UNISYS

Entry/ Medium Systems Group
PASADENA DEVELOPMENT CENTER

1993 5329

V500 DATA TRANSFER MODULE

UNISYS RESTRICTED

ENGINEERING DESIGN SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED, ADDED, DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	ECN 51718	Initial Issue	L. Simpson L. Simpson P. Ghalambor [illegible]	P. Ghalambor [illegible] R. Young [illegible] 7/31/87 H. Li [illegible] A. Gomez [illegible]

PAS 3908 REV. 6-87

EDS43

UNISYS

Entry/ Medium Systems Group
PASADENA DEVELOPMENT CENTER

1993 5329

V500 DATA TRANSFER MODULE

UNISYS RESTRICTED

ENGINEERING DESIGN SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED, ADDED, DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	ECN 51718	Initial Issue	L. Simpson [illegible] P. Ghalambor [illegible]	P. Ghalambor [illegible] R. Young [illegible] 7/31/87 H. Li [illegible] A. Gomez [illegible]

PAS 3908 REV. 6-87

EDS43

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

1 PURPOSE

The purpose of this document is to describe the function and operation of the V500 Message Level Interface Data Transfer Module (MLI-DTM).

2 RELATED DOCUMENTS

1997 5390	V Series Instruction Set Specification
1993 5279	V500 Architecture Specification
1993 5253	V500 I/O Memory Concentrator Specification
2323 7399	Message Level Interface Specification
1993 5295	V500 System Maintenance Controller Spec
1993 5303	V500 Maintenance Subsystem Specification
1993 5337	V500 Fault Detection Design Guidelines
1995 5301	V500 ECL/CMOS/TTL Circuit Rules

3 FUNCTIONAL OVERVIEW

3.1 BASIC FUNCTIONS

The V500 MLI-DTM consists of the following modules:

o I/O Bus Control Module

o Microprocessor Module

o Message Level Interface Control Module

An I/O bus interface has been oesigned for the MLI-DTM to perform the access and communication functions.

It is built of TTL MSI/SSI parts, PALs, PROMs, 1Kx4 static RAMs, and an ALS 2800 gate array. The ALS gate array handles the error correcting code on the I/O bus.

4 I/O BUS DEFINITION

4.1 BASIC FUNCTIONS

The V500 system I/O busses perform the following functions:

o Memory Access. The I/O busses are used by the Data Transfer Modules (DTMs) and System Maintenance Controllers (SMCs) to read, write, and otherwise manipulate System Memory. The actual memory operations are done by the I/O Memory Concentrators (IOMCs).

o Inter-I/O-Module Communication. DTMs and SMCs may communicate with each other using the I/O busses. The basic unit of communication is called a message.

o CPU Communication. Each of the CPUs in a V500 system can inform the I/O subsystem that it wants to initiate an I/O operation; similarly, any CPU may accept an I/O complete interrupt.

4.2 CONNECTIVITY, MULTIPLICITY, AND PERFORMANCE

A V500 system has two independent I/O busses. Each connects to card slots for the following I/O modules:

Single cabinet system:

- o One IOMC slot
- o Six DTM slots
- o One SMC slot

Dual cabinet system:

- o Two IOMC slots, of which at most one may be occupied.
- o Twelve DTM slots
- o Two SMC slots

As shown in Figure 4-1, each IOMC and SMC connects to one bus; DTMs may connect to both busses. A single cabinet system may contain either one or two IOMCs. If only one IOMC is present, the second I/O Bus may still be used for Inter-I/O-Module communication.

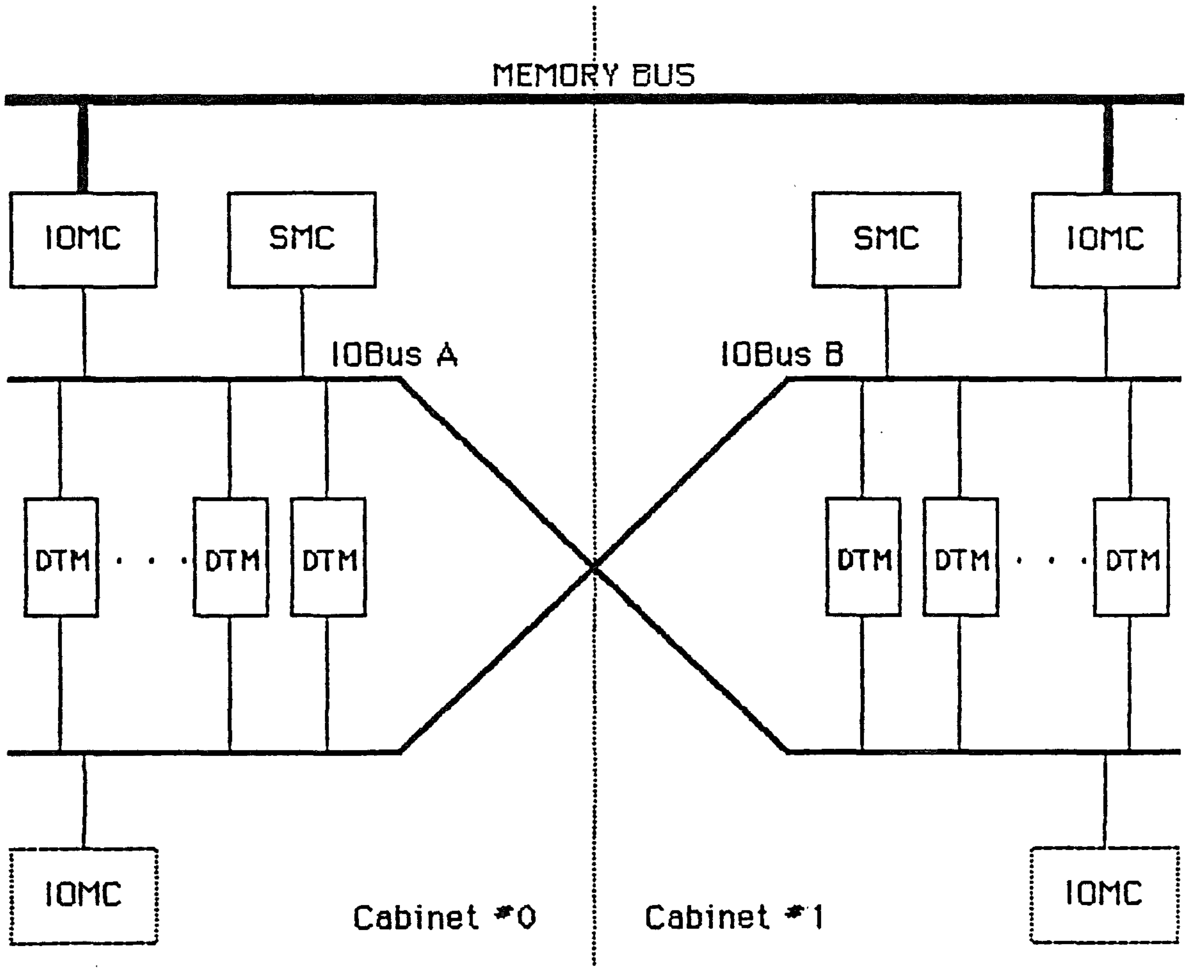

Figure 3-1 I/O Bus Connectivity

Each I/O bus consists of 50 wires; 32 for data, 7 for ECC and 11 for control.

4.3 MESSAGES

The basic unit of communication on the I/O busses is the message. A message consists of one or more 32 bit data words, sent by a module (the initiator) to another module (the responder). The bus protocol allows any module on the bus to be an initiator or a responder.

Once a message has been sent by an initiator to a responder, the responder may, at its discretion, send a reply back to the initiator. The I/O bus connection is not severed until a reply is sent, or the responder decides not to send a reply.

These basic functions are done by sending and receiving messages. For example, if a DTM wants to read data from memory, it sends a message to an IOMC. That message consists of a two digit op-code, together with the address and length of the data it wants to read. The IOMC replies with the data from memory.

Similarly, if a module is to write data to memory, it sends a message to an IOMC. That message consists of an opcode, address, length, and the data to be written. The IOMC replies to this message with an Error word that contains information pertaining to the completion of the command.

Inter-I/O module communication is done similarly. For example, MLI-DTM firmware may be loaded via the I/O bus. The SMC is the initiator, and the appropriate DTM is the responder.

4.4 MESSAGE PHASES

I/O bus operations consist of the following phases:

o Bus arbitration
o Responder selection
c Send message
o Receive reply (optional)

In the first phase, the initiator attempts to gain access to the bus. This may be refused because another module is already using the bus, or because a higher-priority module bid for the bus at the same time. In either case, the initiator encounters a BUS BUSY condition. The I/O bus operation must be retried.

Once the initiator gains access to the bus, the second phase begins. The initiator broadcasts the address of the module it wants to converse with on the bus data lines. Every module connected to the bus checks the address against its own. The addressed module must indicate that it is present, and whether it is reacy to accept a message. If the initiator does not see a Module Busy or a Not Present condition, it may go to the third phase.

In the third phase, the initiator sends the message to the responder, 32 bits every TTL system clock. Once the message has been sent, the responder may either disconnect, or send a reply (phase 4).

4.5 IOMC PECULIARITIES

The IOMC differs significantly from the other modules on the bus. There is at most one IOMC connected to a bus; it provides the interface from that bus to memory, and it contains the logic for CPU communication.

Another peculiarity of the IOMC is that it is the only module on the I/O Bus that is not microprocessor-based. This means that its understanding of particular messages is hardwired, instead of being determined by firmware.

Also, the IOMC never acts as an initiator; it is always a responder. All the IOMC's actions are performed at the request of another module.

4.6 CPU-I/O SUBSYSTEM COMMUNICATION

I/O Complete Interrupt reporting consists of the module that wants to cause an I/O Complete Interrupt sending a message to the IOMC indicating the type of I/O Complete Interrupt (Normal, Real Time, or Error), and which processor to poll first. The IOMC polls each of the CPUs in turn, looking for one that will accept an interrupt of the specified type.

If the IOMC receives a second I/O complete interrupt command before it finds a processor that will accept the first interrupt, it ORs the new interrupt type with the old, and continues to poll the processors. For details, see the IOMC specification.

Initiate I/O and other CPU-to-I/O subsystem communication are more complicated. The IOMCs appear to the CPUs to be two memory boards; one stores the time-of-day, and the other (the IOMC mailbox register) is used for subsystem communication.

Each IOMC mailbox register can be written to by any CPU as if it were memory. Once that register has been written by a CPU, the mailbox register 'memory board' goes busy until the value in the register has been fetched by an I/O subsystem module. As long as the mailbox register is busy, the CPUs may not perform another write to the same register.

If there are two IOMCs in the system, the IOMCs and the CPU's memory controllers (MCACM) cooperate to make the pair of IOMCs look like a single IOMC. When a CPU writes the time-of-day, both IOMCs load their registers. However, only one IOMC (the master) responds to a read time-of-day command. Similarly, MCACM allows a mailbox register write to be issued only if there is an empty mailbox register; if neither mailbox register is empty, the write waits.

Thus, in a single IOMC system, the mailbox register goes busy after a single write, and stays busy until an I/O subsystem module empties the register. In a two IOMC system, two write mailbox register operations may be performed before the mailbox goes busy; the master IOMC registers the first, and the slave IOMC registers the second. The mailbox goes not busy when either IOMC's register is emptied.

4.6 CPU-I/O SUBSYSTEM COMMUNICATION (Continued)

When an IOMC's mailbox register is written by a CPU, the IOMC captures ten digits of data from the memory data bus, and one digit of address from the address bus. It is intended that the digit of address be used by the microcode to distinguish types of CPU-to-I/O subsystem messages (Initiate I/O, Discontinue I/O, or System Initialization).

Current I/O makes all CPU to I/O subsystem messages look like Initiate I/O requests, in which messages intended for the I/O subsystem hardware, such as execute self-test or Cancel, are sent to the I/O subsystem as Initiate I/O Requests for a fictitious DLP (channel 08). Future I/O (IOCBs) will require new, as yet unspecified, types of CPU to I/O subsystem messages.

Each I/O Bus has two control lines (IIOReqL and IIOAckL) that are used exclusively to manage the corresponding IOMC's mailbox register. IIO stands for Initiate I/O, Req stands for request, Ack for Acknowledge, and L indicates that the signal is active low. IIOReqL is asserted by the IOMC when a mailbox write has been done by a CPU. Several modules on the I/O Bus may monitor this signal, but only one module (the Message Router) responds.

The I/O module that is to be the message router is determined at system initialization time. Currently, the lowest slot numbered DTM is designated as the message router. If the message router module breaks, it is possible to reassign this responsibility to another module.

When the message router notices that the IIOReqL line is asserted, it initiates a message to read the mailbox register. Once the message router has successfully read the mailbox register, it asserts IIOAckL to indicate to the IOMC that the mailbox register is now empty. Both IIOReqL and IIOAckL are dropped, the mailbox 'memory board' goes not busy, and the I/O subsystem is ready to accept another Initiate I/O message from a CPU.

4.7 BUSY MODULES

The second phase (Responder Selection) of the four I/O bus transaction phases is more than the previous description implies. Briefly, reckless use of the Module Busy condition could result in a deadlock.

In the second phase of an I/O bus transaction, the initiator broadcasts the address of the the module it wants to converse with on the data lines of the bus. Every module connected to the bus checks the address against its own. The addressed module must indicate that it is present, and if it is ready to accept a message. If the initiator does not see a Module Busy or a Not Present condition, it may proceed to the third phase.

Consider two modules (A and B). A wants to send a message to B. First, A sets its Module Busy flip-flop, so that any module sending it a message sees a Module Busy condition. A then puts the message into a message buffer, and attempts to send the message. Meanwhile, module B has not been idle. B has a message to seno to A. Of course, B follows the same procedure; it set the Module Busy flip-flop, puts the message into a buffer, and attempts to send it.

Clearly, A's message will not reach B, because B has Module Busy. B's message cannot reach A, because A is also Module Busy. A simple retry by A or B will have the same problem. Until either A or B resets its Module Busy flip-flop, A and B are deadlocked.

Hence, we cannot use the Module Busy flip-flop for all messages. The basic problem is one of buffer management; when a message comes in, where does it go? The current MLI-DTM implementation has a single buffer consisting of ten 1Kx4 static RAM chips, that is used for all messages (incoming and outgoing). A single message may occupy the entire buffer; therefore, if the microprocessor in module A is formatting an outgoing message in the buffer at the same time that a message is coming in from module B, there is no place for module A to put the incoming message.

4.7 BUSY MODULES (Continued)

A simple alternative is to give the incoming message priority over the outgoing one. The microprocessor gives up the buffer, and the incoming message overwrites anything that was in the buffer. The microprocessor must handle the incoming message, and then make another attempt to format and send the original outgoing message. Several such attempts might be required before the message can be sent.

This alternative is known as Non-Busy Message Initiation.

Why not use Non-Busy Message Initiation for all messages? Unfortunately, it is not always possible for the microprocessor to reconstruct the outgoing message. For example, hardware in the DTM puts data from a peripheral device directly into the I/O bus buffer. This means that if the buffer is in use, the entire I/O would have to be retried to reconstruct this data.

Clearly, a design with multiple message buffers would alleviate the problem. Unfortunately, an infinite number of buffers would be required to eliminate it completely.

Modules on the I/O bus use the Module Busy flip-flop and Non-Busy Message Initiation. How does one decide which method to use? Fortunately, there is a simple protocol that prevents deadlock.

Deadlock prevention protocol incorporates the following rules:

1 There exists a set of modules (A) that are never busy.

2 A module that has its Module Busy flip-flop set may only initiate messages to modules in set A.

3 DTMs that are not message routers always initiate messages with their Module Busy flip-flop set.

The IOMCs are never busy. They are only responders; whenever they are doing anything, their I/O bus is in use, so that any module attempting to send a message on that bus sees a Bus Busy condition.

4.7 BUSY MODULES (Continued)

By convention, the message router always uses Non-Busy Message Initiation. Hence, set A consists of the IOMCs and the message router module. SMCs must then abide by Rule 2; when they have their Module Busy flip-flop set, they may only initiate messages to IOMCs and to the message router. SMCs may only send messages to DTMs using Non-Busy Message Initiation.

The message router can only send messages to DTMs and SMCs using a Non-Busy Message Initiation. Since all other DTMs that are not the message router can only perform Module Busy initiation, they can only initiate messages to the IOMCs and the message router.

In fact, a DTM acting as a message router will only set Module Busy while handling data coming from or going to a peripheral, and will only initiate messages to an IOMC during that time.

5 PARTITIONING

5.1 COMPONENTS

The MLI-DTM I/O bus interface is partitioned into four pieces. They are:

- o two I/O bus controllers (IOBC); one for each bus
- o a single 4kbyte buffer
- o a Buffer Control

The four pieces are connected as shown in Figure 5-1.

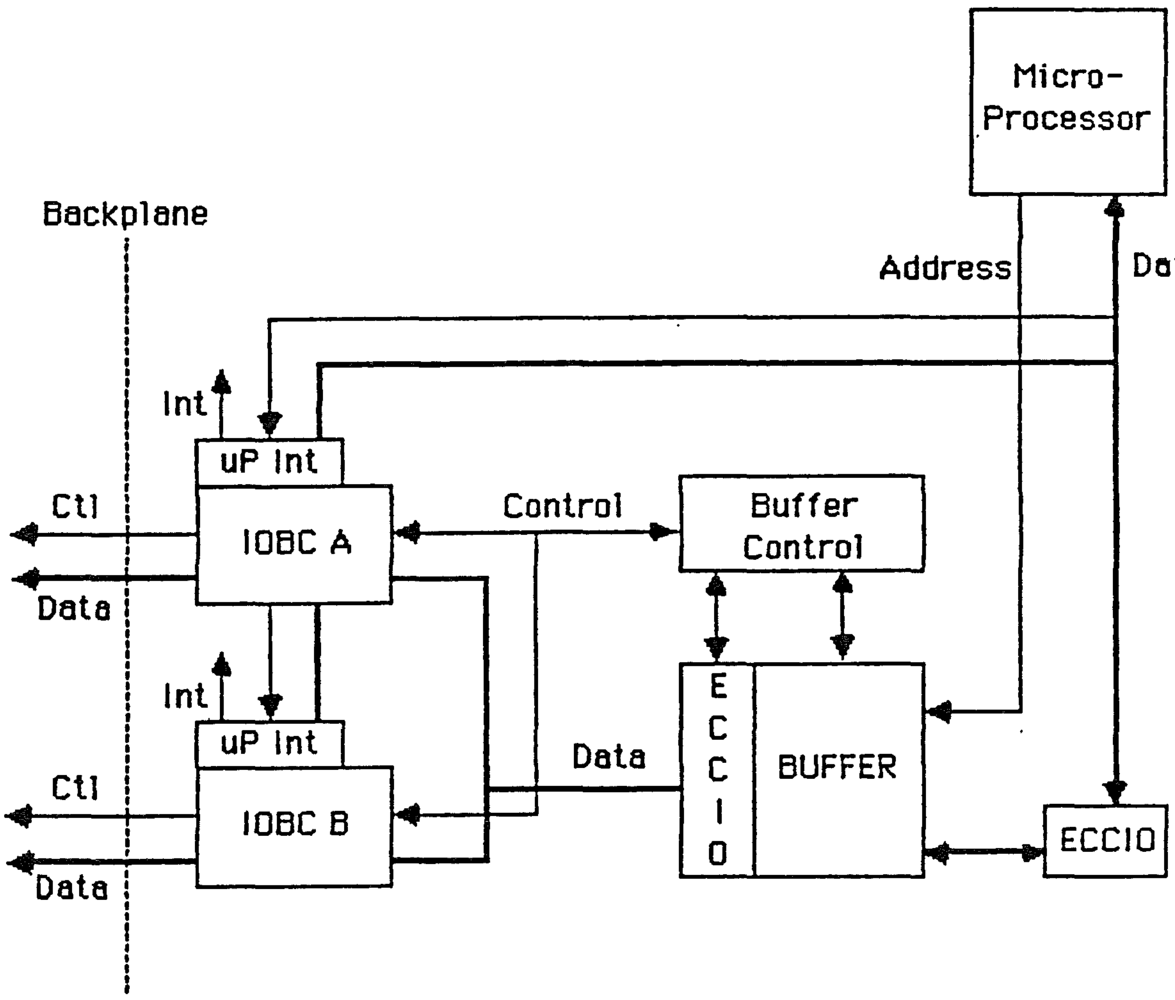

Figure 5-1 I/O Bus Interface Structure

In the MLI-DTM, the MLI Control uses the same buffer. It connects MLI cable to the bases as shown in Figure 5-2.

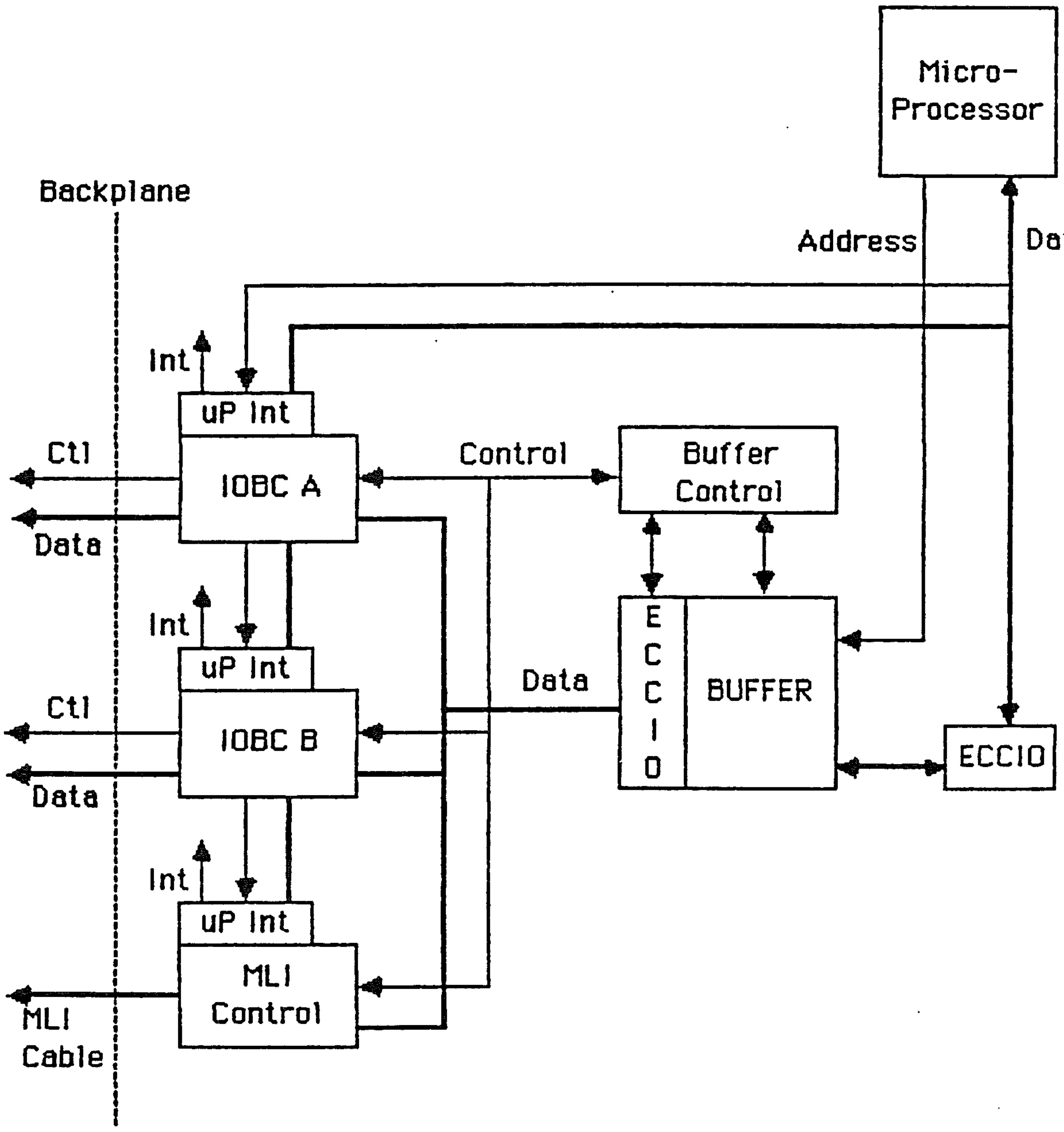

Figure 5-2 DTM-MLI Connection

5.2 COMPONENT FUNCTIONS

When the microprocessor initiates an I/O Bus transaction, it formats the message in the buffer, then commands one of the two IOBCs to start the transaction. The IOBC does the four phases of the I/O Bus transaction (Bus Arbitration, Responder Selection, Message Send, and Reply Receipt) autonomously, reporting back to the microprocessor via a status register when the transaction is complete, or if some condition such as Bus Busy or Module Busy prevents the transaction from being completed.

Similarly, when a message arrives on an I/O Bus, the IOBC puts that message into the buffer, informs the microprocessor (via an interrupt) of its arrival, and then waits for a command from the microprocessor. If the microprocessor wishes to reply to the message, it formats the reply in the buffer, and issues a Continue command to the IOBC. Alternatively, the microprocessor may indicate that there is no reply to this message by issuing a Disconnect command.

The buffer is a 4K byte static RAM (1K 32 bit+ECC words) that is accessible by both IOBCs, the microprocessor, and, in the MLI-DTM, by the MLI Control. It stores messages and replies. It is also stores the Module ID of the target module when the microprocessor wants to initiate an I/O bus transaction.

The Buffer Control performs two basic functions. First, it is responsible for the movement of data into and out of the buffer. On request from an IOBC, it reads the Module ID or data from the buffer, and stores received data into the buffer.

The Module ID is read from address zero in the buffer; data is read from or written to sequential addresses. When an IOBC is sending either a message or a reply, the Buffer Control is also responsible for signalling the end of the data to be sent.

The Buffer Control's second function is buffer allocation and access control.

5.3 BUFFER OWNERSHIP

Before a requestor uses the buffer, it must request ownership of the buffer from the Buffer Control. The Buffer Control receives four Access Request lines, and drives four Access Granted wires (one per requestor). Once a requestor owns the buffer, it writes something into the buffer, and then perhaps loans it to another requestor for processing.

For the microprocessor to initiate an I/O bus transaction, it must first request ownership of the buffer from the Buffer Control. As shown in Figure 5-3, once ownership is granted, the microprocessor writes the target ModuleID and the message to be sent into the buffer, and then issues a Start command to one of the two IOBCs, thereby loaning the buffer to that IOBC. When the I/O bus transaction is complete, the IOBC returns the buffer to the microprocessor, which reads the reply before releasing the buffer. The buffer is then available to any other requestor asking for ownership.

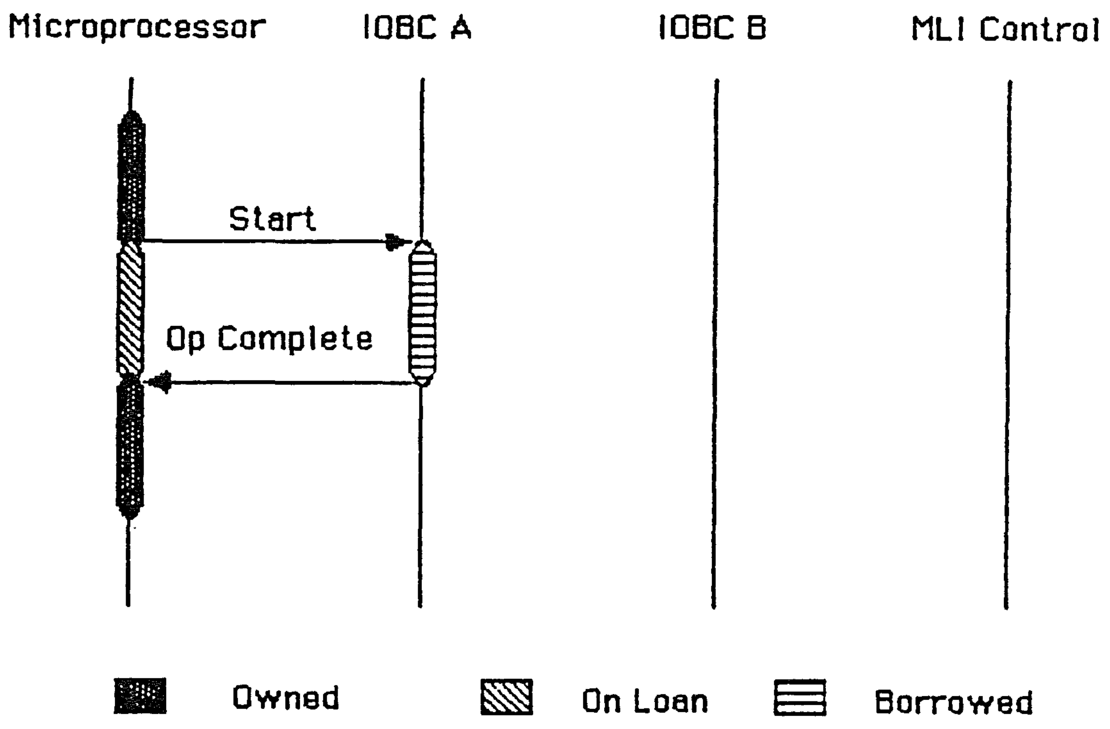

Figure 5-3 Microprocessor Initiate Timeline

5.3 BUFFER OWNERSHIP (Continued

If the IOBC is asked to act as a responder by another module, it first requests ownership of the buffer. Once ownership has been granted, the IOBC signals the initiator that it is ready to receive the message. When the message is received (see Figure 5-4), the IOBC interrupts the microprocessor, thereby loaning it the buffer. The microprocessor reads the message, and decides whether to reply or not. Either a Continue or a Disconnect command returns the buffer to the IOBC, which finally (after sending the reply) releases its ownership of the buffer.

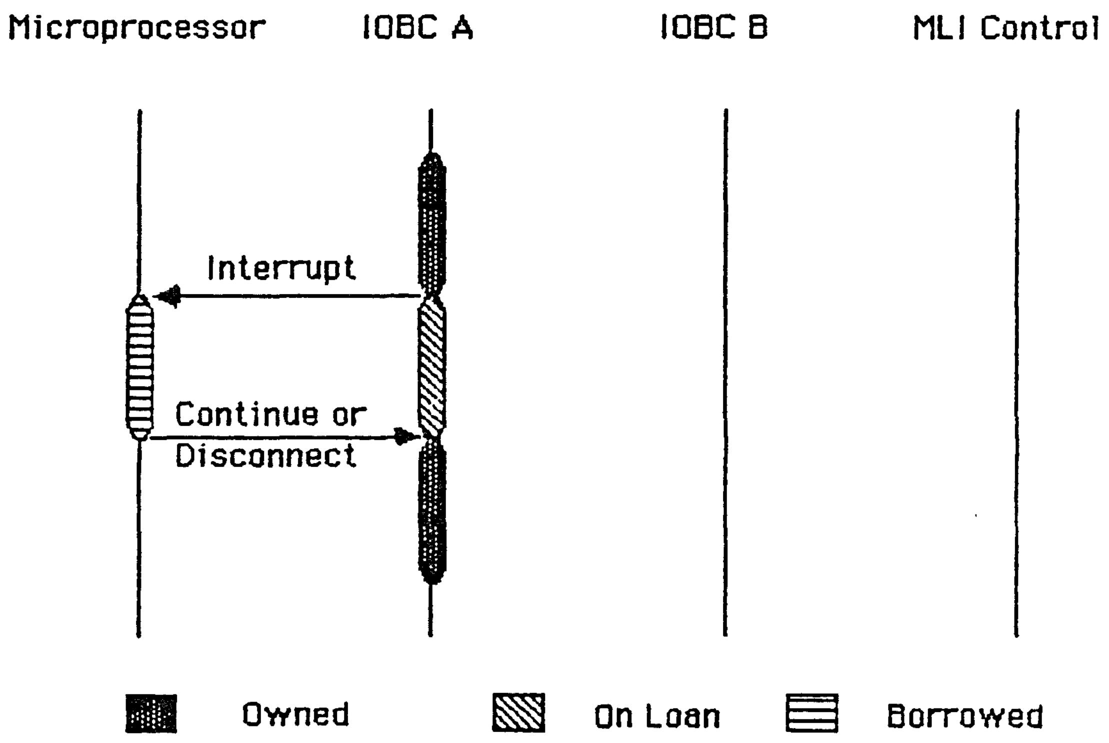

Figure 5-4 IOBC Initiate Timeline

5.3 BUFFER OWNERSHIP (Continued

Use of the buffer by the MLI control in the MLI-DTM is more complex. As shown in Figure 5-5, a DLP does a Poll Request, needing data for a disk write. At the first indication of the Poll Request, the MLI control requests ownership of the buffer. Once ownership has been granted, the Poll Request is allowed to proceed. The MLI Control receives the descriptor links (identifying the I/O) from the DLP, and stores them into the buffer. Meanwhile, the microprocessor has been interrupted, and is polling a status register. The MLI control indicates to the microprocessor that the descriptor links are in the buffer via the status register; this action loans the buffer to the microprocessor.

The microprocessor uses the descriptor links, and other information in the status register, to determine that data must be read from memory. It formats the appropriate I/O bus message for the IOMC in the buffer, and issues a Start command to one of the IOBCs, thereby loaning the buffer to that IOBC.

The IOBC sends the memory read message, and receives the required data. The IOBC indicates that the reply has been received, and returns the buffer to the microprocessor. The microprocessor performs any required setup, and then issues a Start MLI command to the MLI control, thereby returning the buffer to the MLI control. Eventually, the MLI control releases ownership of the buffer.

