector crosstalk. Board and cable crosstalk can be controlled by proper selection of trace spacing and cable construction. The use of source synchronous signaling techniques eliminates the need for a clock recovery circuit and effectively reduces the clock jitter. The usage of on-chip de-skewing circuitry and clock centering to compensate for static skew allows ease of design. EFW tests both clock centering circuitry accuracy and aggregate data setup and hold margins. Finally, an accurate modeling and simulation methodology is essential to guarantee the system timings and hardware functionality.

References

1. J. M. Borkenhagen, R. D. Hoover, K. M. Valk, *EXA Cache/Scalability Controllers*, IBM Enterprise X-Architecture Technology, p.37
2. T. Moe, C. Payton, R. Shearer, S. Willenberg, C.Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31
3. R. Mooney, C. Dike, S. Borkar, *A 900 Mb/s Bidirectional Signaling Scheme*, 1995 IEEE International Solid-State Circuits Conference, p. 38-39, February 1995.
4. J. Broomall, H. Van Deusen, *Extending the Useful Range of Copper Interconnects for High Data Rate Signal Transmission*, 1997 Electronic Components and Technology Conference, p. 196-203, May 1997.

EXA Source Synchronous Memory Design

– Moises Cases, Daniel de Araujo, Dave Guertin, Nam Pham

Introduction

Continued advances in silicon technology have yielded dramatic increases in both circuit speed and wiring density. These advances are shifting the performance challenges from the on-chip circuit-to-circuit interconnections to the off-chip component-to-component interconnections. The goal of high-speed interconnect design is to optimize bandwidth, power, pin count or number of wires and cost. For digital computer systems, the board level interconnect technology is lagging behind the on-chip interconnect technology and the electrical performance of the package, board and cable interconnects are the major limiting factors for high speed and bandwidth. As a result, system level performance is limited by data transfer rates and memory bandwidths, including multi-level caches and memory sub-system. Current processors already operate beyond 1GHz speed leaving a wide gap with the memory subsystem which uses a conventional common-clock technique in the 100MHz range. Innovative signaling schemes such as simultaneous data transmission in two directions over one wire double the effective bandwidth per wire but it is still limited to point-to-point application. A more practical approach is to use a source synchronous signaling scheme to double the data rate without investing in new process technology. The double data rate (DDR) is achieved by a new bus protocol.

Strong PC and server demands for speedier devices have forced leading semiconductor makers worldwide to develop high-performance dynamic random access memory (DRAM) chip technologies which can be used for these new bus protocols such as RAMBUS™ and DDR SDRAM. The initial offerings for DDR registered DIMMs are targeted for the server, workstation, and high-end desktop computer market segments. In addition, microprocessor designs and associated chip sets are fast emerging with similar DDR timing schemes for their system bus, memory hub and I/O hub interfaces. Two different subsystems are used to illustrate the design issues and solutions. A multiple DIMM 100MHz DDR main memory structure is used to illustrate typical multiple load design issues. A simple point-to-point 200MHz DDR L4 cache structure is used to illustrate typical inter symbol interference (ISI) related design issues.

What is Source Synchronous DDR?

In a common-clock timing scheme, a single clock is shared by driving and receiving agents on a bus. Figure 1 depicts a common-clock bus similar to the standard PC100 DIMM memory bus. A complete data transfer requires two clock pulses, one to latch the data into the driving flip-flop and one to latch the data into the receiving flip-flop. The data transfer occurs in the following sequences. First, chipset receives the clock edge (Ck1) from the system clock and makes it available at the buffer output pin. Next, the data is transmitted through a transmission line to the receiving buffer. On the second clock edge (Ck2), the receiver agent latches the data into its flip-flop and makes it available to its internal core. This bus protocol limits itself to lower speeds, shorter trace lengths, sensitivity to component delay, clock skew, and single data capture per clock cycle.

In a source synchronous scheme, a strobe or clock is sent from the driver chip instead of a separate clock source. The data is transmitted to the receiver, and a short time later, a strobe is sent to latch the data into the receiver. The strobe is centered at the data valid window as depicted in Figure 2. This bus protocol removes all the limitations of the common-clock bus, and allows doubling the data transferring rate for the same clock cycle. The bus speed is only a function of the difference in delay (skew) between data and strobe.

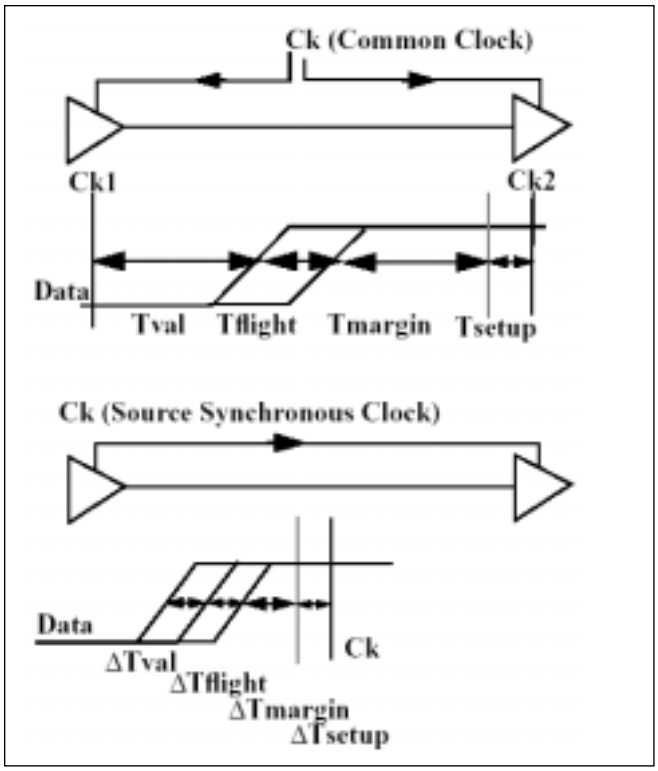


Figure 1 Common and Source Synchronous Clock

Optimization Equation

For the source synchronous scheme, the timing design equation reduces to the minimization of the differential delays (or skews) between the signals and the associated strobe[1]. The basic source synchronous bus timing optimization equations can be calculated using an eye diagram such as Figure 2.

$$T_{vbmin} > T_{DSd} + T_{Sr} + \triangle f_{PCBmax} \quad \textbf{(EQ 1)}$$

$$T_{vamin} > T_{DSd} + T_{Hr} + \triangle f_{PCBmin} \quad \textbf{(EQ 2)}$$

$$(T_{vb} + T_{va}) < 0.5\, T_{cycle} \quad \textbf{(EQ 3)}$$

Where,

T_{vbmin}, T_{vamin} is the minimum time the signal is required to be valid at the receiving components before/after the sampling edge of the strobe.

T_{DSd} is the data to strobe delay skew for the driving component, equivalent to valid time in common clock scheme.

T_{Sr},T_{Hr} is the data to strobe delay skew for the receiving components, equivalent to setup/hold time in common clock scheme.

$\triangle f_{PCBmax}$ and $\triangle f_{PCBmin}$ are the different between data and strobe arrival time.

T_{cycle} is the bus transfer cycle time.

The $\triangle f_{PCB}$ component is a lumped sum effect of many undesirable events in the system. The system propagation delay skew $\triangle f_{PCB}$ is composed of the following elements:

$$\triangle f_{PCB} = (Xtalk + \triangle Z_0 + \triangle length + ISI + T_{vref} + T_{rf})/2 \quad \textbf{(EQ 4)}$$

Where,

Xtalk is the board and connector crosstalk induced delay.

$\triangle Z_0$ is the impedance mismatch effects in multiple board systems.

$\triangle$length is the wiring length tolerances of data lines with respect to the associated strobe.

ISI is the inter symbol interference effects.

T_{vref} is the Vref noise tolerance effects for single ended differential receivers.

T_{rf} is the signal rise/fall delay skew.

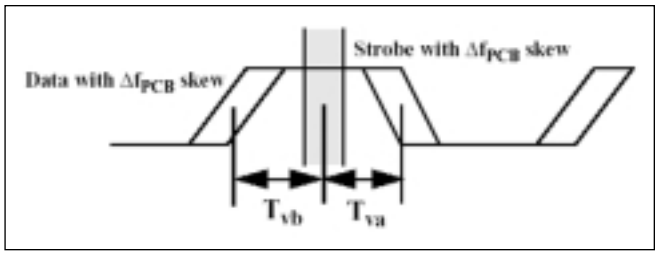


Figure 2 Eye Diagram Timing Allocation

Memory Architecture

The main memory is a 144-bit double width bus running at 100MHz DDR speed. It consists of eight DIMM slots arranged in pairs that can be populated in any order, a pair of DIMMs at a time. The DIMM is the 184-pin, 2.5V, PC200, 72-bit wide, and registered DDR SDRAM. The SDRAM is configured as “x4” bit width with one bank (mono) or two banks (stacked on top of each other) SDRAMs. For 1GB DIMMs, 18 stacked 256Mb SDRAMs mounted on both sides of the DIMM are used. The system memory can be up to 16GB if 512Mb SDRAM DIMMs are used[2].

High performance microprocessors require the use of L4 cache to buffer the processor bus from the slower memory interface and to enhance coherency with other processors in the system. Current microprocessors include a large L3 cache to support a throughput of 3.2GB/s or higher. The system presented here uses 64MB of L4 cache to sufficiently support the system level performance requirement[3]. The L4 bus speed of 200MHz DDR was selected to bridge the processor bus

to the memory 100MHz DDR bus for 3.2GB/s throughput. A typical 4-way multiprocessor system is shown in Figure 3 where five DDR SDRAM of 4Mx32 configuration were used, including parity error bits. The x32 configuration was selected to reduce the number of components and to improve wireability.

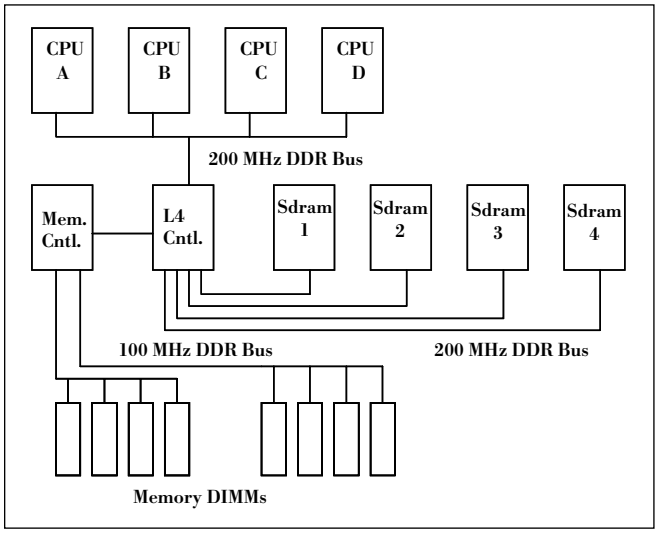


Figure 3 Memory Architecture

Modeling Methodology

A modeling methodology is developed to effectively address all the signal quality design issues for high-frequency source synchronous system designs[4]. This includes reflective noise, crosstalk noise, connector crosstalk effects, overshoot/undershoot voltage, ring-back voltage, settling time, inter-symbol interference, input reference voltage offset, and ground bounce effects. A complete electrical model is constructed for the packaged components, the system boards, the connectors and the add-in cards. Several system topologies are completely modeled to cover a wide range of system-level applications. A set of electrical simulation analyses is then performed, based on a set of simulation and loading conditions, until the noise margin allocations and timing specifications are satisfied. Modeling assumptions are made in order to reduce modeling complexity, and to save on the computer resources needed for simulation. Table 1 shows simulation conditions for typical system-level fast and slow design corners. These simulation conditions include power supply, temperature, and silicon process, and board/card level electrical parameter variation. The fast cases usually address overshoot, under shoot, ring-back and settling time specifications, while the slow cases usually address bus timing and signal slew rate specifications. The simulation cases are further divided into:

1. Read/write cycles from any agent on the net.
2. Impedance mismatching from system board and add-in cards to determine worst-case reflective noise and ISI induced delays.
3. Lightly loaded cases with a single slot populated to determine DC and AC voltage level violations.
4. Heavily loaded cases with fully populated slot configurations to determine worst-case timings and signal slew rates.
5. Wiring length ranges and component's pin placement to determine physical topology options and design guidelines.
6. Cross talk induced delay from connector, board traces and add-in card traces using three-line coupling models.
7. Receiver V_{ref} voltage variation caused by regulator and simultaneous switching noise.
8. Output simultaneous switching noise (SSO) generated by data and strobe switching.

To include all ISI effects, the duration of the preamble is selected to be at least as long as the settling time for each case simulated. System topologies designed for higher clock frequencies require longer preambles. Furthermore, the simulation environment used for this analysis directly includes the effect of reflective noise, and it indirectly includes the effect of cross-talk noise by using effective values for even- and odd-mode characteristic impedance. The effect of input reference offset noise is included as a guard band on the limit of the ring back voltage measurements. Settling time ranges are defined as a percentage of the supply voltage.

Table 1 System Simulation Conditions

Parameter	Slow	Fast
Voltage Sources	Low	High
Temperature	High	Low
Driver/Receiver Models	Slow	Fast
Connector RLC	High	Low
Zo/To (Board+Cards)	Low/High	High/Low

Main Memory Design

Figure 4 shows a typical entry level server implementation using direct attached DDR DIMMs. The riser card consists of eight DIMM slots with the DIMM modules not shown.

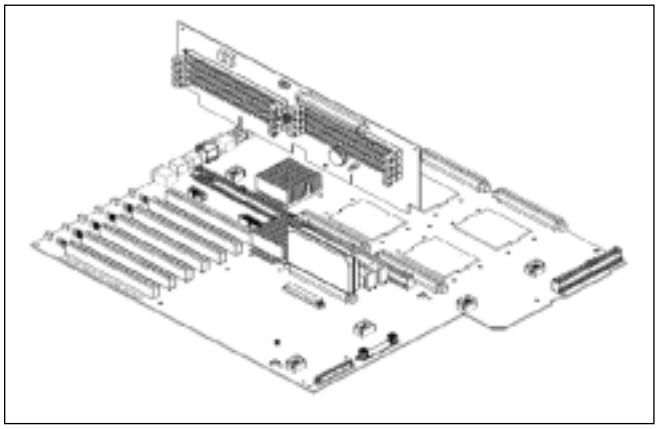


Figure 4 Typical entry level server planar design.

Figure 5 shows a graphical representation of the modeled net topology during a write cycle. Figure 6 summarizes the electrical models for the controller package, the DIMM connector, the riser card connector and the DIMM package. Proper modeling of the DIMM package is essential for accurate simulation results since the inductance of the connector, the DIMM traces and the SDRAM pins greatly affect the magnitude of the reflective noise from the DIMM structure. The controller driver design is optimized for the worst case net topology. In the lightly loaded case, the fast signal slew rate tends to create excessive reflections from the DIMM components, causing a plateau region in the middle of the signal transition. In the heavily loaded case, the slow signal slew rate at the last DIMM location tends to create excessive receiver delay and it is susceptible to V_{ref} variation. These effects reduce the timing margins for the SDRAM components. The driver's effective impedance of 15-20 Ω is selected to satisfy both the slow slew rate and the low impedance requirements.

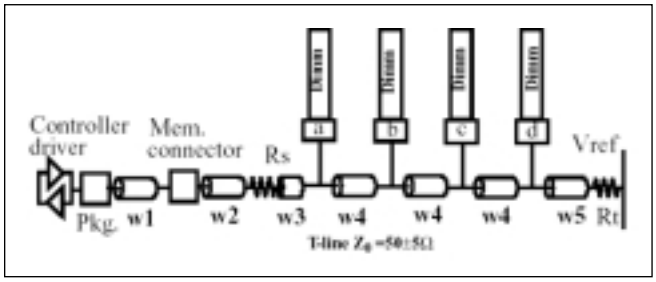


Figure 5 Main Memory Subsystem Topology

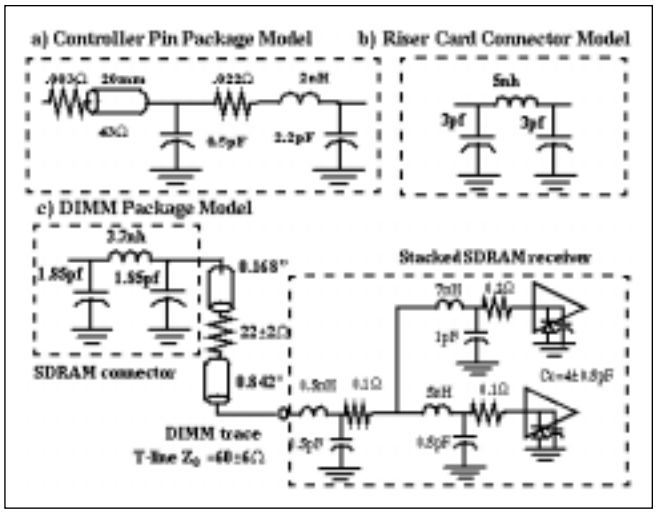


Figure 6 Package and Connector Models

The daisy chain topology of Figure 5 is selected for the data and strobe nets based on a net topology sensitivity analysis. The major concern is a resonant point caused by the DIMM connector and the SDRAM package parasitic, which creates AC signal level violations. A series resistor is placed on the DIMM between these components to dampen the resonance effect. The reflection from the DIMM also creates an ISI effect, forcing the use of a load-end termination technique. A resistor termination R_t connected to Vref was placed near the last DIMM connector to keep the ISI effect within a reasonable range. The reflection also occurs in the read cycle, and it is primarily caused by the impedance mismatch between the lead-in trace and the clustered DIMMs. A series resistor R_s is also inserted between the lead-in trace and the first DIMM connector to reduce the impedance mismatch between the lead-in trace and the trace segments connecting the DIMMs. Both heavy and light loaded cases are simulated to investigate DC and AC level sensitivities to resistor termination values, as well as ISI pattern effects. Cross-talk effects on three-coupled line models and connector effects are analyzed separately to speed up the simulation effort. Figure 7 shows typical simulation results for a write cycle using an eye diagram format for the voltage waveforms at DIMM locations a and d.

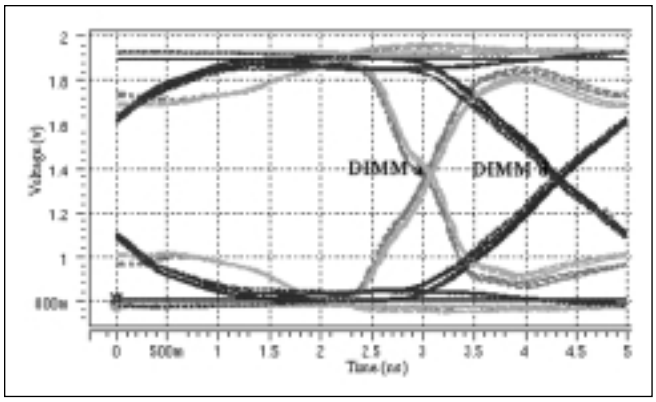


Figure 7 Write Eye Diagram

Table 2 summarizes typical timing allocation for the write and read cycles for the data bus. A similar table for the address and control bus can be constructed. This spreadsheet is used to indicate timing closure for Tva and Tvb, and it is useful to identify the design areas where improvements can be made. The board and card trace impedance is controlled using 50W±10% triplate structures, while the cross talk induced delay adder is minimized using a trace space-to-trace width aspect ratio of greater than three. This corresponds to a coupled voltage level of less than 5% of the aggressor's voltage swing. In order to achieve balanced trace lengths, meander lines can be used to provide fine control of delay skews[5].

Table 2 Summary of Typical Timing Allocation

Parameter	Write(ps)	Read(ps)
0.5 T_{cycle}	5000	5000
T_{Sr} (Controller/SDRAM)	600	650
T_{Hr} (Controller/SDRAM)	600	600
ISI+T_{rf}+$\triangle Z$	360x2	360x2
Xtlk (boards+connectors)	100x2	100x2
Data/Strobe Wire Skew	20x2	20x2
T_{DSd} (Controller/SDRAM)	1000	1600
V_{ref} Tolerance	1340	800
Design Margin	+500	+390

L4 Cache Design

The L4 cache subsystem consisting of three or five SDRAM components where data bus is designed as a point to point 200MHz DDR bus structure, is used to illustrate typical ISI related design issues. Each SDRAM has 32 data bits and an associated strobe. Address and control (Add/Cntl) signals are multiple-drop nets that run at half the speed of the data bus. The 18-bit Add/Cntl signals are wired in a star burst topology. The reflection is controlled by balancing the branches of the star. The Add/Cntl signals are associated to a clock that is a point to point connection.

L4 Timing Allocation

The optimization equations form the basis for creating a timing budget spreadsheet to ensure adequate timing, and to capture opportunity for optimization. Table 3 summarizes a typical timing allocation for data bus write and read cycle. A similar table for the Address/Control bus can be constructed.

