COMPARISON OF IBM AND Digital STORAGE ARCHITECTURES (including Vail)

29-March-1990

TR-9021, Storage Performance Engineering - MLDS/CXO

Abstract:

This report provides information on historical IBM storage achievements and compares the current architectures between IBM and Digital.

Revision/Update Information:

IBM 3390
VAX9000
RA70
RA92
Aspen

Robert Bauman & Dennis Haugh—Authors

Digital Equipment Corporation

Preface

This document provides a description and identifies the differences in the IBM and Digital I/O subsystems from an architectural perspective. Since architectures provide a framework, the limitations of each can be validly discussed. However, actual implementations must be analyzed for valid comparison. The details within this report attempt to isolate the effects of the architecture upon the implementation.

The target audience includes Digital internal personnel only. The level of detail is intended for marketing as well as engineering.

ACKNOWLEDGMENTS

Special thanks is given to Tom Kinlaw and Bernie Goldstein for their assistance in the KDM70 model. Don Matthews provided the illustrations and information for the Vail model. Kathy Woodard and Bernie Goldstein editted numerous flaws in grammar and spelling.

INTENDED AUDIENCE

The audience is: Digital product engineering - primarily SIMG, HPS, MSB, VMS, and ULTRIX; Digital product management and marketing; and sales support personnel. This document is for internal Digital use only; it is not intended for customers or third party product vendors.

CONTENTS

EXAMPLES

FIGURES

TABLES

1 EXECUTIVE SUMMARY

1.1 Scope

The purpose of this document is to educate the reader in the development of IBM's I/O architecture and to develop the ability to intelligently compare Digital to IBM I/O subsystem configurations and performance. The first two chapters of this report introduce the respective I/O architectures of each vendor. Chapter 1 describes the evolution of the IBM I/O architecture from the initial S/360 subsystem to the current 370/XA architecture. Chapter 2 introduces the various current designs of the Digital Storage Architecture (DSA).

In Chapter 3, the IBM and Digital I/O subsystems are compared at an architectural level. Device specifics are eliminated as much as possible. The delays quantified do not include seek time, rotational latency, or the queueing times produced directly by device characteristics. What remains are the delays inherent in the implementations of each architecture.

The comparative analysis is based on a set of 32 devices for the following reasons:

- IBM's DLSE groups strings such that 32 actuators are common to a set of four paths.
- Digital HSC70 can support up to 32 devices.
- Digital KDM70 can support up to 8 devices, hence 4 KDM70s are required.
- Digital VAIL model is based upon eight storage elements, each with four ASPEN actuators.

Appendix A details the effects of "RPS miss" under both architectures. RPS miss is the additional delay induced when no path to memory is available at the time the desired data is ready to transfer. Although this delay accounts for virtually all of the delays encountered in IBM's architecture, it provides only a portion of the delays in Digital implementations. Appendix B provides an assessment of the aggregrate delay for the Digital architecture. This topic is introduced in Chapter 3.

1.2 Summary

The current Digital and IBM I/O architectures are built upon opposite premises. Digital has built its DSA I/O architecture as a special case of communications. IBM views communications to be a special case of I/O. In practice, both have evolved to offload the host CPU of the I/O function by use of microprocessing within the I/O subsystem. Another significant disparity between the two architectures is that IBM's primary storage performance measurement is the largest amount of data delivered in the least amount of time (throughput to response time ratio). Digital's concentration, to date, has been focused on designing subsystems which are capable of delivering the maximum number of I/O requests per second.

In the Digital I/O architecture, a single CI has become too limited. Mixing a CI subsystem with other interconnects, such as XMI in the case of KDM70, allows a more "open-ended" architecture capable of competing with IBM in the single system space. The introduction of multiple host CI ports (VAX9000) is another solution. A single CI now appears topologically similar to an IBM channel. VMS software will address the routing to and from the multiple CI ports. IBM is still ahead in that the DLSE "channel subsystem" offloads routing overhead from the host.

The lack of caching in the Digital I/O subsystem is a serious implementation deficiency; however, it is not an architectural problem. The performance of a cache can be projected by reducing the request rate for a workload to the percentage of cache misses and adding the appropriate cache overhead for hits. Host-based software cache reduces the CI utilization and improves the head-on competitive position significantly. With IBM's introduction of ''fast write'' neither VAXcluster Cache (VCC) nor the HSC cache are sufficient—especially for workloads with a high percentage of writes (like transaction processing). The pending SDI extension to effect ''fast write'' should prevent the same competitive disavantage experienced from the long absence of caching. The VAIL strategy has the added performance advantage of having a nonvolatile RAM, instead of utilizing an adhoc technique with rotating media like the SDI extensions.

IBM's expanded storage is a serious hint at IBM's future directions. Although current 3090 implementations are limited to 2 Gigabytes, the Enterprise Systems Architecture (ESA) limit is 16 Terabytes. Synchronous access to expanded storage affords the opportunity to greatly reduce overall access time delays to rotating media. Single stream performance can be greatly enhanced by a significant reduction in paging times. The use of VAX host memory with VMS global sections currently provides a competitive answer, but the architectural capacity of expanded storage could allow for loading of entire databases as well as executable images.

CHAPTER 1

IBM I/O ARCHITECTURE - SYSTEM 360 TO SYSTEM 370/ESA

1.1 Introduction

This chapter describes IBM's I/O architecture which is defined as the features and functions of a system as seen by a programmer. This architecture will be shown to be evolutionary in nature. Looking back in time this evolution could be seen as a series of new developments for circumventing the limitations of previous generations. More accurately, it is the development of a method for performing I/O to processor external devices. This method has evolved in response to the demands placed on it by customers with growing businesses who required an architecture which could keep pace.

This chapter does not describe specific wire signals or component command streams necessary to perform I/Os. Control program (operating system) dependencies are omitted since these are outside the scope of the underlying I/O architecture. See the detailed reference materials listed in appendix *(POp1 through POp5)* for discussions on these topics. What will be covered is the historical foundation and the current 370 developments in IBM's I/O architecture.

For the last approximately 30 years IBM has continued to improve it's I/O architecture by gradually externalizing I/O management. Two versions of the architecture are implemented in the operating systems and hardware platforms marketed by IBM: System/370 and 370-XA. The fundamental structure of these architectures were initially conceived in the 1950s & 1960s and one of the objectives of this section is to explain those early developments, as well as, the 'why' of how those methods affect todays architecture. Current implementations of these architectures are:

- 937x
- AS400
- 438x
- 3090

Operating systems for these systems are:

- SAA
- VM
- DOS/VSE
- MVS

Both the AS400 and 9370 use emulator, or translator, boards to perform conversions from their internal I/O instructions to those required by the external storage devices sold with the systems. Both are basically 370 machines but the AS400 takes the layering a step further by separating applications from the underlying hardware and control program. This is accomplished by translating the generated applications into processor executable instructions at the

time of task initiation. This departure, from the typical dependency of operating systems and applications on the underlying processor architecture, allows IBM to incorporate new technology into the system package without impacting the users. In fact, IBM could place a 3090 into the core of the AS400, and the customer would be required to change nothing.

IBM's I/O architecture is general enough to handle terminal, tape, disk and forms I/O. In the case of disk, these devices are connected to and rely upon disk controllers, a physical and functional part of the Head-of-String. Disk controllers are attached to one or more Storage Control Units which direct the flow of data through storage paths. Storage Control Units are connected to channels, by way of transmission cables, which are connected to the processor. Controllers are function dependent which means that disk controllers manage disk drives, communications controllers manage their respective devices, etc. The components composing connecting I/O devices to the processor are defined in the following table:

Device	Function
IBM Channel	A card-on-board processor which accepts and executes Channel Programs build by applications or a component of the control program running in the CPU. Channel Programs reside in central storage. The result of a Channel Program processing is a Order for the control units.
Tag	One of two separate transmission cables which connects the channel to the control unit. Maintains channel synch signalling with the CPU.
Bus	The other transmission cable; connects the channel to the control unit and carries interleaved byte or interleaved block data.
Storage Control Unit	Receives and executes orders from channels such as 'Seek Sector 00125 on device 0125'.
Head-of-String	Contains the Read and Write control logic for attached disk storage devices.

1.2 A Short History of IBM's I/O Architecture

1.2.1 *Storage Organizations and Early Channel Architecture*

Storage is and always has been a component, or subordinate of the memory system that serves the processor. Central, Expanded, Disk and other storage medias functions by inexpensively storing information which will be used to keep the processor busy. The expense is relative to how long the information will be stored and how quickly it will be needed once requested by the processor. When data is requested it is requested using some addressing scheme designed to expedite the retrieval of information. When accessing main memory a read or write instruction is directed to a specific location within the central storage complex. Accesses to disk involve an address which corresponds to a record, sector, track or cylinder assigned for data recording.

1.2.1.1 *1954 - Drum Memories*

Prior to the introduction of magnetic cores, magnetic drums were used as central storage. These drums are analogous to disk but had one head per track and rotated at about four times the speed of todays disks *(Haml)*. The IBM 650, introduced in 1954, used a magnetic drum for it's main memory. After the introduction of core memories these drums were delegated to the role of secondary storage device.

1.2.1.2 *1956 - Disk Storage*

IBMs first disk system, the 350 introduced in 1956, organized the stored data in much the same way as it was in memory. The data was stored in fixed length words of 100 characters. Each of these words had an unalterable address associated with it which was stored just prior to the data area. When an area of the storage media was needed for data transfer the hardware performed repeated read and compare operations (Search operations) until the address was found. The processor remained disconnected during this search because the disk control unit served as the search initiator/controller. Once the address was found the data was either read or written sequentially with the controller acting as speed matching buffer by collecting the characters into a block for transfer across the channel. Speed matching buffers are used when there is a disparity between the device transfer speed and the system it's attached to.

1.2.1.3 *1957 - Data Synchronizers*

In 1957 I/O channels were introduced with the IBM 709 processor and permitted the concurrent execution of I/O and CPU operations *(IBM1)*. The channels, termed data synchronizers, acted as I/O processors with specialized I/O instruction and allowed up to six I/O devices to access the processors memory buffers. Memory access was independent of the program running in the CPU and permitted each channel to store and retrieve data directly. Instructions, executed by the CPU, coordinated functions to eliminate conflicts between the processor and channels. For each data synchronizer attached to the 709, a proportionate number of I/O devices could operate. The reason for this is that the path between the synchronizer and the storage device was busy throughout the entire operation, including seek, rotational delay and transfer.

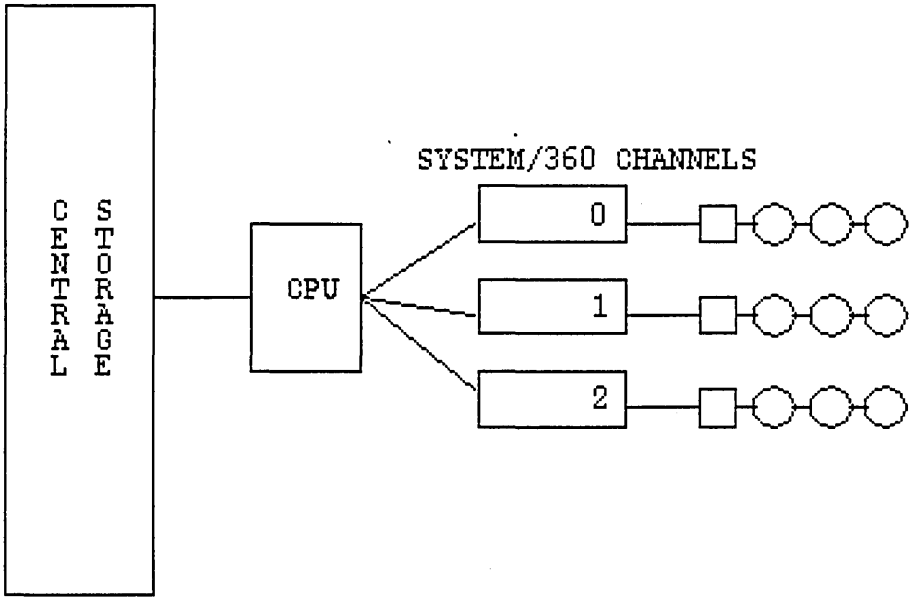
The 709 was also the platform on which IBM's first I/O supervisory program was introduced. This I/O Control System (IOCS) eliminated the need to re-design the synchronization and control procedures when new applications were developed. In other words, previously these procedures were written into the applications using instructions built into the CPUs architecture. I/O completion was tested for by repeatedly executing loops of code which tested the data synchronizers state. With the 7090 processor (1958) IBM implemented I/O interrupts which caused the CPU to branch to an instruction sequence designed to process the interrupt. Interrupts occurred at the completion of an I/O or as the result of some unexpected condition in the storage subsystem. The importance of this development is that it eliminated the 'loops of code' which further promoted the asynchronous execution of I/Os.

Another complexity in these early systems was with the organization of data on disk devices. The disk themselves were organized into a series of fixed length sectors each with a unique address. The programmer wishing to save data on a disk drive was required to hard code the physical address of the record locations in the form of cylinder/track/record. There was no volume index or volume table of contents to protect one programmers data from being over-written by another.

1.2.2 *System 360*

The introduction of System/360 in 1964 embodied a corporate strategy that specified that architectures and implementations would be separate. This architecture provided a general purpose structure to support varied processing demands and to work with current and future I/O equipment. System/360 standardized the aspects of I/O attachment and control by introducing a common interface, a comprehensive interruption system, storage protection, and uniform program control. The common interface prescribed the procedure and specification for attaching I/O devices and was published in an I/O architecture principles of operation *(POp5)*. The following logical diagram depicts the typical System/360 organization:

Figure 1-1: System/360 Channel Architecture



This System/360 standard was the foundation for future platform and device implementations. These implementations would introduce new disk storage control units, microcode controls, disk storage devices, device allocation and storage management software. A channel subsystem, originating with the 709s data synchronizer, was designed into the system to promote concurrence with I/O and CPU operations. I/O interrupts were also taken from the 709s architecture. The System/360 I/O architecture would allow up to 256 channels, each with up to 256 devices attached.

1.2.2.1 *Channel Developments*

System/360s specification stated a reliance on DASD for storing operating system, job control routines, and data. Customer acceptance of this media as a reliable and expedient retrieval mechanism was greatly underestimated. Users soon exceeded the capabilities of the subsystem architecture to meet the throughput and response time demands. One of the limitation of System/360 channels is that they were dedicated during the execution of I/O operations. In the original implementation of the architecture to perform a read, or write, three separate operations were required: seek, search, and data transfer. During the search operation, the channel, interface, and control unit were held open while the search argument was repeatedly

passed from the channel to control unit. While performing the read/write, the System/360 could not connect through the channel to control unit path.

Another problem in early implementations of the architecture was a relative inefficiency with the Input/Output Supervisor (IOS). IOS was run in the central processor and initiated and terminated I/O operations. The IOS disabled interruptions while handling specific functions which impaired I/O. For multiplexer channels, a queue of interruptions built up as I/Os reached completion. Selector channels, which were data streaming single path pipelines, were held open and incapable of performing new operations while waiting for IOS to handle an outstanding interruption which would make the channel available again.

The following ideas were suggested to resolving the I/O problem:

- To use a large record buffer in the control unit that would hold data transferred from the device until the channel became available and then move the data to the host across a selector channel. This would result in a reduction of RPS miss accumulation and provide earlier drive availability.
- To use an I/O Processor with its own instruction set in which IOS would run and have a direct link with main storage, were the programs, data, and channel control information would reside. The functions performed by this IOP would include record blocking/deblocking, error recovery, and address translation. An IOP would also relocate the interrupt lock-out problem.
- To use a block multiplexer channel, which would offer high data rates and block interleaving along with new functionallity for the storage controllers.

The block multiplexer channel was chosen for its performance under the scrutiny of simulators and its cost of development and implementation. New functionallity for control units introduced sector addressing, which allowed the control unit to monitor DASD for a selected sector before beginning the search for the requested record. This replaced the current addressing method of cylinder and track and allowed the control unit to disconnect from the channel while waiting for the appropriate sector to become available. Reconnect then took place and the requested record was transferred to/from the channel.

The block multiplexer channel proposal required a new channel design and a new control unit. The proposed channel would posses a high data rate, and allowed a degree of multiplexing. Logical disconnection of the device from the channel would be permitted between blocks and occurs only if a significant delay is anticipated before another operation can be executed by the device.

The IBM 2880 Block Multiplexer Channel was architected to transfer data at 3.0 Megabytes per second (limited by signal and interface circuitry) while interleaving blocks of data from multiple subchannels. Subchannels are subordinate entities to the channel which act like an independently operating processor that can sustain its own channel program. Subchannels are virtual components of channels and analogously fills the role of an address space in a virtual processor. Each attached I/O device would be activated and managed by a corresponding subchannel.

The System/360 Models 85 and 195 were the first to incorporate the 2880 block multiplexer channel as a common method of attaching and programming all I/O devices. The channel was particularly suited for rotation position sensing devices. The block multiplexer channel was used with rotation position sensing, it permitted a subchannel to be assigned and a channel program to be established for each access arm, with each program monopolizing the channel

for the duration of data transfer. Channel facilities were released during arm movement and during the rotation delay associated with locating the designated record. This channel introduced a third type of channel. Each type would permit different levels of concurrence among channel programs. The following table lists the various types of System/360 channels:

Channel	Description
Selector	1 Subchannel that permits one high-speed transfer of streaming blocks
Byte Multiplexer	256 Subchannels for low-speed transfers of interleaved blocks
Block Multiplexer	256 subchannels for high-speed transfers of interleaved blocks

1.2.2.2 *RPS, Request Queueing & Other Channel Exploitations*

RPS storage was exploited with System/360. RPS storage had to be divided into sectors. The number of sectors per track had to be fixed for each device but could vary among models. During an operation, the 1-byte sector number was sent to the DASD by a new command called SET SECTOR. When it received the sector number, the control unit logically disconnected from the channel until the desired angular position was reached or was about to be reached. Reconnection was then attempted. Monitoring was performed by the disk device with the control unit communicating the disks status with the requesting channel. During disconnection the channel was free to initiate request of attached devices and the control unit was available to accept request from the channel, hence multiplexing. The sector number could be obtained in two ways: When the records on the track are of a fixed length and format, the sector can be derived from the track capacity and the sectors per track. Alternatively, when a record is read, written, or searched, its sector can be derived by a READ SECTOR command.

During SEEK operations similar events took place. After the device was instructed to position the read/write heads the control unit would disconnect from the channel. The I/O channel was then free act upon work for other devices and the control unit became available to handle work for other attached disks.

The request queueing facility of the IBM 2305 used the block multiplexer channel and the rotational position sensing concept by allowing up to eight operations to simultaneously process for a single device. The device was assigned eight device addresses to which operations could be arbitrarily directed. The control unit for the 2305 effectively sorted these operations so that they were handled in the order their respective sectors became ready.

IBM also introduced the dynamic channel selection facility on the System/360 Model 67. The Model 67 was a two processor CPU complex which required that specific channels be dedicated to specific CPUs or that the channels could float. Operating in an extended problem state this system could access all channels and accept I/O interruptions on either of the two system processors. Sounding remarkably like components of the 370-XA architectural specification these facilities were not promoted in System/370.

1.2.2.3 *I/O Interface Improvements*

The I/O interface also required improvements to support the demands of the block multiplexer channel. The I/O interface connects a channel and an I/O control unit and it provides physical and electrical specifications. The original System/360 I/O interface specification was adequate for data rates up to about 1Mbyte/sec for a 100-foot cable. For 20-foot cables the IBM 2301 Drum Storage, with a rate of 1.2 M Bytes per second could be accommodated. Fully interlocked signalling allowed one channel cable connection to sustain data transfer over a range of rates, with both channels and devices controlling the timing of each byte transferred. It did, however, require an electrical signal to be propagated between the channel and the control unit four times for each byte transferred.

Block multiplexer channels, in order to meet the required through-put, required 2 bytes more data transfer capacity and a tightening of the electrical specification for signalling. These improvements were introduced in June, 1970 with the first implementation of the System/370 in the models 155 and 165.

1.2.2.4 *Channel Programming*

Early I/O devices were accessed and controlled by the CPU *(BUC1)*. The CPU instruction set contained specific operation codes for operating and controlling the device. The System/360 and System/370 architectures took these instructions out of the CPU and gave them to smaller special purpose processors - channels. Channels have their own set of instructions, known as channel commands, which are stored in central storage and are fetched, decoded and executed by the channel in association with the target I/O device. The CPU starts the I/O operation by issuing an instruction which gives the first address of the first channel command. Since the specialized I/O commands had been moved off the CPU then only one instruction was required for all I/O operations. The op code of the instruction specifies that the channel should asynchronously Start I/O (SIO).

This method of I/O control is known as channel programming and involves the set-up of a sequence of commands for a channel to execute. The Start I/O (SIO) CPU instruction initiated the channel to find the list of channel commands (CCWs) to execute. These commands direct the channel to the result required by the application; such as read a record. Three key fields are involved:

- Channel Address Word (CAW)
- Channel Status Word (CSW)
- Channel Command Word (CCW)

System/360 removed the I/O subroutine address from the instruction and placed it into a fixed memory location termed the Channel Address Word. This word had to be loaded with the correct subroutine address prior to the execution of the Start I/O operation. What this accomplished was that it allowed I/O devices, their instructions, and the channels controlling the devices to change without affecting the CPUs architecture. The CSW was used to contain channel status information. Execution of an SIO led to an ''initial interrupt'' where the CSW was queried to ensure the channel was accessible. The CSW also was used to contain the completion status for an I/O operation. The CCW provided both the commands—like read and write—and the data pointers for the channel to process. Execution was sequential from the initial CCW pointed to by the CAW unless a Transfer In Channel (TIC) command was encountered to effect a branch.

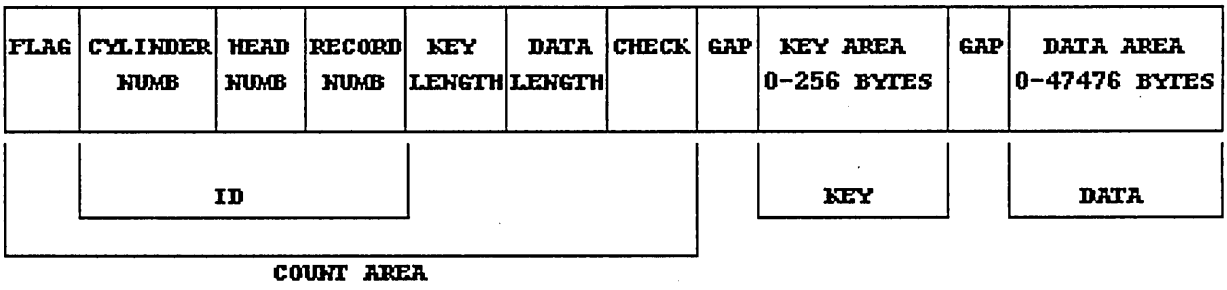
1.2.2.5 *Count-Key-Data*

Another introduction with System/360 was the disk allocation methodology known as Count-Key-Data. CKD organization was introduced with IBM first microcode based storage control unit the 2841. The 2841 control unit provided device dependent interpretation of channel commands and would adapt to the disk record format specified in the 360 architecture. CKD is the method employed by IBM to implement Store Addressing Information *(MAT1)* which permits record level granularity when accessing data.

For example, a physical track can hold as much as 56KBytes of data, but most application records are significantly smaller. In order to directly access these smaller records some greater degree of granularity is required. To implement the finer addressing at the disk level requires that the algorithm be incorporated into both the I/O architecture and the systems access methods. Store Addressing Information is simply a defined address stored on the disk in predefined areas designated by gaps that contain special codes. Logic circuitry into the controllers sense the gap codes and repeated compares can be done until the specified record is located. Since the Store Addressing Information can be in any of the various gaps, between physical sectors, the track can be logically divided into fixed or variable length records each of which contains addressing information in it's header. A significant difference between CKD formatting and FBA is that the former permits the physical record length to be determined by the application allocating the space for storage.

With CKD formatting each record contains a count, key and data areas. The count area holds the physical cylinder, head and track relative record number along with the length of both key and data areas. The optional key area can contain a record index and is used by the control unit to automatically locate the record during SEARCH operations. The data area contains a record limited in size by the data capacity of the track. The following illustration outlines CKD formatting:

Figure 1-2: Count, Key and Data Areas



1.2.3 *System 370*

From the standpoint of I/O, if one thing could be said about the System/370 permutation of the architecture it would be that it was an extension. System/370 added to and improved upon the initial conception of what the I/O design should contain. In 1962, IBM knew that the interruption mechanism and the I/O control formats did not have the required extensibility needed for an evolving architecture. However, costs and performance consequences prevented them from improving the System/360 platform. With the improvements in hardware technology during the late 60s, these facilities became feasible. System/370 was an evolutionary extension of the System/360 architecture for a new set of models and for new releases of programming systems. The objectives of the architecture were:

- Eliminate bottlenecks
- Improve efficiency
- Attach to and operate System/360 I/O devices

The architecture would accomplish this by incorporating the following extensions to System/360s foundation:

- Block Multiplexing
- Command retry
- High-Speed data transfer

Bottlenecks existed in the data request facilities and in the ability of the I/O subsystem to deliver that data expediently. These problems would be addressed by incorporating additional data access methods in the system design. Channels and I/O interfaces posed difficulties; processor speeds were outpacing the ability of the storage to deliver data. New controllers were needed to fully exploit the benefits of microcode-controlled storage directors and the inherit efficiency of Count-Key-Data. In essence the new architecture would bring the complex components back into balance with each other while protecting existing customers investments in subsystems hardware.

