Table of Contents

List of Figures

List of Tables

List of Waveforms

Chapter 1

1. Introduction

The SPARC™† MBus is a private, high-speed interface which connects SPARC™ processor modules to physical memory modules and I/O modules. The specification could be thought of as a generic integrated circuit pin interface specification. The interface is not intended for use as a general expansion bus on a system back-plane spanning numerous boards. Rather, it is intended to operate in a carefully controlled geographical area with the interconnect and associated circuitry located on only one printed wiring board. Modules consist of one or more integrated circuits, one (or more) of which contain the MBus interface.

The two major goals of MBus are that it be simple and that it be compatible with CMOS technology. Simplicity is achieved by having only a few well specified transactions with a minimum of options using a small set of signals. CMOS compatibility is covered by the electrical specifications and protocols.

1.1. Features

- fully synchronous, nominally 40Mhz
- circuit switched
- 64-bit, multiplexed address and data
- 64 gigabytes of physical address space
- multiple-master
- centralized arbitration, reset, interrupt distribution, and clock distribution
- overlapped arbitration with "parking'
- shared memory multiprocessor(MP) signals and transactions
- supports a write-invalidate type of cache consistency protocol

1.2. MBus Levels

The complete MBus Specification has two levels of compliance, Level 1 and a Level 2. Level 1 includes the basic MBus signals and transactions needed to design a complete uniprocessor system. Level 2 introduces additional signals and transactions needed to design a cache coherent, shared memory Multiprocessor.

A device which conforms to Level 1 of the Specification will be classified as a level 1 device while those devices conforming to Level 2 of the Specification shall be referred to as level 2 devices. Level 1 devices will function properly in a level 2 system. Since one of the intents of MBus is to allow for modular SPARC™ solutions, care has been taken to ensure all modules in an MBus system can be compatible.

1.2.1. Level 1 Overview

The Level 1 MBus supports two transactions, Read and Write. These transactions simply read or write a specified SIZE of bytes from a specified physical address. These transactions are supported

† SPARC is a registered trademark of SPARC International

using a subset of the MBus signals, namely a 64-bit multiplexed address/data bus (MAD[63:0]), an address strobe signal (MAS*), and an encoded acknowledge on three signals (MRDY*, MRTY*, MERR*). Additional Level 1 signals support arbitration for modules (MBR*, MBG*, MBB*) as well as interrupt inputs (IRL[3:0]), interrupt output (INTOUT*), reset (RSTIN*), asynchronous errors (AERR*), scan (SCANDI, SCANDO, SCANCLK, SCANTMS1, SCANTMS2) and module identification (ID[3:0]). The MBus reference clock (CLK) completes the signal requirements for a Level 1 system.

It is assumed that there are **central** functional elements to perform reset, arbitration, interrupt distribution, time-out, and MBus clock generation as shown in Figure 1. All modules with the exception of processor and Timeout modules accept an ID[3:0] input which is used as an aid to system configuration. Also the binary value input to a module on ID[3:0] is output as part of MAD[63:0] during the address phase of every transaction.