Table 3 Summary of Typical Timing Allocation

Parameter	Write)ps)	Read(ps)
0.5 T_{cycle}	2500	2500
T_{Sr} (Controller/SDRAM)	500	400
T_{Hr} (Controller/SDRAM)	500	400
ISI+T_{rf}+$\triangle Z$	250x2	250x2
Xtlk	20x2	20x2
Data/Strobe Wire Skew	20x2	20x2
T_{DSd} (Controller/SDRAM)	600	900
V_{ref} Tolerance	100x2	100x2
Design Margin	+120	+20

The last four entries in Table 3 are the components of the DfPCB in Equation 4 to account for the delay skew effects of ISI, non-symmetrical rise and fall signal transition, the board impedance mismatch, wiring tolerance, crosstalk, and noise on V_{ref} signal.

L4 Cache Design Challenges

Given the typical physical constraints such as trace length and net topology imposed on the system implementation, the electrical subsystem response has a series of resonant points for a given clock frequency. This effect increases the ISI delay skew significantly, especially for classical I/O circuit design. Figure 8 shows a partial eye-diagram of a classical CMOS driver design. The ISI delay skew of 720 psec was observed for a net length of 2.5 inches. It violates the 250 psec requirement in Table3.

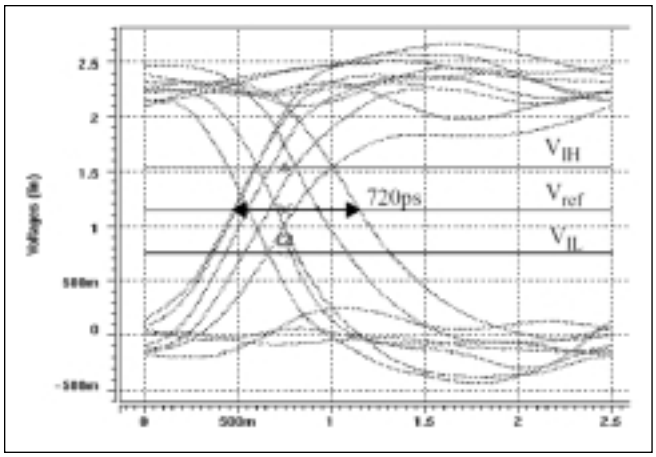


Figure 8 Eye Diagram of Typical Driver Design

The resonant point occurs when the effective electrical length of the signal trace is equal to one half the data capture time. Figure 9 shows typical simulation results for the simple point-to-point non-terminated net where the driver is simply modeled as a time-dependent voltage source in series with a constant resistor to match the characteristic impedance of the trace. The increased ISI induced delay is primarily caused by the parasitic capacitance of the driver's output.

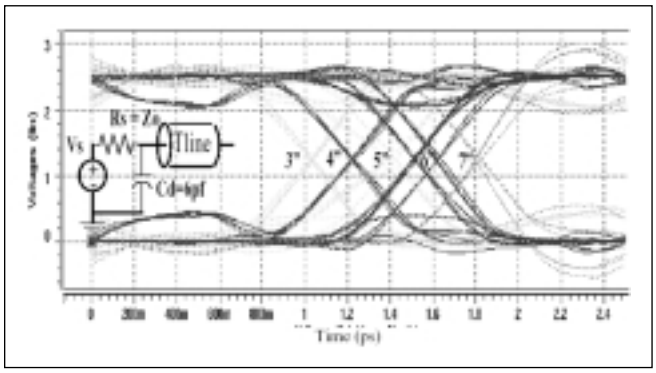
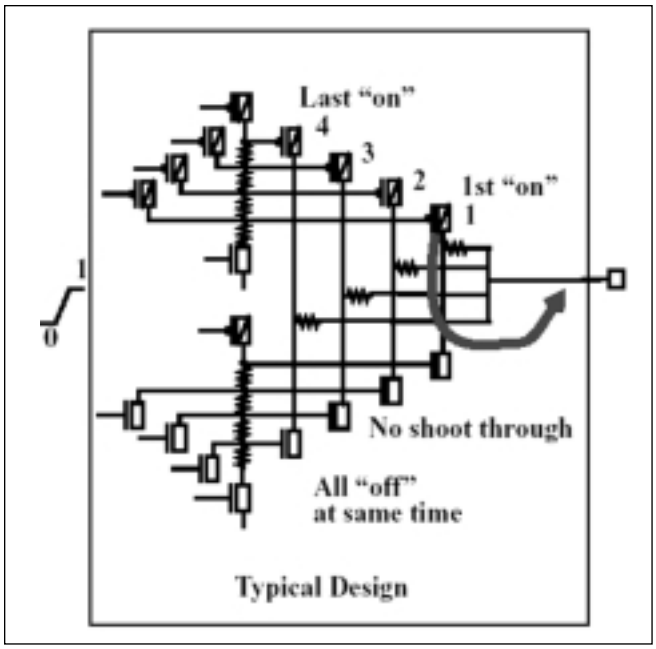


Figure 9 Delay Skew Caused by Parasitic Capacitance

Keeping the net below the resonant length reduces the ISI effect. But, when package and real transistor models are included, the actual PCB trace length is reduced. As an example, with a PQFP and a BGA package on two sides of the net, the PCB trace length reduces from six to three inches. Wiring five SDRAM modules around the controller with net lengths of less than three inches is a challenging task. Increasing the net length longer than the resonant length does not improve the ISI delay skew because the ISI skew curve actually stays constant after maximum ISI has been reached. Therefore attempting to design the net half way between the resonant points is not feasible.

A direct approach to suppress the reflection is to use a split termination scheme. This approach requires a pair of resistors to be placed at each end of the net. Wiring to escape the resistors under the dense module footprint usually requires extra signal layers. Placing the resistor termination on chip provides wire congestion relief but DC power consumption, heat dissipation, silicon cost, and design flexibility may be a concern[6]. Another termination scheme suggests inserting a series resistor in the middle of the net that effectively modifies the resonance and make the ISI skew slightly less on the first few inches. This design also requires precise placement of the resistor in the middle of the net and a strong driver design which is not desirable for SDRAM.

Typical buffer design exhibits both variable source resistance and source parasitic capacitance. The capacitance is a source of ISI jitter but it is generally a fixed number inherent from the CMOS planarization process. The impedance is not a well controlled parameter and changes a lot during the transition time which aggravates the ISI even more. In this system, the driver is designed with stable effective output impedance particularly during the transition time[7]. This can be accomplished by gradually turning off the pull-up (or pull-down) devices while gradually turning on the pull-down (or pull-up) devices, thereby providing continuous current flow through the output stage of the off-chip driver, usually called shoot through current (STC). This basically reduces the net delay skew associated with ISI and it removes the need for both near- and far-end termination. Figure 10 shows the conceptual design of a STC driver where current always has a shoot through path that prevents the driver from ever going into high impedance during transition. Figure 11 shows the ISI induced delay skew of STC and typical push-pull drivers in a typical net setup.



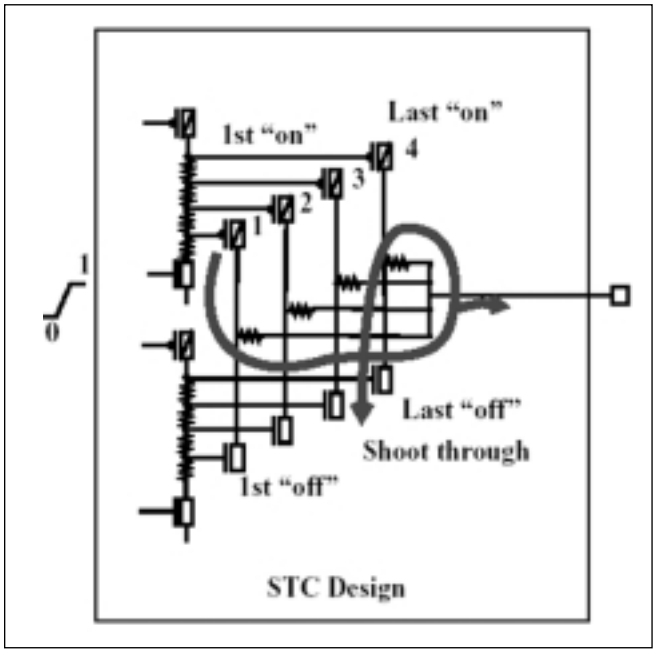


Figure 10 Typical and STC driver design

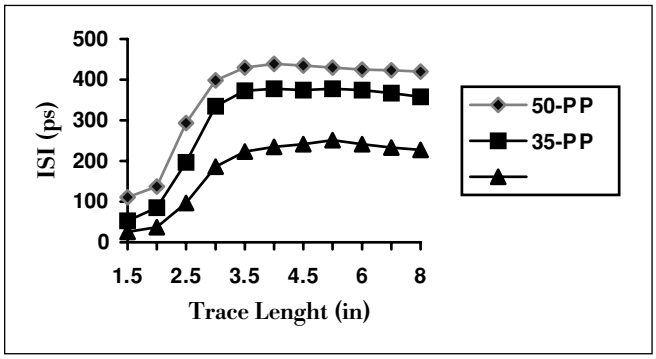


Figure 11 ISI Delay Skew for Various Driver Designs

Conclusion

A design, modeling, and simulation methodology is presented that allows for optimization of the bidirectional source synchronous interface illustrated by point-to-point L4 and multi-drop main memory net. The ISI delay skew is identified as a critical parameter in the system timing allocation budget. Net topology constraints such as total length, balancing trace length within the associated signal group, connector parasitic, and termination tolerance are more critical to the source synchronous bus design. The driver parasitic capacitance has been largely overlooked. It is the main cause of the ISI delay skew for the point-to-point net. For a net with electrical length equal to half the bit time, the reflection speeds up the propagation delay causing delay skew. Design techniques in both driver and net topology are explored to reduce the delay skew. The split-termination scheme reduces ISI skew drastically but it is hard to wire. The STC driver, designed with constant output impedance during the transition, offers the best choice for 400Mb/s source synchronous no-termination net. Finally, an accurate modeling and simulation methodology is essential to guarantee system timings and hardware functionality.

References

1. G. W. Hall, S. H. Hall, J. A. McCall, *High-speed Digital System Design*, 2000 John Wiley & Sons, Inc, p. 190.
2. D. Hurlimann, Y. Ku, H. Schultze, T. Tam, C. Walter, L. Wilson, *xSeries 360, IA-32 Basic Node*, *IBM Enterprise X-Architecture™ Technology* p.71
3. J. Hanna, *IPF Enhanced Node Prototype*, *IBM Enterprise X-Architecture™ Technology* p.85
4. M. Cases, N. Pham, *Design, Modeling and Simulation Methodology for Source Synchronous DDR Memory Subsystems*, Proceedings of 50th ECTC, p. 267-271, May 2000.
5. B. Rubin and B. Singh, *Study of Meander Line Delay in Circuit Boards*, Proceedings of IEEE 8th Topical Meeting on EPEP, p. 193-196, October 1999.
6. H. Muljono, N. Cherukuri, and A. Ilkbahar *High-speed High-bandwidth External Cache Bus with a Center-tapped Termination Scheme*, Hot Interconnects Symposium, August 2000.
7. N. Pham, M. Cases, and D. Guertin, *Design and Packaging Challenges for On-board Cache Subsystem Using Source Synchronous 400 Mb/s Interfaces*, Proceedings of 51th ECTC, p. 123-127, May 29, 2001.

EXA PCI-X Subsystem Design

– Moises Cases, Daniel N. de Araujo, Nam Pham

Introduction

The peripheral component interface (PCI) local bus is a high performance 32-bit or 64-bit bus with multiplexed address and data lines. The bus is intended to be used as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory subsystems. Graphics-oriented operating systems have created a data bottleneck between processors and their peripheral devices. Moving peripheral functions with high bandwidth requirements closer to the processor/memory bus can alleviate this bottleneck. Substantial performance gains are obtained with high bandwidth I/O interfaces (such as GUI, SCSI, LAN, etc.) when a local bus design such as PCI is used. Since the first release of the PCI specifications in 1992, this interface has become ubiquitous in the consumer, workstation and server markets. Other markets such as industrial controls, telecommunications, and high-reliability systems have leveraged the specifications and the wide availability of devices into specialty applications. In 1995 (and later revised in 1998), the PCI specifications were enhanced to accommodate the increased bandwidth requirements of peripheral devices[1]. This allows for peak bandwidth capabilities up to 528 MB/s for 64-bit bus and 66MHz clock frequency. Because of emerging faster I/O technologies (such as Gigabit Ethernet, Ultra 3 SCSI and Fibre Channel), faster system interconnect buses are needed. In 1999, the PCI-X specification was introduced (later revised in 2000) to further increase the I/O bandwidth capability to rates in excess of 1GB/s for 64-bit bus and 133MHz clock frequency[2]. PCI-X provides backward compatibility by allowing devices to operate at conventional PCI modes when installed in conventional systems. Furthermore, PCI-X allows for I/O subsystems with clock frequencies of 66 – 133MHz, and for extended system-loading conditions with hot-plug capability.

The 33MHz PCI I/O bus has become one of the most successful I/O buses ever. Introduced in 1992, we now find PCI everywhere, from laptops to high-end servers. As a result, there is a significant PCI design skill base in the industry. In some respects, designing for PCI-X is actually simpler compared to 33MHz (and 66MHz) conventional PCI, due to the registered nature of the PCI-X definition. However, added design considerations need to be taken into account when creating designs for the higher frequencies of 66MHz, 100MHz, and 133MHz PCI-X.

This paper describes an electrical design, modeling, and simulation methodology for high frequency I/O subsystems[3] utilizing PCI-X in the Enterprise X-Architecture systems. Actual system configurations and timing specifications are used to describe the design methodology, and to optimize the timing equations for the system level interconnects. The timing budget and noise margin allocation for the various components of the optimization equations are discussed in conjunction with their associated critical design parameters including hot-plug requirements.

System Architecture

Architecturally, PCI-X is simply a register-to-register version of the PCI specifications Rev. 2.2. It allows for bus speeds of up to 133MHz while providing backward interoperability with any 3.3V signaling systems or devices. It also provides protocol enhancements for more efficient bus utilization. The PCI-X protocol automatically configures itself to the lowest speed device on the bus.

The following sequence is a conceptual description of a hot-plug insertion case for multiple slot configurations where FET switches are required on the bus signals:

- insert the card in the slot
- determine the capabilities of the card (using M66EN and PCIXCAP)
- power up the card with reset asserted
- hot plug controller requests the bus and receives a grant (the bus is now idle)
- connect the inserted card to the bus with FET switches
- signal the frequency to the inserted card (using DEVSEL#, STOP#, and TRDY#)
- deassert reset and DEVSEL#, STOP#, and TRDY#
- hot plug controller deasserts its bus request and the card is ready to be used

For the single slot case with no FET switches on the bus signals, the basic sequence is as follows:

- hot plug controller requests the bus and receives a grant (the bus is now idle)
- hot plug controller signals the host bridge to ignore bus signals that could change state as a result of inserting and/or powering up the inserted card
- insert the card in the slot
- determine the capabilities of the card (using M66EN and PCIXCAP)
- power up the card with reset asserted (to the card)
- signal the frequency to the inserted card (using DEVSEL#, STOP#, and TRDY#)
- deassert reset and DEVSEL#, STOP#, and TRDY#
- hot plug controller deasserts its bus request
- hot plug controller deasserts signal to host bridge that caused it to ignore bus signals and the card is ready to be used

Design Optimization Equations

For I/O subsystems designed with the common clock signaling scheme, a single clock is used to reference all transactions on the bus. For this technique, the absolute signal propagation delays (flight times) are the limiting factor in system timing calculations, along with clock distribution skew for all agents on the bus. Therefore, the signal integrity and the electrical performance of the package and board interconnects are the major limiting factors to the bus speed. For the common clock scheme, the design optimization problem reduces to the minimization of the signal propagation delay spread across the manufacturing process and the environmental conditions. The basic timing optimization equations are:

$$T_{cycle} > T_{val}(max) + T_{su}(min) + T_{prop}(max) + T_{skew}(max)$$

$$T_h(min) < T_{val}(min) + T_{prop}(min) - T_{skew}(max)$$

Where,

T_{val} is the data valid time,

T_{su} is the data setup time,

T_h is the data hold time,

T_{prop} is the data flight time,

T_{skew} is the clock skew between agents,

T_{cycle} is the bus transfer cycle time.

The system propagation delay, Tprop is the lumped sum effect of many undesirable events in the system. This includes delay components such as the actual board propagation delay, board and connector crosstalk induced delay, board impedance mismatching effects in multiple board systems, reflective noise induced delay caused by electrical discontinuities, inter-symbol interference (ISI) effects, and signal rise/fall delay skew. The ISI is the effect of residual signal settling noise on subsequent transfer cycles.

The system timing budget for the standard PCI-X operating frequencies are shown in Table 1 for the setup and hold timings.

Table 1 Setup and Hold Timing Budget

Parameter	PCI-X 133 MHz	PCI-X 100 MHz	PCI-X 66MHz	PCI 66MHz	PCI 33 MHz	Units
	Setup Timing Budget					
T_{val} (max)	3.8	3.8	3.8	6.0	11.0	ns
T_{prop} (max)	2.0	4.5	9.0	5.0	10.0	ns
T_{skew} (max)	0.5	0.5	0.5	1.0	2.0	ns
T_{su} (min)	1.2	1.2	1.7	3.0	7.0	ns
T_{cycle}	7.5	10.0	15.0	15.0	30.0	ns
	Hold Timing Budget					
T_{val} (min)	0.7	0.7	0.7	2.0	2.0	ns
T_{prop} (min)	0.3	0.3	0.3	0.0	0.0	ns
T_{skew} (max)	0.5	0.5	0.5	1.0	2.0	ns
T_h (min)	0.5	0.5	0.5	0.0	0.0	ns

In addition to system timing specifications, two other sets of electrical specifications have to be satisfied. That is system noise budget and signal integrity specifications. The system noise budget is defined as the difference between the worst case output voltage for the driving agent and the worst case input voltage for the receiving agent. That is, the low noise budget is VIL – VOL and the high noise budget is VOH – VIH. This noise budget is then allocated to the various types of noise in the system such as reflective noise, crosstalk noise, and receiver's input reference offset. The system noise margin allocation for the various sources of noise in the system is shown in Table 2. The signal integrity specifications are defined to guarantee signal quality and to reduce ISI effects. They are overshoot, undershoot and ring-back voltage limits, as well as ring-back settling time limits.

Table 2 PCI-X System Noise Budget

Noise Source	High Noise Budget	Low Noise Budget
Reflective Noise	0.30V_{CC}	0.15V_{CC}
Crosstalk Noise	0.05V_{CC}	0.05V_{CC}
Input Reference Offset	0.05V_{CC}	0.05V_{CC}
Total Noise Budget	0.40V_{CC}	0.25V_{CC}

Electrical Design Considerations

Signal distribution integrity

For high-speed logic circuits, any signal path in the module, card or board can be considered a form of transmission line. These electrical interconnections of components give rise to main lines in parallel with non terminated lines (stubs) or discontinuities modeled as resistor inductor capacitor (RLC) networks. These discontinuities send reflections that create overshoots or undershoots on the main line that could impair the system's performance. Treating each discontinuity as a lumped capacitor is helpful for understanding fundamental dependencies, but for actual design work more accurate modeling is required. Once inductors and resistors are added to the model, the analytical approach is cumbersome and circuit simulation techniques are required. For instance, including inductance and resistance reduces the magnitude of the reflection and the delay on the main line, but it increases the delay of the on-module signal beyond what a lumped capacitance would predict. The goal is to develop design rules that will guarantee the noise margin and timing allocations for all allowable PCI and PCI-X operating modes.

The usage of electrical design guidelines provides proper signal quality to avoid false switching of receivers where appropriate, to assure first incident or first reflected wave switching of receivers at any point on the transmission lines, and to assure the integrity of the timing equation allocations used to predict the overall system performance. As mentioned in the previous section, the signal integrity specifications are defined to assure signal quality and to reduce ISI effects. They are overshoot, undershoot and ring-back voltage limits, as well as ring-back settling time limits. For every change of a signal state, ring-back must not cross the valid input voltage limit when all sources of system noise are included. These parameters are a strong function of the impedance mismatch of the interconnect network, the trace lengths, the signal transition rates, the termination network, and the time-space relationship of the reflective waveforms[4]. Furthermore, these parameters are a function of the system voltage-signaling environment. Analytical techniques based on data from a set of pre-determined simulation runs are used to determine pass/fail conditions for a set of system-level net topologies, and to develop system design guidelines. In general, the procedure followed in the development of the wiring rules is:

1. Classify the logical nets by their characteristics and system design requirements.
2. Determine the electrical characteristics of the drivers and receivers on the net.
3. Determine maximum allowable loaded stub length and load capacitance for each net type.
4. Restrict the line segment lengths, the load capacitance and the number of loads for each net type.