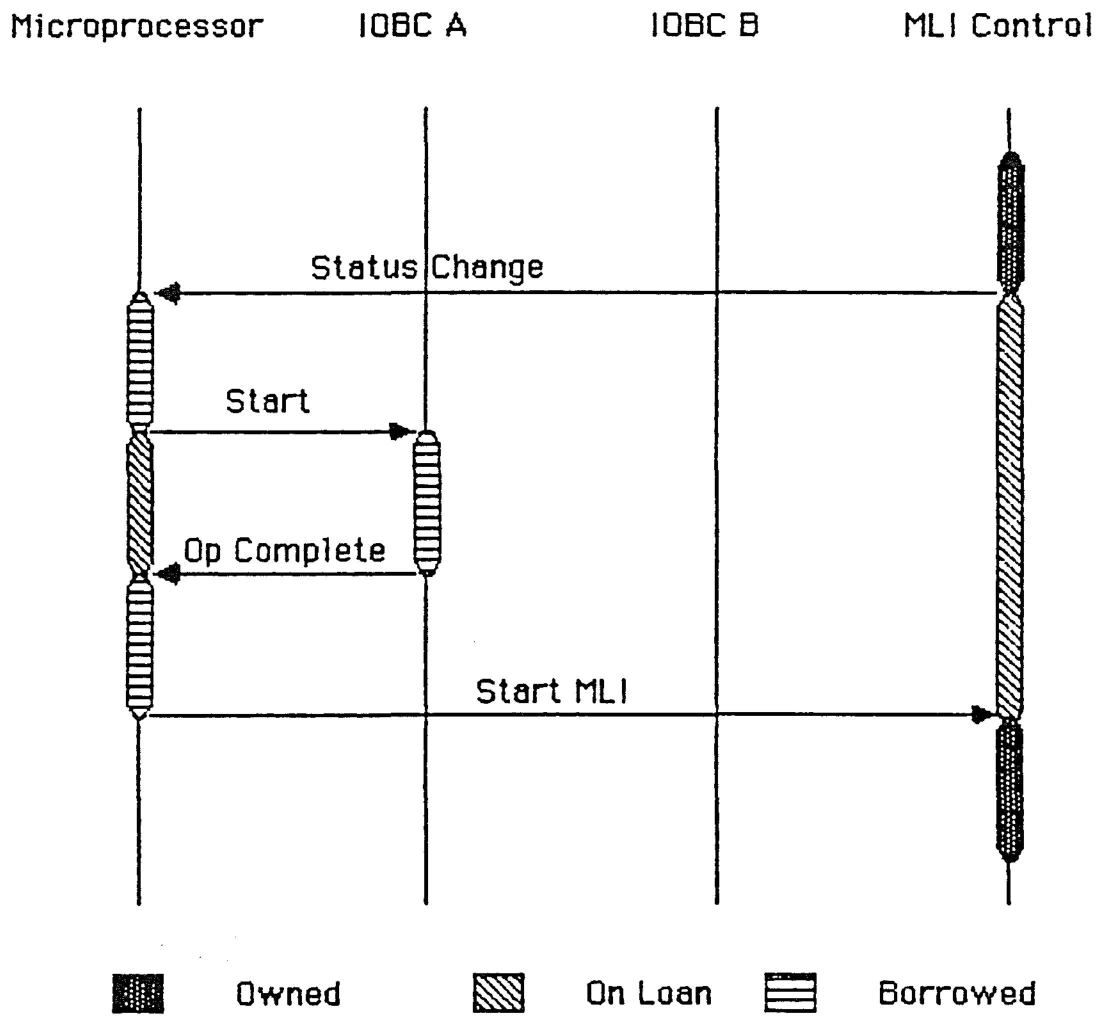

Figure 5-5 Poll Request Timeline

This logic assures that the buffer is owned by only one requestor at a time. Any other requestor that wants to use the buffer must wait until the current owner releases the buffer. While a requestor owns the buffer, it may be loaned to other requestors for processing.

5.4 BUFFER CONTENTION

Simply waiting for ownership handles many (but not all) of the buffer contention problems. In particular, it is used to handle the following cases:

One IOBC (B, for example) may be a responder at a time when the other (IOBC A) is busy as an initiator (microprocessor owns the buffer) or a responder (IOBC A owns the buffer). In this case, IOBC B waits for ownership of the buffer.

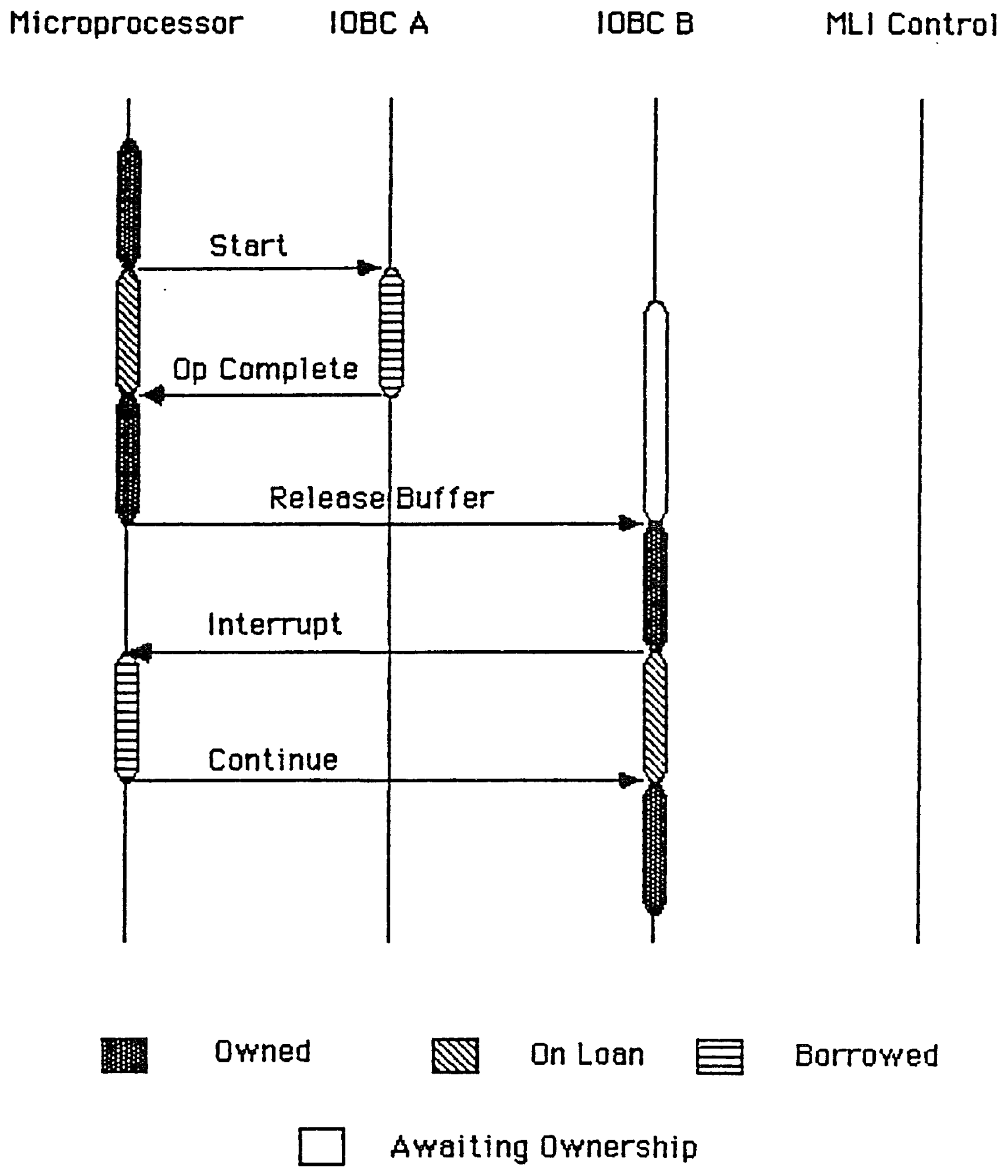

Figure 5-6 IOBC Wait Timeline

5.4 BUFFER CONTENTION (Continued)

Similarly, the MLI Control on an MLI-DTM may receive a Poll Request from a DLP while the buffer is being used by the microprocessor or an IOBC. The Poll Request must wait for ownership of the buffer. The buffer could have been in use for several reasons; the microprocessor could be formatting a message to be sent on an I/O Bus, an IOBC could be receiving a message, or the microprocessor could be preparing to initiate a message to a DLP. This last operation is a Poll Test.

The Poll Test can be initiated despite the fact that a Poll Request is pending. When the Poll Test is complete (see Figure 5-7), the microprocessor releases ownership of the buffer; the MLI Control then obtains ownership, and the Poll Request is allowed to continue.

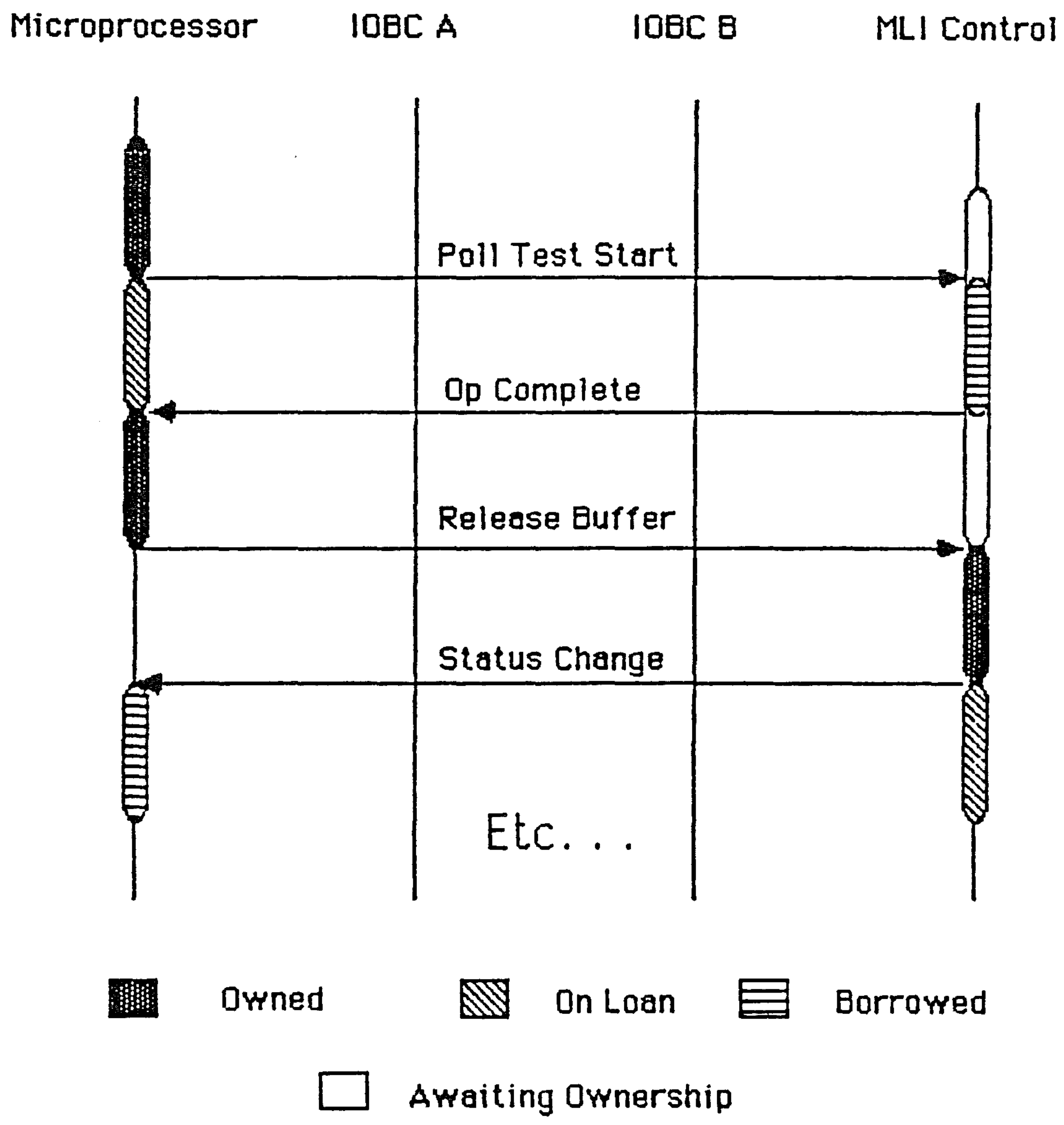

Figure 5-7 Poll Request/Poll Test Timeline

5.4 BUFFER CONTENTION (Continued)

In a sense, the microprocessor also waits for ownership of the buffer. It does so by writing a bit in the Buffer Control's command register, and then continuously polling the Buffer Control's status register, which indicates the requestor that owns the buffer. If the buffer was not owned at the time the microprocessor made its request, the first poll will show that the microprocessor now owns the buffer. However, if the buffer was owned by another requestor when the microprocessor set the bit in the command register, then the microprocessor loops, waiting to be given ownership.

The problem here is that if some other requestor owns the buffer, it is sure to require attention from the microprocessor before it releases ownership. Somehow, we must get the microprocessor out of the poll loop and get it to provide the appropriate attention to the other requestor, without losing track of the request that the microprocessor was originally trying to make.

This problem can be cleanly resolved using the microprocessor's interrupt facility. Whenever another requestor requires the microprocessor's attention, it causes an interrupt. As shown in Figure 5-8, when an IOBC has put something into the buffer, it causes an interrupt. This causes the microprocessor to store a return address that points at an instruction in the poll loop it was executing, and then branch to the appropriate interrupt handler. The code in the interrupt handler looks at the message in the buffer, possibly formats a reply, and issues a Continue or a Disconnect command to the relevant IOBC. The microprocessor then returns to the poll loop from the interrupt handler. When the IOBC finishes sending the reply, it releases the buffer. The microprocessor gains ownership, and falls out of the poll loop.

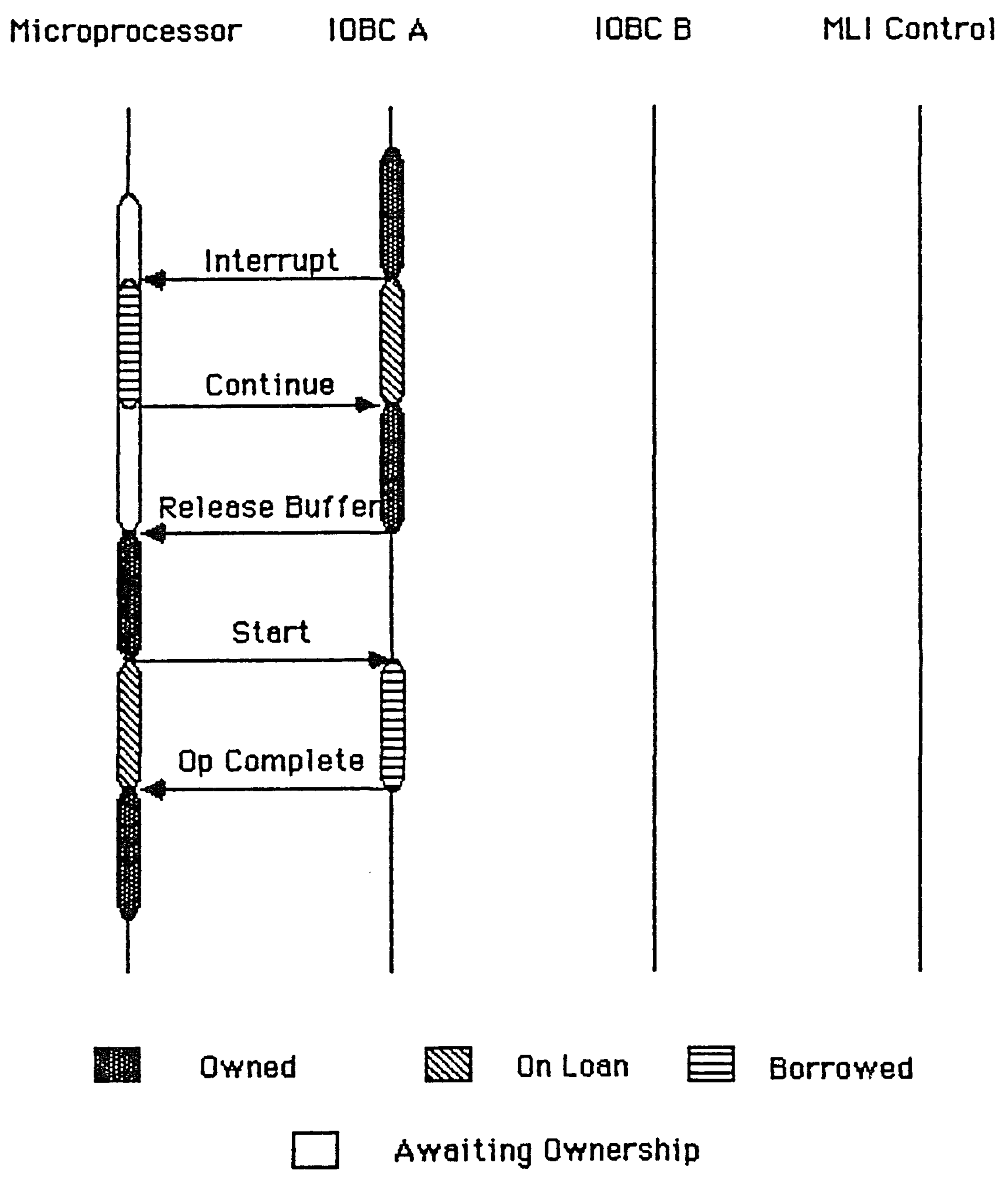

Figure 5-8 Microprocessor Wait Timeline

5.4 BUFFER CONTENTION (Continued)

The interface is different for the MLI Control, but the idea is essentially the same. The interrupt occurs after buffer ownership has been granted to the MLI Control, but before the transfer into the buffer is complete. The microprocessor must poll the MLI Control's status register until the value there indicates that it may access the buffer.

Unfortunately, we cannot always simply wait for buffer ownership. Consider, for example, the case where the microprocessor owns the buffer, and is formatting a message to be sent on an I/O bus. While the message is being formatted, requests come in on both I/O busses for the module to be a responder. We are now deadlocked; the message the microprocessor is formatting cannot go out because both IOBCs are busy, and the incoming messages cannot be put in the buffer because it is owned by the microprocessor.

We have implemented two different solutions to this dilemma; they correspond precisely to 'Module Busy' and 'Non-Busy Message Initiation' previously described.

The mechanism implements Non-Busy Message Initiation is called PreEmpt (see Figure 5-9). It is used in situations in which the microprocessor owns the buffer, and wants to send a message on an I/O bus. The incoming message is given priority over the message the microprocessor is formatting. It preempts the buffer. The actual mechanism is as follows:

The IOBC that is to be a responder requests buffer ownership from the Buffer Control. Since the microprocessor already owns the buffer, the IOBC waits. Eventually, the microprocessor issues a Start command to one of the IOBCs. That IOBC reports immediately that the operation did not complete successfully, and sets a bit in its status register indicating that another requestor wishes to preempt the buffer. The microprocessor must then release the buffer and try again. The responding IOBC then gains ownership of the buffer, indicates to the initiator that it is ready to receive a message, and so forth.

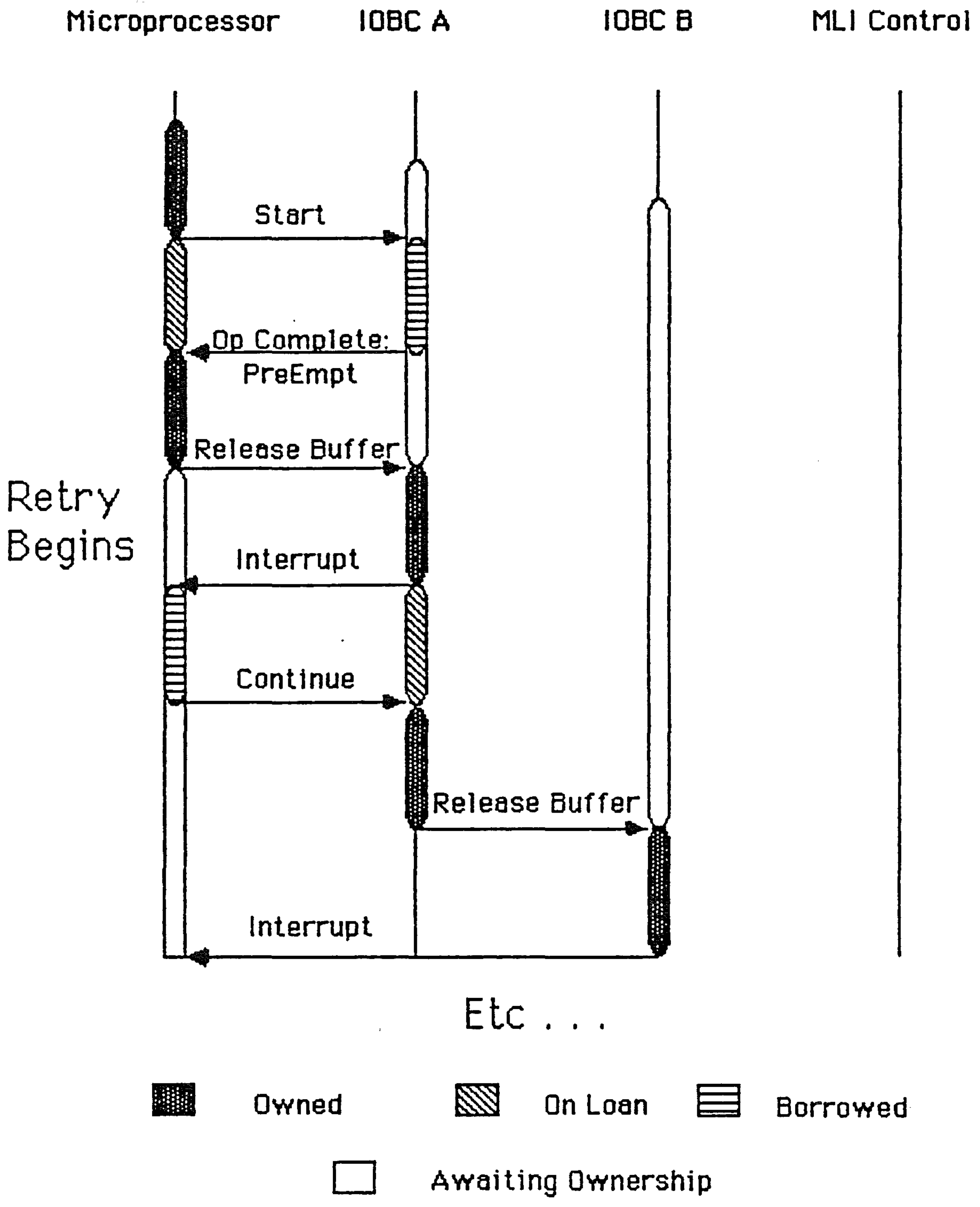

Figure 5-9 Preempt Timeline

5.4 BUFFER CONTENTION (Continued)

Either IOBC can report that any other responder wants access to the buffer. Thus, if the microprocessor issues the Start command to IOBC A, it may receive a Preempt result because IOBC A, IOBC B, or the MLI Control is requesting ownership of the buffer.

Preempt rests on the premise that there is only a small amount of data in the buffer, and that it may be copied or recreated easily after the preempting message has been handled. For data transfers to or from DLPs this is not the case; data transfers tend to be large, and the data is not easily reconstructed. In theory, the microprocessor could copy all of the data out of the buffer into its main memory before releasing the buffer because of a Preempt. This, however, would be time-consuming and inefficient. Instead, the microprocessor sets the Module Busy flag.

At the first sign of a Poll Request from a DLP, the MLI Control requests ownership of the buffer. Once ownership is obtained, the microprocessor is interrupted. The microprocessor then sets the Module Busy flag. By this time, either IOBC could be waiting for ownership of the buffer. As shown in Figure 5-10, the Module Busy flag causes the IOBC to signal the waiting initiator that the module is busy, and break the connection. Any further attempts to send messages to the module are similarly refused.

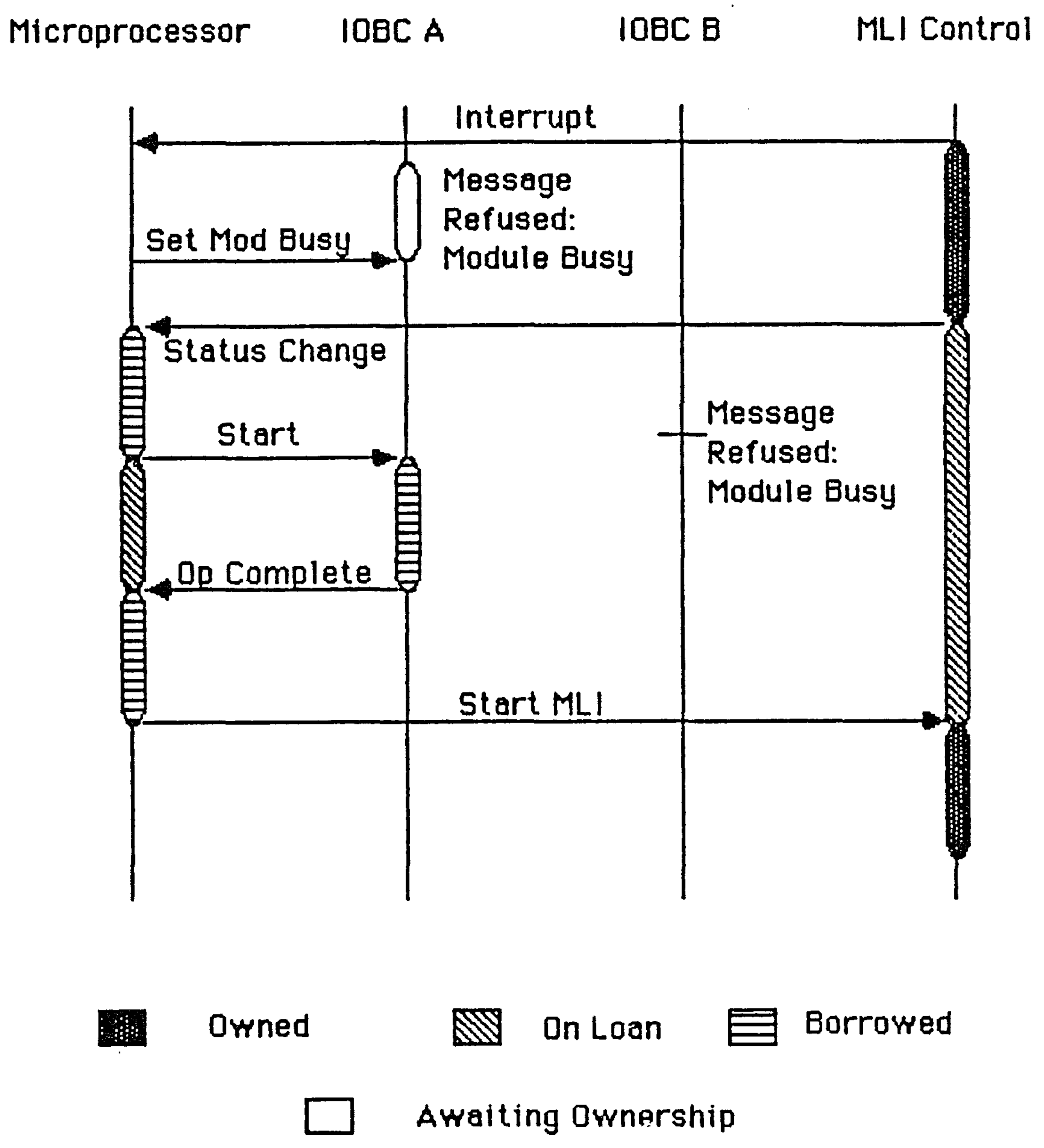

Figure 5-10 Module Busy Timeline

5.4 BUFFER CONTENTION (Continued)

A similar plan is used for a Poll Test when the microprocessor wants to initiate a message to a DLP. The microprocessor first obtains ownership of the buffer, and then sets the Module Busy flag. This effectively refuses any incoming messages on either I/O Bus.

The microprocessor may, at its discretion, use the Module Busy flag in other situations. Two points must be observed, however. First, the buffer must be owned by a requestor before the Module Busy flag is set. This covers several timing holes. Second, the module must adhere to the Deadlock Prevention Protocol previously described.

One final issue regarding buffer contention requires discussion. The ability of the IOBC to handle messages without replies is an optimization designed to improve performance. When no reply is required, we would like to free the bus for other traffic as soon as possible. In particular, we would like to free the bus before the message has been copied out of the buffer into the microprocessor's main memory. Until that copy is complete, we must not accept another incoming message. However, the IOBC releases ownership of the buffer as soon as it disconnects from the bus.

To circumvent this problem, the Buffer Control accepts a command (Ownership Hold) that causes buffer ownership to be denied from all modules when it is released by the IOBC. This prevents the corruption of the buffer by the IOBC or ML1, and allows access to the microprocessor only. The microprocessor should issue this command before it issues The Disconnect (see Figure 5-11). When the microprocessor has finished with the buffer, it issues the ownership release command.

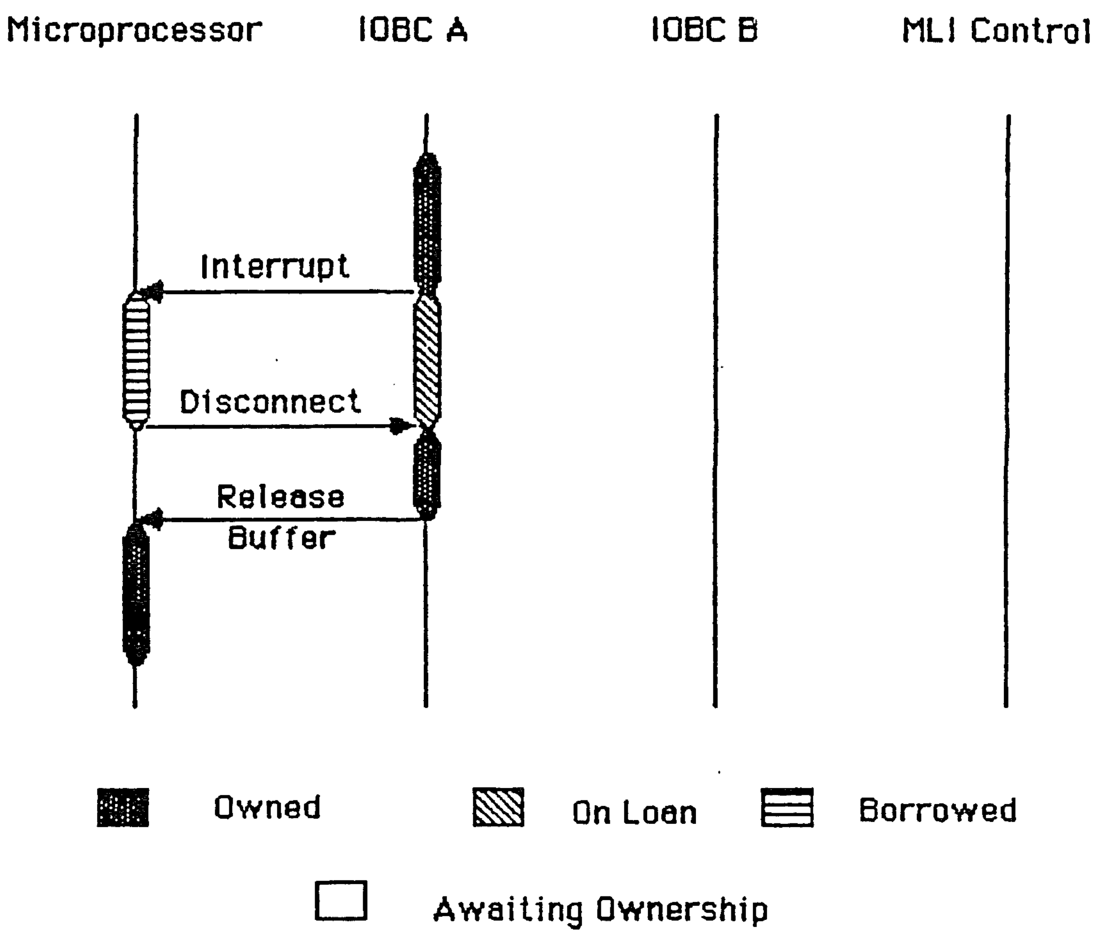

Figure 5-11 Disconnect Timeline

In the future, we may choose to harden the distinction between messages that have replies and those that do not. The hardware in the IOBC would then examine the op code in the incoming message, and issue the Ownership Hold and Disconnect commands directly. This would free the bus almost immediately, rather than after microprocessor had responded to an interrupt.

6 DETAILED DESCRIPTION

6.1 PROGRAMMER'S MODEL

The programmer writing code for the microprocessor sees only the memory and the interrupt interfaces. Everything else must be made to look like memory or an interrupt. In MLI-DTM implementation of the I/O bus interface, two areas of memory are in fact used for other things. The first is the buffer; it looks to the microprocessor like 1024 32 bit words of memory. The second is the register area; there are sixteen 16 bit registers that the microprocessor may read or write as if they were memory locations. Control and monitoring of the IOBCs, the Buffer Control, and MLI Control (in the MLI-DTM) are done by reading and writing the memory locations in the register area.

