C V A X / R I G E L M E M O R Y I N T E R C O N N E C T

COMPANY CONFIDENTIAL

REVISION HISTORY

REVISION	DATE	AUTHOR
0.0	10-MAY-85	Williams/Ives
0.1	17-MAY-85	
0.2	21-MAY-85	
0.3	31-MAY-85	
0.9	7-JUNE-85	

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The XMI is the primary interconnect medium in the CVAX/Rigel family of computer systems. It consists of the protocol observed by a node on the bus, the electrical environment of the bus, the backplane, and the logic used to implement the protocol. The XMI can support four or more processors, eight or more memory subsystems, and up to 4 I/O adapters.

1.1 OVERVIEW

The XMI is a limited length, pended, synchronous bus with centralized arbitration. Several transfers can be in progress at a given time, allowing highly efficient use of bus bandwidth.

Arbitration and data transfers occur simultaneously, with multiplexed data and address lines. The bus supports reads and writes to memory of quadword, octaword, and hexaword (reads only) length. In addition, the bus supports longword length read and write operations to I/O space. These longword operations implement byte and word modes required by certain I/O devices.

The available bus bandwidth of the XMI for each type of transfer is listed below:

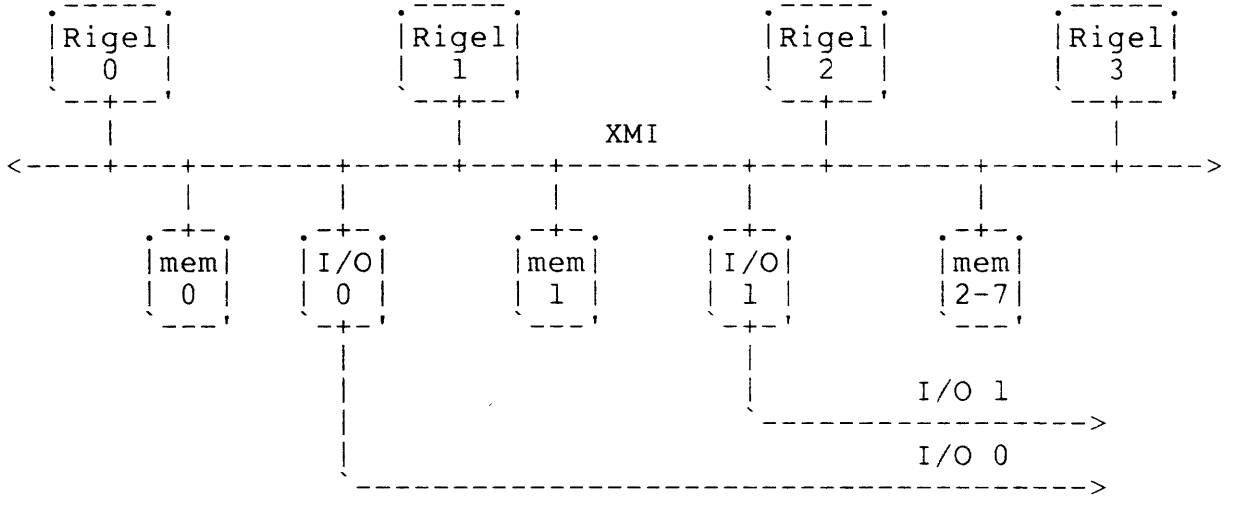
Operation	Bandwidth
Longword Read	25.0 Mbytes/sec
Quadword Read	50.0 Mbytes/sec
Octaword Read	66.6 Mbytes/sec
Hexaword Read	80.0 Mbytes/sec
Longword Write	25.0 Mbytes/sec
Quadword Write	50.0 Mbytes/sec
Octaword Write	66.6 Mbytes/sec

XMI nodes monitor bus activity and check for various conditions. Parity is maintained over the Command/Address/Data field in addition to parity over the control fields.

The hardware implementation of the XMI protocol is based on CMOS gate arrays or standard cells, with separate bus transceivers providing single ended interface to the backplane. Clocks, due to their more critical distribution requirements, may be driven on the backplane differentially and converted to single ended signals on the modules.

The XMI interface hardware is latch-based. The clock system is two-phase, non-overlapping, with clock pulses as wide as possible. Transfers over the backplane begin with the occurrence of a "Phase1" Clock (conceptually the first phase). It is during this phase that new data is initially enabled onto the bus. Data must be at the latch input at about the worst case late leading edge of Phase1 and must be routed from the latch to the actual XMI etch on the backplane quickly so that the XMI is actually driven soon after the worst case late leading edge of Phase1. Receiving latches at a node are opened with an "Phase2" Clock. In the worst case, data will stabilize at the inputs of all receiving latches a bit before the worst case early trailing edge of Phase2. There is a brief time available between the worst case signal receive time near the trailing edge of Phase2 and the time when a signal would be required near the leading edge of the next Phase1 for transmission in the next XMI cycle.

A simple block diagram of a four processor Rigel system is shown below.



1.2 TERMS

In order to clearly describe the transactions which occur on the XMI the following terms are used:

- Node - A node is a hardware device which resides physically on the XMI backplane. There are a maximum of 16 nodes in a CVAX/Rigel system.

- Transfer - A transfer is the smallest quantum of work which occurs on the XMI. Typical examples of transfers are the command cycle of a read, and the command and following data cycles of a write.

- Transaction - A transaction is composed of one or more transfers. Transaction is the name given to the logical task being performed (e.g. read); in the case of the read specifically, the transaction consists of a command transfer followed some time later by a return data transfer. See Commander, Responder, Transmitter and Receiver below.

- Commander - The commander is the node that initiated the transaction in progress. In any write transaction, the commander is the node that requested the write; for reads, the commander is the one who requested the data. The distinction of being the commander in a transaction holds for the duration of the transaction in spite of the fact that in some cases it might appear that the commander changes. A case in point is where the commander initiates a read transaction. It is the responder (data source) that initiates the return data transfer, but the node that requested the data is still the commander.

- Responder - The responder is the complement to the commander in a transaction.

- Transmitter - A transmitter is the node that is sourcing the information on the bus. Using the read transaction as an example, the commander is the transmitter during the command transfer, and the receiver during the return data transfer.

- Receiver - The analog to the transmitter, the receiver is the sink of the data being moved during a transfer.

- Naturally Aligned - Refers to a data quantity whose address could be specified as an offset, from the beginning of memory, of an integral number of data elements of the same size. Characteristic of naturally aligned data items is the fact that the lower bits of their address are zero. All XMI reads and writes transfer a naturally aligned block of data.

- Wraparound Read - Defined to be a octaword or hexaword read operation where read data is returned in a specific pattern in which the specifically addressed quadword is returned first, independent of alignment. The remaining data in the the naturally aligned block of data containing the addressed quadword is returned in subsequent transfers. Below is an example of a octaword wraparound read:

  Example:

  Read Octaword, VAX byte address 00000018 (hex).

```
00000018  First Quadword
00000010  Second Quadword
```

  For purposes of defining the wrapping order for hexaword reads, a hexaword read is decomposed into two octaword reads, with the addressed octaword read data returned first. Within each of the octawords the wrapping order is the same as described above for octawords. Return data for the second octaword maintains the same wrapping order used in the first octaword.

  Example:

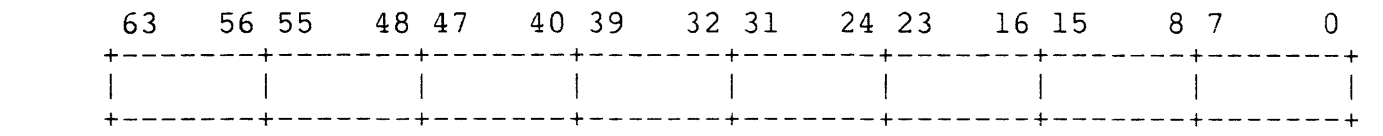
  Read Hexaword, VAX byte address 00000018 (hex).

```
00000018  First Quadword   \ First Octaword
00000010  Second Quadword  /
00000008  Third Quadword   \ Second Octaword
00000000  Fourth Quadword  /
```

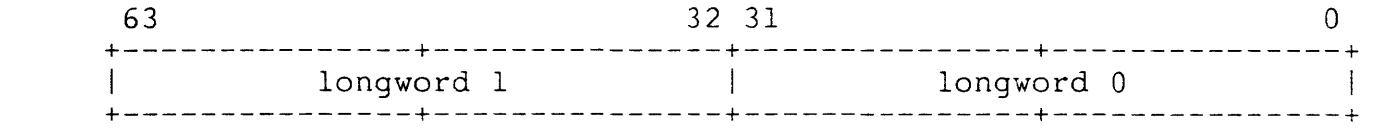
  The XMI protocol requires that all octaword and hexaword reads, both normal and interlocked, are wrapped.

Throughout this document the following numbering conventions will hold. Bits will be numbered from right to left with the leftmost bit the most significant, i.e.

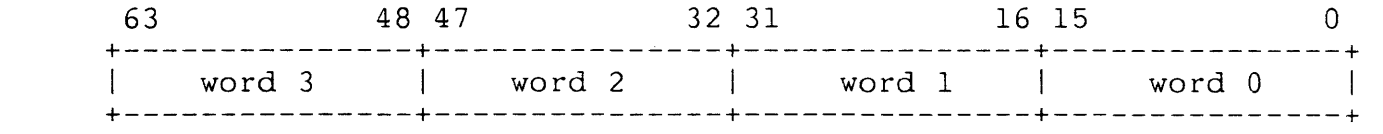
A Quadword is a single 64 bit entity



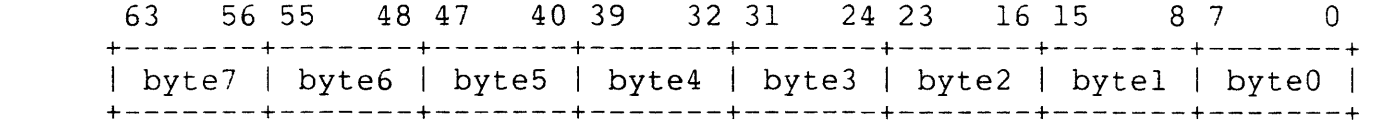
and in a similar fashion, Longwords



and Words



and Bytes.



1.3 FIELDS

The fields of information represented on the XMI are:

Arbitration:	Bus	Node (running total)
XMI HI ARB	(16)	1
XMI LO ARB	(16)	2
XMI HOLD	(1)	3
XMI BUS GRANT	(16)	4
Information Transfer:		
XMI FUNCTION L	(3)	7
XMI DATA	(64)	71
XMI ID	(5)	76
XMI PARITY	(1)	77
Response:		
XMI CONFIRMATION L	(5)	82
XMI PARITY ERROR L	(1)	83
Control:		
XMI RESET	(1)	84
XMI AC LOW	(1)	85
XMI DC LOW	(1)	86
XMI BAD	(1)	87
XMI CLOCKS	(4)	91
Miscellaneous:		
XMI NODE ID	(4 per slot)	95 = Total

For specific details of the signals in the XMI, see Section 2.2. Further information about the clocks can be found in section 2.1.

