HONEYWELL BILLERICA	SPECIFICATION 60149952	REVISION DRAFT
SMALL COMPUTER PRODUCTS ENGINEERING	ENGINEERING PRODUCT SPECIFICATION PART 1 MEMORY QUEUED I/O PROTOCOL	
PREPARED BY M. Raisbeck		
DATE November 6, 1986		

Revision	Authority	Date	Signature	Sheets Affected
Draft	BLCDG7623	11/6/86	M. Raisbeck	All

Table of Contents

TBD

I

INTRODUCTION

1.1 Identification and Purpose of this Specification

This document is the Engineering Product Specification, Part 1, for a new memory queued i/o protocol (MQIO). It sets forth the characteristics and requirements of the memory queued i/o protocol and presents examples of its advantages and its use.

1.2 Scope

The purpose of the memory queued i/o protocol is to serve as a performant, efficient, reliable, and resilient method of i/o and interprocessor communication. This new protocol is designed for use on all new i/o processors, including those for magnetic peripherals, communications, cluster controllers, and others, which will be developed for use on the Honeywell Megabus, on variants of the Megabus, or on future system busses.

1.3 Applicable Documents

A list of reference documentation is provided below in Appendix I.

1.4 Definitions

A list of acronyms and definitions can be found in Appendix II.

II

ARCHITECTURAL OVERVIEW

2.1 I/O Dialogue

Under the memory queued i/o protocol, communications between processors function primarily through memory resident control and data blocks. At system startup, the initializing processor (typically a CPU running system software) sends a block of data called the Initialization Control Block to the target processor. Contained in the Initialization Control Block are pointers to the key i/o data structures. This block is stored in the controller.

During normal system operation, the initiating processor starts an i/o operation by constructing one or more blocks of data in memory. The first of these blocks is called a Request Header, and contains i/o control information. It also contains a pointer to a queue of subsidiary blocks of data called Request Bodies. These Request Bodies describe the particular functions that the target processor (typically an i/o controller) is to perform, and also contain pointers to data buffers. When the Request Header has been built and the necessary Request Bodies have been built and attached to it, the requesting processor attaches the Request Header to the target processor's Request Queue. Figure 1A describes the situation at the start of a simple i/o operation.

At this point, it is the target processor's turn to act. It scans its Request Queue looking for requests. When it discovers the new Request Header, it removes it from the Request Queue and puts it on the Action Queue for safekeeping. Starting with the first Request Body, the processor then works its way sequentially through the queue of Request Bodies performing the requested functions. Figure 1B illustrates the data structure