1.2.3.1 *Access Methods*

Even though this is a departure from the prescription of outlining architectural developments the subject of access methods is important because they eliminate dependency on the design specification. System/360 introduced a series of facilities which allowed users to eliminate the hardcoding of stored data addresses (cylinder, track and relative record) from their access routines. System/370s involvement in this was to standardize by incorporating them into the architecture and to remove the requirement that programmers understand the physical geometry of the storage devices. This information is still used, the programmer is simply insulated from it. The access methods introduced with System/360 were *(CLA1)*:

Acronym	Definition
BDAM	Basic Direct Access Method
BPAM	Basic Partitioned Access Method
SAM	Sequential Access Method
ISAM	Indexed Sequential Access Method

Both the Basic Partitioned and Indexed Sequential Access Methods were new while the others were improvements on hard coded access routines already available. BPAM was designed to eliminate the CKD allocation inefficiencies of small record length datasets; multiple files of 80 byte records could be organized under a single file structure. ISAM was designed to utilize the Index area of the CKD. Both BPAM and ISAM handled the translation of the logical record identifier to a physical address (Cylinder, Track, Record) on disk.

These access methods replaced macro functions in IBMs initial I/O supervisor (IOS). The IOS had disk management functions which were basically an extension of those used for tape (SAM). It did provide for the management of buffers and the blocking of logical records to physical tracks. BDAM allowed for direct access but required that the application provide the physical address of the record(s) by transformation of the logical record identifier to the physical device address before calling IOCS *(BUC2)*.

1.2.3.2 *I/O Interface*

The original System/360 I/O interface specification was adequate for data rates up to about 1 Mbyte/Sec. In special cases for disk devices and for very short cable lengths, a rate up to 1.25 Mbytes/Sec could be supported. Storage technologies employing higher recording densities and buffered devices required higher rates. This required an electrical signal to be propagated between the channel and the control unit four times for each byte transferred. 360 changed the width of the interface and the interface signalling to assist in the expansion of these data rates. Fully interlocked signalling on the I/O interface allowed one channel cable connection to transfer data at a range of rates; both the channel and device controlled the timing of each byte transfer *(AD01)*.

The 2880 Block Multiplexer Channel introduced with the System/360 Models 85 and 195 implemented these signalling improvements. The System/370 I/O interface introduced two additional tag wires *(3601)* to provide the same level of transfer interlocks with only two propagation times per byte transferred. Depending on the control unit this facility allowed for the migration of System/360 I/O devices to System 370 processors. Control units implemented to operate with the System/360 interface can be attached to System/370 channels. On some System/370 channels and control units the bus cable width could be extended optionally to two bytes, doubling the data transfer capacity. As a result of these two additions, the System/370 I/O interface could sustain a data transfer rate of over 1.5 Mbytes/Sec in the 1-byte version and over 3Mbyte/Sec in the 2-byte version, over a 100-foot cable *(DTB1)*. Longer cables had to sacrifice transfer speed.

The cable modification added two wires to the tag interface to provide the same level of transfer interlocks at the expense of only two propagation times per byte transferred. The facility used depends on the control unit, so that control units implemented to operate with the System/360 interface can be attached to System/370 channels. The basic bus interface is 1 byte wide comprising 8 data bits and 1 parity bit. On some System/370 models, the bus width can be extended optionally to 2 bytes, thus doubling its data transfer capacity.

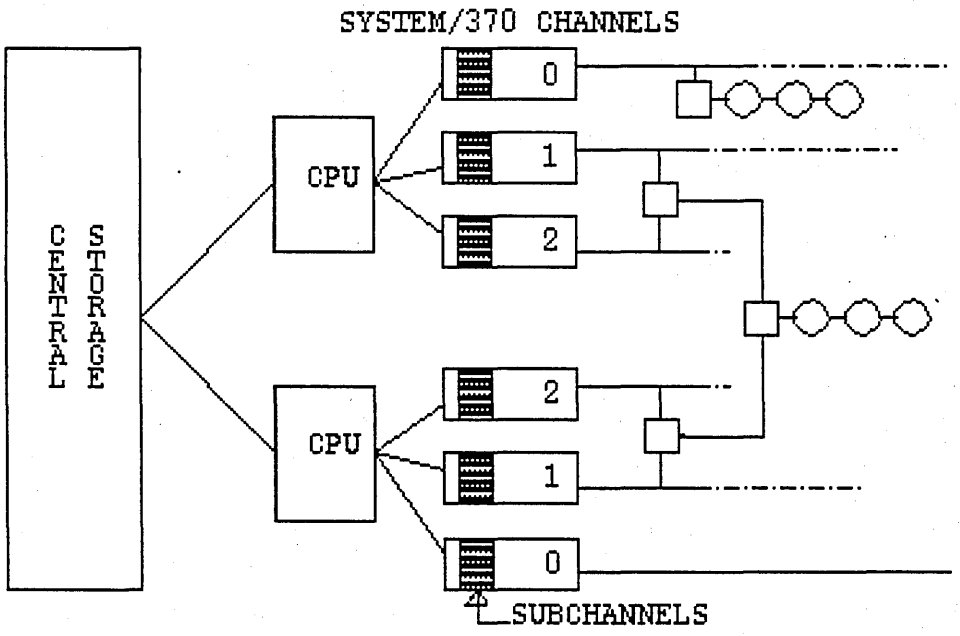
The data streaming mode, introduced with the IBM 3380 Disk Storage, further eliminated the interlocks between the request and response signals during data transfer. Data, with the appropriate tag signals, were sent in the form of fixed length pulses. *(3601)*. Thus, the data rate no longer depended on cable. The IBM 3380 specifications provided a transfer rate of 3M bytes per second over 400 feet with the 1 byte interface.

1.2.3.3 *Channel Structure*

In System/370, a single physical path usually exists between a channel and the attached control units. The IBM 2870 channel is an exception with five paths to attached control units. The physical path is referred to (and still is in 370-XA) as the System/360 and System/370 I/O interface. In System/370, communication between the channel and I/O device occurs by using a subchannel, the physical path, and the control unit. Communication between another channel and the I/O device requires a different subchannel, physical path, and control unit (or the same control unit when a two-channel switch is used). The need for separately identifying the physical path is unnecessary; therefore the term channel also implies the physical path. A channel is distinguishable from another channel by three characteristics: a unique address, a separate set of subchannels for attaching up to 256 I/O devices, and a connected CPU.

In System/370, the channel-to-CPU interface is uniprocessor oriented. Channels are provided in sets attached to a CPU. A System/370 channel can be addressed only by the CPU to which it is connected. Also, a System/370 channel can interrupt only that CPU to which it is connected. In a System/370 multi-processor (MP) system, each CPU has its own set of channels. Thus, the operating system in an MP system must ensure that the correct CPU performs I/O operations with I/O devices that are not attached to channels on each CPU. When a program initiates an I/O operation, the path over which the operation is to take place is specified as part of the START I/O or START I/O FAST RELEASE instruction. The I/O operation either is initiated on that path or is not initiated. If the operation is not initiated, the program is notified, and the program may initiate the operation on a different path to the device, if one is available. The following diagram illustrates the layout of 370 I/O:

Figure 1-3: System/370 Channel Architecture



The channel-to-I/O device interface is single path oriented in another way. Once a chain of I/O operations is initiated with a device, all data, status, and commands for the chain of operations must use the physical channel path over which the first command was initiated *(IBM1)*. In particular, if a device disconnects from the channel path during a chain of commands, as when block multiplexing occurs, the device must reconnect to the same path to continue

executing that chain of commands. If the path to which the device must reconnect is in use when the device is ready to reconnect, the device must wait until the path is free before it can reconnect to continue execution.

1.2.3.4 *Input/Output*

System/370 architecture adds several facilities and functions to input/output (I/O) operations to improve channel utilization, to make control operations more efficient and flexible, and to increase the maximum data rate on the I/O (channel-to-control-unit) interface. This section discusses some of the more important additions.

The System/360 architecture provided for two channel types: a selector channel, and a byte multiplexer channel. The development of the block multiplexer channel for the System/360 Model 85 and 195 and its subsequent standardization in the System/370 architecture added both a high data rate and multiple-device capabilities.

The block multiplexer channel is similar to the byte multiplexer channel in it has a number of subchannels, each associated with an I/O device or a group of I/O devices. The subchannel is the logical entity that controls an I/O operation and contains the addresses, count, and control bits associated with the operation. The channel provides the data paths and controls for communicating with the CPU, main storage, and I/O control units and for associating the proper subchannel with each communications sequence. The difference between the block and byte multiplexer channels is in the level of multiplexing. The byte multiplexer channel can interleave the transfer of individual bytes for different subchannels. However, the block multiplexer channel, designed for high data rates, is limited to interleaving complete blocks of data.

The block multiplexing capability is advantageous when used with rotational position sensing on rotating storage devices, such as disks and drums. The device disconnects from the channel during rotational delay, thereby releasing the channel. When the addressed sector is approached on the track, reconnection is attempted for the transfer of data. If the connection cannot be established when the sector is reached, another attempt is made after a delay of one rotation time. Rotation position sensing is available, for example, on the IBM 2305 fixed head file server *(3602)*. The control unit for this file can appear to have 16 devices, each associated with its own subchannel which can sustain an I/O operation.

Without the block multiplexing capability, I/O facilities required separate START I/O instructions to specify the position of the arm on the disk and the subsequent reading and writing. On the block multiplexer channel, these commands are chained. Thus, the CPU is not interrupted when positioning is complete and the number of instructions is reduced.

Because the block multiplexer channel transfers blocks of data during CPU operation, a new interruption, the channel available interruption, was required to indicate when the channel was free to process a new request. The block multiplexer channel generates this signal when the busy condition no longer exists. The signal is sent to the originating CPU.

A HALT DEVICE instruction also is introduced largely because of the block multiplexer channel. It is similar to the previously available HALT I/O except that, when the channel is busy, only the operation on the addressed subchannel is affected. HALT I/O terminated the current burst operation on the channel and ignored the device address.

The new CLEAR I/O instruction permits freeing the subchannel associated with the addressed device while I/O operations take place at the device. This function is useful for situations involving machine error or reconfiguration of I/O devices and control units.

Finally, an extension is provided to reduce the CPU time to start an I/O operation. When START I/O (SIO) is issued, the channel signals the device to check if the device can execute the command. This involves a number of signal sequences and the associated propagation delays and logic delays in the channel and the control unit *(POp5)*. According to the I/O interface specification the portion of the total delay introduced by the circuitry in the control unit can be as high as 32 microseconds. Additional delays may be introduced by the channel. On a CPU that can perform a few million average instructions per second, the delay from communications with the device can equal a hundred or more instruction executions.

The new instruction START I/O FAST RELEASE (SIOF) *(POp2)* allows the acceptance to be signalled and the CPU to be released as soon as the channel has fetched the channel address word from main storage. The channel subsequently initiates the operation at the device and verifies the command information. Exceptions are signalled by interruptions. Normally such exceptions are infrequent and overall little time is spent processing the interruptions. Some channels, available with early 370 processors and on current 9370 processors, implement the early release on SIOF and instead execute SIOF as SIO. Such implementations are compatible and permit the same program to run with either channel developments.

Most System/370 channels provide the command-retry facility, whereby the channel, in response to a signal from the device, re-executes a channel command. This re-execution is usually invoked when the device or control unit detects a malfunction.

1.3 System 370/XA

1.3.1 System 370/XA - I/O Architecture

The first IBM processor product to implement the 370/XA architecture was the 3081. The 3081 Processor increased the I/O channel capability with two central processors. Each processor accessed channels and central storage by a single system controller. This direct path to central storage permitted dynamic reconnection and eliminated the need for channels to reconnect to the CPU to which they were attached. The system controller moved data between the channel subsystem and central storage, and into the 32K-byte buffer associated with each central processor *(RNG1)*.

The 3081 supported up to 24 channels, which were either byte multiplexer or block multiplexer type. A maximum of four byte multiplexer channels could be configured within the 3081 processor complex, each having an aggregate data rate of up to 500K bytes per second with a burst size of 32 bytes. All block multiplexers had data streaming capability that permitted data rates up to 3M bytes per second per channel. Channels were assigned in two channel sets, one for each central processor, with a maximum of 16 channels per set.

Channels were controlled by the external data controller, an integrated I/O processor within the 3081 processor unit. The external data controller consists of two types of microcode controlled elements.

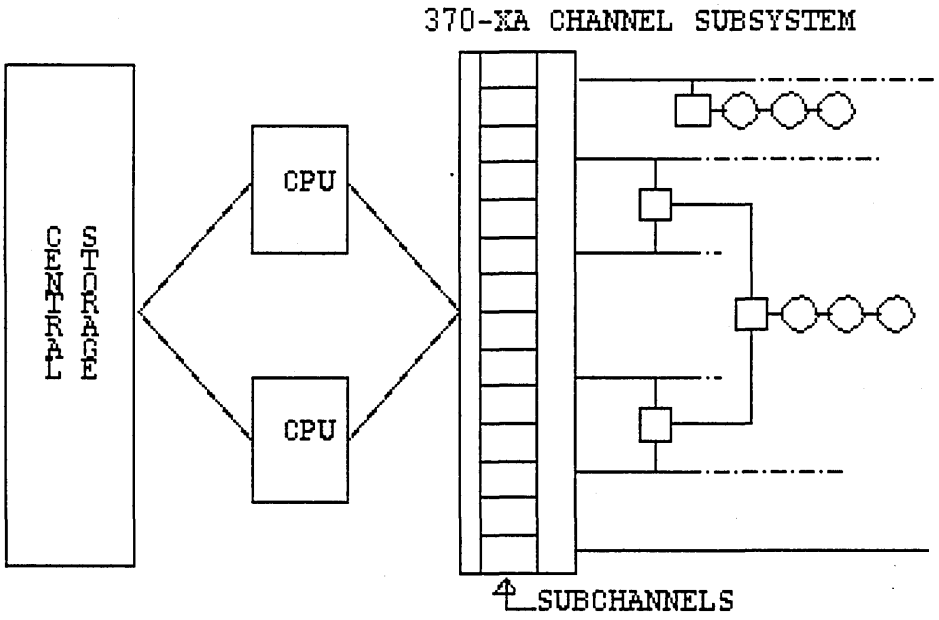
- The channel processor element (CPE) is a special processor which controls I/O instructions and interrupts. It supports a queued interface with the central processor for START I/O FAST RELEASE operations and for I/O interrupts, and handled command/data chaining and all external data controller recovery operations. The channel processor element is driven by vertical microcode having a 2-byte microword and an addressing capability of 32K microwords. A 2K-microword write-able static array contains the most frequently used microwords. A second 256-microword array contains 4 blocks of 64 microwords each. The array is dynamically loaded on demand from a hardware system area located in a portion of central storage. The processor has a 2-byte data path structure and is packaged in one thermal conductor module (TCM) *(RNG1)*.
- The data server element (DSE) handles the control sequencing and data buffering for eight channels. It is packaged on a single TCM, and a maximum of three Data Server Elements can be attached to the CPE. The Data Server Element contains 256 bytes of data buffering per channel. It supports 2-byte transfers with the interface adapters and 64-byte transfers with central storage. The DSE is controlled by horizontal microcode having a 54-bit microword in a 750-microword array. The eight channels are controlled by DSE microcode that is time-shared on an equal round-robin basis. Outboard of the DSE, packaged in a card-on-board technology, each channel has an interface adapter element which drives the I/O interface and contains eight bytes of data buffering.

1.3.2 *What 370/XA did for IBM's I/O Architecture*

370-XA replaced the System/370 I/O functions with a structure that moved I/O management out of the CPU with a completely queued interface between the CPU and the channel subsystem. The 370-XA architecture increased the performance of the large scale systems and structured the I/O architecture to take full advantage of the 3081 and subsequent tightly coupled multi-processors.

The 370-XA channel subsystem allows CPUs the same access capability for all devices attached to the system. The subsystem became a separate component of the processor complex. I/O devices were attached to the Subsystem, rather than to the CPU, as in System/370. This removed the limitation of CPU's initiating I/O functions and accepting interrupts, only to those devices to which they were physically connected. The new architecture allowed any I/O function to be initiated with any I/O device, regardless of the CPU executing the I/O instruction or the physical attachment of the I/O device to the channel subsystem. The channel subsystem was not identified by an address, and permitted attachment of enough I/O devices to correspond with 256 System/370 channels. However, all of the I/O devices attached to the channel subsystem were represented by a single set of subchannels. These characteristics present the appearance of a collection of channels, or what is termed a "channel subsystem." The following illustration shows the 370-XA structure:

Figure 1-4: 370-XA Channel Architecture



The separation of the channels from direct CPU control allowed the following prominent features:

- The channel subsystem architecture allowed devices, like the 3380-AA4, to dynamically reconnect to a free path.
- In 370-XA all I/O busy conditions were handled by the channel subsystem rather than by the CPU program, as in System/370.
- In 370-XA, alternate path are scheduled by the channel subsystem rather than by the CPU program.
- In System/370, channels must be handled by type; that is, there are differences in program action depending on whether a device is attached to a selector channel or a multiplexer channel. In 370-XA, the type of channel path used is not apparent to the CPU program.
- In 370-XA, eight different physical paths can be used when communications takes place between a single subchannel and an I/O device.

- In 370-XA, any CPU in the system enabled for I/O interruption can accept an interruption from any subchannel.

1.3.3 *Input/Output*

Communication between the control program and the channel subsystem regarding an I/O device depends upon the use of a subchannel number. The channel subsystem and the I/O device communicate by using a device address. The subchannel number identifies the target subchannel during the execution of I/O instructions and during the handling of I/O interruptions. The device number is assigned during installation of the I/O device and bears no relationship to the physical attachment of that I/O device. Compatibility of addressing between the channel subsystem and the I/O device has been maintained from System/370 to 370-XA since the device number is still used in 370-XA for administrative or operator communication purposes. Separating device number and device address allowed customers complete freedom when assigning device numbers.

In 370/XA, each I/O device is assigned to a different subchannel. Architecturally the device number has no relationship to the device address used in the communication path between the channel subsystem and the I/O device or the channel path to which the I/O device is physically attached. Consequently, physical addressing changes can be made between the channel subsystem and the attached I/O device, without impacting the control program. Subchannel numbers must be assigned in contiguous ascending order, starting with zero. This presents a problem in multiple system installations with shared DASD because it removes what might otherwise be a convenient system unique identifier.

The channel subsystem, during initiation of an I/O function, tests the availability of channel paths to the associated I/O device. The testing result reports one or more available channel paths. One of these paths is selected during initiation of an I/O function. If a busy condition occurs, an alternate path, from the set, is chosen. If another busy condition occurs, another path is selected if one is available. This function is performed without the involvement of the control program.

I/O interruption requests from individual I/O devices can be assigned to any one of eight maskable interruption subclasses. In effect this created interruption subclasses assigned to I/O devices. Masking of these subclasses is provided by use of a control register in each CPU. Subclass assignments are made to each subchannel during the execution of MODIFY SUBCHANNEL. Under System 370, a similar feature was designed into the channel subsystem. However, all of the attached I/O devices (to 256) were likewise being masked. In 370-XA, assignment of subclasses to subchannels provides greater flexibility in controlling I/O interruptions from I/O devices. For example, a software-controlled priority-interruption methodology can be employed where only the lowest priority programs are executed with all subclasses enabled, and successively higher-priority programs are executed with fewer subclasses enabled. As a result, low-priority programs can be interrupted by all I/O devices, but high-priority programs can be interrupted by only a few devices *(POp3)*.

1.3.4 *Monitoring*

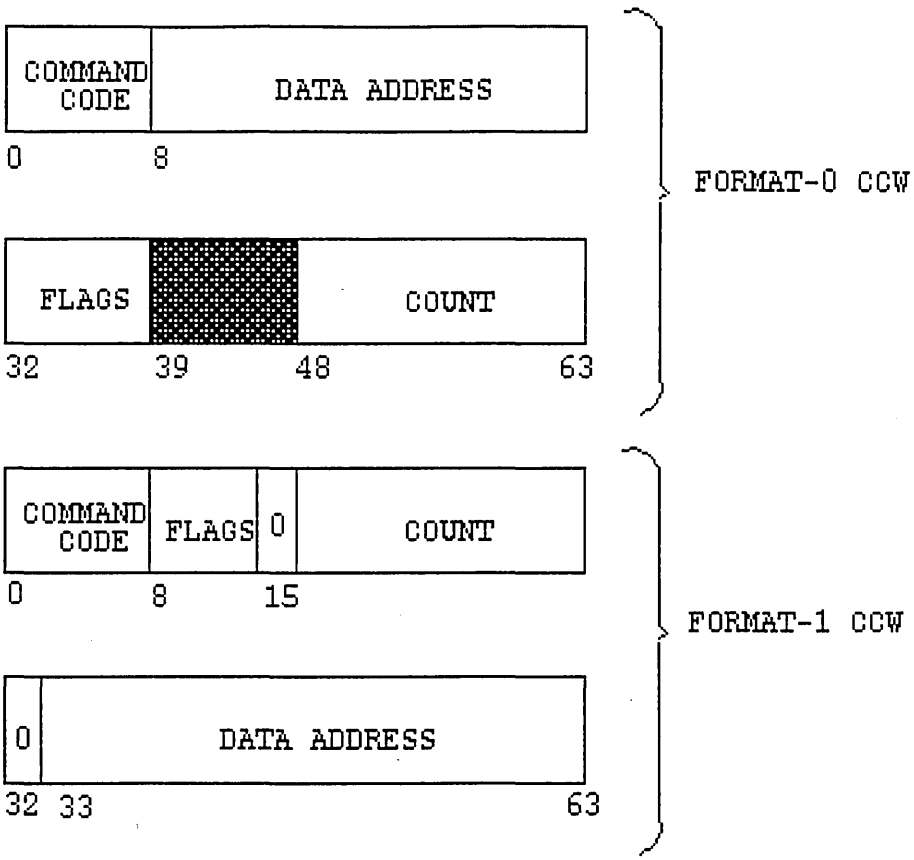
The monitoring of the channel subsystem provides I/O resource usage data in the form of measured information maintained in central storage. This information is available to the Resource Management Facility (RMF), which assists in managing the performance of the system. The RMF also performed this function in System/370; however, most of the information was obtained through sampling techniques that examined the delays or busy conditions encountered by the control program while attempting to initiate I/O operations. A monitoring facility in 370-XA was added because of the changes to the internal interfaces of the control program and because the I/O queue management and busy-handling functions were moved into the channel subsystem. The monitoring facility provides measured elapsed time parameters in main storage that described the extent of I/O resource usage, delay time, and I/O contentions encountered during execution of I/O operations. The data is accumulated on a subchannel basis and made available as each operation completes.

The instruction SET CHANNEL MONITOR places the channel subsystem in the monitoring mode and identifies the starting location, in main storage, where the measured data is accumulated. Control bits provided by the control program and placed in the subchannel during execution of the MODIFY SUBCHANNEL command selectively enable or disable a subchannel for monitoring.

1.3.5 *24 & 31 Bit Channel Commands*

A 31-bit data address in a new CCW format allows for the direct use of 31-bit addresses in channel programs. In System/370, 31-bit addressing of I/O data can only be accomplished by use of the Indirect Data Address Word (IDAW), and all CCWs and IDAWs must reside in the first 16M bytes of storage. In 370-XA, two modes of operation are provided: a compatible 24-bit addressing mode for executing old CCWs (Format-0 CCW) and a 31-bit address mode for executing the new format CCWs (Format-1 CCW). When Format-1 CCWs are specified, the CCWs and IDAWs may reside anywhere in storage. The mode is controlled by a bit passed to the channel subsystem during the execution of the START SUBCHANNEL instruction. In 370-XA, depending upon the setting of the control bit, direct addressing in either the 24-bit or 31-bit mode applies for the entire channel program being executed. Mixed CCW formats within a channel program are not allowed. The two CCW formats are illustrated as follows:

Figure 1-5: 370 Channel Command Word Format



The indirect-data-address word (IDAW) was extended to 31-bits when 26-bit real addressing was introduced in the system/processor architecture prior to XA. At the time this was the only way the channel could access data beyond the 16-Mbyte limit. The 31-bit format IDAW is now also used in 370-XA for application compatibility. 370-XA also introduced a new channel command word format called Format-1, with a 31-bit data address field. The address formats used in I/O operations are subject to and subordinate to the addressing mode specified in the address space PSW.