1.4 DATA TRANSACTION TYPES

The XMI supports the following types of data transactions:

- Longword Read (I/O space only)
- Quadword Read
- Octaword Read
- Hexaword Read
- Longword Interlock Read (I/O space only)
- Quadword Interlock Read
- Octaword Interlock Read
- Hexaword Interlock Read
- Longword Write Masked (I/O space only)
- Quadword Write Masked
- Octaword Write Masked
- Longword Unlock Write Masked (I/O space only)
- Quadword Unlock Write Masked
- Octaword Unlock Write Masked
- Quadword Tag Bad Data
- Octaword Tag Bad Data

Reads cause the transfer of data from the responder to the commander. Writes cause the transfer of data from the commander to the responder. Longword commands transfer 4 Bytes and likewise quadword, octaword and hexaword commands transfer 8, 16, and 32 Bytes. Interlocked variations of read commands are intended to do the same thing as the regular reads, but they also invoke a mutual exclusion mechanism. Interlock reads cause the setting of a lock bit associated with the location; whereas unlock writes cause the clearing of the lock bit. During periods when a location is locked, subsequent interlock read operations to that location will result in the responder returning a "Locked" non-data response instead of read data. All writes are masked and are accompanied by a set of mask bits that specifies which bytes out of the data transfer unit are to be written. Any arbitrary pattern of bytes can be written with the masked writes.

The Tag Bad Data command can be used by a commander to mark a memory location as containing incorrect data. A read to a location which is marked bad results in the return of the "Bad Data, Tagged" non-data response instead of read data.

1.5 INTERRUPT TRANSACTION TYPES

The XMI supports the following types of interrupt transactions:

- Interrupt Request (INTR)
- Interrupt Acknowledge (IDENT)
- Interprocessor Interrupt (IPINTR)

The INTR and IDENT transactions are used to implement device interrupts. An I/O node will issue an INTR transaction to a processor(s) in order to interrupt the processor at a specified IPL. In response to the INTR a processor node will issue an IDENT transaction directed to the interrupting I/O node soliciting an interrupt vector. An INTR transaction can be broadcast to multiple processor nodes. In this case the first processor responding with IDENT receives the interrupt vector; all other processors, upon seeing an IDENT directed to the interrupting device, should clear the interrupt pending condition. If IDENTs are issued simultaneously by two or more processors, the first to gain the bus will service the interrupt while the other(s) will perceive a passive release.

The IPINTR transaction is used to implement VAX interprocessor interrupts. An IPINTR directed to a processor(s) results in the processor(s) being interrupted at IPL 14 with a vector of 80H, as required by the VAX SRM. Since the value of the interrupt vector is fixed, IPINTR transactions do not require a corresponding interrupt acknowledge cycle.

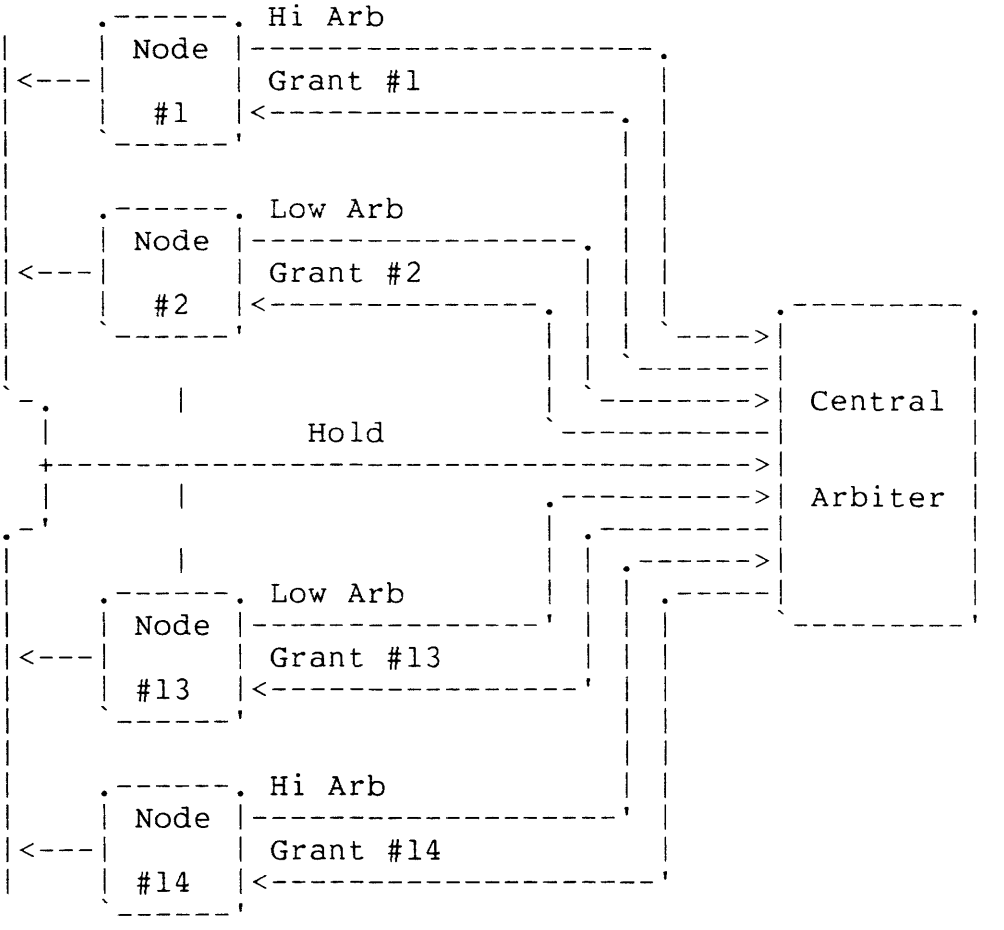
1.6 ARBITRATION

The XMI protocol as implemented in CVAX/Rigel, will support at most 16 nodes. Of these, all may desire the use of the bus at any given time. Arbitration cycles occur simultaneously with data transfer cycles using a set of lines dedicated specifically to this purpose.

When a node desires ownership of the bus, it asserts either its Low (Commander) or High (Responder) Arbitration line. During any given cycle, all nodes have the opportunity to request the bus. The arbiter receives all the requests and decides which node shall get the bus. In the next cycle, the selected node begins its transfer.

The XMI has an additional arbitration line, HOLD, which is a WIRE-OR'ed signal shared by all nodes. Assertion of HOLD guarantees that the current XMI transmitter will be granted ownership of the bus in the next cycle, independent of the values of HI ARB and LOW ARB. The primary use of HOLD is for multi-cycle transfers, allowing the current transmitter to acquire consecutive cycles. HOLD has an additional purpose in controlling XMI traffic. Should a node have difficulties in keeping up with bus traffic, such as a CPU backing-up on cache invalidate operations due to XMI write traffic, it may assert HOLD to temporarily suppress the start of additional XMI transfers.

CENTRALIZED ARBITRATION
****



The XMI arbitration scheme consists of three priority classes: HOLD, HI ARB and LOW ARB. HOLD has highest priority and, as mentioned above, guarantees that the current transmitter will be granted the bus in the next cycle. The next priority class is the collection of HI ARB requests. Within this class priority is distributed in a round-robin manner. The lowest priority class is the collection of LOW ARB requests. Within this class priority is distributed in a round-robin manner. If no node requests the bus during a certain cycle, the bus will default to a NoOp function code and correct parity.

1.7 OTHER NODE RESPONSIBILITIES

In addition to the interactions required to transfer data on the XMI, nodes are also responsible for more general operations which can be best described as housekeeping functions, as listed below.

1.7.1 Timeout/Interlock

In an effort to detect the failure of a module or transfer as early as possible, XMI transactions will be subject to a timeout. The timeout period will be <tbs> us and will be monitored by the commander in all cases. When a node begins arbitrating for the bus to issue a command, it will begin incrementing a timeout counter, which, when it reaches the extreme of its counting range, will cause an error response. An I/O adapter will send an interrupt to the CPU. A timeout error on a CPU node will cause the CPU to initiate some form of error recovery or machine check. Completion of the transaction before the timeout period has expired, will result in the interface clearing the timeout counter and continuing.

Another form of timeout mechanism will be used in the case of interlocked transactions. The Commander may issue a interlock read command to a memory or I/O location which is already locked. When this occurs the Commander should either monitor the XMI for a subsequent unlock write or implement some form of backoff algorithm before retrying the interlock read. If the location remains locked after <tbs> attempts, the node should abort the operation and report an error.

1.7.2 Automatic Retry

XMI transactions involve the possibility that a node can get access to the bus, only to find that the resource it wanted was unavailable. This results in the issuance of a 'busy' confirmation to a Command cycle by the responding node. Transfers in this category will be retried by the Commander. If the resource remains busy after <tbs> attempts, the node should abort the operation and report an error.

1.7.3 Parity

All nodes will monitor parity of the bus. If bad parity is detected during a bus cycle, the devices detecting the error will assert XMI PARITY ERROR L during confirmation cycle and ignore the cycle. The specific error recovery performed by the commander in response to XMI PARITY ERROR L assertion depends on the command and XMI CONFIRMATION code received. If bad parity is detected during the second or later cycles of multi-cycle transfer, such as write data following a write command, the transaction will be aborted.

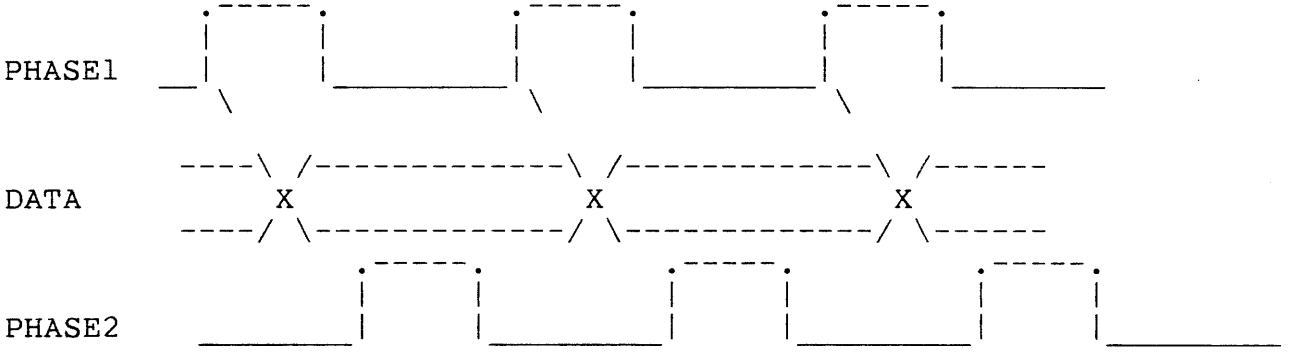
CHAPTER 2

FUNCTIONALITY

2.1 TIMING

Transactions on the XMI take place in discrete cycles delineated by the clock. The XMI uses a 2 phase clock system with the two phases called "Phase1" and "Phase2". The asserted states of the two clock phases do not overlap in time. Within this limitation, including worst case skew across the entire system, the asserted states of the two clock phases are essentially equal and as wide as possible. The period of the clock is normally 80ns. For purposes of testing and debug, the XMI clock system can be set to produce a longer period as well. The system and all XMI devices do not need to be designed to operate correctly with a clock period shorter than 80ns but they shall be designed to operate correctly with periods of up to 320ns. The XMI clock will not be stopped for any reason.

All XMI signals shall be driven onto the XMI from Phase1 latches and shall be received from the XMI into Phase2 latches. The bus usage diagrams in this section will represent data transfers as a sequence specifying the contents of the different fields of the XMI as a function of bus cycle, assuming that data is driven onto the bus enabled by a "Phase1" and received into a latch enabled by an "Phase2".