This section is divided into five subsections; they are:

1 the buffer

2 the registers

3 the interrupts

4 basic protocol

5 error handling

6.1.1 BUFFER

The buffer consists of 1024 words of 32 bits each. In the MLI-DTM, it occupies addresses 00600000 (hex) to 00600FFF (hex). The microprocessor may only manipulate entire words; partial word operations to the buffer are not supported. Since the addresses quoted above are byte addresses, this implies that the least significant two bits of the address must be zero. In other words, the first location in the buffer is address 00600000 (hex), the second is 00600004 (hex), the third is 00600008 (hex), and so forth.

Note that the addresses and insistence on whole word operations are implementation dependent; they may differ for the SMC or future DTMs.

When the microprocessor wants to initiate a message, the address of the target responder must be written into the first buffer location. The responder's address must be encoded by the microprocessor as follows:

Address	Target Responder
000000F0	*IOMC
000000B7	*SMC
00000081	DTM Slot #1, Cabinet #0
00000012	DTM Slot #2, Cabinet #0
00000063	DTM Slot #3, Cabinet #0
00000024	DTM Slot #4, Cabinet #0
00000055	DTM Slot #5, Cabinet #0
000000C6	DTM Slot #6, Cabinet #0
00000039	DTM Slot #1, Cabinet #1
000000AA	DTM Slot #2, Cabinet #1
000000DB	DTM Slot #3, Cabinet #1
0000009C	DTM Slot #4, Cabinet #1
000000ED	DTM Slot #5, Cabinet #1
0000007E	DTM Slot #6, Cabinet #1

* Although there are (possibly) two IOMCs and two SMCs in a system, at most one is connected to each I/O Bus. We have chosen to give both IOMCs (and both SMCs) the same address.

6.1.1 BUFFER (Continued)

This encoding allows the responder's hardware to correct single bit errors and detect double bit errors in this module ID without using the ECCIO chip (which may well be in use for other things at the time).

If the microprocessor wants to act as an initiator, the message to be sent should be placed in the buffer immediately following the target's module ID. The IOBC puts the reply (if any) in the buffer immediately following the original message.

Message and Reply formats are specified in Section 6.8 (BUS PROTOCOL).

When a message is received by an IOBC, it is placed in the buffer starting at the first location. If the microprocessor wants to send a reply, that reply should be placed in the buffer immediately following the original message.

The microprocessor must not read or write the buffer when it is possible that another requestor is using it; data corruption will result. If the microprocessor is to be the initiator, it must first request and gain ownership of the buffer. Once ownership is obtained, the microprocessor is free to write (or read) the buffer. When the microprocessor issues the Start command to an IOBC, the buffer is loaned to that IOBC. The microprocessor may not access the buffer until an Op Complete status has been reported, or the microprocessor resets the IOBC.

When an IOBC receives a message, it places it in the buffer (which it owns), then causes an interrupt. The interrupt implicitly loans the buffer to the microprocessor. Thus, after an IOBC interrupt caused by a received message (see Interrupts, following) the microprocessor may read and write the buffer.

If the microprocessor chooses to send a reply, it writes the reply in the buffer, and then issues the Continue command. This command returns the buffer to the IOBC; the microprocessor must not access the buffer after issuing this command.

6.1.1 BUFFER (Continued)

Alternatively, the microprocessor might choose not to send a reply. In this case, the Disconnect command instructs the IOBC to release the I/O bus and ownership of the buffer. If the microprocessor wants to access the buffer after issuing the Disconnect (for example, to copy the message to the microprocessor's main memory), then it must issue the Ownership Hold command to the Buffer Control before issuing the Disconnect to the IOBC. When the microprocessor has finished with the buffer, it must then release ownership by issuing the Release Ownership command to the Buffer Control.

6.1.2 REGISTERS

The second area of microprocessor memory that is used for other things is the register area. In the MLI-DTM, there are sixteen 16 bit registers that control and monitor the two IOBCs, the Buffer Control, and the MLI Control. To the microprocessor, these registers appear to be a 32 byte area of memory. The following table lists the 16 registers by name, for each MLI-DTM address, and a brief description. Some registers are read only or write only; this is indicated by a W or R in the Type column.

Name	Address (hex)	Type	Used By	Description
BUFFCMND	430000	W	Buffer Cont.	Command Register
BUFFSTAT	430002	R	Buffer Cont.	Status Register
	430004	R	MLI Cont.	Result Register
	430006	W	MLI Cont.	Command Register
	430008	R	MLI Cont.	Status Register
	43000A	R	Configuration.	ID register
	43000C	R/W	MLI Cont.	Byte counter (upper)
	43000E	R/W	Mli Cont.	Byte counter (lower)
	430010		Reserved	
	430012		Reserved	
BUFFADDR	430014	R/W	Buffer Cont.	Buffer Start Address
BUFFLEN	430016	R/W	Buffer Cont.	Message/Reply Length
ABUSSTAT	430018	R	IOBC A	Status Register
BBUSSTAT	43001A	R	IOBC B	Status Register
ABUSCMND	43001C	W	IOBC A	Command Register
BBUSCMND	43001E	W	IOBC B	Command Register

Note that both the address range (hex 0043xxxx) and the irividual addresses within that range are completely implementation dependent; they may differ in the SMC and future DTMs.

Each register is divided into many fields. A description of the registers and fields follows.

6.1.2.1 BUFFCMND

This register issues commands to the Buffer Control. These include commands relating to buffer ownership, Module Busy, fault reporting, and fault detection testing. Two bits in this register are also used to acknowledge interrupts from the two IOBCs (there were not enough bits in their respective command registers).

The following table lists all the Buffer Control commends. The first column names each command, and the second lists the value that must be written into the BUFFCMND register to issue that command.

Command Name	Write Value (hex)
Request Buffer Ownership	0001
Release Buffer Ownership	0010
Set Module Busy	0100
Reset Module Busy	0080
Ownership Hold Command	0002
Interrupt Acknowledge, IOBC A	0800
Interrupt Acknowledge, IOBC B	8000
Reset Error Latches	0400
Test Fault Detector #1	0048
Test Fault Detector #2	0044
Test Fault Detector MLI	1040
Die	0200
Test Die	0240

REQUEST BUFFER OWNERSHIP COMMAND

This command is issued when the microprocessor wants to own the buffer, in order to act as an initiator. After issuing the command, the microprocessor must poll the BUFFSTAT register until it is granted ownership by the Buffer Control.

6.1.2.1 BUFFCMND (Continued)

RELEASE BUFFER OWNERSHIP COMMAND

This command indicates that the microprocessor has finished using the buffer, and that ownership may be granted to any other requestor.

SET MODULE BUSY COMMAND

This command sets the Module Busy flag. If any other module tries to send a message to this module while the Module Busy flag is set, it will be refused with a Module Busy indication.

RESET MODULE BUSY COMMAND

This command resets the Module Busy flag.

OWNERSHIP HOLD COMMAND

This command prevents buffer ownership from being granted to a new requestor after it has been released by the current owner. It is used with the Disconnect IOBC command, when the microprocessor wants to release the I/O Bus before it releases the buffer. This command must be followed by a Release Buffer Ownership command.

The Release Buffer Ownership command that follows an Ownership Hold command does not negate the effect of a prior Request Buffer Ownership command. This allows the Ownership Hold and Release Buffer Ownership commands to be used in an interrupt handler without disturbing an underlying Initiate process that has Requested Buffer Ownership, but has not yet been granted it. See Section 6.3 (BASIC PROTOCOL) for details.

6.1.2.1 BUFFCMND (Continued)

INTERRUPT ACKNOWLEDGE COMMAND, IOBC A

This command should be issued in the interrupt handler for interrupts from IOBC A, after the ABUSSTAT register has been read to ascertain the cause of the interrupt. Normally, this command causes the interrupt line to the microprocessor to be dropped. However, if another interrupt condition has arisen before this command is issued, the interrupt line remains asserted, and the ABUSSTAT register is changed to reflect the new interrupt condition.

INTERRUPT ACKNOWLEDGE COMMAND, IOBC B

This is the Interrupt Acknowledge command for IOBC B. See IOBC A description for details.

RESET ERROR LATCHES COMMAND

During the Responder Selection phase of an I/O Bus Transaction, the initiator broadcasts the encoded Module ID of the target responder on the data lines of the I/O Bus. All modules on the bus check if the initiator wants to talk to them. In the process, they also check for single and double bit errors in the module number. The presence of such an error is latched in the BUFFSTAT register. Every module on the bus performs this check.

From the microprocessor's point of view, the bits in the BUFFSTAT register may be set at any time: the microprocessor may not ever be apprised of the I/O Bus operation that encountered the error. Hence, the bits in the BUFFSTAT register should be polled on a regular basis. If an error has been latched, the microprocessor should record the fact and reset the latches using this command.

6.1.2.1 BUFFCMND (Continued)

TEST FAULT DETECTOR 1 COMMAND

The Buffer Control contains two fault detectors. This
The Buffer Control contains two fault detectors. This
command forces Fault Detector 1 to report an error to the SMC via the ModBroken signal, while at the same time asserting ModNotBroken.

The microprocessor should own the buffer when this command is issued. See Request Buffer Ownership and Release Buffer Ownership commands.

TEST FAULT DETECTOR 2 COMMAND

This command tests fault detector number 2.

TEST MLI FAULT DETECTOR COMMAND

This command tests the fault detector on the MLI.

DIE COMMAND

This command causes the ModBroken signal to be asserted to the SMC. It is used when the microprocessor discovers that something is drastically wrong, and cannot continue normal processing.

6.1.2.1 BUFFCMND (Continued)

TEST DIE COMMAND

This command tests the Die signal,

The following table presents the commands above in a slightly different way. Each bit, or group of bits in the BUFFCMND register is described separately.

Bit(s)	Field	Function
15	IntAckB	Interrupt acknowledge to IOBC B
14-13	not used	Should always be 00
12	FrcMliErr(0)	Used for fault detection testing on the MLI.
11	IntAckA	Interrupt acknowledge to IOBC A
10	uPNoCheckRst	Resets latches that hold single and double bit error information during target module number broadcast.
9	Dead	Causes ModBroken to be signalled to the SMC.
8	BusySet	Sets Module Busy.
7	BusyReset	Resets Module Busy.
6	IgnoreErr	Used for fault detection testing.
5	not used	
4	ReleaseBuffer	Buffer ownership is released.
3-2	CForceErr(1:0)	Used for fault detection testing.
1	OwnHold	Ownership Hold command.
0	RequestBuffer	Request Buffer Ownership.

6.1.2.2 BUFFSTAT

This read only register contains fields that indicate which requestor (if any) owns the buffer, whether the module is on-line or off-line, the Module's ID, and any single or double bit errors that occur in module numbers broadcast during the Responder Selection phase of an I/O Bus transaction.

BUFFSTAT is in the following format:

Bit(s)	Name	Active	Description
15	OffLineH	H	module is offline.
14-12		X	not used; value unspecified.
11	BNoCheckDBE	H	DBE in module number, bus B
10	BNoCheckSBE	H	SBE in module number, bus B
9	ANoCheckDBE	H	DBE in module number, bus A
8	ANoCheckSBE	H	SBE in module number, bus A
7-4	ModuleIdL(3:0)	L	Module ID associated with this backplane slot.
3	DAccGrntL	L	MLI owns the buffer
2	CAccGrntL	L	Microprocessor owns the buffer
1	BAccGrntL	L	IOBC B owns the buffer
0	AAccGrntL	L	IOBC A owns the buffer

SBE stands for single bit error; DBE stands for double bit error.

The Active column indicates whether the corresponding register bit is active high (H), active low (L), or unspecified (X). For example, if the BUFFSTAT register contained the hex value 70EE, then the module is online, no SBEs or DBEs have been latched, the module ID is 1, and IOBC A owns the buffer.

OFFLINE

Bit 15, when active, indicates that the module has been set offline by the SMC. The SMC does this by setting a bit in the card's maintenance chain. When the module is offline, it is unable to drive the backplane, or respond to incoming messages.

6.1.2.2 BUFFSTAT (Continued)

NOT USED
Bits 14:12 are not used. No guarantees are made about the values these bits will assume; they should be ignored.

ERROR LATCHES
Bits 11:8 latch SBEs and DBEs that occur in the broadcast module ID during the Responder Selection phase of an I/O Bus transaction. These bits can be reset by issuing the Reset Error Latches command.

MODULE ID
The ModuleIdL (3:0) field enables the microprocessor on a DTM to determine which backplane slot it is plugged into. The most significant bit (ModuleIdL(3)) indicates which cabinet the card is in, while the three least significant bits indicate the slot number (1 through 6) within the cabinet.

When the module is offline, the ModuleIdL field is sourced from a register on the maintenance chain, and therefore may not accurately indicate the slot into which the module is plugged. This was done to allow testing of various circuits in the two IOBCs.

BUFFER OWNERSHIP
Bits 3:0 indicate which requestor (if any) owns the buffer. At most, one of these bits should be active (low) at any one time; if multiple bits are active, then the Buffer Control is broken. Thus, legal values for this field are (in hex):

7	MLI owns the buffer
B	microprocessor owns the buffer
D	IOBC B owns the buffer
E	IOBC A owns the buffer
F	no one owns the buffer

If no one owns the buffer (F), two possibilities exist. Either no one is requesting the buffer, or the buffer was set in 'Hold' by the microprocessor. The microprocessor is responsible for verifying that the buffer has been released after the 'Hold' command. This allows IOBC and MLI to resume their normal operations.

6.1.2.3 BUFFADDR

This register is used by Buffer Control to address the buffer. The address contained in this register is a word address, relative to the beginning of the buffer (remember that the buffer consists of 1024 32 bit words). The address of the first word in the buffer is hex 0000, the second is 0001 hex, and so forth. Since there are only 1024 words in the buffer, the most significant six bits of the address are not used, and must be zero. In other words, the address of the last word in the buffer is 03FF (hex).

BUFFADDR is cleared automatically whenever buffer ownership changes. It is incremented by Buffer Control whenever a word is read from or written to the buffer by either IOBC or the MLI control.

The microprocessor can read and write this register. However, it is not expected that the microprocessor will write BUFFADDR in the course of executing an I/O Bus transaction. The microprocessor must not write BUFFADDR when either IOBC or the MLI Control is using the buffer; if the microprocessor reads BUFFADDR while another requestor is using the buffer, the value read is not specified, and may change at any time during the microprocessor's read cycle. In other words, BUFFADDR is subject to the same access restrictions as the buffer itself.

When the module acts as an initiator, a message is sent and a reply received. After the IOBC has reported Op Complete (see ABUSSTAT), the microprocessor may read BUFFADDR, which contains the address of the first word after the reply in the buffer. If there was no reply, it points to the first word after the original message.

When the module is a responder, the IOBC puts the message into the buffer, and interrupts the microprocessor. The microprocessor then reads BUFFADDR to determine the length of the received message (BUFFADDR points to the location one word after the received message). Since the received message is placed in the buffer starting in the first location, BUFFADDR contains the length (in words) of the message.

6.1.2.3 BUFFADDR (Continued)

The facility to write the BUFFADDR was provided for use with the MLI control, and for testing. Since BUFFADDR is cleared when buffer ownership changes, any value written to BUFFADDR has effect only until buffer ownership is released.

If BUFFADDR is written after the microprocessor gains ownership of the buffer for an initiate, then the Target Module Id and the message to be sent will be read by the IOBC, starting at the specified buffer address. Similarly, if BUFFADDR is written after an interrupt, before issuing a Continue command to the IOBC, then the IOBC will read the reply from the buffer, starting at the specified address.

6.1.2.4 BUFFLEN

BUFFLEN is used by the microprocessor to inform the buffer control of the length of a message that is to be sent on an I/O Bus. When the microprocessor is acting as an initiator, it loads this register with the length of the message (in words), minus one in hex.

For example, to send a message seventeen words long, BUFFLEN is loaded with hex 0010. Note that the length here is the length of the message only; it does not include the Target Module Id word. When the module is a responder sending a reply, BUFFLEN is loaded with the length of the reply minus one before issuing the 'Continue' command to the IOBC.

To send a one word message or reply, load BUFFLEN with 0000 (hex).

This register should not be read or written while an IOBC or the MLI control could be accessing the buffer. BUFFLEN is counted down every time a word is read from the buffer.

6.1.2.5 ABUSCMND AND BBUSCMND

These registers are used by the microprocessor to issue commands to IOBC A and IOBC B, respectively. They differ only in address. The commands issued via these registers relate to sending messages and replies, acknowledging Initiate I/O Requests, clearing the IOBC, and testing the IOBC's fault detectors.

The following table lists all the IOBC commands. The first column names each command; the second lists the value that must be written into the register (ABUSCMND or BBUSCMND) to issue that command.

Command Name	Write Value (hex)
Start Normal	0001
Start Emergency	0081
Continue	0002
Disconnect	0004
Reset Master	0009
Reset Listener	000A
IIOAck	0040
Test Fault Detector #1	8100
Test Fault Detector #2	8200
Test Fault Detector #3	8400
Test Fault Detector #4	8800
Test Fault Detector #5	9000
Test Fault Detector #6	A000
Test Fault Detector #7	C000

The commands function as described in the following text:

o START NORMAL. The microprocessor uses this command to indicate to the IOBC that there is a message (and Target Module Id) in the buffer, which the IOBC should attempt to send. The microprocessor should own the buffer at this time. This command implicitly loans the buffer to the IOBC.

6.1.2.5 ABUSCMND AND BBUSCMND (Continued)

- START EMERGENCY. This command is identical to Start Normal, except that during the bus arbitration phase, the IOBC asserts its emergency priority level instead of its normal priority level. Emergency priorities are higher than all normal priorities; thus, if only one module asserts emergency priority, it is sure to win control of the bus.

- CONTINUE. This command is used by the microprocessor to send a reply. After the microprocessor has examined the received message and formatted the reply in the buffer, this command informs the IOBC that the reply is ready for transmission.

- DISCONNECT. This command indicates that the message just received does not need a reply; hence, the IOBC should disconnect from the bus.

  When the IOBC disconnects from the bus, it also releases ownership of the buffer. The other IOBC, or the MLI control, could gain ownership and use it immediately. To prevent this, the microprocessor must issue an Ownership Hold Command to the buffer control before issuing the Disconnect.

- RESET MASTER. This command clears the initiator portion of the IOBC. If it was connected to the I/O Bus at the time, the connection will be severed. It is expected that this command will only be used for error recovery (such as when the microprocessor times out an I/O Bus transaction). See Microprocessor Microcode for further details.

- RESET LISTENER. This command clears the listener portion of the IOBC. If it was connected to the I/O Bus at the time, the connection will be severed. This command is not used.

- IIOACK. This command causes the IOBC to assert the IIOAckL signal to the IOMC. This command should only be used by the Message Router, in response to an IIOReq interrupt. The IOBC handles the backplane timing details specified in Section 6.8 (BUS PROTOCOL).

6.1.2.5 ABUSCMND AND BBUSCMND (Continued)

- TEST FAULT DETECTOR. Each IOBC contains seven fault detectors. This command forces the specified fault detector to report an error to the SMC via the 'ModBroken' signal, while at the same time asserting 'ModNotBroken'.

  The microprocessor should own the buffer when this command is issued (see Request Buffer Ownership and Release Buffer Ownership commands).

  The following table presents the commands above in a slightly different way. Each bit, or group of bits in the register is described seperately. In the table, an x at the beginning of the name can be replaced by A or B.

Bits	Name	Function
15	xIgnoreErr	Fault detection testing
14-8	xForceErr(6:0)	Fault detection testing
7	xIEmerg	Use with Start; emergency priority during bus arbitration.
6	xIIOGot	Causes IIOAckl to IOMC.
5-4	not used	Must be zero.
3	xReset	Resets the IOBC. Bits 0 and/or 1 indicate whether the initiator (set bit 0) or responder (set bit 1) or both (set both bits 0 and 1) are to be reset.
2	xDisc	Disconnect - no reply required.
1	xContinue	Continue - send reply in buffer.
0	xStart	Start I/O Bus Comnmand.

6.1.2.6 ABUSSTAT AND BBUSSTAT

These two registers allow the microprocessor to determine the status of the two IOBCs. The microprocessor inquires about the status in order to:

o determine the cause of an interrupt

o after issuing a Start command, determine whether the I/O Bus operation has completed, and if so, with what results.

One field (in fact, a single bit) is devoted to the first case; the rest of the register is devoted to the second.

In the following table, an x at the beginning of a name can be replaced with an A or a B.

The two registers are as follows:

Bit(s)	Field	Active	Description
15	xIIOIntH	H	Interrupt reason.
14		X	not used
13-11	xBidErr(2:0)	L	Bus arbitration errors.
10	xErrLine	H	Error report from IOMC.
9-8	xMBE-xSBE	H	MBE/SBE detected by I/O Bus ECCIO.
7-5	xSelRes(2:0)	H	Responder selection result.
4-2	xBidRes(2:0)	H	Bus arbitration result.
1	xPreEmptL	L	Another requestor wants to PreEmpt the buffer.
0	xOpCompL	L	Operation complete.

6.1.2.6 ABUSSTAT AND BBUSSTAT (Continued)

Notes:

SBE stands for single bit error; MBE stands for multi-bit error.

The column headed 'Active' indicates whether the corresponding register bit is active high (H), active low (L), or unspecified (X). For example, if the ABUSSTAT register contains the hex value 682A, then Operation Complete is asserted.

Bit 15 (xIIOIntH) is independent of the the rest of the register. It indicates the reason for an interrupt from this IOBC. The rest of the register is used by the microprocessor to monitor the IOBC after a Start Command has been issued.

The fields in xBUSSTAT are used as follows:

o INTERRUPT REASON. When the microprocessor is interrupted by an IOBC, bit 15 (xIIOIntH) indicates if the interrupt occurred because a message has been received (xIIOIntH low), or because the IOMC asserted IIOReqL on the backplane. IIOReql indicates that there is something in the mailbox register for the message router (xIIOIntH high). All modules except the message router ignore the xIIOIntH high interrupts.

  The value of bit 15 does not change after an interrupt until the microprocessor issues an Interrupt Acknowledge command for the appropriate IOBC via the BUFFCMND register.

  If the microprocessor is interrupted because a message has been received (IIOIntH low), then it examines xSBE and xMBE to determine if the data arrived intact.

6.1.2.6 ABUSSTAT AND BBUSSTAT (Continued)

o OPERATION COMPLETE. Bit 0 (xOpCompL) indicates to the microprocessor that the I/O Bus transaction requested by the Start Command has completed. Other bits in the register indicate whether the operation was successful, or why it failed.

Bit 0 also plays a significant role in synchronizing the IOBC with the microprocessor. After the microprocessor issues a Start command, various values appear in the register as the I/O Bus transaction progresses. Since the microprocessor and the IOBC run with different clocks, the register might be changing while it is being read. For this reason, the value read from the register cannot be trusted unless the IOBC has stopped changing it. This is indicated by xOpCompL being low.

Bit 0 is set high by the Start command, and remains high until the I/O Bus transaction is complete. The microprocessor polls the xBUSSTAT register regularly after issuing the Start command, testing only the xOpCompL bit. Once the xOpCompL bit has been tested and found low, the microprocessor must read xBUSSTAT again to ensure that the values of all bits in the register are consistent.

When the I/O Bus operation times out, the microprocessor might try to interpret bits in the xBusStat register while xOpCompL is high. If the microprocessor issues a Start Command, and Op Complete is not asserted within a reasonable time, then something is broken; perhaps the IOBC, the responder, or the backplane. Information in the xBUSSTAT register can be used to help isolate the faulty module.

If the I/O Bus operation times out, the microprocessor reads the xBUSSTAT register twice, and compares the two values read. If they are equal, diagnosis can be attempted. Details of timeout values and fault handling procedures appear in Section 7 (ERROR DETECTION AND HANDLING).

6.1.2.6 ABUSSTAT AND BBUSSTAT (Continued)

c BUS ARBITRATION RESULT. Once Op Complete has been asserted, the microprocessor examines the xBidRes(2:0) field. Possible values for these three bits are as follows:

xBidRes(2:0)	Description
100 or 111	BUSBUSY
001	PreEmpt or Arbitration Error
010	IOBC obtained the bus

All other values indicate that the IOBC is broken.

If xBidRes(2:0) = 100 (binary), then a BUS BUSY condition has occurred (the I/O Bus was already in use by another module). The microprocessor retries the I/O Bus operation.

If xBidRes(2:0) = 001 (binary), there are three possibilities:

a. This, and some other module bid for the bus at the same time; the other module had higher priority.

b. A PreEmpt occurred.

c. A hardware failure was detected.

The microprocessor examines xPreEmptL and xBidErr(2:0) to determine how to proceed. If PreEmptL is inactive (high), and xBidErr(2:0) = 111 (binary), then case (a) occurred, in which the other module had higher priority. In this case, the microprocessor should retry the I/O Bus operation.

If xBidRes(2:0) = 010 (binary), then the bus arbitration phase completed successfully, and this module has access to the bus. The microprocessor examines xSelRes(2:0) to find out how the remaining phases of the I/O Bus transaction completed.

6.1.2.6 ABUSSTAT AND BBUSSTAT (Continued)

- PREEMPT. Bit 1 (xPreEmptL) is valid only when xOpComplL is active (low), and when xBidRes(2:0) = 001 (binary). In those circumstances, xPreEmptL will be low if a PreEmpt condition has arisen. A PreEmpt occurs if either IOBC, or the MLI control discovers that it needs the buffer to accept an incoming message between the time the microprocessor is granted ownership of the buffer, and the time it issues the Start command.

  When the microprocessor discovers a PreEmpt, it releases ownership of the buffer. The PreEmpting requestor then gains ownership of the buffer, puts the message in the buffer, and causes an interrupt. The microprocessor may retry the original operation after the PreEmpting message has been handled.

- BUS ARBITRATION ERRORS. Bits 13-11 (xBidErr(2:0)) are valid only when xOpComplL is active (low), and when xBidRes(2:0) = 001 (binary). In those circumstances, any of the three bits in this field being active (low) indicates that the bus arbitration hardware for this I/O Bus failed. The IOBC in question should not be used again until some sort of testing has been carried out.

  The meanings of the individual bits are as follows:

xBidErr(0)
The priority comparator asserted Equal and Less simultaneously, and is therefore broken, or the module requested normal priority, but was granted emergency priority (or vice versa).

xBidErr(1)
The module asserted its priority, but bus arbitration logic reported no requests.

6.1.2.6 ABUSSTAT AND BBUSSTAT (Continued)

xBidErr(2)
The module asserted its priority, but bus arbitration logic reported only requests of lower priority.

If any of these three bits is active (low), then the hardware is broken. In all three cases, the IOBC is the primary suspect, probably causing 90% or more of the failures of this type. The backplane is the major secondary suspect; it is extemely unlikely that another module connected to the I/O Bus could cause these failures.

c RESPONDER SELECTION RESULT.

Once Op Complete has been asserted, and xBidRes(2:0) = 010 (binary), then the microprocessor examines the xSelRes(2:0) field. Possible values for these three bits are as follows:

xSelRes(2:0)	Description
010	Module not present
101	Illegal disconnect
110	Module busy
001	Normal termination

All other values indicate that the IOBC is broken.

Module Not Present indicates that the target responder is either physically not present, or is offline. A module may go offline for a number of reasons:

The SMC may put the module offline by shifting the appropriate value into the maintenance chain.

A fault detector may detect a failure and take the module offline.

Stop logic set up in the module may take the module offline.

In all three cases, the SMC is aware of the actual status of the module.

6.1.2.6 ABUSSTAT AND BBUSSTAT (Continued)

An Illegal Disconnect occurs when the target responder goes offline in the middle of the I/O Bus transaction. Module Not Present, in contrast, occurs when the module is offline before the transaction begins.

If either an Illegal Disconnect or a Module Not Present occurs to a responder that was previously answering messages, then simply retrying the operation is inappropriate. Some kind of extended recovery, probably coordinated by the SMC, is required.

Module Busy indicates that the target responder is unable to accept a message at this time. The operation is retried.

Normal Termination indicates that a message was sent, and a reply (perhaps) received without a control type mishap. It does not indicate that the data was received unblemished. The error bits (xErrLine, xSBE, and xMBE) must be examined.

o xSBE AND xMBE. These two bits indicate whether this module's ECCIO chip detected any single bit errors (xSBE) or multi-bit errors (xMBE) during the I/O Bus transaction. Single bit errors are corrected automatically. Multi-bit errors cannot be corrected, although retrying the operation may allow recovery. Possible values for these three bits are as follows:

xMBE,xSBE	Description
00	No error
01	Single bit error (send)
10	Single bit error (receive)
11	Multiple bit error

6.1.2.6 ABUSSTAT AND BEUSSTAT (Continued)

- xERRLINE. This bit indicates whether the IOMC asserted the Error backplane signal at any time during the I/O Bus transaction. The Error signal indicates that the IOMC is returning an error word. The last word in the reply should be checked to determine the type of error (if any), and the correction action needed.

  If the problem occurred while the message was being sent, then a word will be added to the responder's reply, specifying the type of error that occurred (see Section 6.8.2 MESSAGE FORMATS for more detail).

  To check that an I/O Bus transaction has completed sucessfully, the microprocessor must:

  (1) Poll xBUSSTAT continuously (checking for timeout), waiting for xOpCompl to be asserted. The value read could be ANDed with 0001 (hex) and checked for equality with 0000 (hex).

  (2) Read xBUSSTAT again, checking for successful completion. The value read could be ANDed with 07FD (hex) and checked for equality with 0028 (hex).

  If test (2) fails, then the analysis procedure outlined above should be followed to determine the cause of the abnormal termination. Figure 6-1 summarizes that analysis.

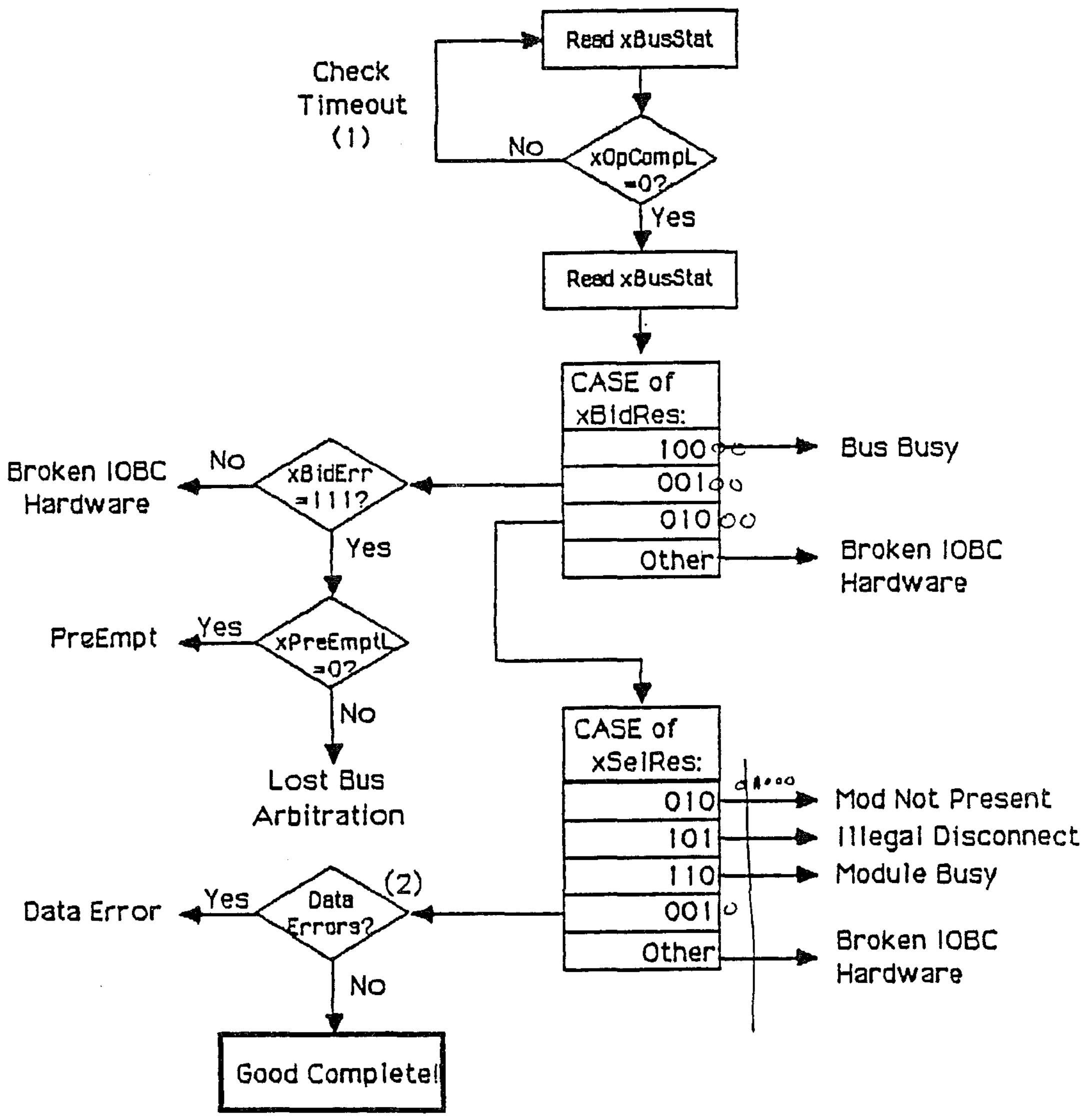

Figure 6-1 xBUSSTAT Register Analysis Flow Chart

Notes:

(1) Timeout values, and actions to be taken when a timeout occurs are described in Section 7 (ERROR DETECTION AND HANDLING).

(2) A Data Error occurs if xSBE, xMBE, or xErrLine is active.

6.2 INTERRUPTS

In the MLI-DTM, I/O Bus-related interrupts are generated by the IOBCs. Each IOBC has an associated interrupt line to the microprocessor. An IOBC generates an interrupt because a message has arrived, and has been put into the buffer, or because the IOMC has asserted IIOReqL on the backplane, indicating that there is something in the mailbox register for the message router.