The 370-XA channel subsystem architecture executes CCWs that were defined for System/370, as well as the new Format-1 CCWs. Regardless of the format of the CCWs, the same channel and control unit operations can be executed. The portions of the architecture dealing with the initiation of I/O operations and with the I/O interruption mechanism were changed extensively from those implemented in System/370.

1.3.6 *Dynamic Reconnection*

The dynamic reconnection facility allows an I/O device to reconnect to any available path to continue execution of a chain of commands in a channel program. The instruction MODIFY SUBCHANNEL (MSCH) describes to the subchannel, by means of mask bytes, the set of available channel paths. This facility is controlled by a mode setting (multipath mode) in each subchannel. The MSCH instruction is also used to enable and disable the dynamic reconnection ability. The control program IOS reports the set of available channel paths for which reconnection is permitted to the I/O device. This capability, together with the channel

path management capability, allows the channel subsystem and I/O devices to chose the first available path to initiate or resume execution of a chain of I/O operations.

1.3.7 System 370/XA - Developments in Channel Path Selection

In System/370, I/O operations are initiated by the START I/O instruction, which identifies the channel and the device address on the specified channel. START I/O causes the channel to fetch the channel-address-word (CAW) from a fixed location in real storage. The CAW contains the subchannel key and designates the location in storage from which the channel subsequently fetches the first CCW. If the specified channel is busy at the time START I/O is executed, the operation is not initiated and the program is notified. If the specified channel is available, the CPU is delayed while the channel attempts to initiate the operation at the device. The length of time required is determined by the I/O device and, in some cases, may be more than 100 microseconds. While attempting to initiate the operation at the I/O device, the channel may receive a control-unit-busy indication, in which case the operation is not initiated and the program is notified. If other available channels in the configuration are connected to the device, the program can repeat the procedure, specifying a different channel in the instruction. In this instance, multiple control-unit-busy indications are possible, with a resulting CPU delay.

The START I/O FAST RELEASE function was introduced with System/370 to reduce CPU delay in initiating the operation at the device. This function allows the CPU to execute the next instruction as soon as the channel is available. Thus, processing continues while the channel, in parallel with the CPU, attempts to initiate the operation at the device. On encountering a control-unit-busy condition, the channel notifies the program with an I/O interruption so that operation initiation can be attempted on an alternate path. In some configurations, the additional processing to handle the interruptions reporting control-unit-busy more than offset the gain from the START I/O FAST RELEASE function. As a result, the function was modified to cause the CPU to wait until the control unit was engaged before the CPU could execute the next instruction. Since the time required to determine a control-unit-busy condition is less than the time to initiate an I/O operation, the START I/O FAST RELEASE function remains faster than the original START I/O function.

START I/O FAST queueing was introduced for System/370 as a result of the control-unit-busy problem. With this function, the channel would return (send back) an I/O request to the program only if the desired subchannel was busy executing an operation. If any other busy condition was encountered, the channel waited for the busy condition to clear and then initiated the operation. This approach offered performance improvement in some cases. However, an I/O request could be queued in one channel because of a busy condition while other channels with paths to the desired device are idle.

In 370-XA, operations are initiated by the START SUBCHANNEL instruction which, unlike START I/O or START I/O FAST RELEASE, does not specify the channel path. Since there is only one subchannel for each device in the system regardless of the number of paths that exists, the program specifies the subchannel corresponding to the desired I/O device. The program loads a register with the subchannel number. START SUBCHANNEL also specifies the address of the Operation Request Block (ORB) which contains the address of the first CCW to be executed. Except for a busy subchannel, the I/O request is accepted for execution. Unlike START I/O FAST queueing, in System/370, the I/O operation is not queued for a specific channel path. Rather the channel subsystem selects an available path, from those assigned to the subchannel, and attempts to initiate the operation. If busy conditions are encountered

on all paths, the I/O request remains queued in the subchannel until the operation is initiated on one of the channel paths.

1.3.8 *Application Affects*

In 370-XA, the functions performed by the channel subsystem while it is executing a channel program (addressing storage, counting data bytes, command and data chaining, etc.) are compatible with those performed in System/370. However, additional functions can be invoked by the program or device to modify certain aspects of channel program execution.

As mentioned earlier, CCWs may be either Format-0 (24 bit addressing) or Format-1 (31-bit addressing). Even if the new Format-1 CCWs are specified the same chains of commands used under the System/370 architecture are possible. This allows the 370-XA addressing scheme to be used even though the attached storage devices do not support the entire suite of 370-XA Channel Commands.

Address limit checking, if used by the program, can also affect the execution of channel programs. Address limit checking is a storage protection mechanism for I/O that separates program controlled real storage into two segments. A boundary address is passed to the subchannel in order to control where retrieved data can be placed. It is available in both System/370 and 370-XA with different possible results. When used under 370-XA, the channel subsystem compares the data address with the boundary address. If an address limit violation occurs, a program check is indicated even though the CCWs may appear valid according to System/370 rules. For System/370 applications this posed a potential complication. This required that error analysis be interrogated and if necessary made aware of this new checking feature *(POp3)*.

The dynamic reconnection facility of the 370-XA channel subsystem architecture also affects program execution. Under System/370 when a channel program is initiated with a device, the path used to initiate the operation must be used to execute the entire channel program. This caused users to attach few devices to a channel and thereby ensure the channel path for a time-dependent reconnection. Rotational devices experience a long delay time (one revolution) if a channel path is not available when needed. The dynamic reconnection facility allows a device that disconnects from a channel path during an operation to use any path to the system in reconnecting. In multiple path to control unit configurations a higher activity on each interface was made possible and an interface to the system had a higher probability of being available.

I/O interruptions allow the channel to inform the program of the status of I/O operations, as well as external events at devices. An I/O device transfers a channel end to the subchannel when an operation with the channel is ended and a device end when the operation is completed at the device. 370-XA systems reduce I/O interruptions. In System/370, the first status indication is accepted by the subchannel and then presented to the program by means of an I/O interruption. In this case, the I/O interruption is completed in only a few microseconds since it is not necessary to contact the device. However, the second status indication, if it is presented as a separate sequence, is held pending at the device, and the channel must select the device to retrieve the status as part of the I/O interruption procedure. In some cases these I/O interruptions may take an excess of 100 microseconds because of device delay. In 370-XA, both statuses are accepted by the subchannel, thus reducing the time required for I/O interruption.

1.3.9 *More On Interruption Processing*

The I/O interruption is initiated in 370-XA when an interruption is pending in a subchannel and the interruption subclass is allowed in any processor in the configuration. The channel subsystem interrupts an enabled processor and stores the interruption code, leaving the subchannel in the status pending state. The interruption code provides the subchannel number to the program receiving the interruption and allows that program to gain control of the appropriate control blocks prior to clearing the status with the TEST SUBCHANNEL instruction. In a multi-processor configuration, this process prevents one processor from initiating an operation with a device when a second processor may be handling status from the device.

Another new aspect of interruption processing in 370-XA is the TEST PENDING INTERRUPTION instruction. In System/370, after one I/O interruption is handled, it is customary to enable the CPU for I/O interruptions again to see if any other interruptions are pending. If so, the interruption procedure is repeated, with all the programming overhead associated with the required state switching. In 370-XA, the program can determine if another interruption is pending by means of the TEST PENDING INTERRUPTION instruction. If there is another interrupt pending, the program can clear the interruption request and determine which subchannel caused it. It is then possible to clear the status information from the subchannel by issuing the TEST SUBCHANNEL instruction, using the subchannel number provided by TEST PENDING INTERRUPTION. This procedure can be repeated until all pending interruptions are cleared, without the intervening saving of machine state descriptions. Since I/O interruption conditions are available to all CPUs in a multiprocessing system, the program has the option of allowing all CPUs to handle I/O interruptions or specifying that a single CPU process all interruptions.

1.4 ESA/370 - Architecture for Modern Times

When IBM started delivering computer systems to customers an I/O subsystem capable of delivering 1.2 MBytes/Sec was considered more than adequate. This limitation soon became a bottleneck for customers whose growing appetite for data rapidly outpaced the ability of the manufacturer to deliver it. Change to meet that demand was mandatory and came in the form of elimination of artificial constraint and improvements in the efficiency of the architecture. Subsequent improvements in devices enhanced the I/O design's ability to deliver data. ESA/370 is this type of evolution. Specifically it is not a correction of the previous generation; it is an addition designed to promote faster delivery of data. This section discusses how this is done and what the features of the design are.

The central issue here is an old one. The time it takes to obtain access to a block of data plus the time required to transfer that data to main memory is what determines the performance of the data retrieval system. In other words, the rate (access per second) at which a data retrieval system can deliver data is a key determinate in how well the overall system performs. It affects the system by causing more than necessary visits to the data, causing the processor to wait for the required data and incurring the associative overhead to the response time of a transaction.

Traditional database applications use multiple files or databases and depending on the complexity of the transaction several to many visits will be made to those files. Relational database products have exacerbated this I/O profile by simplifying unplanned inquiries and introducing data browsing. Add to that the increasing use of on-line reporting, image processing and the rest of the boutique means for storing information. So as transaction complexity grows so does the amount of data accessed by those transactions (Amdahls Law) This presents a

significant problem to installations committed to delivering an agreed to terminal response time.

Certainly the storage technology industry can't help here, at least not in the near term. Expanded storage, therefore, gets incorporated into the architecture. Not the I/O specific architecture but the architecture of the system. On a 3090 processor the scale of time is measured in billionths. The amount of time to retrieve information from disk vs Expanded Storage is hundreds of millions or hundreds of thousands times longer, respectively. Expanded storage is used not like an electronic disk but like an extension of central memory. There are no I/O's (initiation, locate, transfer or interrupt) to Expanded Storage. What Expanded Storage will do is assume a more critical role so that it functions both as a fast paging device and a data storage device.

1.5 Disk Storage

The following table of storage devices includes those which are perceived as the most significant in current IBM systems:

Device	Capacity(MBytes)	Cylinder(KBytes)	RPM	Avg Seek
9332-200	200.3	149.504	3119	19.5
9332-400	400.6	147.456	3119	19.5
9335-B01	521.5	265.68	3623	18
3380-K	1890	712.1	3600	16
3390-1	946	849.9	4210	9.5
3990-2	1890	849.9	4210	12.5

CHAPTER 2

DIGITAL I/O ARCHITECTURE

2.1 Introduction

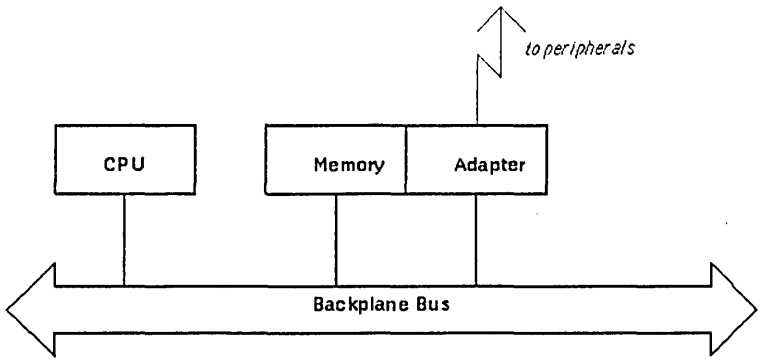
This chapter discusses the current Digital I/O architecture. Just as the IBM/370 is the foundation for IBM's image, the VAX is the foundation for Digital's image. Unlike classic mainframes, the VAX has no I/O processor in the architecture. There is no concept of channel programming. Evolving from the minicomputer, VAX systems support I/O using classic communications layering.

2.2 VAX Computer Systems

VAXes have been built with numerous backplane buses—SBI, CMI, BI, NMI, and XMI. Originally the devices available to connect to these systems were built to connect to the PDP11 UNIBUS. Adapters were built to convert between the backplane signal and UNIBUS signals. MASSBUS was developed as an improved interface to mass storage but was still supported via adapters. With the advent of VAXclusters, global (remote) storage required decoupling mass storage from the local system. This led to the definition of the Digital Storage Architecture (DSA) as a distributed I/O architecture. The typical implementation of this architecture is the HSC server.

The VAX architecture contains no special instructions for I/O initiation. Memory-mapped command and status registers are used that can be manipulated with regular memory reference instructions. These registers communicate with a device adapter and are protected by privileged memory management. Conceptually, the adapter registers are an extension of primary memory in the physical address space. Refer to Figure 2-1. The design of the adapters may use the page tables set up by system software.

Figure 2-1: VAX Overview



VAX overview

The register transfer level of the VAX is a Complex Instruction Set Computer (CISC) with a fairly orthogonal instruction set. There are 16 General Purpose Registers (GPR); however, the last four are used for stack control and program counter. There is also a Processor Long Word (PSL) that holds the current state of the processor.

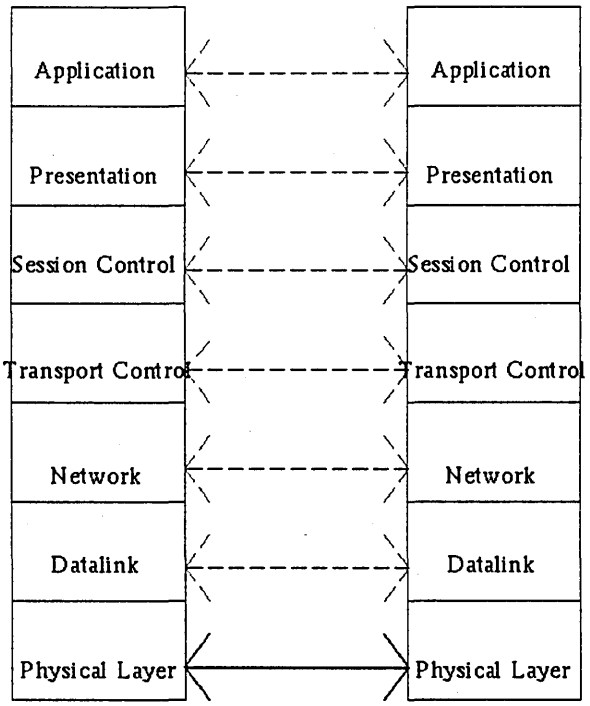
All processors that accommodate multiple interrupts must have a technique to resolve the order of processing in the event that more than one interrupt (type) occurs at a given point in time. The prioritization in the VAX architecture is done via Interrupt Priority Levels (IPL), where a higher level can preempt a lower level. Processing at a given level also blocks all lower-level processing. IPLs are used for both unsolicited and solicited (exceptions) interrupts.

A peculiarity of the VAX is the interaction of the ASTLVL register and the REI (return from interrupt) instructions. After the PSL is popped into a temporary register, the ASTLVL register is checked for a potential AST (asynchronous service trap) delivery. If there is a pending AST that can be delivered, an IPL 2 interrupt is triggered, pushing the PSL back on the stack and transferring control to the AST delivery mechanism. Even though an REI instruction cannot raise the priority level, a high priority level (such as used by an ISR) can trigger execution at a level that is between its IPL and the IPL at the time of the interrupt. Thus, hardware efficiently supports I/O completion (as well as other asynchronous events) at a higher processor priority than "normal" processing. Such a system is classically referred to as being "interrupt driven".

2.3 DSA concepts

The underlying principle of DSA is that I/O is a special case of communication. To explain the concept, consider the ISO Open System Interconnect (OSI) architecture. Figure 2-2 depicts the model. Interfaces vertically connect the individual layers on each side. The interfaces are essentially subroutine calls where the upper layer calls the next lower layer and lower levels may raise attention to work for upper levels. Protocols are the horizontal flow between each side's mirrored layer. Typically only the lowest level protocol actually involves hardware interfacing. Protocols define data packet format, the synthesis/decomposition, and subsequent processing. To simplify the distinction between interface and protocol, protocols define the communications between peer layers while interfaces are between adjoining layers.

Figure 2-2: ISO model



In classic communications architectures there are two types of transports—those built upon datagram services and those built upon virtual circuits. Datagram services force the higher-level protocols to be resilient in ensuring message receipt from end-to-end. Virtual circuit transports exhibit flow control and guarantee delivery of messages (in order) to the higher layers.

Flow control is a generic term for ensuring that neither side of a communications link exhausts the resources (usually memory) at the other end. There are numerous flow control techniques, but the most common is referred to as "windowing" (see [NETW76]). With windowing each node keeps track of the "credits" it has issued and the credits it has received for a given link (connection). As long as the node at the other end grants at least one credit, that node may send flow-controlled traffic. It also has the responsibility of extending credits to the other end when applicable.

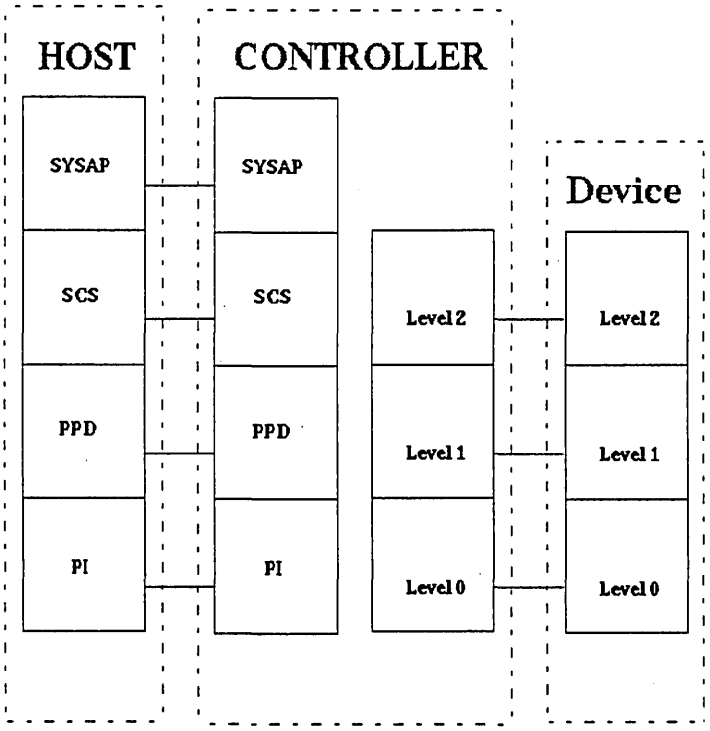
The general form of DSA actually incorporates two communications architectures as shown in Figure 2-3. Three node types are described by this conceptualization—hosts, controllers, and devices. Controllers appear abstractly as translators between SCA and (in this case) SDI/STI (also referred to as SxI).

Communications datalinks normally fall into either the store-and-forward (point-to-point) or broadcast. Store-and-forward networks "route" messages from one node to another. Each intervening node receiving the message determines which node to forward the message to next to achieve a path to the final destination. In a broadcast topology a message is transmitted on a common wire to every node. Each node then filters messages so that messages for other nodes are ignored. Referring to Figure 2-2, the network layer is necessary for store-and-forward datalinks, but not for broadcast topologies.

2.3.1 *System Communications Architecture*

The basic System Communications Architecture (SCA) is four-level architecture for the CI. When the Ethernet is used, this architecture is expanded to seven layers. Both form broadcast topologies.

Figure 2-3: DSA Overview



2.3.1.1 *Computer Interconnect*

The top level of the architecture is the System Application (SYSAP) layer. The protocol is the Mass Storage Communications Protocol (MSCP). It is a master/slave protocol where the host sends command packets (like read and write) and the controller sends response packets

(with status). The software component in the host is the class driver. The controller software is an MSCP server.

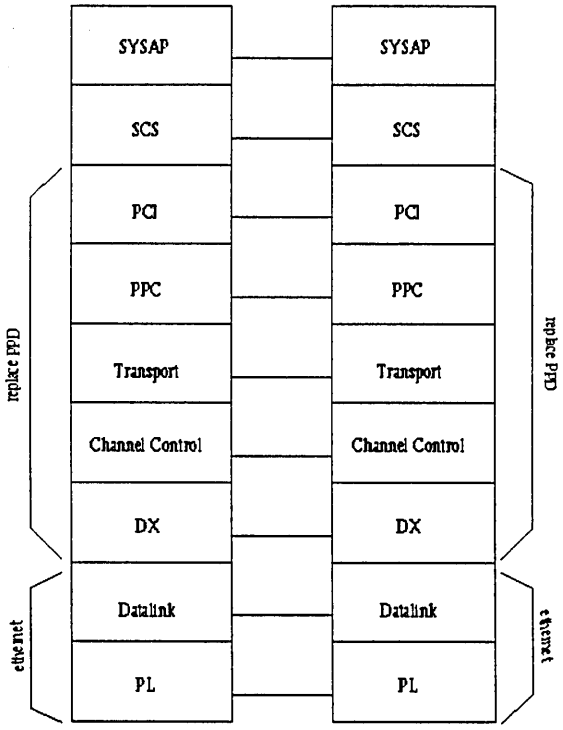
The second layer is SCS. It basically provides the connection service for the SYSAP layer. Connections exist between class drivers and MSCP servers and provide flow control via credit management for sequenced messages.

The next layer, PPD, is implemented by port drivers in the host. The controller software is implementation dependent. This layer provides "virtual circuits" (VC) to the SCS layer. These virtual circuits are similar to classic virtual circuits. A VC is required for some kinds of datagrams, as well as all sequenced messages. Sequenced messages are currently flow controlled by a single sequence bit for both send and receive.

The physical layer provides the electronics and data encoding for the upper layers. The CI has two paths, each with a send and a receive cable. The encoding is synchronous Manchester encoding with data and clock on the same cable.

2.3.1.2 *Ethernet*

Figure 2-4: SCA with Ethernet



SCA/NI is a nine-layer architectural extension of SCA that allows the Ethernet to be used as a datalink for Local Area VAXclusters (LAVCs). Basically, the PPD layer is replaced by five layers to support the additional abstractions of rails and channels for LAVCs. The PI layer is replaced by the standard Ethernet datalink and physical layers. In this architecture, the distinction among host, controller, and device is absent. In the LAVC environment there are special nodes such as boot nodes, but the notion of controller does not exist.

The following layers are substituted for the PPD layer in SCA/CI:

1. PCI
2. PPC
3. Transport
4. Channel control
5. Datagram Exchange

The top layer replacing PPD, the PCI layer, defines the interface between the port (device) and the port driver. The next layer (PPC) provides the virtual circuit services to PCI. The transport layer is next and actually provides the guarantee delivery service for the virtual circuits of the PPC layer. Channel control is responsible for the network topology, providing the channel abstraction. The final PPD layer replacement is the Datagram Exchange (DX) layer. It provides the datagram services upon which rails, channels, and virtual circuits are built.

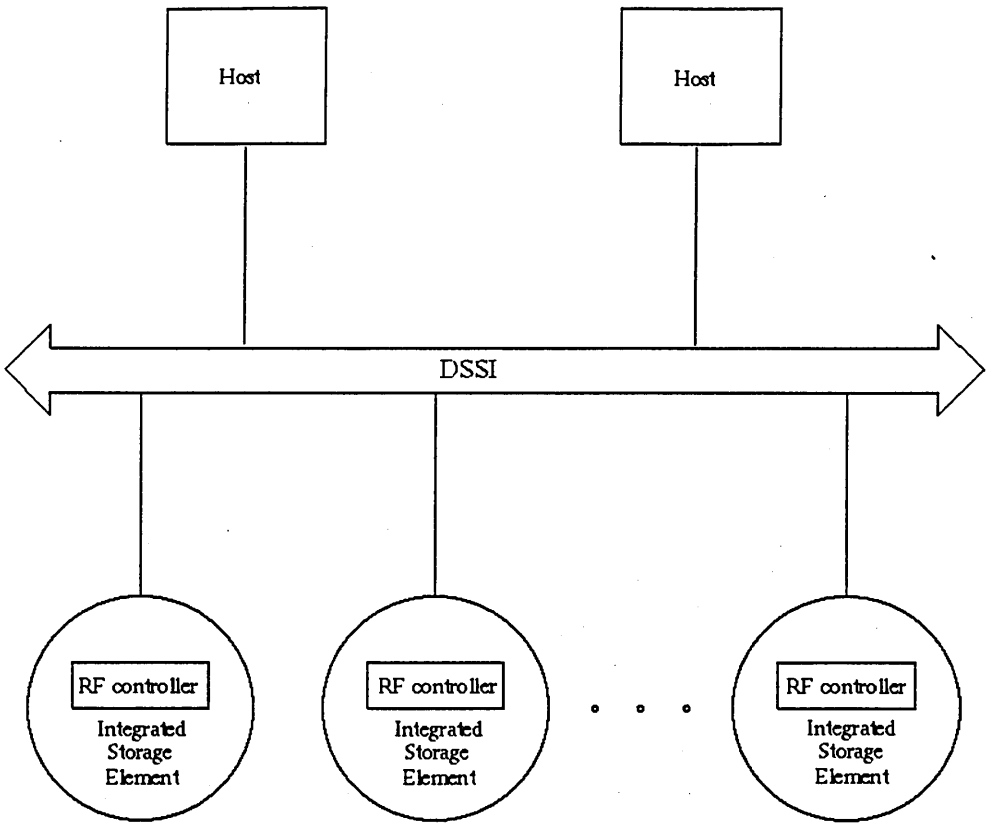
The standard Ethernet datalink is responsible for basic arbitration (in the absence of collisions), nodes address filtering, and collision recovery. The physical layer defines the Manchester encoding and collision detection mechanisms, as well as the tranceiver and cable characteristics.