Critical PCI/PCI-X nets are distributively loaded nets. Distributively loaded nets are classified into two categories, depending on the relationship between the electrical length of the line segments and the signal transition time: tightly distributed where the load spacing is electrically less than half the signal transition time, and loosely distributed where the load spacing is electrically greater than half the signal transition time. For all distributed nets, the critical parameters are the load spacing, the load capacitance, the number of loads, and the signal transition time. The signal transition time for the PCI/PCI-X drivers are controlled by the electrical specifications to reduce ringing and simultaneous switching noise. For tightly distributed capacitively loaded lines, the minimum load spacing and the maximum load capacitance are determined by the driver's capability to drive such a loaded transmission line. The characteristic impedance of the line is actually lowered by the load capacitance, while the associated propagation delay is increased. This effective impedance causes an incomplete voltage transition on the line while the driver is switching logical state. This initial voltage level could cause extremely long switching delays for receivers on the line, violating the timing allocation and imparing the system performance. The PCI/PCI-X specifications for the output buffer basically specify that the effective impedance of the driver should be roughly 25 ohms based on a first reflective wave switching assumption for the worst case receiver on the line. For loosely distributed capacitively loaded lines, the noise amplitude is determined by the fastest signal transition time and the maximum load capacitance. For the fastest signal transition, the maximum allowable load capacitance is determined when the amplitude of the primary reflection on the main line is equal to the worst-case threshold voltage for the receivers on the net. For this type of net, an output buffer designed to the PCI/PCI-X specifications will drive the net stronger than if the loads are tightly distributed, thereby providing an opportunity for a first incident wave switching net design.

Furthermore, the increase in bus speeds in today's system designs requires controlled impedance designs. The proper choice of board impedances is essential to properly controlled crosstalk noise, switching noise, noise tolerances, reflections, and bus delays. Proper selection of board stack-up is essential to minimize impedance variations as the result of improper current return paths, plane-to-plane process variations, and adjacent line switching effects. A triplated stripline structure is recommended since it provides the best controlled impedance scenario when the proper power plane is assigned to one of the voltage planes in the structure.

Power distribution integrity

Ideally, power or ground planes on a typical board should be able to source or sink large amounts of current instantaneously. But in reality, this is not the case since these planes have finite non-ideal parasitic values. Any significant amount of current drawn through the planes will have a voltage drop associated with it due primarily to the inductive effect associated with practical board planes. This effect causes the voltage at the component pins to be at a lower potential than at the actual supply output pin. The network combination of the power supply, the voltage regulator modules (VRM), bulk capacitors, high-frequency decoupling capacitors, and the power planes themselves is usually referred to as the power distribution system (PDS). Typically, the power supplies and the voltage regulators have a relatively slow response time and they cannot service instantaneous power requirements for devices switching at relatively high frequencies[5].

The power distribution design considerations outlined in this section are targeted to provide a high quality platform design with optimized signal integrity, timing margins, and power distribution. In general, critical high-frequency interfaces must be routed in a symmetrical stripline stackup to provide clean return paths from component to component through either ground (GND) or appropriate power (VDD) planes. Use of adjacent power or ground plane pairs are recommended to provide clean and equal current return paths, and for additional board decoupling capacitance. If the appropriate power plane is not available as an immediate current return path due to board thickness limitations, additional high-frequency coupling capacitors between the appropriate power planes are needed in the area where these signal traces are located to minimize inductive effects during the switching time, and to reduce electromagnetic interference (EMI) effects. All power planes should be decoupled to ground in such manner as to provide for reasonable management of the switching currents to which the power plane and its supply path are subjected. This is platform dependent and not detailed in this section. It is the board/ system designers' responsibility to ensure that the platform meets all the component specifications. It is strongly recommended that a comprehensive simulation analysis is performed to ensure all component specifications are met. This is particularly important if a design deviates from the recommended design considerations and guidelines.

While executing regular code or changing state, the component's core current experiences a load change that has a ramp shape function with pulses superimposed on it. The pulses occur at multiples of the frequency of operation. If the power distribution system (PDS) is designed such that it has low impedance across similar frequency range, the low noise will be ensured. A typical power distribution circuit schematic for performing frequency domain analyses to determine the appropriate set of board decoupling capacitors for a particular component on the board is shown in Figure 1. The analysis is performed to guarantee power supply tolerance specifications at the component pins. The bulk, ceramic, and inter-plane capacitors are selected in conjunction with package and component models to ensure low effective impedance at the circuit level as shown by the arrows in Figure 1. Electrolytic or tantalum capacitors provide large reservoirs of charge but the parasitic inductance associated with their core and leads limit the maximum frequency at which current can be delivered. Servicing relatively fast switching events is best handled by high-frequency, low-inductance capacitors. These capacitors should be placed close to the device being decoupled to minimize the parasitic resistance and inductance associated with board traces and vias. Inter-plane capacitors between power and ground planes also contribute to the reduction of impedance at high frequency. Therefore, modeling a decoupling capacitor at high frequencies requires accounting for the parasitic inductive and resistive components as well as the bulk capacitor as shown on Figure 1.

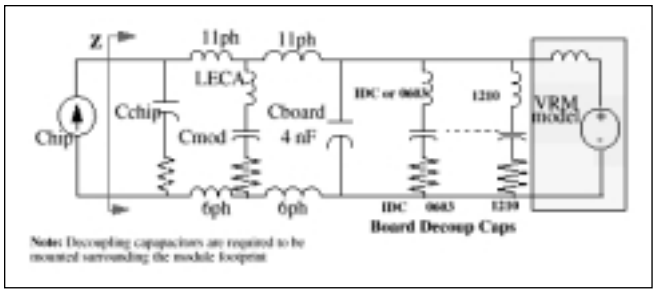


Figure 1 Typical Circuit Schematic for Power Distribution

The general guideline for placing capacitors is to place high-frequency ceramic capacitors as close as possible to the module. Tantalum capacitors can be placed anywhere. To maximize the efficiency of the high-frequency decoupling capacitors, the loop area and loop inductance must be minimized. Placing vias directly at the pad is the best way to attach decoupling capacitors since it is the smallest loop inductance. But for 603 size capacitors, current manufacturing does not allow vias in the pad. Mounting them in parallel of like value (with alternate VDD and GND vias to reduce inductance) on the same side of the card is best (Figure 2). The traces need to be as short and as wide as possible. Otherwise, 603 capacitors can be mounted in opposing pairs of like value power/ground connections (Figure 3). In this case, there could be two vias per pad to reduce inductance. If feasible, make pairs into lines and then pack as close as possible. Capacitor pairs should decouple the same supply, and each capacitor in the pair (or line) should be the same value (to keep currents equal). Top and bottom side pairs may use the same vias. If backside ceramics are to be used, then a pair of power and ground planes should be placed as close to the backside surface as well (the power planes need to cover the module and the backside ceramic footprint area thoroughly). Vias associated with backside ceramic pairs can share those with frontside ceramic pairs.

The board cost increases exponentially as a function of board thickness. To keep the thickness under 100mil, many power planes are partitioned within a layer to supply different power to the components. This requirement creates two problems: Inter-plane capacitance reduction, and signal current return path discontinuity. The inter power plane capacitance can be increased by copper filling all unused signal planes and using generous number of vias to stitch the ground planes together. Signals crossing the splits create power supply noise. The wiring rule is strictly imposed to avoid this crossing problem. Signals running parallel to the split must be 50 mils away from the split. Signals of a bus must be referenced to their bus power supply and ground planes. This triplated structure ensures the fastest and lowest power supply noise. In case of violation, a high number of high-frequency capacitors are used to decouple the wrong reference plane to ground near the component side.

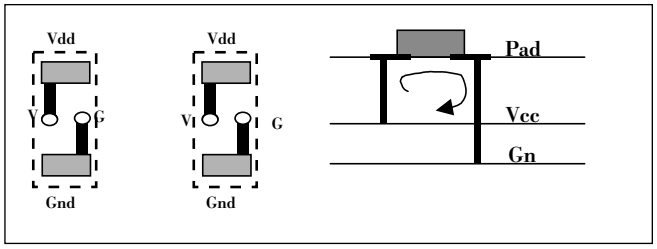


Figure 2 Optimum Capacitor Placement

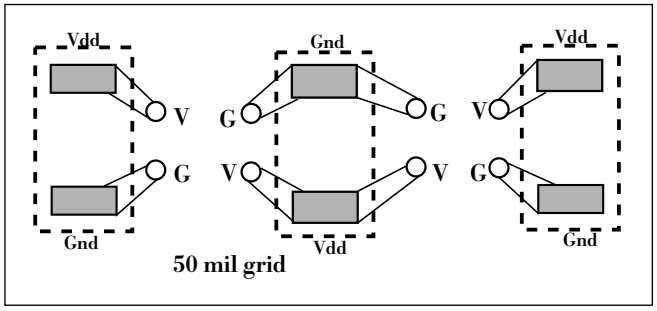


Figure 3 Alternate Capacitor Placement

Connector and add-on card effects

Connectors behave like multiple coupled RLC networks. The number of sections is critical for proper modeling and simulation, and it is a strong function of the frequency range of operation. When properly modeled, it basically reduces the magnitude of the reflection and the delay adder on the main line as compared to a lumped capacitance model, but it increases the propagation delay of the signal path through the connector assembly. Another important design parameter is connector crosstalk. Proper signal to power/ground ratio is required to reduce the crosstalk allocation for connectors on the PCI subsystem to less than 5% of the signal voltage swing. Properly selected signal integrity simulation topologies should include fully coupled connector models. Depending on connector pin arrangement, extra ground pins are carefully assigned to shield signal crosstalk. Usually, a rule of one ground pin for every four signals is used. A signal must be referenced to its power supply and ground planes on both sides of the connectors. In case of violation, the wrong power plane is

capacitor decoupled at both the component and connector end of the plane, and the capacitors are placed within 0.5" of the signal.

An add-on card behaves like a loaded stub on a main line and it should be modeled properly. The maximum allowable stub length is determined by the fastest signal transition and the maximum required load capacitance (as specified by the PCI/PCI-X specifications) when the oscillatory reflections on the line are minimized. From time domain considerations, an unterminated line could cause severe reflections on the main line and on the stub itself, yielding the condition known as “ringing.” This implies that the input impedance of the stub is inductive in the range of frequencies where most of the energy content of the signal lies, causing severe oscillations on the main line. Using a circuit simulation program and time-domain analysis techniques, the maximum amplitude of the reflections on the main line and the corresponding decaying time of reflections on the main line can be determined for various stub lengths and capacitive loads. It is found that if the electrical length of the stub is less than or equal to one-quarter of the signal transition, the reflection from the stub’s equivalent capacitance reasonably matches the maximum reflection from the actual loaded stub over a wide range of load capacitance values. In addition, the decaying time of the reflections is considerably reduced for this condition, approaching the decaying time of the stub’s equivalent lumped capacitance value.

The effect of removing the FET switch devices is to somewhat reduce the maximum allowable lead-in trace length (i.e., host controller to first slot). The effect of the FET switch is basically to reduce the reflective signal voltage on the main line, but it introduces some signal attenuation due to the effective series resistance of the FET switch devices. The effective series resistance of the FET switch devices should be kept below five ohms. Using single-input/ multiple-output controlled FET structure devices tends to increase the maximum allowable total net length since it effectively reduces the magnitude of the reflective noise on the main line. The use of a single-input/multiple-output controlled FET basically removes the trace length between the switch devices. An example for a 2-slot structure with FET switches is shown in Figure 4.

The power island associated with each slot isolates the hot-plug slot power from the main power. Critical signals are not allowed to cross the power plane split interface, which restricts wiring under the power island area. Decoupling capacitors to ground planes are needed in the area where the split plane occurs, but decoupling capacitors between islands are not recommended.

Electrical Modeling Methodology

A modeling methodology is developed to effectively address all the signal quality design issues for high-frequency common-clock system designs. This includes reflective noise, crosstalk noise, connector crosstalk effects, overshoot/undershoot voltage, ring-back voltage, settling time, inter-symbol interference, input reference voltage offset, and ground bounce effects. A complete electrical model is constructed for the packaged components, the system board, and the add-in cards (including hot-plug switches and add-in card connectors). Several system topologies are completely modeled to cover a wide range of system-level applications. This includes topologies from a single-connection host bridge with a single add-in card slot to a single-connection host bridge with multiple add-in card slots (up to four slots). These configurations were analyzed with and without hot-plug switches, and clock frequencies of 66, 100 and 133MHz depending on system configuration and loading conditions. A set of electrical simulation analysis is then performed based on a set of simulation and loading conditions until the noise margin allocations and timing specifications are satisfied. Modeling assumptions are made in order to reduce modeling complexity, and to save on the computer resources needed for simulation. Table 3 shows simulation conditions for typical system-level fast and slow design corners. These simulation conditions include power supply, temperature, and silicon process, as well as board/card level electrical parameter variation with process and temperature. The fast cases usually address overshoot, undershoot, ring-back and settling time specifications, while the slow cases usually address bus timing and signal slew rate specifications. The simulation cases are further divided into:

1. Read/write cycles between any agents on the net.
2. Impedance mismatching from system board and add-in cards to determine worst-case reflective noise and ISI induced delays.

3. Lightly loaded cases with a single slot populated to determine DC and AC voltage level violations.
4. Heavily loaded cases with fully populated slot configurations to determine worst-case timings and signal slew rates.
5. Wiring length ranges and component's pin placement to determine physical topology options and design guidelines.
6. Crosstalk induced delay from connector, board traces and add-in card traces using three-line coupling models.

Table 3 System Simulation Conditions

Parameter	Slow (w/c)	Fast (b/c)
Voltage Sources	Low	High
Temperature	High	Low
Driver/Receiver Models	Slow	Fast
Connector RLC	High	Low
Zo/To (Board+Cards)	Low/High	High/Low

Figure 4 is a circuit schematic representation of the selected topology for a two-slot system configuration with hot-plug switches. Similar topologies are used for other system configurations. The daisy chain topology is selected based on a net topology sensitivity analysis. To include all ISI effects, the duration of the preamble is selected to be at least as long as the settling time for each case simulated. System topologies designed for higher clock frequencies require longer preambles. A signal test pattern consisting of every possible combination of the 5-bit preamble is used for this analysis. Furthermore, the simulation environment used for this analysis directly includes the effect of reflective noise, and it indirectly includes the effect of crosstalk by using effective values for even- and odd-mode characteristic impedance. The effect of input reference offset noise is included as a guard band (5% of the supply voltage) on the limit of the ring back voltage measurements. Settling time ranges are also defined as 5% of the supply voltage. The minimum device spacing in the system configurations analyzed assures minimum Tprop.

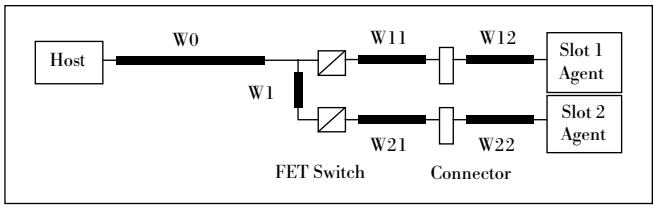


Figure 4 Circuit Schematic for a 2-slot Configuration

The goal of the simulation methodology is to automatically generate all the required simulation test cases and associated net lists to step through all selected wire length possibilities. This includes wire trace impedance and velocity ranges for all traces on the net, power supply, temperature and process variations for all components on the net, and impedance mismatch effects. The graphical simulation results are transformed into a text matrix format containing the specific simulation case, the overshoot/undershoot voltage, the ring-back voltage, the settling time and the net propagation time. All measurements are automatically extracted from the corresponding voltage waveforms in accordance to the PCI-X specification[2]. The resulting database is then compacted, sorted, and analyzed using standard database software algorithms. Object-oriented scripts are used to properly generate all the netlists required for the circuit simulator programs. Data consistency is guaranteed through the use of a naming convention associated with the data files, data directories, agents, circuit nodes, simulation conditions, and symbolic links[6].

During the simulation and extraction process, some files and directories are made permanent, while others are made temporary in order to reduce the computer resource requirements and speed-up the data preparation and simulation process. Examples of permanent files are control and model files required by the simulator programs. Examples of temporary files are signal, topology and parameter statement files needed for constructing all the required simulation test cases. A hierarchy directory is then automatically generated for each system configuration to assure data consistency and integrity. This directory is used to generate all the appropriate symbolic links to the corresponding data files. A top-level script is then developed to launch all the required simulations for a user-specified system topology and/or a specific test case. An example of the data distribution for the extracted measurement parameter, Tprop, in a two-slot system configuration and 100MHz clock frequency is shown in Figure 5.

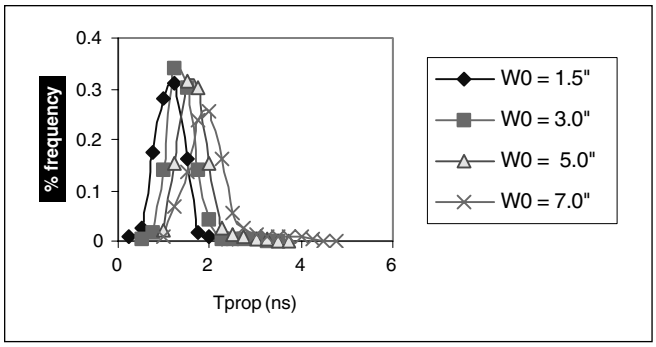


Figure 5 Tprop Distribution for a 2-slot Configuration

Conclusion

A design, modeling and simulation methodology is presented that allows for optimization of the common-clock timing equations in practical I/O subsystem environments using PCI/PCI-X interface. For system timing closure, the critical design parameters are the host-bridge lead-in trace length, the connector spacing, and the stub length in the add-in cards. An accurate modeling and simulation methodology is essential to guarantee hardware functionality primarily due to the sensitivity of the system timing parameters to reflective noise, intersymbol interference, signal coupling, and impedance mismatch effects.

Acknowledgments

The authors would like to thank Dan Neal, Daniel Ortmann and John Mitby from the IBM Server Group for their valuable contributions to this work.

References

1. PCI Local Bus Specifications, Rev. 2.2, December 1998.
2. PCI-X Addendum to the PCI Local Bus Specifications, Rev. 1.0a, July 2000.
3. T. Moe, C. Payton, R. Shearer, S. Willenberg, C.Wollbrink, *EXA I/O Bridge*, IBM Enterprise X-Architecture Technology, p.31
4. J. Lee, E. Shragowitz, *Overshoot and Undershoot Control for Transmission Line Interconnects*, Proceedings of the 1999 Electronic Components and Technology Conference, p. 879 - 884, June 1999.
5. T. Westerhoff, *Optimize Power Distribution Analysis in High-Speed System Designs*, p. 123 – 134, Electronic Design, November 2000.
6. M. Cases, N. Pham, *Design, Modeling and Simulation Methodology for PCI-X Subsystems*, Proceedings of the IEEE 9th Topical Meeting on Electrical Performance of Electronic Packaging, p. 33 - 36, October 2000.

Partitioning in the EXA Technology

– Jim Bozek

Introduction

It is the mission of an enterprise computing center to provide dependable, efficient, production-level services for its users. To accomplish this, such installations must provide services that offer:

- Reliability
- Availability
- Performance

In addition, to meet the demands of a growing user community and changing support models, computing resources must be re-configured, upgraded, expanded, deployed, and replaced. Sustaining these activities results in further demands:

- Scalability
- Serviceability
- Manageability

To maintain a competitive edge, computer vendors endeavor to meet these objectives while delivering products that are cost effective. At the same time, increasing performance and decreasing cost has strongly impacted competition in the Standard High Volume (SHV) server market, while SHV server offerings encroach upon a market traditionally occupied by mainframe products.

One way in which IBM has attempted to realize customer demands and maximize competitive features is to design, build, and deliver computer systems based upon a scalable Hardware (HW) platform using low-cost commodity parts. An extension of this theme is to provide Software (SW) support that exploits the re-configurable – or partitionable – feature of this platform.

It is the intention of this paper to show how the IBM Enterprise X-Architecture (EXA) addresses the requirements of an enterprise computing environment as stated above. Specifically, it describes the multi-node EXA platform HW and shows how it differs from conventional single-node SHV solutions in terms of its scalability. In addition, the EXA partitioning model and an overview of partition configuration and management SW are presented.

Background

Shared Memory Architectures

By far, at the current time, the most prevalent systems in the SHV server market are based upon *Symmetric Multiprocessor* (SMP) architecture. The term Symmetric refers to the fact that all CPUs have equal access to shared memory in terms of latency. SMP architectures are also referred to as *Uniform Memory Architectures* (UMA). Current SMP architectures employ a memory hierarchy that, in addition to main memory, is composed of a structured set of cooperating memory buffers, or caches, to reduce effective memory access latencies and bus bandwidth requirements. Hence, these systems are referred to as *Cache-Coherent Uniform Memory Architectures* (CC-UMA).