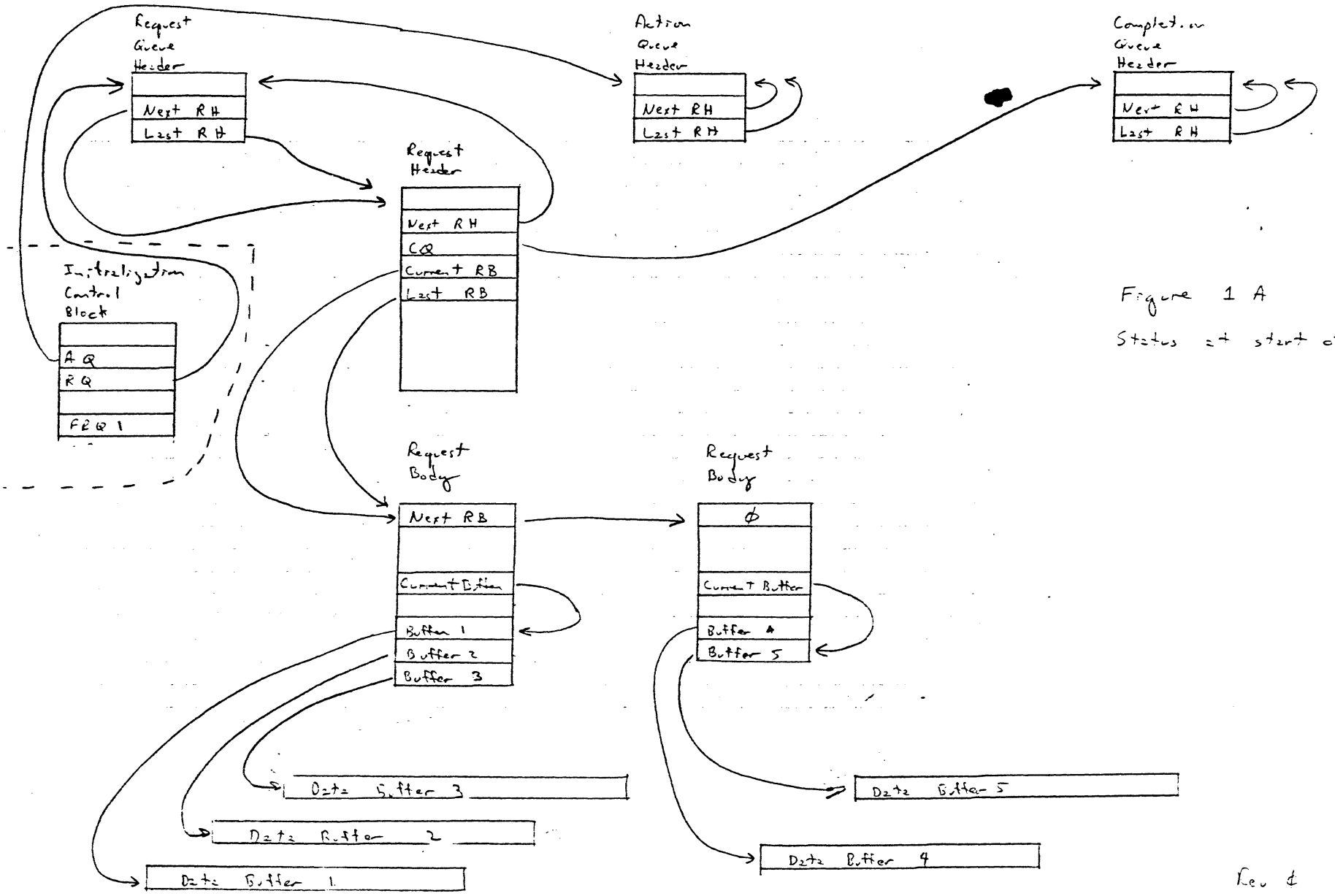


Figure 1 A

Status at start of I/O

Rev 4
10/31/85

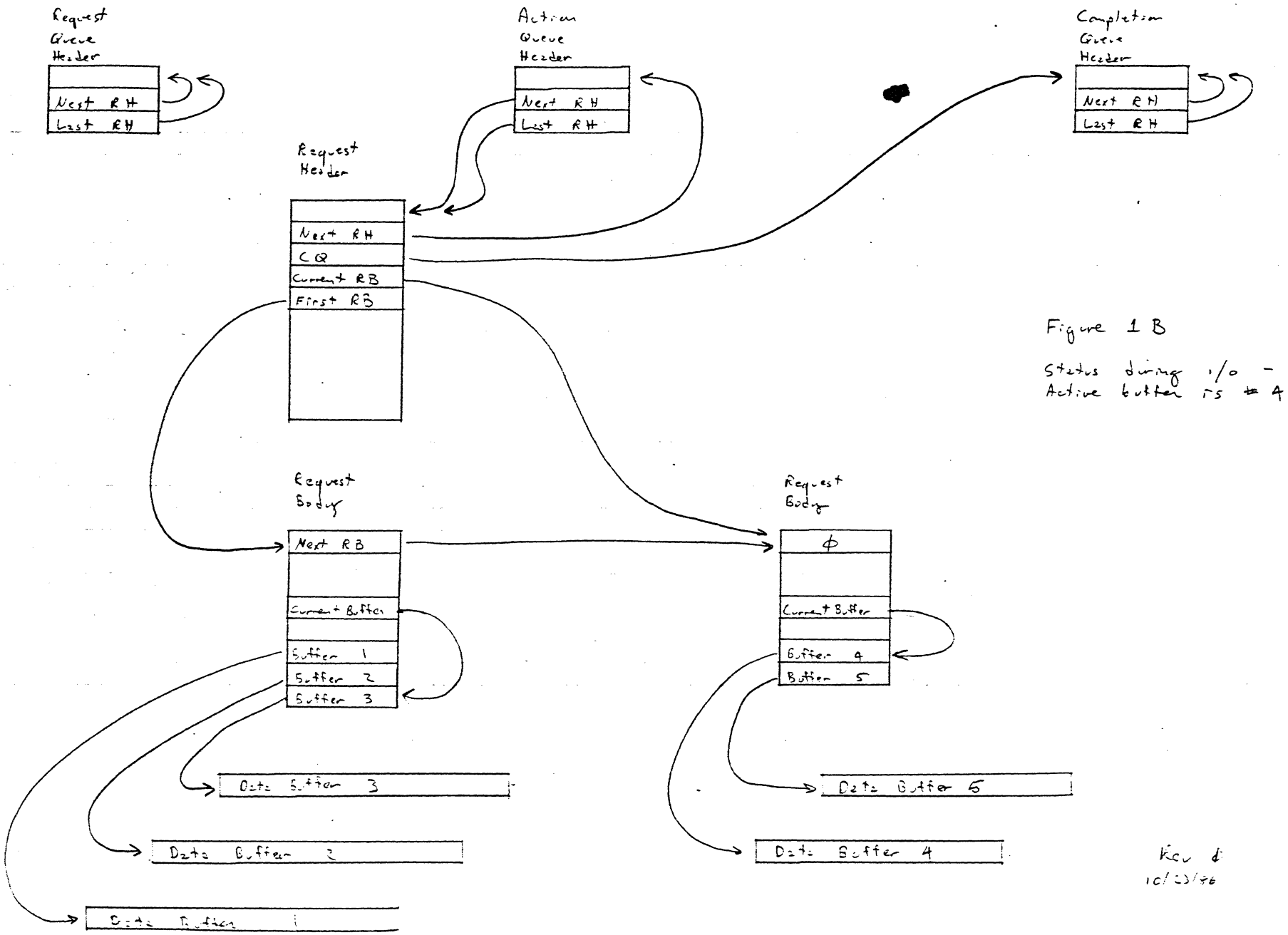


Figure 1 B

Status during I/O -
Active buffers = 4

Rev d
10/23/86

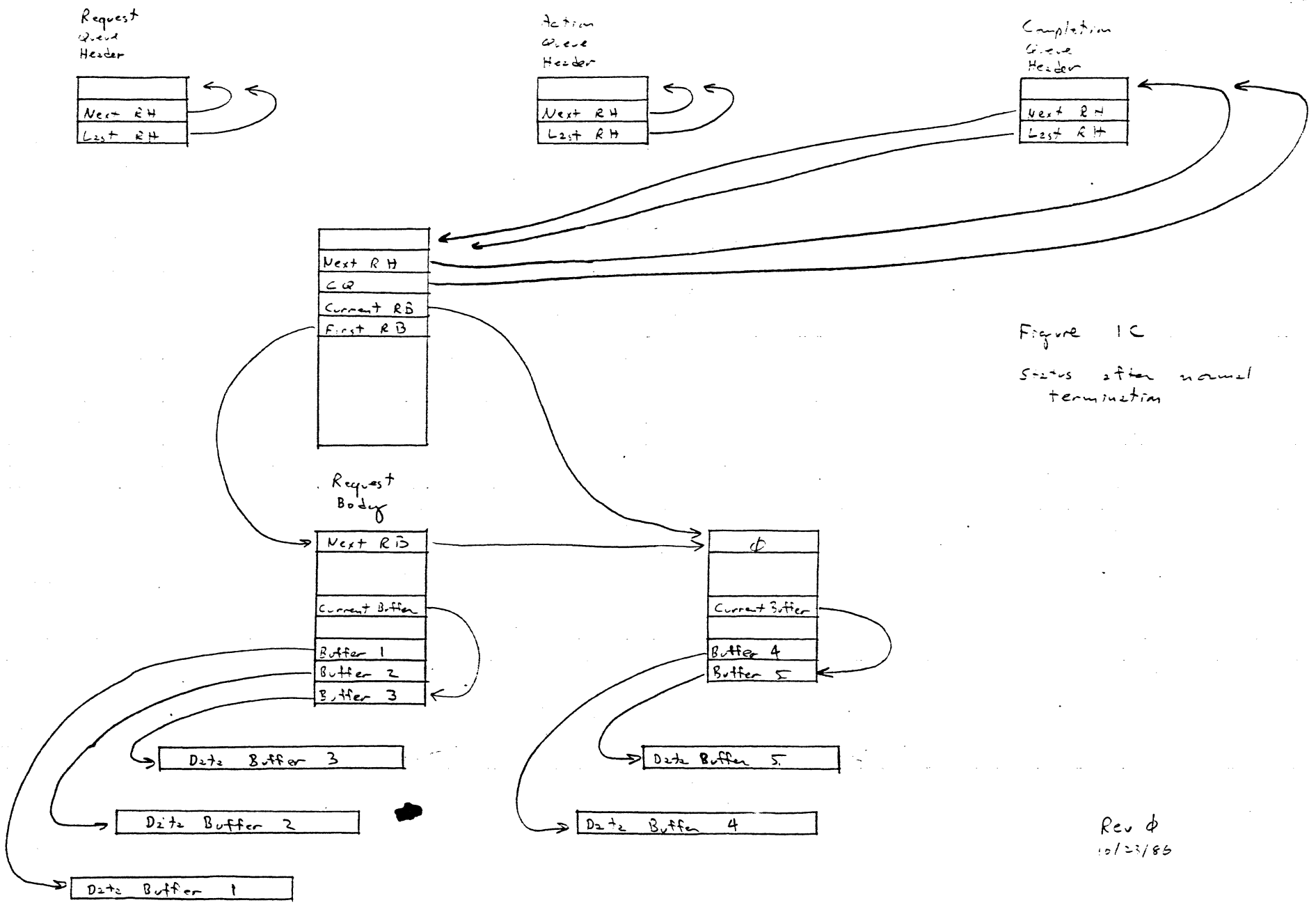


Figure 1C

Status after normal termination

Rev ϕ
10/23/85

relationships while i/o is in progress.

When all functions are complete or when an abnormal termination event occurs, the target processor sets status information in the Request Header, removes it from the Action Queue, and attaches it to the Completion Queue. The initiating processor finds the Request Header on the Completion Queue. Figure 1C illustrates this situation. Removing the Request Header from the Completion Queue, the initiating processor analyzes the information, thus completing the transaction.

In the case of unsolicited events, such as device attention interrupts, there is a fourth group of data structures called the Free Request Queues. These contain ready to use Request Headers, complete with linked Request Bodies. The signalling processor dequeues a Request Header (the choice of Free Request Queue to pick it from is discussed later), puts it on the Action Queue, fills in a few appropriate fields, places the received data, if any, into the data buffers, and attaches the whole thing to the Completion Queue.

2.2 Memory Address Handling

A second important feature of the memory queued i/o protocol lies in the handling of memory addresses. Since this protocol is designed to be used in a virtual memory environment, it will often be convenient to pass virtual rather than physical addresses to the target processor. In order to do this, virtual-to-physical address mapping information must be made available to the target. This is accomplished by placing copies of several segment descriptors in each Request Header. It becomes the target processors responsibility to handle the access checking and memory mapping necessary to resolve virtual addresses.

2.3 Comparison with Direct Interface I/O

The memory queued i/o protocol resolves several efficiency, performance, and integrity problems that have developed as old direct interface i/o (DIIO) was applied to large and multiprocessor systems. With direct interface i/o, CPU-to-controller control communication was direct rather than through memory. If a controller could not keep up with the data from the CPU, it had to reject (NAK) the transfer, which forced the driver software to reissue the command, or hold up the CPU (WAIT), momentarily stopping the entire bus. If two or more CPUs attempted to reach the controller simultaneously, the service

delays got bad enough on some configurations that the system's hang protection timers would expire, causing traps and other erratic system behaviors and requiring the use of configuration constraints and driver retry logic in operating system software. These momentary overload problems are eliminated with the memory queued i/o. The target processor can process requests at its own pace without interfering with the performance of other parts of the system, and in particular without WAITing or stalling the initiating processor.

A second significant change is the potential elimination of CPU context swapping caused by the arrival of device interrupts. With direct interface i/o, task completion was signalled by interrupting the CPU. This either caused a context swap in the operating system, or, if the interrupt were rejected, caused the controller to reject further commands and retry the interrupt until such time as it was accepted. While still permitting the interrupt mechanism to be used, the new i/o also allows for the completion of tasks without requiring immediate attention from the operating system.

Finally, the memory queued i/o permits command chaining. This, combined with the move of many address calculation chores from the CPU to the controller, eliminates the need for a dedicated driver process for each device, reduces the need to move data back and forth between system buffers and user buffers, and facilitates the sharing of i/o resources by dissimilar coprocessors and operating systems.

III

FUNCTIONAL REQUIREMENTS

3.1 General

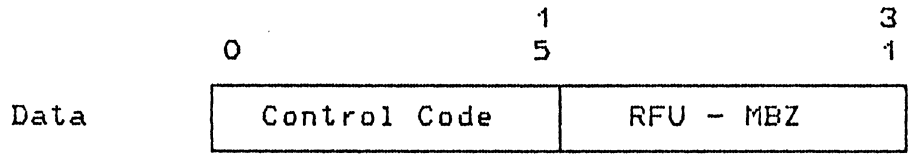
3.1.1 Direct Interprocessor I/O

Processors supporting the memory queued i/o protocol retain support for a small set of direct interprocessor i/o commands. These serve primarily initialization, identification, and diagnostic functions. The detailed descriptions of these commands as presented below presuppose a Megabus attachment. It is assumed, however, that they could be implemented under any bus architecture that may eventually be chosen, while still retaining the same software appearance.

A processor is permitted to make one of two responses to a direct i/o command. It may either accept (ACK) the command or reject (NAK) it. Waits are not permitted, despite the fact that they are allowed by the Megabus architecture. For the Output Control function, there is the further limitation that control codes #8000 (initialize) and #4000 (stop i/o) may not be NAK'ed.

3.1.1.1 Output Control Word - Function Code #01

	A	H 0	7 8	1 1 7 8	2 3
Address	MBZ	MBZ	Target Channel		000001



3.1.1.1.1 Control Code #8000 - Initialize

This command causes the target processor to return to the same state that it enters after system power on except that soft firmware memory, if present, is not cleared. Any i/o in process is terminated abruptly and without notification to the target processor. All other fields and registers in the target processor are returned to their power on state, which is described in detail in Section 3.1.2 below. QLT's, if present, are executed. The initialize affects not only the particular channel addressed, but also all other channels that reside on the same processor. Device resets and/or recalibrates are issued where appropriate. This command must always be accepted by the target processor. It may not be NAK'ed.

3.1.1.1.2 Control Code #4000 - Stop i/o

This command causes the target processor to terminate any i/o in progress. The current Request Header will be updated to reflect the status of the in-process i/o, and the Request Header will then be enqueued to the Completion Queue. If there is no i/o currently in process, the Stop I/O will be treated as attention interrupt, causing the processor to remove a Request Header from Free Request Queue 0, fill in the appropriate fields, and enqueue it on the Completion Queue. This command must always be accepted by the target processor. It may not be NAK'ed.