2.3.1.3 *Digital Storage Systems Interconnect*

DSSI is a broadcast datalink that supports a maximum of eight nodes. The nodes may be (theoretically) any mix of hosts and storage nodes. The bus is an 8 bit parallel path capable of approximately half of the bandwidth of a single CI path.

In implementation, the storage nodes have the controller embedded in the device to form an Integrated Storage Element (ISE). This ensures linear performance increases up to the capacity of the DSSI itself.

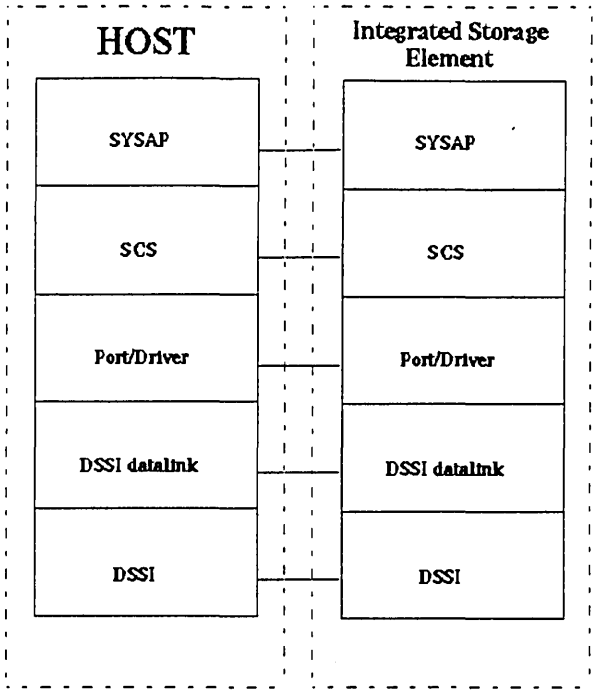
Figure 2-5: DSSI Topology



Eight nodes max (including hosts)

The protocols for SCA/DSSI show that the top three-layers are virtually identical to SCA/CI. The addition of a DSSI datalink appears to be more of refinement over SCA/CI in that the CI version lumps the CI datalink in with PPD. The DSSI architects have chosen to make the distinction with a separate layer.

Figure 2-6: SCA with DSSI



2.3.2 *Standard Device Interfaces*

In keeping with the communications model, standard device interfaces have been defined. These interfaces are "realtime" in that the flow of data is synchronized with the actual physical transfer to the moving media (disk or tape). The Standard Disk Interface (SDI) and Standard Tape Interface (STI) are both three-layer models. Refer to Figure 2-3.

The top level (level 2) commands provide a master/slave protocol that allows a controller to handle drive diagnosis, status, topology, and initiate seeks.

Level 1 of SxI is a symmetric protocol that details the meanings of the control frames of level 0. The protocol is asymmetric for other than framing purposes—the controller initiates read/write activity, and the device responds. Level 1 commands provide group and track selection. Formatting, reads, and writes are also initiated via these commands.

The bottom level is the electrical signal level. The raw clock rate ranges between 5MHz and 22.5MHz, depending upon the device. The interface is symmetric. Four physical wires are present with two for controller and device states. The other two wires are shared for data movement and level 2 traffic. Exchanges across the data movement lines are 32 bits comprised of a 16-bit SYNC character and a 16-bit control frame.

2.3.3 *Device Data Formats*

DSA tape devices adhere to the ANSI standards for tape devices.

Digital Standard Disk Format (DSDF) is a Fixed Block Architecture (FBA). The physical disk sectors are divided into two ''spaces'': LBN and RBN. The SDI version of DSDF adds XBN space to define the entire disk and reserves DBN space for diagnostics. The controller provides the mapping services for each of these spaces. The MSCP protocol predominantly allows for addressing LBN space. RBN space can be addressed to accommodate host-based bad block replacement. XBN space basically defines a mapping for the entire formatted disk.

LBNs are the disk sectors that are reserved for system and user data areas. LBN space maps the major portion of the disk.

RBNs are replacement blocks. With the SDI implementation, RBNs map to two different regions:

1. Primary revectors ''on track''.
2. Secondary revectors to dedicated cylinders.

The first region is a collection of sectors on-track with the LBNs. They are reserved to avoid seeks for the revectoring operation. Accessing a revectored block on-track involves reading the sector header. The second region is a separate band of cylinders reserved for revectors that occur after the on-track RBNs have been filled. A control table (RCT) is duplicated according to drive geometry to control allocation of the second region. Allocation of a secondary revector involves updating every copy of the RCT, as well as indicating the revector in the original LBN sector header. Accessing a secondary revectored block involves reading the RCT to determine the RBN location.

The last sector mapping space is for diagnostic use. DBN space is reserved to allow an area of the disk where diagnostics can freely write as well as read.

2.4 DSA Implementations

The large system implementation of SCA is built upon the CI datalink. The current host adapters for this datalink are derived from the CI780 (or CI750), depending on the host backplane bus. These adapters are only capable of single active path operation. The XCD (XMI to CI) adapter is a new design to support simultaneous dual path operations.

DSSI provides the datalink for medium and low-end systems. DSSI is a proprietary bus similar to the SCSI interconnect. The PPD and PI layers are different from those of SCA/CI, but the SYSAP (MSCP) and SCS layers are the same. Topologically, DSSI is very much like the CI. One major distinction is that the controller for DSSI devices is integrated with the HDA.

2.4.1 *DSA controllers*

DSA controllers can be connected either locally to the backplane bus or globally via a datalink such as DSSI or CI. The following table outlines the current DSA controllers.

Table 2-1: DSA controllers and statistics

Controller	Host Connect	Device Connect	Max Devices	Req/sec	Data Rate(MBytes/sec)
UDA	Unibus	SDI	200	200	.75
KDA50	Q-Bus	SDI	4	200	1.2
KDB50	BI-Bus	SDI	4	200	1.7
KDM70	XMI-Bus	SxI†	8	700	5.4
HSC70	CI	SxI†	32	1150	4.2
KFQSA‡	Q-Bus	DSSI	7	190	1.5
EDA640‡	CPU card	DSSI	7	360	1.5

†Either SDI for disk or STI for tape

‡These are adapters rather than controllers

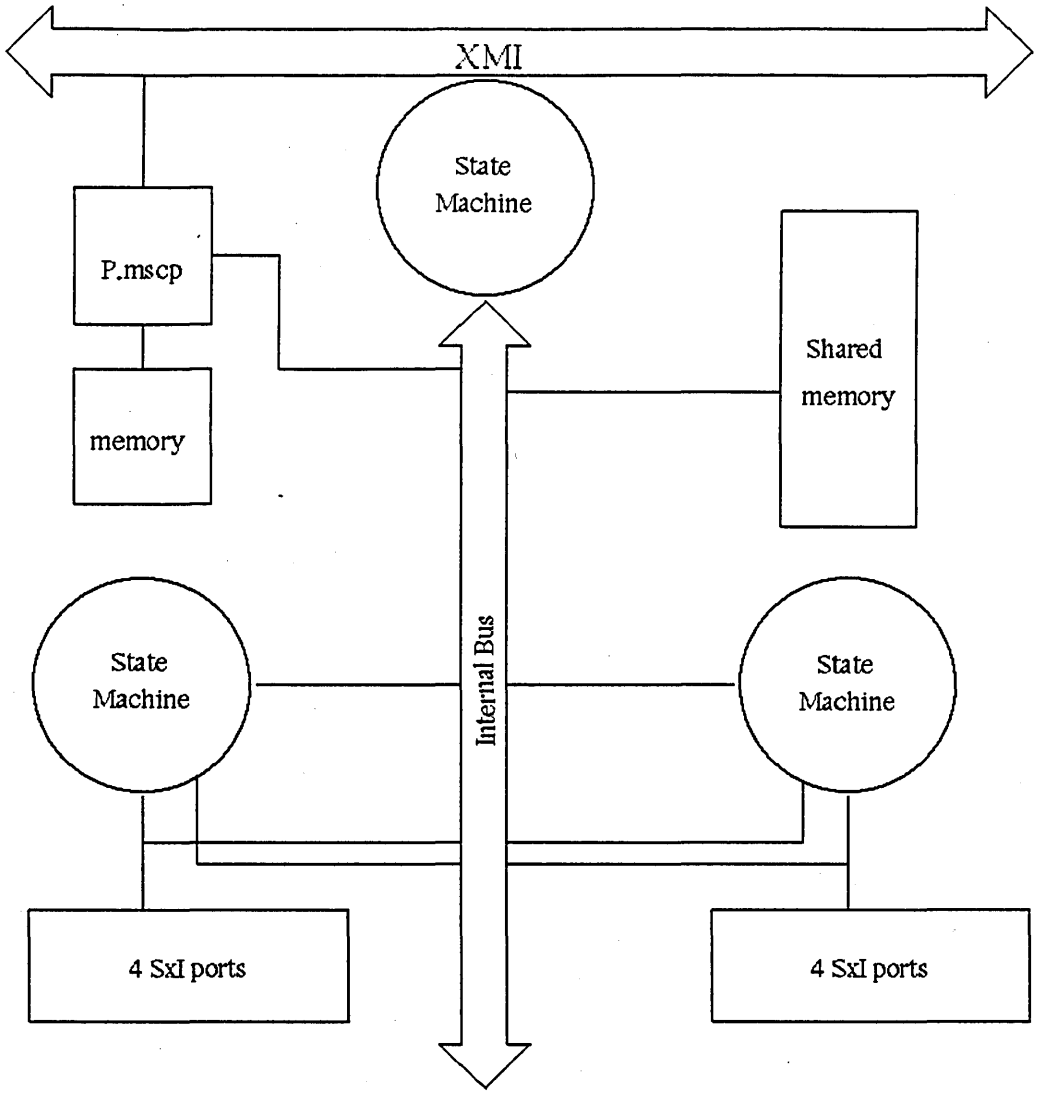
2.4.1.1 *Board Level Controllers*

All of the SxI controllers listed in the previous table are board-level controllers except the HSC70 server. Most of these controllers have a similar architecture that incorporates a 2901 bit slice processor as its basis. The port driver for these adapters employ a "ring buffer" interface with the controller microcode.

The KDM70 is radically different from the other board-level controllers in that the center of processing is a cVAX (labeled P.mscp in Figure 2-7). It decomposes incoming MSCP requests into "work blocks" for the SxI state machines and sends the end packets back upon completion.

The eight backend ports are divided into two four-port blocks. The two SxI state machines "float" between the ports. The KDM70 is connected directly to the XMI backplane. The XMI is capable of a peak 100 megabyte/second burst but is limited to 66MBytes/sec in this application. The internal KDM70 bus is 32 bits wide with a 200-nanosecond cycle time for a 16MBytes/sec bandwidth. The throughput is further constrained by a total of two state machines for SxI transfers. Each SxI path can produce a maximum of 2.7MBytes/sec with an ESE with enhanced protocol.

Figure 2-7: KDM70 Overview



2.4.1.2 *HSC*

The HSC family of controllers is the one example of a global storage controller that is shared among VAXcluster nodes. It implements the SCA/CI architecture completely.

Before launching into a description of the HSC server internals a discussion of the naming convention is in order. The following table describe the HSC components as named based upon PMS notation derived by Bell and Newell in 1971 (see [ARCH71]):

Table 2-2: HSC Naming Conventions

Letter	Meaning	Function
P	Processor	To execute programs from RAM
K	Controller	To execute programs from PROM
M	Memory	To store instructions and data

In the HSC server, the difference between P's and K's is that P's execute programs from RAM while K's execute microcode from PROM. The convention uses a dot to delimit between the major component and the specific type of that component. For example, K.ci refers to the CI controller.

There are three types of K's:

1. K.ci for connecting to the CI.
2. K.sdi for interfacing with disks.
3. K.sti for interfacing with tapes.

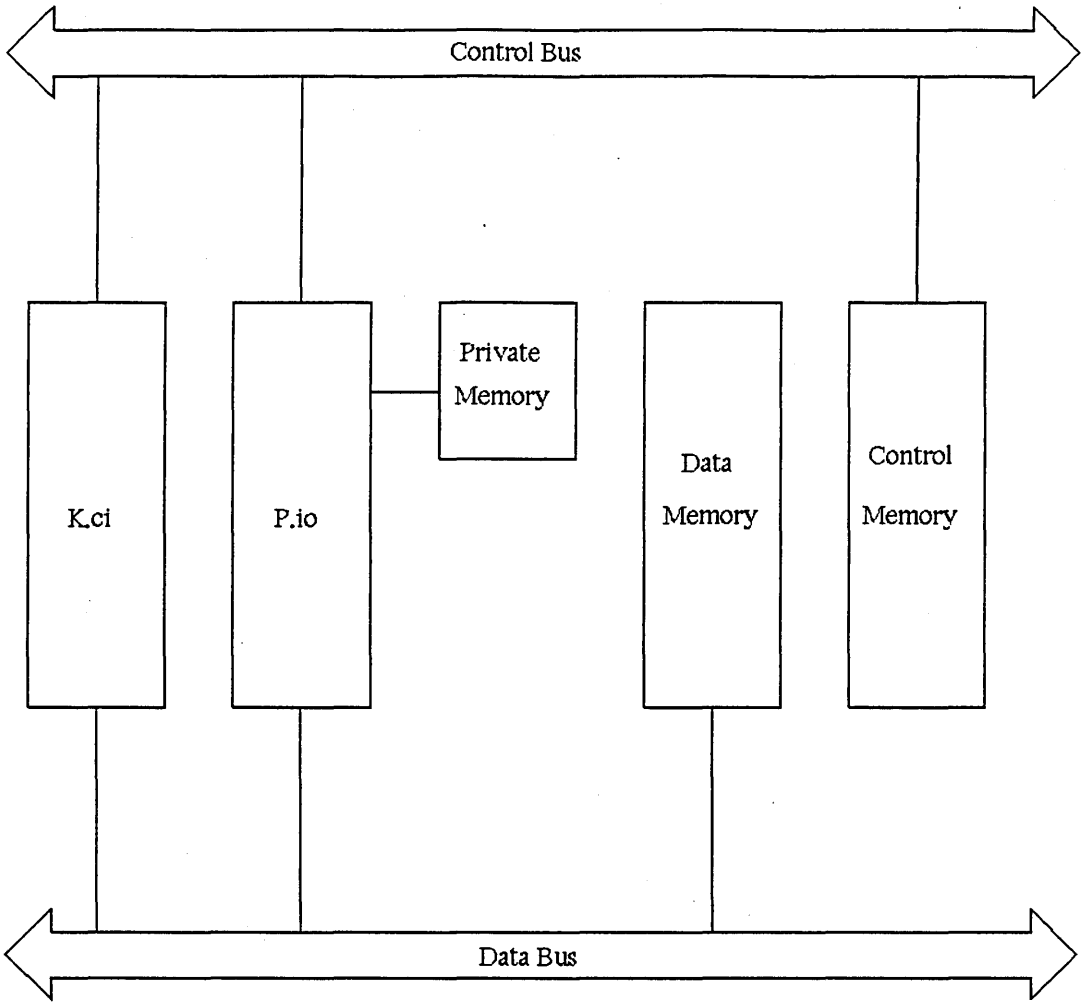
There are three major memories:

1. Control Memory
2. Data Memory
3. Program Memory

Two buses provide the building blocks for the HSC server. The control bus allows access to "control memory" which is shared among all K's and the P.io for communication and synchronization. The data bus is a 13.3MBytes/sec bus that transfers read and write data in and out of "data memory". Data memory buffers between the realtime SxI interface and transfers across the CI. There is a third private bus between the P.io and its program memory.

The center of the P.io is a PDP11 processor chip. The other active elements in the HSC server (called "requestors") are K.ci, K.sdi, and K.sti. The HSC hardware is pictured in Figure 2-8 only the K.ci controller.

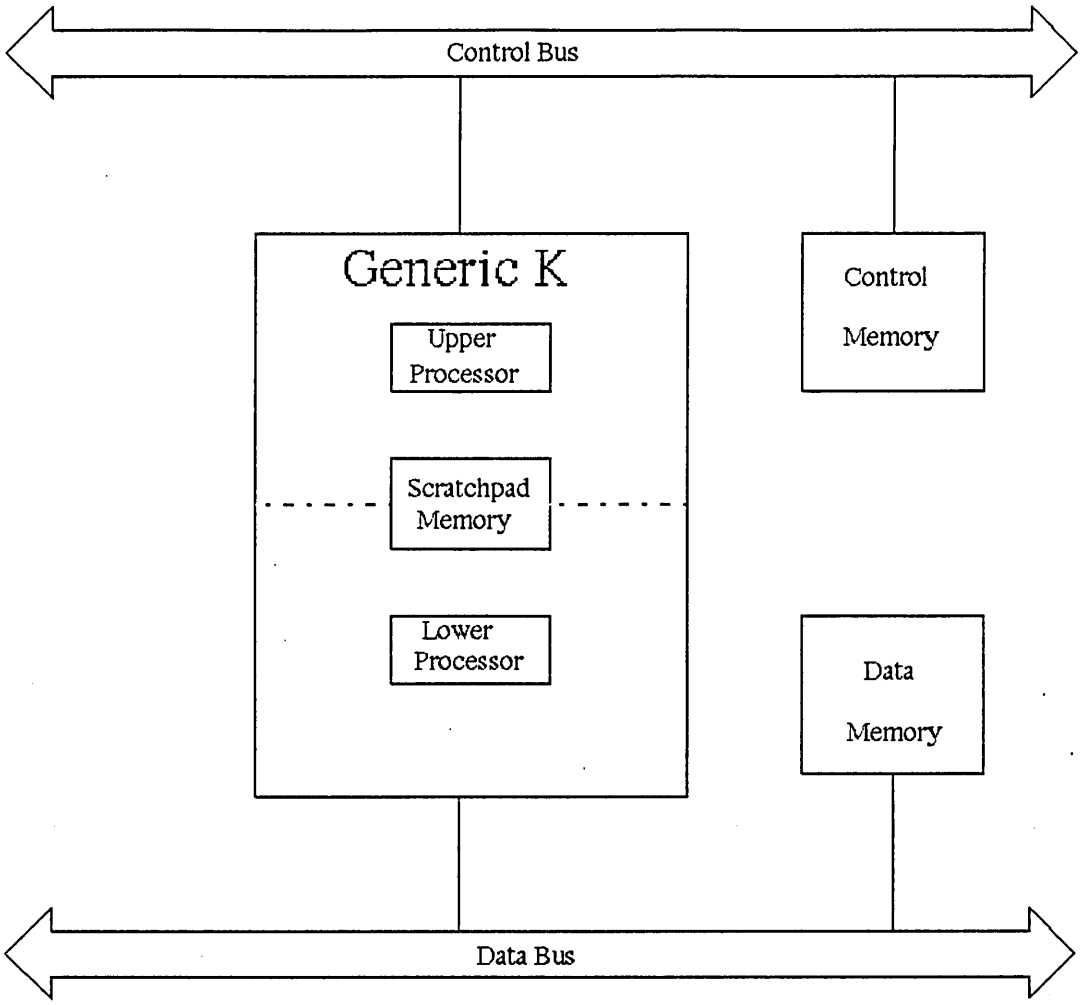
Figure 2-8: HSC Without Disk/Tape Requestors



Each of the K's use 2901 bit slices as pictured in Figure 2-9. The upper processor's primary role is accepting requests from the P.io software, while the lower processor's purpose is to move data memory across the data bus. The upper processor cannot interface with the data bus, nor can the lower processor interface with the control bus. The upper and lower processors communicate through the shared ''scratchpad'' memory that is onboard.

The K.ci is somewhat anomalous in that its primary mission is to interface with the CI. Its lower processor moves data to and from data memory, but an additional buffer (PLI/PILA) exists between the standard K design and the actual CI interface. K.ci's upper processor serves as the primary data memory allocator for the entire HSC software subsystem. It frees buffers when an MSCP end packet is sent to the host without involving the P.io software.

Figure 2-9: HSC generic Krequestor



2.4.2 *DSA Devices*

DSA devices can be divided into two categories according to the interconnect. The midrange and low end systems devices connect via DSSI while the large system connect by either SDI (for disk) or STI (for tape). DSSI devices use the ''RF'' prefix. Disk devices that connect to SDI use the ''RA'' prefix. STI tape drives are named with the ''TA'' prefix.

The following tables describe the characteristics of current DSA devices.

Table 2-3: DSA disk devices

Device	Capacity (MBytes)	Cylinder Size (KBytes)	RPM	track to track (ms)
RA60	176	76	3600	7.5
RA70	281	181	4000	5.5
RA80	116	434	3600	7.5
RA81	435	357	3600	7.5
RA82	596	427	3600	5.5
RA90	1161	449	3600	4.0
RA92	1436	475	3405	3.0
RF30	150	111	3600	4.0
RF71	400	296	3600	4.0
ESE20	120	1024	N/A	1.4

Table 2-4: DSA tape devices

Device	max ips	max bpi	transfer rate
TA78	125	6250	763KB/s
TA81	75	6250	458KB/s
TA90	78	38000	2.4MBytes/sec

2.5 VAIL Concepts

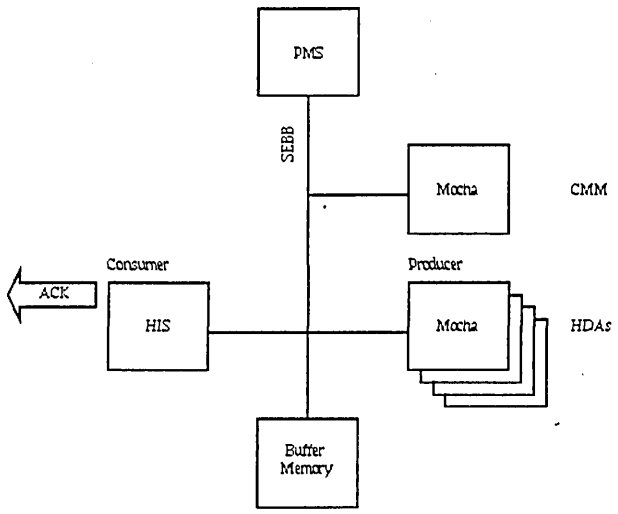
Vail does not affect the SCA (CI) architecture, but it does redefine protocols. The two major changes are elimination of SxI and extending CI to include the concept of ''base nodes'' and ''subnodes''. SxI is eliminated by introducing a ''storage element'' architecture that takes advantage of current technology.

In addition, to the storage element architecture, a cabinet architecture is defined to facilitate packaging issues. In essence one can view the cabinet architecture as defining the base node.

2.5.1 *Storage Element Architecture*

A storage element is similar internally to an HSC server in that there is a CI port, a processor to handle MSCP interpretation, ''controllers'' to actually perform the data movements, and a shared memory. The internal bus structure is entirely different from that of an HSC server, and the devices reside inside the cabinet.