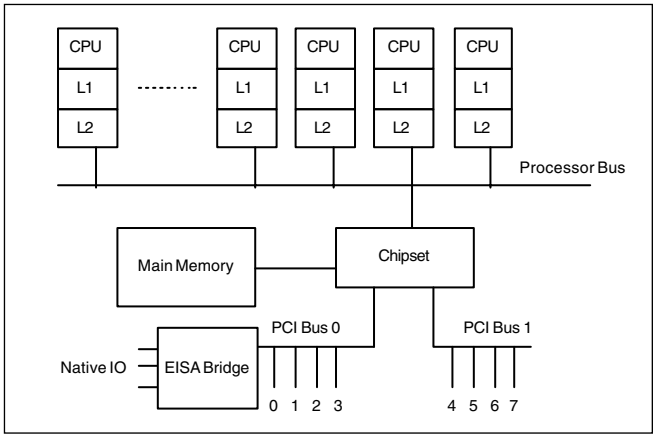


Figure 1 CC-UMA Platform

A simple CC-UMA architecture is illustrated in Figure 1. The principal components are: the Processor Bus, which includes logical Interrupt and Cache Busses; the CPUs; the chipset, which includes memory controller and IO expansion (e.g., PCI) Host Bridge; system memory; and, EISA Bridge, which supports connection of Native IO devices such as serial/parallel, mouse, and keyboard ports. Typically, such systems are designed and built with commodity components according to standard specifications such as Server Design Guide[1], MPS[2], ACPI[3] and EFI[4].

The CPUs, system bus, the chipset, and the EISA Bridge are often of an Intel design, though CPU, chipset and

host bridges are designed and manufactured by other vendors (e.g., IBM, Motorola, MIPS, DEC, SUN, etc.).

The primary advantages of the SMP servers are the relatively low cost, due to the use of commodity parts and standards, software availability[5, 6], standard device availability[7, 8, 9, 10], and the widespread expertise within the user community.

A prominent disadvantage of the SMP architecture is its inability to scale beyond a relatively small number of processors due to bus bandwidth limitations. The *Cache-Coherent Non-Uniform Memory Architecture* (CC-NUMA) was developed in an effort to address this problem.

An example of a CC-NUMA implementation is shown in Figure 2. In this design, the CC-NUMA system is composed of multiple SMP building blocks or nodes. The nodes are connected through a coherency agent that interfaces to the FSB of the node on one side and provides a standardized memory interconnection on the other side. This structure allows memory physically resident on each node to be shared globally, enabling access to memory on one node by processors or devices on another node using HW support.

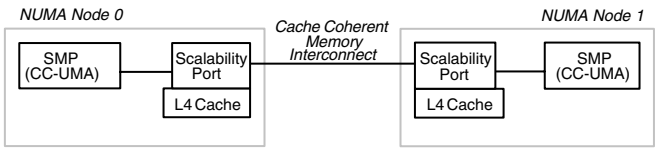


Figure 2 2-node CC-NUMA Platform

Some CC-NUMA implementations extend the memory hierarchy to include a Level 3 (L3) or, where an L3 cache is local to each CPU, a Level 4 (L4) cache. Where extended caches are present (referred to here as an L4 cache), a local L4 cache facilitates Intra-Node caching. In conjunction with the L1, L2, and L3 caches of the processors on the local processor bus, the local L4 cache is intended to boost performance among the CPUs on the local node by mitigating overall access time to local memory. *Remote* cache performs essentially the same function in the memory hierarchy, but in relation to Inter-Node memory accesses, or accesses to memory physically resident on another node. It is widely accepted that adding the L4 cache results in a performance gain in multi-processor platforms, though not without associated complexity and cost. Figure 2 shows a system that includes a remote L4 cache.

Primary advantages of the CC-NUMA over conventional mainframes are the relatively low cost, due to the use of commodity parts and standards, software availability, standard device availability, and the widespread expertise within the user community. An advantage of CC-NUMA over SMP is the potential to scale the system by adding more nodes, thereby enabling expansion beyond the conventional limit (e.g., tens of processors). For additional information, refer to[11, 12].

Partitioning

In this context, *Partitioning* is the process of combining (composing), separating (decomposing), or re-configuring platform HW resources into integral computing units. A fundamental goal of partitioning is to provide the computing center with the flexibility to control the manner in which system resources are distributed, utilized, serviced, protected, and managed.

Partitioning is discussed as being *Logical* or *Physical*, and is further categorized as being *Fixed*, *Static*, or *Dynamic*. The ability to segment and reconfigure multiprocessor systems has been available for several years in the mainframe market. With these systems, a *Logical Partitioning* model has been employed to enable the customer to apply discrete system resources (e.g., physical memory, processors, IO channels) to tasks as needed[13, 14, 15]. With the advent of commercially viable multi-node systems based upon a high-speed interconnection of nodes built from self-contained, fully functional SMP building blocks, the ability to provide cost effective *Physical Partitioning* has become practicable.

Whereas *Logical Partitioning* (LPAR) allows re-configuration of individual system resources on a hardware platform specifically built to support it, *Physical Partitioning* (PPAR) enables reconfiguration of multi-node systems, or *complexes*, along boundaries based upon building blocks referred to as nodes. LPAR enables a fine-grained approach to partitioning, but requires more complicated hardware and operating system features. In contrast, PPAR requires less complicated hardware and can operate under current commodity operating systems (e.g., Windows, Linux, or Netware) – it is therefore less expensive.

PPAR is realized using the hardware technology specifically designed to implement it. Figure 3 illustrates the key structural components related to PPAR. As shown, a Node is a physical aggregation of packaged system

resources and the smallest manageable unit of physical partitioning. A single standalone node is capable of sustaining one and only one partition. A *Complex* is a formal and physical aggregation of multiple nodes – the nodes in a multi-node complex are interconnected through a *Scalability Link*. Each scalability link can be programmed to support three modes of operation: coherent, inter partition communication (IPC), or isolated. An example of this is shown between selected nodes in figure 3. A *Partition* is a formal organization consisting of a single node (or multiple physical nodes interconnected through a scalability link into a CC-NUMA configuration) capable of executing a single image of an operating system. Partitions may be comprised of nodes in the same complex.

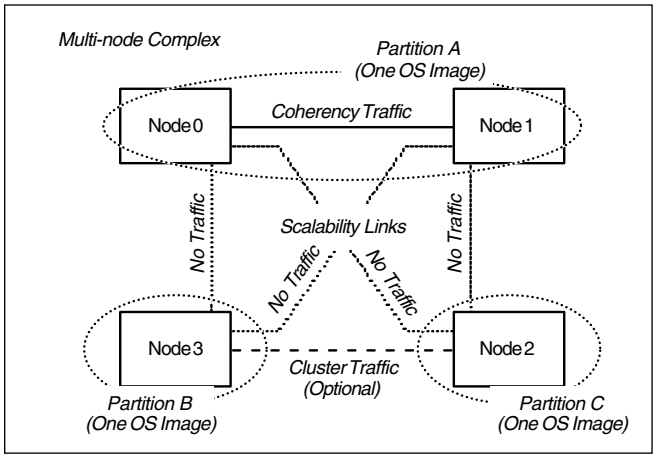


Figure 3 Structural Components of Physical Partitioning

Partitionable platforms based upon EXA are integrated into an enterprise computing environment as illustrated in Figure 4.

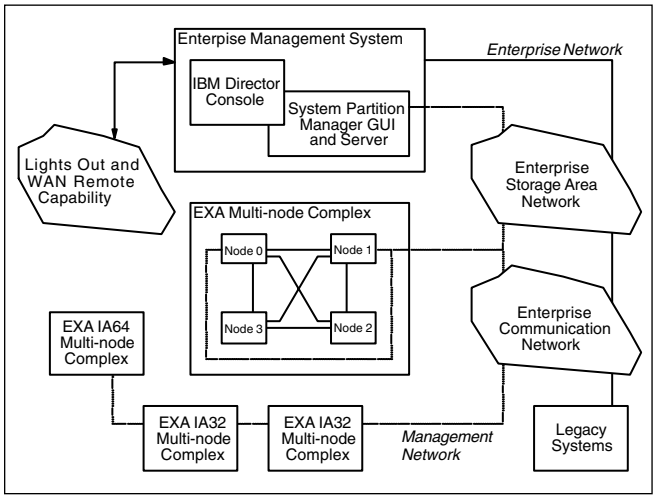


Figure 4 Idealized Enterprise Computing Environment

As defined here, *Physical Partitioning* is divided into three types:

- Fixed
- Static
- Dynamic

Fixed Partitioning (FPAR) is a partitioning model that enables re-configuration of a multi-node complex along nodal boundaries. Typically, FPAR requires shutdown and power-down of all partitions, physical re-cabling, power-up, and reboot of an OS image on each partition. In terms of platform support, this is the least complicated, but it is the most time consuming and least flexible of the three types of partitioning. By and large, FPAR facilitates manufacturing and delivery of scalable multi-node systems. Though it represents a more efficient and economical upgrade path to the customer, FPAR does not address the regular, day to day workload management needs at customer sites outside those supported by an operating system executing on each partition. One approach to addressing these needs is through Static Partitioning.

Static Partitioning (SPAR) is a model of partitioning that enables reconfiguration of a multi-node complex after (possible) shutdown and restart of an operating system executing on the effected partitions and nodes. This provides the customer with the ability to reconfigure an EXA platform without forcing him to interrupt usage on the entire complex. A similarity between FPAR and SPAR is that the platform must be reconfigured while the OS is not running (i.e., prior to booting the OS). The key difference between FPAR and SPAR is that the latter offers the ability to independently configure and service individual partitions (and nodes) through software, without necessarily having to shutdown the OS, physically power-down and power-up HW, and restart the OS on unaffected partitions. SPAR is considered a requirement for commercially competitive multi-node products of today.

Dynamic Partitioning (DPAR) is a type of partitioning that supports *On-Line Insertion* (OLI) or *On-Line Removal* (OLR) of nodes within a multi-node complex. Similar to SPAR, DPAR enables the ability to independently manage and service individual partitions through software, without having to shutdown, physically power-down, power-up, and restart unaffected partitions. The key advantage of DPAR over SPAR is the ability to

reconfigure the complex while a single OS, or multiple operating systems on multiple partitions are executing, thus maintaining availability during service, maintenance, upgrade, or SW migration.

The EXA Platform Architecture

An EXA platform is constructed from an approach based upon CC-NUMA. EXA employs two key features that extend the CC-NUMA traits described earlier:

- L4 caching scheme that supports both local or remote operation
- programmable memory interconnect

Exploiting the local or remote caching feature enables an EXA platform to provide competitive performance in both single-node and multi-node configurations. The programmable interconnect enables the ability to configure the EXA platform as a four through 16 CPU partition.

A four CPU IA32 EXA node is illustrated in Figure 5. The illustrated node, based upon an IBM XA-32™ chipset, includes a Memory/IO controller (MIOC) and the PCI-X I/O Bridge (PCI-X IOB), and a cache/scalability Controller 32 (CSC32). On the IA32 platform, the CSC32 chip provides the programmable memory-interconnect (in the form of SMP Expansion Ports) and supports local or remote use of the XceL4™ Server Accelerator Cache. The primary differences between the 32-bit and the 64-bit EXA platforms are the processors (IA32 vs. IPF) and the scalability chips (CSC32 vs. CSC64), respectively. Though an EXA node can contain up to 32 DIMMS, the example in the figure illustrates a 16 DIMM implementation.

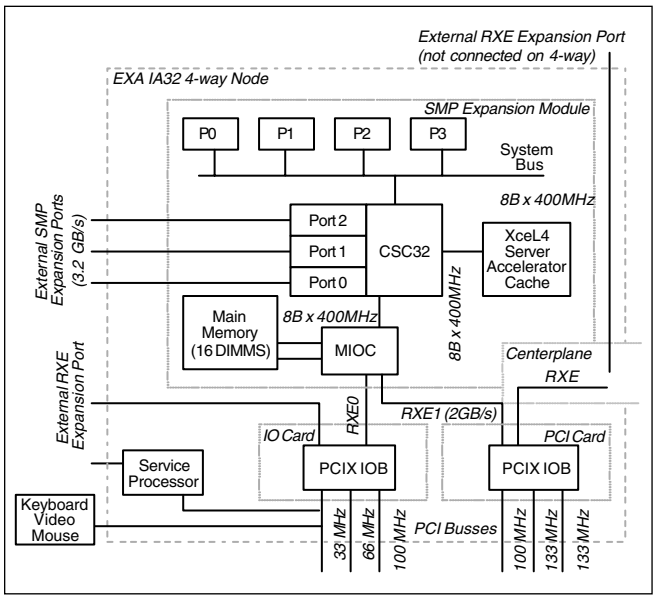


Figure 5 An EXA Node

Included in each node is an independently powered service processor that provides an interface to a management network. The service processor is a small, self-contained computer and real time operating system (RTOS) that provides control, diagnostic, and management functions prior to power-on of the host node and during runtime.

With respect to a multi-node system containing more than two nodes, an independent scalability link exists between each node and each other node (as illustrated in Figure 3). Using a point-to-point direct connection, multiple scalability ports are employed to support a system containing from one to four nodes.

The SMP Expansion Ports, which provide the end points of a scalability link, are configured to support the same mode (e.g., Isolated, IPC, or Coherent) and type of operation (e.g., programmed IO, remote DMA, shared memory). This is evident in the context of a 2-node system. However, in the context of multi-node system where each node is directly connected to each other node, each node has an independent relationship with each other node. A physical connection is used to join the two nodes similar to that shown in Figure 2. With EXA, the SMP Expansion ports connecting the pair of nodes are programmable through the System Bus on each node and configured to support one of three modes of operation:

- Isolation Mode
- IPC Mode
- Coherency Mode

Isolation Mode

In *Isolation Mode*, a scalability port is prevented from interaction with the neighboring node at each endpoint of the link. This is accomplished by placing the scalability port of each node in a state that inhibits signals from passing through the scalability link. In this way, activities on a neighboring node are physically prevented from effecting the operation of the local node and accessing its resources through the link.

Isolation Mode is used to prevent race conditions, timing, and electrical problems during the initialization and boot process. It is also used to prevent undesirable effects due to error or malicious intent and to provide logical independence when the platform is partitioned and where security or administrative policies dictate such operation. Isolation mode is the default behavior

of an EXA node when power is applied. This is illustrated by the connection between Node 0 and Node 3 (or Node 1 and Node 2) in Figure 3.

IPC Mode

In IPC Mode, the scalability link between two adjacent nodes provides for a low-latency, message-based communication mechanism between the interconnected nodes. This mechanism may support more than one type of data transfer (e.g., messages, remote DMA, shared memory). In IPC mode, caching attributes of the memory mapped or shared memory region are defined as required by the OS and the system SW supporting IPC. IPC connectivity is illustrated between Node 2 and Node 3 in Figure 3.

Coherency Mode

In Coherency Mode, the L4 cache is incorporated with scalability port features to form a fully cache-coherent memory interconnection between adjacent nodes connected through the scalability link. In this mode, multiple nodes are integrated into a single set of resources and a single global address space capable of executing a single image of an operating system. A coherent configuration is illustrated between Node 0 and Node 1 in Figure 3.

Partitioning a Multi-Node EXA Platform

A primary difference between a *non-partitionable* and a *physically partitionable* platform is that the design and manufacturing process determines the configuration and scalability of a *non-partitionable* platform, whereas a *partitionable* platform enables the customer to determine configuration and scalability at the time of need. For example, once delivered to the customer, a non-partitionable system cannot be reconfigured to add more or less processors, memory, or IO buses than the platform was designed and manufactured to support. A physically partitionable platform is extensible and scalable – it enables the customer to determine how the system will be grown and configured according to need after delivery of the product.

The methods used to configure and manage partitions on an EXA platform are:

- Local Partition Configuration
- Remote Partition Management

A key difference between *Local Partition Configuration*, also referred to as Manual Partitioning and *Remote Partition Management* is that local partition configuration is used to partition only a single local complex (i.e., those nodes that are interconnected through a scalability link) by visiting each (local) node within the complex. In contrast, Remote Partition Management allows management of multiple complexes and partitions without being in close physical proximity to platforms under management. Both are performed prior to executing the OS on the platform when supporting static partitioning.

Local Partition Configuration

Local partition configuration, illustrated in Figure 6, makes use of the BIOS setup menu on each node. This method requires no other equipment other than the components present in a standard configuration (e.g., keyboard, video, mouse). Service and support is the primary application of local partition configuration.

Figure 6 Local Partition Configuration Using BIOS Setup Menu

Remote Partition Management

In contrast to local partition configuration, remote partition management requires an additional external system connected to the management network. Employing a management tool with a graphic user interface (GUI) running under an OS executing on the external system (as shown at the top of Figure 4), a user can manage multiple complexes connected to the management network.

Using the console, the user employs an interface that offers an abstraction different than that provided by the local partition configuration methods. In this case, the user specifies or creates a complex or a partition, manages nodes, associates them with partitions, activates or deactivates partitions, and monitors events with a networked user interface from a remote location. In

addition, the console may be located at a geographically distant physical location from the actual platform being partitioned. An example of the GUI is shown in Figure 7.

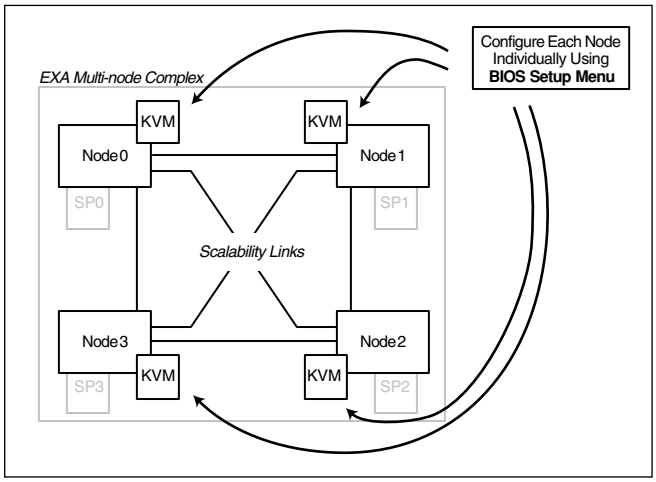


Figure 7 Remote Partition Management Console

In support of Static Partitioning (SPAR), the service processor on each node, communicating with the GUI through the IBM Director infrastructure and the out of band (OOB) management network, stores partition information into nonvolatile storage on each node. Once the configuration is defined, the user activates the partition, and the boot process on each node makes use of the locally stored partition information.

Consistent with a Static Partitioning (SPAR) model, if an OS is running on any node in the partition, that node must be shutdown in order to activate a newly defined partition. As discussed earlier, this is a primary difference between *Dynamic Partitioning* (DPAR) and *SPAR*: DPAR allows reconfiguration without shutting down the OS and while the platform remains available for use. Because of this, DPAR requires significant support from a DPAR enabled operating system, while FPAR and SPAR typically require little or no specific functional changes to an operating system[1]. When an OS is enabled to support On-Line Insertion (OLI) and On-Line Removal (OLR) of nodes, the same general user methodology is used to re-partition the platform while the OS is executing.

An overview of remote management architecture is illustrated in Figure 8. The shaded rectangle at the top of the figure represents an IBM Director management agent executing on an EXA partition using OS services while the OS is executing. The Graphic User Interface, based upon an extension to the IBM Director Console, is pictured at the bottom right. The IBM Director Server, along with a management extension, is pictured at the bottom left. The GUI and Server can execute on separate machines or the same machine, or they can execute on the platform itself.

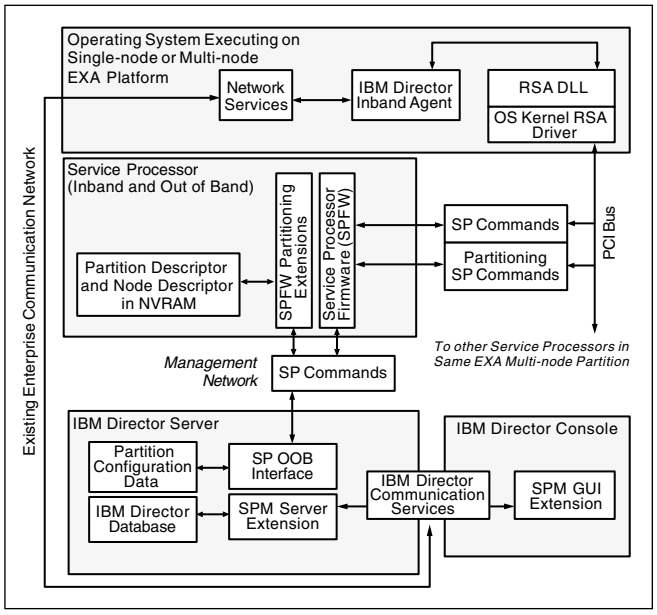


Figure 8 System Partition Manager

The shaded rectangle at the center of the figure represents the service processor (SP) that is contained within each EXA node. The IBM Server extension executes on a machine with a Network Interface Card (NIC) interface to the management network in order to manage the EXA platform prior to power on or OS boot. The network shown along the left side of the figure enables the communication that supports management with existing tools once the OS is executing.