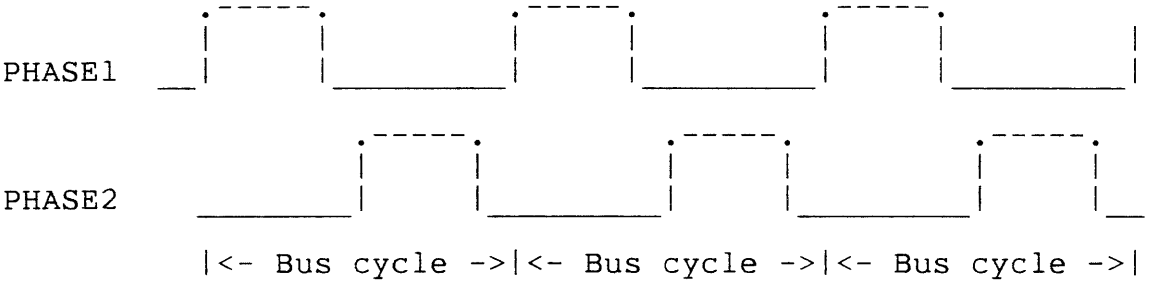


2.1.1 Clock System

The clock system in the XMI will provide the CPU, memory and I/O nodes with a 2 phase non-overlapping clock that will not stop as long as power is supplied to the system.

2.1.2 Bus Cycle

A Bus Cycle is equal to the time between normal "Phase1" clocks provided by the system timing.



2.2 PINS AND FIELD DEFINITION

2.2.1 XMI FUNCTION<2:0>

The Function field encodes the function being performed on the bus this cycle. All possible functions and their encodings are shown below:

FUNCTION	XMI FUNCTION FIELD | 2 |	1 |	0 |
No-op Cycle (NOOP)	0	0	0
Read Response Cycle			
Good Data (GRDn)	1	0	n
Corrected Read Data (CRDn)	1	1	n
Non-Data Response (NDR)	0	1	1
Write Data Cycle (WDAT)	0	1	0
Command Cycle (CMD)	0	0	1

n = sequence identification
0 -> Data cycles 0 & 2
1 -> Data cycles 1 & 3

2.2.2 XMI DATA<63:00>

The use of this field is multiplexed between command and data information. On data cycles the lines represent 64 bits of information; on Command cycles the lines represent command code, address and mask information.

2.2.3 XMI ID <4:0>

During the Command cycle and return data cycles, the ID field contains the commander's ID. This ID is used to identify the source of the request on the Command cycle and to associate returning data with the commander who issued the request on return data cycles.

An XMI Commander can have only one outstanding transaction at any time. An XMI node is allocated two Commander IDs, enabling each node to have two commanders and up to two transactions in progress at any given time.

The ID codes used by a node are assigned as follows:

Node Name	IDs
I/O Node 0 (XBI0)	0000X
I/O Node 1 (XBI1)	0001X
I/O Node 2 (XBI2)	0010X
I/O Node 3 (XBI3)	0011X

<tbs>

Fixed ID codes are required for the identification of interrupt sources and to provide for XMI to BI address translation. I/O devices can only reside in slots 0 - 3. When less that four I/O devices are used, the remaining slots may be used for CPUs or Memory.

2.2.4 XMI HI/LO ARB, BUS GRANT, HOLD

Each node on the XMI has 3 dedicated arbitration lines that connect it with the centralized arbiter. These are HI (responder) ARB, LO (commander) ARB, and BUS GRANT. In addition, there is a shared HOLD line. The actual signal names are these prefixed by XMI and the node name. All 3 arbitration lines are available in each XMI slot to allow either a CPU, memory or I/O Adapter to reside there.

The CPU asserts LO ARB if it wishes to gain the use of the XMI to begin a new transfer. If the arbiter decides to give a node the bus, the arbiter will assert (specific node) BUS GRANT during the same XMI cycle. BUS GRANT should be received by the requesting device at the end of the bus cycle and used to enable the node's drivers onto the XMI.

A memory asserts HI ARB if it wishes to use the XMI to complete a transfer, such as returning read data to the CPU. If the memory has the XMI for the current cycle, then it may reserve the XMI for the next cycle by asserting HOLD during the current cycle. HOLD is used to acquire additional cycles in a multi-cycle transfer, and is used by a commander for write transfers and by responders for contiguous data cycles for octaword and hexaword transfers. If the node asserts HOLD during the current cycle then the arbiter will certainly assert (specific node) BUS GRANT for the next cycle.

2.2.5 XMI PARITY

Parity on the XMI is computed over three fields; DATA <63:00>, FUNCTION <2:0>, and ID <4:0>. Even parity is used, where the "exclusive OR" of all bits including the parity bit is a "0". With an idle bus all fields will be "0", with the correct parity being a "0", there is no need for a node to explicitly drive it.

2.2.6 XMI CONFIRMATION<4:0>L

The confirmation lines are a five bit ECC protected field used by the receiver to notify the transmitter of the status of the data transfer. Valid responses and their respective encodings are shown below:

Confirmation	CNF<4:0>L
No Response (NACK)	0 0 0 0 0
Receiver Accepts Data (ACK)	1 0 1 0 1
Responder Busy (BUSY)	1 1 0 1 0
Deferred Response (DEF)	0 1 1 1 1

- No Response -- Indicates that no receiver has accepted a command cycle or that a data cycle contained a parity error.

- Receiver Accepts Data -- Indicates that a responder has accepted a command cycle or that a receiver has accepted a data cycle. ACK'ed read command cycles indicate that the responder will return a read response cycle at a later time. ACK'ed write command cycles are assumed successful.

- Responder Busy -- Indicates that the responder is temporarily unable to accept the command cycle. The commander should re-issue the command at a later time.

- Deferred Response -- Indicates that the responder has accepted the command, but is unable to verify completion status at this time. The responder will return a non-data response cycle indicating completion status at a later time. The deferred response in intended to be used only for I/O Space write transactions.

Confirmation timing

Cycle	0	1	2	3	4	5	6
Function	Cmd	Cmd	Wdat	Wdat	Cmd	Noop	Cmd
Conf			0	1	2	3	4

As mentioned above, XMI CONFIRMATION<4:0>L is an ECC protected field. It is vital to bus integrity that a commander knows exactly what the responder has done. This is particularly important in the case of an interlocked command or a write to an I/O register. The data and command field are adequately protected by parity from one bit errors. The implications of a corrupted CNF response justify the additional bits required to provide single bit error correction of the CNF code.

The ECC field consists of two data bits, CNF<1:0> and three check bits, CNF<4:2>. XMI CNF<3:2> is a duplication of XMI CNF<1:0> and XMI CNF<4> is computed as XMI CNF<1> xor XMI CNF<0>. Devices will check a received CNF code by verifying:

```
Test 1    0 = CNF<2> xor CNF<0>
Test 2    0 = CNF<3> xor CNF<1>
Test 3    0 = CNF<4> xor CNF<1> xor CNF<0>
```

If all tests pass, the CNF data is error free. If tests 1 and 2 fail, the CNF data contains a double bit error which is uncorrectable. Otherwise the data contains a correctable single bit error. If test 3 passes then CNF<1:0> contains valid data, otherwise the duplicate copy located in CNF<3:2> should be used.

2.2.7 XMI PARITY ERROR L

The signal XMI PARITY ERROR L is used to signal to a commander that a parity error has occurred. The signal obeys the same timing as XMI CONFIRMATION<4:0>L.

2.2.8 XMI RESET

The signal XMI RESET is used by the system to clear all XMI nodes and to initiate self test, as defined in the BI SRM. It provides a system wide mechanism for initiating an XMI power-up and initialization sequence.

2.2.9 XMI AC LOW

The AC LOW signal indicates that AC power is below the level specified for correct operation of the power supply, as defined in the BI SRM. XMI AC LOW comes onto the backplane from the power system.

2.2.10 XMI DC LOW

The DC LOW signal indicates that the system DC power is out of spec and acts as a reset to system devices. It also exhibits additional funtionality as defined in the BI SRM. XMI DC LOW comes onto the backplane from the power system.

2.2.11 XMI BAD

The XMI BAD line is identical to the BI BAD signal and, in fact, may be electrically connected to the BI BAD line of the I/O subsystem.

2.2.12 XMI CLOCKS

There are two basic clocks used for system timing on the XMI. The asserted states of the two clocks, called Phase1 and Phase2, do not overlap in time. Phase1 is essentially used to transmit data on the XMI and Phase2 is used to receive data.

2.2.13 XMI NODE ID<3:0>

Each slot on the XMI backplane will be wired with a unique 4 bit ID code. This code will be used by each node to define their Commander IDs and CSR addresses. XMI NODE ID<3:0> corresponds to bits XMI ID<4:1> of a node's Commander IDs.

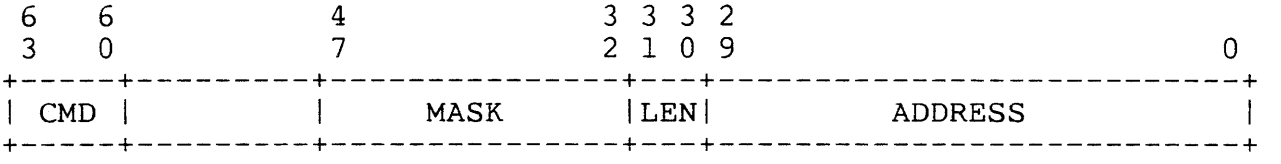
2.3 XMI CYCLE FORMAT

2.3.1 No-Op Cycle

A function code of 000, the bus default value, indicates an unused XMI cycle and should be ignored by all XMI Responders. During this cycle the values of DATA<63:00> and XMI ID<4:0> are unspecified. However, the value of XMI PARITY should be consistent with the values of DATA and ID to to avoid unnecessary logging of bus errors. The default value of XMI PARITY will produce correct parity when DATA and ID are not driven.

2.3.2 Command Cycle Format

A function code of 001 identifies an XMI Command cycle. The XMI Command cycle is used by a Commander to initiate a XMI transaction. During this cycle the Commander drives its Command ID on XMI ID<4:0> and drives command information on DATA<63:00>. Below is a definition of the various field in DATA<63:00> during the command cycle.



2.3.2.1 Command Field - The command field, located in DATA<63:60>, defines the specific transaction being initiated by the Command cycle.