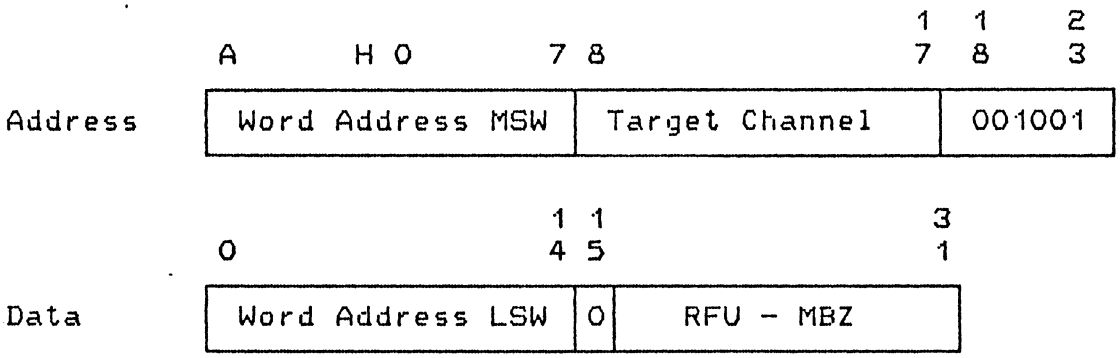
3.1.1.1.3 Control Code #0100 - Start i/o

This control code signals the target processor to search its Request Queue and start processing the highest priority (ie., first) Request Block. If no Request Block is found on the Request Queue, or if there is a Request Block from this Request Queue currently being processed, this command is ignored. If no valid Initialization Control Block has been made known to the target processor, bit 15 of the Function Status Word is set.

3.1.1.1.4 Control Codes #2000, #1000, #0800, #0400, #0200

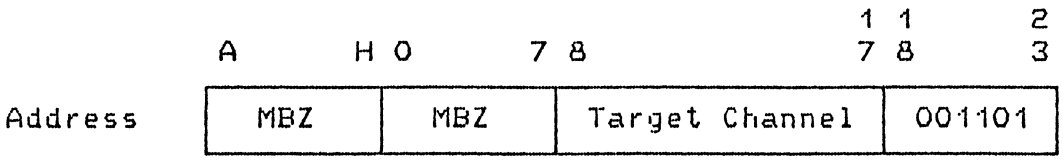
These control codes are supported by processors which have loadable firmware. Details are documented in the System Maintenance Facility EPS-1.

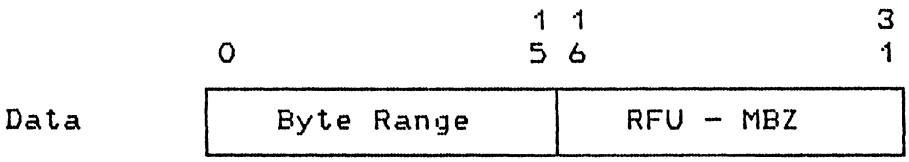
3.1.1.2 Output Initialization Control Address - Function Code #09



This function passes to the target processor the physical memory address of the Initialization Control Block. The function is usually launched by way of the IOLD instruction, which initiates an output range (Function Code #0B) after the address is sent. If bit 15 of the data word (the byte offset) is not zero, bits 13 and 15 of the Header Status Word are set and the processors internal copy of the ICB and the ICB address are reset as described in Section 3.1.2. Otherwise, the Status Words are reset as described in Section 3.1.2. The format of the Initialization Control Block is described in Section 3.1.3.1 below.

3.1.1.3 Output Initialization Control Range - Function Code #0D





This function passes to the target processor the size of the Initialization Control Block. The quantity is an unsigned 16 bit integer in units of bytes. The value is treated as modulo 4 bytes, thus the low order 3 bits of the range are ignored by the target processor. The minimum permissible range is 40 bytes (#28) and the maximum is 1052 bytes (#41C). The output range function is usually launched by means of the IOLD instruction, which outputs an address, followed by a range. Upon receipt of a Function Code #0D, the target processor will commence reading of the Initialization Control Block. Any of the following situations or events will cause the target processor to set bit 15 of the Header Status Word and to reset as described in Section 3.1.2 its private copies of the ICB and the ICB address:

1. the range is less than #28 (40) or greater than #423 (1059)

2. the processor is unable to read the entire Initialization Control Block

3. the Initialization Control Block address is null

4. the Output Queue address or any of the Free Request Queue addresses is null or invalid

Bits 12, 13, 14, and 15 of Status Word 1 will be set as appropriate. Otherwise, Status Words are set as described in Section 3.1.2.

3.1.1.4 Output Firmware Address - Function Code #29

This function pertains only to processors having loadable firmware. Details may be found in System Maintenance Facility EPS-1.

3.1.1.5 Output Firmware Range - Function Code #2D

This function pertains only to processors having

loadable firmware. Details may be found in the System Maintenance Facility EPS-1.

3.1.1.6 Output Bootload Address - Function Code #39

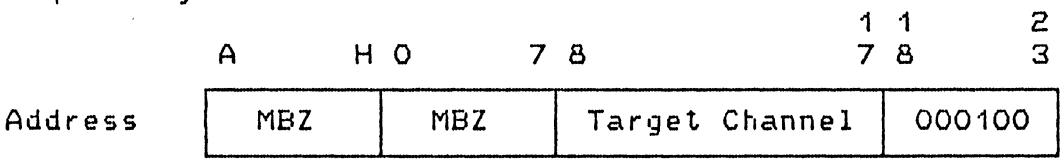
TBD

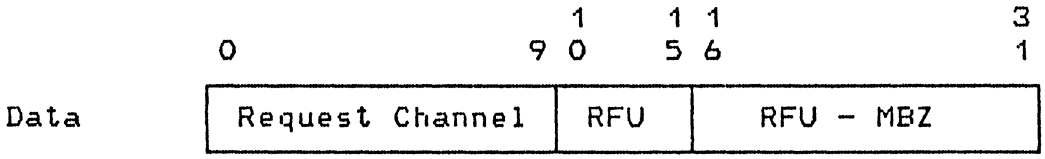
3.1.1.7 Output Bootload Range - Function Code #3D

TBD

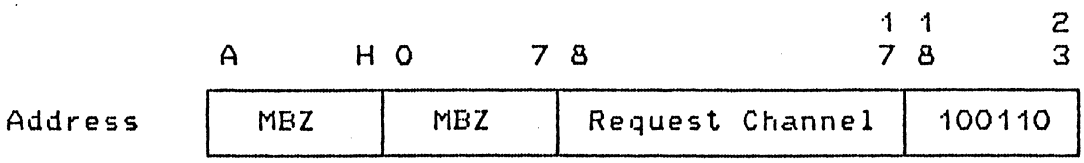
3.1.1.8 Input Revision Status - Function Code #04

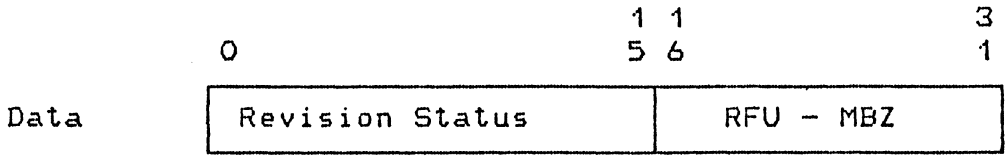
Request Cycle





Response Cycle

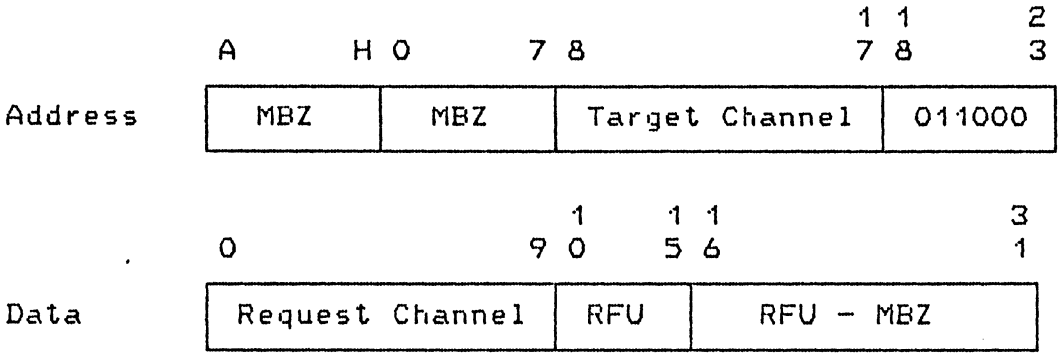




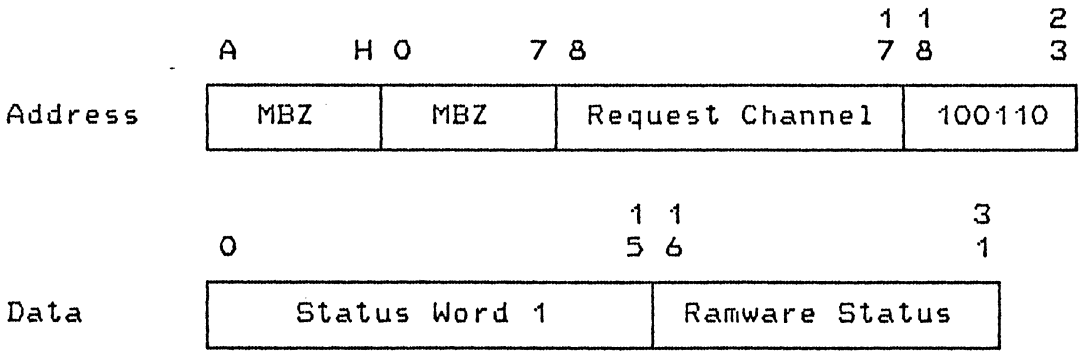
This function returns a 16 bit hardware/firmware revision status. The precise meaning is dependent on the particular controller and device referenced.