It must be emphasized that this is very much an implementation decision. The SMC and future DTM's microprocessors may be interrupted for different reasons. For example, future DTMs may be interrupted by the buffer control when buffer ownership has been granted to the microprocessor.

When the microprocessor has accepted the interrupt, and read the appropriate xBUSSTAT register to determine the cause of the interrupt, the microprocessor must issue an Interrupt Acknowledge command for the interrupting IOBC to the buffer control. This command is issued via the buffer control rather than directly to the IOBC because the IOBC's command registers do not have enough bits available.

The method of determining the interrupt reason, and acknowledging the interrupt, is implementation dependent. A future design might choose instead to use vectored interrupts and the microprocessor's interrupt acknowledge cycle rather than the current MLI-DTM scheme.

In the current implementation, the IIOREQL interrupt is disabled on all I/O modules except the message router during system initialization. This is done for performance considerations. If the message router module breaks, this interrupt is enabled on the module that is going to be assigned the message router responsibilities.

If both interrupt conditions are simultaneous, or nearly so, the hardware insures that only one interrupt at a time is presented to the microprocessor. The value in xBUSSTAT does not change until the Interrupt Acknowledge command is issued.

6.3 BASIC PROTOCOL

This section describes how the commands, status bits, and interrupts relate to each other. It describes the flow for a normal message both from the standpoint of the initiator, and from the standpoint of the responder.

6.3.1 NORMAL INITIATOR FLOW

Figure 6-2 shows the protocol followed by the microprocessor when initiating an I/O Bus transaction.

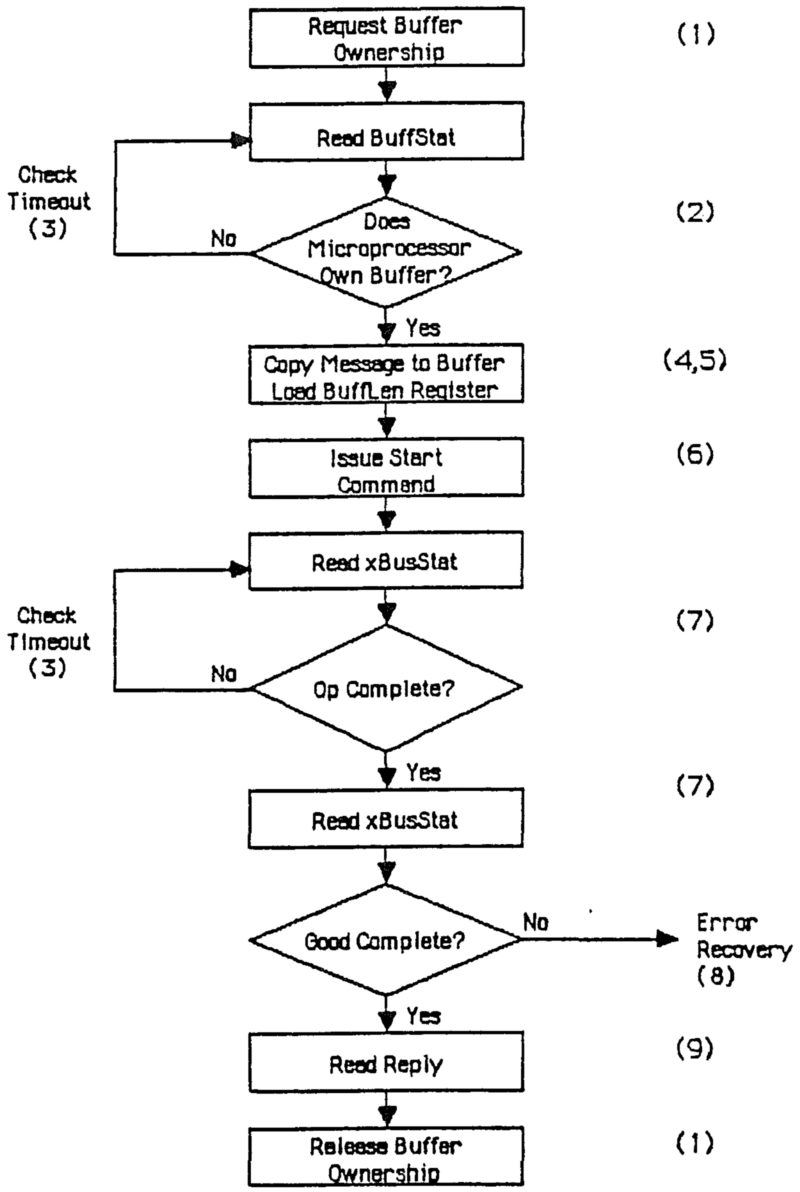

Figure 6-2 Initiator Flow Chart

6.3.1 NORMAL INITIATOR FLOW (Continued)

Notes:

1 Buffer control commands are described in Section 6.1.2.1.

2 The Buffer Status Register (BUFFSTAT) is described in Section 6.1.2.6.

3 Timeout values and actions to be taken when a timeout occurs are described in Section 7 (ERROR DETECTION And HANDLING).

4 It is suggested that the message be formatted in the microprocessor's main memory, and then copied into the buffer, so as to minimize the time spent with the Buffer Owned.

5 The Buffer Length register (BUFFLEN) is described in Section 6.1.2.4.

6 IOBC commands are described in Section 6.1.2.5. Replace the x with A or B, depending on the IOBC being used.

7 IOBC Status register (xBUSSTAT) is described in Section 6.1.2.6.

8 Error recovery is described in Section 7.

9 It is suggested that the reply be moved out of the buffer as quickly as possible, so as to minimize the time spent with the Buffer Owned.

6.3.2 RESPONDER FLOW

When an IOBC interrupt occurs, it could be because a message has arrived, or because an IOMC has something for the message router in its mailbox register.

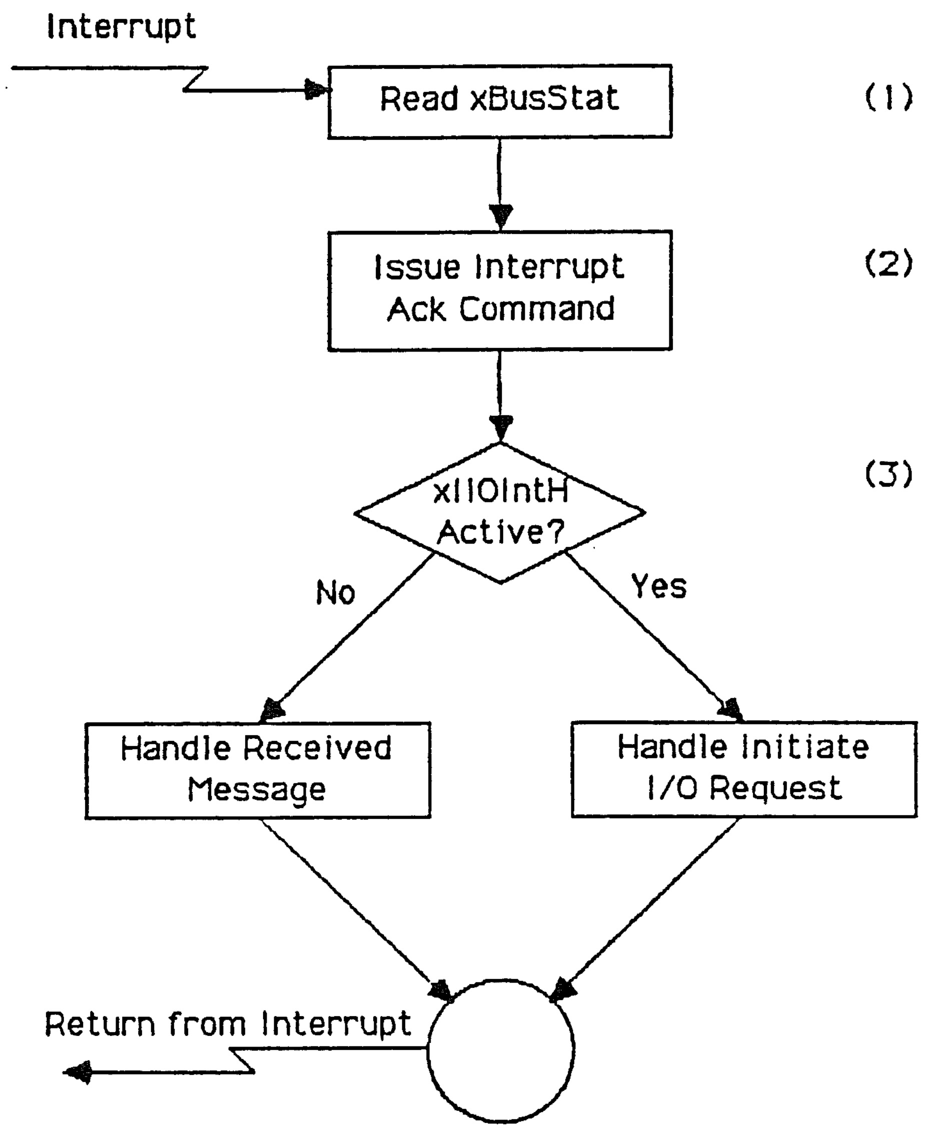

Figure 6-3 Interrupt Flow Chart

6.3.2 RESPONDER FLOW (Continued)

NOTES:

1 IOBC Status register (xBUSSTAT) is described in Section 6.1.2.6. The x can be replaced by A or B, depending on which IOBC caused the interrupt.

2 Buffer control commands are described in Section 6.1.2.1.

3 Value of xIIOIntH in xBUSSTAT read at step (1) should be used; xBUSSTAT may change at any time after the Interrupt Acknowledge command is issued.

The two cases are handled as shown in figures 6-3 and 6-4.

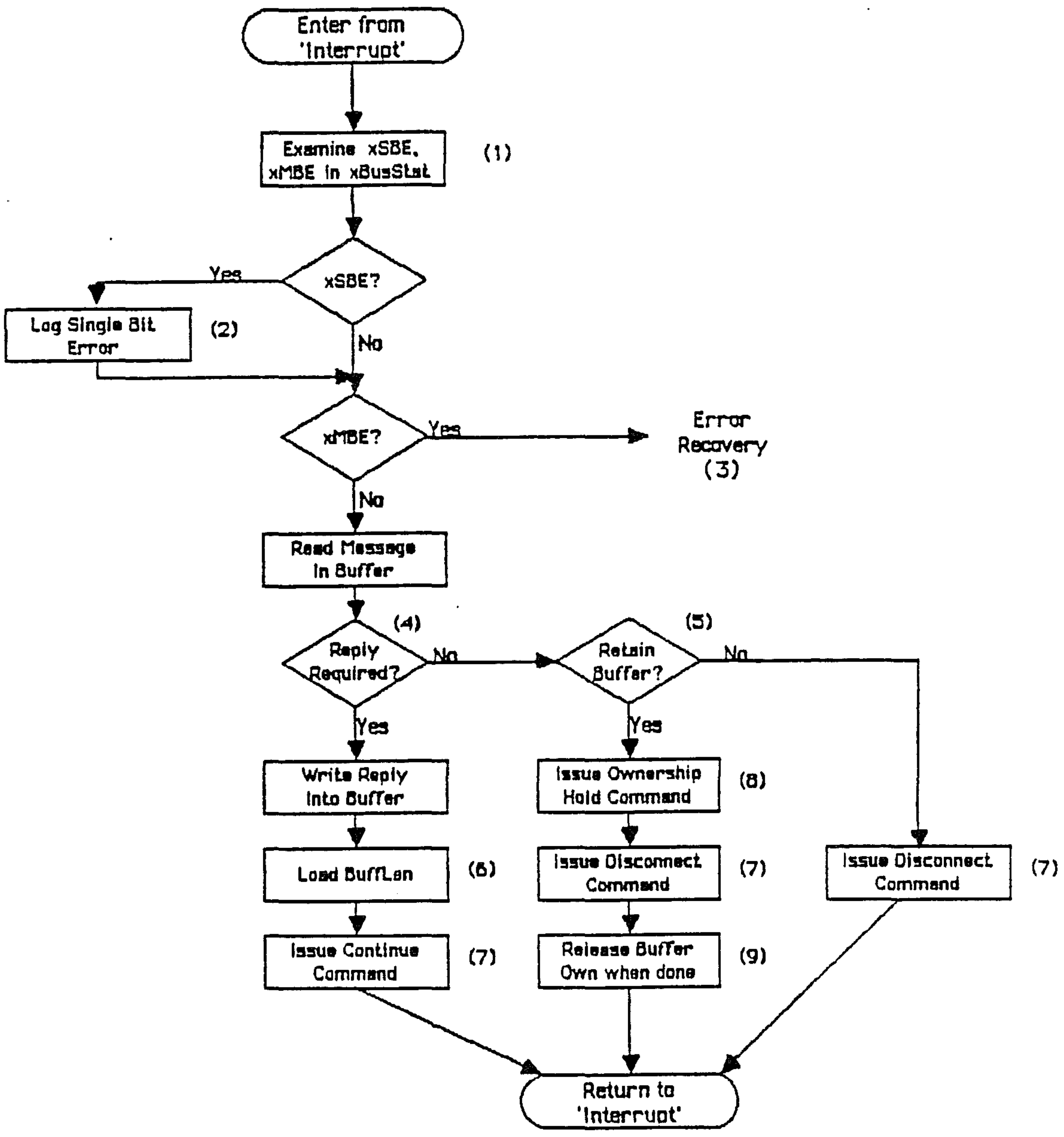

Figure 6-4 Incoming Message Handling Flowchart

6.3.2 RESPONDER FLOW (Continued)

NOTES:

1 IOBC Status register (xBUSSTAT) is described in Section 6.1.2.6. Replace the x with A or B, depending on which IOBC caused the interrupt.

2 Single bit error logging is discussed in Section 7 (ERROR DETECTION And HANDLING).

3 See Section 6.4 (ERROR RECOVERY).

4 This decision depends on the op code embedded in the message, and the presence or absence of errors. Op codes and their meanings are discussed in Section 6.8 (BUS PROTOCOL). If an error has occurred, then a reply specifying the kind of error is required.

5 If the microprocessor is to release the I/O Bus but continue using the buffer, then the Yes branch must be taken here.

6 The Buffer Length register (BUFFLEN) is described in Section 6.1.2.4.

7 IOBC commands are described in Section 6.1.2.5.

8 Buffer control commands are described in Section 6.1.2.1.

9 This buffer control command should only be issued when the microprocessor has finished using the buffer.

When an interrupt occurs because the IOMC has something in its mailbox register for the message router, the microprccessor must proceeds as shown in Figure 6-5.

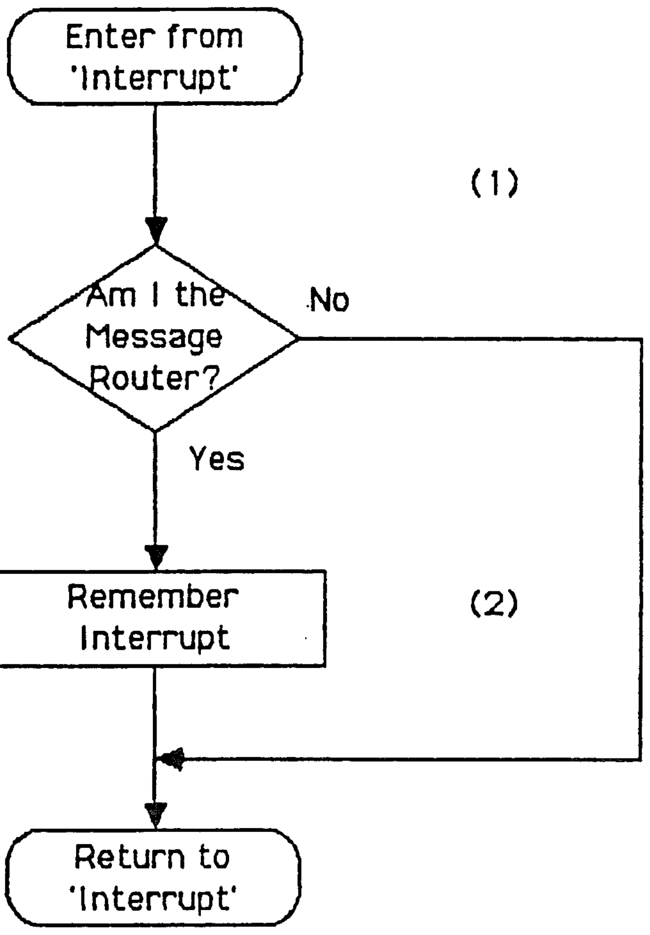

Figure 6-5 Initiate I/O Request Flow Chart

6.3.2 RESPONDER FLOW (Continued)

NOTES:

1 Only one module in the system is the message router (refer to IOMC PECULIARITIES, Section 4.5). All modules except the message router ignore this interrupt. A hardware mask prevents the interrupt. The mask can be set by shifting in (through the data chain) the appropriate bits by the SMC.

2 The message router must respond to the interrupt. However, the response requires at least one I/O Bus transaction (a read of the IOMC mailbox register). I/O Bus operations cannot be done within an interrupt handler. Hence, the fact that the interrupt occurred is recorded. Reading the mailbox register and processing the IIO request is handled later. Information recorded includes which bus caused the interrupt (each IOMC has its own mailbox register).

Eventually, any outstanding interrupts will have been handled, and the microprocessor returns to some underlying process. In the message router, this process must periodically check if an Initiate I/O Request interrupt has been recorded. If there is an interrupt, it is handled as shown in Figure 6-6.

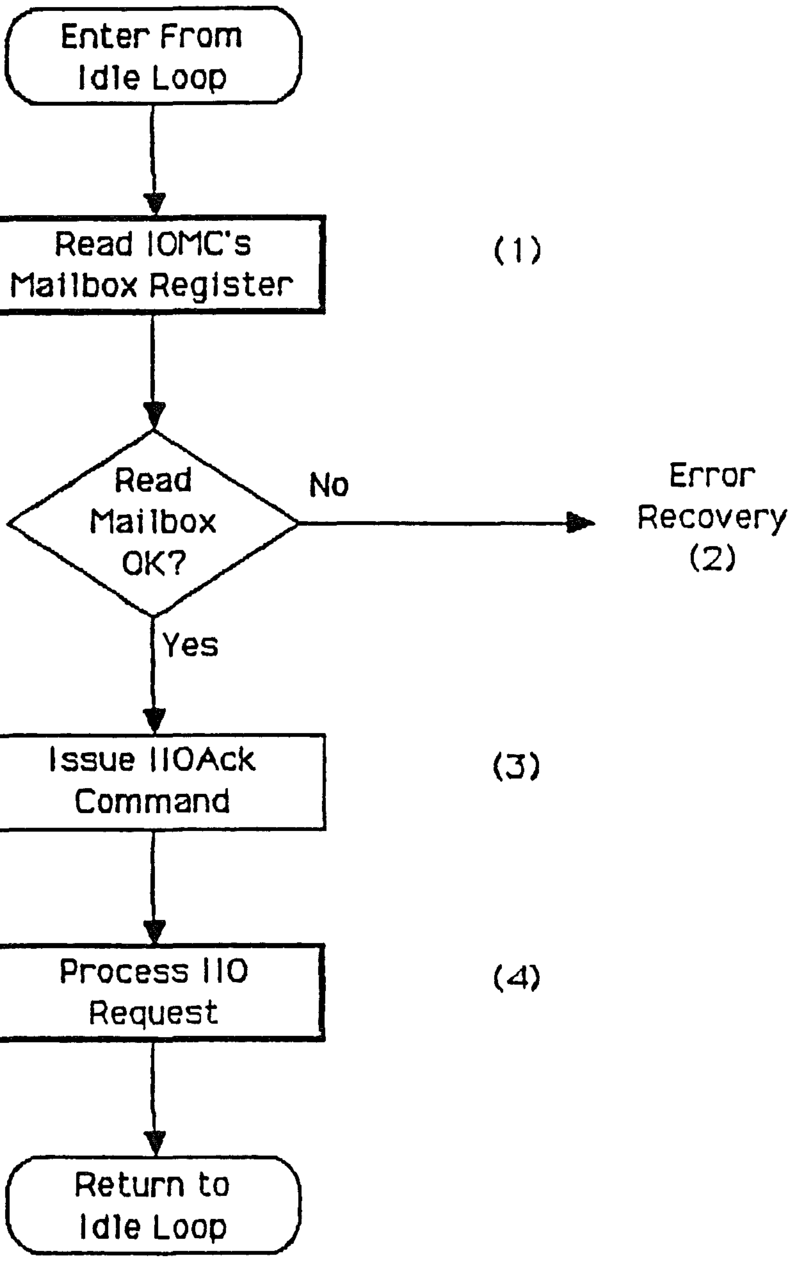

Figure 6-6 Mailbox Read Flow Chart

6.3.2 RESPONDER FLOW (Continued)

NOTES:

1 This box shows a complex sequence of operations. Basically, the module must be an initiator, and send a Read Mailbox message to the IOMC. The IOMC replies with the mailbox contents.

2 Error recovery is discussed in Section 6.4 (ERROR RECOVERY).

3 IOBC commands are described in Section 6.1.2.5.

4 This box shows a complex sequence of operations, which are not yet specified.

Since some of the flows above are initiated by interrupts, interactions between the flows are possible. The flows fall into two distinct categories; interrupt handler flows, and underlying process flows. The interrupt handler flows include the Interrupt Flow Chart, the Initiate I/O Request Flow Chart, and the Incoming Message Handling Flow Chart. The underlying process flows include the Initiator Flow Chart and the Mailbox Read Flow Chart.

Initiate I/O Request interrupts may occur at any time, from either IOBC. For modules other than the message router, the interrupt is essentially ignored, and any flow in process at the time, whether an underlying process, or interrupt handler, is continued.

In the message router, the interrupt is remembered, and any other flow is allowed to complete before the Mailbox Read flow is executed. A second Initiate I/O Request interrupt from the same IOBC is not possible until the IIOAck command has been issued. This means that there is no need to queue the fact that the interrupt occurred; a simple flag per IOBC suffices. However, it may be necessary to queue the IIO Request read from the mailbox.

6.3.2 RESPONDER FLOW (Continued)

In the case of incoming message interrupts, a requestor (IOBC or MLI Control) must own the buffer before it can cause an interrupt. This means that, if the microprocessor owns the buffer, incoming message interrupts cannot occur. Furthermore, once an IOBC owns the buffer, the other IOBC and the MLI Control cannot cause incoming message interrupts until buffer ownership is released; that is, until the incoming message handling flow is complete.

As an example, suppose that the microprocessor is attempting to send a message at the same time that an IOBC starts to receive an incoming message (assume that the Module Busy flag is not set). The sequence of events depends on who gets ownership of the buffer first. If the IOBC gets ownership first, then the sequence of events is as shown in Figure 6-7.

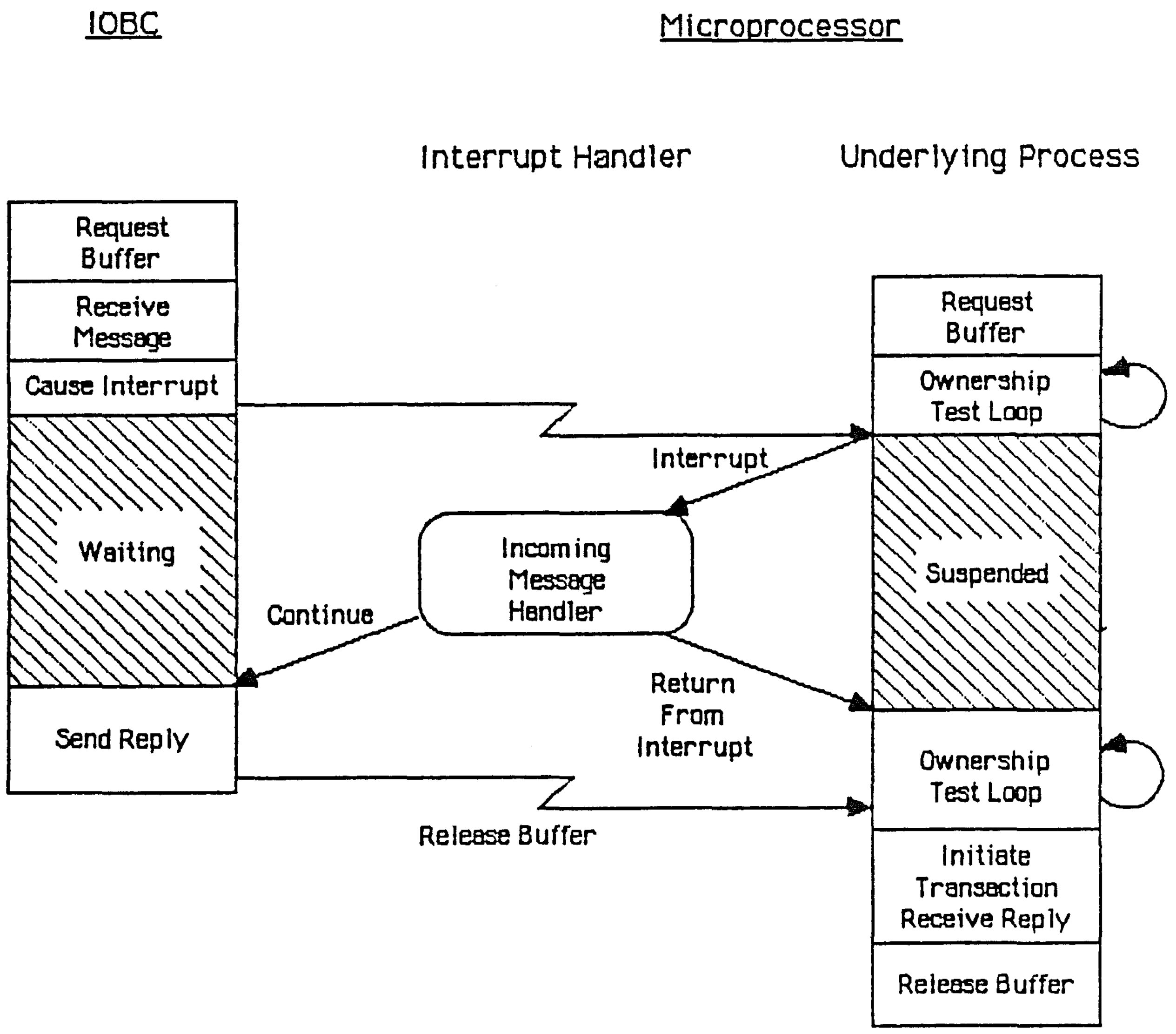

Figure 6-7 Microprocessor-IOBC Interaction (IOBC Gets Buffer)

6.3.2 RESPONDER FLOW (Continued)

The interrupt may occur any time before the microprocessor gets into the ownership test loop; in any case, the underlying process cannot exit that loop until the interrupt has occurred, the incoming message handling flow has been executed, buffer ownership has been released, and the microprocessor has returned from the interrupt handler to the underlying process. At that time, the microprocessor gains ownership of the buffer, and underlying process falls out of the ownership test loop.

If the incoming message does not require a reply, the flow is similar; Figure 6-8 shows a subtlety in buffer ownership.

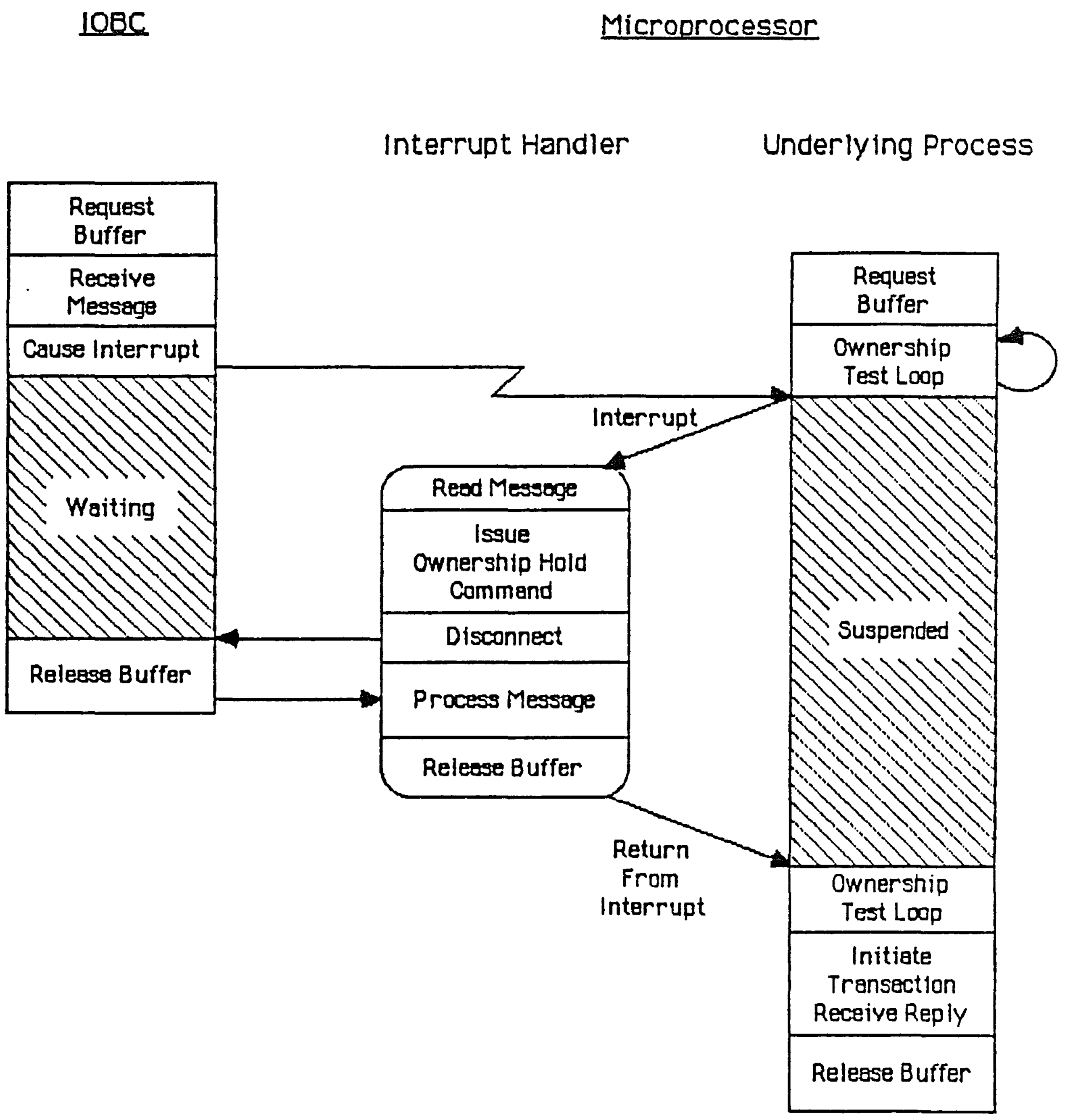

Figure 6-8 Microprocessor-IOBC Interaction (Disconnect)

6.3.2 RESPONDER FLOW (Continued)

The subtlety relates to the effects of the various Release Buffer Ownership commands. When the IOBC receives the Disconnect command, it disconnects from the I/O Bus, and releases ownership of the buffer. Because the microprocessor issued the Ownership Hold Command before it issued the Disconnect, buffer ownership reverts to the microprocessor. When the interrupt handler has finished with the buffer, it issues a Release Buffer Ownership command to the buffer control.

A problem is that the microprocessor (in the interrupt handler) owns the buffer; releasing ownership normally indicates that the microprocessor's use of the buffer, which started with a Request Buffer Ownership command, has finished. As Figure 6.8 shows, the underlying process has not started to use the buffer. Releasing ownership at this point would prevent the underlying process from ever getting out of the ownership test loop.

The buffer control hardware takes care of this problem. It remembers the Ownership Hold Command, and treats the Release Buffer Ownership command that follows it (the one in the interrupt handler) differently from the one issued in the underlying process.

If the microprocessor gets ownership of the before the IOBC, then the IOBC cannot cause an incoming message interrupt. The sequence of events is shown in Figure 6-9.