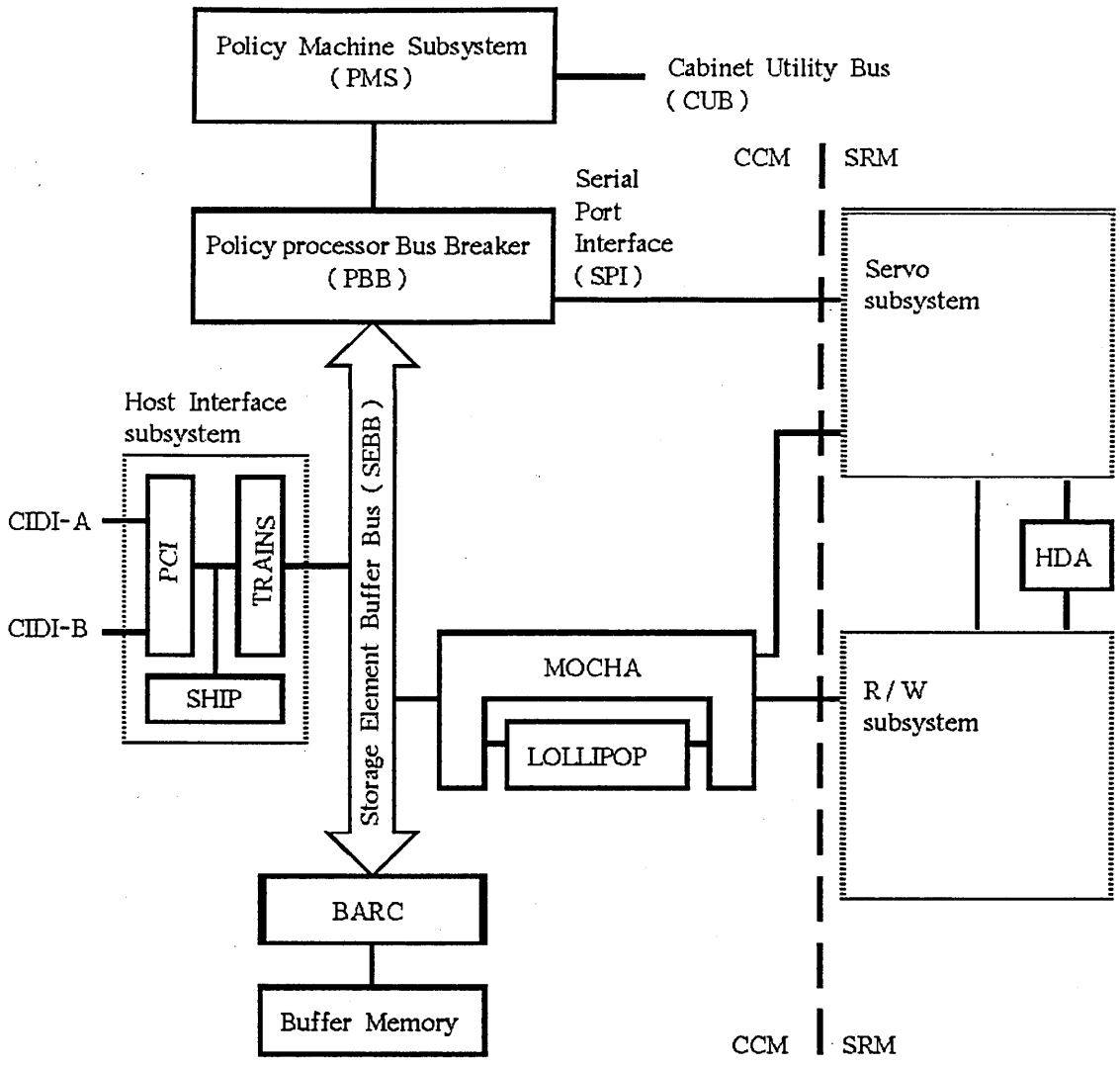
Figure 2-10: Storage Element Overview



The CI port for a Vail storage element is fed from the CID, but the port should be able to connect directly to the CI. These custom chips are being designed for the CI port - PCI, SHIP and TRAINS.

The MSCP interpreter processor complex is built upon a cVAX. The software is being evolved from the KDM70 software. Although the device services software is predominantly new, the executive is a port to the new environment with the Storage Element Buffer Bus (SEBB) and new custom chips.

Figure 2-11: Storage Element



The K.si's in the HSC server are replaced by the MOCHA/LOLLIPOP chips, which move data between buffer memory and the read/write subsystem. LOLLIPOP performs the ECC function as data passes through the MOCHA. The MOCHA may actually interface to either an HDA or a Cache Memory Module (CMM). The amount of memory on a CMM depends on the actual Vail device.

2.6 Cache Implementations

The only cache product offering today is the data cache function within ULTRIX. VMS systems today have four file system caches that expedite directory operations only. A full discussion of caching appears in the Digital/IBM comparison section of this report.

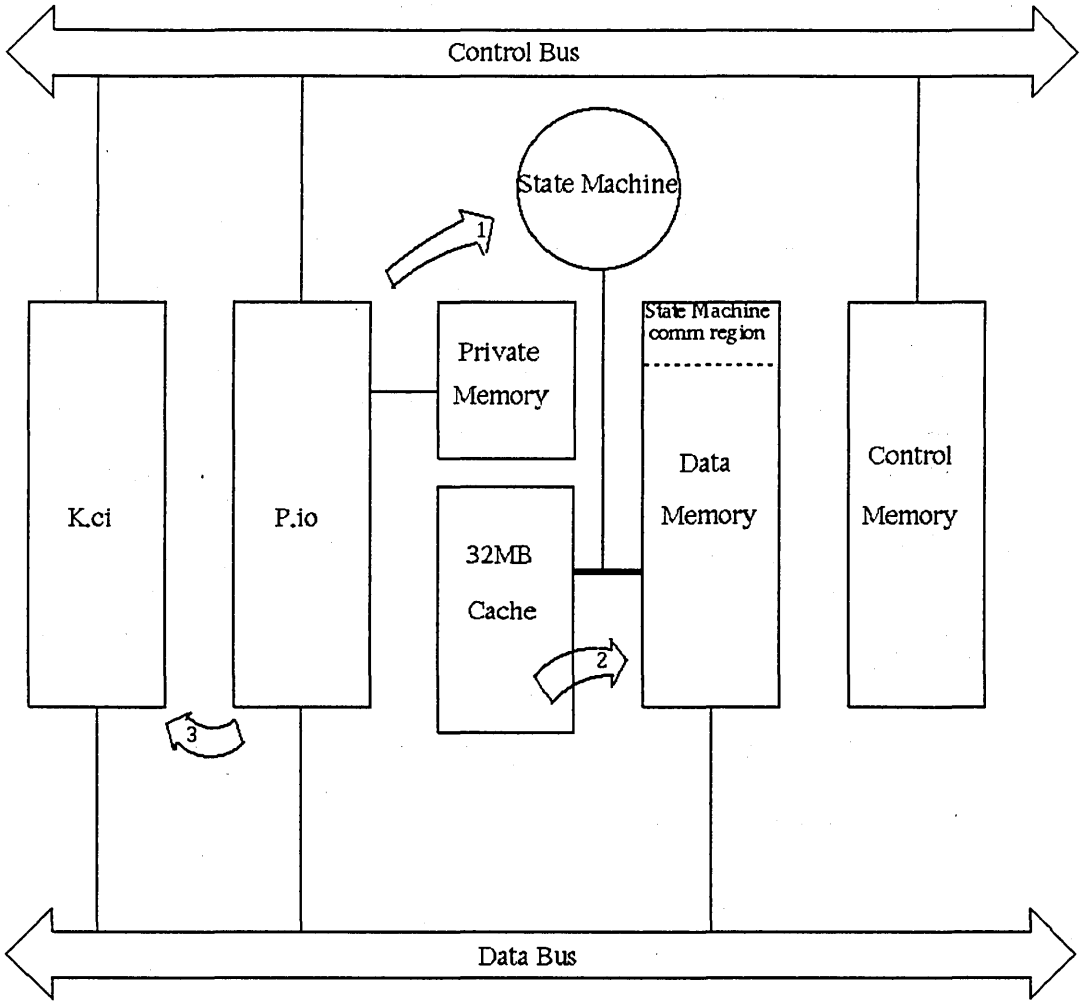
Over the last three years, a multilevel cache strategy has been evolving. Three levels can be viewed as host, controller, and device. Earlier performance studies showed that the most performance benefit seems to be from a combination host (software-based) writethrough cache and a write-behind cache as close to the device as possible. Currently SDI requires static dual-porting with failover. This requirement makes a reliable write-behind cache in a controller difficult to develop because any disconnect between the controller and drive would lead to potential lost/corrupted data. The problem is being addressed by reserving one of the surfaces on the disk for copying "dirty data" (essentially without rotational latency). The technique requires SDI to be extended so that the device logs "dirty blocks" until the target write is completed (by the controller). In this way, the dirty blocks can be copied from the reserved surface to the actual target block in the event of failover. This "fast write" (probabalistically) incurs the overhead of a head switch time over that of a nonvolatile write-behind cache.

The HSC cache will be a 32-MByte cache. It will be provided a second port to data memory rather than using data bus cycles. Private memory is used by the P.io to control the index structures for the cache. The cache will be a writethrough. "Fast write" is being implemented independently of cache; however, cache will be required for configurations where data memory is insufficient (e.g., HSCs that also support tapes).

When a cache hit is detected by the P.io software, it builds a command list for the state machine in a reserved portion of data memory (labeled state machine comm region). Refer to arrow 1 on Figure 2-12. The state machine then moves the data from the cache memory to data memory. The P.io polls for completion of the data move. It then sends the request to move the data out the CI port onto K.ci.

Cache allocation involves similar steps, where the P.io builds a command list for the state machine to move from data memory to cache memory. Other details depend upon whether the allocation is due to a read, readahead, or write.

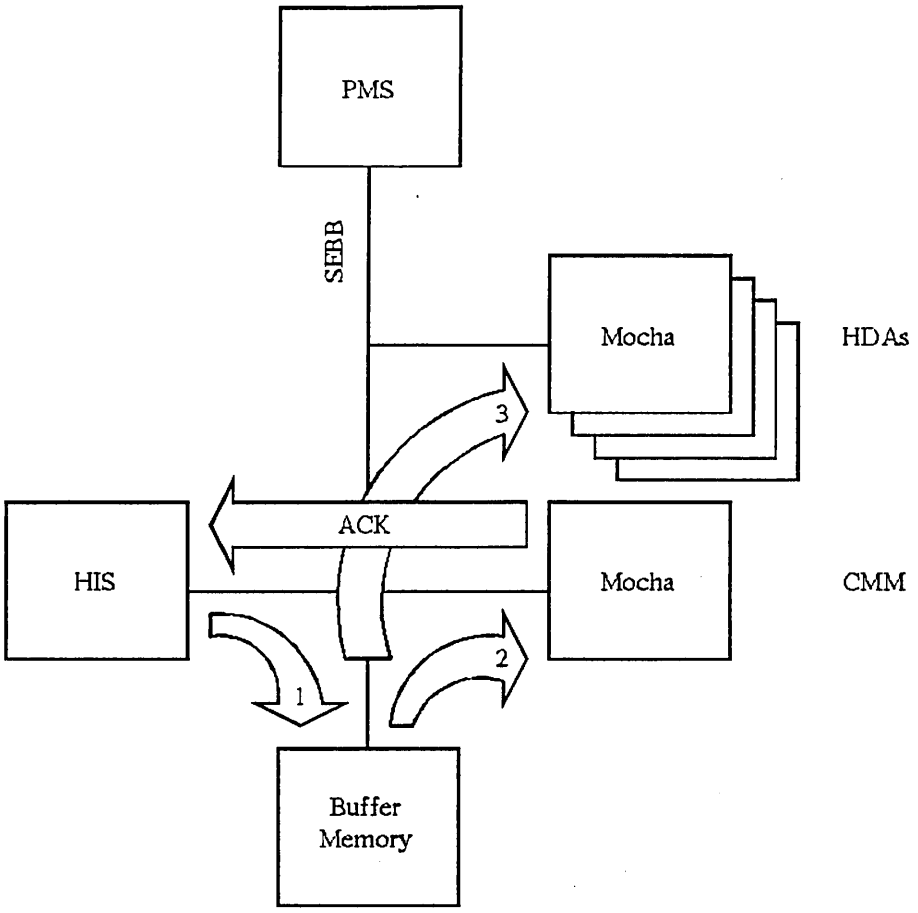
Figure 2-12: HSC cache read hit



VMS is in the process of implementing a host-based software cache that can be declared to any size. The only performance concern with host cache is the potential lock manager traffic requirements for synchronizing to prevent "stale data". To date, the stale data problem has not been quantified. There is a desire to do an "unreliable write-behind" cache within the host. Such a cache would benefit temporary work files. The actual implementation would involve an immediate completion to the application program, followed by discarding the actual completion posting.

One big advantage of the Vail storage element architecture is the ease with which a write-behind cache can be implemented. The design merely involves adding a Cache Memory Module (via MOCHA) and the appropriate Cache Manager software in the policy processor.

Figure 2-13: Vail Cache Flow



The preferred method of caching is to first look into buffer memory then to search the CMM for a cache read. Write data is first copied into buffer memory (Figure 2-13, arrow 1). For cached writes data is copied from buffer memory into CMM memory (Figure 2-13, arrow 2). CMM memory is nonvolatile and ECC protected. At that point an acknowledgment is sent back to the host, thus effecting a "write-behind" cache. The data is retained in the buffer memory until actually transferred onto media (arrow 3), at which time it becomes expendable. Figure 2-13 demonstrates a 3-step method. Just as the HSC "fast write" will require cache memory under certain configurations, it is possible for Vail storage elements to encounter workloads where retaining data in buffer memory may be infeasible. In those cicumstances a 4-step method becomes necessary. In the 4-step method the buffer memory is released as soon as the acknowledgement is sent back to the host (after the data is resident in CMM memory). Then when the target write is performed, the data is copied from CMM memory back into buffer memory (step 3) before it is finally moved onto the media (step 4).

CHAPTER 3

IBM VERSUS DIGITAL I/O ARCHITECTURES

3.1 Introduction

The current Digital and IBM I/O architectures are built upon opposite premises. Digital has built its DSA I/O architecture as a special case of communications. IBM views communications to be a special case of I/O. In practise, both have evolved to offload the host CPU of the I/O function by use of microprocessing within the I/O subsystem.

The two companies have designed their respective I/O subsystems for different environments. IBM has used an open-ended I/O architecture capable of supporting an indeterminantly powerful system. Digital has designed an I/O subsystem to satisfy requests from multiple systems of bounded I/O appetite. IBM's environment has grown out of an architectural philosophy which focuses on many servers that reduce, if not eliminate, queuing. Referring to Appendix C, IBM has evolved the I/O subsystem from the monolithic processor; Digital has built its I/O subsystem for loose coupling.

Both IBM and Digital have numerous product offerings that implement the overall I/O architecture. This section compares IBM's DLSE to Digital's SCA/CI. These two architectural implementations represent the top of the line for each. The KDM70 controller is quantified because of its use with the VAX 9000.

In the quantifications that follow, every effort has been made to isolate the comparison based upon architecture rather than details of implementation. To that end, device characteristics, such as seek, rotational latency, transfer times, and queueing times have not been quantified. However, since systems are actually implementations of an architecture, the current products of each vendor have been used as necessary to determine the delays inherent in the architectures themselves.

Performance is dictated by workload as well as by the system configuration. In point of fact architectural evolution results in modifications to enhance performance by taking advantage of certain characteristics of the workloads processed. The proliferation of caching implementations is an example. Likewise, there are certain architectural peculiarities that affect the characteristics of the workloads themselves. The differences between IBM and Digital workload characteristics are qualified - with a modest amount of quantification.

3.2 Physical I/O Functional Comparison

This section attempts to compare components of the two vendors on a functional scale.

3.2.1 *Dimensions*

IBM's DLSE subsystem is built upon the channel subsystems architecture. Each processor, within each model series, has a limited number of channels which it can support. The 3090J series is currently limited to a maximum of 128 channels. Each channel can of support up to 256 subchannels. These channels can be linked in groups of four to up to 64 controllers, each with up to four device strings. Each string can have up to 32 devices or addressable storage ''actuators'' apiece. This equates to supporting up to 4096 storage addresses per channel of which 256 can simultaneously be active through the subchannel virtual processors.

A DSA system (perhaps VAXcluster) built upon the CI can have up to 32 nodes. With a single host there can be up to 31 HSCs. HSC90s will support up to 48 devices each. This allows for 1488 devices on a CI. A Vail configuration would allow for 31 base nodes (Arapaho cabinets) with up to 64 Aspens or 48 Cedars, equaling 1984 or 1488 devices, respectively. A VAX9000 with up to 16 CIXCDs can have a very large storage subsystem—over 23,000 devices!

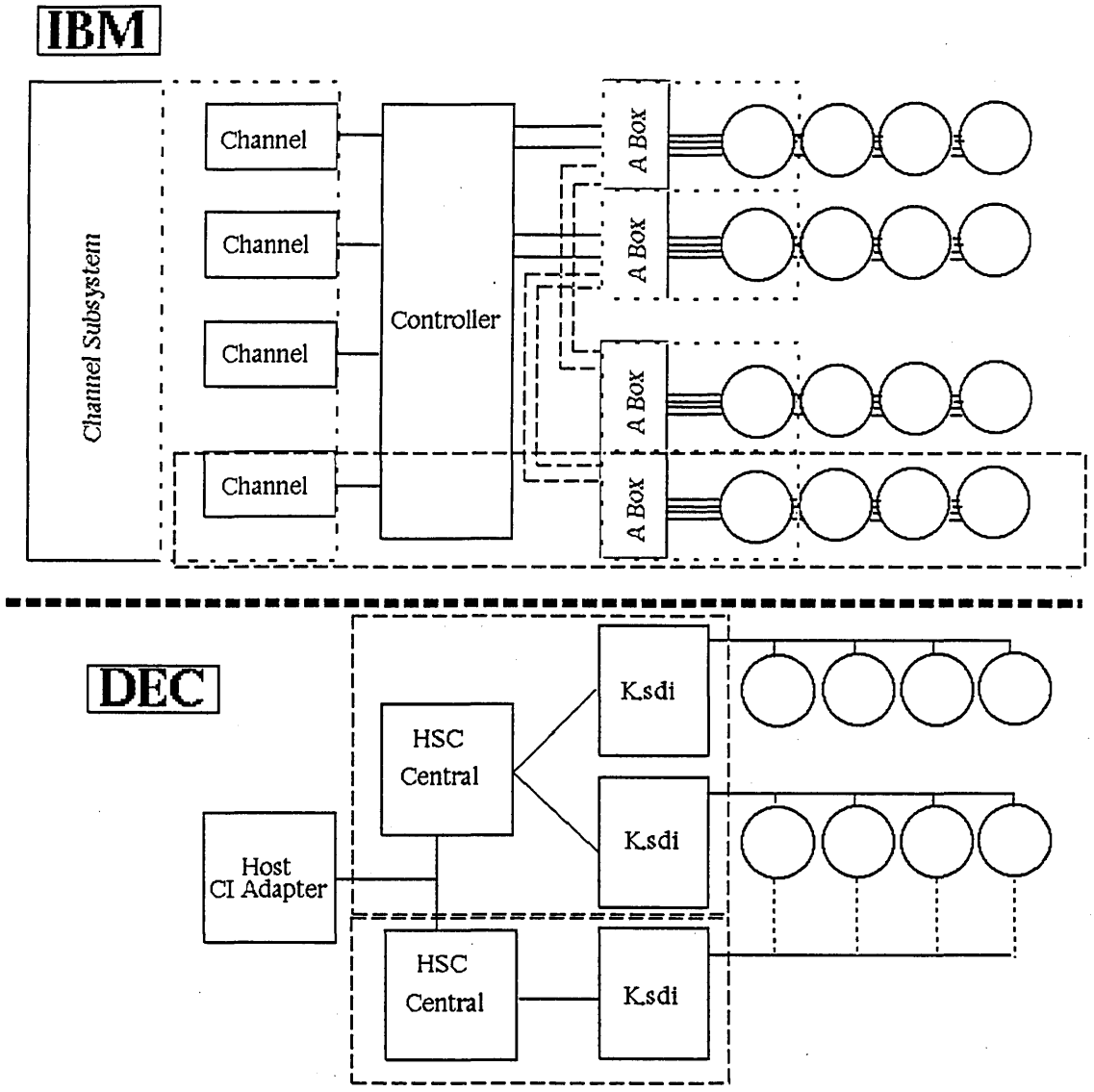
Comparing subsystem bandwidth, each IBM channel is capable of either 3MByte/sec or 4.5MByte/sec. Since the architecture allows for up to 256 channels, an aggregate of over 1 gigabyte/second is possible. A single CI can of supporting 12MByte/sec with both paths. Each host is further restricted by the bandwidth of its single CI adapter. Current adapters are capable of between 1.5MByte/sec and 2.5MBbyte/sec and are constrained to using one path at a time. On the other hand a VAX9000 with 16 CIXCD ports (each capable of 6MByte/sec) can consume 96MBytes per second. Notice that the number of CI ports per host is not an architecture limitation, but a system design limitation.

Digital does not queue requests in the host; IBM does once the virtual subchannel is busy. Looking at the controller level, an HSC70 server can process 1150 requests/second. This is realistically bound to approximately 400. Scaling for an HSC90 produces almost 1400 request/second potential and 480 realizable. An IBM 3990 can process 300 requests/second. This leads to a Digital subsystem capable of 14,880 requests/seconds and an IBM subsystem capable of 19,200 requests/second—ignoring caching or solid-state devices.

3.2.2 *Analysis*

Figure 3-1 demonstrates the physical similarities between IBM's DLSE architecture and Digital's DSA implementation with HSC servers. The dotted box around the A-boxes and the first disk indicates that an A-box contains a device. The dotted box around HSC central and the K.sdi's depicts the physical boundary of an HSC controller. The dotted lines to the bottom set of disks indicates (statically) unused paths to the devices through an alternate HSC.

Figure 3–1: DLSE vs. HSC



The first complication to be addressed is between IBM's channel subsystem and Digital's CI. From a physical perspective, the CI adapter and an IBM channel can be viewed as roughly equivalent. However, IBM has an advantage since they support multiple channels as opposed to a single CI path. The dotted box around channel, controller, A-box, and devices demonstrates the single-threaded view that this would impose. The analogy fails immediately upon recognizing that the CI allows for multiple controllers. A more meaningful comparison is to consider the host CI port and the CI itself to be equivalent to the IBM channel subsystem. The only flaw in this analogy is that currently multiple CI adapters and multiple CI media are not supported.

Comparing an IBM controller with "HSC central" is more direct. Figure 3-2 shows the internal paths through a DLSE controller. Any storage path can be utilized by any connected string at a given instant in time. The supported parallelism is four transfers.

Figure 3-2: DLSE Controller

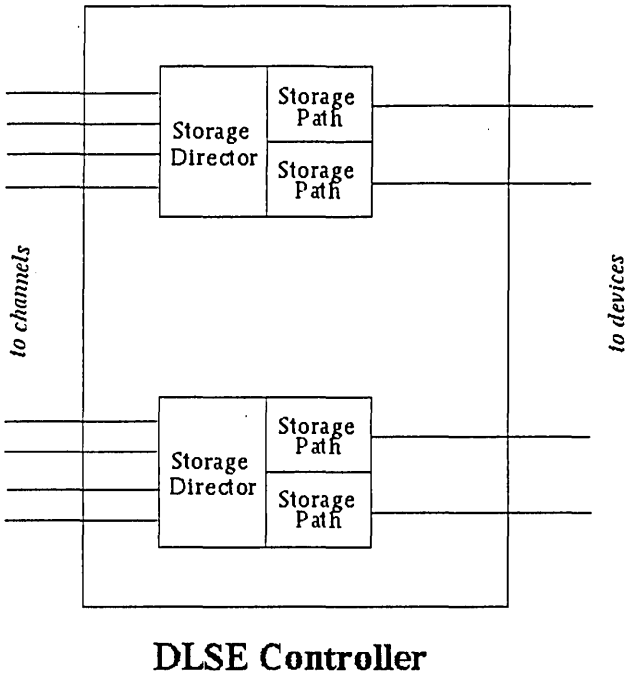
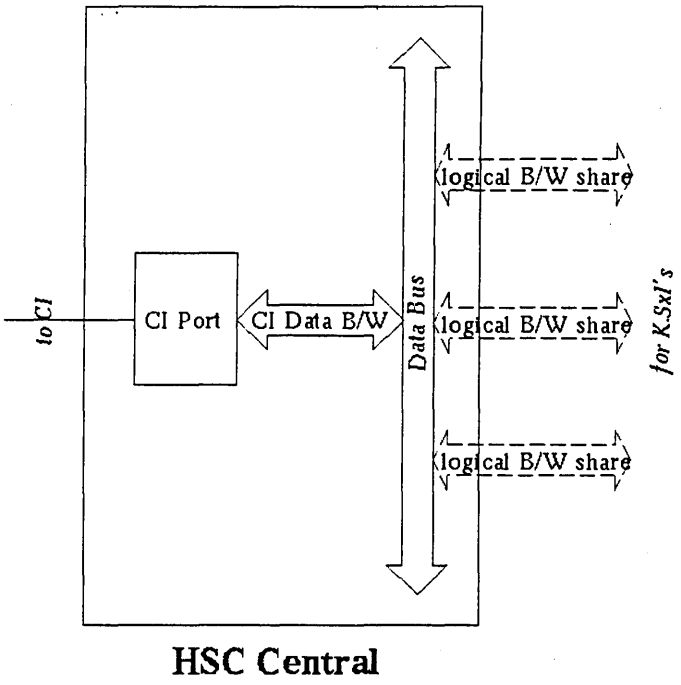


Figure 3-3 shows a very abstract picture of data flow control through the HSC controller. The CI port is guaranteed approximately 6MByte/sec of the databus bandwidth by the internal design. The remaining bandwidth is apportioned among the K.sdi's by logical allocation. In general three parallel disk transfers are supported (up to 5 for slower devices). Unlike IBM which must have a separate tape controller, a single HSC controller can support both tapes and disks.