3.1.1.9 Input Hardware Status Word 1 - Function Code #18

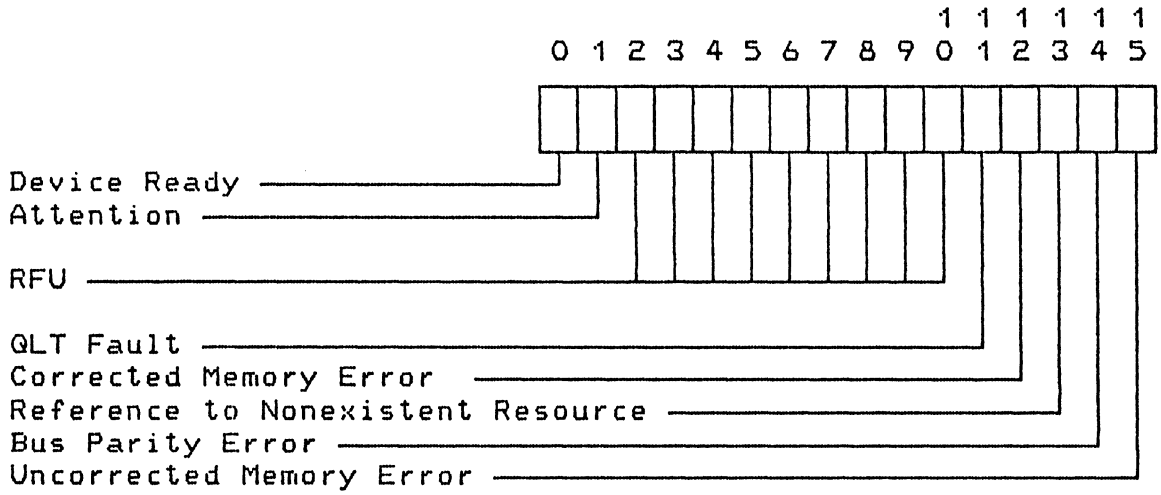
Request Cycle



Response Cycle



This command is included primarily for diagnostic purposes, and would not be used by the operating system under normal circumstances. Status Word 1 is also returned in the Request Body at completion or termination of a function.

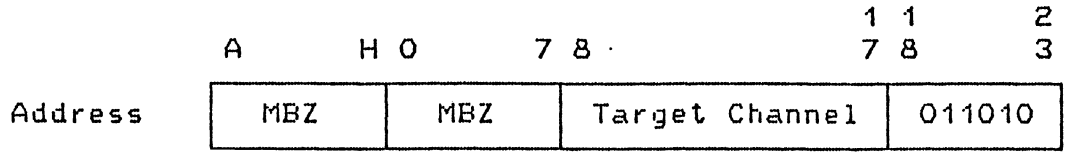


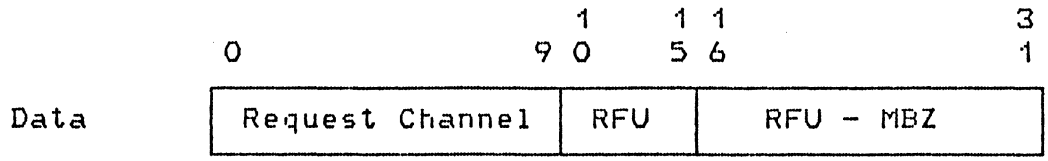
Bits 1 and 12 are reset after Status Word 1 is read, either through this function code or by being placed in a Request Body. Bits 11, 13, 14, and 15 are reset only by the initialize and stop i/o functions of the Output Control command.

For controllers which have loadable firmware, the low order 16 bits of returned data contain the Ramware Status Word. This information is not software visible on current machines, and is used only by the System Maintenance Facility. Details of the Ramware Status may be found in the System Maintenance Facility EPS-1.

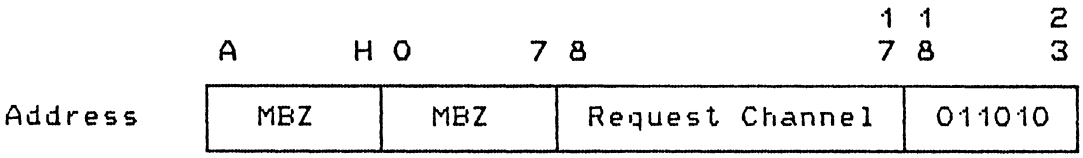
3.1.1.10 Input Hardware Status Word 2 - Function Code #1A

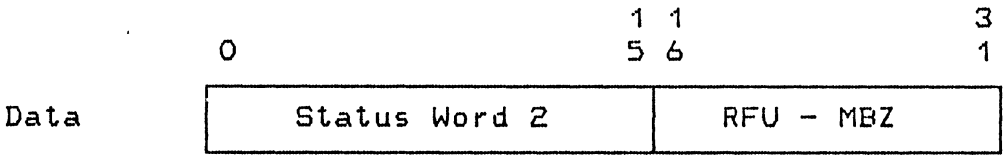
Request Cycle





Response Cycle

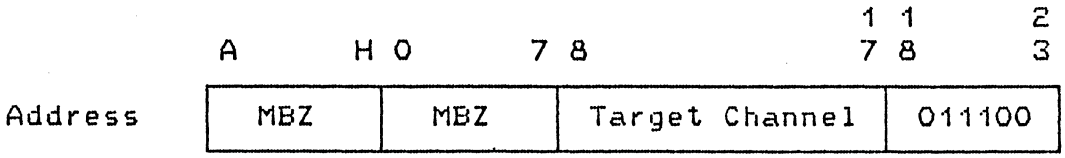


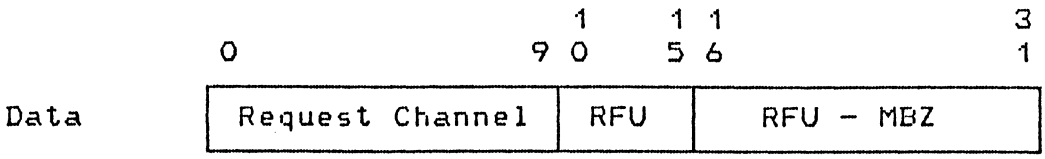


This command is included primarily for diagnostic purposes, and would not be used by the operating system under normal circumstances. Status Word 2 is also returned in the Request Body at completion or termination of a function. The precise meaning of this status word is device and controller specific.

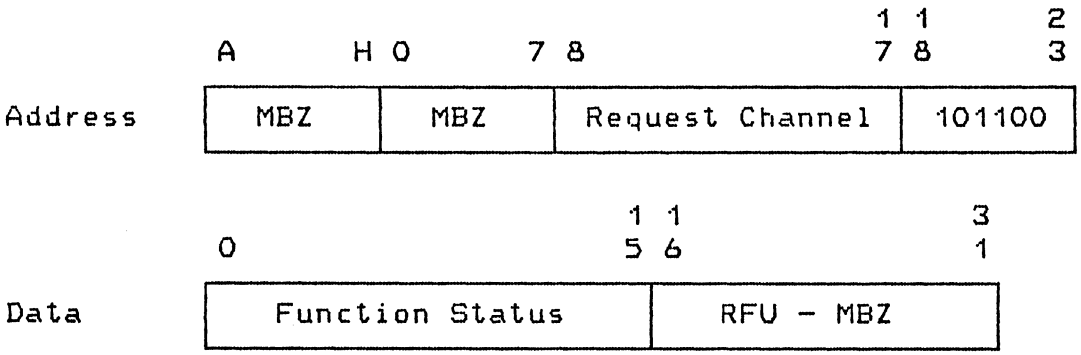
3.1.1.11 Input Function Status - Function Code #1C

Request Cycle

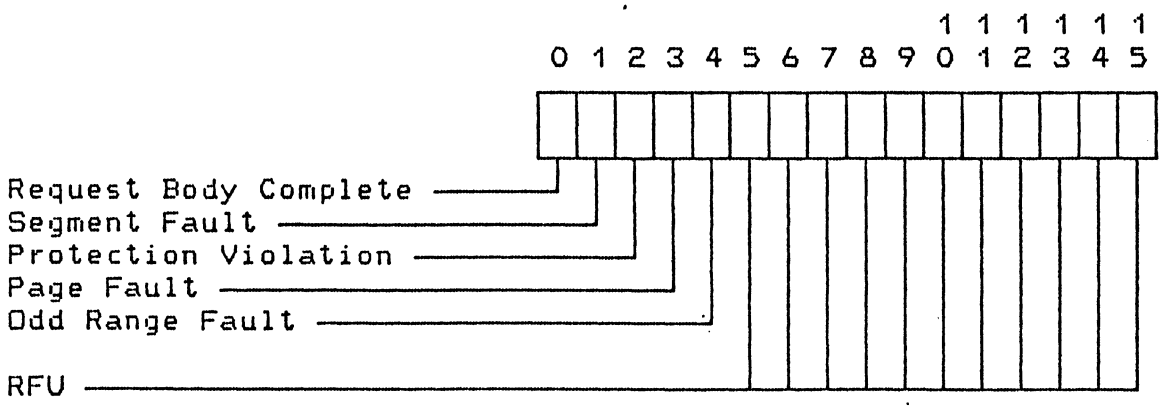




Response Cycle

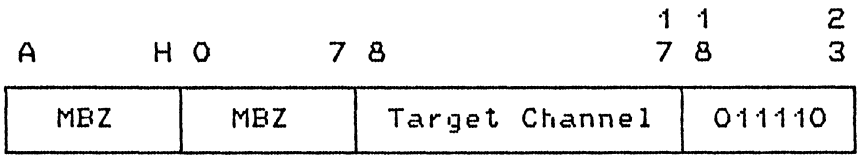


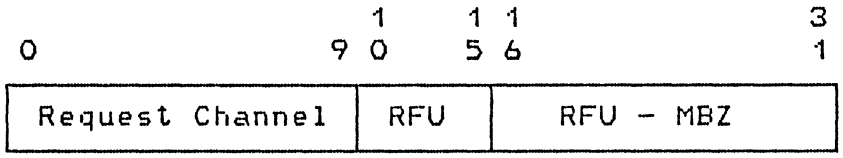
This command is included primarily for diagnostic purposes, and would not be used by the operating system under normal circumstances. The Function Status Word is also placed in each Request Body immediately before completion or termination of a function.