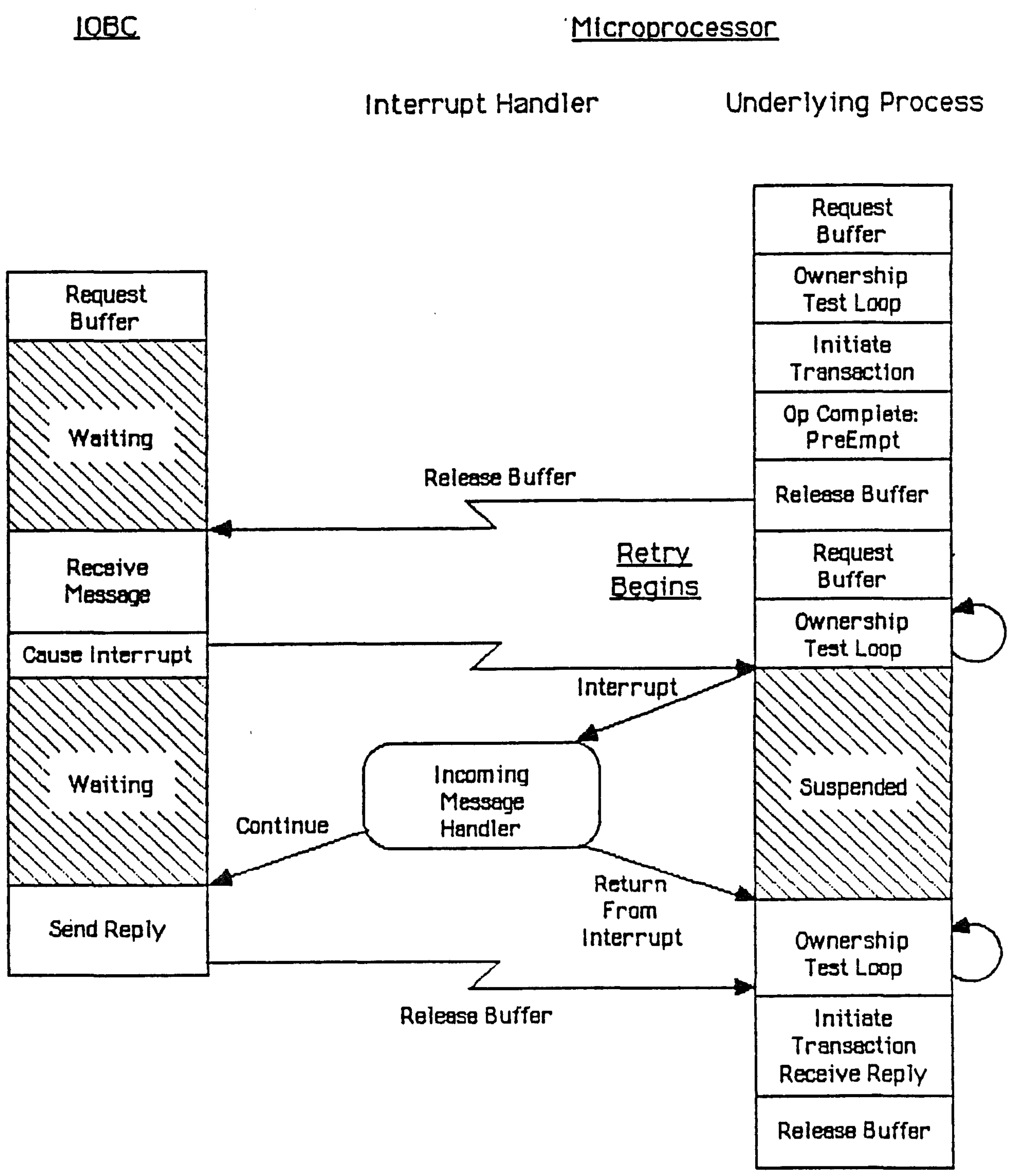

Figure 6-9 Microprocessor-IOBC Interaction (Micro Gets Buffer)

6.3.2 RESPONDER FLOW (Continued)

The IOBC waits for buffer ownership while the microprocessor executes the initiator flow. When the microprocessor issues the Start command, Op Complete with PreEmpt is reported immediately in the xBUSSTAT register. The microprocessor releases buffer ownership, and starts to retry the I/O Bus operation. The IOBC immediately takes the buffer, and the the flow proceeds as if the IOBC had gotten buffer ownership first.

Interactions between interrupt handlers are governed by buffer ownership, and the microprocessor's interrupt priority structure. As noted above, the buffer ownership mechanism prevents other incoming message interrupts while any requestor owns the buffer. This does not prevent an interrupt handler from being interrupted between the time it releases buffer ownership and the time it executes the Return From Interrupt instruction.

In the basic M68020 microprocessor, seven interrupt levels are provided. While an interrupt handler is running at a particular level, other interrupts at that level and below are masked. In the MLI-DTM, IOBC A causes level 5 interrupts, while IOBC B causes level 4 interrupts. This means that the interrupt handler for IOBC B can be interrupted by IOBC A. The following table summarizes the possible interactions.

Handler Running	Interrupted By
IOBC B, Initiate I/O	IOBC A, Initiate I/O IOBC A, Incoming Message
IOBC B, Incoming Message	IOBC A, Initiate I/O *IOBC A, Incoming Message

* This combination can only occur after the IOBC B incoming message handler has released buffer ownership.

IOBC A interrupt handlers cannot be interrupted, because they run at level 5, while IOBC B handlers run at level 4.

6.4 ERROR RECOVERY

I/O Bus errors fall into three categories; Normal errors, Unusual errors, and Broken errors. Normal errors are those that occur during normal operation. They are:

o BUS BUSY. The I/O Bus was in use by another module when the microprocessor issued the Start command.

o LOST BUS ARBITRATION. This module and another module bid for the I/O Bus at the same time. The other module had higher priority.

o PREEMPT. Either IOBC or the MLI control requires buffer ownership for an incoming message.

o MODULE BUSY. The target responder is unable to accept a message at this time.

Two errors are classified as unusual:

o MODULE NOT PRESENT. The target responder is either physically not present, or is offline.

o ILLEGAL DISCONNECT. The target responder did not complete the I/O Bus transaction successfully.

These errors do not occur during normal operation, but may be caused by stop logic, or by fault recovery in another module.

Finally, there are the Broken errors. These indicate that some piece of hardware is broken. They include:

o TIMEOUTS. An operation did not complete in the required time.

o DATA ERRORS. A single- or multi-bit error occurred during data transfer.

o BROKEN HARDWARE. The value in a register (xBUSSTAT or BUFFSTAT for example) is not valid.

6.4 ERROR RECOVERY (Continued)

This section deals only with the Normal errors. Unusual and Broken errors are described in Section 7 (ERROR DETECTION AND HANDLING).

Bus Busy and Lost Bus Arbitration errors can be retried immediately. If the target responder is connected to both I/O Busses, the retry uses the other bus. If this module has set the Module Busy flip-flop, then it is not necessary to release the buffer before retrying. In all other cases, the buffer must be released and reacquired before retrying. The outgoing message must be copied into the buffer again.

FreEmpt errors can also be retried immediately. In this case, however, the buffer must be released and reacquired before retrying. The outgoing message must be copied into the buffer again because the PreEmpting message will have destroyed it.

The retry waits for buffer ownership; during that time, the incoming (preempting) message causes an interrupt.

A Module Busy condition indicates that the target responder will be busy for some time. The I/O Bus transaction must be retried. The buffer is released, and reacquired between retries. Again, the outgoing message must be copied into the buffer again because activity between retries may have destroyed it.

6.5 I/O BUS SIGNAL GLOSSARY

This glossary of I/O Bus signals contains the following information about each signal:

Logical Name	The name that is normally used to refer to the signal. This name does not indicate which I/O Bus the signal is part of, nor does it follow the normal signal naming conventions.
Signal Names	The actual board and backplane names by which this signal is known, both on the backplane and on each of the cards. There must be at least one name for each I/O Bus here.
Sourced by	Indicates whether the signal is sourced by the initiator, the responder, or both.
Active	Indicates whether the signal is active high or active low.
Electrical Type	The different electrical types are described in detail in Section 6.7 (DRIVERS AND RECEIVERS).
Description	A brief description of the signal's use.
Timing	Bus timing requirements for the signal.

6.5.1 I/O BUS SIGNALS

Data(38:0)

Signal Names	I/O Bus #0 - TIO1DATA-nnP and I/O Bus #1 - TIO2DATA-nnP
Sourced by	Initiator during message send; or responder during reply send.
Active	High
Electrical Type	Tristate data
Description	Four bytes of data (Data(31:0)) and seven bits of ECC (Data(38:32)) are transferred on these lines every system TTL clock during the message and reply phases of an I/O Bus transaction. The ECC code is specified in Section 6.8 (BUS PROTOCOL). These signals are also used during the bus arbitration and responder selection phases of a transaction. During bus arbitration, each contending module drives a different bit low on the data bus. The module driving the highest numbered bit wins. Physically, each module drives its Normal Request or Emergency Request pin; each card slot has these two outputs wired to (card slot unique) data bit wires. The bus termination makes the signals that are not driven (either because the module is physically not present, or because it does not want the bus) appear high. During responder selection, the target responder's ID with correct ECC is broadcast on the bus, as if it were data. See Section 6.1.1 (BUFFER) for valid target addresses.

6.5.1 I/O BUS SIGNALS (Continued)

Timing	Data transfer occurs on every TTL clock. During data transfer, data must be stable at the receiver on the rising edge of the clock. The signals listed following define when a data transfer is taking place. For timing during bus arbitrarion and responder selection, see BusReq and Modreq.

6.5.2 BUSREF SIGNALS

Signal Names	I/O Bus #0 - TIO1BUSREF$P, TIO3BUSREF$P and I/O Bus #1 - TIO2BUSREF$P, TIO4BUSREF$P.
Sourced by	All modules (except the IOMC) must be capable of sourcing this signal; one module is selected (by the maintenance processor) at system initialization time. The source will be changed unless the module chosen fails at initialization. The enable for this signal is controlled by a bit on each module's maintenance chain. The enable must remain stable while the maintenance chain is shifting; shifting the maintenance chain must not affect this signal unless the enable bit is changed by the shift.
Active	High
Electrical Type	Tristate control
Description	This signal is the system TTL clock divided in half. It controls the timing for I/O Bus startup. Each module on the bus must latch BusRef, and use it to determine when to sample or drive the BusReq, BusBusy, ModReq, ModPresent, ModAvail, and ModBusy signals. All of these signals must be asserted with the falling edge of BusRef.
Timing	This signal must always be valid for correct bus operation. The I/O Bus controller behavior is undefined if this signal is not present.

6.5.3 BUSREQ SIGNALS

Signal Names	I/O Bus #0 - TIO1BUSREQ$N, TIO3BUSREQ$N or I/O Bus #1 - TIO2BUSREQ$N, TIO4BUSREQ$N
Sourced by	Initiator.
Active	Low
Electrical Type	Open Collector Control 1
Description	When a module wants to use the bus, and the bus is not busy (both BusBusy and BusReq are false), it must drive this signal true (low), and at the same time drive its priority to the data bus.
Timing	On the same clock that this signal is driven, latched BusReq is checked. If latched BusReq is true (low), then this signal and the priority bit drivers are driven false (disabled) for one TTL clock. If latched BusReq is false, then this signal is driven for four TTL clocks (two BusRef). During the first BusRef, the priority bit is driven, and priority is resolved. During the latter half of the second BusRef, the priority bit is disabled to allow the bus to settle before the target address is brodcast.

6.5.4 BUSBUSY SIGNALS

Signal Names	I/O Bus #0 - TIO1BUSBUSYN, TIO3BUSBUSYN or I/O Bus #1 - TIO2BUSBUSYN, TIO4BUSBUSYN
Sourced by	Initiator
Active	Low
Electrical Type	Open Collector Control 1
Description	If, after two BusRefs have passed, the requesting module with the highest priority still wants to use the bus, it drives BusBusy true (low), and drives it until the I/O Bus transaction is complete.
Timing	The module that won the bus bidding drives this signal true on the same clock that it drives BusReq false. In normal operation the signal is reset two BusRef clocks after ModPresnt goes high. The responder monitors this signal, and aborts the current operation if this signal was false before the end of the I/O transaction.

6.5.5 MODREQ SIGNALS

Signal Names	I/O Bus #0 - TIO1MODREQ$N, TIO3MODREQ$N I/O Bus #1 - TIO2MODREQ$N, TIO4MODREQ$N
Sourced by	Initiator
Active	Low
Electrical Type	TriState Control
Description	On the same clock that the module first drives BusBusy, it must also drive ModReq, and broadcast the address of the target responder on the data wires.
Timing	This signal and the target address are driven for one BusRef clock only. They are asserted on the same clock that the module first drives BusBusy. The target module (if present) should have detected its address by the end of this BusRef clock.

6.5.6 MODPRESENT SIGNALS

Signal Names	I/O Bus #0 - TIO1MODPRSNN, TIO3MODPRSNN or I/O Bus #1 - TIO2MODPRSNN, TIO4MODPRSNN
Sourced by	Responder
Active:	Low
Electrical Type	TriState Control
Description	Upon detecting its Module ID (along with ModReq), any module that is not offline must drive this signal true (low). This indicates only that the module is present and online; it must be asserted regardless of the module's ability to accept a message. A module may be offline for a number of reasons; the SMC may put the module offline by shifting the appropriate value into the maintenance chain, a fault detector may detect a failure and take the module offline, or stop logic set up in the module may take the module offline. Dropping ModPresent before the end of the I/O Bus transaction indicates to the initiator that the connection should be broken.
Timing	This signal is driven active (low) by the responding module on the same clock edge that the initiator drives ModReq false. It is reset one BusRef clock after either ModAvail or ModBusy is driven false. The initiator monitors this, and indicates an illegal disconnect to the microprocessor if ModPresent did not abide by this timing.

6.5.7 MODAVAIL SIGNALS

Signal Names	I/O Bus #0 - TIO1MODAVALN, TIO3MODAVALN or I/O Bus #1 - TIO2MODAVALN, TIO4MODAVALN
Sourced by	Responder
Active	Low
Electrical Type	TriState Control
Description	After asserting ModPresent, the target responder must check to see if it can accept the incoming message. When it can, it drives ModAvail true (low). If it cannot accept the message, it indicates that it is busy by driving ModBusy true (low). Either ModAvail or ModBusy must be driven true to complete the responder selection phase of the I/O Bus transaction. This must occur within the timeout period (the initiator checks for this timeout). ModAvail is held true until the responder decides that the I/O Bus transaction is complete.
Timing	This signal can be asserted on any falling edge of BusRef after ModPresent is asserted. It is reset (also on a falling edge of BusRef) when the responder wants to terminate the message.

6.5.8 MODBUSY SIGNALS

Signal Names	I/O Bus #0 - TIO1MODBUSYN, TIO3MODBUSYN or I/O Bus #1 - TIO2MODBUSYN, TIO4MODBUSYN
Sourced by	Responder
Active	Low
Electrical Type	TriState Control
Description	This signal indicates that the responder it is unable to accept the incoming message. It is driven true instead of ModAvail, and is subject to the same timing constraints. Use of this signal must abide by the Deadlock Prevention Protocol described in Section 6.8 (BUS PROTOCOL).
Timing	This signal has similar timing as ModAvail, but it is asserted for one BusRef clock only.

6.5.9 TRANSFER SIGNALS

Signal Names	I/O Bus #0 - TIO1TRANSFRN, TIO3TRANSFRN or I/O Bus #1 - TIO2TRANSFRN, TIO4TRANSFRN
Sourced by	Initiator during message send, or responder when reply is sent.
Active	Low
Electrical Type	TriState Control
Description	This signal is used to coordinate the transmission of data. Transfer is sourced by whichever module is sending the data. During the message transmission phase, it is sourced by the initiator; when the initiator has completed sending the data, it must tristate disable Transfer so that the responder may use it to coordinate the transmission of a reply.
Timing	It is asserted two system TTL clocks before the first data is put on the bus, and negated two system TTL clocks before the last data is put on the bus. In other words, Transfer is asserted for the number of clocks required to transfer the data, starting two clocks ahead of the data.

6.5.10 ERROR SIGNALS

Signal Names	I/O Bus #0 - TIO1ERROR$$N, TIO3ERROR$$N or I/O Bus #1 - TIO2ERROR$$N, TIO4ERROR$$N
Sourced by	IOMC
Active	Low
Electrical Type	Open Collector Control 2
Description	This signal is used by the IOMC to indicate that an error word is going to be returned in the reply.
Timing	This signal can be asserted any time when ModAvail is asserted, for at least one clock.

6.5.11 IIOREQ SIGNALS

Signal Names	I/O Bus #0 - TIO3IIOREQ$N or I/O Bus #1 - TIO4IIOREQ$N
Sourced by	IOMC
Active	Low
Electrical Type	Open Collector Control 2
Description	This signal is used by each IOMC to indicate that there is something in its mailbox register for the message router.
Timing	This signal may be asserted at any time, independent of any other activity on the I/O Bus, and must be held until the message router responds with IIOAck.

6.5.12 IIOACK SIGNALS

Signal Names	I/O Bus #0 - TIO1IIOACK$N or I/O Bus #1 - TIO2IIOACK$N
Sourced by	Message Router
Active	Low
Electrical Type	Open Collector Control 2
Description	This signal is used by the message router to indicate to an IOMC that it has read the mailbox register successfully, and that the IOMC may now accept another mailbox write operation from a MCACM. This signal is used only in conjunction with IIOReq. It is independent of any other activity on the I/O bus.
Timing	IIOAck must be driven true (low) for one TTL clock.

6.5.13 NORMAL REQUEST SIGNALS

Signal Names	I/O Bus #0 - TIO1NORMREQN, (On card name only) I/O Bus #1 - TIO2NORMREQN, (On card name only)
Sourced by	Initiator.
Active	Low
Electrical Type	Open collector request
Description	This signal is used during bus arbitration; it is used by a card to drive its Normal priority onto the appropriate data bit of the bus. Each card drives this signal on a fixed pin, regardless of the slot it is plugged into. That pin at each slot is connected on the backplane to the wire in the data bus appropriate to that slot. The signal name is an on-card name only; on the backplane, the signal name will be TIOxDATA-nnP, where x=1 for I/O Bus #0 and x=1 for I/O Bus #1, and nn depends on the card slot.
Timing	See BUSREQ

6.5.14 EMERGENCY REQUEST SIGNALS

Signal Names	I/O Bus #0 - TIO1EMERGRQN (On card name only) I/O Bus #1 - TIO2EMERGRQN (On card name only)
Sourced by	Initiator
Active	Low
Electrical Type	Open collector request
Description	This signal is similar to NormalRequest, except that it drives Emergency Priority rather than Normal Priority onto the bus.
Timing	See BUSREQ.

6.6 BACKPLANE PINS

The two I/O Busses are arranged as shown in Figure 6-10.

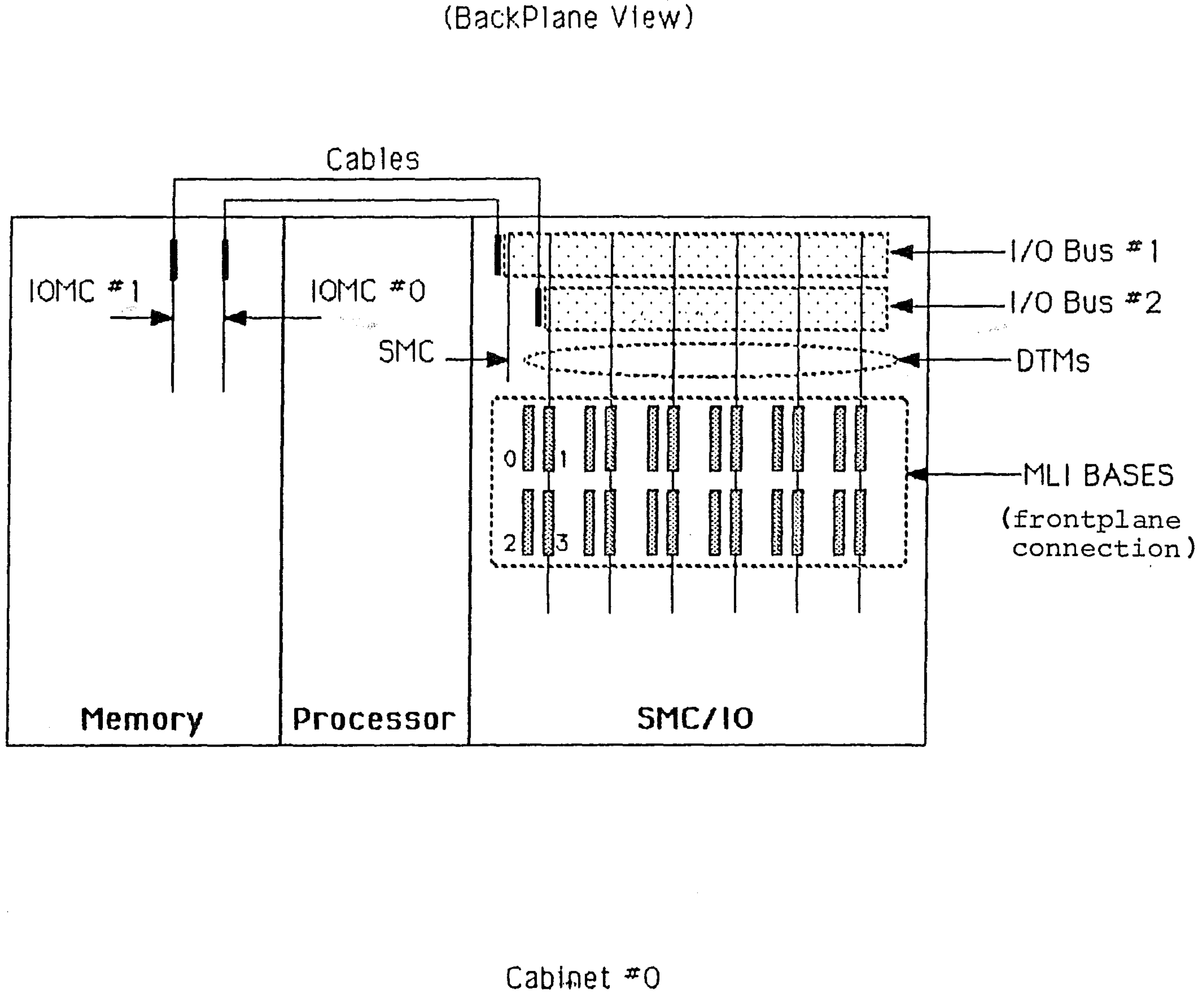

Figure 6-10 I/O Bus Layout

6.6 BACKPLANE PINS (Continued)

Only DTMs connect to both busses. Backplane signal locations are identical for both IOMCs, the SMC, and I/O Bus #0 connections for the DTMs. A second set of backplane pins provides the I/O Bus #1 connections for the DTMs.

Each I/O Bus consists of 50 signals; 11 for control, and 39 for data. The data signals consist of 32 information bits and 7 error correction bits.

Control signals are both driven and received by each module on the bus. In the DTM backplane, the connection between source and load is on the backplane; a DTM drives each control signal through one backplane pin, and receives it through another. This allows fault detection circuitry on the card to check that each control signal is actually being driven on the backplane.

On-card signal names are different for the driver and receiver. Driver signals are named TIO1xxxxxxxx (for I/O Bus #0) and TIO2xxxxxxxx (for I/O Bus #1), while the equivalent receiving signals are named TIO3xxxxxxxx (for I/O Bus #0) and TIO4xxxxxxxx (for I/O Bus #1). Backplane signals have a single name (TIO1xxxxxxxx or TIO2xxxxxxxx) as appropriate.

On the SMC and IOMC, the drivers and receivers are connected on the card; there were not enough pins on these modules to support the scheme used with the DTM.

Normal and Emergency Requests TIOxNORMREQN and TIOxEMERGRQN are connected on the backplane to different data bus bits, depending on card slot location and bus (I/O Bus #0 or #1). NormalRequest wires are connected to data bits 0 through 12 inclusive; EmergencyRequest wires use bits 16 through 28. Each group of 13 bits is further divided; data bits 0 and 16 are used by the SMC, and the other twelve bits are divided by cabinet (six for DTMs in each cabinet).

6.6 BACKPLANE PINS (Continued)

Both cabinets in a two cabinet system are identical; DTMs may be plugged into any slot in either cabinet without changing straps. These two facts necessitate some complexity in the way that the NormalRequest and EmergencyRequest wires are connected.

A DTM has a different priority on each bus. For example, a DTM in slot #6 of cabinet #0 using bus #0 has priority over all the other modules in the same cabinet; however, it competes with DTMs in cabinet #1 that are using bus #1. These modules have higher priority than all the modules in cabinet #0 (using bus #0).

Actual bit number connections are as follows:

Slot	NormalRequest		EmergencyRequest	
Number	Bus #0	Bus#1	Bus #0	Bus #1
1	Bit 1	Bit 7	Bit 17	Bit 23
2	Bit 2	Bit 8	Bit 18	Bit 24
3	Bit 3	Bit 9	Bit 19	Bit 25
4	Bit 4	Bit 10	Bit 20	Bit 26
5	Bit 5	Bit 11	Bit 21	Bit 27
6	Bit 6	Bit 12	Bit 22	Bit 28

Note that a module can determine which slot it is plugged into from the Maintenance ID jumpers and the Cabinet jumper on the backplane. The three upper bits of the Maintenance ID jumpers are the slot number. The Cabinet jumper is not required during bus arbitration, but is required during responder selection.

6.7 DRIVERS AND RECEIVERS

This section specifies the electrical requirements for Drivers and Receivers used on boards that connect to an I/O Bus. Almost all I/O Bus signals are bi-directional; the only exceptions are Error, IIOReq, IIOAck, NormalRequest, and EmergencyRequest. Therefore, both loading and driving characteristics are specified for for each electrical type.

The I/O Bus must continue to function if either cabinet of a two cabinet system is powered off. Since there is no electrical isolation between cabinets, this requires that the drivers and receivers not load the bus excessively when powered off. Therefore, these specifications include a Powered Off Maximum Load (specified in mA) when the bus is at the specified (high) voltage.

The signals that make up the I/O Bus fall into the following electrical classes:

TriState Data
TriState Control
Open Collector Control 1
Open Collector Control 2
Open Collector Request

6.8 I/O DATA BUS PROTOCOL

This section describes the error correction code used on the I/O data bus, the message formats, and gives some typical timing diagrams of an I/O Bus transaction.

6.8.1 ERROR CORRECTION CODE

Data transfer on the I/O Bus is protected by a modified Hamming error detection and correction code. This code corrects all single bit errors, detects all double bit errors, and detects all errors within a four bit section. It also detects almost all other multiple bit errors (but not all). This code also protects data in the I/O buffer RAM, and the microprocessor dynamic RAM and EPROMS.

6.8.1 ERROR CORRECTION CODE (Continued)

The following table shows the code, and the bits monitored.

Modified Hamming Code (DTM)

		6	5	4	3	2	1	0
00	:=	x	x			x		
01	:=	x	x					x
02	:=	x	x				x	
03	:=	x	x	x				
04	:=	x		x	x			
05	:=		x	x		x		
06	:=	x		x				x
07	:=	x		x			x	
08	:=		x	x				x
09	:=		x	x			x	
10	:=			x	x	x	x	x
11	:=	x		x		x		
12	:=				x	x	x	
13	:=				x		x	x
14	:=			x	x			x
15	:=		x		x			x
16	:=	x			x		x	
17	:=	x			x	x		
18	:=	x	x		x			
19	:=		x	x	x			
20	:=			x		x		x
21	:=	x	x	x	x	x		
22	:=		x			x	x	
23	:=	x				x	x	
24	:=		x			x		x
25	:=	x				x		x
26	:=			x		x	x	
27	:=				x	x		x
28	:=					x	x	x
29	:=		x				x	x
30	:=	x					x	x
31	:=			x			x	x

6.8.1 ERROR CORRECTION CODE (Continued)

Bits 2 and 4 of the ECC portion are inverted after being generated, thus disallowing the case of all zeros (0000000 has a calculated ECC of 00, but when driven out of the the error correction gate array (the ECCIO), the ECC seen is 14). Consequently, these two bits are inverted back in the receiving ECCIO before the syndrome (see following table) is generated.

	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
6 5 4	3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 0 0		*	32	33	+	34	+	+	28	35	+	+	13	+	27	12	+
0 0 1		36	+	+	31	+	20	26	+	+	14	+	+	+	+	+	10
0 1 0		37	+	+	29	+	24	22	+	+	15	+	+	+	+	+	+
0 1 1		+	08	09	+	05	+	+	+	19	+	+	+	+	+	+	+
1 0 0		38	+	+	30	+	25	23	+	+	+	16	+	17	+	+	+
1 0 1		+	06	07	+	11	+	+	+	04	+	+	+	+	+	+	+
1 1 0		+	01	02	+	00	+	+	+	18	+	+	+	+	+	+	+
1 1 1		03	+	+	+	+	+	+	+	+	+	+	+	21	+	+	+

6.8.2 MESSAGE FORMAT

This section specifies the I/O Bus message format. The IOMC is the only module on the I/O Bus that is not microprocessor based (message handling is hardwired, instead of being determined by firmware). Therefore, IOMC messages are described separately.

6.8.2.1 DTM-IOMC MESSAGE FORMAT

IOMC messages consist of Memory Interface messages (reading and writing to memory), XM Interface messages (Interrupts), and IOMC-only messages (Read H/W register and Read Time of Day).

Memory Interface messages to the IOMC have the following format:

```
ccOOOOOOTA
ccOPAAAAAA
ccxHHHLLLL
ccdata
    .
    .
```

Where:

cc =	Valid ECC code for the rest of the word
TA =	Target address (FC for IOMC)
OP =	1 byte hex op code (see following)
AAAAAA =	6 least significant BCD digits of the beginning address of the memory access.
HHH =	3 most significant BCD digits of the beginning address of the memory access.
LLLL =	number of bytes (hexadecimal) of data to be transferred to or from memory.
data =	4 bytes of data per word (if the command is a memory write). If the command is a memory read, no data is sent.

6.8.2.1 DTM-IOMC MESSAGE FORMAT (Continued)

The memory commands are:

RDMEM	- Read Memory	Op = D0
RDMEMT	- Read Memory Translated	Op = D8
RDWLCK	- Read with Lock	Op = F0
WRMEM	- Write Memory	Op = A0
WRMEMT	- Write Memory Translated	Op = A8

XM interface messages to the IOMC have the following format:

```
ccOOOOOOTA
ccTYOOOOON
cc00000000
ccOPAAAAAA
ccxHHHLLLL
ccdata
.
```

Where:

TY	=	Type of interrupt
N	=	Number of first XM to poll (0,1,2, or 3)
cc	=	Valid ECC code for the rest of the word
TA	=	Target address (F0 for IOMC)
OP	=	1 byte hex op code (see following)
AAAAAA	=	6 least significant BCD digits of the beginning address of the memory access.
HHH	=	3 most significant BCD digits of the beginning address of the memory access.
LLLL	=	number of bytes (hexadecimal) of data to be transferred to or from memory.
data	=	4 bytes of data per word (if the command is a memory write). If the command is a memory read no data is sent with the command.

The XM Interrupt types are:

IOCNORM	- I/O Complete Normal	TY = 82 (hex)
IOCREAL	- I/O Complete Real-time	TY = 81 (hex)
IOCEXP	- I/O Complete Exception	TY = 84 (hex)

6.8.2.1 DTM-IOMC MESSAGE FORMAT (Continued)

IOMC-only interface messages have the following format:

```
    cc000000TA
    ccMC000000
```

Where MC is the 1-byte op code. The commands are:

```
    RDTOD    - Read Time-of-Day Counter  Op = MC Hex
    RDMAIL   - Read Mailbox Register     Op = MC Hex
```

The IOMC reply has the following format:

```
     ccdatadata
         .
         .
         .
         .
     ccextra
     ccerrors
```

Where:

```
        data =  is the reply (if any)
       extra =  extra unused word (if any)
      errors =  1 word of error descriptor (if any)
```

The errors field is decoded as follows:

```
          bits          function
          ----    ------------------------------
           3:0  - XMEMERR$$P(3:0) from memory bus
           8:4  - XMEMSRC$$P(4:0) from memory bus
           9:1  - Overflow error on IOMC
          10:1  - ECCIO single bit error
          11:1  - ECCIO double bit error
          12:1  - MDECC 0/1 single bit error
          13:1  - MDECC 2/3 single bit error
          14:1  - MDECC multiple bit error
```

If multiple errors occur in the memory transfers, the most fatal error is logged and reported.

Refer to the IOMC Specification for additional information on the IOMC commands and replies.

6.8.2.2 DTM-SMC MESSAGE FORMAT

Refer to the SMC Specification for information on DTM-SMC messages.

6.8.2.3 MESSAGE ROUTER-DTM MESSAGE FORMAT

Inter-DTM communications have the same basic message format. DTMs do not talk to each other unless one of them is the message router. The message format is:

```
ccU00000TA
ccD00000MY
ccCOMMAND
ccdata
     .
     .
```

Where:

```
          cc = Valid ECC code for the rest of the word
          TA = Target address
          MY = Initiator address (target address if it is
               to be a responder)
     COMMAND = Op code, where:
                Identify = 10000000 (hex)
                Transfer mailbox = 12000000 (hex)
         data = Valid data (if any)
```

Refer to the DTM microprocessor microcode for additional information about these messages and responses.

6.8.3 I/O BUS TIMING DIAGRAMS

ıhis section contains typical timing diagrams for I/O Bus transactions. Figure 6-11 shows the timing for a read operation, in which the initiator sends the op code, and receives data as a reply.