The XMI command field is patterned after the BI I<3:0> field with similar decoding as follows:

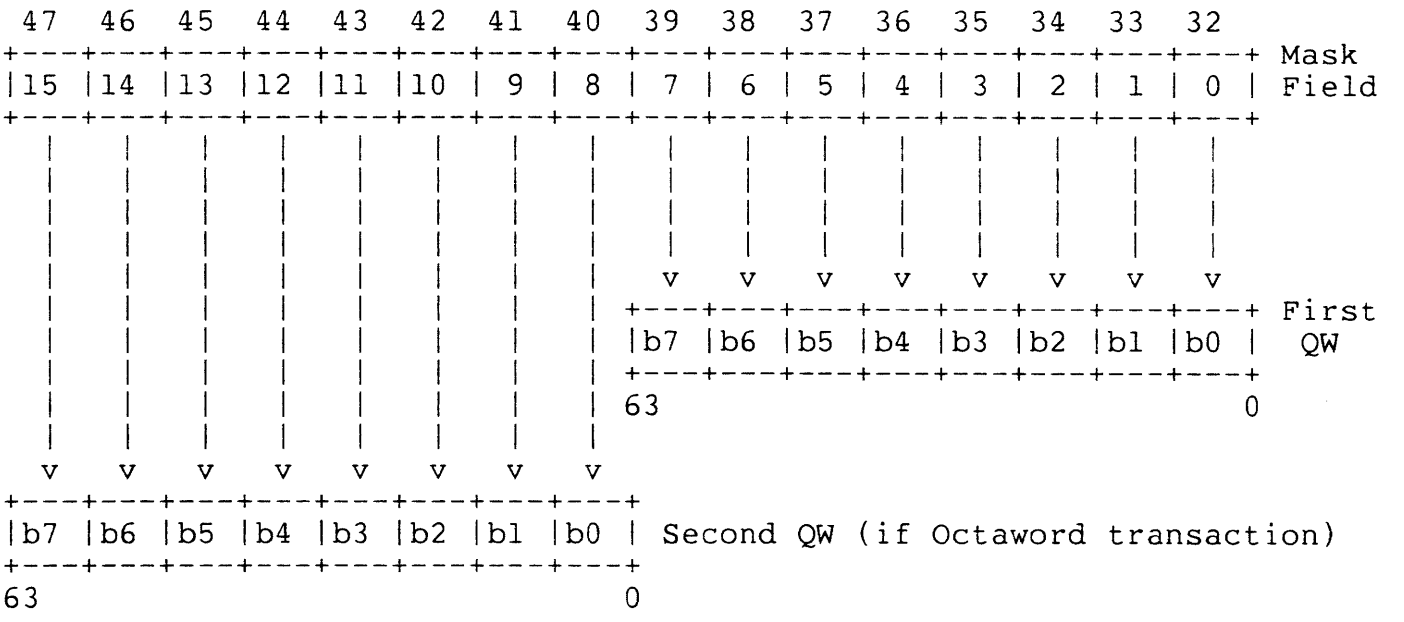
DATA<63:60>

63	62	61	60	Command
0	0	0	0	Reserved
0	0	0	1	Read
0	0	1	0	Interlock Read
0	0	1	1	Reserved
0	1	0	0	Reserved
0	1	0	1	Tag Bad Data
0	1	1	0	Unlock Write
0	1	1	1	Write Masked
1	0	0	0	Interrupt
1	0	0	1	Ident
1	0	1	0	Reserved
1	0	1	1	Reserved
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
1	1	1	1	IP Intr

2.3.2.2 Mask Field -

The mask field is located in DATA<47:32> during the Command cycle. It is used to supply write mask information for write transactions and to specify destination mask for INTR and IPINTR interrupt transactions. The correspondence between bits in the mask and bytes of the data quadword during a write transaction is shown below:

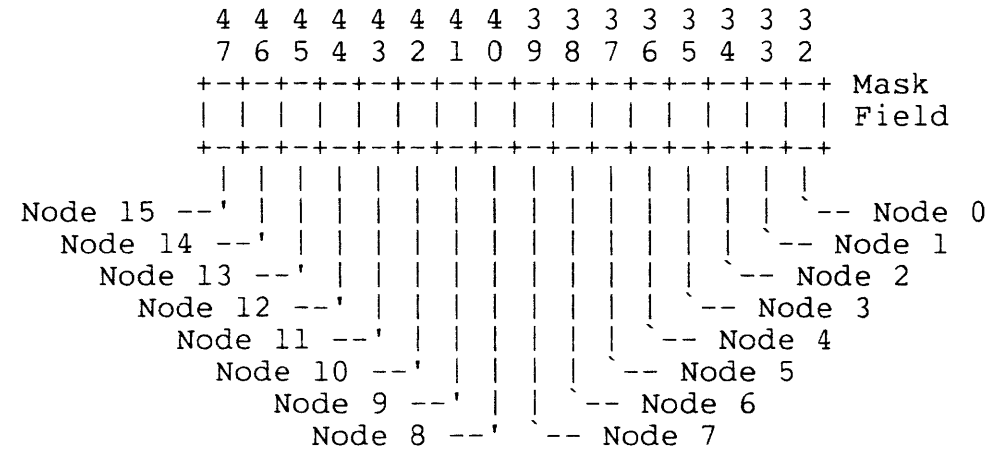
DATA<47:32>



The maximum masked write transaction is an octaword, which requires 16 mask bits in the upper longword of the command. Mask bits which are ones specify that the corresponding bytes of the following quadwords are to be replaced with the analogous byte of the data. Mask bits which are zero disable modification of the corresponding bytes in memory.

The relationship between bits in the mask and the destination of an interrupt transaction is shown below:

DATA<47:32>

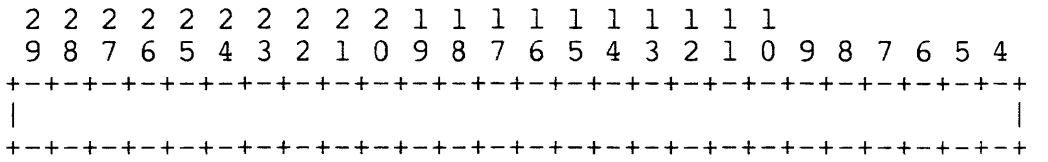


2.3.2.3 Length Field - The length field, located in DATA<31:30>, is used to define the number of words in the XMI data transfer. It is patterned after the BI Size field, except for hexaword which was not supported by the BI. The length field is decoded as follows:

DATA<31:30>

31	30	Size
0	0	Hexaword (BI unused code)
0	1	Longword
1	0	Quadword
1	1	Octaword

2.3.2.4 Address Field - The lower 30 bits of the command cycle, DATA<29:00>, define the address of an XMI read or write transaction. The number of significant bits in the address depends on the transaction type and length, as shown below.



	3 2 1 0
Read Longword	s s s s
Read Quadword	s d d d
Read Octaword	s d d d
Read Hexaword	s d d d
Write Longword	s s s s
Write Quadword	s d d d
Write Octaword	d d d d

s = significant
d = don't care

From the above figures, it can be seen that the lower four bits are significant address bits or don't care, depending on the function being requested. For longword length transactions, A<1:0> are only significant when dealing with a word-mode and byte-mode transaction in I/O space. A<1> is required for word mode and A<1:0> is required for byte mode. Octaword write transfers are assumed to be naturally aligned, allowing the lower bits of the address to be don't care status. In the case of reads, however, the situation is different because the memory does wraparound reads. Even though the operand size would indicate that some number of lower bits can be don't care, they are significant since all wrapped reads need to identify the quadword to be transferred first. This is reflected in the table above. On longword reads to I/O space, address bit<1> is listed as significant. This is due to the fact that there is no explicit READ WORD function on the XMI. When a read is directed toward a word oriented device (a BUA on the BI for instance) A<1> becomes significant in that it specifies which word is to be read from that device. The relationship between the high and low words, the state of A<1> and the data bits is:

A<1> = 1 => high word => Data<31:16>
A<1> = 0 => low word => Data<15:00>

The data returned on the opposite word of the one specified will still have correct parity, however its data is unspecified.

In the case of a longword oriented device A<1> is ignored as an address bit, and a full longword of data is returned for a read operation.

2.3.3 Write Data Cycle

A function code of 010 identifies an XMI Write Data Cycle. Write Data Cycles follow the XMI Command cycle during an XMI write transfer. During this cycle the Commander drives its Command ID on XMI ID<4:0> and drives write data on DATA<63:00>. The full 64 bits of data are used during quadword and octaword length writes. For longword length writes, only the lower longword, DATA<31:00>, is used. In this case the value of the upper longword is unspecified. In either case the full 64 bits of data are used when checking XMI PARITY.

2.3.4 Good Data And Corrected Read Data Response Cycles

A function code of 100 - 111 are used to identify return data in response to a READ, INTERLOCK READ or IDENT transaction. Codes 100 and 101 indicates that a read completes without error. Codes 110 and 111 indicates that a read operation encountered an error which was successfully corrected using ECC. The read data response code contains a sequence ID used to identify when a read data cycle has been lost due to an XMI parity error. Function codes 100 and 110 are used to return even data cycles (cycles 0 and 2), while codes 101 and 111 are used to return odd data cycles (cycles 1 and 3).

During a read data response cycle, the Responder drives the Commander's ID on XMI ID<4:0> and read data on DATA<63:00>. The full 64 bits of data are used during quadword and octaword length reads. For longword length reads, only the lower longword, DATA<31:00>, is used. In this case the value of the upper longword is unspecified. In either case the full 64 bits of data are used when checking XMI PARITY.

2.3.5 Non-Data Response Cycle

A function code of 011 defines a Non-Data Response Cycle. This transfer is used to signal the unsuccessful completion of a read transaction, as well as the completion status of an I/O Space write transaction whose command cycle received a "deferred" confirmation. During this cycle the Responder drives the Commander's ID on XMI ID<4:0> and the response code on XMI DATA<63:62>. The value of the remaining data bits, DATA<61:00>, are unspecified; however, they must contain a value consistent with XMI PARITY. The Non-Data Response Code is decoded as follows:

DATA<63:62>		
63	62	Non-Data Response
0	0	Successful
0	1	Locked
1	0	Bad Data -- Tagged
1	1	Bad Data -- Uncorrectable Memory Error

- Successful -- The "Successful" response is used to indicate that a transaction whose command cycle was confirmed as "deferred" has completed successfully.

- Locked -- The "Locked" response is used to indicate that the location specified in an interlock read transaction is already locked.

- Bad Data, Tagged -- This response is used to indicate that the location specified in a read or interlock read transaction was marked bad by a previous Tag Bad Data transaction.

- Bad Data, Uncorrectable Memory Error -- This response is used to indicate that the location specified in a read or interlock read transaction contained a multiple bit error that could not be corrected.

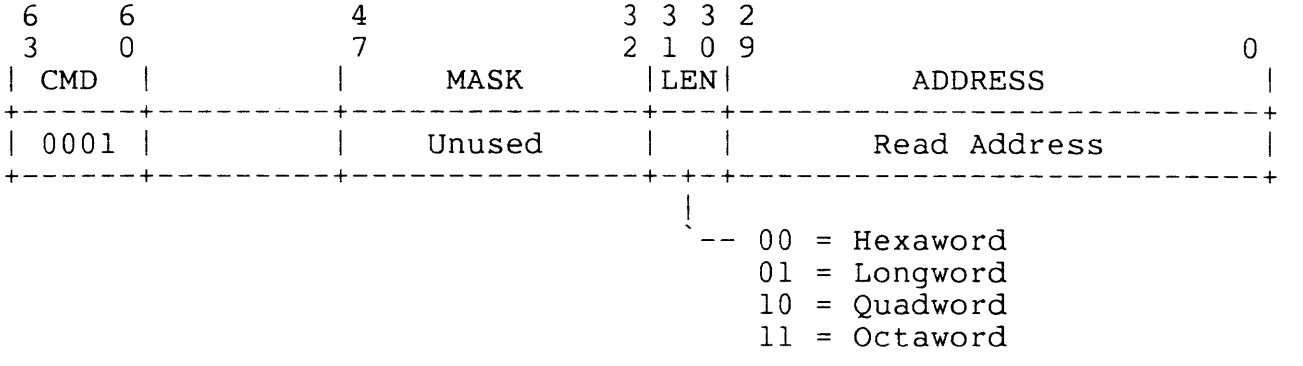
2.4 TRANSACTION TYPES

The XMI supports the following basic types of data transactions:

2.4.1 Longword, Quadword, Octaword And Hexaword Read Transactions

These transactions are used to move a longword, quadword, octaword, or hexaword of data from the responder to the commander. The data is naturally aligned and wrapped. A read transaction is initiated by the commander driving the XMI address and function lines to represent a Read Long, Read Quadword, Read Octaword, or Read Hexaword. The read command cycle is decoded by the interfaces in the system, and the one which recognizes its address latches that address and command. This device is the responder. Some time later, when the responder has the requested data, it initiates a return data transfer. Multiple transfers may be necessary to transfer all of the quadwords in a given octaword or hexaword transaction. The commander, which has been monitoring the bus traffic waiting for its return data, latches the information. The commander issues its own ID in the ID field during the command cycle. The responder returns this same ID with the return read data and this is the way the commander recognizes the return read data it requested.