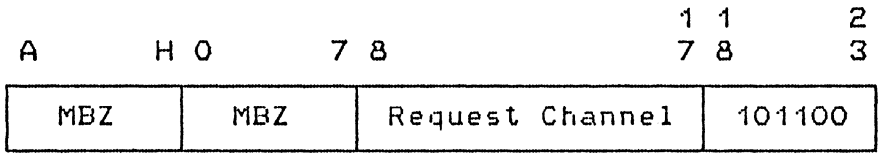
3.1.1.12 Input Header Status - Function Code #1E

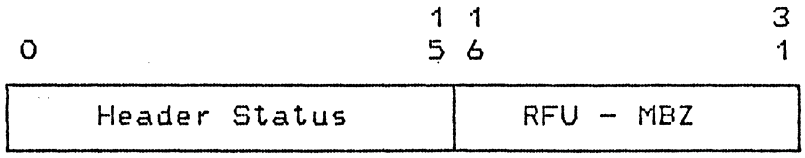
Request Cycle



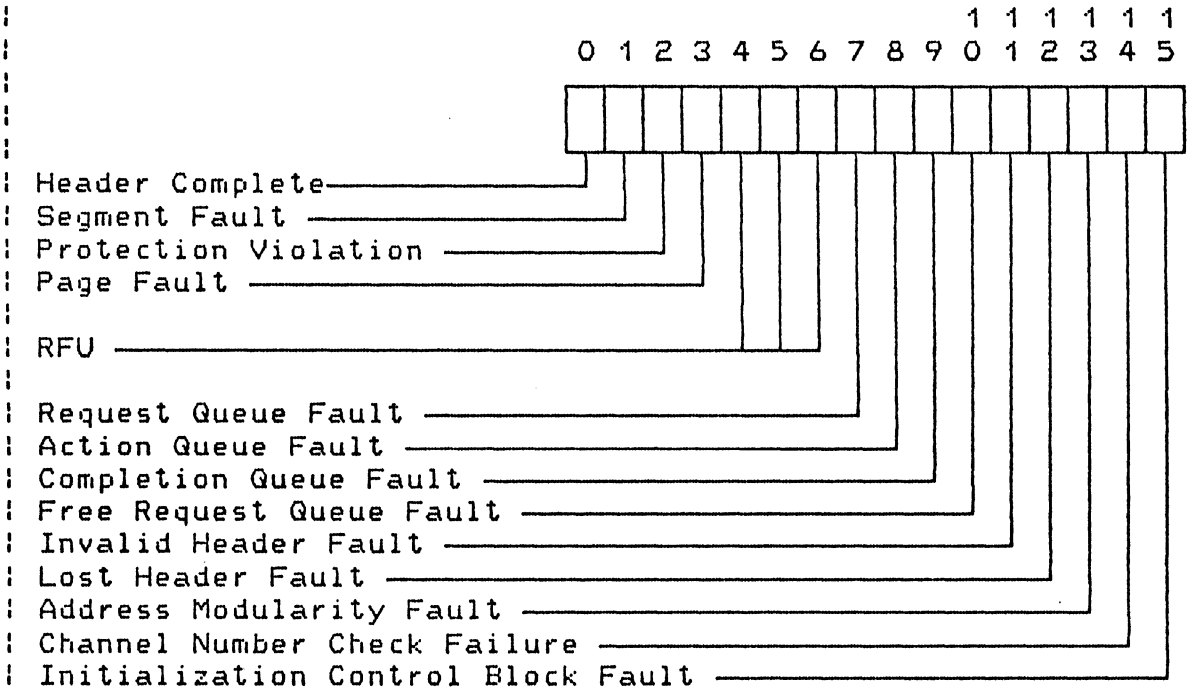


Response Cycle





This command is included primarily for diagnostic purposes, and would not be used by the operating system under normal circumstances. The Header Status Word is also returned in the Request Header immediately before completion or termination of a request.



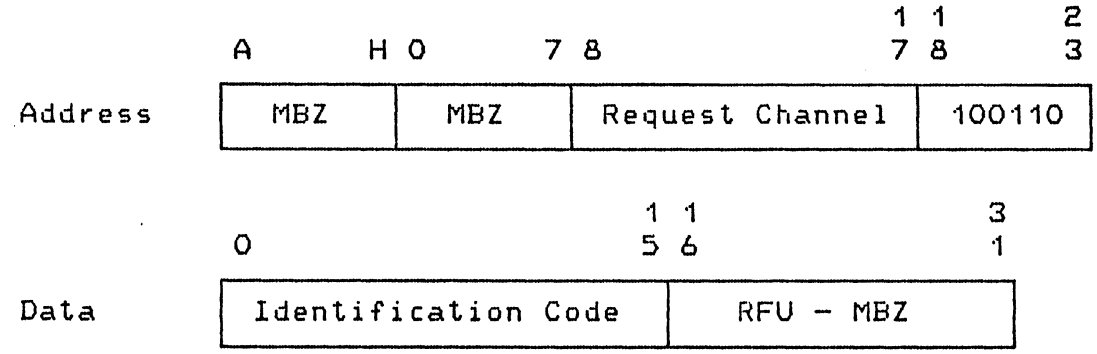
3.1.1.13 Input ID - Function Code #26

Request Cycle

	A	H 0	7 8	17 18	23
Address	MBZ	MBZ	Target Channel	100110	

	0	9 10	15 16	31
Data	Request Channel	RFU	RFU - MBZ	

Response Cycle



This function returns a 16 bit code which identifies the addressed channel. Specific values are assigned for each controller and/or device type. These values are documented in the MRX Megabus EPS-1.

3.1.1.13 Input Firmware Load Status - Function Code #3E

This function is supported only for processors having loadable firmware. Details may be found in the System Maintenance Facility EPS-1.

3.1.2 Processor Initial State

After power up, and after an Initialize command has been received (see Section 3.1.1.1.1), the processor and each channel on it assume the following state:

a) The processor's internal ICB copies are set with all addresses null, and with all ICB Control Flags set to #8000 (physical address mode). The processors internal ICB address fields are set to null.

b) All error and attention status bits except Hardware Status Word 1, bit 11 (QLT status) are reset to zero. QLT status and Bit 0 of each Hardware Status Word 1 (device ready) are set as appropriate.

c) All channels are quiescent, but are prepared to receive commands. Those channels that maintain a stored interrupt level have that level set to 0 (no interrupts).

d) After a full power up (ie., not a power fail restart),

all loadable firmware space is cleared. In all other cases, loadable firmware is left intact.

3.1.3 Data Structure Descriptions

3.1.3.1 Initialization Control Block (ICB)

Before i/o can commence under the memory queued i/o protocol, the target processor must be made aware of the location in memory of the necessary data structures. This is accomplished by constructing in memory a structure called an Initialization Control Block (ICB), which is then made known to the target processor. The target processor retains and uses a private copy of this structure in its own internal RAM.

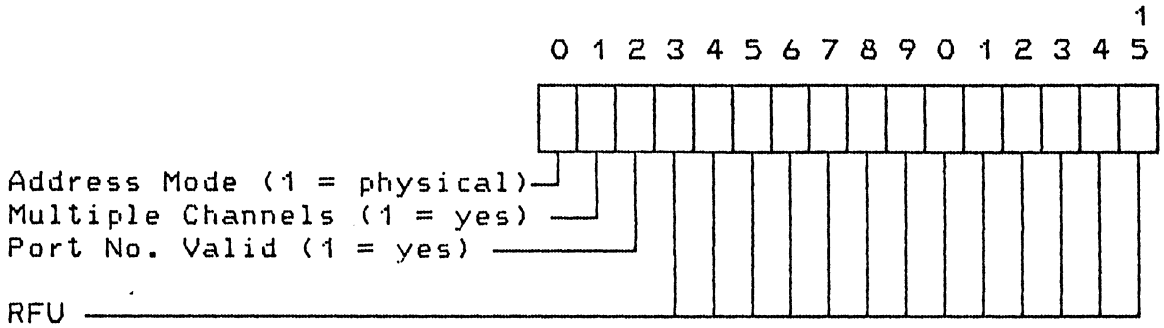
If the target processor is functioning in virtual address mode, the all the addresses in the RHs and the next RB pointer in the RBs are resolved and validated using the three I/O Segment Descriptor (IOSD) fields in the ICB. The data buffer pointers in the RBs are resolved using the IOSDs in the RHs. This address resolution process is described more fully in Section 3.1.4 below. If the target processor is in physical address mode, the IOSDs are ignored and the addresses in the data structures are treated as physical memory addresses. In either case, however, the queue addresses in the ICB itself are physical, not virtual. This address mode selection is made via Bit 0 of the ICB Control Flag Word for all addresses except data buffer addresses, and in Bit 0 of the RH Flag Word for data buffer addresses.

The format of the ICB is as follows:

Word	Contents
0 - 3	I/O Segment Descriptor 1
4 - 7	I/O Segment Descriptor 2
8 - 11	I/O Segment Descriptor 3
12	ICB Control Flags
13	RFU - MBZ
14 - 15	Action Queue Header Address
16 - 17	Request Queue Address
18 - 19	Free Request Queue 1 Address
20 - 21	Free Request Queue 2 Address
. . .	
	Free Request Queue N Address

There must be at least one Free Request Queue address specified; there may be as many as 254. If either the Request Queue address or the first FRQ address are null or invalid, the Action Queue Header address is non null and invalid, or any subsequent FRQ address is invalid, bit 15 of the Header Status Word will be set. This check will take place as soon as the ICB is received by the target processor.

Word 10, the ICB Control Flag Word, has the following contents:



Bit 0 (Address Mode) determines whether addresses in the RBs and RHs are to be interpreted as physical or as virtual; see Section 3.1.4 for further details.