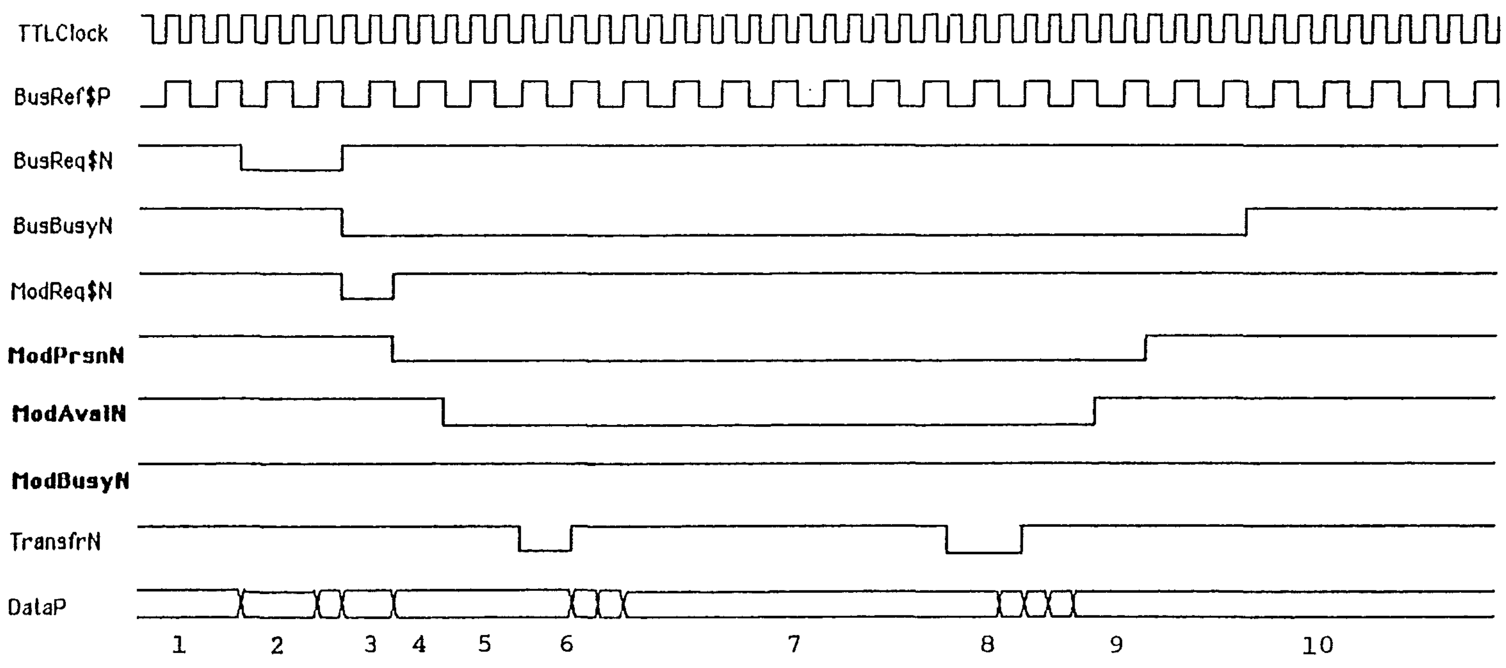

Figure 6-11 I/O Bus Timing - Response

6.8.3 I/O BUS TIMING DIAGRAMS (Continued)

NOTES:

1 Bus not busy

2 Bidding for bus (drive BusReq and Priority)

3 Driving ModReq and the target address

4 Responder is present

5 Responder is available

6 Initiator sends message (OP and address)

7 Initiator waits for response

8 Responder response (three word of reply are shown)

9 Responder does a legal disconnect

10 Initiator disconnects

6.8.3 I/O BUS TIMING DIAGRAMS (Continued)

Figure 6-12 shows a typical write operation, in which the initiator sends the op and data, and the responder indicates that it has performed correctly by performing a legal disconnect without sending a reply.

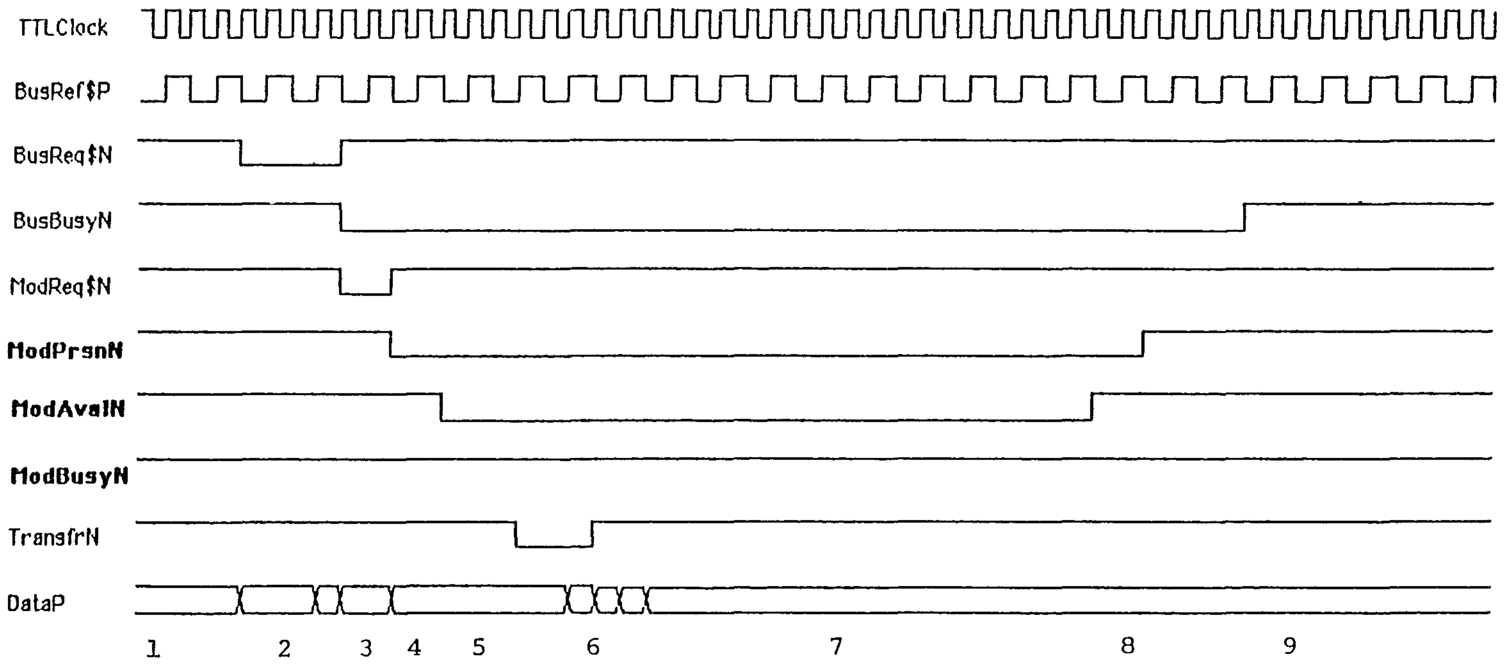

Figure 6-12 I/O Bus Timing - No Response

6.8.3 I/O BUS TIMING DIAGRAMS (Continued)

1 Bus not busy

2 Bidding for bus (drive BusReq and Priority)

3 Driving ModReq and the target address

4 Responder is present

5 Responder is available

6 Initiator sends message (OP, address, and one word of data)

7 Initiator waits for response

8 Responder does a legal disconnect (therfore, no reply)

9 Initiator disconnects

6.8.3 I/O BUS TIMING DIAGRAMS (Continued)

Figure 6-13 illustrates the situation in which the target is present, but is busy. The ModBusy signal is shown delayed, to emphasize that the initiator will wait for either ModPresent or ModBusy before taking any further action.

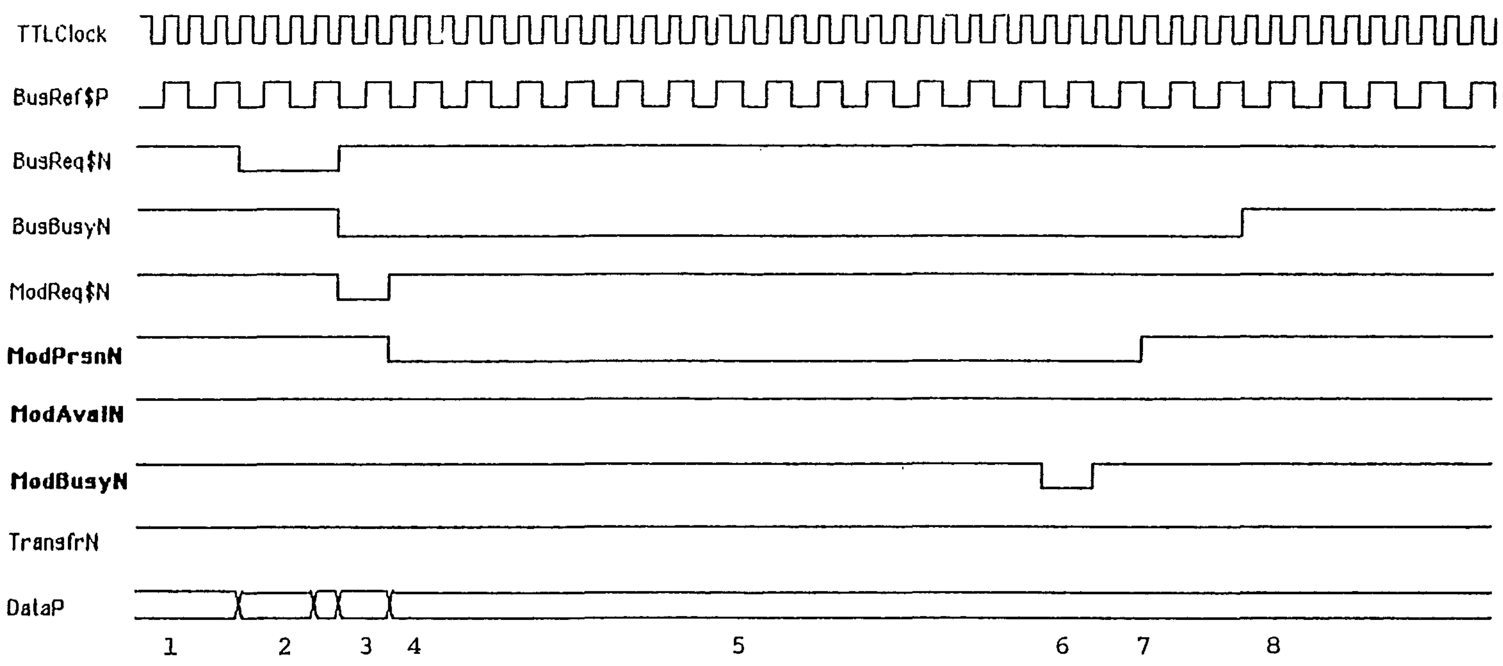

Figure 6-13 I/O Bus Timing - Responder Busy

6.8.3 I/O BUS TIMING DIAGRAMS (Continued)

1 Bus not busy

2 Bidding for bus (drive BusReq and Priority)

3 Driving ModReq and the target address

4 Responder is present

5 Initiator waiting for ModAvail or ModBusy

6 Responder is busy

7 Responder doing a legal disconnect

8 Initiator disconnecting

6.9 MICROPROCESSOR PHYSICAL INTERFACE

This section describes the interface between the I/O Bus-related logic and the microprocessor in the MLI-DTM. The microprocessor interfaces to the buffer, buffer control, and the IOBCs. Each interface is described separately.

6.9.1 BUFFER INTERFACE

The buffer (see Figure 6-14) is a part of the microprocessor's memory. It consists of ten 2149H 1Kx4 static RAMs, plus the address, data, and control lines.

The buffer RAMs may be addressed by the microprocessor, or by the buffer control's address register. Ten address lines are required; bits 11:2 of the microprocessor's address bus are routed through two AM2966 RAM drivers.

In addition, there is address decoding logic associated with the microprocessor that examines several high order address bits, and generates the SelectBusAL signal when the microprocessor wants to address the buffer. This signal, when active (low), tristate disables the buffer control's address register, and tristate enables the AM2966 drivers.

The buffer RAMs connect to 39 bidirectional data lines (32 bits of data, and 7 ECC bits). One RAM bit is unused. The 39 wires are connected to the I/O Bus's ECCIO chip, and to five 74F245 bi-directional driver chips that connect to the microprocessor's data bus. The 74F245's control lines (direction and tristate enable) are sourced from logic associated with the microprocessor.

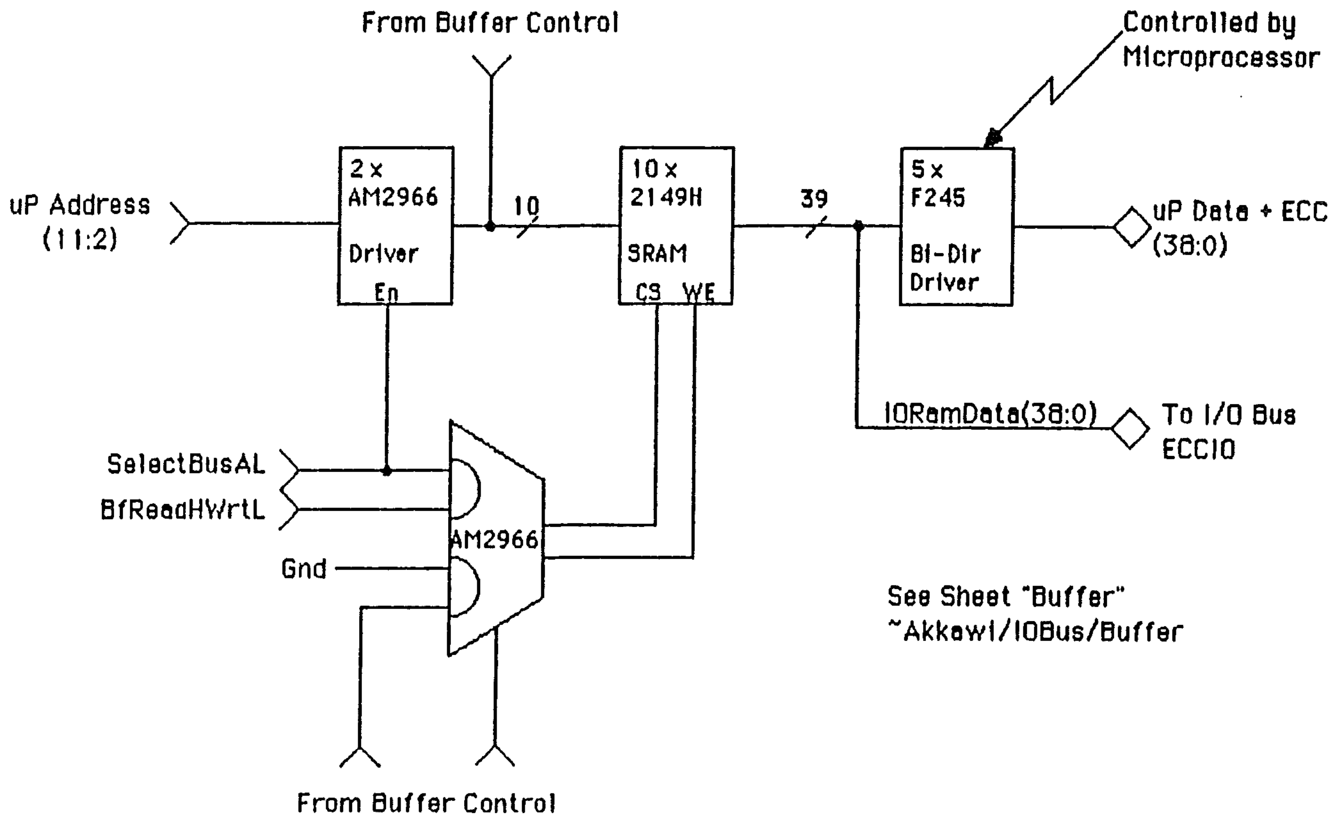

Figure 6-14 Microprocessor - Buffer Interface

6.9.1 BUFFER INTERFACE (Continued)

The 2149H RAMs have Chip Select and Write Enable inputs. A 74F241 used as a multiplexor sources these signals, either from the buffer control, or logic associated with the microprocessor. The source selected is determined by the buffer control. If either IOBC or the MLI-Control wants to access the buffer, the buffer control sources these two lines.

When the I/O Is Not Active signal is true (neither IOBC nor the MLI Control wants access to the buffer), logic associated with the microprocessor must generate the buffer's RAM Write Enable and Chip Select signals.

Note that, if the microprocessor tries to access the buffer while it is being used by an IOBC or the MLI Control, data corruption will occur. The microprocessor will take control of the address lines, the data lines will be double driven, and the RAM control lines will be sourced from the buffer control.

In summary, the microprocessor to buffer physical interface consists of the following signals:

UPADDRESS(11:2)

Purpose	Address in buffer that the microprocessor wants to access.
DTM signal name	BfAdr(11:2)
DTM source	Buffered version of microprocessor's address bus.

UPDATA(38:0)

Purpose	Bidirectional data lines. Bits (38:32) are ECC; bits (31:0) are data.
DTM signal name	IORamData(38:0)

SELECTBUSAL

Purpose	When I/O is not active, Chip Select to buffer RAMs. Tristate enables uPAddress to RAMs and disables buffer control's address counter.
DTM Signal Name	SelectBusAL
DTM Source	High order address decode PAL.

6.9.1 BUFFER INTERFACE (Continued)

BFREADHWRTL

Purpose:	When I/O is not active, Write Enable to buffer RAMs.
DTM Signal Name	BfReadHWrtL1, BfReadHWrtL2 (two copies for loading reasons)
DTM Source	Buffered version of microprocessor's Read/Write signal.

6.9.2 BUFFER CONTROL, IOBCs, AND MLI-DTM INTERFACE

The microprocessor accesses the buffer control, the IOBCs, and (for the MLI-DTM) the MLI control by reading and writing certain memory addresses. Those memory locations are occupied by registers, not RAMs. The use of each register is described in section 6.1 (PROGRAMMER'S MODEL).

Each register is 16 bits wide. Unlike the buffer (and MLI-DTM, and SMC main memory), ECC is not kept for these registers. Hence, the logic associated with the microprocessor must treat the RAM and the registers somewhat differently. Physically, the registers are isolated from the microprocessor's data bus by two 74F245 bi-directional buffers.

The registers reouired for the buffer control, the IOBCs, and the MLI control are all grouped in one address range. Several high order microprocessor address bits are decoded to produce the MliIOBCSelL signal, which when active (low), indicates that the microprocessor wants to access one of the registers. Since there are fewer than 16 registers, the four low order address bits can select the register required.

The register accessed for each combination of the low order address bits is determined by the programming of the control PALs shown in Figure 6-15; the selection is arbitrary.

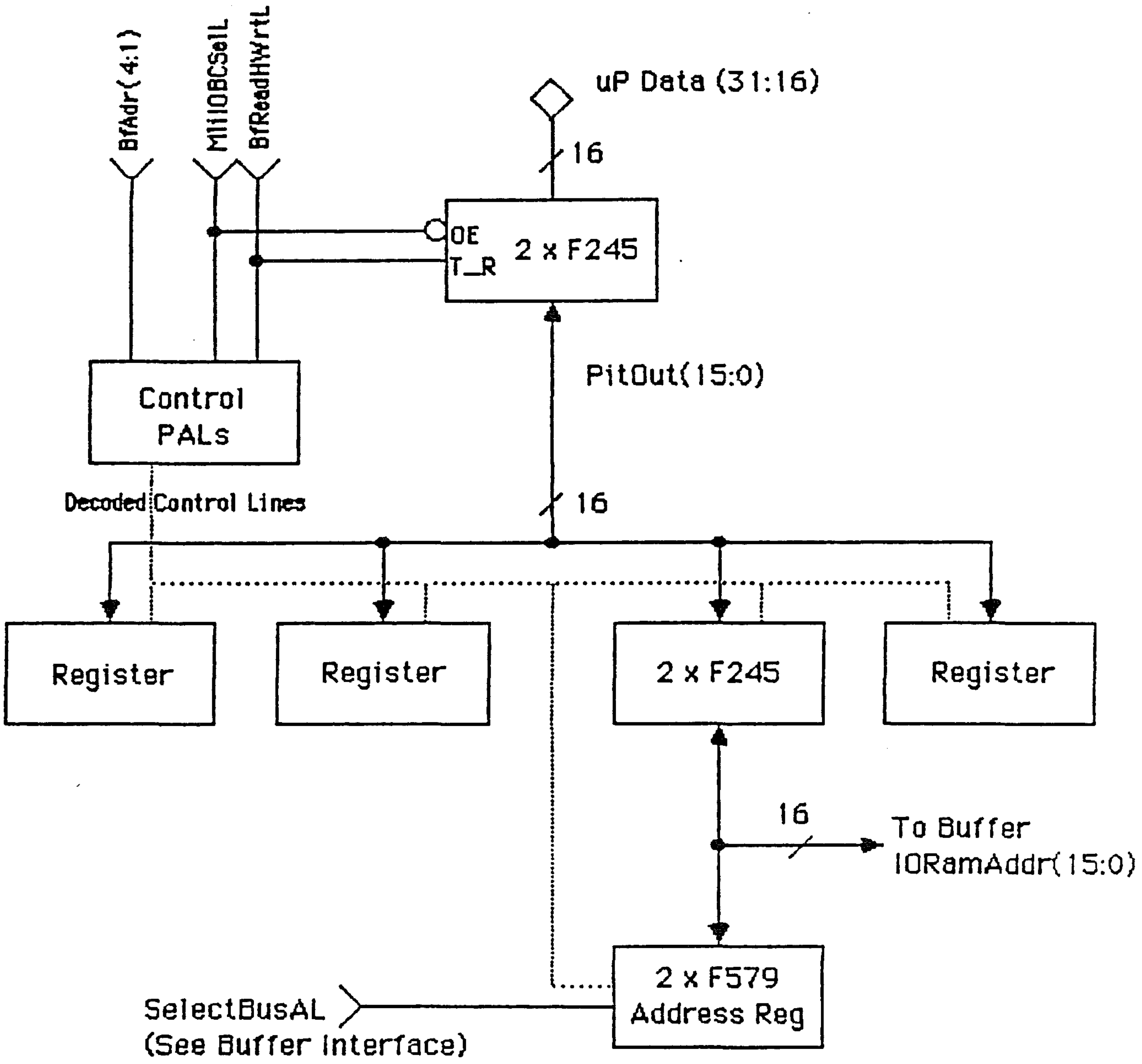

Figure 6-15 Microprocessor to Register Interface

6.9.2 BUFFER CONTROL, IOBCs, AND MLI-DTM INTERFACE (Continued)

The buffer control's address register is isolated from the microprocessor's data bus by an extra pair of 74F245s (see Figure 6-16) because the 74F579s counters used for that register have bi-directional data pins. This does not affect the microprocessor interface.

There are two control lines from the logic associated with the microprocessor to the register control PALs; MLiIOBCSelL, and BfReadHWrtL, which indicates whether the microprocessor wants to read (High) or write (Low) the specified register.

Because the microprocessor and I/O Bus logic in the MLI-DTM are clocked by different clocks, there is a possibility of metastability problems at the interface. This is not a problem for the buffer itself, because RAMs are essentially asynchronous devices. These problems are handled by the register control PALs. If the timings shown in Figure 6-16 are adhered to, there will be no metastability problems.

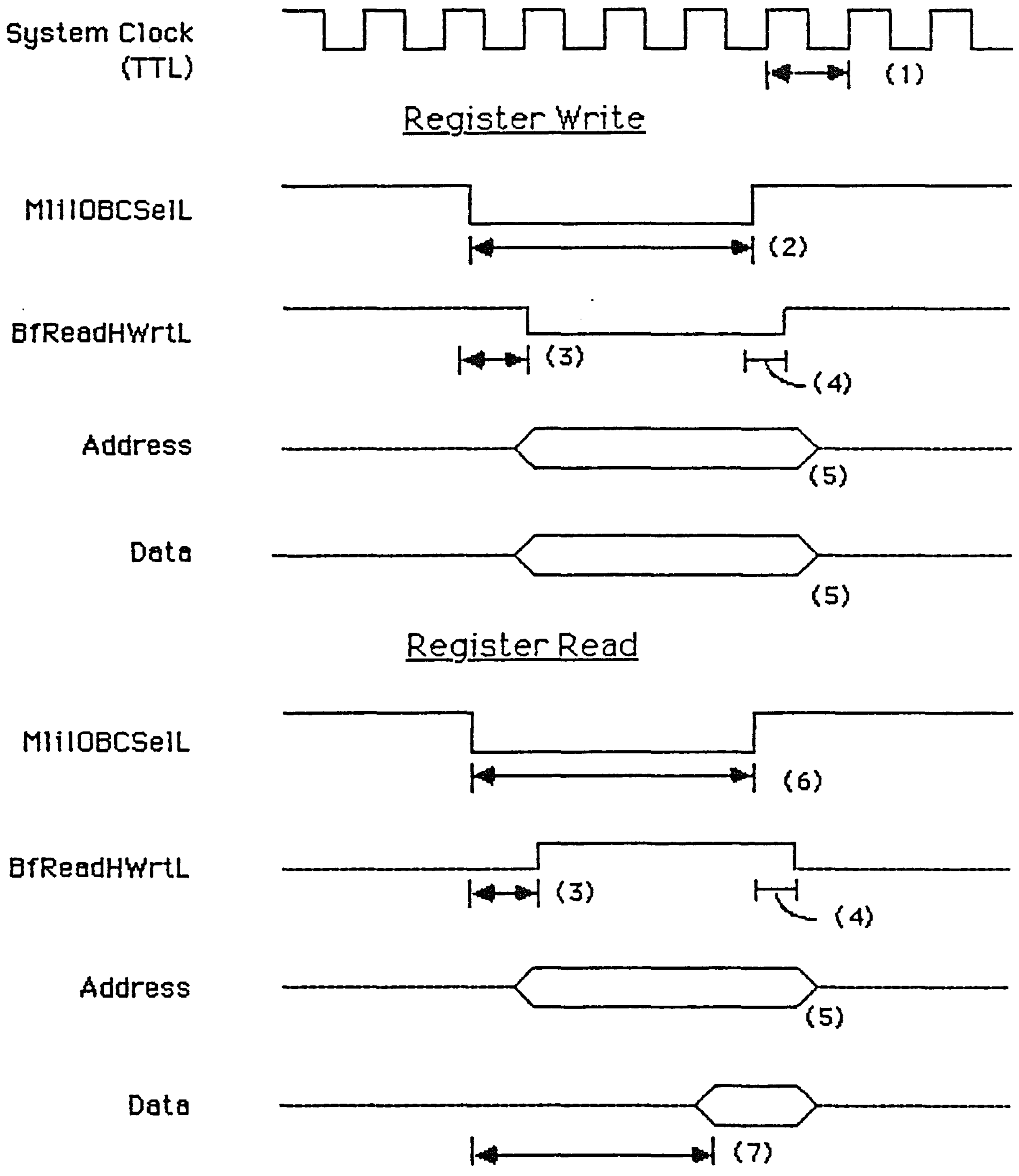

Figure 6-16 Register Access Timing Diagram

6.9.2 BUFFER CONTROL, IOBCs, AND MLI-DTM INTERFACE (Continued)

NOTES:

1 The TTL system clock runs at 96 nanoseconds.

2 During a register write cycle, MliIOBCSelL must be asserted for at least 3.5 system TTL clocks.

3 BfReadHWrtL must be valid no later than one system TTL clock after MliIOBCSelL is asserted.

4 BfReadHWrtL must remain valid at least until MliIOBCSelL is negated (the hold time is 0 ns).

5 Set up and hold time requirements for these signals are the same as for BfReadHWrtL.

6 During a register read cycle, MliIOBCSelL must be asserted for at least 3 system TTL clocks.

7 Data is available 3 system TTL clocks after MliIOBCSelL is asserted. Data remains valid until MliIOBCSelL is negated.

6.9.2 BUFFER CONTROL, IOBCs, AND MLI-DTM INTERFACE (Continued)

In summary, the microprocessor to (buffer control, IOBC, and MLI control) register physical interface consists of the following signals:

uPAddress(4:1):

Purpose	Indicates which register the microprocessor wants to access.
DTM Signal Name	BfAdr(4:1)
DTM Source	Buffered version of microprocessor's address bus.

uPData(31:16):

Purpose	Bidirectional data lines. No ECC.
DTM Signal Name	BusData(31:16)
DTM Source	Microprocessor Data Bus

MliIOBCSelL:

Purpose	Indicates that microprocessor wants to access one of the registers.
DTM Signal Name	MliIOBCSelL
DTM Source	High order address decode PAL.

BfReadHWrtL:

Purpose	Indicates whether the microprocessor wants to read or write the register.
DTM Signal Name	BfReadHWrtL1, BfReadHWrtL2 (Two copies for loading reasons)
DTM Source	Buffered version of microprocessor's Read/Write signal.

6.9.3 BUFFER TO BUFFER CONTROL INTERFACE

The Buffer controller generates the following signals to interface to the buffer, buffer address counter, and buffer limit counter. The signals are:

AIOACTIVEL	I/O Active (active low). This signal indicates that the I/O Bus controller or MLI needs the buffer. This signal enables the following two signals (making them valid).
AIOReceiveL	I/O Receive active low. This signal (if low) indicates that data is being written into the buffer. If high, it indicates that data is being read from the buffer. This signal drives the IORamWEL on the buffer.
RAMCSL	RAM Chip Select active low. This signal drives the IORAMSelectL (chip select) signal on the buffer. It is always low during a read from the buffer (AIOReceiveL = 1). When writing to the buffer (AIoReceiveL = 0), this signal will be a 40 nanosecond low pulse. This pulse, generated through a delay line, falls during the clock period in which both data and address to the buffer are stable.
BusCounterEnL	Buffer Bus Counter Enable active low. This signal increments the buffer address counter. If this signal is low on a rising edge of the clock, the address counter increments by 1.
GiveMeDataL	This signal decrements the limit counter. If this signal is low on the rising edge of the clock, the limit counter decrements by 1.
DoneL	If limit counter is zero, this signal will be low. This signal is not used by the buffer controller; it is passed directly to the I/O Bus controller.
CounterClearL	Counter Clear active Low. This signal resets the address and limit counters when no one owns the buffer.

6.9.4 IOBC - BUFFER CONTROL INTERFACE

This section describes the I/O Bus controller (IOBC), and the IOBC to buffer interface.

The IOBC handles data transfers on the I/O Bus. It is the interface between the I/O Bus control signals and the internal signals of the module (the module being the DTM in this case).

The IOBC is responsible for getting the I/O Bus (when asked to) and initiating a transfer sequence. The IOBC gives a signal that indicates the termination of the transfer, and the reason for the termination (normal termination, bus busy, pre-empted, illegal disconnect).

The IOBC is also responsible for recognizing when the module is being called upon. It asserts the required signals on the I/O Bus, receives the message, and terminates the connection sequence, either directly, or after sending a response back to the initiator.

6.9.4.1 INTERRUPTS

Each IOBC generates its own interrupt signal. This signal is asserted (driven low) when the IOBC wants to cause an interrupt; it remains asserted until explicitly reset by the microprocessor. The microprocessor resets the interrupt by writing to the buffer control's command register, in which there is a bit for each IOBC.

The M68020 generates an Interrupt Acknowledge signal when it accepts an interrupt from a device. This signal does not directly reset the interrupt signal because:

o The interrupt acknowledge, being synchronized with the microprocessor's clock, was not long enough to guarantee that the PAL generating the interrupt signal (which is on system TTL clock) would see it.

6.9.4.1 INTERRUPTS (Continued)

o There are two possible reasons for the IOBC to signal an interrupt: either a message has arrived, or IIOReqL is asserted, thereby indicating that there is something in the IOMC's mailbox register for the message router. The microprocessor must read a status register to determine which type of interrupt occurred. The PAL that generates the interrupt signal does not know when the status register is read.

These problems could be solved differently for the SMC or future DTMs. For example, the microprocessor could see separate interrupts for message arrival and IIOReqL events, and the Interrupt Acknowledge could be synchronized with the system TTL clock. Further, the SMC or a future DTM could choose to be interrupted at different points in the message cycle. Each case would require separate investigation.

The interrupt signal is generated by a PAL16R4, running on system clock. There are several spare inputs and outputs on the PAL; there is substantial flexibility.

The microprocessor (and/or associated logic) must be prepared to handle any metastability problems that occur because the interrupt signal is clocked by the system TTL clock, while the microprocessor runs on a different clock.

In summary, the MLI-DTM implementation has an interrupt line from each IOBC to the microprocessor:

InterruptA (InterruptB):

Purpose	Indicates that IOBC A (IOBC B) requires attention from the microprocessor.
DTM Signal Name	INTLevel5L (IOBC A), INTLevel4L (IOBC B)

6.9.4.2 IOBC DATA PATH

The data path is 39 bits wide (32 data bits, 7 ECC bits). This path is bidirectional, capable of transferring data at TTL system clock speed (one 39 bit word per system TTL clock). The IOBC indicate whether this bus is sending or receiving data. Loading the data into the buffer and checking ECC are are not a function of the IOBC; they are a function of the buffer controller.

The IOBC does not look at the data being transferred on the bus. Therefore, it cannot distinguish the op, length, or address information from data. The IOBC treats all the data in the same way, and transfers the different types of data without interruptions.

When the IOBC is requested to act as an initiator, it expects a valid module-to-talk-to number on the data bus upon demand.

The IOBC data path consists of a set of 74F245 bidirectional drivers that drive or receive data to or from the backplane, and a set of 74F646 bidirectional registers that interfaces the backplane drivers to the internal data path. All data transferred (in either direction) are stored for at least one clock in the 74F646 registers.

6.9.4.3 IOBC INTERNAL CONTROL PATH

The IOBC internal control path can be subdivided into the microprocessor interface and the buffer interface.

The microprocessor interface issues commands and returns results, which the microprocessor can use to decide the next step of action (such as retry if the bus is busy).

The buffer interface is a set of signals that is used by the buffer control to synchronize data transfers with the buffer.

6.9.4.4 BUFFER-IOBC INTERFACE

The Buffer-IOBC interface consists of signals that are used by the buffer controller to handle buffer reads and writes. Following is a listing of these interface signal and their function. Since the DTM has two busses and one buffer, there are two IOBC signals, one for each bus. All IOBC-1 signals have an A as a prefix, while IOBC-2 signals have a B.