Summary and Conclusion

The Enterprise X-Architecture extends HW scalability beyond that of conventional single-node SHV server platforms. EXA further extends the feature set of current multi-node platform architectures to enable static and dynamic partitioning. Together, the combination of these features endeavors to increase scalability, availability and serviceablity in a cost-effective package.

The configuration and partition management model used with EXA products is based upon existing IBM Director tools, service processors, and infrastructure. This model provides a consistent and seamless approach to both pre-boot (OOB) and runtime management of EXA platforms.

The integration of EXA HW and partition management SW forms a product that focuses on and addresses the needs of the high end server market. The Enterprise X-Architecture product is well-suited to the requirements and objectives of enterprise computing environments.

Acknowledgements

Thanks to Deborah McDonald for the screenshot shown in Figure 7.

References

1. *Hardware Design Guide Version 2.0 for Microsoft® WindowsNTServer*, Intel Corporation and Microsoft® Corporation, July, 1998.
2. *Multiprocessor Specification (MPS)* 1.4, Intel, May, 1997.
3. *Advanced Configuration and Power Management Initiative* (ACPI rev 1.0b), Intel, Microsoft®, Toshiba, February, 1999.
4. *Extensible Firmware Interface Specification, Version 0.91*, Intel Corporation, July, 1999.
5. *Running LINUX*, Matt Welsh, Matthias Kalle Dalheimer, Lar Kaufman, O'Reilly & Associates, August, 1999.
6. *Windows 2000 Product Guide*, Microsoft® Corporation, December, 1999.
7. *PCI Local Bus Specification*, Version 2.2, December 18, 1998, PCI Special Interest Group.
8. *PCI-X Addendum to the PCI Local Bus Specification*, Version 1.0, September 22, 1999, PCI Special Interest Group.
9. *PCI-to-PCI Bridge Architecture Specification*, Version 1.1, December 18, 1998, PCI Special Interest Group.
10. *Next Generation I/O: A New Approach to Server I/O Architectures*, Technical White Paper, AberdeenGroup, Inc., February, 1999.
11. *STiNG: A CC-NUMA Computer System for the Commercial Marketplace*, T. Lovett and R. Clapp, Proceedings of the 23rd International Symposium on Computer Architecture, May 1996.
12. *The Synfinity Interconnect Architecture: A Cost-Effective Infrastructure for High-Performance Servers*, Wolf-Dietrich Weber, Stephen Gold, Takeshi Shimizu, Thomas Wicki, and Winfried Wilcke (HAL Computer Systems) Pat Helland, (Microsoft® Corporation), Association for Computing Machinery, 1997.
13. *Slicing the AS/400 with Logical Partitioning: A How to Guide*, IBM Corporation, August, 1999.
14. *LPAR Dynamic Storage Re-configuration*, Don Boos, IBM Corporation, 1997.
15. *DISCO: Running Commodity Operating Systems on Scalable Multiprocessors*, Edouard Bugnion, Scot Devine, Mendel Rosenblum, 1997, Computer System Laboratory, Stanford University, Stanford, CA

[1] FPAR and SPAR functionality require no additional or specific support from the Operating System. However, multi-node (CC-NUMA) partitions, which may be a onsequence of configurations defined by the user, could potentially require OS modifications to execute or operate as expected on a CC-NUMA partition.2

Performance Analysis of EXA Technology

– Dan Colglazier, Rick Harper, Larry Whitley

Introduction

To accurately assess the performance implications of design tradeoffs in future systems, one must be able to accurately predict their performance. Measurements are very useful for understanding the performance of current systems. Unfortunately it is impractical to build hardware prototypes of the design alternatives to measure their performance. Instead, flexible software models are used as prototypes of the design alternatives to determine their performance. Two models, the EXA Spreadsheet Model and the EXA Simulation Model, were used to determine the performance of design alternatives. Each required similar inputs: design timings and sizings and workload statistics. Design timings and sizings come from the system designers proposing the various design alternatives. Workload statistics are derived from measurements, trace analysis, cache simulation and predicted future software characteristics.

The performance analysis process used is shown in Figure 1. The inputs to the process are shown in ovals. The various models and tools are shown in rectangles. The model and tool outputs are shown in hexagons. Design timings and sizings are direct inputs to both the EXA Spreadsheet Model and the EXA Simulation Model. Measurements and future software characteristics are directly used to produce workload statistics. Traces are the input to the trace analysis tools and the cache simulator which uses a patented methodology. Outputs from the cache simulator and trace analysis tools along with measurements and future software characteristics produce the workload statistics. These along with the design timings and sizings are the inputs to the models which produce the system performance results. This paper will go through this process in detail and show the results produced by the models of some of the design alternatives.

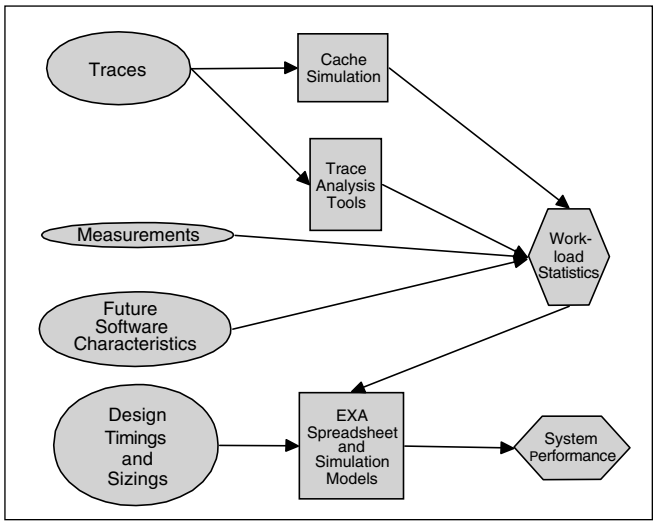


Figure 1 Performance Analysis Process

Tracing

Traces are a recording of the history of particular events of interest. To study the EXA design alternatives, system bus traces were used. These traces are a history of all the transactions on the system bus over a period of time. Most of these transactions were caused by cache misses in the processor. Each trace record contains the type of request that it is, the address of the request, the snoop result, the response it received, and timing information. These traces were used to determine the cache statistics of the various proposed caches as well as to determine the future system bus request rates which differ from current rates due to different processors being used in the future designs.

A tool called RTS (Real-time Tracing System) was designed and built specifically for capturing system bus traces. Some of the goals of the design of RTS were to produce error free traces that were as long as possible and not impact the system being traced. There was also a need to have a short time between the capture of traces in order to quickly produce as many traces as possible in the short amount of time that the systems being traced were available. Software-based tracing techniques require that the system being traced be slowed down or periodically stopped in order to record the trace. This takes a lot of time and may alter the way the system being traced functions. These types of

techniques did not meet the goals of not impacting the system being traced and producing as many traces as possible in a short amount of time. Therefore, hardware-based tracing was used which was transparent to the system being traced.

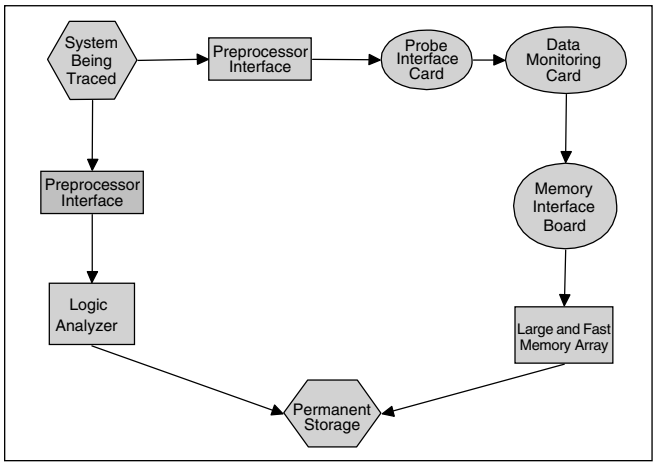


Figure 2 The RTS Tool

A diagram of the RTS tool is shown above in Figure 2. Special hardware designed and built for RTS is shown in ovals. Vendor produced hardware used for RTS is shown in rectangles. The RTS tool is composed of a number of different parts to get the needed signals, reshape and rearrange them as necessary, and store them. Two preprocessor interfaces are used to get the needed system bus signals from two different processors. Since the system bus is shared among up to four processors, both processors, and thus both preprocessor interfaces see the same signals. One of the preprocessor interfaces, is attached to a logic analyzer which captures as much of the first part of the trace as it has the capacity to do. This copy of the first part of the trace is compared to that captured by the RTS to verify the tool is working properly and capturing accurate traces. The second preprocessor interface sends the system bus signals to the Probe Interface Card. The preprocessor interface outputs the system bus signals in a format that conforms to a proprietary format. The Probe Interface Card converts these signals to transistor-transistor logic (TTL) signals to be used by the next stage of the RTS, the Data Monitoring Card.

The Data Monitoring Card performs a variety of functions. The system bus upon which the traces were captured ran at 100MHz. The traces are written to temporary storage which can not record them that quickly in one stream. Therefore, the Data Monitoring Card splits the 100MHz stream of signals into two streams of 50MHz signals. The Data Monitoring Card filters out unneeded signals to reduce the amount of space each bus cycle needs in storage. The smaller the amount of information stored for each cycle, the larger the number of cycles that can be captured since the size of the temporary storage is fixed. The 50MHz signals are split again into address signals and control signals with 32 bits in each. The address signals are only valid on the system bus during the first two cycles of a request, so only those two cycles are stored in the temporary storage for each bus transaction. The control signals can occur on any bus cycle. Two methods are used to store these signals. The first is to store the control signals for all bus cycles. This preserves all the timing information in the trace but stores a lot of unneeded cycles. The second method is to filter out the unneeded cycles to produce traces longer in time. With this method, time stamps are also recorded along with the signals in order to know the number of bus cycles between each bus transaction. The tradeoff is that some information is lost.

The Memory Interface Board transfers the trace data from the Data Monitoring Card to the temporary storage. Its main function is to convert the TTL signals to emitter coupled logic (ECL) signals. Two boxes are used as the temporary storage for the traces. Each of our boxes contains 24GB of memory providing a total of 48GB for each trace. The interface board for the boxes can store 32 bits at up to 90MHz. The RTS uses two of these cards in each box, one for the address signals and one for the control signals. Each box captures one of the 50MHz streams. When a trace fills up the 48GB of storage, the trace is complete. The trace is then written to permanent storage and a new trace is started. A number of traces are captured from each trace system so that the most representative one can be selected.

As mentioned above, the system bus traces are captured into four files - two containing address bits and two containing control bits. The tools that use these traces expect them to be in a single file. In addition, the tools expect the traces to contain records with all the signals pertaining to a particular bus request in them. The system bus is pipelined. This means multiple requests can be on the system bus at the same time in various phases. A program called XLATE is used to read the four files, find and put together all the bits pertaining to a particular bus request, and output these

records in order as a translated trace. XLATE also performs error and parity checking on the traces and produces statistics on the correctness of them.

Once the traces have been translated, the most representative one from a set of similar traces is selected. This is done by using each trace as input to a cache simulator (IMPACT which is described later) run with a number of cache configurations. The output of each run is analyzed along with counter data captured with the trace, and output from a trace statistics gathering tool (FP which is described later) for each trace. The key statistics from all the traces are averaged and each key statistic from each trace is looked at to see how close it is to the average. A trace can be selected as representative if all of its key statistics are close to the average and it tracks the average well when things like the cache size are changed. The trace that most closely matches and tracks the averages of the key statistics is chosen as the trace to use from that set of traces.

Just as important as selecting representative traces is selecting representative workloads to trace. Three commonly used and publicly available workloads were traced. Each was traced on systems set up and tuned specifically to run these workloads. The first was TPC-C which is an on-line transaction processing benchmark. It was produced by the Transaction Processing Performance Council (TPC). This benchmark is representative of a typical large server system. The second was the SAP, Standard Applications Benchmarks produced by the SAP Benchmark Council. This benchmark reflects standard business scenarios and tests the hardware and database performance of SAP software. The third was Lotus Notesbench produced by the Notesbench Consortium. This benchmark is representative of mail and database usage and is for evaluating the performance of mail and database servers. These three workloads provide a good representation of real applications running on Intel-based servers.

Most of these traces were captured on systems with four processors. One TPC-C trace was captured on a system with eight processors. This particular system has two system busses each requiring an RTS tool. Special consideration had to be taken to keep the two halves of this trace synchronized. Our earliest traces were captured before the size of the memory array was increased. These traces captured between fifteen and twenty seconds of system bus activity during which seventy to ninety million bus requests took place. Our later traces using 48GB of memory captured over two minutes of system bus activity during which over one billion bus requests took place. Studies done on these traces found them sufficiently long to produce accurate performance statistics for the EXA design alternatives.

Cache Simulation

The cache simulator is called IMPACT – Input-driven MultiProcessor Analysis Cache Tool. It is written in C. The basic approach of IMPACT is to model the target cache directories and keep counts of the occurrences of the events of interest. The target cache directories are represented by three-dimensional array structures. The three dimensions are: 1) to identify a particular cache, 2) to identify a particular congruence class, and 3) to identify a particular row in a congruence class. Each directory entry contains the address of the cache line, timing information used for cache line replacements, and the state of the cache line (i.e. exclusive, shared, etc.). IMPACT reads the input trace, one bus request at a time, performs whatever actions would take place on the target cache(s) for that request, and increments the counts of whatever events occur. IMPACT also has a special optional mode in which a bus request can generate additional requests to simulate situations such as prefetching.

In addition to the input trace, IMPACT reads in a parameter file which it uses to configure the target caches and how they operate. Three of these parameters are the cache size, the line size, and the set-associativity. A wide variety of cache sizes can be simulated, limited only by the amount of storage required to hold the directory structures. Any reasonable line size can be specified keeping in mind that the smaller the line size, the more cache directory entries there are. The set-associativity determines the size of each congruence class and can be any reasonable size. These three parameters determine two of the three dimensions in the target cache directory array structures: number of congruence classes and number of rows. Sectored caches can also be simulated by IMPACT using the input parameter specifying the number of caches lines per sector. In addition, these sectors can be loaded by line or by the entire sector by specifying the load size parameter.

IMPACT is quite flexible in the system configurations it can model. The caches of up to 32 processors can be simulated. IMPACT can simulate private caches (one cache per processor), shared caches (one cache for all processors), and semi-shared caches (more than one processor sharing a cache with multiple caches existing). IMPACT can model one or two levels of caches. With two levels of caches, the following combinations can be modeled: private first-level caches with private, semi-shared or shared second-level caches and semi-shared first-level caches with semi-shared or shared second-level caches. The two levels of caches interact in both directions. The first-level caches send their misses and writebacks to the second-level caches. If the second-level caches are enforcing inclusion in the first-level caches, then invalidate requests are sent to the appropriate first-level caches when cache lines are removed from the second-level cache. An input parameter specifies the number of first-level caches attached to a particular bus. Other input parameters specify how many processors share each first-level and second-level cache. Each level of cache can use a variety of replacement algorithms including true and pseudo least-recently-used replacement and random replacement. The two cache levels can implement different coherency algorithms which determine where lines are kept and in what states they end up.

Besides specifying the cache and system configurations and policies, the input parameters specify a number of options for running the simulation. The maximum number of transactions simulated during a run is specified in an input parameter. Another parameter specifies the number of transactions used to prime the cache without statistics being collected on them. Transactions can be limited to a specific address range in two input parameters. A number of model outputs are optionally requested by input parameters. Among these are traces of the target cache activity which start at a specified transaction number and are a specified number of transactions long. Another useful optional output is the number of cache lines in each state at a specified interval throughout the simulation run. Memory maps of the transactions and cache misses can also be optionally produced.

IMPACT produces an output file containing the counts of the interesting events that occurred during the simulation run. Most counts are broken out by requester and transaction type and are produced for each cache if appropriate. The most generally useful statistics produced are total trace requests, cache references, cache misses, replacement writebacks, and snoop writebacks. In addition, other less generally useful statistics are kept such as the number of cache hits not to the most recently used line. This helps determine how often the cache directory must be updated. Another of these is cache hits which act likes cache misses due to coherency requirements. If two levels of caches are being simulated, statistics are kept on their interaction. Among these are the number of back invalidations and the number of writebacks that these cause. Other interesting statistics that are included in the output are line state and state transition counts, snoop results, and the count of lines replaced without being referenced when doing prefetching. The IMPACT output file is typically brought into a spreadsheet where various counts are combined to produce needed statistics such as cache miss rates.

Workload Statistics

Characteristic statistics of each workload can be obtained through measurements. Current Intel processors contain two performance counters which can be programmed to count a number of events. These counters are used to obtain the processor cache request rates and miss rates on each workload. In addition, the writeback transaction rate can be determined using the counters, and the percentage of replaced lines causing writebacks can be calculated. Another useful count is the number of snoops found modified which cause snoop writebacks. The counters are also used during tracing to count the number of instructions completed and the number of processor cache references during the trace.

In addition to the information from the counters, the traces provide a number of useful workload statistics. While translating the traces from bus event traces to bus transaction traces using the XLATE tool, statistics on the trace are counted. Among the most interesting are the total number of bus cycles of a trace, the average number of requests on the bus awaiting responses, and the number of times snoops are stalled before completing. A program called FP was written to produce additional statistics directly from the traces. Its original purpose was to determine the number of unique cache lines referenced in a trace by each requester over time. FP also counts the number of each request type from each

requester. This is used along with the timing information from the trace to calculate the rates of each type of request by requester. In addition, FP counts the responses, snoop responses and cache attributes by request type for each requester. FP also keeps a count of the response times for each requester and provides a map of where in memory each requester is making references.

By tracing the system bus, a record is made of all the requests on the bus over a period of time. These statistics provide a good picture of how the workloads perform on current hardware. Future processors will perform differently resulting in different request rates on the system bus. Future memory hierarchies must be designed for these future request rates. If the highest-level cache in the processor changes, the request rate on the system bus is directly affected. Since future processors will have a wide variety of caches, a methodology was needed to account for these different caches. Traditional cache simulation requires a trace of the inputs to a cache. A bus trace is the outputs of the highest-level cache in the processor. The inputs of this cache are not available to trace without turning off the cache and greatly affecting the system being traced. Therefore, a new method of cache simulation was needed and developed that uses the outputs of the highest-level cache in the processor as the inputs to the simulation of another cache that will be the highest-level cache in a future processor. This methodology was issued a United States Patent.

A trace of cache outputs does not contain everything needed for cache simulation as does a trace of cache inputs. Information from the cache hits is lost. The cache hits are needed to determine the total number of references to the cache, but this information is collected using the counters while the cache output trace is being captured. With the same references, a larger cache will contain the vast majority of the entries of a smaller cache. Cache hits in a smaller cache will almost always be cache hits in a larger cache. The cache hits to a smaller cache are then not needed to simulate a larger cache. Because of this, the cache misses of a smaller cache can be used as input to a cache simulation of a larger cache. Because the cache hits to the smaller cache are not seen in the simulation of the larger cache, the cache replacement algorithm is affected. Cache hits usually cause the hit line to be replaced later than the other lines referenced earlier. This information is lost to the cache simulator of the larger cache, but this lost information has only a small effect on the number of misses and is not a significant problem.

Most caches keep track of the state of their lines, for example whether they have been modified. In writeback caches, lines that have been modified must be written back to memory when they are removed from the cache. A cache simulator must keep track of these line states accurately in order to produce correct statistics concerning writeback activity. Once a cache line has been designated as the only copy, it can be modified internally with a future reference. This reference will be a cache hit and will not appear in the cache output trace. Therefore the cache simulator will not know when these lines have been modified. This lost information is significant and has a very large effect on the writebacks in the cache simulator, but this difficulty can also be handled using the technique described in the following paragraph.

Besides cache misses, writebacks are also captured in the cache output trace. Writebacks occur when a modified line is replaced or another cache wants a copy of the modified line. All modified lines are eventually written back to memory. Therefore, the writeback references are a record of which lines have been modified. Sometime before the writeback occurred, the line was modified. This occurred most likely between the last reference to this line in the miss trace before the writeback and the writeback itself. For each writeback, this last reference before it is found and marked using a program called WBMARKS. The cache simulator reads these marks and keeps track of the lines that have been marked. When a line is to be replaced or wanted by another cache, it is the only copy, and it has been marked, the simulator treats it as if it had been modified. This method has been used quite successfully to simulate caches larger than the ones in the processors in the systems that were traced. If statistics were needed on caches too small to accurately simulate directly, a number of larger caches with similar configurations except for cache size were simulated and the results were used to predict the statistics of the desired cache using curve-fitting.