Bit 1 (Multiple Channels) indicates, when set, that the Request Queue contains RHs for multiple device channels. This tells the target processor to check the Local Channel Number field in each RH for validity before commencing i/o, and to set bit 14 of the Header Status Word if it is invalid. If Bit 1 is not set, the target processor will check the validity of the Local Channel Number field only if the field is non-zero.

Bit 2 (Port Number Valid) indicates, when set, that the Port Number field in the RBs are to be validated and used by the target processor.

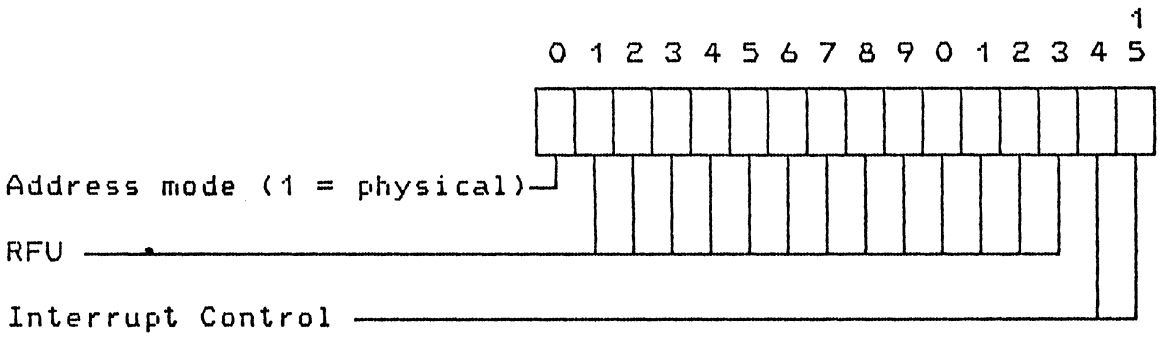
3.1.3.2 Request Queue

TBD - If the Request Queue address or any addresses encountered in the enqueueing or dequeueing process is found to be invalid, bit 7 of the Header Status Word will be set.

The format of a Request Header is as follows:

Word	Contents
0	Flags
1	Queue Priority Word
2 - 3	Pointer to Next RH
4 - 5	Pointer to CQ Header
6 - 7	Pointer to Current (Last) RB
8 - 9	Pointer to First RB
10	Header Status
11	Ring Number
12	Local Destination Channel No.
13	Local Port No.
14	Local Source Channel and Level
15	Active Channel No.
16 - 19	IOSD 1
20 - 23	IOSD 2
24 - 27	IOSD 3
28 - 31	RFU
beyond	Device Specific Extension

The Flag Word is set by software and referenced by the processor. It contains the following indicators:



Interrupt Control
- 00 = suppress all interrupts
- 01 = interrupt only if CQ empty
- 10 = interrupt when header complete
- 11 = RFU

If Bit 0 (address mode) of the Flag word is set, data buffer addresses in the RBs will be treated as physical; otherwise, they are translated and validated using the IOSDs in the RH as described in Section 3.1.4.

The Priority Word is a 16 bit unsigned quantity which controls the enqueuing and dequeing process, and is used as described in the MRX EPS-1 for the CPU queue instructions. It is set by software and referenced by the processor. Section 3.1.5 below describes the queue management process in detail.

The Pointer to the Next RH is set by software and referenced by the processor. It is modified by the processor during the dequeuing process. The address can be virtual or physical, depending on the address handling mode selected in the ICB.

The Pointer to the CQ is set by software and referenced by the processor. It may be either virtual or physical, depending on the address handling mode.

The Pointer to the Current RB is set by software, and is managed by the processor as is works its way through the RB list. It is always set to point to the current active RB. At the normal completion of a sequence of RBs, it will point

to the last RB. If termination is abnormal, it will point to the RB that was active at the time of termination. It may be either virtual or physical, depending on the address handling mode.

The Pointer to the First RB is set by software and referenced by the processor. It may be either virtual or physical, depending on the address handling mode.

The Header Status Word is set by the processor to reflect any of a variety of faults that can occur during processing of a RQ. Immediately before termination or completion of processing on the header, Bit 0 is set to indicate that processing is complete. Section 3.1.1.12 describes the Header Status Word in detail.

The Ring Number is used for read and write access checking of data buffers when operating in virtual memory mode. Section 3.1.4 describes this memory management in detail.

The Local Destination Channel Number specifies the physical i/o channel on the local bus to which the RH is directed. It is set by software and referenced by the processor. If Bit 1 of the ICB Control Word (see Section 3.1.3.1) is set, this channel is can be used by the processor to chose from among possible device channels, and the channel will be checked for validity. This permits requests for multiple physical channels to be placed in a single RQ. If Bit 1 of the ICB Control Word is zero, the channel number will be ignored if it is zero, otherwise, it will be checked for validity. If Bit 2 of the ICB Control Word is set, indicating the presence of a Local Port Number, then only the portion of the Local Destination Channel Number which pertains to the controller base channel will be validated. Validity errors result in abnormal termination and the setting of Bit 14 in the Header Status Word.

The Local Port Number specifies a controller port number, and is valid only if Bit 2 of the ICB Control Word (see Section 3.1.3.1) is set. It is set by software and used by the processor to chose from among ports on a controller. It is of particular use in situations in which multiple controllers are sharing queues. This subject is presented in detail in Section 4 below.

The Local Source Channel Number is set by the software and

referenced by the processor. Typically, it will be a CPU channel number. Appended to it is a level number. These fields are used by the processor when sending an interrupt. It is not verified by the processor.

The Active Channel Number is set by the processor to reflect the physical channel that is acting on this RH.

The IOSDs in the RH are used only in the resolution of the Data Buffer Pointers in the RBs. The detailed contents of the IOSDs is discussed in Section 3.1.4.1 below.

The format of a Request Body (RB) is as follows:

Word	Contents
0 - 1	Pointer to Next RB
2	Function Code
3	Hardware Status Word 1
4	Hardware Status Word 2
5	Function Status Word
6 - 7	RFU
8 - 9	Current Buffer Descriptor Ptr.
10	No. of Send Buffers
11	No. of Receive Buffers
12 - 15	Buffer Descriptor 1

.
.
.

Buffer Descriptor N

TBD - descriptions, esp Function Status.

The format of a Buffer Descriptor is as follows:

Word	Contents
0 - 1	Pointer to Data Buffer
2	Range
3	Residual Range

3.1.3.3 Action Queue

The Action Queue serves as a repository for Request Blocks which are currently in use. It is an optional structure. If an Action Queue is present, each Request Block will be enqueued to it as soon as it is detached from a Request Queue or Free Request Queue, and will remain there until such time as it is placed on a Completion Queue. This feature assists system resilience and problem diagnosis by eliminating, to the extent possible, the likelihood that a request will "disappear" due to hardware malfunction.

The presence of an Action Queue is indicated to a processor by providing a valid physical address in the Action Queue Header Address field of the ICB. If the address is null, the Action Queue processing step is skipped. If the Action Queue address or any addresses encountered in the enqueueing or dequeueing process is found to be invalid, bit 8 of the Header Status Word will be set.

3.1.3.4 Completion Queue

TBD - If the Completion Queue address or any address encountered in the enqueueing or dequeueing process is found to be invalid, bit 9 of the Header Status Word will be set.

3.1.3.5 Free Request Queue

TBD - If a Free Request Queue address or any address encountered in the enqueueing or dequeueing process is found to be invalid, bit 10 of the Header Status Word will be set.

3.1.4 Memory Address Handling

3.1.4.1 Virtual Address Handling

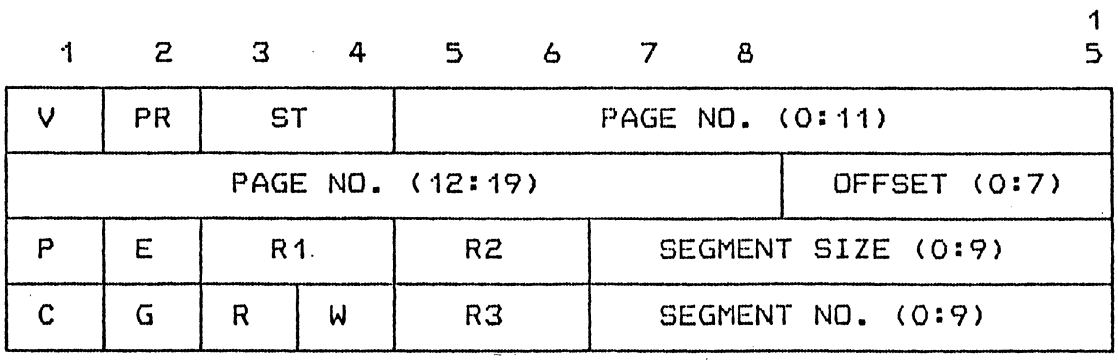
Processors may function in either virtual or physical address handling modes. In the first of these, all pointers are treated as physical memory addresses, and are subject neither to address translation nor to access checking. In this mode, I/O Segment Descriptors (IOSDs) are ignored.

In the normal situation, processors will be functioning in virtual address mode. In this mode, both translation and access checking will be done on all pointers. The method closely parallels that described in the MRX EPS-1. The mechanism to change address handling mode is detailed in Sections 3.1.3.1 and 3.1.3.2 above.

Physical or virtual address handling may each be chosen two points. Bit 0 of the ICB Control Flag Word determines whether addresses in the RHs and the next RB Pointer in the RBs are to be treated as physical or virtual. If virtual addressing is selected, the three IOSDs in the ICB are used for mapping and address control. Bit 0 of the RH Flag Word determines whether data buffer addresses in the RHs are to be treated as virtual; if so, the three IOSDs in the RH are used.

Note that the pointers in the ICB will always be physical, regardless of the processor address handling mode.

The format of an IOSD is as follows:



The individual IOSD fields have the meanings listed below. Those fields marked RSU are ignored:

V - if set to 1, the IOSD is valid. If set to 0, the IOSD is ignored.