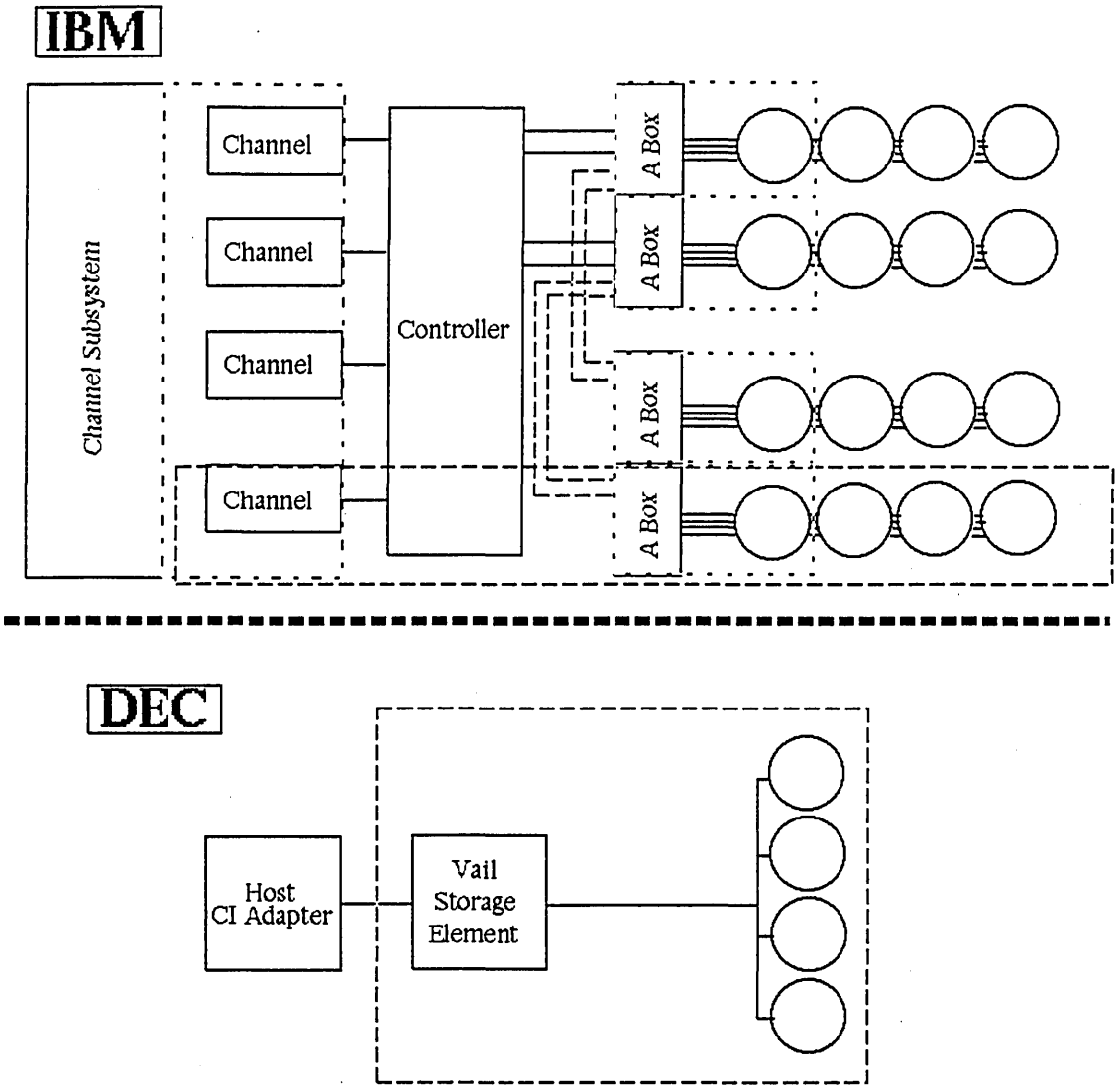
Figure 3-3: Logical internal HSC flow



A set of four DSA devices connected to a K.sdi and an IBM string comprised of A and B boxes are essentially identical. Even though they are connected differently, the A-box and K.sdi are single filters through which four devices transfer data. An IBM device can have a transfer rate up to 4.5MByte/sec while SDI constrains a device to roughly 2.2MByte/sec (once protocol overhead is subtracted).

The Vail architectural changes are demonstrated in Figure 3-4. Replacing SDI with SEBB allows for a transfer rate of up to 50MByte/sec but is realistically constrained to approximately 30MByte/sec. The constraining factor to throughput ends up being the CI at approximately 12MByte/sec with both paths.

Figure 3-4: DLSE vs. Vail



3.3 Anatomy of an I/O

For the purposes of the following discussion, an I/O channel is an abstract path between a host system and a device. From a system perspective, a channel is a resource. As CPU speeds have increased the management of channel resources has become more critical to performance.

A read or write operation to a disk can be broken down into the following phases:

- Host initiation
- Seek
- Rotational positioning
- Data transfer

- Post Processing

A static channel must be dedicated for a particular I/O operation from host initiation until the postprocessing. This is in essence an IBM selector channel. Multiplexer channels allow the channel to be allocated only during initiation, RPS reconnect and data transfer, and postprocessing; however, the same channel is required for all phases. Allowing each phase of the I/O to utilize any channel defines the concept of ''floating channels''. IBM's DLSE essentially implements floating channels from the channel subsystem out to the device. The advantage of floating channels is the reduction in path contention since any phase of an I/O can complete as long as any channel is available.

Digital's CI can be viewed as a pair of floating channels, since packets for a given I/O can be sent across either path. From the perspective of a device back to the host, the SDI only allows ''static dual porting'' where only one path to the device is possible at a time. Vail will have no second port at the SEBB, but dual porting may be supported by a second CI adapter.

With a head-on comparison, Digital's CI out performs a single IBM channel. It takes three IBM channels to match the throughput potential. In DLSE implementations, 4 channels are connected to the individual storage devices. A VAX9000 with 16 CI ports has an I/O subsystem roughly equivalent to an IBM system with 48 channels.

3.3.1 *Host Initiation*

In the IBM/360 the SIO (START I/O) instruction was expensive. It required an initial interrupt as well as channel allocation. SIOF eliminated the initial interrupt, but there was still the potential for a subsequent interrupt upon channel allocation failure. By extensive use of microprocessing, the current IBM channel subsystem has eliminated SIOF failure interrupts with the new START SUBCHANNEL (SSCH) command.

The VAX queueing instructions are used to queue requests into ''physical'' adapter memory. The overhead involved in these queueing operations is on the same scale as the SSCH command.

The real difference between IBM and Digital host initiation is logical software. Although IBM does not have a larger point-to-point channel scheme, the microprocessing in the channel subsystem has offloaded the host operating system from having to dynamically select paths. The physical path selections are handled inside the channel subsystem. The question then becomes one of weighing the relative expense between building an IBM channel program versus the expense of building an MSCP packet and passing it through the port driver. This basic service time favours IBM; however, queuing time is another consideration.

With the Digital architecture, request queuing takes place at the adapter rather than inside the VAX. The constraint today is the bandwidth of the adapters. The IBM architecture causes request queueing to occur in the host whenever a subchannel is unavailable. With today's channel subsystem, this does not occur unless all possible paths to a device are already busy. Theoretically IBM's architecture is more finite because the subchannels are restricted by configuration. Queueing at a Digital adapter is tied more to adapter bandwidth than to a request depth. The increased bandwidth of the CIXCD should significantly reduce the queueing at the host adapter, but one can expect the bottleneck to move to either the CI wire itself or into the controllers (either for their CI ports or availability of buffer memory).

3.3.2 *Seek*

Seek times are overlapped for both architectures. Seek profile comparisons of various workloads between the two vendors are of interest at the device level rather than as an architectural concern.

3.3.3 *Rotational Positioning and Data Transfer*

As with seek, rotational latency is of interest at the device level; however, the delay induced by a lost revolution is an architectural concern. RPS miss is the loss of a device revolution when a path to memory is not available.

3.3.3.1 *Non-realtime Transfer*

The realtime path for IBM is closer to the host than Digital's. Once the data is copied into some buffer memory, RPS miss is replaced with communications latency and other queueing delays. RPS miss in the Digital architecture is not as comprehensive as RPS miss in the IBM architecture. The Digital I/O subsystem has more queueing delay points since the buffering is further from the host. It can be argued that IBM RPS miss is equivalent to HSC RPS miss plus the queueing delays at K.ci, the CI path, and the host adapter. Vail would have the same delays; however, the CI path will become a bottleneck to the overall workload (with multiple storage elements) before the Vail CI adapter will.

The IBM channel subsystem has the communications delay associated with each link in the chain, plus the channel subsystem overhead and the storage director overhead. However, other than cache implementations (covered later) there is no buffering of data anywhere in the channel.

Digital's communications overhead is more complex than IBM's. The HSC server overhead includes the cost of the MSCP to SDI interpretation, SDI overhead, and the transfer from data memory to the CI. Queueing delays to the SDI interface and delays at the CI adapters at both the host and HSC server should be added with RPS miss for a valid comparison with IBM RPS miss. Vail eliminates the MSCP to SDI translation; however, the potential latency between data memory and the CI port still exists. Although each Vail CI port is expected to be capable of 6 to 12 MByte/sec, the CI path becomes a potential bottleneck with multiple storage element configurations.

3.3.3.2 *RPS Miss Quantification*

In the past, RPS miss was a very significant contributor to queueing time in the IBM I/O subsystem. Each of the steps to multiplexer channels, to DLS, and to DLSE have been primarily to reduce RPS miss. Appendix A displays the RPS miss quantifications for equivalent 32-actuator DLSE, HSC, KDM70, and Vail configurations. Uniform loading across actuators is assumed. RPS miss for HSC is predominantly attributable to multiple drives sharing a single K.sdi. KDM70 will only incur RPS miss when devices outnumber the two state machines.

The delay associated with RPS miss is determined by first finding the probability of an RPS miss. Then the probability is used to calculate the expected number of misses (as a fraction) per successful I/O. The RPS miss overhead is the product of expected misses and the time for one rotation of the device.

3.3.3.2.1 *IBM model*

A DLSE four-channel set behaves like an M/M/4‡ [KLEI]. An Erlang-K distribution is used to calculate the probability of all paths being busy when a reconnect is attempted.

Example 3-1: DLSE Four 3380 Strings

```
Given:
 1. 23.5 IOs/second optimal for a 3380K AT 35msec Resp
    time
 2. 4096 blocksize
 3. 3.0MB/s channel speed
 4. 32 3380K actuators behind a 3990-M2 storage controller
 5. Utilization of a single path using M/M/4:
                    Path Busy per SIO - (Path Util by Device / 4)
 SingleUtil(SU) = ---------------------------------------------
                     (1 - (Path Util by Device / 4)
 6. Probability that a device reconnecting will find all
    four storage directors busy due to other devices
    by Erlang-4:
                              32 * SU^4
    P(4Busy) = ---------------------------------------
               (3 + (9 * SU) + (12 * SU^2) + (8 * SU^3)
 7. RPS miss due to paths busy:
                          16.6msec * P(4Busy)
                   RPS = -------------------
                              1 - P(4Busy)
implies:
 1. 781 IOs per second; 195.25 per path
 2. Each path is 58.6% busy
 3. Each device contributes 7.3% to path busy
 4. Path service time is 3.0msec including channel to HOS
therefore,
 P(4Busy) = Probability of four directors busy = 25.9%
 RPS miss = .0166 * .259 / 1 - .259 = 5.8 msec
```

3.3.3.2.2 *Digital models*

Each Digital product exhibits different RPS miss behaviour because buffering is done within the "controller", and each controller has a different topology.

3.3.3.2.2.1 *HSC*

Quantifying RPS miss for Digital HSC-based systems requires two separate analysis points.

1. K.sdi
2. the data bus

RPS miss is only relevant between data memory and the device. The Example 3-2 calculates the delays for a fully loaded HSC controller with RA90s. The assumption that all devices are evenly loaded (to 44ms response) is done to provide a worst-case analysis. As demonstrated the data bus is a non-issue, contributing less than 2 microseconds to aggregate delay. The contention for K.sdi ports is significant, accounting for almost 5 milliseconds of delay.

‡ Kendall's notation

Example 3-2: HSC RPS miss

```
   Fully Configured K.sdi

Given:
 1. 38 requests/second max for RA90 for typical workload
 2. 8 sectors/request for typical workload
 3. 2.1MB/s SDI max throughput
 4. 4 RA90s on a K.sdi
 5. RPS miss due to path delay
           sum(other device utilization)
    P(RPS miss) = ----------------------------
    1 - utilization by this device

implies:
 1. 304 sectors/second per RA90
 2. 4300 sectors/second per K.sdi
 3. each RA90 uses 7% of SDI

therefore,
 P(RPS miss) = .21/.93 = 22.6%
 #misses/IO  = .226/.784 = .288

aggregate delay:
 .288 * 16.666... = 4.80 msec

NOTE: 8-port
 P(RPS miss) = .49/.93 = 53%
 #misses/IO  = .53/.47 = 1.13
 ===> K.sdi will not support 38 req/sec/device

   Data Bus Contention

Given:
 1. 38 requests/second max for RA90 for typical workload
 2. 8 sectors/request for typical workload
 3. 4 RA90s on a K.sdi
 4. 8 K.sdi's each with 4 RA90s
 5. 3 concurrent transfers supported by data bus
 6. P(RPS miss) = P(4 concurrent transfers)

implies:
 1. 1216 sectors/second per K.sdi
 2. 13516 sectors/second on data bus (6.6MB/s)
 3. each K.sdi uses 9% of data bus

therefore,
 P(RPS miss) = SUM[(.09)^i] for i in [4,8] = .007%
 #misses/IO  = .007/.993 = .00007

aggregate delay:
 .00007 * 16.666... = 1.1 usec

NOTE: 8-port K.sdi:
 each K.sdi (can) use 18% of data bus
 P(RPS miss) < (.18)^4+..+(.18)^8 = .1%
 #misses/IO < .001
 ==> delay < 17.5usec
```

Total RPS delay

```
RPS delay = K.sdi contention + databus contention
   = 4.8ms + 1.1us = 4.8ms
```

The following examples analyze an RA70 and an RA92. These examples demonstrate the effect of device rotational speed upon the actual impact of RPS miss. Notice that the expected number of misses is dependent upon workload rather than device.

Example 3-3: HSC RPS with RA70

```
   Fully Configured K.sdi

Given:
 1. 38 requests/second max for RA70 for typical workload
 2. 8 sectors/request for typical workload
 3. 2.1MB/s SDI max throughput
 4. 4 RA70s on a K.sdi
 5. RPS miss due to path delay
           sum(other device utilization)
    P(RPS miss) = ----------------------------
    1 - utilization by this device

implies:
 1. 304 sectors/second per RA70
 2. 4300 sectors/second per K.sdi
 3. each RA70 uses 7% of SDI

therefore,
 P(RPS miss) = .21/.93 = 22.6%
 #misses/IO  = .226/.784 = .288

aggregate delay:
 .288 * 15.0 = 4.32 msec
```

Example 3-4: HSC RPS with RA92

```
   Fully Configured K.sdi

Given:
 1. 38 requests/second max for RA92 for typical workload
 2. 8 sectors/request for typical workload
 3. 2.1MB/s SDI max throughput
 4. 4 RA92s on a K.sdi
 5. RPS miss due to path delay
           sum(other device utilization)
    P(RPS miss) = ----------------------------
    1 - utilization by this device

implies:
 1. 304 sectors/second per RA92
 2. 4300 sectors/second per K.sdi
 3. each RA92 uses 7% of SDI

therefore,
 P(RPS miss) = .21/.93 = 22.6%
 #misses/IO  = .226/.784 = .288

aggregate delay:
 .288 * 17.62... = 5.07 msec
```

K.sdi saturation can be analyzed by using the single server model in the Appendix A for RPS miss by setting the expected number of misses per I/O to 1. It is interesting to observe the effect of increasing from 4 ports to 8 ports on a K.sdi.

```
given: u = K.sdi utilization per device
 x = probability of a miss
 m = expected # misses per successful I/O

  m = x/(1-x) = 1
  ==> x = 1-x    ==> x = .5

 for 4-port K.sdi
  ==> 3u/(1-u) = .5
  ==> 3.5u = .5
  ==> u = 14.3%    (307.5KB/s per device or 1230KB/s)

 for 8-port K.sdi
  ==> 7u/(1-u) = .5
  ==> 7.5u = .5
  ==> u = 6.66%    (143KB/s per device or 1144KB/s)
  NOTE: saturates under 36 req/s with 4KB requests
```

To keep things in perspective, an HSC70 controller supporting 32 RA90s at 38 requests/second would contribute approximately 4.8 milliseconds of RPS queueing delay at 1216 requests/second, or 4.75MByte/sec. The HSC70 controller can currently only process 1150 request/second and 4.2MByte/sec. The P.io limits the request handling and K.ci limits the data rate. 48 RA90s on an HSC90 at 38 requests/second per drive would be 1824 requests/second and over 7 MBytes/sec overall. An HSC90 P.io is projected to decompose about 1400 requests/second while the new K.ci should support under 5 MBytes—the bottlenecks remain the same..

3.3.3.2.2.2 *KDM70*

The internal bus does not contribute to RPS miss because its speed is greater than twice the speed of SDI, and there are only two state machines to perform transfers. The KDM70 behaves like an M/M/2‡ queueing system [KLEI]. The probability of both state machines being busy when a device is ready to reconnect follows an Erlang-K distribution, where K equals 2. It analogous to IBM's DLS subsystems. To support 32 devices, 4 KDM70s are necessary. In Example 3-5 the XMI has been ignored because each KDM70 can produce at most 5.4MBytes/sec. Therefore, 4 KDM70s can support 21.6MBytes/sec on a 66MByte/sec XMI bus.

‡ Kendall's notation

Example 3-5: KDM70 RPS

```
given:
 1. 30 requests/second per RA90
 2. 8 sectors/second
 3. 2.1MByte/s SDI max rate
 4. 8 RA90s
 5. 2 state machines
 6. 16MByte/sec internal bus

implies:
 1. 240 sectors/second per RA90
 2. 4300 sectors/second per state machine
 3. each I/O uses .186% of a state machine
 4. each RA90 uses 5.58% of a state machine
 5. each state machine is 22.3% utilized
 6. Residual SM Busy is 22.3% - 2.76% ~= 19.5%

therefore,
 Residual utilization of a single state machine per device
                    Residual SM Busy - (SM Util by Device / 2)
 SingleUtil(SU) = ------------------------------------------
                    (1 - (SM Util by Device / 2)

         = (.195-.028)/.972 = .172

 Probability that a device reconnecting will find all
 both state machines busy due to other devices
 by Erlang-2:
                2 * SU^2
    P(2Busy) = ---------  = 2*(.172)^2/1.172 = .050
                1 + SU

aggregate delay:
     16.6msec * P(2Busy)
     ------------------- = 16.6*.050/.950 = .877msec
        1 - P(2Busy)
```

3.3.3.2.2.3 *Vail*

Because of the bandwidth of the SEBB, the initial Vail offerings will significantly reduce RPS miss. With the SEBB being 50MByte/sec and the CI being around 12MByte/sec, there is approximately a 25MByte/sec pad for RPS prevention over the CI bottleneck point. At one request per revolution, four Cedars will provide a load (8KB per request) of 1920 sectors/second, or 3% of the SEBB. Because the MOCHA chips actually have 2 sectors worth of buffering the single path model has to modified slightly. The SEBB is a synchronous bus that provides for bursts of up to 8 bytes. The overhead of 3 or 4 cycles per burst (depending upon write or read to/from buffer memory) diminishes the usable bandwidth of the SEBB to about 30MByte/sec; however, the MOCHA buffering implies that 32 consecutive arbitration rejections must occur before the pipeline can fill and an actual RPS can occur. Assuming independent arbitration attempts leads to taking .03 to a power of 32, leading to a negligible probability.

Example 3-6: SEBB Contention

```
   SEBB Contention

Given:
 1. 38 requests/second per CEDAR/ASPEN
 2. 8 sectors/request for typical workload
 3. 30MB/s SEBB max throughput
 4. 4 CEDAR/ASPEN on SEBB
 5. P(RPS miss) on SEBB with MOCHA buffering
                              sum(other device utilization)
    P(arbitration reject) = ----------------------------
                              1 - utilization by this device

    P(RPS miss) = P(arbitration reject)**32

implies:
 1. 304 sectors/second per CEDAR/ASPEN
 2. 61440 sectors/second on SEBB
 3. each CEDAR/ASPEN uses .49% of SEBB

therefore,
 P(RPS miss) = (.010/.995)**32 = (.010)**32 ~= 0 for CEDAR
             = (.015/.995)**32 = (.016)**32 ~= 0 for ASPEN
 #misses/IO  = 0/.995 = 0 for both CEDAR/ASPEN

aggregate delay:
 0 * 16.666... = 0. usec for CEDAR
 0 * 11.111... = 0. usec for ASPEN
```

3.3.3.3 *Cumulative Delays*

A comparison of RPS miss is inadequate to compare the throughput potential between IBM DLSE and the Digital I/O subsystems. Digital subsystems have additional queueing delays. The cumulative delays for all subsystems are plotted in Appendix B.

3.3.3.3.1 *HSC*

The RPS miss asymptote for an HSC controller is around 4000 requests per second. An HSC70 P.io cannot process more than 1150 requests per second, nor can the K.ci transfer more than 4.2 Megabytes per second. For the sake of simplicity the delays for both are modelled as M/M/1 queueing systems in Example 3-7.

```
M/M/1 queueing model queueing time formula:
```

$W_q = \lambda / (\mu(\mu - \lambda))$
where
W_q = queueing time
λ = arrival rate
μ = service rate

Example 3-7: HSC Cumulative Delay

K.SDI miss

Given:
1. 30 requests/second max for RA90 for typical workload
2. 8 sectors/request for typical workload
3. 2.1MB/s SDI max throughput
4. 4 RA90s on a K.sdi
5. P(RPS miss) = sum(other device utilization) / (1 - utilization by this device)

implies:
1. 240 sectors/second per RA90
2. 4300 sectors/second per K.sdi
3. each RA90 uses 5.6% of SDI

therefore,
P(RPS miss) = .168/.944 = 17.8%
#misses/IO = .178/.822 = **.216**

aggregate delay:
.216 * 16.666... = **3.60 msec**

DATABUS miss:
Given:
1. 30 requests/second max for RA90 for typical workload
2. 8 sectors/request for typical workload
3. 4 RA90s on a K.sdi
4. 8 K.sdi's each with 4 RA90s
5. 3 concurrent transfers supported by data bus
6. P(RPS miss) = P(4 concurrent transfers)

implies:
1. 960 sectors/second per K.sdi
2. 13516 sectors/second on data bus (6.6MB/s)
3. each K.sdi uses 7.1% of data bus

therefore,
P(RPS miss) = SUM[(.071)^i] for i in [4,8] = .0025%
#misses/IO = .000025/.999975 = **.000025**

aggregate delay:
.000025 * 16.666... = **.42 usec**

TOTAL miss:

3.6msec

P.io Queueing Delay

1. λ = 960 req/sec
2. μ = 1150 req/sec

$W_q = \lambda / (\mu(\mu - \lambda)) = 960/(1150(1150-960)) =$ **.0044sec**

K.ci Queueing Delay

1. λ = 7680 sectors/sec
2. μ = 8602 sectors/sec

$W_q = \lambda / (\mu(\mu - \lambda)) = 7680/(8602(8602-7680)) =$ **.00097sec**

```
Total Delay = RPS miss + P.io queueing + K.ci queueing
     = 3.6 + 4.4 +.97 = 8.97 msec delay
```

Consider that an HSC90 P.io will handle about 1400 requests/second and the K.ci will support about 5 MBytes/second, the P.io queueing becomes 1.56msec; and K.ci queueing reduces to 293usec. Since the RPS miss remains the same (for the given configuration), the total cumulative delay would reduce to 4.85msec.

3.3.3.3.2 *KDM70*

Both the XMI interconnect and the internal KDM70 bus are fast enough that no queueing delays can occur for either bus. However, queueing delays do occur at the processor. A M/M/1† is used for simplicity. Again, there is no additional delay at the XMI bus with 4 KDM70s.

Example 3-8: KDM70 Cumulative Delay

```
given:
 1. 30 requests/second per RA90
 2. 8 sectors/second
 3. 2.1MByte/s SDI max rate
 4. 8 RA90s
 5. 2 state machines
 6. 16MByte/sec internal bus

implies:
 1. 240 sectors/second per RA90
 2. 4300 sectors/second per state machine
 3. each I/O uses .186% of a state machine
 4. each RA90 uses 5.58% of a state machine
 5. each state machine is 22.3% utilized
 6. Residual SM Busy is 22.3% - 2.76% ~= 19.5%

therefore,
 Residual utilization of a single state machine per device
                  Residual SM Busy - (SM Util by Device / 2)
 SingleUtil(SU) = -----------------------------------------
                  (1 - (SM Util by Device / 2)

        = (.195-.028)/.972 = .172

 Probability that a device reconnecting will find all
 both state machines busy due to other devices
 by Erlang-2:
              2 * SU^2
   P(2Busy) = ---------  = 2*(.172)^2/1.172 = .050
               1 + SU

aggregate delay:
     16.6msec * P(2Busy)
     ------------------- = 16.6*.050/.950 = .877msec
        1 - P(2Busy)


   P.mscp Queueing Delay
```

1. λ = 240 req/sec
2. μ = 700 req/sec

Example 3-8 Cont'd on next page

† Kendall's notation

Example 3-8 (Cont.): KDM70 Cumulative Delay

$W_q = \lambda / (\mu(\mu - \lambda)) = .240/(700(700-240)) = .75$**msec**

```
Total Delay = RPS miss + P.mscp queueing
     = .877 + .75 = 1.63 msec delay
```

A KDM70 solution significantly reduces cumulative delay over an HSC solution. The XMI bus is much faster than the CI interconnect. RPS miss is reduced by the ''floating channels'' of having state machines. Four KDM70s have less queueing delay at the processor than a single HSC P.io.