Once cache statistics are determined on the highest-level cache in the target processor, it is quite easy to

calculate the target system bus request rates. Through measurements and trace analysis, the miss rates and bus request rates of the tracing system are known. Bus requests are mostly the misses of the highest-level cache in the processor. Therefore, the bus request rates of the target system are pretty much the bus request rates of the tracing system multiplied by the ratio of the miss rates between the target system and the tracing system. Writeback rates can be calculated similarly.

The XceL4 Server Accelerator Cache (XceL4 Cache) is large enough to contain the contents of the caches in the processors. In the processor chip, requests that are hits in a larger cache and are misses in a smaller cache are almost always hits in the XceL4 Cache (L4). Therefore, the number of cache misses in the L4 stays basically the same for any reasonable cache size in the processors, and the L4 can be simulated directly using the system bus traces to produce miss rates. Another vital statistic for studying the design alternatives for EXA was the probability of where each request type would be found: local processor cache, local L4, local memory, remote memory, remote L4, or remote processor cache. These probabilities are determined using two levels of cache simulation: 1) the processor caches and 2) the L4s. It is important to have the proper interaction between the two levels of simulated caches and among the caches at the same level. IMPACT has the capability to do this and the flexibility to perform different algorithms to simulate different coherency schemes and different policies between the two cache levels. These probabilities are an important input to the EXA Spreadsheet Model which will be described below.

The techniques described above are very useful for determining the statistics of current workloads on the future target systems. In addition to the hardware changing in the future, the software will also change. This must also be accounted for. EXA systems with more than four processors will use a Non-Uniform Memory Access (NUMA) architecture. Most of today's software is written for Uniform Memory Access architectures. The workload statistics from the measurements and cache simulation show this to be the case for the benchmarks that were traced. Because of the emergence of systems using NUMA architecture, software is being enhanced to increase the percentage of local memory references which increases system performance in NUMA architecture systems. The statistics have been adjusted for this to produce statistics of future software running workloads on the future target systems. These are the workload statistics used in the EXA performance models.

EXA Spreadsheet Model Methodology For Estimating System Performance

For the purposes of supporting design and workload variation studies, a linear spreadsheet model is used to project system-level performance in terms of TPC-C transactions per minute (tpmC).

Abstracted System Structure

This model uses an abstracted view of an 8-way EXA configuration shown in Figure 3. The 8-way configuration consists of two "nodes", each of which contains four processors connected to a system bus. These nodes will be referred to as quads. Each processor contains a cache. Also connected to the system bus is a memory controller providing access to the local L4, the local memory, and the SMP Expansion Port which provides access to remote caches and memories.

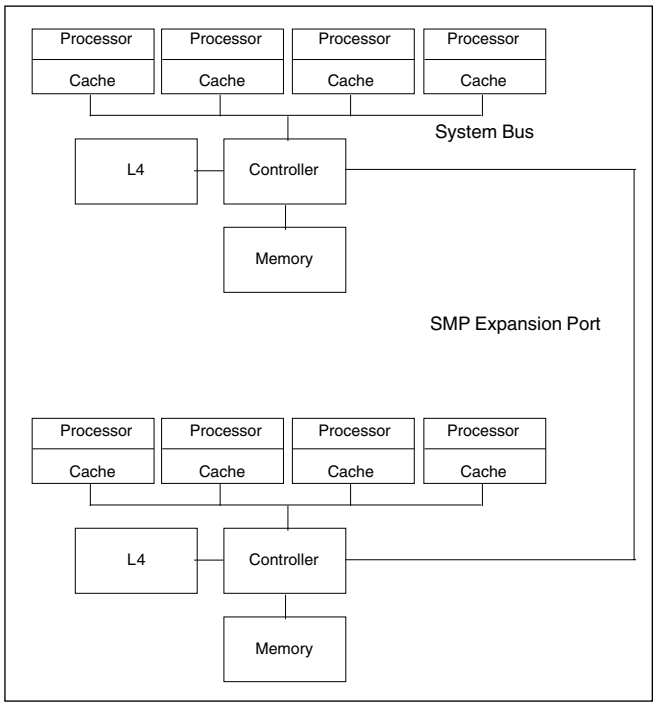


Figure 3 8-way EXA Configuration

Table 1 below shows the parameters of the configuration that are relevant to this analytical method.

Table 1 Configuration Parameters

Processor	1.6 GHz Intel Xeon Processor MP Family
Processor Cache	1 MB, 64B line, 8-way Set Associative
L4	32 MB, 64B line, 4-way Set Associative
Local Processor Cache Latency	80 ns
Remote Processor Cache Latency	625 ns
Local L4 Latency	85 ns
Remote L4 Latency	565 ns
Local Memory Latency	315 ns
Remote Memory Latency	715 ns

Overview of Calculation Methodology

The overall equation for calculating tpmC is:

tpmC =
instructions per second / transaction path length * 60

The transaction path length is the average number of instructions per new order transaction and is obtained by measurements taken on benchmark systems. For a given set of studies, it is held constant for all configurations being analyzed. With ongoing optimization, the path length decreases over time (and thus the absolute tpmC changes over time, all other factors being held constant), but the method still yields useful relative comparisons.

The instructions per second delivered by a certain number of processors with a given memory hierarchy is calculated as:

Instructions per second =
number of processors * processor clock rate / processor cycles per instruction

The number of processors and processor clock rate are fixed for a given design point, while the processor cycles per instruction (CPI) is a reasonably complex function of a number of parameters. It is broken into two components: Core CPI and External CPI. The Core CPI is the number of cycles per instruction of the processor core, assuming an infinite processor cache size, and is to a first approximation a function of the workload characteristics and internal processor and cache architecture. It is obtained from a variety of mutually corroborative sources, such as empirical measurements, the microprocessor vendor, and comparison with known designs.

The External CPI is the contribution to overall CPI that arises due to nonzero latencies of the external cache and memory hierarchy and is a function of the number of processor cache misses per instruction (MPI), the latencies of the various elements of the memory hierarchy beyond the processor cache, and the distribution of accesses to those elements. Assuming the average processor cache miss latency and the processor clock rate are in compatible units (e.g., microseconds and Megahertz), then:

External CPI =
processor cache MPI * average processor cache miss latency / processor clock rate

The processor cache MPI is given by the cache simulation described above. For a single-node EXA system having four processors on a single system bus, a shared L4, and a shared local memory, the average latency of a processor cache miss generated by a given processor is:

Average processor cache miss latency =
Probability of hitting in another processor's cache on that system bus * local processor cache latency +
Probability of hitting in local L4 * local L4 latency +
Probability of hitting local memory * local memory latency

To calculate the average processor cache miss latency for multinode EXA configurations, it is necessary to know the proportion of accesses that are fulfilled in local versus remote processor caches, L4s, and memories. These quantities are collectively referred to as the locality of reference. They are a strong function of the workload and can range from uniform for non-NUMA optimized workloads to highly localized for NUMA workloads. Thus the formulation for average latency of a processor cache miss is:

Average processor cache miss latency =
Probability of hitting in another processor's cache on the local system bus * local processor cache latency +
Probability of hitting in another processor's cache on a remote system bus * remote processor cache latency +

Probability of hitting in local L4 * local L4 latency +
Probability of hitting in remote L4 * remote L4 latency +
Probability of hitting local memory * local memory latency +
Probability of hitting remote memory * remote memory latency

The probability of hitting in various local and remote locations is given by the cache simulator and workload analysis tool described above, and the various latencies are provided by the system designers.

Exa Simulation Model

The EXA Simulation Model is implemented in C++ using CSIM. The inputs to the model are similar to the EXA Spreadsheet Model described above. The simulation model is cycle accurate and is intended to answer questions of performance that rely on queuing information. Specifically, the model was used to explore the design of several components of the EXA chip set: notably the L4 design, memory design, and SMP Expansion Port design.

The approach to the design of the model goes through 3 steps:

1. Class Definition
2. High level class implementation (for all classes)
3. Low level class implementation (for a subset of the classes)

The class definition and high level model implementation will be described followed by a description of the low level model implementation for one of the components of the chip set, the L4.

C++ Class Definition

When implementing a large model in C++ using the CSIM class library, it is advisable to first put together a class diagram that will serve as a guide for future implementation. This class diagram can serve as the basis for organizing the model and organizing a team of people that will implement different parts of the model.

Generally, the classes follow the physical partitions of the chip set. For EXA, there are three modules: Memory/IO Controller (MIOC), Cache/Scalability Controller 32 (CSC32), and Cache/Scalability Controller 64 (CSC64). Likewise, there will be three major classes in the model named, appropriately, MIOC, CSC32 and CSC64. In the chip set, CSC64 was planned to be derived from the CSC32 design. Therefore, the CSC64 class was derived from (inherits from) the CSC32 class in the C++ model.

Each of these major classes contains objects built on smaller classes within them. For example, the CSC32 class will contain an L4 object that is created from an L4 class. Likewise it contains three SMP Expansion Port objects that are each derived from a ScalabilityPort class. As can be seen in the class diagram in Figure 4, there are several inheritance relationships in the model. Beyond the CSC32/CSC64 relationship there is a relationship in the processor buses and another between the I/O ports and the SMP Expansion Ports.

The Bus class is an abstract class that factors out the common elements between the Intel Itanium™ Processor Family bus and the Intel Xeon™ Processor MP Family bus. MIOC implements the Xeon bus and may either attach several Xeon processors or may attach a CSC32 or a CSC64. CSC32 also implements a Xeon bus, while CSC64 implements an Itanium bus. While CSC32 and CSC64 implement different processor buses, they use the same Directory and L4 classes. The necessary differences in the use of these classes did not justify separate classes for each and was instead handled with parameters.

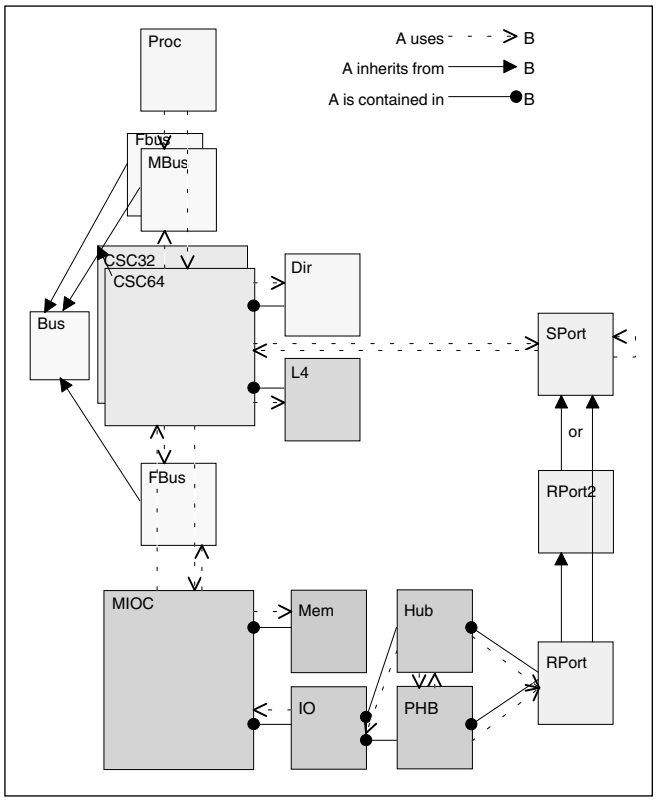


Figure 4 Class Diagram

Another inheritance relationship can be seen in the relationship between the I/O ports on the MIOC and the SMP Expansion Ports on CSC32 and CSC64. The I/O ports are updated implementations of I/O ports on pSeries and iSeries systems. The SMP Expansion Ports are higher speed and more powerful. One would at first think that the SPort should inherit from RPort, but looking into the details of the modifications, it turned out that it was easier to do the reverse.

Building And Running The Model

The class diagram represents the structure of the C++ code, but an object diagram is needed to fully appreciate the structure of the model. The object diagram in Figure 5 shows a fully configured Intel Itanium Processor Family based system, interconnected with SMP Expansion Ports.

When the model is run, it goes through three phases:

1. Build model objects
2. Interconnect model objects
3. Run model

The model contains run-time controls that describe various configuration parameters: number of nodes (CSC64/MIOC pairs), number of processors, size of memory, size of L4, etc. Each of these objects are instantiated by the model without any knowledge of where the other objects that make up the model are located. After all objects are built, the model interconnects each object with the other objects it needs to know about (using methods built into the object for this purpose). The arrows with dots on one end indicate pointers within an object pointing to another object in

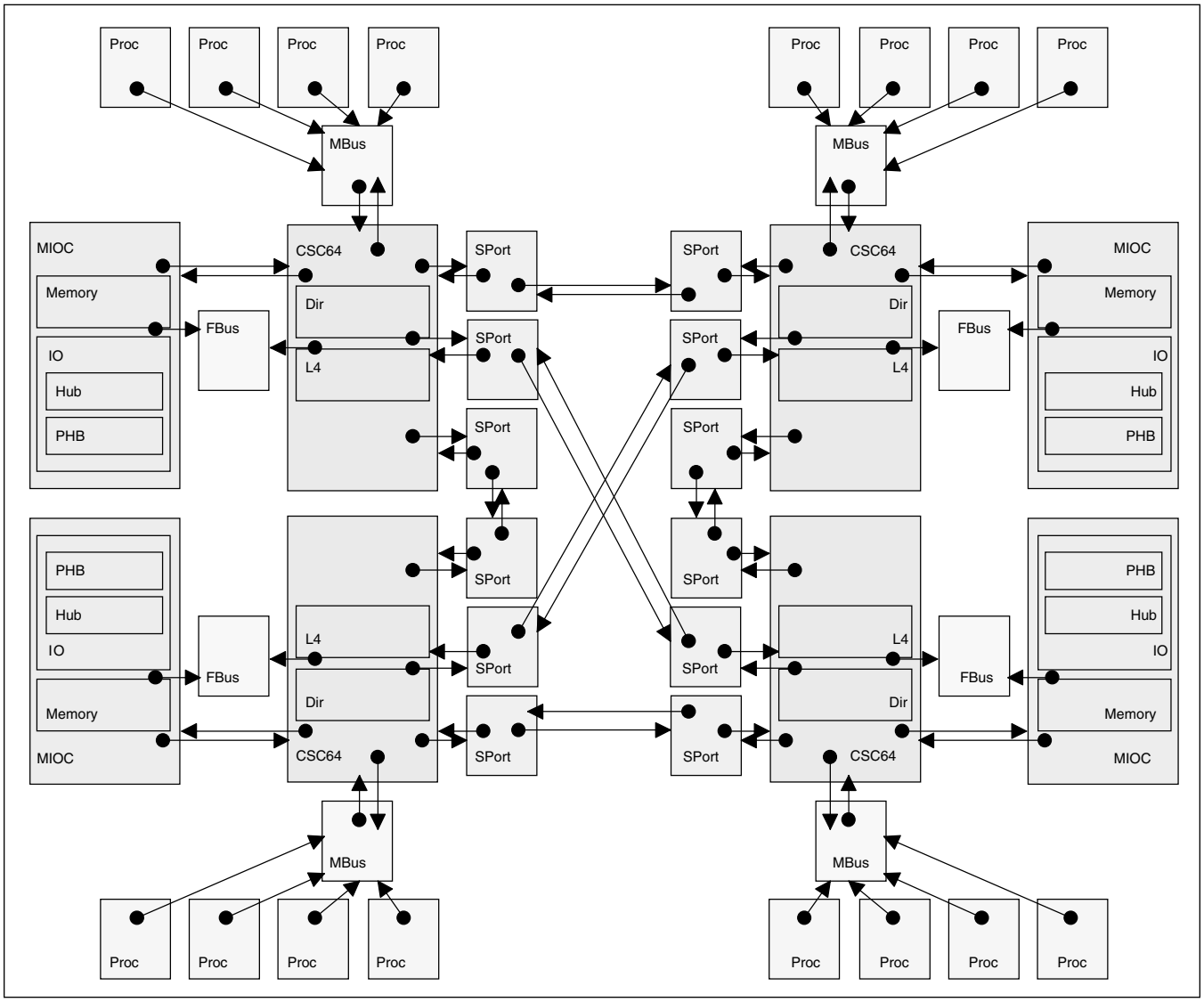


Figure 5 Object Diagram

the model. The purpose of the interconnect phase of building the model is to define these pointers.

Finally, once the model is built and interconnected, it is run by providing stimulus from both the processors and the I/O subsystem. Special care is taken to coordinate the model inputs with that of the EXA Spreadsheet Model so that the two will produce similar results in lightly loaded test cases.

High Level Class Implementation

The weakest point of any simulation model that attempts to deal with a high degree of detail is the length of time it takes to write the model. This length of time delays the point at which useful information can be gleaned from the model, and this delay is in conflict with the goal of providing performance feedback to the designers in the early architectural design phase. With the EXA Simulation Model, an iterative approach was adopted to implementing the model. The model was first implemented at a higher level of abstraction. As the behavior of the model at that level of abstraction was understood, certain classes were re-implemented at lower levels of abstraction (higher levels of detail), often iteratively, until the detailed understanding that was sought was achieved. The advantage of this sort of approach is that the entire model did not need to be implemented at a high level of detail. Only those parts that deserved special attention received this treatment.

Looking back on the EXA simulation model experience, this approach was only partially successful. The model was available sooner than it might have been if a full detailed approach had been pursued. However, it was not available in as timely a manner as desired largely due to unnecessary levels of detail creeping into the supposed high level implementation. In future models experimentation with this method will continue, perhaps restricting the use of classes to force higher levels of abstraction.

L4 Example

One class that went through the iterative process was the L4 class. The addition of an L4 was one of the more interesting aspects of the EXA chip set. The L4 data storage is off-chip double data rate DRAM (Dynamic Random Access Memory) controlled by an on-chip directory, queues and sequencers.

The L4 does provide improved performance over a non-L4 design for most commercial workloads, but the focus here will not be on that performance improvement. Rather, the focus will be on the design of the L4 queues that help achieve that performance.

One of the primary purposes of a simulation model is to assist the designers in exploring alternative queuing algorithms for various parts of the system. The goal is to achieve maximum throughput at a minimum latency and minimum chip area. These goals are in apparent conflict, but with clever queuing algorithms surprising results can be achieved.

It is in this area where C++ with the CSIM class library shows its strength. Other modeling tools are adequate when standard queuing algorithms are in use (First Come First Served, Priority, Processor Shared, Round Robin, etc.), but when there is a need to explore non-standard queuing algorithms, C++ provides the level of control necessary, and CSIM the fundamental constructs to implement it.

The goals of the L4 design are to provide the lowest possible latency and highest possible throughput to processor fetches. The goal for L4 stores is primarily to stay out of the way of the fetches. Given these goals, the queuing algorithm for the L4 starts out as a straightforward priority scheme where processor fetches get higher priority than processor stores.

To complicate matters, the double data rate DRAMs have good performance when presented with a sequence of fetches or a sequence of stores, but their performance degrades significantly when they switch from fetch to store or from store to fetch. It actually does not take a simulation model to understand that a simple priority scheme, combined with this behavior of the double data rate DRAMs, will lead the DRAMs to commonly exhibit a fetch-store-fetch-store-fetch pattern of access. The only time two fetches will be done in sequence is when they arrive at the cache close together. Since stores are likely to be delayed, any holes that open up in the fetch sequence are likely to be filled by stores. This yields the worst possible performance for the processor fetches.

One solution for this, explored by using the simulation model, is to introduce a more complex queuing discipline to the control of the L4. The store queue receives three additional controls, called “high water mark”, “low water mark”, and “number of writes”. The low water mark provides a threshold above which the store queue will

behave as above and below which it will do nothing at all. (For the purpose of this discussion, address conflicts between the fetch and store queue will be ignored.) The high water mark provides a second threshold that moves the priority of the store queue above the priority of the read queue. The "number of writes" control determines the number of writes that will be performed in sequence or until the queue falls below the low water mark, whichever comes first. This is a sensible queuing algorithm given the goals and the limitations of the double data rate DRAM. However, simulation shows that the end result is that the queue length of the store queue is most probably in the neighborhood of the low water mark leading to a behavior very similar to the one the design is trying to avoid in the first place.

Other explorations are also possible. The normal operation of the L4 is not speculative. When an access enters the L4 read queue, it must wait for a signal from the directory indicating that it may proceed when there is data in the cache for this request. (If the directory indicates that it should not proceed, the entry is deleted from the queue.) It is natural to wonder if it would be better to just let the access to the L4 go ahead and throw away the result if it turns out to be invalid.

Using the Object Oriented (OO) capabilities of C++, a new L4Speculative class is created that inherits from the L4 class and modifies the behavior appropriately. The changes are guaranteed to be isolated to the new class and, as the methods accessing that class are unchanged, the rest of the model is unaffected. The results for the workloads run gave the advantage to the non-speculative L4. It turned out that when doing speculative accesses, the additional utilization of the L4 sequencers and buses introduced additional delays which negatively impacted system performance.