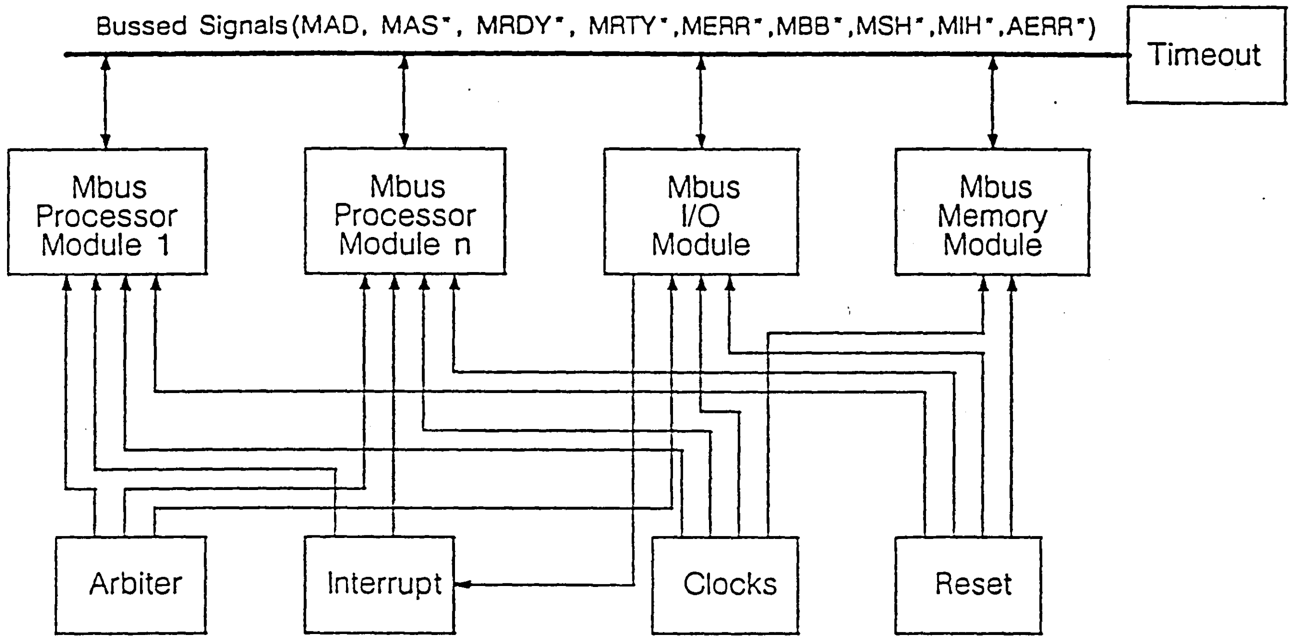


Figure 1 – MBus System Elements and their Connectivity.

1.2.2. Level 2 Overview

The Level 2 MBus includes all Level 1 transactions and signals and adds four transactions and two signals to support cache coherency. This is to facilitate the design of shared memory multi-processor systems. In Level 1, details of the caches inside modules are not visible to the MBus transactions. This changes with level 2, where many aspects of the caches are assumed as part of the new MBus transactions. To participate in sharing between processor caches on an MBus using Level 2 transactions, a cache must minimally support a "write back" policy, an "allocate" policy on write misses, and have a block or sub-block size of 32 bytes. Cache lines are assumed to have at least five states (invalid, exclusive clean, exclusive dirty, shared clean, and shared dirty).

The additional transactions present in Level 2 systems are Coherent Read, Coherent Invalidate, Coherent Read and Invalidate, and Coherent Write and Invalidate. The two additional signals are Shared (MSH*) and Inhibit(MIH*). Coherent transactions, with one rare exception (Coherent Write and Invalidate used with write through caches) have SIZE = 32 bytes. The cache coherency protocol is a "write invalidate" protocol, where the cache being written issues a Coherent Invalidate transaction if the line is not exclusive. This indicates to all caches that they should immediately invalidate the line since it will contain "stale data" after the write completes. All caches "snoop" Coherent Read transactions and assert MSH* if the address of the transaction is present in their cache. By observing MSH*, caches can update the state of the lines they hold. If a cache is the "owner", it asserts the signal MIH* to tell memory not to send data and then supplies the data to the requesting cache. Coherent Read and Invalidate, and Coherent Write and Invalidate are simply the combination of a Coherent Invalidate with either a Coherent Read or a Write. Their purpose is to reduce the quantity of MBus transactions needed and thus conserve bandwidth. Figure 2 shows a simplified MBus MP system.