3.3.3.3.3 *Vail*

Example 3-9 shows the delays associated with 8 storage elements with 4 Cedar HDAs each. The CI delay can either be in the adapter or in the CI paths. For Example 3-9 the load is light enough that the queueing occurs in the adapter. Workloads with a high enough data rate will experience queueing in the CI paths.

Example 3-9: Vail Cumulative Delay

```
   SEBB Contention

Given:
 1. 38 requests/second per CEDAR/ASPEN
 2. 8 sectors/request for typical workload
 3. 30MB/s SEBB max throughput
 4. 4 CEDAR/ASPEN on SEBB
 5. P(RPS miss) on SEBB with MOCHA buffering
                            sum(other device utilization)
    P(arbitration reject) = ----------------------------
                            1 - utilization by this device

    P(RPS miss) = P(arbitration reject)**32

implies:
 1. 304 sectors/second per CEDAR/ASPEN
 2. 61440 sectors/second on SEBB
 3. each CEDAR/ASPEN uses .49% of SEBB

therefore,
 P(RPS miss) = (.010/.995)**32 = (.010)**32 ~= 0 for CEDAR
             = (.015/.995)**32 = (.016)**32 ~= 0 for ASPEN
 #misses/IO  = 0/.995 = 0 for both CEDAR/ASPEN

aggregate delay:
 0 * 16.666... = 0. usec for CEDAR
 0 * 11.111... = 0. usec for ASPEN


   CI Adapter Delay
```

1. λ = 1216 sectors/sec
2. μ = 24576 sectors/sec (12MB/sec)

$W_q = \lambda / (\mu(\mu - \lambda)) = 1216/(24576(24576-1216) =$ **2.1usec**

Example 3-9 Cont'd on next page

Example 3-9 (Cont.): Vail Cumulative Delay

```
    CI Path Delay
```

1. λ = 1216 sectors/sec
2. μ = 2448 sectors/sec (1.5MB/sec)

$W_q = \lambda / (\mu(\mu - \lambda)) = 1216/(2448(2448-1216) =$ **403usec**

```
    P.mscp Queueing Delay
```

1. λ = 152 req/sec
2. μ = 700 req/sec

$W_q = \lambda / (\mu(\mu - \lambda)) = 152/(700(700-152)) =$ **.40msec**

```
Total Delay = RPS miss + CI adapter queueing + P.mscp queueing
     = 0. + .0021 + .4 = .402 msec delay for CEDAR
     = 0. + .0021 + .4 = .402 msec delay for ASPEN
```

Overall, Vail appears to reduce RPS miss by the speed of the SEBB. It also improves MSCP interpretation queueing times by increasing the number of processors for a given number of HDA over the HSC server. Although the Vail CI adapter is to be capable of approximately 12MByte/sec, the number of storage elements on the CI becomes a divisor of the overall CI bandwidth. This means that a configuration with eight storage elements will reduce the usable bandwidth for a single storage element to approximately 1.5MByte/sec, assuming balanced loading. The CI now becomes the potential bottleneck. Multiple CIs allows configuration flexibility to alleviate this bottleneck by spreading storage elements across base nodes. There are packaging issues to consider..

3.3.4 *Post Processing*

One of the weaknesses in the IBM I/O architecture has been the processing of simultaneous interrupts from multiple devices. The IBM/370 architecture only defines a single cell for an I/O termination interrupt. Interrupts are serialized until appropriate addressabillity is established in the interrupt handler to save current context. This can require multiple intermediate context saves before a subsequent interrupt can occur.

The multiple stacks of the VAX architecture greatly simplify the handling of simultaneous interrupts. The use of IPLs for serializing segments of interrupt processing facilitates interrupt handling (via forking) and reduces overhead. IPLs use firmware to effect system ''processes'' as opposed to using software simulation like IBM must.

Although the VAX provides an advantage in the handling of simultaneous interrupts, VMS SMP has a disadvantage in post processing as opposed to MVS. MVS uses ''prefixing'' to allow post processing to occur on any processor. The only limitations are the actual physical connections imposed by the configuration. In VMS SMP post processing is forced to processor affinity to a ''control processor''—based upon the adapter port. The ''control processor'' can become a bottleneck to an I/O intensive workload.

3.4 Characteristic Workload Differences

Configuration and workload essentially dictate the performance of a computer system. System architecture not only defines potential configurations, but it also leads to certain characteristics of the workloads that systems built upon the architecture will process. More precisely, the initial implementations of an architecture dictate characteristics such as request sizes. Smaller request sizes presented to the I/O subsystem implies an increase in access time delay (latency) for an equivalent amount of data. There are three components of current IBM systems that generally reduce request rate to provide the data needed to satisfy a CPU's "I/O appetite":

1. Disk Data Format
2. Caching
3. Expanded Storage

In referring to the graphs in the appendices, it is important to remember that IBM workloads will typically present significantly fewer I/O requests per second to the I/O subsystem than a Digital workload will present to the Digital subsystem. The result is that IBM can satisfy the I/O appetite of a faster CPU with less inherent delay.

3.4.1 *Request Rate versus Data Rate*

Data rate is the amount of data that is made available to the CPU (via the I/O subsystem) for processing. Amdahl's law states that the required data rate is (roughly) proportional the power of the CPU. There is no equivalent law that ties CPU power to request rate. CPU utility implies processing of raw data, not I/O requests. Providing a given amount of data via more (smaller) I/O requests has the following effects:

- Increase in aggregate access time delays to random storage devices
- Increase in context switching in CPU

Increasing aggregate access time delays leads to poorer single threaded performance. It also diminishes path utilization. Although diminished path utilization decreases path delays, the aggregate access time delays are more significant - until path saturation is reached. Access time delays are typically on the order of 20ms (12ms seek and 8ms rotational latency). As can be seen from Appendix B, path delays are less than half of this figure until near path saturation.

A workload with more, smaller I/O requests inherently cause more context switching because more I/O termination interrupts occur. There are actually three reductions in CPU utility caused by increased overhead for I/O processing:

1. I/O initiation service call handling
2. Context switching overhead
3. I/O posting overhead

3.4.1.1 *Bandwidth*

Transmission of smaller amounts of data can decrease the usable bandwidth of a path. The IBM channel is less prone to this effect than Digital's CI because the IBM channel interface is point-to-point. Digital's CI can experience a significant reduction in bandwidth due to the increase in arbitration time slots to transmit an equivalent amount of data. For example, a 3Kbyte transfer between a host with a CI780 and an HSC with a 512byte K.ci requires 6 data packets to be transferred. The same transfer between a VAX9000 with an XCD and an HSC with a 4Kbyte K.ci can be done with one data packet. The former requires 5 additional "handshake" packets as well as arbitration for 5 additional data packets. Therefore, the decrease in usable bandwidth by smaller transfers is 10 arbitration times and the transmission time for 5 "handshake" packets.

3.4.2 *Disk Data Format*

The count-key-data format of IBM devices has historically forced users to be aware of the effects of block sizes upon performance as well as capacity utilization. With the current 3380 devices, typical block sizes are on the order of 22Kbytes. 3390 will increase this to more like 25Kbytes per block. Ignoring arguments about user-friendliness, the consequence of this awareness is that sequential access streams for an IBM workload will be on the order of 22Kbytes per request. This does not mean that the average MVS installation uses 22-25Kbyte blocksizes, both database and VSAM organizations are optimally blocked at 4Kbytes. Digital workloads have been measured to typically have request sizes averaging between 4Kbytes and 8Kbytes. This means that Digital workloads may exhibit nearly three times the request rate of an IBM workload in order to provide the same amount of data to the CPU.

3.4.3 *Caching*

Caching further reduces the request rate seen by the I/O subsystem. In general, any cache hit eliminates a physical I/O request. However, pre-fetching data in larger chunks than originally requested can also reduce request rates to the external subsystem for sequential access streams.

3.4.4 *Expanded Storage*

IBM's expanded storage introduces a whole new level in their storage hierarchy, between main memory and controller caches. Main memory is accessed synchronously (at a hardware level) by the CPU. Controller cache memory is accessed asynchronously by the I/O (software) subsystem. Expanded storage is accessed synchronously by the software subsystem. That is, filling main memory from expanded storage occurs in less time than a context switch, so the operating system does not remove the currently executing process while the data movement transpires. The results are improved single-threaded performance and a reduction in context switches.

Digital can use VAX main memory to compete with expanded storage - for now. VAX memory and 3090 implementations of expanded storage are both limited to 2gigabytes. Global sections actually allow better performance than expanded storage, but expanded storage implementations can grow to 16Terabytes. Entire databases may be loaded into expanded storage in the future (by HSM, for example). Overall performance may be dramatically reduced by virtually eliminating access delays imposed by rotating media.

3.5 Caching

Caching is a technique that gives the appearance of a large, fast memory from the combination of a small, fast memory and a slower, large memory. Data migrates to and from the small, fast memory to take advantage of frequent access. For storage caches, the slower memory is typically a rotating device such as a disk, while the cache memory is RAM. Finding data in the cache that coincides with a request is referred as a cache hit.

There are two perspectives when analyzing the effects of caching on the I/O subsystem. From the device perspective caching tends to lower the request rate, thereby decreasing the overall response time of the devices by reducing queueing time. From the system perspective the request rate throughput increases because the service time from cache is significantly less than that of the devices.

Although caching impacts performance, the impact is not architectural. The effect of implementing a cache can be approximated by reducing the request rate of a given workload to that of cache misses only.

3.5.1 *Principles*

The basic premise under which cache works is that data has a locality of reference, just as instruction locality of reference works for CPU caches. Locality may be either spatial or temporal. Spatial locality assumes that there is high probability that related records are located nearby. Temporal locality is the premise that a record will be used again in the near future.

Caches are classified by the handling of writes. Caches that send completion notices before writing the data to the physical media usually out perform those that only capitalize upon the presence of read data.

3.5.1.1 *Read Policies*

Readahead refers to reading more data than requested based upon spatial locality. Replacement is not directly affected by this policy.

Prefetch is the reading of anticipated data for sequential and random access patterns. The immediately requested data is not loaded into the cache once the data is sent because the cache assumes that the data will not be read again. Sequential algorithms dictate that following blocks will be needed and are staged into the cache; random algorithms specify that the next likely block required will be either the preceding or following data. With sequential access prefetched blocks are released once requested since the record will not be needed again; random access maintains the data in the cache under a least recently used purge algorithm.

3.5.1.2 *Write Policies*

A cache can handle writes in a numbered of ways. The simplest action for a cache is to ignore the write other than to invalidate an entry for the block if it is present. This is referred to as ''writethrough'' cache without write allocation. The writethrough cache could be extended to include write allocation, but the cache update must occur after the write to physical media.

The next alternative write policy is ''write-behind'' or ''fast write''. Response time is improved for writes by considering the cache itself to be the destination for the data and returning an I/O completion. The write to the physical media is performed according to some subsequent scheduling algorithm that attempts not to impede read requests but optimize write ordering. Making the cache nonvolatile is imperative.

"Writeback" caching goes one step beyond write-behind caching by trying to completely eliminate some write requests. The theory behind this cache policy is that data once written may again be written soon, thus avoiding the first write to physical media as unnecessary. However, when a reallocation of a block has not been written to physical media, the subsequent operation must wait for the "dirty" write to complete.

3.5.1.3 *Replacement*

The first choice of any cache placement should be to utilize blocks that are considered free. Sequential access recognition and write invalidation provide means by which a free list of blocks might be retained.

The main replacement strategies are sequential and least recently used (LRU). They both apply to read replacements. The sequential strategy relies on identification of a sequential access stream to purge blocks for reuse as they are read from the cache. LRU works under the assumption of temporal locality. Blocks are taken from an LRU list only when necessary for a new allocation.

Writes affect replacement by either invalidation or allocation. Write invalidation occurs for a write hit when cache is not filled by write data. Such a strategy only makes sense for a writethrough cache. Write allocation can be applied to writethrough, write-behind, or writeback cache. When write allocation with writethrough caching is used, a source of free blocks is eliminated, but the tradeoff is the probability of use of written data in the future by reads.

3.5.2 *IBM*

A host software cache was not as feasible as controller cache for IBM because of the architectural limit on host central memory. ESA's implementation of expanded storage is in essence a host cache allocated through supervisor services. The current implementation of central storage on an IBM 3090-600S tops out at 512 megabytes with an additional 2 gigabytes attachable through expanded storage. The 3090 limit on central storage is 2 gigabytes; for expanded storage it is 16 terabytes. The limitation of expanded storage is physical, not architectural. Expanded storage looks like any other storage device; however, the path between expanded storage and central storage is so fast that the data movement is faster than the path for the operating system software to perform a context switch to another process. As a consequence, I/O to/from expanded storage is performed synchronously, without any CPU rescheduling being performed. From a physical perspective, I/O with expanded storage differs from disk I/O in that expanded storage is contained within the processor complex as opposed to a disk which is disjoint from the processor.

3.5.2.1 *Controller Cache*

IBM offered cache in the past with the 3880-M13/M23 storage controllers; the current cache product is the 3990-M3. Cache sizes for this controller are 32, 64, 128 or 256 megabytes. This cache controller is segmented in 16Kbyte areas and supports up to eight concurrent operations. The 3990-M3 uses the LRU algorithm for management of cache segments.

3.5.3 *Digital*

Today Digital does not have a cache product offering on the market. For the last 2 years, engineering has been evolving a multilevel caching strategy.

Qualitatively, caching is more effective when it is closer to where its data will be used. Thus, keeping the cache in the host primary memory has the least latency. The tradeoff is system memory and CPU cycles. With the high architectural limit on primary memory for the VAX, a host memory cache is feasible. It is unclear whether CPU cycles that are used to perform the cache move would not sit idle waiting for an actual I/O operation.

3.5.3.1 *Host Software Cache*

Host caching of shared data across systems must update all participating systems within a VAXcluster. The update notifications add latency to the operation and consume CI bandwidth. As yet the effects of the update notifications have not been quantified against real workloads.

3.5.3.2 *HSC cache*

The HSC 32MByte cache will be Digital's first implementation of controller-based cache. Initial implementation will be a writethrough cache. ''Fast write'' is being implemented independently of the cache. It will require two physical writes to the device. The first write will go to a reserved surface and written at the first possible sector. A log will be kept in a RAM on the device. The host end notification is then sent, followed by scheduling of the actual target write (complete with seek and rotational delay). The data is retained on the special surface for recovery in case of fail over. Once the target write is complete the log entry is cleared. The cost of this technique is one full surface of capacity and (normally) the latency of a head switch for the ''fast write''.

3.5.3.3 *Vail Cache*

Figure 2-13 shows the design of the Vail cache. The cache is composed of two separate memories: the buffer memory and the CMM. All data flows in and out of buffer memory, between the MOCHAs and the CI port. Only write data flows into CMM memory; however, subsequent reads to blocks already in the CMM are satisfied from the CMM memory. Since only one path between the HDAs and the cache memories exist, writeback or write-behind cache is fairly straight forward. Just as criticality of data memory in the HSC can dictate the need for cache memory to support ''fast write'', intensity of Vail buffer memory can dictate a ''4-step method'' of write caching. The 4-step method frees buffer memory while queueing or a seek transpires. The additional movement to refill the data from CMM memory may be well worthwhile.

3.6 Conclusions

The introduction of multiple CI support significantly improves Digital's competitive position for both capacity and number of devices.

Although DLSE greatly reduces RPS miss overhead, HSC-based VAXclusters suffer less RPS delay than DLSE for workloads over 600 requests/second. Since typical workloads for each subsystem are less than 400 request/second, DLSE in fact incurs less RPS delay than HSC-based subsystems. The weakness in the HSC controller is the single path through the K.sdi. The bandwidth of the Vail SEBB virtually eliminates RPS.

When cumulative delays are considered, the HSC server has about the same asymptote as DLSE. DLSE has less delay in the region exhibited by most workloads. The bandwidth of the SEBB allows Vail to perform very similarly to DLSE below 400 requests/second and outperform it above that point. The increased CI port bandwidth and additional MSCP interpreter processors enable Vail to surpass DLSE's asymptote.

The VAX architecture has an advantage over the IBM systems' ability to process simultaneous I/O termination interrupts. However, each processor's actual overhead must be considered, as well as the software handling in making valid comparisons.

IBM's I/O architecture has addressed the routing problems of multiple point-to-point paths from a single system. They have offered a shared controller to implement a loose coupling solution. Digital has addressed loose coupling by VAXclusters, which implement global lock management and a communications architecture. We have not yet addressed the problem imposed by a monolithic (or tightly coupled) system that produces a demand that exceeds the capacity of a single channel.

The multilevel cache strategy of VCC and HSC cache will finally fill a gap. Although the HSC "fast write" incurs a head switch time over the ideal nonvolatile RAM implementation, it answers a competitive need.

Host memory for Global Sections can compete with Expanded Storage to a degree; however, using VAX memory cannot compare with the size of a 16 terabyte expanded storage for reducing page fault waits. IBM can load images or data into expanded storage and service page faults (from expanded storage) faster than the required processing to perform a context switch to another process.

A limitation with VMS SMP has come to light. It appears that processor affinity exists for the VMS fork dispatcher. The net effect is that all I/O completion processing occurs on a single CPU within a tightly-coupled system, making VMS a hybrid SMP with a "control processor". It would be unreasonable to expect a truly pure SMP. There are points where pure SMP can actually degrade performance over a compromise AP system, but this is NOT such a case. An I/O intensive workload can easily create a bottleneck in the "control processor" with three or more CPUs. IBM's MVS uses "prefixing" to avoid the problem. We have a competitive exposure.

3.6.1 *Digital I/O Competitive Weaknesses*

Although Digital has some advantages in the I/O subsystem, as VAX9000 and future systems deemphasize VAXclusters, there are both short term and long term weaknesses as compared to IBM's I/O subsystem. The short term weaknesses have mostly been addressed:

1. The lack of caching products.
2. Limitations in the single host CI port as opposed to multiple IBM channels for I/O extensibility.
3. Processor affinity of the VMS fork dispatcher.

The numerous caching products in development will address the first issue. The introduction of the VAX9000 has addressed the second issue. The third issue is a VMS issue.

There are some long term weaknesses that are not currently being addressed:

1. The requirement for higher requests rates for same amount of data transfer.

2. No long term answer to expanded storage.

The introduction of caching products should bring the the request rates down toward that of IBM's subsystems; however, higher request rates are inherent in Digital workloads because of smaller request sizes. The introduction of expanded storage has further reduced the requests rates to the IBM subsystem. Although utilizing VAX main memory (via global sections) effects a similar reduction in I/O requests, the 16 terabyte limit of expanded storage offers solutions not available through the use of global sections.

APPENDIX A

RPS QUANTIFICATION

The following graphs demonstrate the architectural delays for the key products of IBM and Digital.