The speculation described above was being done in the context of processor fetches. There is also a form of speculation that can be done in controlling the store queue that will benefit the read queue. Recall that the main problem with the double data rate DRAMs is the switch time from fetch to store or from store to fetch.

As mentioned above, processor fetches enter the L4 fetch queue and are held there awaiting release or cancellation by the directory. Again using C++'s OO capabilities, we create a new L4ReadMaybe class that speculatively stops writes when there is one or more unreleased operations stored in the L4 fetch queue. If the directory releases the fetch to the L4, the writes have been stopped or are already in the process of stopping and the fetch's queuing time behind the writes is significantly reduced. If the directory cancels the fetch, the operation disappears and the writes will start up again after missing a few beats. Analysis of this showed a significant improvement over the normal non-speculative store queue and the feature was included in the EXA design.

Workload And System Design Impact On Performance Using The EXA Spreadsheet Model

This section shows the use of the EXA Spreadsheet Model to assess the performance impact of certain design modifications and potential variations in workload. For simplicity, normalized curves will be presented for 4-way and 8-way systems where the performance of a 4-way is defined to be equal to one. Also, only a few of the myriad design possibilities that have been analyzed with this methodology are being shown.

Locality Of Reference

The locality of reference that is assumed for the analysis of the EXA system is based on anticipated changes to a NUMA-unaware workload which was obtained by driving the 8-way EXA cache simulation with the 8-way trace taken at Raleigh. This workload had the conditional localities of reference shown in Table 2 averaged over all latency-incurring transaction types.

Table 2 Workload Localities of Reference

	Local	Remote
Processor Cache	43	57
L4	70	30
Memory	50	50

The processor cache locality (the conditional probability that a processor's cache miss hits in the cache of another processor on the same 4-way node, assuming that it hits in a processor cache somewhere) appears to closely approximate a uniform distribution of 3/7, while the memory locality (i.e., the probability that an L4 miss hits in local memory) is 50%, also implying a uniform distribution of such accesses across the entire system. Interestingly, the L4 provides a locality that is noticeably better than 50%.

The characteristics of a NUMA-aware workload were not empirically available for the anticipated EXA operating system, database, and workload, so they were extrapolated via discussion with various groups within IBM such as Beaverton, Rochester, Raleigh, and Research. These characteristics were what was believed might be reasonably obtained with diligent use of emerging NUMA APIs (Application Programming Interfaces) and are shown in the Table 3 below.

Table 3 NUMA-aware Workload Localities of Reference

	Local	Remote
Processor Cache	51	49
L4	88	12
Memory	89	11

Note that the processor cache locality is only slightly improved, because even with the use of memory locality APIs, it is expected that a given line that is found in another processor's cache is probably a frequently used line such as a lock, and all processors are accessing that lock regardless of where they are. Memory locality has substantially improved, and this is believed to be the main impact of utilizing the anticipated NUMA memory locality APIs. L4 locality has improved somewhat, largely as a collateral result of the improvement in memory locality. This NUMA workload will be the baseline for the evaluations shown in this section.

Figure 6 shows the effect of varying memory locality from 0% (memory references are distributed 50-50 between the local and the remote node in the 8-way) to 100% (all memory references are fulfilled on the local node.) Note that because of its L4, the EXA design is not particularly sensitive to memory locality because most processor cache misses either hit in another processor's cache, or in an L4.

However, a dramatic result is achieved when processor cache locality is increased from 0% (a hit to a same node processor's cache has 3/7 probability and a hit to a different node processor's cache is 4/7) to 100% (the probability of a hit to a same node processor's cache is 100% and the probability of a hit to a different node processor's cache is 0%). This is shown in Figure 7. Admittedly, achieving processor cache locality is a harder programming problem than achieving memory locality because

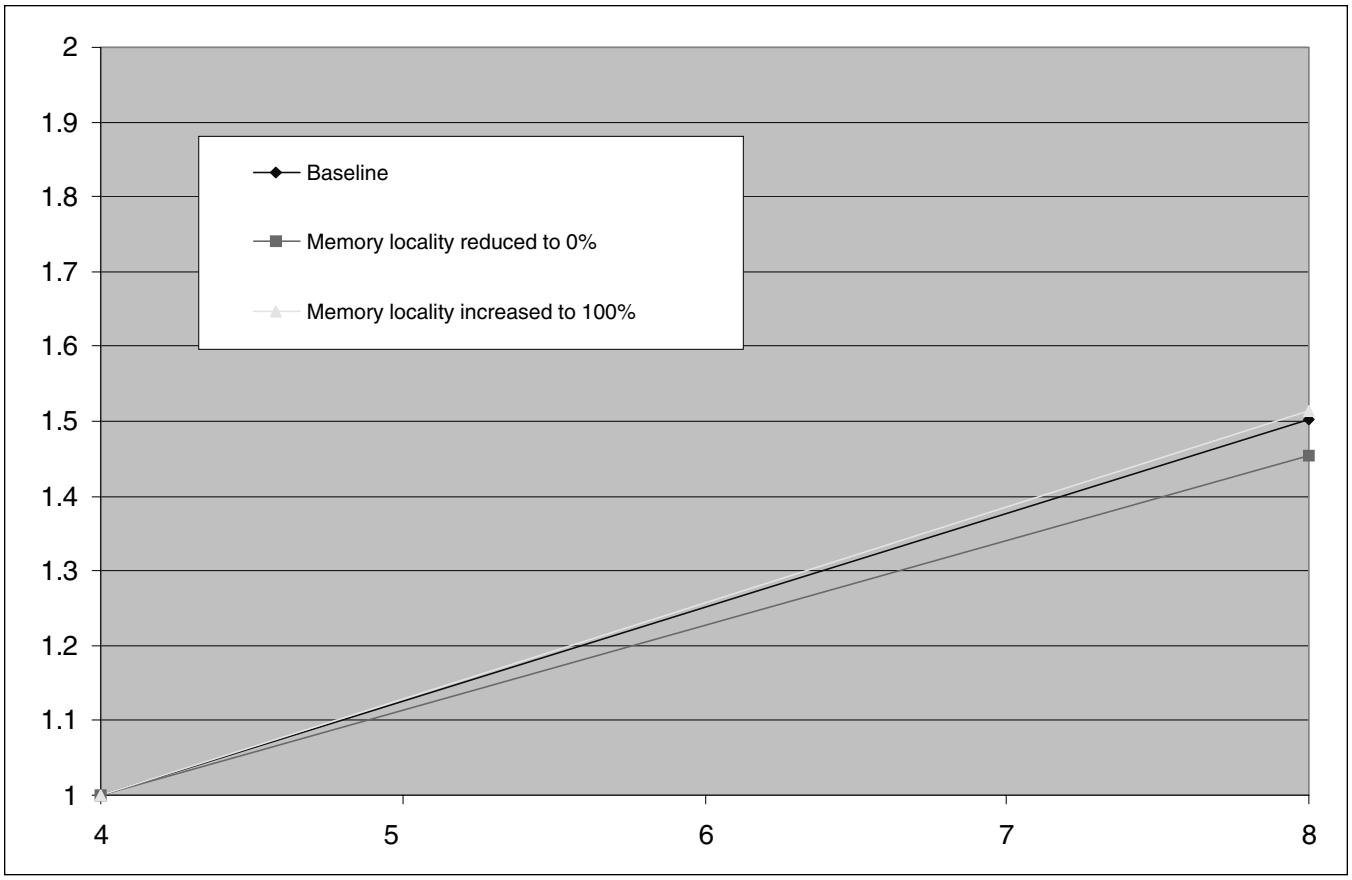


Figure 6 Effect of Varying Memory Locality

it implies confining sharing to a small set of processors via lock breakup and other techniques, and shared data structures tend to be uniformly shared by all processors in a system. However, the large potential performance impact could motivate pursuing this optimization.

Latency

The next two studies show the effect of changing local and remote latencies. In the first example, local L4 and memory latencies are reduced and augmented by 10%. The sensitivity study shows that a 2% change in 4-way performance is obtained for a 10% change in local L4 and local memory latency as shown in Table 4.

Table 4 Local L4 and Memory Latency Sensitivity Study

4-way Baseline	1.00
Local Latencies Reduced by 10%	1.02
Local Latencies Increased by 10%	0.98

The effects of reducing and increasing remote latencies by 10% are shown in Figure 8. Note that in this study, remote processor cache, remote L4, and remote memory latencies were all changed by +/- 10%. The results show that, all other things being equal, a 2.4% change in 8-way system performance is obtained for a 10% change in remote latency.

Cache Configuration

The final study presented shows the effect of doubling the L4 size from 32MB to 64MB. Because of limited directory space in the cache controller chip, to achieve this the line size of the cache was doubled from 64 to 128 bytes.

Figure 9 shows the performance projections. At the 4-way point, the larger cache provides a small improvement in performance. At the 8-way point, however, a slight performance degradation is seen. This interesting conclusion is obtained because even though the absolute L4 hit rate has increased, the relative probability that a hit in another processor's cache or in L4 must access a remote cache has increased. This can be seen in Table 5 which shows the localities of reference for the two cache designs. The performance degradation at the 8-way point may not occur in the future with the anticipated NUMA memory locality APIs.

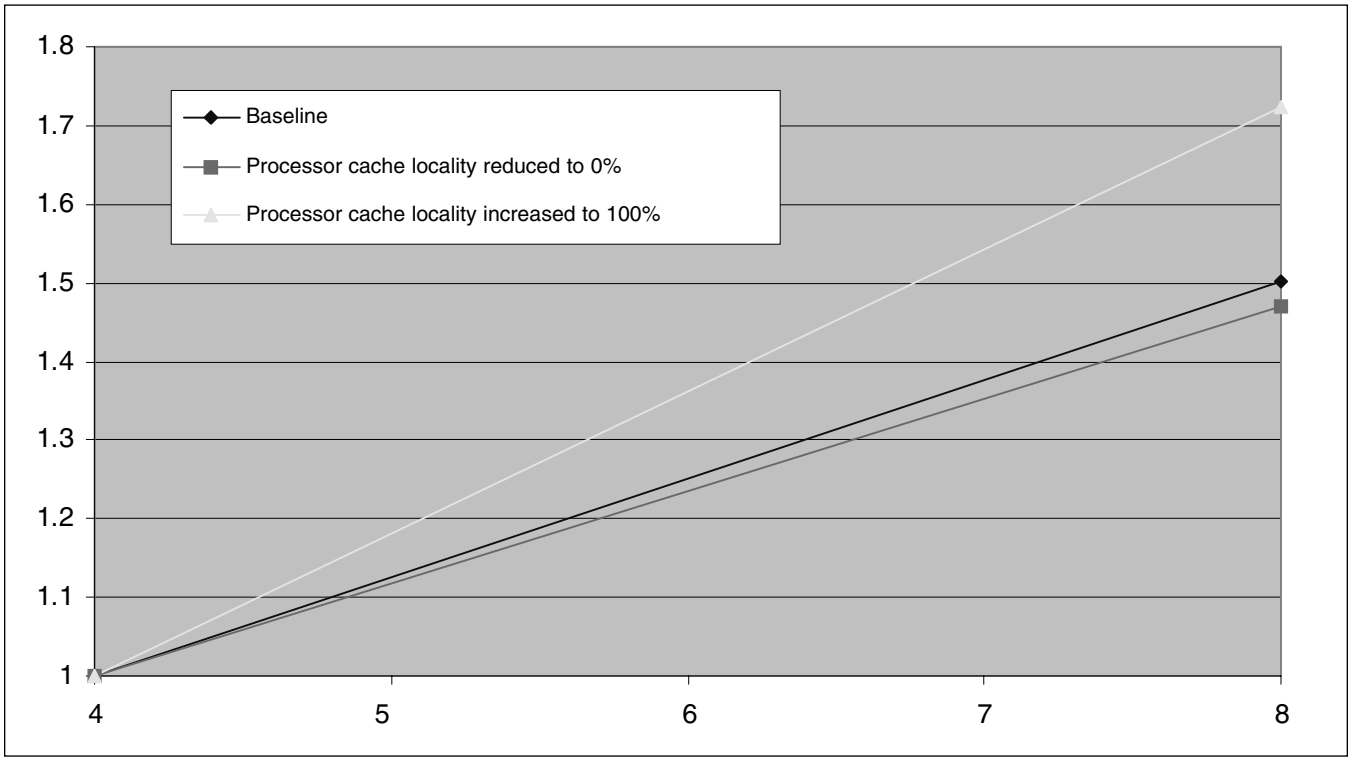


Figure 7 Effect of Varying Processor Cache Locality

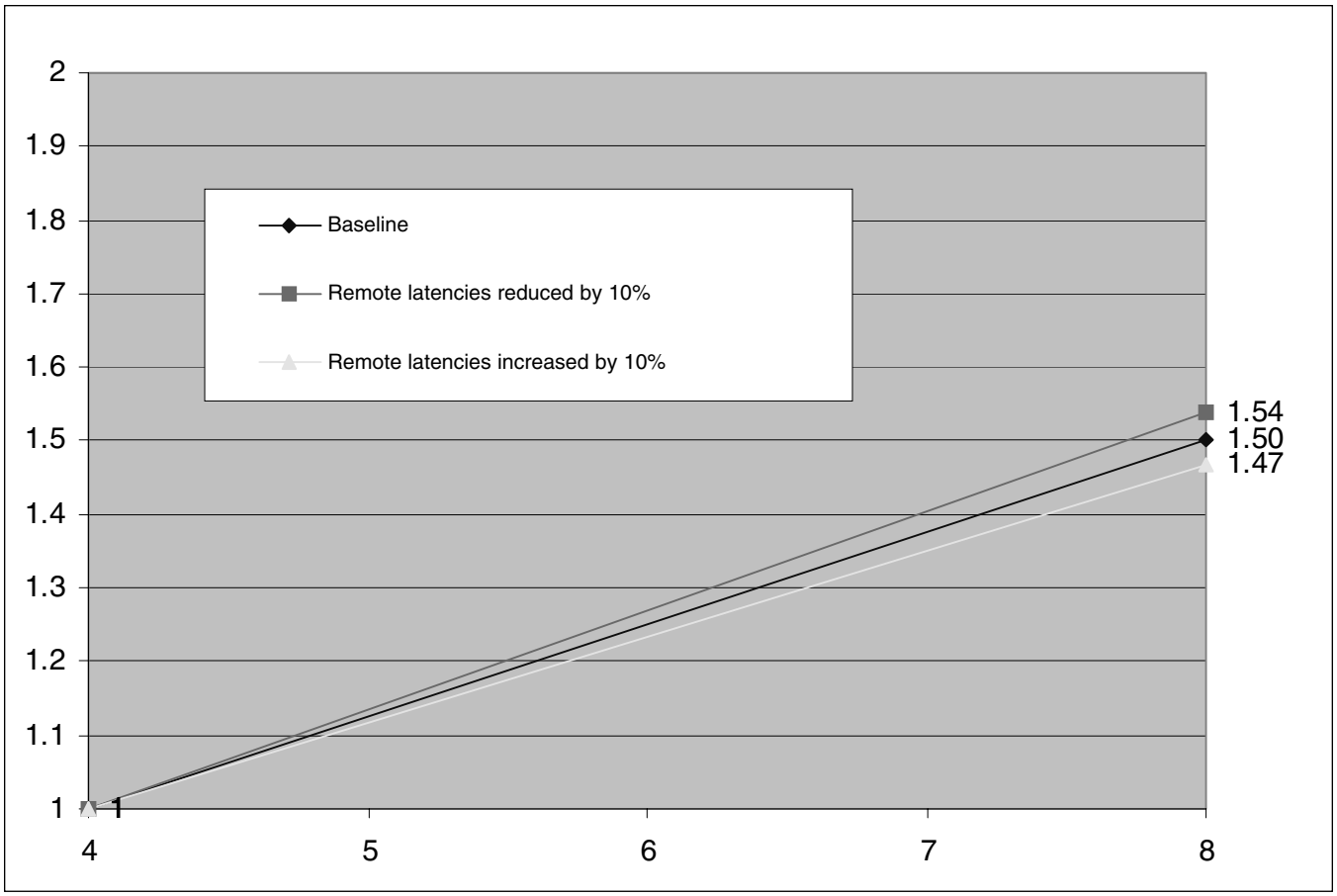


Figure 8 Effect of Varying Remote Latencies

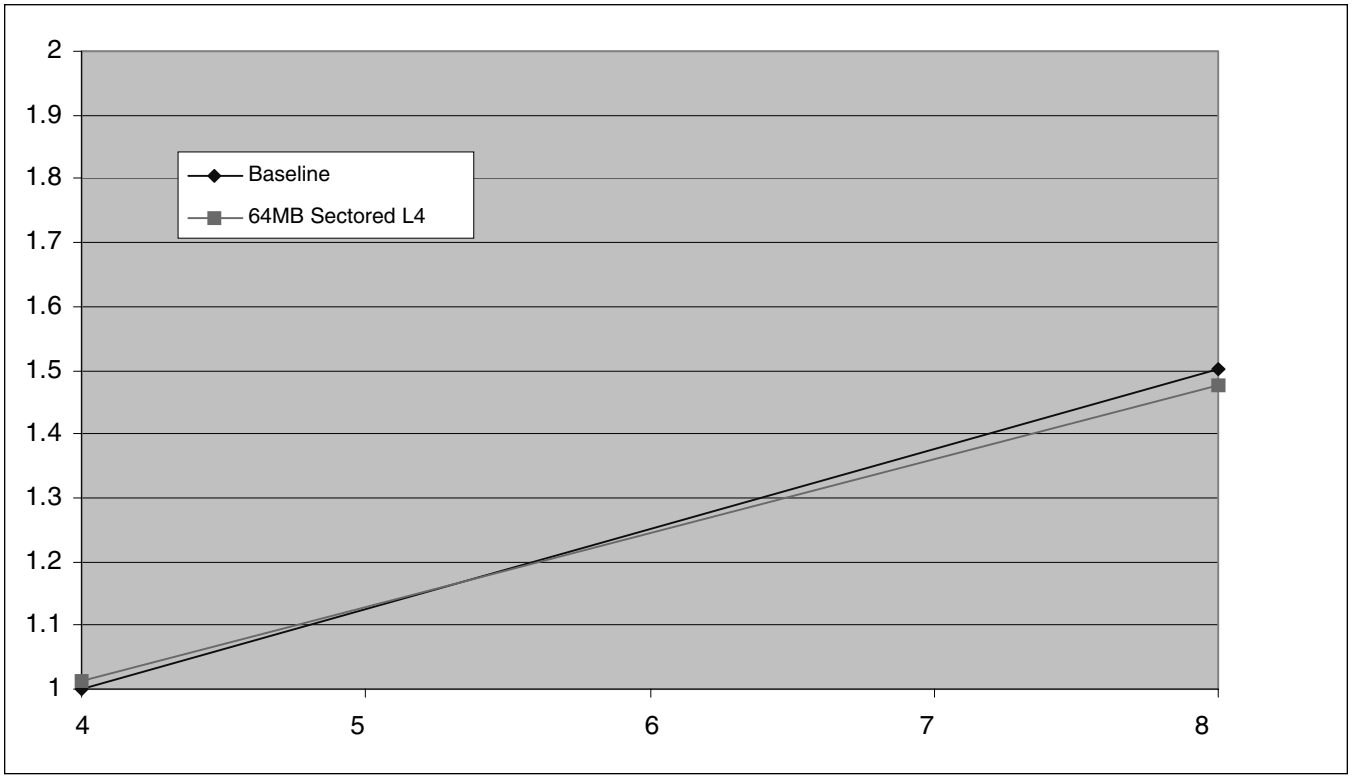


Figure 9 Effect of Doubling the L4 Size

Table 5 Localities of Reference For Two Designs

32 MB L4, 64B Line, non sectored		
	Local	Remote
Processor Cache Hit	10.59%	13.79%
L4 Hit	44.68%	19.79%
Memory Hit	10.03%	1.11%
64 MB L4, 128B Line, 2 Sectors		
	Local	Remote
Processor Cache Hit	11.97%	15.62%
L4 Hit	39.61%	23.08%
Memory Hit	8.75%	0.97%
Data Reads Only		

Conclusion

The performance analysis process used in the design of the Enterprise X-Architecture has been described. This included the collection of measurements and traces, trace analysis, and cache simulation which were all used along with future cache characteristics to produce workload statistics. The workload statistics and design timings and sizings were used as the inputs to the two models: the EXA Simulation Model and the EXA Spreadsheet Model. How the models were used to guide the design for peak performance has also been shown. An example of the use of the EXA Simulation Model was to look at various queuing algorithms for the L4. An algorithm was found that showed a significant performance improvement and was included in the EXA design. The EXA Spreadsheet Model showed that the EXA design is not particularly sensitive to memory locality, but is quite sensitive to processor cache locality. It also showed a 2% change in 4-way performance for a 10% change in local latency and a 2.4% change in 8-way performance for a 10% change in remote latency. Finally, the EXA Spreadsheet Model showed that a larger cache is not always better as an increase in the relative probability of accessing a remote cache produced a slight performance degradation in an 8-way system.