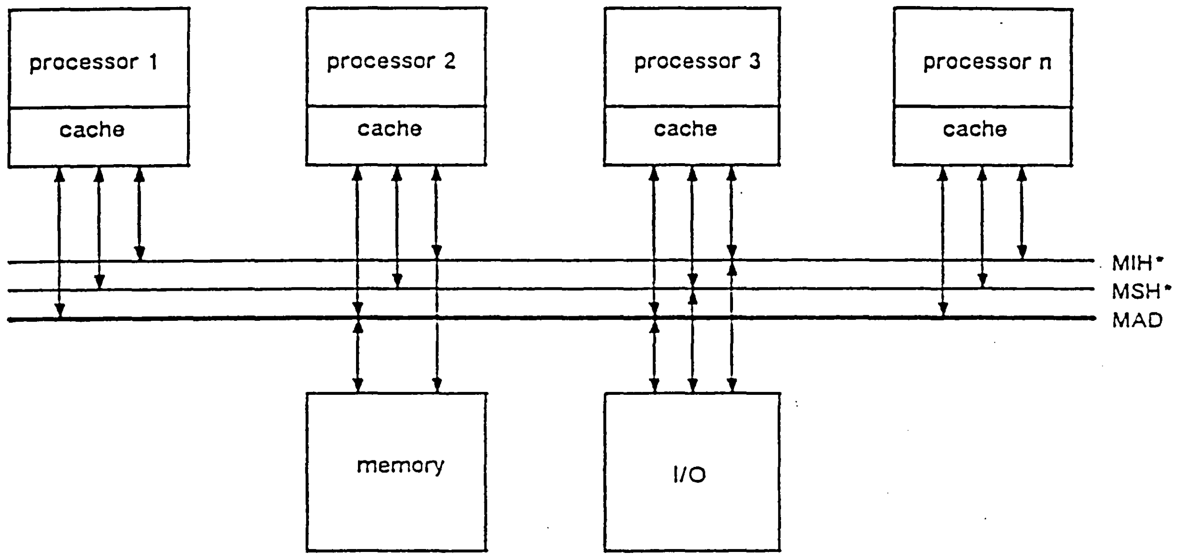


Figure 2 - Standard Level-2 Shared Memory MP.

1.3. Definitions

0x	0x as a prefix to a number, indicates that the number is hexadecimal.
MASTER	A module is an MBus master when it "owns" the MBus. Masters assert the signal MAS* to initiate transactions.
SLAVE	A module is an MBus slave when it responds to transactions directed at it. Slaves assert the signals MRDY*, MRTY* and MERR* as an acknowledgment to an MBus master.
BLOCK	A 32 byte data burst transfer size. This quantity is equal to the block (or sub-block) size of the system caches.
TRANSACTION	A transaction is a complete MBus operation such as a Read or Write. It is made up of one address cycle and one or more data acknowledgment cycles. Due to RETRY and RELINQUISH and RETRY acknowledgments a transaction may be repeated many times.
BURST TRANSFER	A multi-data-cycle MBus data transfer. MBus allows 16-byte, 32-byte, 64-byte,and 128-byte burst transfers.
BLOCK TRANSFER	A special case of MBus burst transfer, where the data transfer size is equal to the block size (32 bytes).

OWNED	A block of data is said to be owned when there is one (and only one) cache in the system which is responsible for writing it back to memory as well as supplying the block when requested by other caches. If no cache is the owner, memory is considered the owner.
SHARED	A block of data is said to be shared when more than one cache in the system currently possesses a valid copy of it.
EXCLUSIVE	A block of data is said to be exclusive when there is only one cache in the system which contains a valid copy of it.
DIRTY	A block of data is said to be dirty when it has been written to in a write-back cache. Dirty blocks must be written back to main memory if displaced (victimized). A dirty block is "OWNED" by that cache block.
VALID	A block of data is said to be valid when it is present within a processor's cache and can be supplied to that processor upon request. Valid means the cache line is in one of four states: exclusive clean, exclusive dirty, shared clean, or shared dirty.
WRAPPING	This concerns the order in which data will be delivered for multi-cycle transfers. For MBus, Read transfers larger than 8 bytes require multiple cycles. Wrapping implies that the data to be delivered first is specified by the low order address bits. i.e. the transfer address may not be burst size aligned.

1.3.1. Compliance and Additional Features.

Full compliance with either Level 1 or Level 2 implies a certain minimal functionality within the MBus modules, particularly Level 2 modules where details of the module caches are exposed. Modules may contain more than this level of functionality either to enhance performance, or to enable building a broader range of systems from MBus components. This is at the discretion of module designers and is implementation specific. An example of a performance enhancement is reflective memory support where memory is updated by observing the data being transferred from cache to cache . An example of functionality enhancement is support for simple second level caches. It is not within the scope of the MBus specification to cover details of these enhancements. MBus is merely a defined set of transactions operating on a defined set of signals. Module enhancements will be covered as "application notes" by module designers and vendors. See appendix B for some insight into the nature of the two enhancements mentioned above.