PR - RSU (privilege indicator)

ST - RSU (segment type)

P - if set to 1, segment is paged.

E - RSU (execute-permit indicator)

R - if set to 1, and if the Ring Number in the RH is less than or equal to R2, data may be read from this segment. If an unpermitted attempt to do so occurs, Bit 2 of the Function Status Word will be set and processing of the RH is terminated. Note that this corresponds to a data write.

W - if set to 1, and if the Ring Number in the RH is less than or equal to R1, data may be written into this segment. If an unpermitted attempt to do so occurs, Bit 2 of the Function Status Word will be set and processing of the RH will terminate. Note that this corresponds to a data read.

R1 - highest (ie., least privileged) ring number into which a memory write (data read) is allowed

R2 - highest (ie., least privileged) ring number from which a memory read (data write) is allowed

R3 - RSU (highest call bracket ring number)

C - RSU (compatibility indicator)

G - RSU (gate segment indicator)

Segment Size - if P = 0, this is the size, in pages, of this segment. If P = 1, this is the size of the page table.

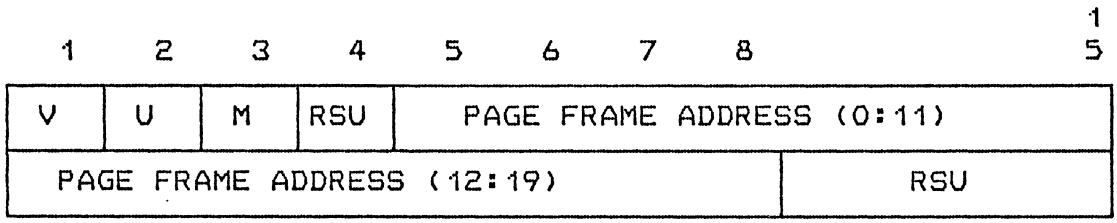
Segment Number - valid segment number for this segment.

Page Number - if P = 0, this field defines the start of the non-paged segment in main memory. If P = 1, this and the Offset field define the start in main memory of the Page Table for this segment.

Offset - if P = 0, this field must be zero. If P = 1, this field and the Page Number define the start of the Page Table for this segment.

Fields R, W, R1, and R2 are meaningful only for IOSDs in the RHs. For the IOSDs in the ICB, they are ignored.

Segments may be either paged or unpaged. If paged, the IOSD defines a Page Table which contain two word Page Descriptor entries of the following form:



Within a Page Descriptor (PD) entry, fields have the following form:

V - RSU (valid indicator)

U - PD used indicator. This field is set by the processor using memory lock (read-modify-write) whenever data is written from or read into the memory defined by the PD.

M - PD modified indicator. This field is set by the processor using memory lock (read-modify-write) whenever data is read into the memory defined by the PD.

Page Frame Address - 20 bit physical page frame address

TBD - address calculation description

3.1.4.2 Structure Address Constraints

Two important constraints are applied to the addresses of the RBs and RHs. First, they must be located on modulo 16 word boundaries. If a pointer to one of these structures is encountered for which the low order 4 bits of the word address are not zeroes, the MQIO operation in progress will terminate abnormally and bit 13 of the Header Status will be set. If the offending address is an RB pointer, the RH will be enqueued on the Completion Queue. If it is an RH pointer

that violates the rule, the controller will attempt to signal the fault as if it were an attention item, ie., it will attempt to dequeue a RH from FRQO and posting it on the Completion Queue. If the controller cannot find a legal RH on FRQO, it will set bits 12 and 13 of the Header Status.

The second limitation on RBs and RHs is that they not span page boundaries. Violation of this rule may be undetected by the target processor. Only the address of the first word of either of these two structures will be checked for validity.

These two restrictions are placed in order to optimize processing speed and to insure the integrity of pointers.

3.1.4.3 Data Address Constraints

In order to maximize performance, it is a requirement that all data buffer addresses be word aligned. In keeping with this restriction, the Buffer Pointers in the RBs are word pointers. In addition, within an RB, all ranges except the last one in an set (ie., the set of input buffer descriptors or the set of output buffer descriptors) must be even. If this rule is violated, bit 4 of the Function Status Word will be set and the data transfer will terminate abnormally. The last range in a set may be odd; in this case, the data transfer will overrun one byte.

3.1.5 Handling of the Queuing Process

The Request Queue, the Action Queue, the Completion Queue, and the Free Request Queues all are or can be shared by both software and hardware, thus the handling of these queues must be consistent and must conform to the queue management standards presented in Section 3.15 of the MRX EPS-1. Briefly, each queue has a queue header, which contains a pointer to the first item in the queue, a pointer to the last item in the queue, and a lock word. For DPS6 and DPS6+ software, it is assumed that these will be managed using the CPU queue instructions QOH, QOT, DQH, DQA, SQH, and SQA.

For non-DPS6 CPUs and for other processors, the following sequence must be observed. To gain access to the queue, the lock word must first be checked and set. This must be done using a read-modify-write cycle (lock cycle) to insure that only one user at a time can have access to the queue. If a

processor finds that the queue lock word bit 15 is set, it must wait a short time, then try again to get access to the queue. If bit 15 is not set, it must be set, and the processor's channel number must be placed in bits 0 through 8.

Having gained access to the queue, a processor may search by following the chain of frame pointers. To dequeue an item, a processor sets the frame pointer from the prior queue frame or header to the value contained in the next frame pointer of the frame to be dequeued. To enqueue an item, the next frame pointer in the prior queue frame is set to point to the new frame, and the next frame pointer in the new frame is set to the value previously in the prior frame. Note that processors using the MQIO always "queue on tail", that is, an item being linked to a queue is placed in the queue after all items having lesser or equal priority word values, and before any items having greater priority word values.

3.1.6 I/O Scenarios

TBD - illustrative examples

3.2 Device and Processor Specific I/O

3.2.1 Magnetic Peripherals

TBD - collection of statistics

3.2.2 Communications

TBD

3.2.3 Cluster Controller

The format of the device specific extension to the RB is as follows:

Word	Contents
32	Source Node No.
33	Destination Node No.
34	Destination Channel No.
35	Free Request Queue No.
36 - 37	Software Id
38 - 47	RFU

IV

SOFTWARE IMPLEMENTATION AND APPLICATION

4.1 Sharing of Queues

In the simplest form of MQIO, each peripheral channel has its own set of queues. There is, however, nothing in the design of MQIO that precludes queue sharing. The queue locking mechanism employed by the DPS-6 queue instructions is used by all MQIO controllers to maintain queue consistency, and in fact, it is anticipated that Completion Queue sharing will be the norm for most system software.

Sharing of Request Queues can be done in three different ways, all supported by the MQIO protocol. In the first scheme, each Request Header contains the peripheral channel number of its target. Controllers associated with the queue each scan the queue looking for Request Headers that pertain to them; when a controller finds a Request Header that belongs to it, the Header is handled as described above. This queue sharing method may offer some advantages for the driver software, but conceptually it is little different from having independent queues.

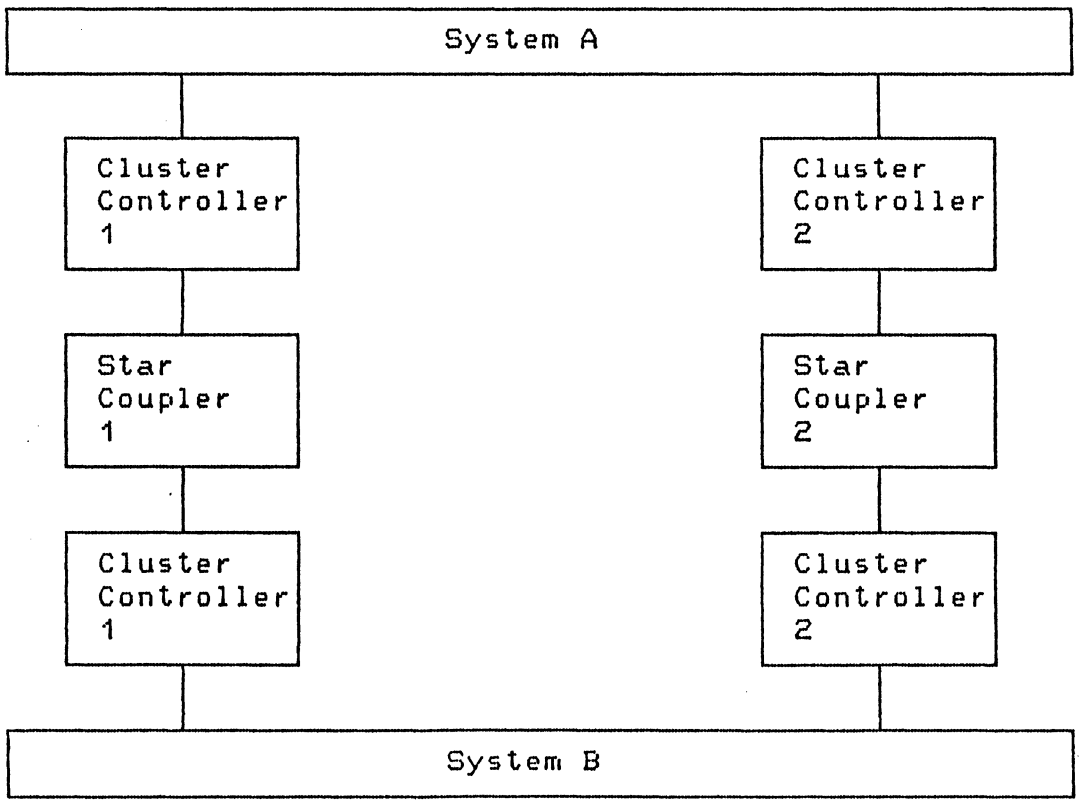
With the second method, Request Headers do not contain target peripheral channel numbers. Thus, any controller on the queue will seize and act upon any Request Header that it finds. This method is appropriate when all channels associated with the queue can perform the same functions. This scheme is used in the multiple cluster controller case described below. A variant of this scheme permits some Request Headers to be marked with channel numbers, and some to be left unmarked, thus allowing the driver to explicitly direct some i/o activities to

particular channels while permitting others to be handled by any available controller.