AAccReqL	(BAccREQL) ACCess REQuest active Low (cntl-out). This signal is asserted by the IOBC to tell the module that it is being asked to act as a responder, and that the module should grant buffer access to write the input message. Valid responses to AccReq are Busy or AccGrnt. This signal remains asserted until a Busy is received, or after the reply is sent to the initiator.
AAccGrntL	(BAccGrntL) ACCess GRAnted active Low (cntl-in). This is an answer to AccReqL; it tells the IOBC that it has access to the buffer.
ABuffReqL	(BBuffReqL) Buffer Request active Low (cntl-out). This signal indicates that the IOBC wants to use the buffer to send or receive a message or reply. In send mode (sending data from the module to the I/O Bus), it is asserted with GiveMeData, and reset three clock after GiveMeData is reset. In receive mode, it is asserted with HereIsData, and reset one clock after HereIsData is reset. That is, it is asserted during the time that the internal data path is active.
ASndLRcvH	(BSndLRcvH) Send If Low Receive If High (cntl-out). This signal indicates whether the IOBC is in the send or receive phase of a connection. This signal has the same timing as BuffReq in the send phase (driven low during this time), and is high when receiving or idling.

6.9.4.4 BUFFER-IOBC INTERFACE (Continued)

AHereIsDataL	(BHereIsDataL) Here Is Data active low (cntl-out). The IOBC asserts this signal one clock before valid data is placed in the IOBC data bus register (from the I/O Bus data line), so a module can set up for writing into the buffer. This signal is reset when the last data is in the IOBC register.
AGiveMeDataL	(BGiveMeDataL) Give Me Data active low (cntl-out). This signal indicates to the module that the IOBC is expecting valid data on the data bus to transfer to the I/O Bus. This signal is asserted two clocks before data is expected on the bus, and is reset when DoneL is asserted.
DoneL	Done active low (cntl-in). DoneL is expected by the IOBC two clocks before the last valid data is placed on the IOBC internal data bus (including error words, if any).
PreEmptL	Pre-empt active low (cntl-in). This signal is generated by the buffer controller when the MLI or IOBC is requesting use of the buffer while the buffer is owned by the microprocessor. The IOBC samples this signal during the bus bidding operation, and if set, terminates its bidding, giving a pre-empt condition as the result of the termination.
AModNoL	(BModNoL) Module Number Request active low (cntl-out). This indicates to the buffer that the IOBC is the initiator, and that it expects a valid module to-talk-to number to remain on the internal data bus for as long as this signal is asserted. This signal is asserted one clock before ModReq, and is reset with ModReq.

6.9.4.4 BUFFER-IOBC INTERFACE (Continued)

ABusReqL	(BBusReqL) Bus Request active low (cntl-out). This signal indicates that the IOBC has been asked to act as an initiator, and has started to bid for the bus. This signal is a backplane interface signal that is also used by the buffer controller to load the module to-talk-to number in its register (the ECCIO), so it is ready when the ModNo signal is asserted.

The buffer controller uses this signal to organize the reading and writing of the buffer. It also generates the signal to increment the buffer address counter and decrement the limit counter. (decrementing of the limit counter is done only when data is being taken out of the buffer). The interface signal to the buffer are described in Section 6.9.3 (BUFFER TO BUFFER CONTROL INTERFACE).

6.10 FUTURE DTMs

Future DTMs are constrained only by the backplane definition of the I/O Bus. A future implementation may choose to use all, some, or none of the current implementation. This note is intended simply to outline some of the possibilities.

In a future DTM (see Figure 6-17), the microprocessor might want to be interrupted at different points in the I/O Bus protocol. For example, the MLI-DTM and waits for an I/O Bus transaction to complete by polling the appropriate IOBC's status register. The microprocessor has nothing better to do at that time. The SMC has adopted the same strategy, not because there is nothing better to do, but because the overhead in putting the task to sleep, and then waking it up again in response to an interrupt is about the same as the time spent polling. A future DTM, with a faster microprocessor, might indeed be able to perform useful work while the I/O Bus transfer is in process. The required hardware change is trivial.

A future DTM might also choose to change the buffering scheme: perhaps allowing multiple buffers, or partitioning a single larger buffer. The current IOBC design does not care what sort of buffer (or how many of them) it is attached to. For example, to support multiple buffers, the Buffer control must:

a Allocate a buffer to an IOBC when it asserts Access Request. Access Granted to the IOBC indicates that a buffer is available.

b Indicate to the microprocessor which buffer is allocated to an IOBC, so that an incoming message can be found after an interrupt.

c Ensure that, after a 'Start' command, the IOBC is given access to the appropriate buffer when it asserts Buffer Request.

d Ensure that, after a 'Continue' command, the IOBC is given access to the buffer it owns when it asserts Buffer Request.

The notions of buffer ownership and loaning buffers carry over directly to the multiple buffer case: each buffer may be owned or loaned in precisely the same fashion as in the single buffer case.

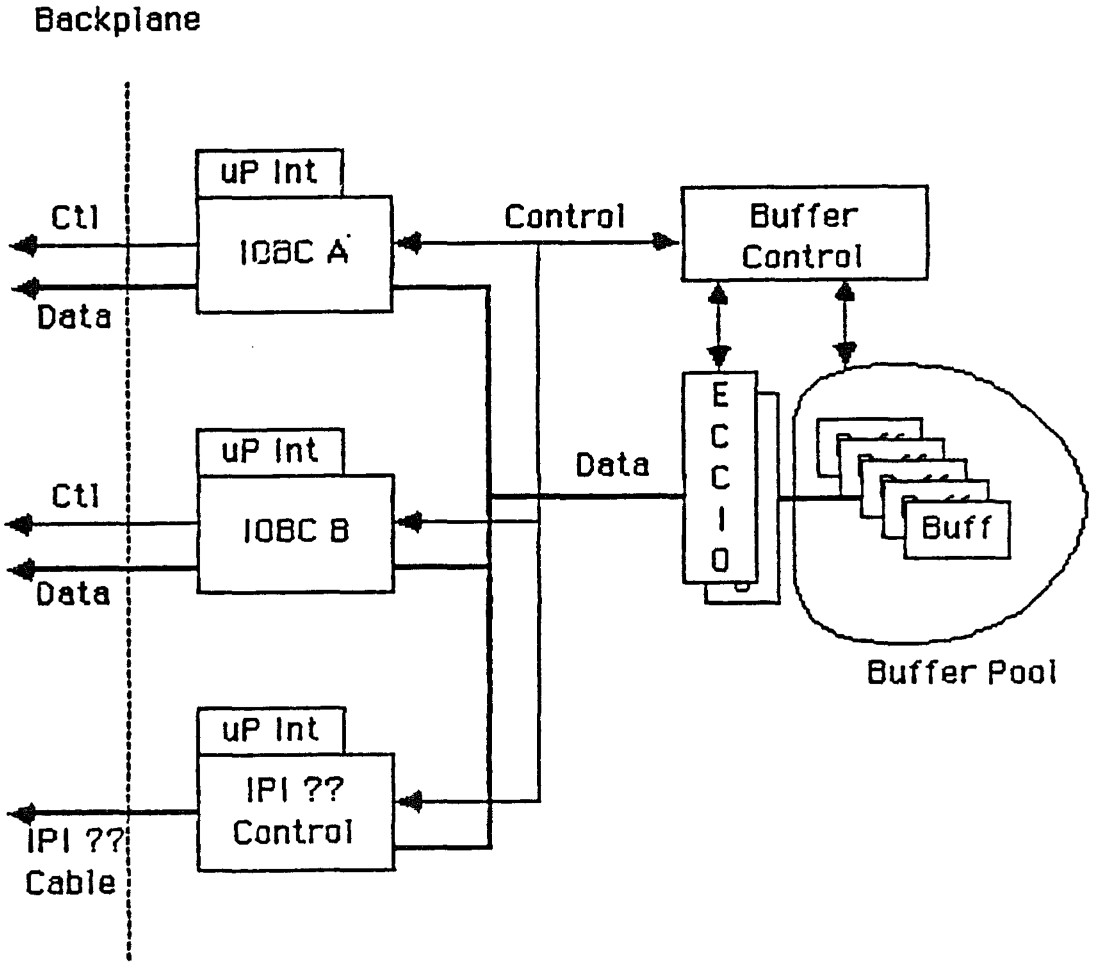

Figure 6-17 Future DTM Structure

7 ERROR DETECTION AND HANDLING

This section describes the error detection capabilities of the DTM, and the way they are currently being handled.

Errors are classified as being fatal or nonfatal. Fatal errors are those detected by the hardware checking logic that force the BROKEN signal to be asserted to the SMC. Fatal errors stop the clock on the board to prevent any further corruption to the source of the error. Examples of such fatal errors are the parity checking logic on the MLI control store, or the parity checkers in the I/O Bus controller.

Nonfatal errors are errors that are either directly corrected, such as single bit errors on the I/O Bus data transfer, or errors in operations that can be retried, such as multiple bit errors on the I/O Bus data transfer.

7.1 DTM FATAL ERRORS

The DTM reports fatal errors to the SMC by setting bits in the MODBROKEH register, which causes the JMODBROKE$$N signal to the SMC be asserted on the backplane (if the module is on-line). Because all cards in the I/O portion on the backplane share the same JMODBROKE$$N signal, MODBROKEH is also registered on the maintenance chain. This allows the SMC to isolate the broken module without stopping the clocks.

The MODBROKEH register on the maintenance chain is called CMODBROKE. The CMODNOTBRK bit in the maintenace register (the registered value of MODNOTBROKEH signal) covers the case of when the maintenance chain is shifted right while self testing is being done, and both MODBROKEH and MODNOTBROKEH are being toggled simultanously.

After identifying the broken module, clocks are stopped to that module, the data chain is shifted out, and MODBRKCTRL register is checked.

7.1 DTM FATAL ERRORS (Continued)

The MODBRKCTRL register consists of the following bits:

bit name	function
CMODBROKEH	Driver of MODBROKEH signal
CMODNOTBRK	Driver of MODNOTBROKEH signal
CCTLERROR	Error indicator for buffer controller and MLI
CIGNOREERR	Test error indicator for buffer ctl and MLI
BBUSERROR	Error indicator for IOBC B
BBUSIGNERR	Test error indicator for IOBC B
ABUSERROR	Error indicator for IOBC A
ABUSIGNERR	Test error indicator for IOBC A

Although CMODBROKEH and CMODNOTBRK exist on the data and maintenace chains, the bits are different. The maintenance chain bits are the registered values of the data chain bits.

When broken MODBRKCTRL register should contain one of the following values:

MODBRKCTRL	source of error
A0	Buffer controller or MLI
88	I/O Bus controller B
82	I/O Bus controller A

Now the error source is narrowed to a given module on the board. To find the portion of the board that detected the error, the following registers should be checked:

1 Case of buffer controller or MLI (MODBRKCTRL = A0)

register	active	source
CMLIRAMPAR	high	MLI control store
CMLIINTPAR	low	MLI internal parity
CTIMEOUTDS	low	(when low validates CINTLEVEL7)
CINTLEVEL7	low	uP is mulfunctioning (if CTIMEOUTDS is low)
CERROR2RCV	low	uP detected fatal error
CERROR2RCV	low	Buffer ownership
CERROR2RCV	low	Buffer CNTL PAL

7.1 DTM FATAL ERRORS (Continued)

2 Case of IOBC B (MODBRKCTRL = 88)

register	active	source
BBUSNONERR	high	The bus monitor PALs (for IOBC B) BusCheck1 or BusCheck2
BIERROR6	high	Target Address decoder PROM (IOBC B)
BIERROR5	low	IOBC B LIS2 PAL
BIERROR4	low	IOBC B LIS1 PAL
BIERROR3	low	IOBC B BID2 PAL
BIERROR2	low	IOBC B BID1 PAL
BIERROR1	low	IOBC B DATA PAL
BIERROR0	high	IOBC B MASTER PAL

3 Case of IOBC A (MODBRKCTRL = 82)

register	active	source
BBUSNONERR	high	The bus monitor PALs (IOBC A) BusCheck1 or BusCheck2
BIERROR6	high	Target address decoder PROM (IOBC A)
BIERROR5	low	IOBC A LIS2 PAL
BIERROR4	low	IOBC A LIS1 PAL
BIERROR3	low	IOBC A BID2 PAL
BIERROR2	low	IOBC A BID1 PAL
BIERROR1	low	IOBC A DATA PAL
BIERROR0	high	IOBC A MASTER PAL

When the source of the error is a PAL, the error might be due either to the PAL malfunctioning, or to the PAL detecting an illegal condition on its input.

Path tests can diagnose this error, and determine if the error was transient (making it possible to reset the board and put it back on line), or fatal, thus keeping the board offline.

7.2 DTM NONFATAL ERRORS

DTM nonfatal errors consist of single and multiple bit errors on the data, timeout errors, and illegal responses to the microprocessor.

The I/O Bus data transfers and the microprocessor memory are protected by a single bit error correction, double bit error detection code.

Single- and multibit errors detected on I/O Bus transactions are reported to the microprocessor using the ABUSSTAT and BBUSSTAT registers. SBE need only be logged; for multibit errors, the operation must be retried. The microprocessor microcode controls the number of retries, and the action taken if the operation continues to fail.

See the microprocessor microcode for timeout values and illegal responses. The following text describes a microprocessor timeout on an I/O Bus initiation.

The microprocessor times out on a Start I/O Bus command after .6 milliseconds. If, after this amount of time has elapsed and no OpComplete is detected, the BusStat should be read, read two consecutiv results match. This result is logged. The microprocessor then resets the I/O Bus controller by issuing a Reset command, and the operation is retried. If detailed analysis on the logged BUSSTAT value is needed, refer to the equations for the BID1 and BID2 PALs.

8 MAINTENANCE

The DTM has a set of Pathtests that can test, as well as bring up, the I/O Bus controller and DTM controller sections. There are self-diagnostics for the microprocessor that check its functionality.

Stop conditions can be set on the DTM to assist in diagnosing errors. The stop conditions and test options are described in the following text.

8.1 DTM STOP LOGIC

The DTM has several stop condition options. These options deal mainly with the MLI. The stop conditions can be used separately or in AND or OR combinations. AND'ed options can assert the STOP_AND signal on the backplane. OR'ed options assert the STOP_OR signal on the backplane. In addition, the DTM can perform a self stop on any of the options (self stop means to stop the clock internally before SMC).

The options are:

stop on MLI STATUS = n
stop on MLI ERROR
stop on MLI EMERGENCY INTERRUPT
stop on MLI NORMAL INTERRUPT
stop on EXTERNAL TRIGGER

These functions are as follows:

MLI STATUS=n	Compares MLI's four status lines with n, where n is a user-specified single digit hex number. The condition is met when status is equal to n.
MLI ERRORS	The condition is met if a MLI vertical parity error, a MLI LPW error, or an illegal MLI status transition occurs.
MLI EMERGENCY INTERRUPT	This condition is met when the MLI requests an emergency interrupt.
MLI NORMAL INTERRUPT	This condition is met when the MLI requests a normal interrupt (normal poll test).
EXTERNAL TRIGGER	This condition is met when an external strobe is detected on the frontplane trigger pin. There is a choice of using an active high or an active low signal, as well as whether the strobe input is used directly or latched before using.

8.2 SETTING STOP CONDITIONS

STOP OR

MLI Status = n
1. Set REG YDOCPSTAT to (F-n) [complement of n]
2. Set REG YDOSTCOREN = 1

MLI Errors
1. Set REG YDOANYERROREN = 1

MLI Emergency Request
1. Set REG YDOEMERGOR = 1

MLI Normal Request
1. Set REG YDOINTOR = 1

EXTERNAL TRIGGER

1. Set YDOFPPINV and YDOFPLLATCH as follows:

```
   0  0 for triggering on unlatched active high level.
   0  1 for triggering on   latched active high level.
   1  0 for triggering on unlatched active low  level.
   1  1 for triggering on   latched active low  level.
```

2. Set the YDOFPPOREN register to 1

These conditions can be set separately or in combination

To enable the internal self-stop on any of the preceding OR conditions, (separately or jointly) set REG YDOSLFSTPEN = 1.

8.2 SETTING STOP CONDITIONS (Continued)

STOP AND

```
MLI STATUS = n  1. Set REG YDOCPSTAT to (F-n)
                2. Set REG YDOSTCANDEN = 1
                3. Set REG YDOSTOPANDL = 1
                4. Set REG YDOSTOPANDH = 0

MLI ERRORS      1. Set REG YDOANYERANDEN = 1
                2. Set REG YDOSTOPANDL = 1
                3. Set REG YDOSTOPANDH = 0

MLI EMERG REQ.  1. Set REG YDOEMERGAND = 1
                2. Set REG YDOCOMANDEN = 1
                3. Set REG YDOSTOPANDL = 1
                4. Set REG YDOSTOPANDH = 0

MLI NORM  REQ.  1. Set REG YDOINTAND = 1
                2. Set REG YDOCOMANDEN = 1
                3. Set REG YDOSTOPANDL = 1
                4. Set REG YDOSTOPANDH = 0
```

EXTERNAL TRIGGER

```
1. Set REG YDOFPPINV and REG YDOFPLLATCH to:
     0  0 for triggering on unlatched active high level .
     0  1 for triggering on   latched active high level .
     1  0 for triggering on unlatched active low  level .
     1  1 for triggering on   latched active low  level .
2. Set REG YDOFPPANDEN = 1
3. Set REG YDOCOMANDEN = 1
4. Set REG YDOSTOPANDL = 1
5. Set REG YDOSTOPANDH = 0
```

These conditions can be set separately or in combination.

To enable an internal self-stop on any of the AND conditions (separately or jointly), set REG YDOSLFSTPEN to 1 (same as that for OR condition, if self stop is to be enabled stop will occur on the first stop condition whether it is an AND or an OR).

8.2 SETTING STOP CONDITIONS (Continued)

When resetting the previous conditions, it is better to reset the bits individually, rather than resetting the whole chain. It is also preferable to perform all the setting (and resetting) of the bits simultanously, in one shift operation.

I/O Bus stops can occur in the IOMC. Refer to the IOMC specification for details.

The microprocessor does not know that the clocks are stopped. Any writes to the interface registers (which require the clock to be running) are not done, although the microprocessor microcode thinks that they are.

The microprocessor WRITE SELF STOP option can cause a STOP_OR condition with an automatic self stop. If set, this option causes the DTM to perform a self stop and assert the OR STOP condition whenever the microprocessor performs a write to any of the I/O Bus, MLI, or buffer registers.

To set up UP_WRITE_SELFSTOP, set REG YDOCUPSTOPEN = 1

This condition is independent of the other conditions.

8.3 TEST OPTIONS

This section describes test options used in the DTM pathtest to facilitate testing. There is a set of test options on each I/O Bus controller, and one on the buffer controller. These options are described in the following text.

8.3.1 IOBC TEST OPTIONS

Bits 0 and 1 on each of the IOBCs control the I/O Bus controller data bus. The bits are ATestMode(1:0) for bus A, and BTestMode(1:0) for bus B. Their functions are described in the following table.

xTestMode(1:0)	Description
00	Normal case
01	Like normal case except that the I/O Bus backplane drivers will not be disabled if the clock to the board are stopped.
10	Disable F245 backplane data drivers, but enable the F646 registers to drive toward the backplane.
11	Disable F245 backplane data drivers, but enable the F646 registers to drive toward the ECCIO.

These test modes can be used to test the F646s, or to probe the backplane when clocks are stopped.

8.3.2 BUFFER CONTROLLER TEST OPTIONS

The CTestMode(2:0) bits help in testing the I/O Bus internal data path. The lower two bits help in ECCIO control, and the upper bit forces the buffer control PAL to remain in a given state regardless of other inputs.

CTestMode(1:0)	Description
11	Normal case
01	Force the ECCIO to select data from the RAM (when loading) with all of its data drivers disabled.
10	Force the ECCIO to select data from the backplane (F646 registers) when loading with drivers toward the RAM enabled.
00	Force the ECCIO to select data from the backplane (F646 registers) when loading with its data drivers disabled.

CTestMode(2)	Description
1	Normal case
0	Forces the control PAL into a special mode, in which the upper five bits remain unchanged, while the lower three will be high. This PAL controls reading and writing into the buffer as well as incrementing the address counter.

9 HARDWARE OVERVIEW AND IMPLEMENTATION

The DTM microprocessor section consists of the following sections:

Microprocessor
Microprocessor address bus decoder
Microprocessor data path
Microprocessor data bus controller
Resident program memory
Data scratchpad
Data scratchpad refresh
Data buffer
MLI control registers
I/O Bus control registers
SMC interface register
Reset and interrupt controller

The microprocessor fetches and executes microcode stored in resident memory, and stores internal data structures in the scratchpad. The microprocessor uses the I/O Bus control registers to access I/O Bus A and I/O Bus B to communicate with other modules on the I/O busses. The microprocessor also uses the MLI control register to communicate with the MLI control store during data transfer.

9.1 MICROPROCESSOR

The microprocessor is the central processing unit, and handles the management of I/O operations. The microprocessor can access the resident program memory, data scratchpad, data buffer, I/O Bus control, MLI control registers, SMC interface registers, and watchdog timer registers.

The microprocessor fetches and executes instructions from the resident memory, and transfers computed data to and from the data scratchpad. The microproessor interfaces with the I/O Bus controller and MLI controller through I/O Bus control registers and MLI control registers, respectively.

The microprocessor section with implemented with a single chip MC68020R16 microprocessor. The microprocessor has its own clock, which runs asynchronously to the system clock.

The microprocessor clock is 16.67 MHz, which is derived from a 33.33 MHz oscillator with divide-by-two circuitry. It is buffered with a 74F241 clock driver.

The microprocessor address and control busses are buffered with four 74F241 line drivers, and the data bus is buffered with four bidirectional 74F245 transceivers.

9.2 MICROPROCESSOR ADDRESS BUS DECODER

The microprocessor address bus is 32 bits wide, and can address 4096K bytes of memory (see Figure 9-1). The address bus decoder decodes the microprocessor address bus and enables the component addressed by the microprocessor.

The microprocessor address bus is decoded in two levels. The first level is implemented with a PAL16L8B programmable array. The PAL level decodes the memory range occupied by the resident program memory, data scratchpad, data buffer, maintenance register, timer registers, I/O Bus control registers, and MLI control registers. The second level is implemented with two PAL16L8Bs and one PAL16R4B. This second level decodes individual registers in the memory range occupied by the I/O Bus control registers and MLI control registers.

```
address       size        component
-------       ----        ---------

             ------------
00000000 |              |
         |   16K x 32   | Resident program memory
0000FFFF |______________|
         |              |
         |______________|
00200000 |              |
         |  512K x 32   | Data scratchpad
003FFFFF |______________|
         |              |
         |              |
         |______________|
00410000 |     1 x 32   | Maintenance register
         |______________|
         |              |
         |______________|
00430000 |     1 x 16   | Buffer command register
         |______________|
00430002 |     1 x 16   | Buffer status register
         |______________|
00430004 |     1 x 16   | MLI result register
         |______________|
00430006 |     1 x 16   | MLI command register
```

9.2 MICROPROCESSOR ADDRESS BUS DECODER (Continued)

address	size	component
00430008	1 x 16	MLI status register
0043000A	1 x 16	MLI identification register
0043000C	1 x 16	MLI high byte counter
0043000E	1 x 16	MLI low byte counter
00430014	1 x 16	I/O buffer address register
00430016	1 x 16	I/O buffer limit register
00430018	1 x 16	I/O Bus-A status register
0043001A	1 x 16	I/O Bus -B status register
0043001C	1 x 16	I/O Bus-A command register
0043001E	1 x 16	I/O Bus-B command register
00600000		Data buffer
	1K x 32	
00600FFF		

Figure 9-1 Microprocessor Memory Map

9.3 MICROPROCESSOR DATA PATH

The microprocessor can access the resident program memory, the data scratchpad, and the data buffer via a 32 bit data path, or via a second path consisting of 32 data bits with 7 additional bits for error detection and correction (a modified Hamming code is used).

In a microprocessor read cycle from data scratchpad or data buffer, data is read through the first data path. The second data path is also enabled to check the error detection and correction code.

If a multiple bit error or single bit error is detected, the microprocessor read cycle is aborted, and is handled by bus error exception.

During microprocessor writes to the data scratchpad or data buffer, the second data path is enabled, and generates the error detection and correction code. During reads or writes to the registers, the first data path is enabled since the registers do not have error detection and correction capabilities.

The first data path is implemented with four 74F373 transparent latches and four bidirectional 74F245 transceivers. The second data path is implemented with one ECCIO gate array, four 74F245 bidirectional transceivers, four 74F241 line drivers, and four AM2966 drivers.

9.4 MICROPROCESSOR DATA BUS CONTROLLER

The microprocessor data bus controller is the heart of the microprocessor section. It is implemented as two level state machines.

The first level state machine interprets the microprocessor address and control busses, and determines the type of the microprocessor bus cycle.

The bus cycle is classified as one of the following:

refresh cycle
read cycle
read retry cycle
read modify write cycle
write cycle
interrupt acknowledge cycle

The bus cycle can have either 1, 2, 6, or 8 wait states.

The access device is either 8-bit, 16-bit, or 32-bit.

The second level state machine interprets the signals from the first level state machine and the address decoder. It then enables the proper data path to the accessed device.

9.5 RESIDENT PROGRAM MEMORY

The resident program memory is the firmware store from which the microprocessor fetches instructions and immediate operands. The microprocessor can read this resident program memory in byte, word, or longword mode at byte, word, or longword boundary.

The resident program memory (see Figure 9-2) is implemented on five 27S128 16Kx8 EPROM chips. Four of them contain instructions and immediate operands, and the fifth one contains the error detection and correction code.

```
                 error
               correction
                  code        ---------- data -----------

    address      38-32      31-24    23-16    15-08    07-00
                 -------    ------------------------------------
    000000      |       |  |        |        |        |        |
       .        |  ROM  |  |  ROM   |  ROM   |  ROM   |  ROM   |
       .        |   4   |  |   3    |   2    |   1    |   0    |
    00FFFF      |_______|  |________|________|________|________|
```

Figure 9-2   Resident Program Memory

9.6 DATA SCRATCHPAD

The data scratchpad is an internal data store in which the microprocessor stores and retrieves its temporary data. The microprocessor can either read or write this data scratchpad in byte, word, or longword mode at byte, word, or longword boundaries.

The data scratchpad is implemented with ten dual 256Kx4 DRAM chips. As shown in Figure 9-3, these DRAMs are organized to form two banks, each of which is 256Kx40.

```
              error
           correction                         data
              code
          38-36 35-32   31-28 27-24 23-20 19-16 15-12 11-08 07-04 03-00
address   ___________   _______________________________________________
200000   |     |     | |     |     | memory bank #1  |     |     |     |
         | RAM | RAM | | RAM | RAM | RAM | RAM | RAM | RAM | RAM | RAM |
         |  9  |  8  | |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
2FFFFF   |_____|_____| |_____|_____|_____|_____|_____|_____|_____|_____|
300000   |     |     | |     |     | memory bank #2  |     |     |     |
         | RAM | RAM | | RAM | RAM | RAM | RAM | RAM | RAM | RAM | RAM |
         |  9  |  8  | |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
3FFFFF   |_____|_____| |_____|_____|_____|_____|_____|_____|_____|_____|
```

Figure 9-3 Data Scratchpad

9.7 DATA SCRATCHPAD REFRESH

The data scratchpad dynamic memory must be refreshed at 4 millisecond intervals. The refresh is controlled by two 8-bit 74F579 bidirectional binary counters, a 74F240 8-bit tristate driver, and a PAL16R4 programmable array. The 74F579s are clocked by the microprocessor clock; they signal the PAL16R4B to refresh at 4 millisecond intervals. The PAL16R4B then requests ownership of the microprocessor address bus. When ownership is granted, the data bus controller recognizes the refresh cycle and enables the 74F240 and 74F579s to drive the refresh address on the microprocessor address bus.

9.8 DATA BUFFER

The data buffer is a temporary data store for transfers between the microprocessor, I/O Bus control, and MLI control sections. Ownership of the data buffer is controlled by the buffer ownership control of the I/O Bus control section.

9.9 IOBUS And MLI CONTROL REGISTERS

The IOBUS and MLI control registers control and monitor the activities of the I/O Bus and MLI. Refer to Section (I/O BUS) and Section (MLI CONTROL) for details.

9.10 SMC INTERFACE REGISTER

The SMC interface register is part of the SMC interface chain. It handles communications between the microprocessor and the SMC interface controller. The SMC interface controller shifts in the command into the SMC interface register, then sets the Interrupt flip-flop in the SMC interface chain.

The interrupt causes the microprocessor to read and execute the command, and to write the result back to the SMC interface register. The SMC interface controller then shifts the result from the SMC interface register and resets the Interrupt flip-flop.

The SMC interface register is implemented with eight PAL16R4B programmable arrays that are driven by the system clock when in shift mode, and by the microprocessor clock when in microprocessor access mode. The output of these devices is displayed on 8 LED packages.

9.11 RESET AND INTERRUPT CONTROLLER

The maintenance processor resets the microprocessor section by clearing the data chain.

The microprocessor can be interrupted by the following external events at different priority levels:

SMC maintenance command at level 6
IOBUS-A attention at level 5
IOBUS-B attention at level 4
MLI poll request at level 2

The interrupt controller is implemented with a PAL16R4 programmable array that detects external events and generates interrupts, and a PAL16L8 array that acknowledges the external event when the interrupt is processed.

10 MESSAGE LEVEL INTERFACE

10.1 MESSAGE LEVEL INTERFACE CONTROLLER

The Message Level Interface Controller (see Figure 10-1) is an interface between the DTM's internal modules, and the DLP base module. The DTM can support four MLI ports; each port can handle up to 8 DLPs. A DTM can communicate with up to 32 DLPs in 4 base modules.

The controller follows MLI protocol, including parity generation and checking, external event synchronization, and data transfer.

The function of MLI controller is to establish connection with a DLP, and to provide the strobes and control signals for the transfer of data between the DLP base and the DTM internal modules.

Data transfers within the DTM are through a 1K X 39 bit buffer (32 data bits, plus 7 ECC bits) implemented on ten 1K X 4 static RAMs.

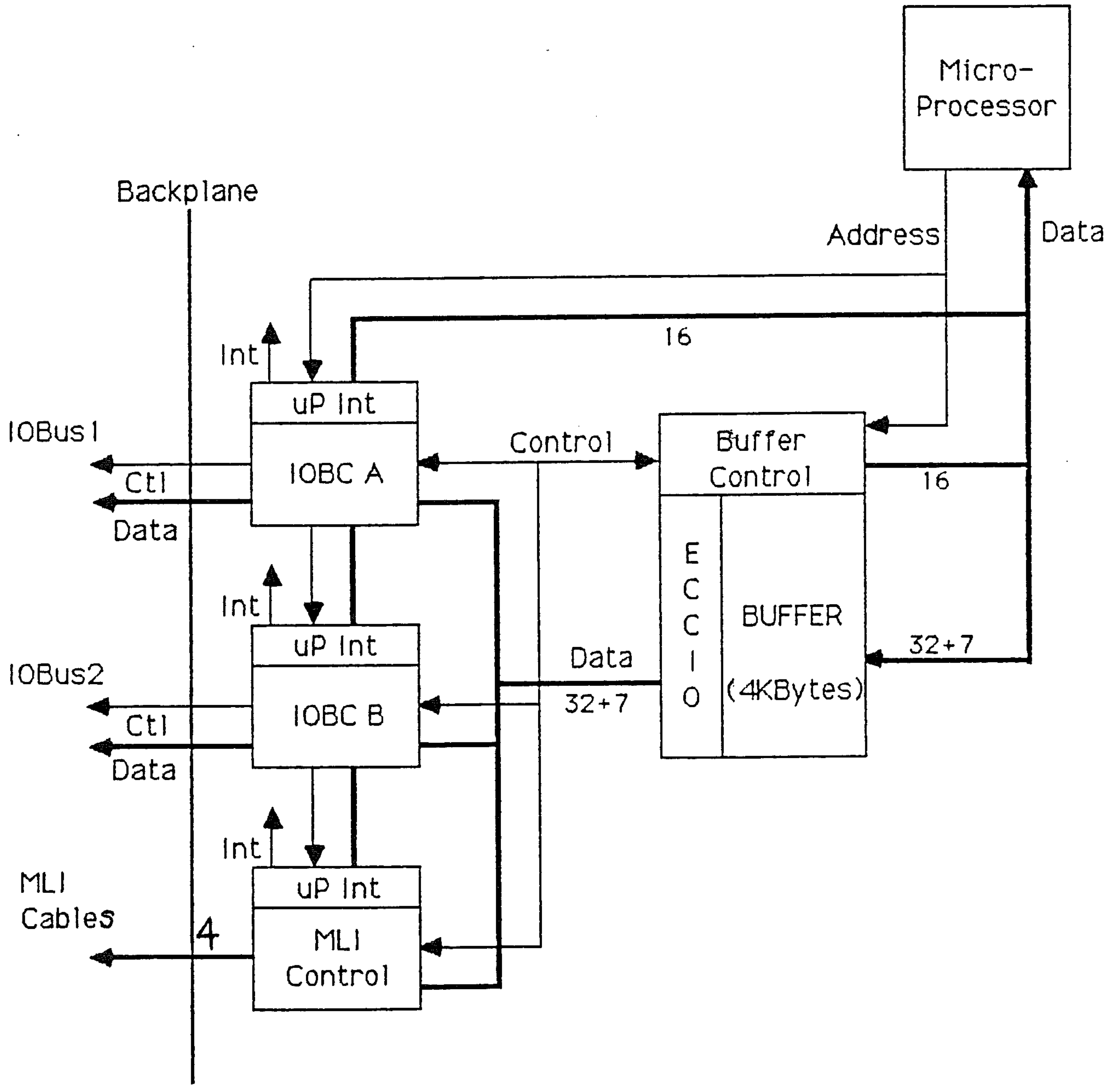

Figure 10-1 MLI Controller

10.2 OPERATIONAL OVERVIEW

The controller can be started by a Poll Test or a Poll Request. A Poll Test is initiateo by the microprocessor (68020); a Poll Request is initiated by a base module. In a Poll Test, the microprocessor assembles a message in the buffer, selects the base, and requests a connection to a DLP. The message contains the DLP address in the base, an I/O descriptor, and two words of descriptor links. This happens at the begining of an I/O operation.