Read Command

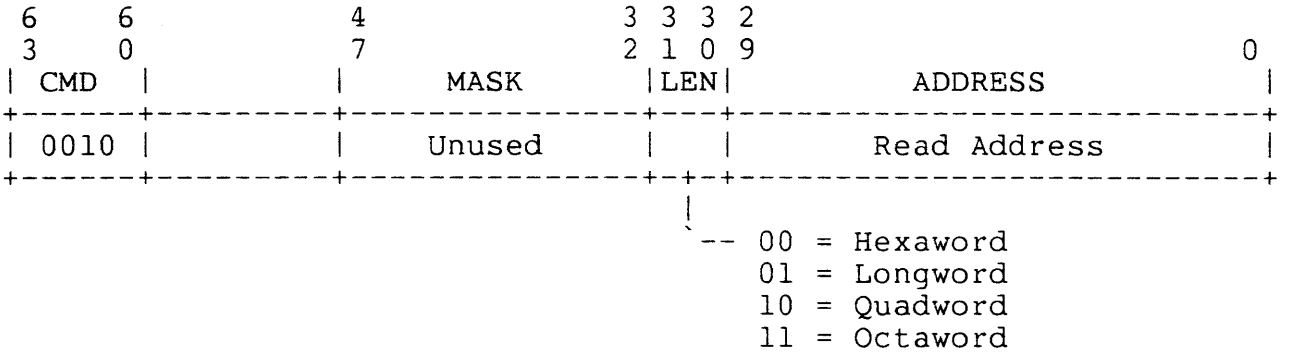


2.4.2 Longword, Quadword, Octaword And Hexaword Interlock Read Transactions

These interactions work just like the non interlocked versions described above but there is a bit more functionality. The exact effect of an interlocked transaction depends on the state of the interlock bit in the memory. If the memory is already locked, it responds to this read request with a "locked" non-data response and no data is returned. This signifies to the commander that the shared memory structure is not available; if, however, the memory is not locked, receipt of this request locks the memory to further interlocked read requests and provides the data contained

in the addressed location(s) to the commander. The corresponding transaction 'Unlock Write' is required to undo the memory lock. See below. The interlocked read transaction is used to gain access to a shared object in memory.

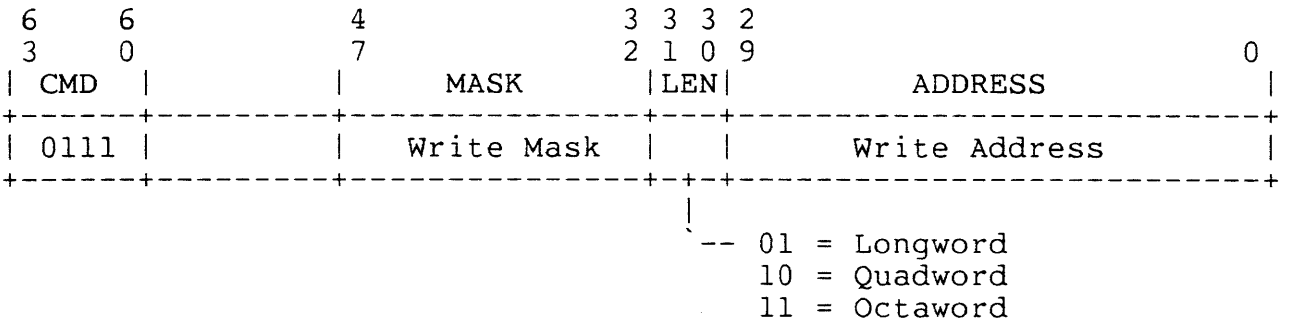
Interlock Read Command



2.4.3 Longword, Quadword And Octaword Write Masked Transactions

These transactions are used to move a pattern of bytes that fits in a longword, quadword, or octaword from the commander to the responder. The longword, quadword, or octaword block is naturally aligned. The commander gains the XMI and sends a command cycle specifying a Write Long Masked, Write Quadword Masked or a Write Octaword Masked, a byte mask, and the desired address. It follows this with one or two cycles of write data. All interfaces on the XMI decode the address and the one that recognizes the address becomes the responder. The responder accepts the command, address and data and performs the requested write. The mask field that accompanies each command and address is completely unrestricted. Each bit in the 16 bit mask field corresponds to a byte of data in the associated one or two quadwords. If the bit is 0 that byte is not written; if the bit is 1 that byte is written. The price that is paid for this flexibility is that the memory (if memory is the responder) must perform a read/modify write for a partially masked quadword, whereas it can perform a full write if all the mask bits are asserted for a given quadword.

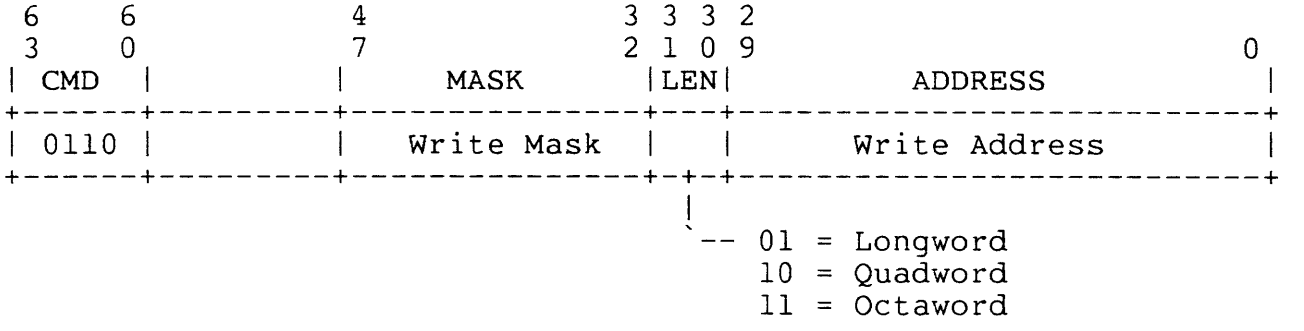
Write Masked Command



2.4.4 Longword, Quadword And Octaword Unlock Write Masked Transactions

The Unlock Write transaction is the complement to the Interlock Read. When a node successfully gains the lock in the memory, and performs the required access to the shared structure, it must then relinquish the lock when it is finished. It accomplishes this by performing a Write Unlock to the memory with whatever data is appropriate. The memory, which has been monitoring the bus traffic, notices that the transaction requested is a Write Unlock. This condition allows it to unlock the memory and to write the data as requested. All unlock writes are masked.

Unlock Write Masked Command

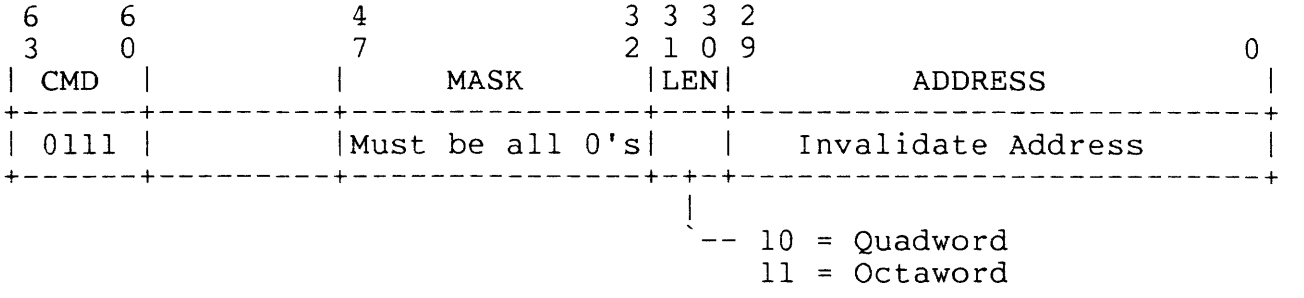


2.4.5 Invalidate Transactions

A commander can perform an invalidate transaction by issuing a Write Quadword Masked or Write Octaword Masked command with the mask field equal to all zeros. The commander gains the XMI and sends a command cycle specifying a Write Quadword Masked or Write Octaword Masked, a byte mask of all zeros, and the invalidate address. It follows this with one dead cycle, in place of write data, to maintain a minimum transaction time. Each invalidate responder accepts the command and address and performs the requested address comparison. The invalidate block size (QW, OW) is specified in the A<31:30> length field.

Confirmation of an invalidate command is provided by the memory whose address space contains the invalidate address. This is consistant with memory acknowledgement of normal masked writes, even though an invalidate will otherwise be ignored by a memory node.

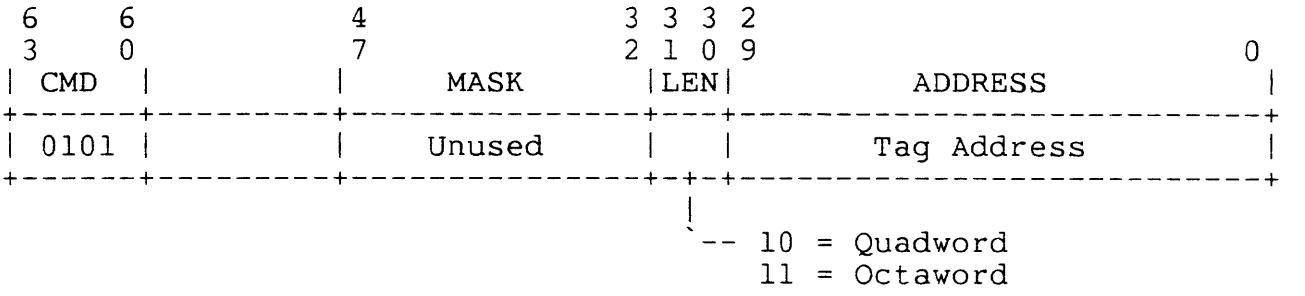
Invalidate Command



2.4.6 Quadword And Octaword Tag Bad Data Transactions

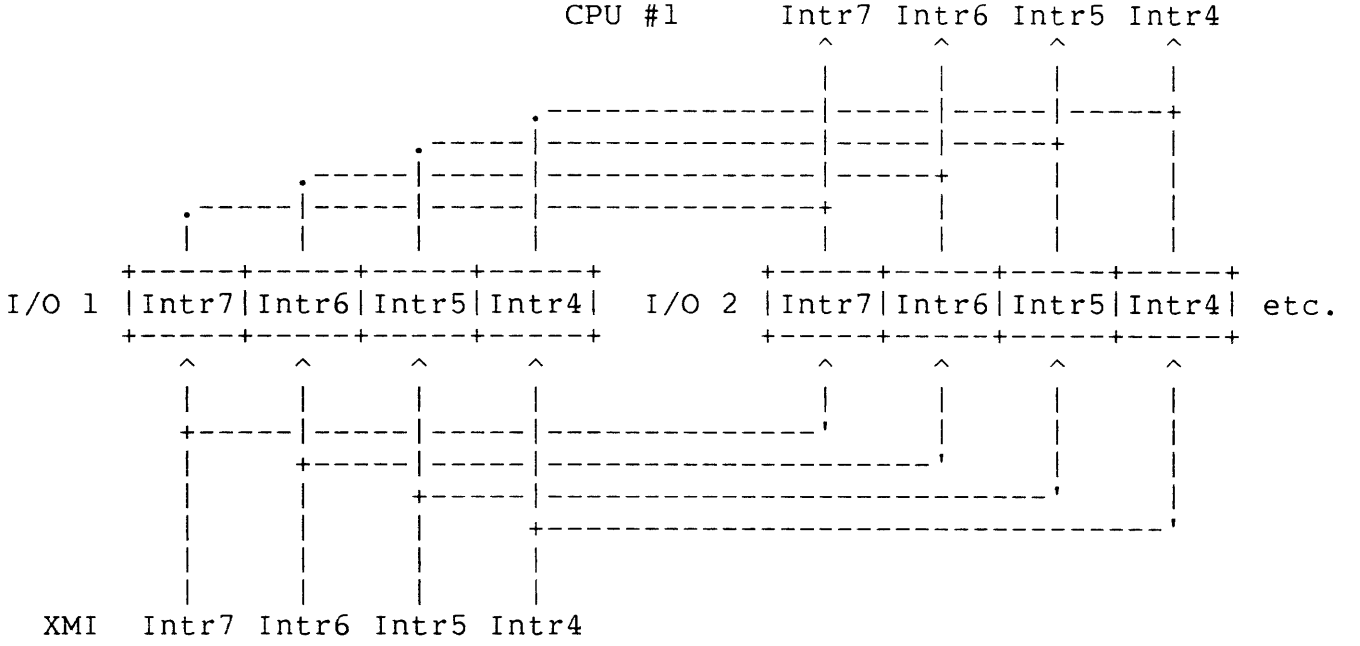
The Tag Bad Data transactions are used to mark locations in memory as containing incorrect data. Subsequent reads to a location marked bad should return the Bad Data - Tagged non-data response in place of read data. A write transaction with all mask bits asserted can be directed to the marked location to clear the bad data status. The Tag Bad Data consists of a single cycle command transfer.