Authors

Wayne M. Barrett
Server Group

Mr. Barrett is an Advisory Engineer, currently working on Intel chipset development. Since joining IBM in 1983, he has worked as an ASIC designer focusing on the areas of I/O bus, processor mezzanine bus, and multiprocessor system bus interfaces. He has been a member of the development teams for the S/36, S/38, AS/400, iSeries, pSeries and xSeries. He received BS EE and ME EE degrees from Brigham Young University.

Lee Blackmon
Server Group

Mr. Blackmon is an Advisory Engineer working in the area of Intel Chipset Development. He has been a memory subsystem logic designer for several IBM products including S/36, AS/400, iSeries, pSeries, ASCI, and now the Enterprise X-Architecture™ chipset. Mr. Blackmon has received an Outstanding Technical Achievement award for his work on S80 Chip Development. He has two patents issued, four more patent applications filed, and 16 invention disclosures published. Mr. Blackmon joined IBM in 1981 after receiving a BSEE degree from Bradley University, Peoria, IL.

P. Maurice Bland
@server xSeries Group

Mr. Bland is a Senior Technical Staff Member in the @server xSeries system development organization, where he is responsible for high end Intel based server system design and architecture. Since joining IBM in 1979, he has designed many PC-based products, ranging from laptops to servers. Mr. Bland has reached an 11th invention achievement plateau, having received over 25 patents in IO bus bridging, systems power management, memory subsystem design, and fault tolerant system design. He received a BS degree in electrical engineering from the University of Kentucky in 1978.

John M. Borkenhagen
Server Group

Mr. Borkenhagen is a Senior Technical Staff Member serving as technical lead for xSeries Chipset Design. His interests are in memory subsystem micro architecture, multiprocessor efficiency, memory coherency, and hardware multithreading. Mr. Borkenhagen joined IBM at Rochester, MN in 1984 after receiving B.S. and M.S. degrees in Electrical and Computer Engineering from the University of Wisconsin, Madison. He has filed 20 patents and has received a corporate award, two outstanding technical achievement awards and an outstanding innovation award.

Jim Bozek
@server xSeries Group

Mr. Bozek received a Bachelor of Music in Theory and Composition from Arizona State University, Tempe, AZ, in 1980. He received a Master of Computer Science from the University of Illinois in Urbana, IL in 1984.

During his career, Mr. Bozek has developed a diverse set of skills in the field of computer software design and development. During that time he has had the opportunity to work in the areas of Computer Graphics and Imaging, Scientific and Engineering Visualization, Digital Audio and Multimedia, Distributed Object Technology, Unix and Windows OS development, Platform and System Management, VLSI design, and silicon compilers. Most recently, his main career focus has been in the area of system software and firmware related to large scale systems and NUMA architectures.

Mr. Bozek works at the IBM Center for Microsoft® Techologies (ICMT) in Kirkland, WA.

Jeffrey D. Brown

Server Group Development

Mr. Brown is an IBM Distinguished Engineer in @server Hardware Development and has been influential in setting the direction of server hardware development for iSeries, pSeries, and xSeries system for the last five years. His knowledge of SMP system design covers all components within the Central Electronic Complex including processors, cache and memory hierarchy, and IO Subsystems.

Mr. Brown holds BS and MS degrees in Electrical Engineering as well as a BS Degree in Physics from Washington State University. He holds seven patents related to various aspects of SMP system and processor design.

Moises Cases

@server *xSeries Group*

Mr. Cases has thirty-two years of progressive experience in very-large scale integration (VLSI) chip and package designs, in system level electrical and package designs, and in complex project and people management. Presently, he is an IBM Senior Technical Staff Member and team leader for system electrical design and integration of @server xSeries Servers, responsible for signal and power distribution integrity, and system level timings for complex multiple board system designs. He obtained a Master of Science in Computer Engineering from Syracuse University, NY in 1979, Master of Science in Electronic Engineering from New York University, NY in 1973, and Bachelor of Science in Electrical Engineering from City College of New York, NY in 1969. He is a member of various Technical Program Committees for IEEE's CPMT, ECTC and EPEP groups. He is chairman of Modeling and Simulation Committee for ECTC and General Chairman of the 1999 IEEE Systems Packaging Workshop. He has received fourteen U.S. Patents, six IBM Invention Achievement Awards, seven IBM Authors Recognition Awards and four IBM Outstanding Technical Achievement Awards. Moises has published thirty-six external technical papers on numerous IEEE's proceedings and transactions. He is a member of Eta Kappa Nu and Tau Beta Pi.

Daniel J. Colglazier

@server *xSeries Group*

Mr. Colglazier is an Advisory Engineer in the @server xSeries performance group. He is responsible for determining the performance of future systems, concentrating on the memory hierarchy. He joined IBM in 1984 working on the performance of IBM's largest systems. He has received a patent and an Outstanding Technical Achievement Award for developing performance techniques and tools. He received a B.S. degree in computer engineering in 1983 and a M.S. degree in electrical engineering in 1984, both from the University of Illinois.

Daniel N. de Araujo

@server *xSeries Group*

Mr. de Araujo is an Advisory Engineer in the IBM @server xSeries High Speed Interconnect and Package Analysis group.

Mr. de Araujo has five years of experience in board and chip design, simulation, and validation in high-end servers and high volume consumer desktops. He joined IBM's Personal Systems Division in 1997 in Research Triangle Park, NC. In 2001 he moved to Austin, TX and joined the @server xSeries High Speed Interconnect and Package Analysis group. He obtained a Master of Science in Computer Engineering from North Carolina State University in 2000, Magna Cum Laude, and a Bachelor of Science in Electrical Engineering from Michigan State University in 1997, Summa Cum Laude. He is a member of IEEE, Tau Beta Pi, and Eta Kappa Nu, and has one publication.

Sudhir Dhawan

@server *xSeries Group*

Dr. Sudhir Dhawan, a Senior Technical Staff Member, has been working on the development of the Enterprise X-Architecture and the associated systems. Dr. Dhawan joined IBM in 1983 after completing his Ph.D in computer engineering from the University of New Mexico, Albuquerque, New Mexico. After joining IBM he has had responsibility for chip development for the I/O subsystem, memory and cache subsystem and processor. Dr. Dhawan has been named Master Inventor by IBM and has several inventions and publications.

Bob Drehmel

Server Group

Mr. Drehmel is a Senior Engineer working in the area of Intel Chipset Development. He has been a memory subsystem logic designer, team lead, and architect for several Server Group projects and now the Enterprise X-Architecture™ chipset. Mr. Drehmel has received two Outstanding Technical Achievement awards, one for his work on common memory controllers and another for S80 Chip Development. He has nine patents issued, five more patent applications filed, and 18 invention disclosures published. He joined IBM in 1985 after receiving a Bachelors degree in Electrical Engineering from the University of Minnesota.

Anthony D. Drumm

Server Group

Mr. Drumm is a Senior Technical Staff Member working on design automation systems, primarily logic synthesis and optimization. Anthony's interests include timing optimization, integration of placement and logic synthesis, and automation for high performance designs. Mr. Drumm joined IBM in Endicott, New York in 1977. He earned an M.S. degree in computer engineering from Syracuse University and a B.S. degree in electrical engineering from The Ohio State University. Mr. Drumm holds eight patents and has received an outstanding technical achievement award and an outstanding innovation award.

Todd Greenfield

Server Group

Mr. Greenfield is a Staff Engineer working in the area of Intel Chipset Development. His work on the Enterprise X-Architecture™ chipset includes the interrupt logic and cache memory interface. He joined IBM in 1996 after receiving an MS degree in Computer Engineering from Iowa State University in 1993 and a BSEE degree from South Dakota State University in 1991.

David L. Guertin

Server Group

Mr. Guertin is a Staff Engineer currently working in Rochester Circuit Technology on the design of I/O circuits for the CU-11 ASIC IO Library and custom I/O circuits for the Enterprise X-ArchitectureTM Chipset used in the NetFinity Intel based server systems. Since joining IBM Burlington in 1985, he has worked in the areas of I/O circuit design, embedded SRAM design, mixed-signal chip PD and custom image design. Mr. Guertin has two patent issues and one patent file related to performance compensation and di/dt control of off-chip driver circuits. He received his BS in Chemical Engineering in 1977 and BS in Electrical Engineering in 1984 both from the University of Minnesota..

Jim Haidinyak

@server xSeries Group

Mr. Haidinyak joined IBM in September 1980. He has worked in product development and test his entire career with IBM. He has held positions as a test and simulation manager in DASD controllers and numerous product development management positions in the RS/6000 division. He is currently a product development manager in the @server xSeries group, responsible for the development of the xSeries RXE-100 Remote Expansion Enclosure. Mr. Haidinyak received his MBA from the University of Phoenix in 1984, his BS in Electronic Systems from Southern Illinois University in 1982 and his AA in Electronics Technology from Southern Illinois University in 1980.

Jim Hanna

@server xSeries Group

Mr. Hanna is a Senior Technical Staff Member currently working on system design and development for xSeries servers. He joined IBM in Boca Raton, Florida in 1973 after receiving a BS in Electrical Engineering from Texas A&M University. Since moving to Austin in 1976, he has worked in product development on word processing systems, laptop PCs, desktop PCs and RS/6000s as well as his most recent work on high end xSeries servers.

Richard E. Harper

Mr. Harper has been a Research Staff Member at the IBM T. J. Watson Research Center since 1998. Previously, he was a Senior Technical Consultant at Stratus Computer in Marlboro, MA from 1995 to 1998, and a Principle Member of the Technical Staff at the Charles Stark Draper Laboratory from 1987 to 1995. He received his PhD in Computer Systems Engineering from the Massachusetts Institute of Technology in 1987, his Masters of Science in Aeronautical Engineering from Mississippi State University in 1978, and his Bachelors of Science in Physics from Mississippi State University in 1977. His interests are in the design, analysis, and implementation of highly reliable high-performance computing systems.

Russell D. Hoover

Server Group

Mr. Hoover is a Senior Engineer working in the area of Intel Chipset Development. His primary interest is in SMP memory subsystem micro-architecture. Mr. Hoover has received two Outstanding Technical Achievement awards for his work on RIO I/O link development and pSeries S80 chip development. He has 11 US patents and 18 disclosures published. He joined IBM in 1982 after receiving a BSEE degree from the University of Nebraska, Lincoln, NE.

Dan Hurlimann

@server xSeries Group

Mr. Hurlimann is a Senior Engineer in the System Development group responsible for the IBM xSeries 360. He joined IBM Corporation in 1984 after receiving a BS in Electrical Engineering from Georgia Institute of Technology. He is the lead in implementing the Enterprise X-Architecture and IBM XA-32 chipset into the IBM xSeries 360. Previously, Mr. Hurlimann was the development manager for the IBM Netfinity 8500R 8-way server. He has developed and managed various RS/6000 systems in addition to adapter development. He has a patent and numerous technical disclosures.

Joe Kirscht

Server Group

Mr. Kirscht is an Advisory Engineer working in the area of Intel Chipset Development. Prior to working on the Enterprise X-Architecture™ chipset, he worked on the memory controller design and was also responsible for the mainstore memory card design for the iSeries and pSeries S80 servers. He was also responsible for the mainstore card designs for two previous generations of the iSeries. Mr. Kirscht has one patent issued and one filed and two invention disclosures published. He joined IBM in 1989 after receiving a BSEE from the University of Minnesota. He received an MSEE degree from the University of Minnesota in 1996.

Randy S. Kolvick

Server Group

Mr. Kolvick is a Senior Engineer working in the xSeries High End Server Development, and is the x440 lead engineer. He was also the lead engineer on the x350 and 7000 M10 servers. His IBM career includes work on network adapters, switches, routers, printers, security, and banking products. His job roles included management, chip/card/system development, tools, card/box testing, and software. He has two patents issued. Mr. Kolvick joined IBM in 1982 after receiving a BSEE degree with honors from Georgia Tech in Atlanta, GA.

Jim Marcella

Server Group

Mr. Marcella is a Senior Engineer working in the area of Intel Chipset Development. His primary interest is in SMP memory subsystem design. He joined IBM in 1980 after receiving his BSEE degree from the University of Minnesota, Minneapolis, MN. Mr. Marcella has received two Outstanding Technical Achievement awards for work on memory controller designs for the iSeries and pSeries servers. He has four issued US patents and 28 disclosures published.

Timothy Moe

Server Group

Mr. Moe is a Staff Engineer working in the area of I/O ASIC Development. He joined IBM after receiving his Bachelor of Science in Electrical Engineering from the University of Minnesota in 1992. As an ASIC designer, he has worked on several I/O Hub and Bridge designs. Prior to his ASIC design work he was involved with system simulation and verification.

Kyle L. Nelson

Server Group

Dr. Nelson is a Staff Engineer working in the area of System Verification. He joined IBM in 1988 and has designed and tested processor microcode and worked in the system integration and test area prior to working in system verification. Currently he is responsible for applying formal methods to system verification. Dr. Nelson received a BA in Physics from Bethel College in 1988, a BS EE from Washington University in 1988, a MS EE in 1993 from the University of Minnesota, and a Ph.D. in ECE from Carnegie Mellon University in 1999.

Calvin Paynton

Server Group

Mr. Paynton is an Advisory Engineer working in the area of I/O ASIC Development. He joined IBM after receiving his Bachelor of Science in Electrical Engineering from the University of Missouri-Rolla in 1988. He has been an I/O ASIC logic designer and verfication specialist for several I/O hubs, bridges and memory controllers used in AS/400, iSeries, RS/6000, pSeries, and now xSeries with the Enterprise X-Architecture. He also served as team leader for the initial release of the IOB.

Nam Pham

@server xSeries Group

Mr. Pham received a BSEE from Michigan Technological University in 1988 and a MSEE from Syracuse University, NY in 1991. He joined IBM Microelectronics in 1988 as a semiconductor reliability engineer. In 1993, he joined the IBM Motorola Somerset Design Center in Austin as a PowerPC™ application engineer, and later joined the packaging and interconnect development for PowerPC™ microprocessor designs. Since 1999, he has been doing the package and system electrical design for the xSeries workstation development group.

Harry M. Schultze

@server xSeries Group

Mr. Schultze earned a BS degree in Electrical Engineering in 1972 and a MS degree in Electrical and Communications Engineering in 1973 from Clarkson College of Technology. In September of 1973, he joined IBM Kingston working in the communications network hardware area. He later moved on to workstation development, communications adapter development, and graphics development. In December of 1998, Mr. Schultze joined the xSeries group, managing the Enterprise X-Architecture™ Technology architecture area.

Robert Shearer

Server Group

Mr. Shearer is a Staff Engineer working in the area of I/O ASIC Development. He joined IBM after receiving his Bachelor of Science in Computer Engineering from Iowa State University in 1997. He also received a Master of Science in Computer Engineering from the University of Minnesota in 2001. He has one Outstanding Technical Achievement Award.

Dave Shedivy

Server Group

Mr. Shedivy is a Staff Engineer working in the area of Intel Chipset Development. Prior to working on the coherence logic of the Enterprise X-Architecture™ chipset, he worked on logic design and simulation model development of IO Subsystem chips for the iSeries and pSeries. He has received one US patent. He joined IBM in 1991 after receiving a BSEE degree from the University of Wisconsin, Madison, WI.

Nusrat Sherali

@server xSeries Group

Mr. Sherali is an Advisory Engineer in the System Development group responsible for the IBM xSeries product RXE-100 Remote Expansion Enclosure. He joined IBM Corporation in 1985 after receiving an MS in Mechanical Engineering from University of Pennsylvania. He received a MBA degree from SUNY, Binghamton in 1992. Mr. Sherali received many awards for his work on printed circuit technology development. In his prior position Mr. Sherali was a manager in the p-Series new product introduction development area. He also held several technical program management positions in @server manufacturing and development.

Tommy Tam

Server Group

Dr. Tam is a Senior Engineer working in the area of firmware development. He joined IBM in 1990 and has been POST/BIOS team lead for more than 10 Intel-based personal or server systems. His familar areas are multi-processor, memory and I/O subsystems. Dr. Tam received a B.S. in Mechanical Engineering from National Taiwan University in 1982, a M.S in Mechanical Engineering from Louisiana State University in 1985 and a Ph.D. in Computer Science from the University of Texas at Dallas in 1990.

Pete M. Thomsen

Server Group

Mr. Thomsen is an Advisory Engineer working in system development for xSeries servers. He joined IBM in 1991 and has been designing symmetrical multi-processor based systems since 1993. His area of concentration has been in processor and memory subsystem design and validation. He currently holds six patents with six additional patents pending. He received his BS EE in 1991 from the University of Minnesota.

Kenneth M. Valk

Server Group

Mr. Valk is an Advisory Engineer working in the area of Intel Chipset Development. He joined IBM in 1992 and prior to working on the coherence logic of the EXA chipset was responsible for designing the link protocol logic of the HSL I/O links of the iSeries and pSeries servers, and the clustering logic in the iSeries servers. Mr. Valk has received one Outstanding Technical Achievement award and three issued patents. Five more patent applications have been filed. He received a BS CSE in 1991 and a ME CSE in 1992 from Rensselaer Polytechnic Institute.

Cindy Walter

Server Group

Ms. Walter is an Advisory Engineer, in the IBM Server Group, at the Austin, Texas facility. She joined IBM in 1984 as a Manufacturing Test Engineer. She received an Excellence Award from the Personal Systems Group for her work in re-engineering the Assembly and Test process. In the Server Group, she has been responsible for managing the bring-up process of the first components in the EXA Chipset. She currently holds three patents, with four additional patent applications filed. Ms. Walter received a BSEE degree from Purdue University in 1983.

Scott Willenborg

Server Group

Mr. Willenborg is a Staff Engineer working in the area of I/O ASIC Development. He joined IBM after receiving his Bachelor of Science degree from Iowa State University in 1993. His career started in System Simulation and Verification moving to ASIC chip design in 1995. As an ASIC design engineer, he has worked on several I/O hubs and bridge chip designs, focusing primarily on bus interface and interrupt handling logic. Mr. Willenborg worked with the industry work group in development of the PCI-X local bus specification. He has two filed patent applications, both related to his work on the PCI-X bus interface.

Lawrence Whitley

Server Group

Mr. Whitley is a Senior Engineer, in the IBM Server Group, at the Rochester, Minnesota facility. He is currently working on hardware performance projects. After receiving a B.S. degree in electrical engineering from the University of Missouri at Columbia in 1969, he joined IBM and has worked on the design of processors, I/O, and system control programming for several IBM systems, including the System/32, System/34, and System/36. More recently, as a part of the design process, he has created performance models of processor memory subsystems, I/O subsystems, and I/O interconnection fabrics for the AS/400, RS/6000, and Netfinity servers. Mr. Whitley has received Outstanding Technical Contribution awards and holds three patents.

Curt Wollbrink

Server Group

Mr. Wollbrink is an Advisory Engineer working in the area of I/O ASIC Development. He joined IBM after receiving his Bachelor of Science in Electrical Engineering from Iowa State University in 1992. He has been an I/O ASIC logic designer for several I/O hubs and bridges used in AS/400, iSeries, RS/6000, pSeries, and now xSeries with the Enterprise X-Architecture. He also served as team leader for the IOB. He has one filed patent application.

David E. Wood

Server Group

Mr. Wood is an Advisory Engineer working in the area of Verification. He joined IBM in 1992 and has been a member of the verification team his entire career. He is now a verification team leader heading the verification efforts on various iSeries, pSeries, xSeries, and zSeries chips. He received a BS EE in 1992 from the University of Minnesota.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States and/or other countries:

Active memory
AS/400
IBM
IBM (logo)
IBM Enterprise X-Architecture
IBM XA-32
IBM XA-64
Memory ProteXion
RS/6000
S/390
ServerGuide
XpandOnDemand Scalability

The following terms are trademarks of other companies:

Intel	Intel Corporation
Intel Itanium	Intel Corporation
Intel Xeon	Intel Corporation
Java	Sun Microsystems, Inc.
Linux	Linus Torvalds
Linux RedHat	RedHat, Inc.
Microsoft Windows	Microsoft Corporation
Novell	Novell, Inc.
Novell Netware	Novell, Inc.
OpenLinux	Caldera International, Inc.
SuSE Linux	SuSE, Inc.
Unix	X/Open Consortiium
Unixware	Santa Cruz Operations (SCO)
WindowsNT	Microsoft Corporation
WindowsNT Enterprise	Microsoft Corporation
Windows 2000 AS&S	Microsoft Corporation

International Business Machines Corporation
11400 Burnet Road
Austin, Texas 78758

Printed in the
United States of America

For additional quantities contact
pkeeling@ca.ibm.com

59P6495