With the third method, certain Request Headers are marked with a device port number, but not with a channel number. Section IV below presents the details of this approach.

4.1.1 Multiple Cluster Controllers Sharing Queues

For resiliency and throughput reasons it is convenient to have multiple paths between nodes of a cluster. To do this, the following configuration might be used:

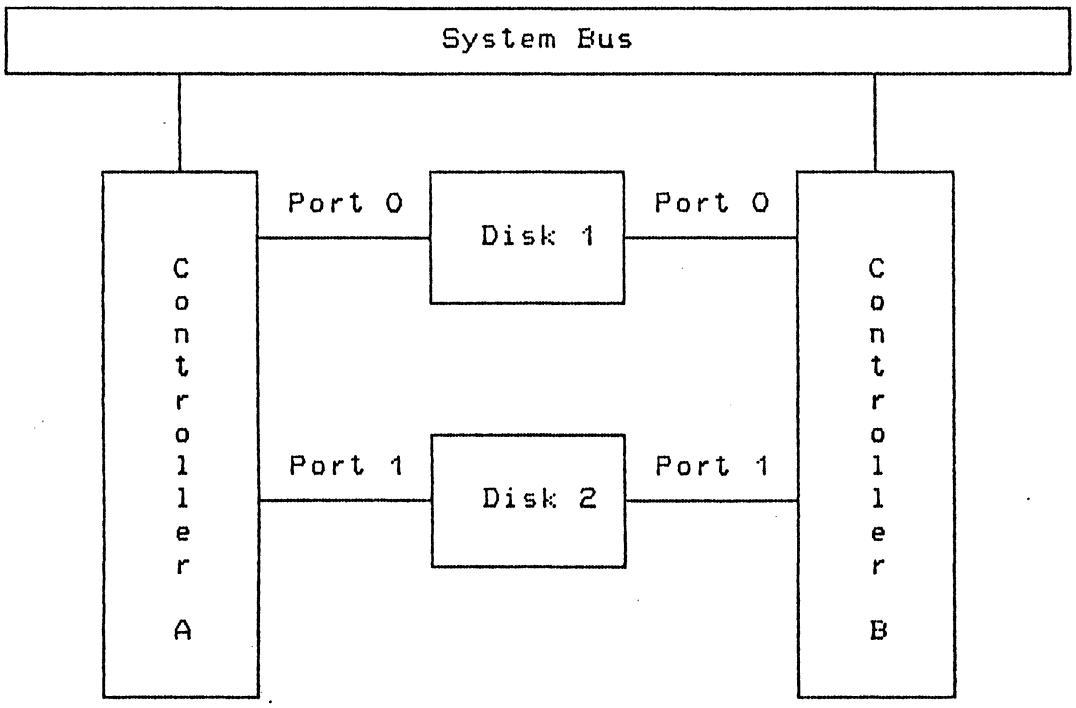


If on each system both cluster controllers share the same queues, throughput could be nearly double that of a single cluster connection. Except in the failure case, the existence of more than one cluster controller could be completely transparent to the driver software. In the case of failure during data transfer, the software would have to unwind the failed transaction and disable the failed cluster

link. For other failures, it would suffice merely to disable the failed path. For even greater throughput and/or fault tolerance, three or more cluster links could be established, all sharing the same queues. Except for collisions at the queue locks and on the system busses, there should be no friction between multiple cluster links configured in this fashion.

4.1.2 Multiple Disk Controllers with Multi-port Drives

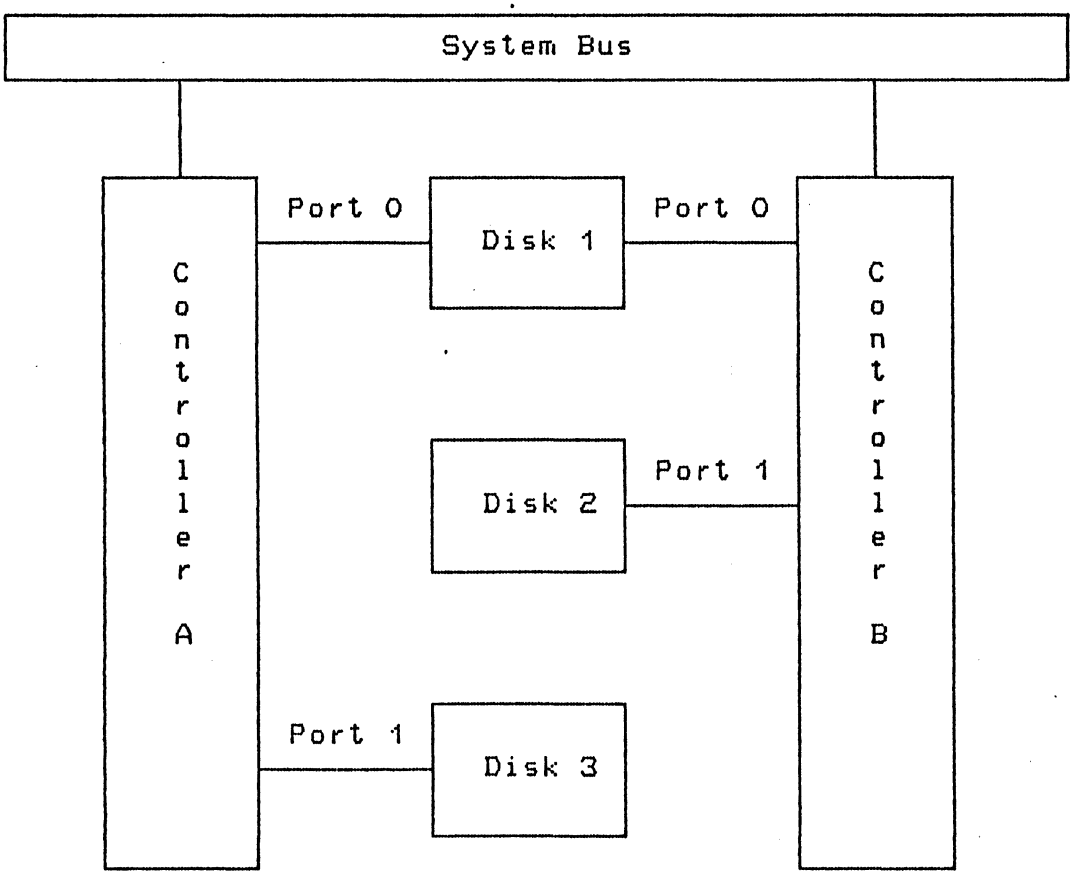
Consider the following configuration:



This combination offers both throughput and resiliency advantages over a similar single controller configuration, but with current i/o techniques, a penalty would be paid in the driver software. This penalty arises primarily because each view of each disk is identified by a separate channel number. By contrast, under MQIO, Request Headers could be marked with port numbers (as the term is used here, port number refers to the controller's port number, not the disk device's port number) but not channel numbers. Both controllers could share the same set of queues, and either

controller could handle i/o to a particular dual port disk. This behavior would be largely invisible to the drivers. If a controller failure were detected, recovery software would have to unwind the failed operation, which could be found on the Action Queue, and disable the failed controller. Otherwise, physical i/o processing would continue as before, although at a slower pace. If multi-port disks were available, this scheme could be expanded with more controllers.

Queues under MQIO may be at the physical channel level as well as at the controller level. This leads to useful variant of the shared queue technique which could be applied to the following configuration:



Here, controllers "A" and "B" could share a set of queues for disk 1, but disks 2 and 3 could be driven from separate queues. Alternatively, all three disks could share the same

queue set, but the Request Headers for disks 2 and 3 could be tagged with channel numbers, while the Headers for disk 1 could be tagged only with port numbers. Either way, access to disk 1 (perhaps the system disk) would be substantially immune from a single controller failure.

4.2 Configuration Considerations

It is generally recognized that the configuration process in future operating systems must be much less visible to the user than it is today under VS1. This, combined with the new dimension of configuration possibilities presented with MQIO, requires substantial help at the hardware level. It must be possible on new MQIO controllers to answer the following questions:

1. Which peripherals are connected to which controllers? In many cases, this is trivial to determine, but there currently are some 4, 6, and 8 port disk controllers for which mere knowledge of device id's may be insufficient.

2. Which channels on two controllers correspond to the same dual ported device? It is of course possible to determine this by writing to a device from one port, then reading it from the other. This is quite inconvenient, and a better method should be sought.

It would be best if a uniform solution suitable for all MQIO controllers could be found and implemented. A good place to start would be to require all such controllers to support a supervisory channel separate from the peripheral channels. This supervisory channel could supply controller-wide configuration information as well as other special diagnostic and resiliency functions.

Appendix I

Reference Documentation

60149789, MRX System and Central Subsystem EPS-1, Revision D

60149831, DPS6 32 Bit Product System Management Facility EPS-1, Revision E

60149832, MRX Megabus EPS-1, Revision E

Appendix II

Abbreviations

AQ	- Action Queue - see Section 3.1.3.3
CPU	- Central Processor Unit
CQ	- Completion Queue - see Section 3.1.3.4
DIIO	- Direct Interface I/O
EPS-1	- Engineering Product Specification, Part 1
FRQ	- Free Request Queue - see Section 3.1.3.5
I/O	- Input/Output
IOSD	- I/O Segment Descriptor - see Section 3.1.4
ICB	- Initialization Control Block - see Section 3.1.3.1
MBN	- Must Be Null Address
MBZ	- Must Be Zero
MQIO	- Memory Queued I/O Protocol
NULL	- An Address of All Zeroes
RAM	- Random Access Memory
RFU	- Reserved for Future Use
RB	- Request Body
RH	- Request Header
RQ	- Request Queue - see Section 3.1.3.2
RSU	- Reserved for Software Use