Tag Bad Data Command



2.4.7 INTR And IDENT Transactions

The XMI will have an INTR and IDENT command, much the same as the BI. Any of the four (max) I/O devices can send out an INTR to one or more CPU nodes, as designated by a destination mask. Each CPU XMI interface will have a set of 16 interrupt pending flops, 4 wide for BR<7:4 and 4 deep for each of the 4 I/O devices. A composite interrupt will issued to the CPU at each BR level.



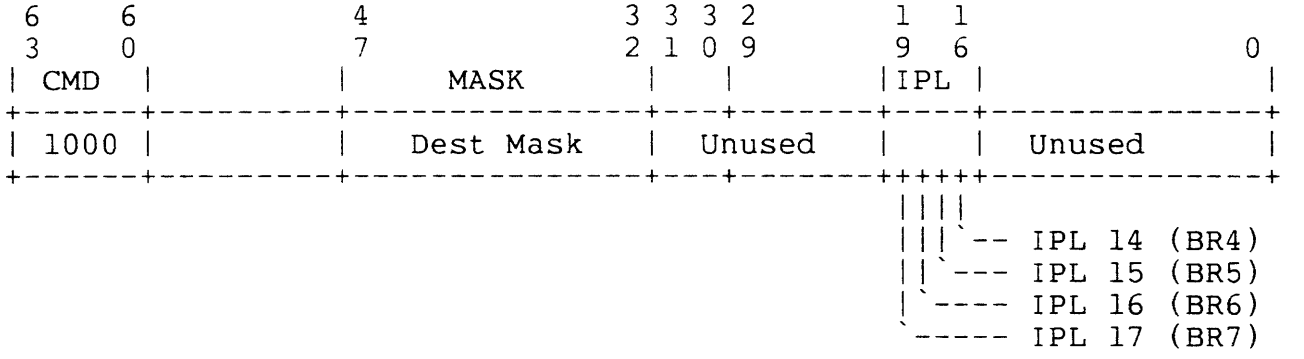
One of the processors will eventually issue an IDENT at a selected level <7:4> and its XMI interface will choose one interrupting node to send it to. The same XMI interface will then clear that I/O interrupt pending flop to the CPU, but others

(if any) will remain in parallel to maintain the CPU interrupt request. An interrupt vector will eventually be sent to the CPU, which will perform the necessary service routine and then send out another IDENT, etc.

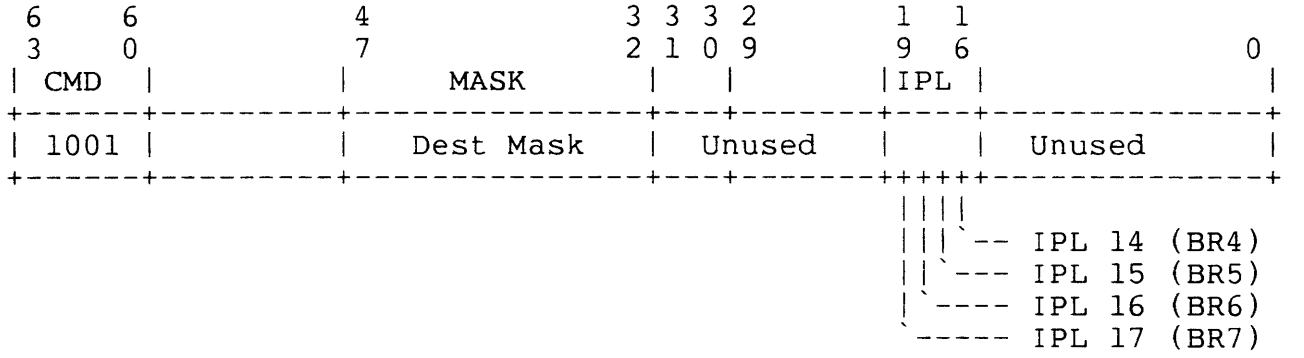
Interrupting nodes do not have to reissue their interrupts after one node/level is serviced. Each CPU bus interface will monitor the XMI for IDENTs issued by another. An IDENT issued by one CPU node to an interrupting device will cause the other interrupted nodes to clear their corresponding interrupt pending flop to the CPU, if it had been previously set. An interrupting node is not be allowed to send more than one interrupt at the same level to one or more CPUs.

It is possible that more than one CPU will issue an IDENT to its XMI interface for the same interrupt. The first CPU node to win the XMI will process the interrupt and the bus interfaces at the other processor nodes will clear their corresponding interrupt pending flops. However, the processors of the losing nodes will still have an IDENT outstanding which has to be resolved. This will be handled in the same manner as a passive release, in which the interrupting device never responds with a vector.

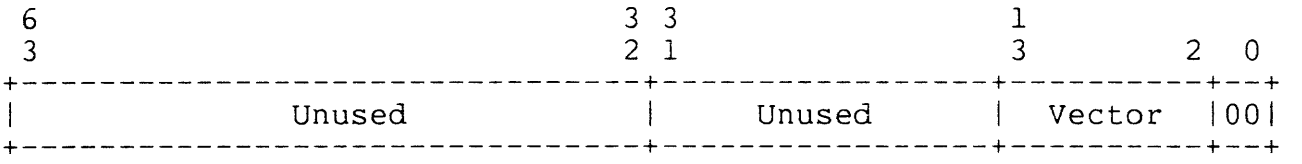
INTR Command



IDENT Command



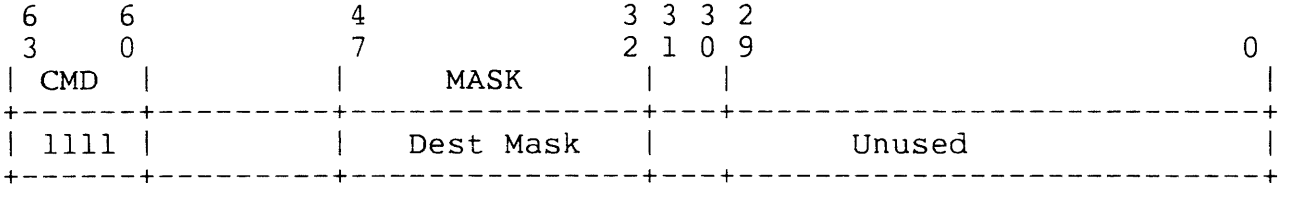
IDENT Response (Good Data Read Response -- Function Code = 100)



2.4.8 IPINTR Transactions

Interprocessor Interrupt is a single cycle transfer used for interprocessor communications. The IP INTR transaction contains a 16-bit destination field (1 bit per node) indicating which nodes are to be interrupted. Each processor node maintains a set of 16 IP INTR pending flops, one per node, readable by software. Processors targetted by an IP INTR transaction are interrupted at IPL 14 (BR4) with an interrupt vector of 80H, as specified by the VAX SRM, and additionally set the IP INTR pending bit corresponding to the commander of the IP INTR transaction. The pending bit may be cleared by a subsequent write to the IP INTR pending register by software.

IPINTR Command

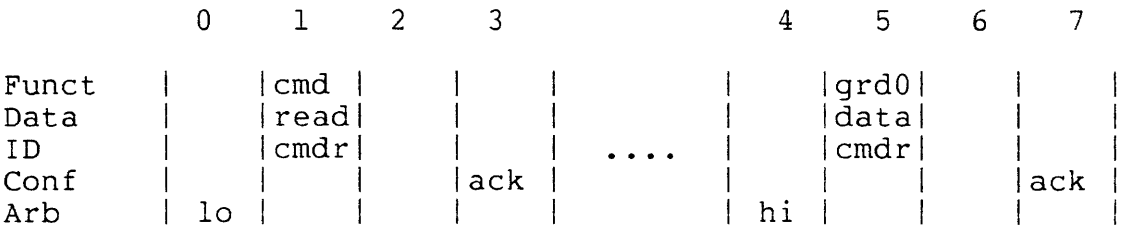


2.4.9 Timing Diagrams

The following sections will show the different types of bus transactions on a cycle by cycle basis.

2.4.10 Single Quadword Reads

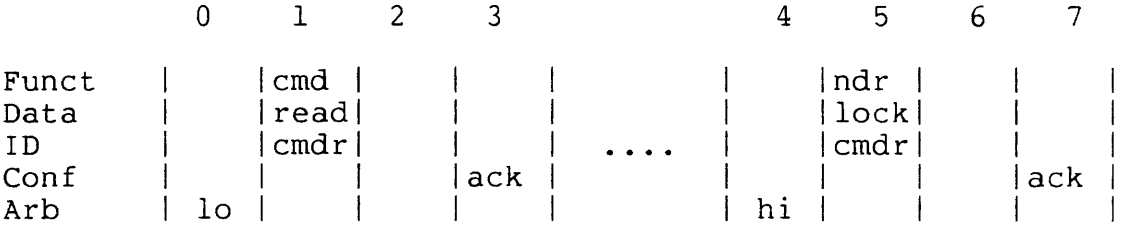
Included in this category of transactions are four types, 1) Read Longword, 2) Read Longword Interlocked, 3) Read Quadword, and 4) Read Quadword Interlocked. These reads consist of a command transfer followed by a return data transfer as shown below:



The two transfers are the command (funct = 'cmd') and the read data response (funct = 'grd0'). The commander arbitrates for the bus in cycle 0, and wins. In cycle 1 it drives the function, command, address of the read, and its own ID (for use later to identify the returning data). In cycle 3, the responder confirms receipt of the information.

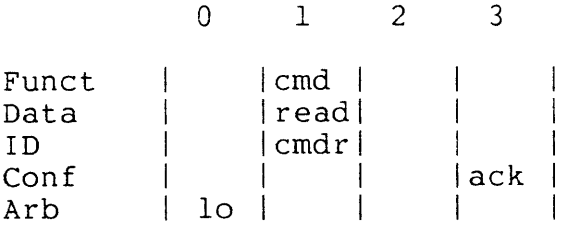
Sometime later (call it Cycle 4) the return data transfer begins with the responder arbitrating for the bus. Having won it, in cycle 5 it drives the function, the data, and and the commander's ID. The status of the returning data is specified in the read response function code, either 'good read data' or 'corrected read data'. The commander, having been monitoring the bus for its returning data, sees the ID match, and the fact that the transfer contains returning data and latches the information.