In a Poll Request, a return message (the descriptor links) is sent by the DLP, and connection to system is requested. After a successfull Poll Test or Poll Request, a state machine monitors the DLP's status transitions and begins the next action to be taken by the MLI controller.

10.3 HARDWARE OVERVIEW

As shown in Figure 10-2, the MLICM is composed of following blocks:

MLI and 68020 interface
Data path controller
Strobe logic (synchronous and asynchronous)
MLI external control
MLI internal control
Parity checking:
    LPW generator
    Vertical parity generator/checker
    Internal parity error detector.
DLP status detection and control
Poll Test, Poll request, Control store

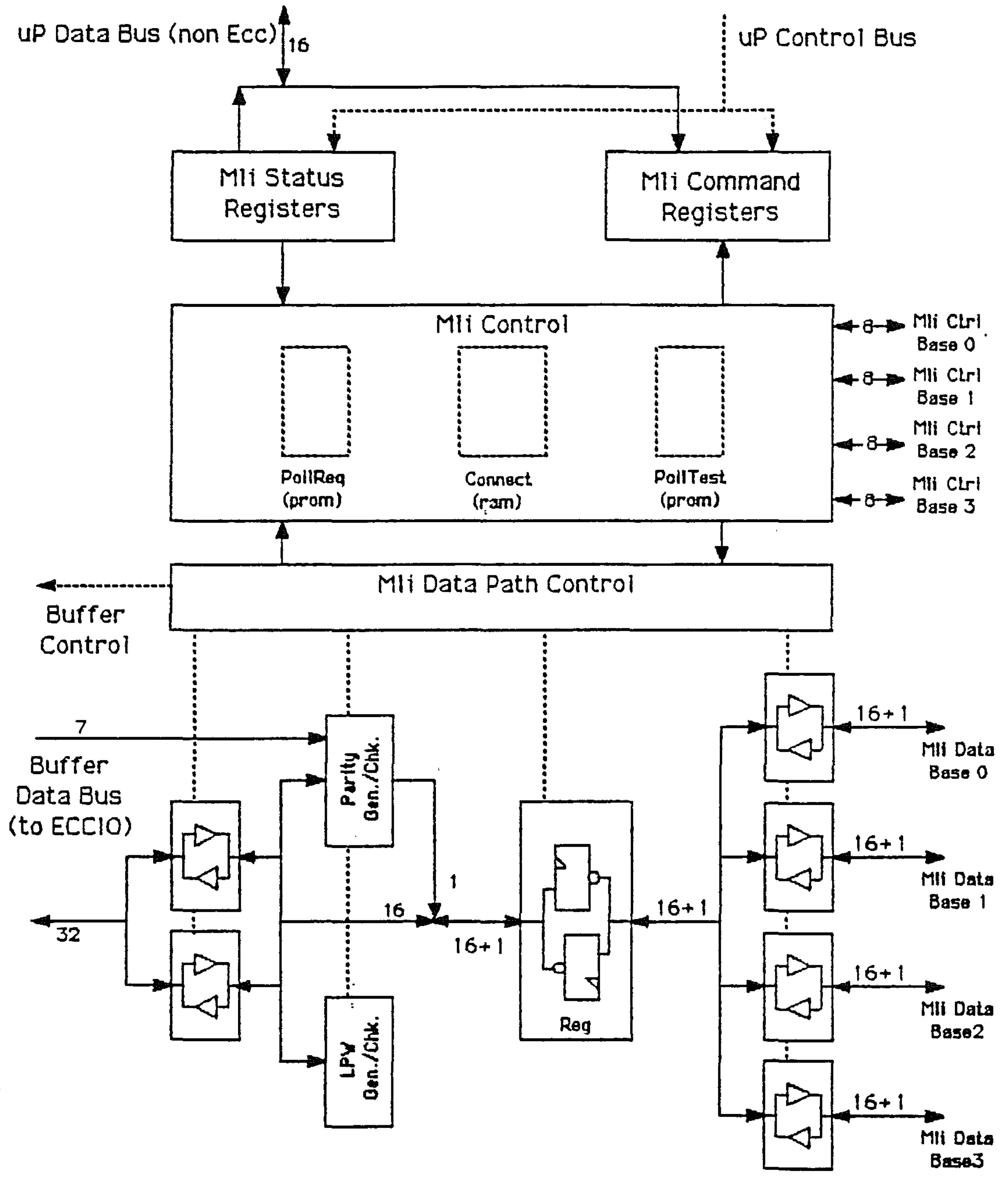

Figure 10-2 MLI Control Module

10.3.1 MLI - MICROPROCESSOR INTERFACE

All monitoring and loading of this interface is done through a Peripheral Interface and Timer (PIT). The microprocessor and MLI controller interface is composed of the following segments:

Byte counter
Command register
Status register

10.3.1.1 BYTE COUNTER

The Byte Counter selects the length of a Read or Write operation. It is an 8 digit decimal counter that is loaded by the microprocessor for a specific I/O before each data transfer. It is read by the microprocessor after a data transfer is completed. A memory mapped address is assigned by the microprocessor for reading and writing the byte counter.

The byte counter can decrement by 4 or 2, and can increment by 2. Normally the counter decrements by 4 with each strobe from the DLP. At the end of data transfer (read or write) it updates itself by incrementing or decrementing by 2. This adjustment is used if the DLP is a character DLP (can handle 1 byte data transfer).

Monitor logic on the output of the byte counter informs MLI control of the status of the counter (negative, or value less than 6), and a PROM and two 74F194s make the 2 least significant digits of the counter.

The counter decrements after an acknowledgement of a received strobe from the DLP during data transfer. This signal (INHF2) enables the counter. Two signals from the MLI Control Store (MLICS) determine the counter modes as follows:

BACKUP	XFERL	INHF2	Result
x	x	0	no count
0	0	1	count down by 4
0	1	1	hold
1	0	1	count down by 2
1	1	1	count up by 2

10.3.1.2 COMMAND REGISTER

The command register is an eight bit field used by the microprocessor to initiate or assist the MLI controller during an operation. The microprocessor treats this register as an addressable data field.

The command register is implemented with three PALs (16R4). The register bit assignment is as follows:

bit	function
0	START MLI. After a connection between the DTM and the DLP is established, the MLI controller stops itself several times during an I/O operation. The microprocesor monitors the status of the MLI controller, then sets this bit to restart the MLI controller.
1	CONNECT CLEAR. By setting this bit, the microprocessor tells the controller to disconnect from the base module.
2	POLL TEST SET. To initiate a Poll Test, the microprocessor sets this bit after a message is built and loaded into the buffer.
3	SELECTIVE CLEAR. The microprocessor uses this bit to issue a selective or a master clear to the DLP.
4	MODE 16. This bit tells the controller that only the most significant 16 bits of data in the buffer are valid during a send operation, or to only write 16 bit of valid data during a receive operation. This mode is used when controller is sending op codes and links to the DLP, or receiving links and result descriptors from the DLP.

10.3.1.2 COMMAND REGISTER (Continued)

5 6 7 Before initiating a Poll Test, the microprocessor specifies the base for which the Poll Test is initiated. Bit 5 enables loading of the base register. Bits 6 (base register bit 0) and bit 7 (base register bit 1) selcct the base module as follows:

bit 7	bit 6	base
0	0	0
0	1	1
1	0	2
1	1	3

The base 0 frontplane connector is on top.

10.3.1.3 STATUS REGISTERS

The status register is composed of a Condition register and a DLP Status register.

The conoition register is used by the MLI controller to ask the microprocessor for assistance. A four bit field is assigned for this purpose. The condition register is implemented in a PAL(16R4).

Bits in the condition register are assigned as follows:

MP	MLI
0	idle
1	PTEST to PREQ - Connection PTEST did not complete - Disconnect
2	Error used on Poll Test
3	Receive R/D during Poll Test
4	I/O descriptor successfully sent
5	Result descriptor successfully received
6	Descriptor links in the buffer (receive R/D next)
7	Descriptor links in the buffer (read burst next)
8	Descriptor links in the buffer (write burst next)
9	Request for disconnect
10	Break on read burst
11	Break on write burst
12	Entry after a successful Poll Test
13	Op was sent; ready to send descriptor links
14	PTEST to PREQ convert PTEST did not complete Initiate PTEST again
15	Illegal

The microprocessor monitors the DLP status bits. The DLP status bits are monitored directly from the frontplane, or after they are registered.

10.4 DATA PATH CONTROL

Data path control is responsible for movement of data between the I/O buffer and the MLI data registers. A two bit state machine is the heart of the data path control logic. After the MLI controller resumes buffer access, the controller generates control signals to the ECCIO and the I/O buffer address counter.

Data path control is implemented on five PALs. The state representation is due to the limited number of product terms per PAL. The bit assignment is as follows:

state(1)	state(0)	
1	1	State 0; clear or idle
0	1	State 1 and state 2 used during burst mode
1	0	State 3
0	0	Invalid; not used

The I/O buffer is 32 data bits wide; the MLI data bus is 16 bits wide. Therefore, the data path controller must determine when and how to do the following:

o Increment the address counter

o Load the MLI data register

o Use the correct portion of the ECCIO. When sending, the correct portion of the ECCIO to be sent to the data register must be enabled, and when receiving, the MLI data register must be enabled to the ECCIO.

The data path controller operates in the following modes:

Send Mode 16
Receive Mode 16
Send Burst
Receive Burst

10.4 DATA PATH CONTROL (Continued)

SEND MODE 16

This mode is used when the DTM is attempting to send an op and descriptor links to the DLP. In this mode, the microprocessor assembles the message in the buffer, then tells the MLI controller to start a Poll Test. The microprocessor also puts the controller in 16 bit data transfer mode. When the MLI controller is in 16 bit mode, only the 4 most significant digits of the buffer are valid; the rest are cleared by the microprocessor.

RECEIVE MODE 16

This mode is used when the DTM is receiving descriptor links or result descriptors from the DLP. In this mode, the 4 least significant digits of data from the DLP is written into the I/O buffer as zeroes; the 4 most significant digits contain data (descriptor links or result descriptors).

SEND BURST

This mode is used when the controller is doing data transfers (send). All strobe turnarounds are asynchronous (see strobe logic). This mode is initiated after a successful Poll Request and reception of the links.

RECEIVE BURST

This mode is used when the controller is receiving. Strobe turnarounds are asynchronous (see Strobe Logic). This mode is initiated after a successful Poll Request and the reception of the links.

10.5 STROBE LOGIC

Because ML1 strobes from the DLPs are received asynchronously, it is necessary to latch them. Four R-S flip-flops latch the strobes received from the bases. These latches are reset by the reply strobe from the DTM.

In synchronous mode (non-data), the latches are reset by the low portion of the clock cycle. In asynchronous mode (data burst) they are reset by a derivative of the delay component used for strobe generation.

The output of of the latches is fed into a multplexor that selects strobes from one of the bases. The latch is reset in asynchronous mode only after the MLI data is valid, and a strobe is sent to the DLP.

For the controller to run in asynchronous mode, the status sent by the DLP must be 4 (read) or 8 (write), the controller must be in data mode, and the byte counter must have a value greater than 6 (digits). Strobe generation in

In synchronous mode, strobe generation is done by the control store. When the controller is not in data mode (and is not halted), the strobe turnaround is as follows:

- o Receive the DLP status with strobe; allow time-bound metastability resolution for strobe latches.
- o Load the new DLP status.
- o Decode the status transition into an action needed in the control store (see Action Decode)
- o Send the response strobe.

10.6 MLI EXTERNAL CONTROL

The MLI control signals are Terminate, Channel Select, Address Select, and Access Grant. These signals are used in Poll Test and Poll Request procedures, or when the DTM is connected to a DLP.

10.7 MLI INTERNAL CONTROL

The MLI controller can be in one of the following states:

- o Poll Test
- o Poll Request
- o Connection
- o Idle

The transitions occur from Poll Test or Poll Request to Connection, and from Connection to Idle state. Poll Test is initiated by the microprocessor; Poll Request is initiated by the base unit. All DLP state transitions are decoded in Connection state. The microprocessor clears these states and forces the controller to go to IDLE.

10.8 PARITY GENERATION AND CHECKING

The LPW register is built from four registered PALs. It monitors the MLIDATA bus during send mode, and the MLIRAMDATA bus in receive mode. These PALs are controlled by two mode lines as follow:

MODE	ACTION
0	Shift
1	Hold
2	Calculate LPW (depending on direction of data transfer, Exclusive-OR the registered value with the proper input.
3	Initialize (set to FFFF, all ones)

Because of the limited number of I/O pins on PALs, the ShiftIn and ShiftOut bits are shared with Direction (send or receive), and LPWZERO (LPW=0), respectively.

Vertical parity is checked and generated on the MLI data bus using two 74F280 parity comparator/generators. The internal parity detector is built on a 20R4 PAL. This PAL compares the ECC portion of the ECCIO with the parity from the MLI data register. Refer to the PAL equations for additional details.

10.9 DLP STATUS DETECTION AND CONTROL

After a connection to a DLP is established, each DLP strobe validates a new status from the DLP. A multiplexor decides what base the status belongs to. After the strobe is latched, the first clock is used to avoid metastability and noise problems, and on the new DLP is latched in a F194 on the second clock.

This logic is implemented in a 20R4 PAL. This PAL generates the mode lines to the F194s that capture the new and old status transitions. The new and old status is used to determine the next action to be taken by the DTM.

10.10 POLL TEST, POLL REQUEST, AND CONTROL STORE

If the MLI controller is not in Idle mode, it must be in one of the following modes, using the control store shown:

Poll Test - PROMs
Poll Request - PROMs
Connect - Static RAM

10.10.1 POLL TEST

After the microprocessor initiates a Poll Test, the PROMs take over the DLP status monitoring and action generation of the MLI controller. There are three 2Kx4 PROMs, with even parity across the PROM's outputs.

The inputs to these PROMs are used to process the Poll Test. The inputs include a 2 bit counter, 4 bits of status from the MLI, MLI control signals (Address Select and Channel Select), the registered strobe from the DLP, and a feedback from the PROM called Rtime, which is used to to notify the microprocessor of problems detected during poll test.

During Poll Test, the DLP address is read from the I/O buffer and put into the MLI data register. Also, the drivers for the base are enabled. At the same time, a 3 bit counter (Poll Test counter) is initialized to zero. This counter counts six, and validates the DLP address on the MLI bus for at least three clocks before the MLI controller can assert Channel Select.

On next clock, the MLI controller asserts Address Select, and waits for a proper status from the distribution card. After receiving the proper status from the DLP, control is transferred to the MLI control store. The Poll Test PROMs are disabled.

10.10.2 POLL REQUEST

To enable the Poll Request PROM (512x8), the following conditions must be met:

- o There must be an interupt from a base
- o An I/O buffer access must be granted

After the I/O buffer access is granted, the MLI controller clears the base register and sends Access Grant to the bases. Any base that requires the DTM's attention responds by driving global priority onto the MLI data lines, followed by a strobe.

The MLI controller resolves the global priority from the base that has interrupt asserted. If multiple bases request connection, the one with the highest global priority is granted access.

If multiple bases request connection, and if they have the same global priority, the lowest numbered base is granted connection.

There are 3 2Kx4 PROMS used for global priority resolution.

The base number is loaded into the base register. Channel Select, followed by Address Select is sent to it. The selected base sends an acknowledgment strobe, and the MLI controller drops Channel Select. Now, the connection is completed. The Poll PROM is disabled, and control goes to the MLI control store.

10.10.3 CONTROL STORE

A successful Poll Test or Poll Request enables the MLI control store, which is 8Kx24, and is implemented in three 8Kx8 static RAMs. The output of the control store is latched into a set of 16R4 PALs. Three F299s are used to load the control store.

The microcode for the MLI control store is written in Algol. Following is list of the control store output bits.

[23:1]	EnLPWErr	Enable LPW error to be latched into the
[22:4]	NOTUSED	unused
[18:1]	SNDST	Set SEND flip-flop
[17:1]	HALTMLI	Halt the MLI controller
[16:1]	CONTPARIT	Control store parity (even)
[15:4]	CONDITION	Condition monitored by microprocessor
[11:1]	CTRBCK	Back up the byte counter
[10:1]	LPWCLR	Clear LPW register
[09:1]	LPWTB	Enable LPW onto MLI data bus
[08:1]	XFERST	Enable byte counter to count (up/down)
[07:1]	ILLEGL	Illegal status transition
[06:3]	TOGLES	Three toggles used for seouencing
[03:4]	ACTION	Next action to be taken

Following are the control store input bits:

[12:1]	HRFCMP	Host Return Field Comparison used for PO to Poll Reouest conversion.
[11;1]	TERM	Terminate (MLI control signal)
[10:3]	toggles	Three toggles used for sequencing
[07:4]	OSR	Old DLP status
[03:4]	NSR	New DLP status

The combinations of the new DLP status, old DLP status, terminate, and toggles are decoded to generate an action for the MLI controller.

10.10.3 CONTROL STORE (Continued)

The following table defines the individual actions:

value	action
0	Set terminate if the byte counter is odd
1	Set data mode (burst)
2	Sync mode (!data); terminate if counter is not OK
3	Not used
4	Send sync (!data)
5	Receive sync (!data)
6	Send C address if present, else set C-ADD error
7	Receive LPW; do not return a strobe. If counter is odd, set address error.
8	Back up the byte counter (not used)
9	Receive LPW with strobe
10	If byte counter is odd, set address error
11	Send strobe (SIO)
12	Send line turn strobe; to avoid parity errors, do not load the bidirectional register
13	No action
14	No action
15	No action

APPENDIX A - TABLES

A.1 INTERRUPT PRIORITY

Priority Level	Interrupt
7	not used
6	SMC attention
5	I/O Bus-A attention
4	I/O Bus-B attention
3	not used >
2	MLI poll request
1	not used

A.2 MEMORY MAP

Device	Address Range	Width	Accessibility
Resident Program Memory	000000-00FFFF	B,W,L	Read-only
Dynamic Data Memory	200000-3FFFFF	B,W,L	Read and Write
Maintenance Register	410000	L	Read and Write
IO Buffer Cmnd Reg	430000	W	Write-only
IO Buffer Status Reg	430002	W	Read-only
MLI Result Reg	430004	W	Read-only
MLI Command Reg	430006	W	Write-only
MLI Status Reg	430008	W	Read-only
MLI Identification Reg	43000A	W	Read-only
MLI High Byte Counter	43000C	W	Read and Write
MLI Low Byte Counter	43000E	W	Read and Write
I/O Buffer Address Reg	430014	W	Read and Write
I/O Buffer Limit Reg	430016	W	Read and Write
I/O Bus-A Status Reg	430018	W	Read-only
I/O Bus-B Status Reg	43001A	W	Read-only
I/O Bus-A Command Reg	43001C	W	Write-only
I/O Bus-B Command Reg	43001E	W	Write-only
I/O Buffer	600000-600FFF	L	Read and Write

A.3 SMC INTERFACE REGISTER COMMANDS

The SMC interface register handles the following types of commands (the type is specified in bit 31:30).

value	command type
0	not used
1	MCP configuration
2	SMC diagnostic
3	SMC maintenance

A.4 SMC MAINTENANCE COMMANDS

Opcode = SMC Interface Register Bit<29-24>
Parameter = SMC Interface Register Bit<23-00>

Opcode	Paremeter	Command
00	<value>	Load Memory Pointer
08		Read Byte Through Memory Pointer
09		Read Word Through Memory Pointer
0A		Read Long Through Memory Pointer
0B	FF	Take Snapshot of Microprocessor
0B	00	Read Microprocessor Data Register D
0B	01	Read Microprocessor Data Register D
0B	02	Read Microprocessor Data Register D
0B	03	Read Microprocessor Data Register D
0B	04	Read Microprocessor Data Register D
0B	05	Read Microprocessor Data Register D
0B	06	Read Microprocessor Data Register D
0B	07	Read Microprocessor Data Register D
0B	08	Read Microprocessor Address Register
0B	09	Read Microprocessor Address Register
0B	0A	Read Microprocessor Address Register
0B	0B	Read Microprocessor Address Register
0B	0C	Read Microprocessor Address Register
0B	0D	Read Microprocessor Address Register
0B	0E	Read Microprocessor Address Register
0B	0F	Read Microprocessor Address Register
0B	10	Read Microprocessor Program Counter
0B	11	Read Microprocessor Status Register
0B	20	Read Channel Status
10	<value>	Write Byte through Memory Pointer
11	<value>	Write Word through Memory Pointer
12	<value>	Write Longword high 16-bit through M
13	<value>	Write Longword low 16-bit through M

A.5 MCP CONFIGURATION COMMANDS

Opcode = SMC Interface Register Bit<29-24>
Parameter = SMC Interface Register Bit<23-00>

opcode	parameter	command
0D	<value>	Load Base-D
0F	<value>	Load Base-F
10	<value>	Load QwikDisk Base Address Word-1
11	<value>	Load QwikDisk Base Address Word-2
12	<value>	Load QwikDisk Base Address Word-3
13	<value>	Load QwikDisk Base Address Word-4

A.6 ERROR LOGGING

Data Scratchpad Location	Field Description
200040	IOMC underflow count
200044	IOMC overflow count
200048	IOMC ECCIO single bit error count
20004C	IOMC ECCIO multiple bit error count
200050	IOMC MDECC01 single bit error count
200054	IOMC MDECC23 single bit error count
200058	IOMC MDECC mutiple bit error count
20005C	Memory Bus single bit RAM error count
200060	Memory Bus single bit Bus error count
200064	Memory Bus multiple one bit RAM error count
200068	Memory Bus multiple one bit BUS error count
20006C	Memory Bus multiple bit RAM error count
200070	Memory Bus multiple bit Bus error count
200074	Memory Bus Error Parity error count
200078	Memory Bus MDC malfunction error count
20007C	not used
200080	I/O Bus timeout error count
200084	I/O Bus bus busy error count
200088	I/O Bus arbitration error count
20008C	I/O Bus preempt error count
200090	I/O Bus module not present error count
200094	I/O Bus module busy error count
200098	I/O Bus illegal disconnect error count
20009C	I/O Bus single bit send error count
2000A0	I/O Bus single bit receive error count
2000A4	I/O Bus multiple bit error count
2000A8	I/O Bus hardware error count
2000AC	not used
2000B0	not used
2000B4	not used
2000B8	not used
2000BC	not used
2000C0	Miscellaneous error count

A.7 BACKPLANE PIN ASSIGNMENTS

The following table lists the I/O Bus signal names and backplane pin assignments for the 10MCs, the SMC, and the Bus #0 connection to the DTMs.

Signal Name	Card	BP
TIO1DATA-00P	A05	A7P4
TIO1DATA-01P	C05	A7P6
TIO1DATA-02P	A06	A8P4
TIO1DATA-03P	C06	A8P6
TIO1DATA-04P	A07	A9P4
TIO1DATA-05P	C07	A9P6
TIO1DATA-06P	A08	B0P4
TIO1DATA-07P	C08	B0P6
TIO1DATA-08P	A09	B1P4
TIO1DATA-09P	C09	B1P6
TIO1DATA-10P	A10	B2P4
TIO1DATA-11P	C10	B2P6
TIO1DATA-12P	A11	B3P4
TIO1DATA-13P	C11	B3P6
TIO1DATA-14P	A12	B4P4
TIO1DATA-15P	C12	B4P6
TIO1DATA-16P	A13	B5P4
TIO1DATA-17P	C13	B5P6
TIO1DATA-18P	A14	B6P4
TIO1DATA-19P	C14	B6P6
TIO1DATA-20P	A15	B7P4
TIO1DATA-21P	C15	B7P6
TIO1DATA-22P	A16	B8P4
TIO1DATA-23P	C16	B8P6
TIO1DATA-24P	A17	B9P4
TIO1DATA-25P	C17	B9P6
TIO1DATA-26P	A18	C0P4
TIO1DATA-27P	C18	C0P6
TIO1DATA-28P	A19	C1P4
TIO1DATA-29P	C19	C1P6
TIO1DATA-30P	A20	C2P4
TIO1DATA-31P	C20	C2P6
TIO1DATA-32P	A24	C6P4
TIO1DATA-33P	C23	C5P6
TIO1DATA-34P	A23	C5P4
TIO1DATA-35P	C22	C4P6
TIO1DATA-36P	A22	C4P4

A.7 BACKPLANE PIN ASSIGNMENTS (Continued)

Signal Name	Card	BP
TIO1DATA-37P	C21	C3P6
TIO1DATA-38P	A21	C3P4
TIO1BUSREF$P	C27	C9P6
TIO1BUSREQ$N	C26	C8P6
TIO1BUSBUSYN	C28	D0P6
TIO1MODREQ$N	A26	C8P4
TIO1MODPRSNN	A27	C9P4
TIO1MODAVALN	A29	D1P4
TIO1MODBUSYN	A28	D0P4
TIO1TRANSFRN	C29	D1P6
TIO1ERROR$$N	A25	C7P4
TIO3IIOREQ$N	C24	C6P6
TIO1IIOACK$N	C25	C7P6

The DTMs drive and receive the control signals through different pins. Those listed above are the driver pins; the receiver pins are as follows:

Signal Name	Card	Pin
TIO3BUSREF$P	C32	D4P6
TIO3BUSREQ$N	C33	D5P6
TIO3BUSBUSYN	C31	D3P6
TIO3MODREQ$N	A33	D5P4
TIO3MODPRSNN	A32	D4P4
TIO3MODAVALN	A30	D2P4
TIO3MODBUSYN	A31	D3P4
TIO3TRANSFRN	C30	D2P6
TIO3ERROR$$N	A34	D6P4

In addition, NormalRequest and EmergencyRequest are driven by the DTMs on the following pins:

Signal Name	Card	Pin
TIO1NORMREQN	A35	D7P4
TIO1EMERGRQN	C35	D7P6

These two pins are wired on the backplane to the appropriate data bus wires.

A.7 BACKPLANE PIN ASSIGNMENTS (Continued)

For the DTMs, there are a second set of backplane pins for I/O Bus #1, as follows:

Signal Name	Card	BP
TIO2DATA-00P	A45	F0P4
TIO2DATA-01P	C45	F0P6
TIO2DATA-02P	A46	F1P4
TIO2DATA-03P	C46	F1P6
TIO2DATA-04P	A47	F2P4
TIO2DATA-05P	C47	F2P6
TIO2DATA-06P	A48	F3P4
TIO2DATA-07P	C48	F3P6
TIO2DATA-08P	A49	F4P4
TIO2DATA-09P	C49	F4P6
TIO2DATA-10P	A50	F5P4
TIO2DATA-11P	C50	F5P6
TIO2DATA-12P	A51	F6P4
TIO2DATA-13P	C51	F6P6
TIO2DATA-14P	A52	F7P4
TIO2DATA-15P	C52	F7P6
TIO2DATA-16P	A53	F8P4
TIO2DATA-17P	C53	F8P6
TIO2DATA-18P	A54	F9P4
TIO2DATA-19P	C54	F9P6
TIO2DATA-20P	A55	G0P4
TIO2DATA-21P	C55	G0P6
TIO2DATA-22P	A56	G1P4
TIO2DATA-23P	C56	G1P6
TIO2DATA-24P	A57	G2P4
TIO2DATA-25P	C57	G2P6
TIO2DATA-26P	A58	G3P4
TIO2DATA-27P	C58	G3P6
TIO2DATA-28P	A59	G4P4
TIO2DATA-29P	C59	G4P6
TIO2DATA-30P	A60	G5P4
TIO2DATA-31P	C60	G5P6
TIO2DATA-32P	A64	G9P4
TIO2DATA-33P	C63	G8P6
TIO2DATA-34P	A63	G8P4
TIO2DATA-35P	C62	G7P6
TIO2DATA-36P	A62	G7P4

A.7 BACKPLANE PIN ASSIGNMENTS (Continued)

Signal Name	Card	BP
TIO2DATA-37P	C61	G6P6
TIO2DATA-38P	A61	G6P4
TIO2BUSREF$P	C67	H2P6
TIO2BUSREQ$N	C66	H1P6
TIO2BUSBUSYN	C68	H3P6
TIO2MODREQ$N	A66	H1P4
TIO2MODPRSNN	A67	H2P4
TIO2MODAVALN	A69	H4P4
TIO2MODBUSYN	A68	H3P4
TIO2TRANSFRN	C69	H4P6
TIO2ERROR$$N	A65	H0P4
TIO4IIOREQ$N	C64	G9P6
TIO2IIOACK$N	C65	H0P6

For I/O Bus #1, the control signal receiver pins are:

Signal Name	Card	Pin
TIO4BUSREF$P	C72	H7P6
TIO4BUSREQ$N	C73	H8P6
TIO4BUSBUSYN	C71	H6P6
TIO4MODREQ$N	A73	H8P4
TIO4MODPRSNN	A72	H7P4
TIO4MODAVALN	A70	H5P4
TIO4MODBUSYN	A71	H6P4
TIO4TRANSFRN	C70	H5P6
TIO4ERROR$$N	A74	H9P4

NormalRequest and EmergencyRequest are driven on the following pins:

Signal Name	Card	Pin
TIO2NORMREQN	A75	I0P4
TIO2EMERGRQN	C75	I0P6

Pins B04 (A6P5) to B85 (J0P5), and D04 (A6P7) to D85 (J0P7) are hard grounds; they are connected directly to the ground plane for all the DTM slots.

APPENDIX B - MESSAGE LEVEL INTERFACE FUNCTIONAL DESCRIPTION

The MLI is a 25-line connection between the MLI controller module and the DLP base. This provides a communication path between the controller and the distribution card, which is by the controller and the MLI to establish a path to a specific DLP (Poll Test).

After connecting to a DLP, the distribution card passes data and becomes transparent to the operation. When the DLP is connected, it sends a 4-bit status code to the MLI. The MLI controller receives the DLP status and determines what state the DLP is in, and what action is required. The distribution card also performs functions for the DLP requesting reconnection (Poll Request).

Some MLI interface lines are unidirectional, and are sent from the MLI controller to the DC. The lines are:

ADDSEL	Address Select. When ADDSEL and Channel Select are made active at the same time, that indicates that the MLI controller is attempting connection to a DLP. After connecting, Channel Select is made inactive. The distribution card that receives Address Select but not Channel Select assumes that the MLI controller is busy.
AG+SIO/	Access Grant or Strobe I/O. This line indicates that a requesting DLP has been granted access during a Poll Request. It also provides the handshake strobe from the controller to the DLP for the information on the data line.
TRM+MC	Terminate or Master Clear This signal terminates a DLP data transfer to selectively clear a DLP.

The following lines are unidirectional from the distribution card to the MLI controller.

LCPST	DLP Strobe. This is a handshake strobe from the DLP for the information on the data lines.

APPENDIX B - MESSAGE LEVEL INTERFACE FUNCTIONAL DESCRIPTION (Continued)

ER+ST8/	Emergency Request or 8-bit of the DLP Status. If Address Select is inactive, this line indicates that a DLP in the base is requesting service. If Address Select is active, this line is the 8-bit of the connected DLP's status.
IP+ST4/	DLP Request, Poll Test Error, or 4-bit of the DLP Status If Address Select is inactive, this line indicates that a DLP is requesting connection. During Poll Test, this line indicates that the address received by the distribution card contained a parity error. If Address Select is active, this line is the e DC cotained a parity 4-bit of the connected DLP's status.
PB+ST2/	Port Busy or 2-bit of DLP status. During a Poll Test, this line indicates Port Busy (multiple distribution card bases such as Shared System Processor only). If Address Select is active, this is the 2-bit of the connected DLP's status.

The following lines are bi-directional:

CS+ST1/	Channel Select or 1-bit of DLP status. This line (in conjunction with Address Select) indicates that the MLI controller is attempting to connect to a DLP in the base. If Address Select is active, this line is the 1-bit of the connected DLP's status.
Parity/	Parity bit for the data lines. This is the parity (odd) line for the information on the data lines.
DATAnn/	Sixteen data lines. These lines transfer the information between the DLP and the MLI controller. The information may be the MLI op code or variants, descriptor links, result descriptors, LP words, or data.

APPENDIX B - MESSAGE LEVEL INTERFACE FUNCTIONAL DESCRIPTION (Continued)

ER+ST8/	Emergency Request or 8-bit of the DLP Status. If Address Select is inactive, this line indicates that a DLP in the base is requesting service. If Address Select is active, this line is the 8-bit of the connected DLF's status.
IP+ST4/	DLP Request, Poll Test Error, or 4-bit of the DLP Status If Address Select is inactive, this line indicates that a DLP is requesting connection. During Poll Test, this line indicates that the address received by the distribution card contained a parity error. If Address Select is active, this line is the e DC cotained a parity 4-bit of the connected DLP's status.
PB+ST2/	Port Busy or 2-bit of DLP status. During a Poll Test, this line indicates Port Busy (multiple distribution card bases such as Shared System Processor only). If Address Select is active, this is the 2-bit of the connected DLP's status.

The following lines are bi-directional:

CS+ST1/	Channel Select or 1-bit of DLP status. This line (in conjunction with Address Select) indicates that the MLI controller is attempting to connect to a DLP in the base. If Address Select is active, this line is the 1-bit of the connected DLP's status.
Parity/	Parity bit for the data lines. This is the parity (odd) line for the information on the data lines.
DATAnn/	Sixteen data lines. These lines transfer the information between the DLP and the MLI controller. The information may be the MLI op code or variants, descriptor links, result descriptors, LP words, or data.