Figure A–1: RPS Miss

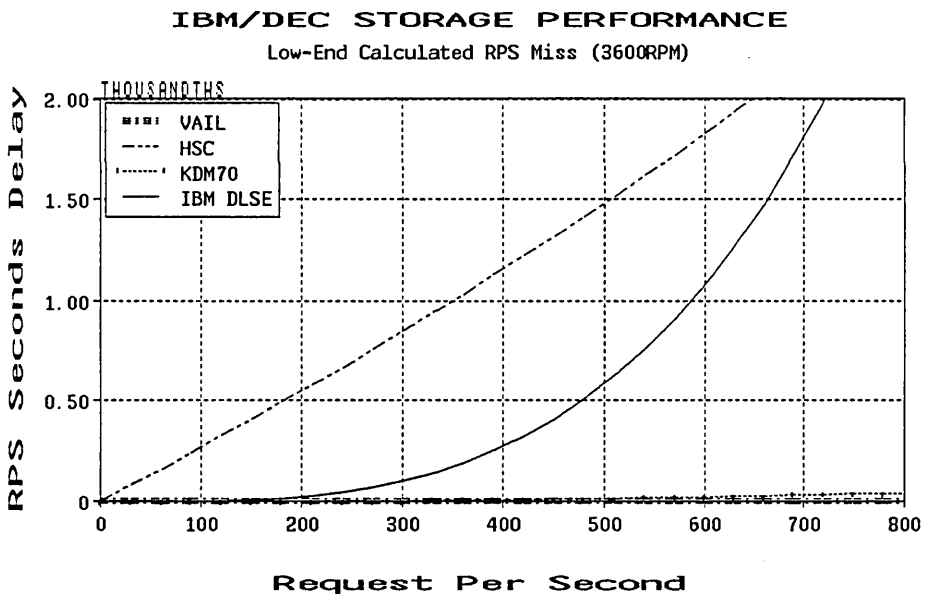


Figure A–1 Cont'd on next page

Figure A-1 (Cont.): RPS Miss

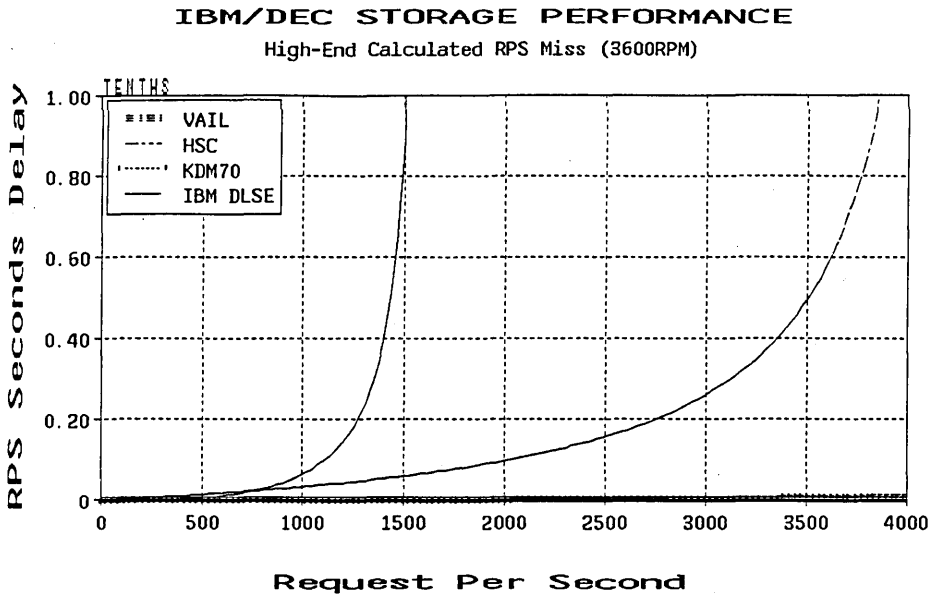


Figure A-1 Cont'd on next page

Figure A-1 (Cont.): RPS Miss

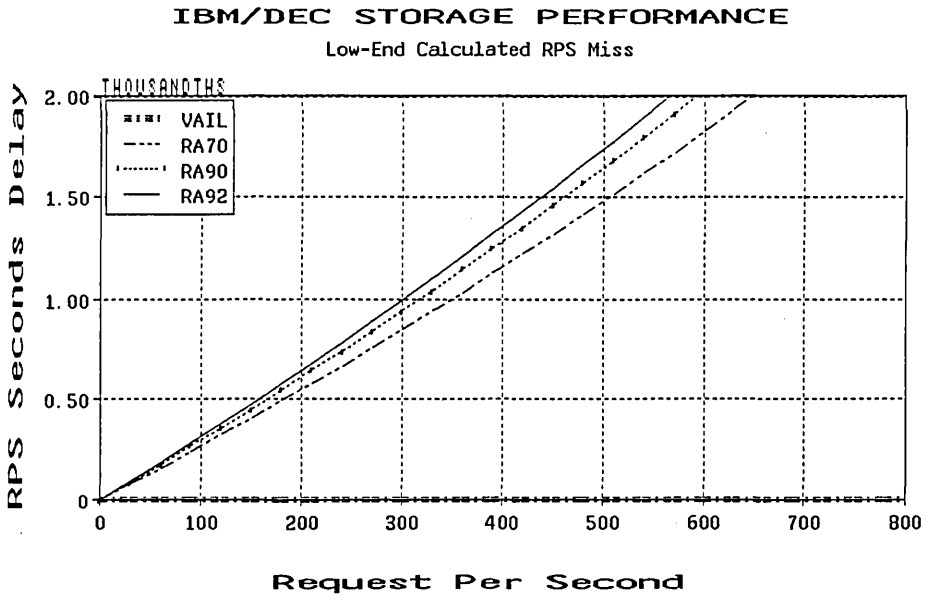


Figure A-1 Cont'd on next page

Figure A-1 (Cont.): RPS Miss

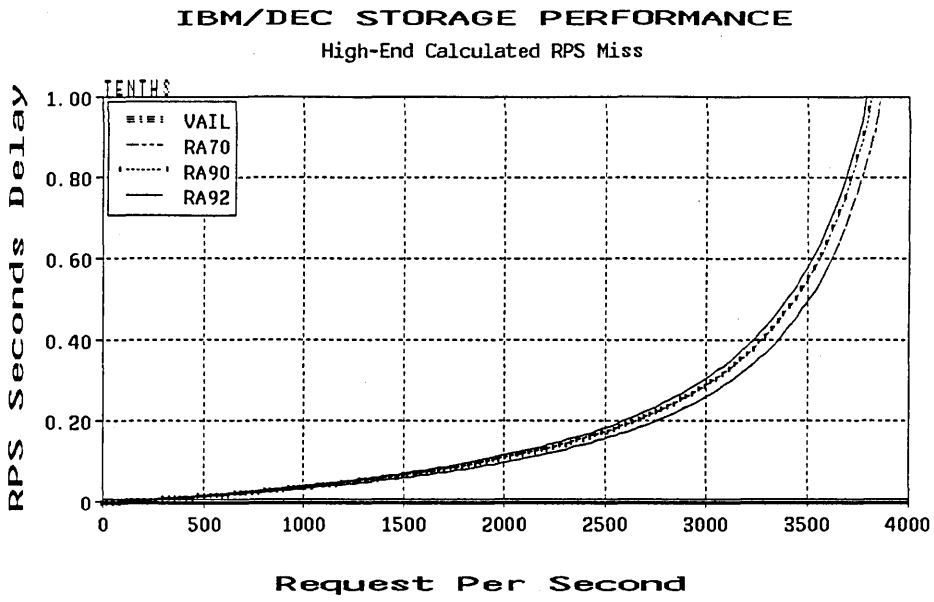


Figure A-1 Cont'd on next page

Figure A-1 (Cont.): RPS Miss

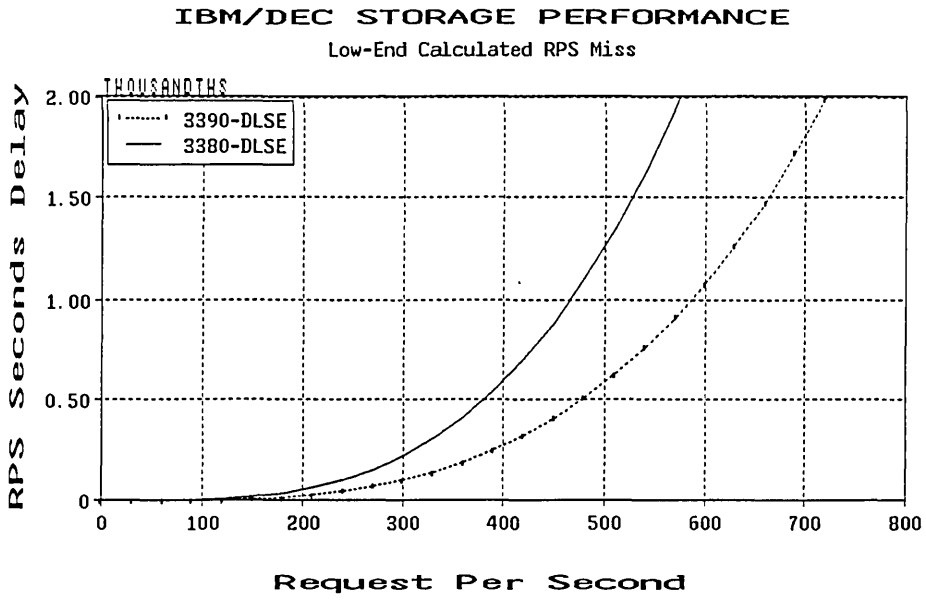
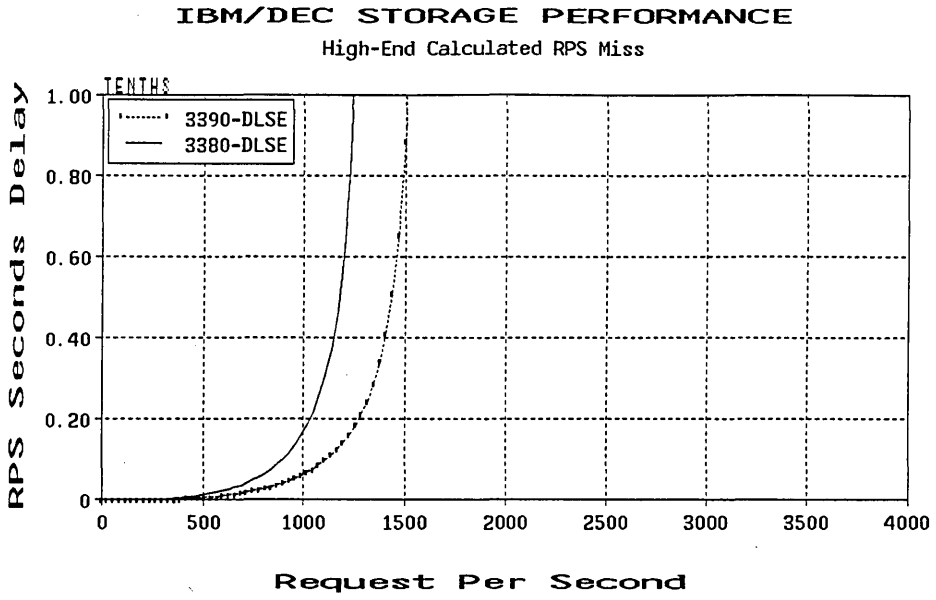


Figure A-1 Cont'd on next page

Figure A-1 (Cont.): RPS Miss



APPENDIX B

CUMULATIVE DELAY QUANTIFICATION

The following graphs display the overall architectural delays due to RPS miss and additional queueing.

Figure B-1: Total Path Delays

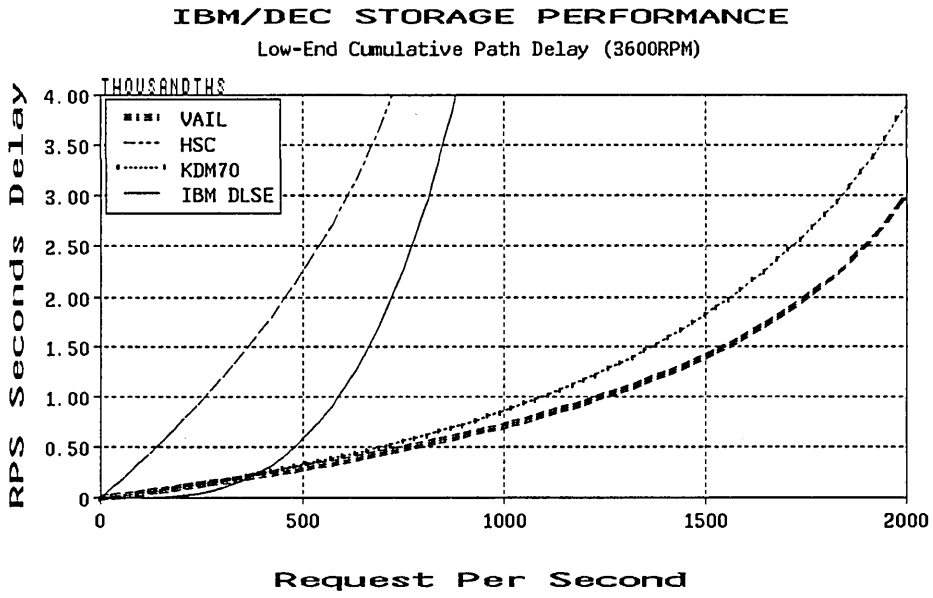
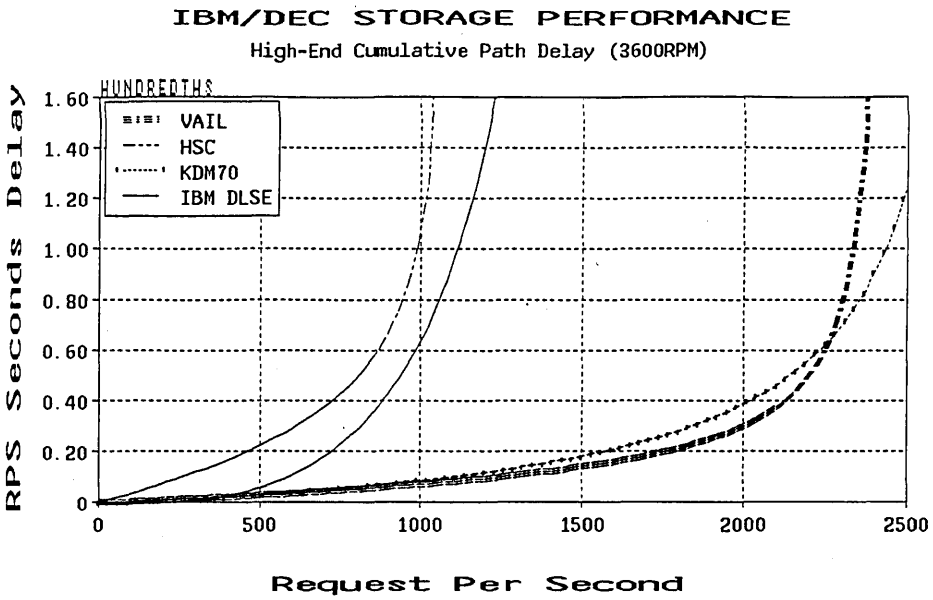


Figure B-1 Cont'd on next page

Figure B-1 (Cont.): Total Path Delays



APPENDIX C

SYSTEM TOPOLOGY

The two companies have designed their respective I/O subsystems for different environments. IBM has used an open-ended I/O architecture capable of supporting an indeterminantly powerful system. Digital has designed an I/O subsystem to satisfy requests from multiple systems of bounded I/O appetite. IBM's environment has grown out of an architectural philosophy which focuses on many servers that reduce, if not eliminate, queuing.

C.1 Monolithic Processing

Monolithic processing refers to a system with a single CPU, memory, and I/O subsystem. This is the type of system from which IBM has derived its current architecture.

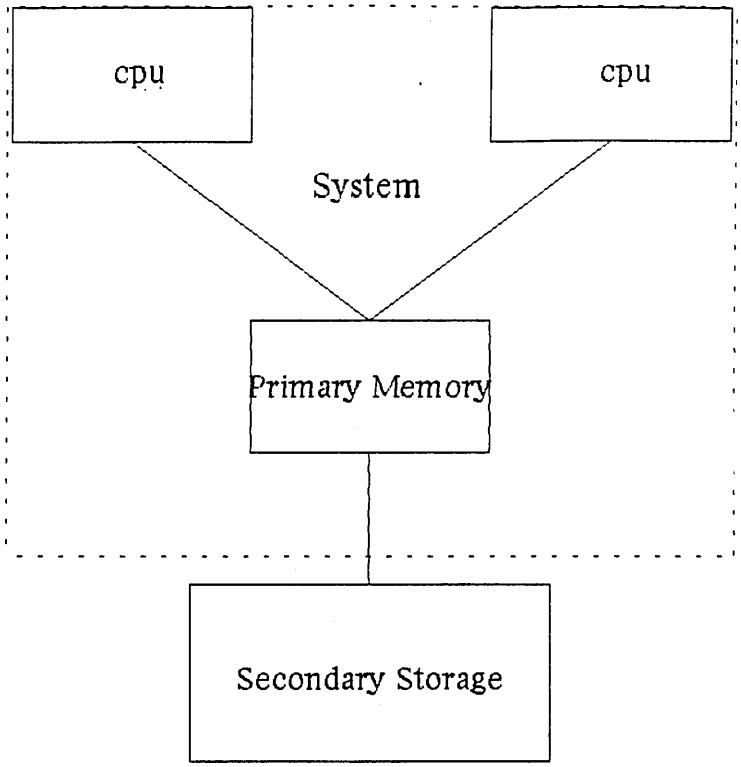
C.2 Coupling

Processor coupling refers to memory sharing. Tight coupling means that multiple processors share primary memory. Loose coupling implies that the processors share secondary storage. In the abstract, loose coupling is different from a network because a network is a collection of systems that share the communications medium rather than any real storage medium.

C.2.1 Tight Coupling

The term tight coupling refers to the sharing of main memory by multiple CPUs.

Figure C-1: Tight Coupling



C.2.1.1 Attached Multiprocessing

Asymmetric multiprocessing is the simplest step from monolithic processing. In this configuration a second processor can only execute enough kernel mode code to dispatch to a user program. All interrupts, exceptions, memory management, and scheduling are handled by a control processor. Scheduling functions include job control, processor dispatching, and I/O initiation. The restriction may be physical or software only. Typical workloads quickly saturate the control processor when more than one attached processor is configured.

C.2.1.2 Symmetric Multiprocessing

Symmetric multiprocessing refers to a tightly coupling where all CPUs can perform all functions. To serialize critical sections, the operating system software must use memory variables (locks) that are shared among the participating processors.

IBM configurations that have channels attached to the CPUs are somewhat restrictive in that requests can only be sent down a channel from the appropriate CPU.

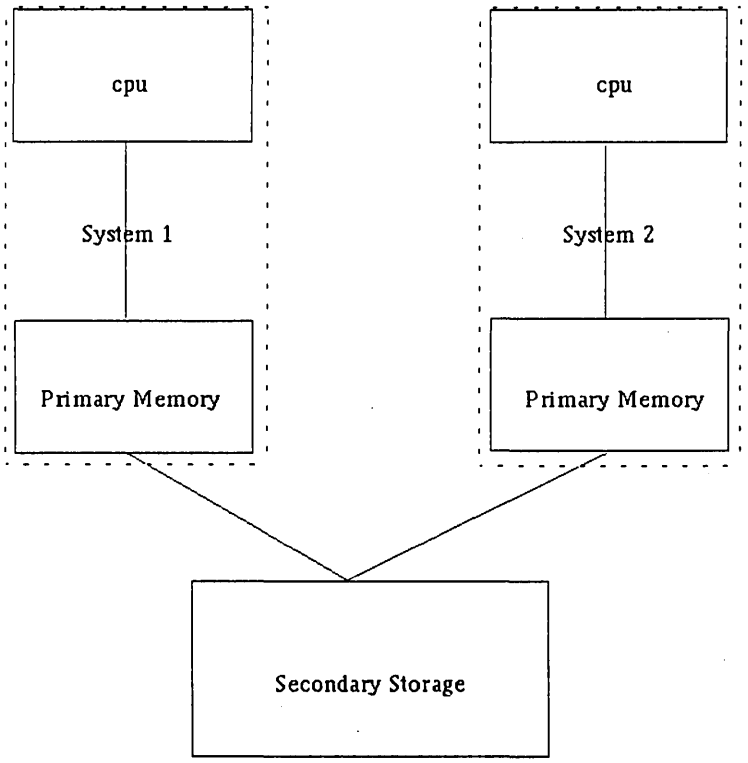
The VMS version 5 operating system is the first symmetric multiprocessor support. To convert to symmetric multiprocessing (SMP) the IPL synchronization had to be changed to use shared memory variables (Mutexes). Rather than completely redesign the operating system, the Mutexes were implemented to work with IPLs.

C.2.2 Loose Coupling

Loose coupling is the sharing of secondary storage among a number of systems. An interlock technique similar to the shared memory variables used by symmetric multiprocessing must be used to avoid data corruption and deadlock. There are two basic topologies to effect loose coupling. The sharing systems must obey the necessary rules of utilizing interlocks to prevent deadlocks and data corruption.

The simpler topology employs a single external point among the participating systems that performs the primitive interlock functions upon request. IBM shared DASD is example of this topology where a shared controller is used. Software protection against deadlock is incomplete and requires operations management of applications.

Figure C-2: Loose Coupling



VAXclusters represent the other form of loose coupling where there is adequate software support to prevent deadlocks and data corruption. VAXclusters employ a "partitioned" approach; each system in the VAXcluster performs the primitive interlock functions. The VAXcluster software relies on the the VMS Lock Manager to provide the primitive interlock functions. The Lock Manager itself is more robust than most designs in that it is a general resource function as opposed to a file system-specific function.

Bibliography

Digital SOURCES

[ARCH71] Bell, Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, 1971

[CI81] Thompson, Buzynki, Huthcinson, *Computer Interconnect Specification*, Digital internal document

[DSDF] Parenti, Rubinson, *SDI Implementation of the Digital Standard Disk Format*, Digital internal document

[DSSI] Wrenn, Zayas, *Digital Storage Systems Interconnect (DSSI) Overview*, Digital internal document

[MSCP] Gardner, *Mass Storage Control Protocol*, Digital internal document

[NETW76] A.S. Tanenbaum, *Computer Networks*, Prentice Hall, 1976

[SCACI] Strecker, *Systems Communications Architecture*, Digital internal document

[SCANI] Eldridge, *Systems Communications Architecture NI Port Spec*, Digital internal document

[SDI] *Digital Standard Disk Interface (SDI) Specification*, Digital internal document

IBM SOURCES

[3601] *IBM System/360 Componenet Summary: 3830 Storage Control and 3380 Disk Storage*, GA26-1592, Internation Business Machines Corporation, Data Processing Division, White Plains, NY.

[3602] *IBM System/360 Componenet Summary: 2835 Storage Control and 2305 Fixed Head Storage Module*, GA26-3599, International Business Machines Corporation, Data Processing Division, White Plains, NY.

[360C] J. S. Liptay, "Structural Aspects of the System/360 Model 85; Part II-The Cache," *IBM Systems Journal 7*, 15-21 (1968).

[3880] *IBM 3880 Storage Control Models 1, 2, 3, and 4*, GA26-1661, IBM Corporation, General Products Division, Tucson, Az., 1987.

[3990] *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide*, GA32-0100, IBM Corporation, General Products Division, Tucson, Az., 1987.

[AD01] A. Padegs, "The Structure of System/360; Part IV-Channel Design Considerations," *IBM System Journal 3*, 165-180, 1964.

[CJC1] C. J. Conti, D. H. Gibson and S. H. Pitkowshy, "Structural Aspects of the System/360 Model 85; Part I-General Organization," *IBM Systems Journal 7*, 2-14 (1968).

[DTB1] D. T. Brown, R. L. Eibsen and C. A. Thorn, "Channel and Direct Access Device Architecture," *IBM System Journal 11*, 186-199 (1972).

[GMA1] G. M. Amdahl, G. A. Blaauw and F. P. Brooks, "Architecture of the IBM System/360," *IBM Journal of Research and Development 8*, 87-101 (1964).

[HAM1] F. E. Hamilton and E. C. Kubie, "The IBM Magnetic Drum Calculator Type 650," *Journal of the ACM, 1*, 13-20 (1954).

[IBM1] C. J. Bashe, W. Buchholz, G. V. Hawkins, J. J. Ingram and N. Rochester, "The Architecture of IBM's Early Computers," *IBM Journal of Research and Development 25*, 363-375, 1981]

[KLEI] L. Kleinrock, *Queueing Systems. Volume I: Theory*, Wiley, 1975. *Queueing Systems. Volume II: Computer Applications*, Wiley, 1975.

[POp1] *IBM System/360 Principles of Operation*,GA22-6821, Internation Business Machines Corporation, Data Processing Division, White Plains, NY.

[POp2] *IBM System/370 Principles of Operation*, GA22-7000, IBM Corporation, Poughkeepsie, NY., 1987.

[POp3] *IBM 370-XA Principles of Operation*, SA22-7085, IBM Corporation, Poughkeepsie, NY.

[POp4] *IBM Enterprise Systems Architecture/370 Principles of Operation*, SA22-7200, IBM Corporation, Poughkeepsie, NY.

[POp5] *IBM System/360 and System/370 I/O Interface: Channel to Control Unit, Original Equipment Manufacturer's Information*, GA22-6974, IBM Corporation, Poughkeepsie, NY.

[RNG1] R. N. Gustafson and F. J. Sparacio, "IBM 3081 Processor Unit: Design Consideration and Design Process," *IBM Journal of Research and Development 26*, 12-21 (1982).

[Wil1] M. V. Wilkes, "The Best Way to Design an Automatic Calculating Machine",*Manchester University Computer Inaugural Conference*, Manchester, England, 1951, p.16.

[WYS1] W. Y. Stevens, "The Structure of System/360; Part II-System Implementations," *IBM Systems Journal 3*, 136-142 (1964).

GLOSSARY

Adapter: A device that converts from one bus to another. A distinction between an an adapter and I/O processor is that an adapter is restricted to addressing "registers" in a dedicated area within main memory; I/O programs for an I/O processor may exist anywhere in primary memory.

AMS: A set of utilities for accessing data; Access Method Services

AP: Attached multiProcessing - multiple processors sharing main memory where certain processors are limited in capabilities

AST: Asynchronous Service Trap - a processor dispatch technique that allows a process to handle an asynchronous event with a separate context from the main execution thread

ASTLVL: Asynchronous Service Trap LeVel - an internal VAX register that works in conjunction with the REI instruction microcode to scan for deliverable ASTs based upon processor mode

BDAM: Basic Direct Access Method - a component of AMS

BPAM: Basic Partitioned Access Method - a component of AMS

BSAM: Basic Sequential Access Method - a component of AMS

Central Storage: The main memory and extended storage interface control connected to an IBM 3090 processor

Channel: A device which communicates directly with I/O devices and manages the flow of information between I/O devices and main storage

CKD: Count, Key, Data - IBM formatting method used on 3330, 3350, and 3380 disk drives for record location

Control Unit: Controls the timing of data transfer; adapts the characteristics of I/O devices to a standard form; accepts control signals from the channel and provides indications concerning the status of I/O devices

CRC: Cyclic redundancy check bytes

DASD: Direct Access Storage Device, i.e. a disk drive

Datagram: A communications package that requires the upper layers to ensure delivery and flow control

DSA: Digital Storage Architecture - Digital's current I/O architecture

ECC: Error correction code

ESDS: Data organization scheme under VSAM - Entry Sequenced dataset

FBA: Fixed Block architecture - specifically IBM's implementation, for this paper, such as the 3370

Interface: The communication between two different levels in a hierarchy

I/O Processor: A programmable processor that handles I/O "programs" in a similar way to a CPU. A distinction between an I/O processor and an adapter is that the I/O programs may exist anywhere in primary memory; an adapter is restricted to addressing "registers" in a dedicated area within main memory.

IOS: A component of IBM's operating systems, the I/O Supervisor

IPL: Interrupt Priority Level - hardware enforced processor priority that establishes a full ordering

ISO: International Standards Organization

ISR: Interrupt Service Routine - the first software to execute upon processor interruption. It saves current CPU context and sets up subsequent interrupt handling.

KSDS: VSAM data organization Key sequenced dataset

Main Memory: The main memory used by the central processor

MASSBUS: Mass Storage Bus - did not replace UNIBUS for general peripherals

MSCP: Mass Storage Communications Protocol - the master/slave protocol between host and controller in the Digital I/O architecture.

Mutex: Mutual exclusion variable - a means to control synchronization for data access to prevent corruption

OSI: Open Systems Interconnect

PMS: A sketch oriented technique to represent a system's high level components

Primary Memory: The main memory used by the central processor

PSL: Processor Status Longword - VAX register that contains the processor mode and privilege status information

Protocol: The communication between peer layers. In communications this defines the data formats and their interpretation.

QSAM: Queued Sequential Access Method

Queue Length: The average number of customers/transactions at the service center both waiting and receiving service

RAM: Random Access Memory - solid state chips, usually MOS

REI: Return from Exception or Interrupt - A VAX instruction that returns control to the appropriate stack and IPL after processing either an exception or an interrupt

Residence Time: The average time spent at the service center, by a customer/transaction, both queuing and receiving service

RPS: Rotational position sensing

SCA: System Communications Architecture - Digital's host to storage subsystem architecture

Service Demand: The average service requirement (a quantity) of a customer/transaction

SIOF: Start I/O Fast Release - Instruction in the 370 set which initiates an I/O and does not wait for an acknowledgment, i.e. asynchronous I/O

SIO Rate/Sec: Number of Start I/O's issued to a device over the duration of a single second

SMP: Symmetric MultiProcessing - multiple processors sharing main memory where all processor are equal in capability

Sub-Channel: Channel facilities required for sustaining a single I/O operation

SxI: Generically used to stand for either Digital's SDI for disk or STI for tape

Throughput: The rate at which transactions pass thru the service center

UNIBUS: PDP11 positional priority bus for peripheral connections

Utilization: Portion of time the server is busy

VC: Virtual Circuit - a communications abstractions that provides guaranteed delivery and flow control to higher layers (Digital uses a slightly different meaning)

VCC: VAXcluster Cache - host-based software writethrough cache in VMS

VMS: Virtual Memory System - the main operating system for VAX

VOLSER: Acronym for VOLume SERial number - a user defined label of reference for magnetic storage media

VSAM: Virtual Storage Access Method

VTOC: Volume table of contents a reserved file containing information about datasets allocated on a specific disk

Workload Intensity: Rate at which customers (serviceable interactions) arrive at a service center (i.e. a disk)