If the particular transaction requested had been an interlocked read and if the memory was interlocked, it would have provided a "locked" non-data response in place of the returned data. No further action on the request by the memory would take place.

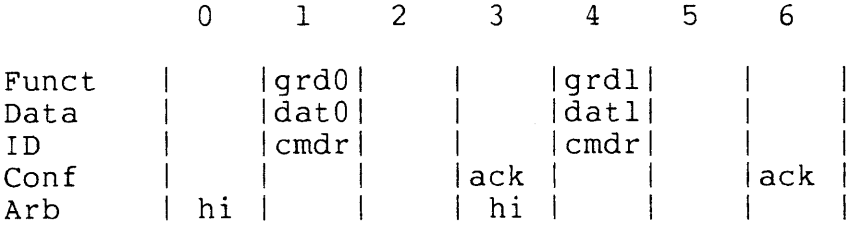


2.4.11 Multiple Quadword Reads

The transactions included in this category are: 1) Read Octaword, 2) Read Octaword Interlocked 3) Read Hexaword, and 4) Read Hexaword Interlocked. These read transactions move multiple quadwords of data from the responder to the commander. The lengths of the data are 16, 16, 32, and 32 bytes respectively. Shown below is the command transfer of the transaction. As in the case of the single quadword transactions above, the interlocked read would merely involve checking the state of the interlocked bit in the memory and qualifying the request based on its state.



The transfer diagrammed above is the command cycle for either of the multiple quadword reads - octaword or hexaword. Below is a diagram of the return data transfer applicable to octaword reads. Bus usage during hexaword reads is shown later. The case shown is a set of read response cycles for an octaword read.



The transfer above moves four longwords of data. The function field of the bus in cycle 1 says 'good read data 0' with the ID field identifying the intended receiver (the transaction commander). Cycle 4 is identified as a 'good read data 1' cycle. Each cycle provides a new quadword of read data, the ID remains unchanged. Read data may be returned in continuous cycles, if desired, through the use of HOLD (see example below). The transmitter asserts HOLD in the first cycle to insure that it maintains use of the bus long enough to complete the transfer. HOLD is considered to be the highest priority arbitration line, and thus guarantees use. Interfaces are constrained to a maximum number of consecutive cycles in which they can assert HOLD. The confirmation is returned to the commander, as always, two cycles after the command cycle.

	0	1	2	3	4
Funct		grd0	grd1		
Data		dat0	dat1		
ID		cmdr	cmdr		
Conf				ack	ack
Arb	hi	hold			

The following sequence illustrates the series of events during a return data of hexaword length.

	0	1	2	3	4	5	6	7
Funct		grd0	grd1	grd0		grd1		
Data		dat0	dat1	dat2		dat3		
ID		cmdr	cmdr	cmdr		cmdr		
Conf				ack	ack	ack		ack
Arb	hi	hold	hold		hi			

2.4.12 Masked Longword And Quadword Writes

There are four types of writes that can occur. These are 1) Longword Write Masked, 2) Longword Unlock Write Masked, 3) Quadword Write Masked, and 4) Quadword Unlock Write Unlock Masked. These transactions move some number of bytes as specified by the mask field. All combinations of the eight mask bits are valid as far as the protocol is concerned, however, not all combinations are implemented by all nodes. See the spec for the particular node in question for further details.

	0	1	2	3	4
Funct		cmd	wdat		
Data		wrtm	data		
ID		cmdr			
Conf				ack	ack
Arb	lo	hold			

The commander arbitrates as usual and upon winning the bus, drives the appropriate write command, the intended address, the data mask, and its own ID and asserts HOLD to signal that it will need the next cycle also. In cycle two, it identifies the cycle as a 'write data' type and provides the write data, but no ID field (it identified itself in the command cycle). Cycles 3 and 4 show the confirmation from the responder.

2.4.13 Masked Octaword Writes

There are two other types of writes that can occur in addition to the ones already considered. These are 1) Octaword Write Masked, and 2) Octaword Unlock Write Masked. In much the same way as the several multiple quadword reads, the multiple quadword writes differ in the length of the data moved, with all else remaining the same. An additional consideration in the case of multiple writes is that they can be masked, whereas, the multiple quadword reads cannot. An octaword write is shown below.

	0	1	2	3			
Funct		cmd	wdat	wdat			
Data		wrtm	dat0	dat1			
ID		cmdr					
Conf				ack	ack	ack	
Arb	lo	hold	hold				

The multiple quadword writes like their read counterparts identify the first cycle of the transfer as a read or write with the length desired; successive cycles are identified as write data cycles. After providing the command information, successive cycles provide new data. HOLD remains asserted, maintaining use of the bus for the commander.

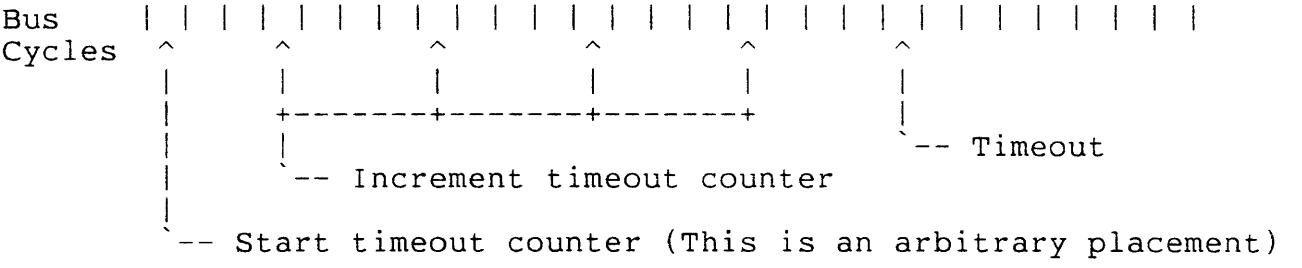
2.4.14 Timeout

The XMI protocol specifies two basic types of timeouts. The first concerns time limits on the length of a transaction, from initial attempts to issue the command to completion of the transaction. This time constraint will identify problems which fall into one of three general classes; 1) no return read data, i.e. you asked for it and you never got it, 2) no access to the bus (can never win an arb cycle), and 3) no access to a device (receiver busy or no response).

Another form of timeout mechanism will be used in the case of the memory subsystem during interlocked transactions. The CPU may issue a read lock command to a memory which is already locked. In this case, the memory will return an ACK confirmation to verify that the command was received, and will then place the read lock in its command queue. If the memory is still locked when the command is processed, the memory will return a LOCKED "non-data" response to the CPU. It is the responsibility of the CPU at that point to monitor the XMI for a subsequent unlock write before retrying the read lock. An alternative to this is for the CPU to implement some form of backoff algorithm before retrying the read lock. In either case, a timeout counter will

be required to resolve a lockup condition.

2.4.15 Timeout Mechanism



Once a timeout occurs, the timed out operation should be halted, since it was not able to complete in the timeout period. This will allow servicing of the resulting interrupt in most cases.

2.5 XMI ERROR HANDLING

The XMI bus protocol has been designed to support recovery from all single bit errors involving transactions directed to memory space. On every XMI cycle, all receivers check parity; any receiver detecting a parity error will assert XMI BAD PARITY, suppress ACK CNF and ignore the cycle. Memory responders will abort a write or unlock write transaction if valid write data cycles do not occur during the timeslots indicated by the transaction length. In the case of interlock read, the memory responder will permanently set the lock bit only when the commander confirms (with ACK) all data cycles in the interlock read transaction.

2.5.1 Error Recovery

Error recovery is the responsibility of the commander; the table below shows the error recovery procedure as a function of XMI command, cycle and error indication. In many cases the XMI error recovery procedure involves re-issuing the failing transaction. Should the transaction fail again on the second attempt, the commander logs a hard error. Error recovery for certain write transactions results in the generation of a invalidate transaction (write mask transaction with all mask bits

deasserted) following the write transaction; this is used to guarantee that all caches remain coherent when one detects a parity error during a write command cycle. Since I/O space references may have side effects, the table below does not cover I/O space error recovery. Also, since a `deferred' confirmation is only used in I/O space, it is not included in the table.

XMI Error Recovery

	Read	Write	Inlk Read	Unlk Write
Command Cycle				
No XMI Bad Parity:				
ACK	OK	OK	OK	OK
NACK	NXM Error	NXM Error	NXM Error	NXM Error
BUSY	Re-issue	Re-issue	Re-issue	Re-issue
XMI Bad Parity:				
ACK	OK	issue Inval after write	OK	issue Inval after unlkwt
NACK	Re-issue	Re-issue	Re-issue	Re-issue
BUSY	Re-issue	Re-issue	Re-issue	Re-issue
Data Cycle				
ACK	-- (not app)	OK	--	OK
NACK	--	Re-issue	--	Re-issue
Read Sequence	Re-issue	--	Re-issue	--
Timeout	Re-Issue	Re-Issue	Re-Issue	Re-Issue

	Intr	Ident	IP Intr
Command Cycle			
No XMI Bad Parity:			
ACK	OK	OK	OK
NACK	Error	Passive	Error
BUSY	Re-issue	Re-issue	Re-issue
XMI Bad Parity:			
ACK	OK	OK (a)	Re-issue(b)
NACK	Re-issue	Re-issue	Re-issue
BUSY	Re-issue	Re-issue	Re-issue
Data Cycle			
ACK	--	--	--
NACK	--	Error (c)	--
Sequence	--	--	--
Timeout	--	Error (c)	--

Notes: (a) May result in passive release (NXM) in certain situations.
(b) May result in extra IP INTR in certain situations.
(c) Even if re-issued, interrupting condition is probably lost.

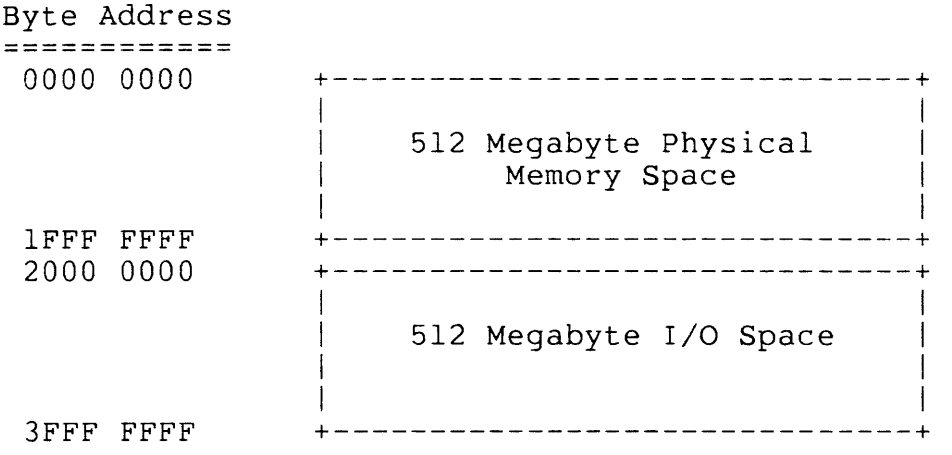
2.6 ADDRESSING

The XMI supports a 1 gigabyte (2**30 byte) address space. As in the 780, this address space is divided into two equal areas. In the low (lower numbered addresses) half resides physical memory; the high half contains the machine's I/O space.

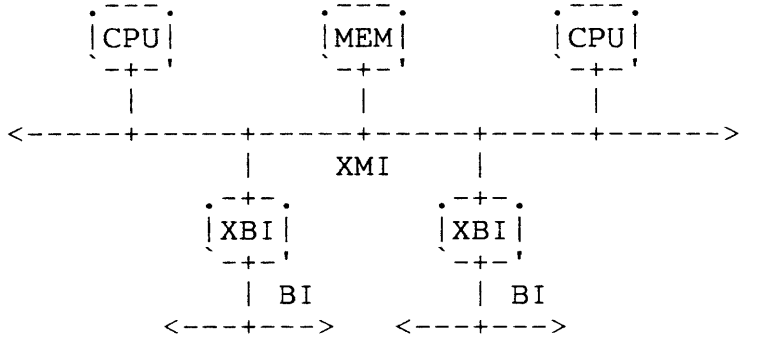
The following description concerns only the addressing methodology used on the XMI and the XBI/BI. Neither the UNIBUS nor the QBUS will be supported on the XMI. Addresses are in hex unless otherwise noted.

2.6.1 XMI Addressing

The figure below illustrates the division of the address space:

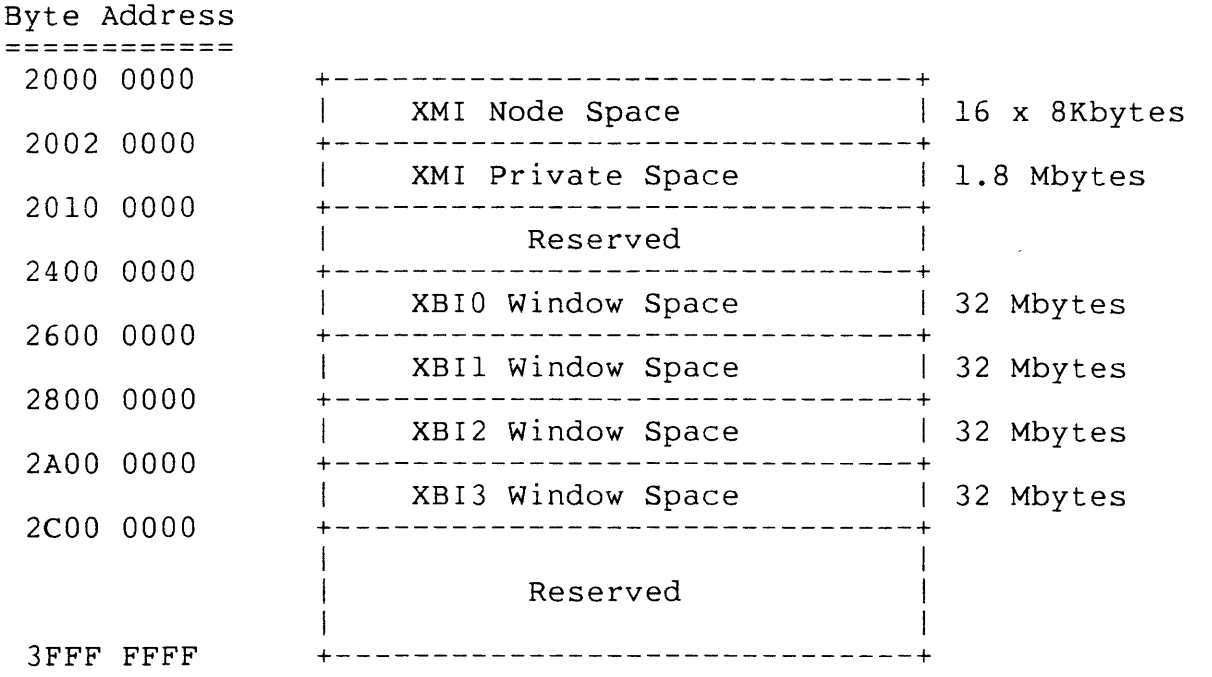


The division of the CVAX/Rigel address space in the I/O region is further defined to accommodate XMI nodes' and BI nodes' needs for address space. Shown below is a diagram of the hierarchy of busses supported in CVAX/Rigel systems.

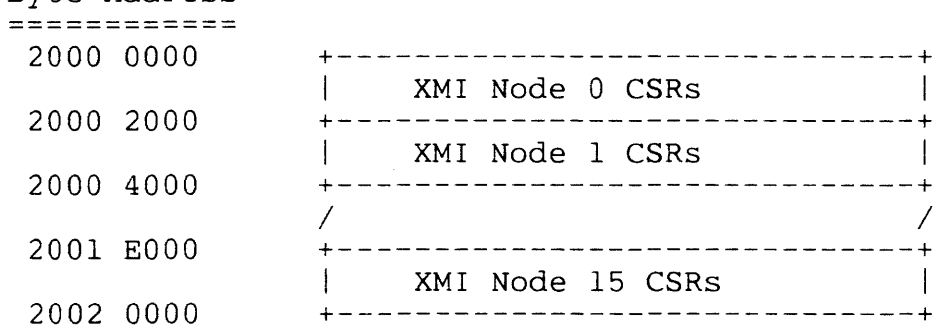


As previously mentioned, the upper two bits of an XMI address are used to define transfer size. A/D<29> selects between the memory and I/O spaces. A/D<29> = 0 selects physical memory space, and A/D<29> = 1 selects I/O. Addresses in memory space merely reference a particular location within the memory subsystem (assuming that the location exists). References to locations in the physical memory space which do not exist in the current

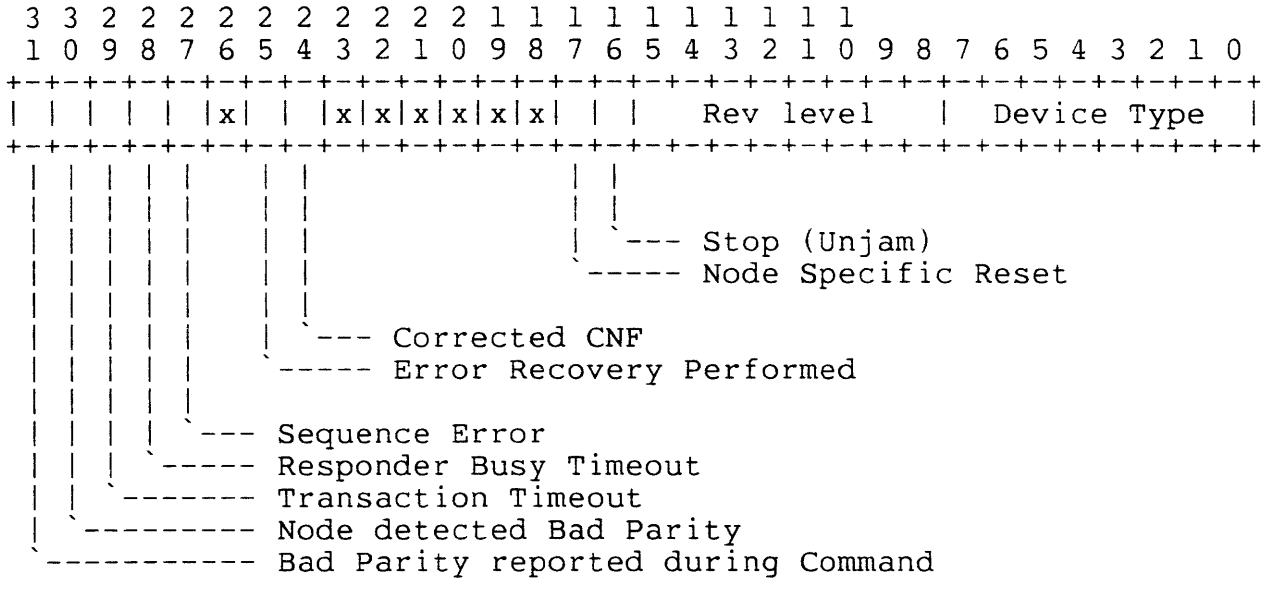
configuration will receive the 'no response' confirmation. Further decoding of the address in this case is done in a manner which is transparent to the programmer.
Within I/O space addresses locations are allocated as follows:



2.6.1.1 XMI Node Space - XMI Node Space is a collection of 16 8Kbyte regions located from 2000 0000 - 2001 FFFF. Each XMI node is allocated an 8K byte region for node control/status registers. The starting address of the 8K region associated with a given node is computed as 2000 0000 + NodeID * 2000.



2.6.1.1.1 Error Register - XMI nodes will be expected to implement an error register which will be used to store the type of error detected so that it can be read by software. Included in the error register will be identification of the node device type and revision level. Format of the XMI error register is as follows:



- Stop -- Stop corresponds the BI STOP function. When the Stop bit is set by writing a 1 to this location, it causes the receiving node to halt its internal sequencers and return to an unjammed state.

- Node Specific Reset -- When written to a 1, causes the receiving node to reset all internal states.

- Corrected CNF -- A single bit error was corrected in the CNF code.

- Error Recovery Performed -- System was able to recover from transient error.

- Sequence Error -- Missing quadword(s) in returned read data.

- Responder Busy Timeout -- Responder unable to accept command cycle.

- Transaction Timeout -- Responder does not complete transaction.

- Node detected Bad Parity -- Parity error observed locally by XMI node.

- Bad Parity reported during Command -- Commander detects parity error during command cycle.

2.6.1.2 XMI Private Space - XMI Private Space is a 1.8 MByte address region containing the reset address as required by the uVAX architectural subset. References to XMI private space will be serviced by resources local to a node, such as local device CSRs and boot ROM, and will not be broadcast on the XMI.

2.6.1.3 XBI Window Space - XBI Window Space consists of four 32 MByte address regions used for XMI to BI transaction windowing. Longword length references directed to an XBI Window Space will be re-issued on the appropriate BI. XMI transactions are translated into the corresponding BI transaction. The BI address for the transaction is computed as 2000 0000 + Offset, where Offset is the difference between the XMI Address and the start of the appropriate XBI Window Space.

2.7 DC LOW AND RESET

DC LOW and RESET function together in exactly the same manner as defined in the BI SRM. Basically, DC LOW indicates that the system DC power is out of spec or that an XMI reset operation has been initiated. DC LOW can be used in combination with XMI RESET to clear all XMI nodes and to initiate self test. It provides a system wide mechanism for initiating an XMI power-up and initialization sequence.

On the trailing edge of DC LOW (deassertion), each node will observe the status of the RESET line. If RESET is not asserted, then DC LOW is being used to clear the control logic of each node, if so allowed. For example, memory would reset all control logic, but the data in each memory location would be preserved. If RESET is asserted, then a power-up condition is recognized and each node will perform its own initialization and self test sequence.

CHAPTER 3

ELECTRICAL DESCRIPTION

The electrical environment of the backplane is as much a part of the XMI as the protocol and hardware. This chapter will define the electrical parameters of the backplane, its physical and transmission characteristics as well as giving information about XMI drivers and receivers.

CHAPTER 4

ISSUES TO BE RESOLVED

1. How do we handle wrapped hexaword reads?
2. Do we need TAG BAD DATA to resolve undetected write errors?
3. Do we need to implement longword writes to memory space?
4. Should we extend the IPINTR and INTR mechanism to support 2 CPUs per node?
5. Should the XMI support greater than 2**30 address space?
6. How do we handle arbiter failures?
7. Should we provide ECC on DATA<63:00>?
8. Is deferred response needed?
9. How many parity bits are needed?
10. Does lack of "don't cache me" preclude some important PMI configurations?