1.4. Basic Assumptions for Level 2

- Consistency Quantity

  The data quantity upon which consistency will be maintained will be a cache block or, when implemented, a cache sub-block. It is also assumed that the same (sub-)block size will be used throughout the system and corresponds to the MBus Block size of 32 bytes.

- Cache Write Policy

  While playing the Level 2 consistency game, all caches in the system will follow a write-back policy with write allocate.

- Cache Consistency Protocol

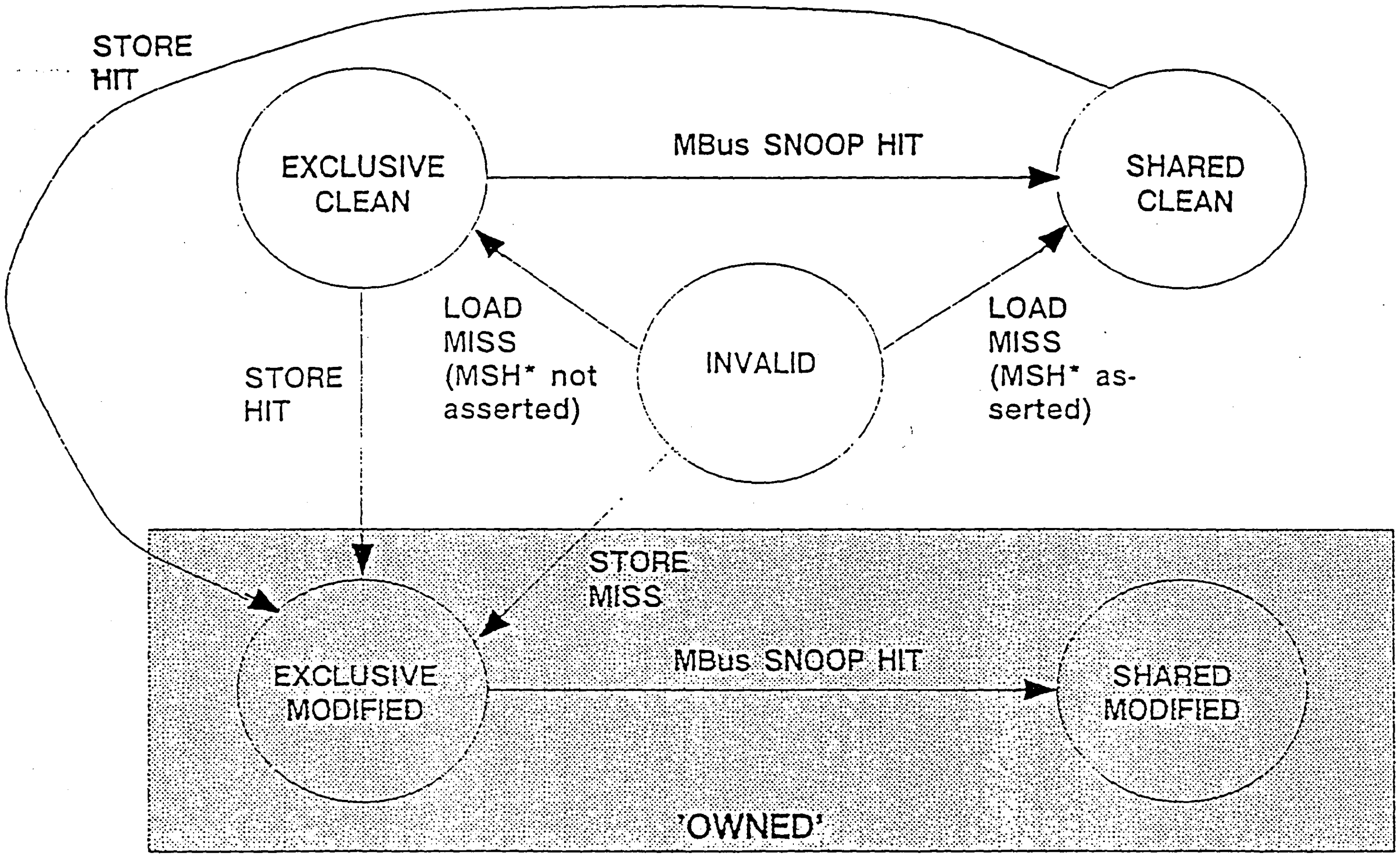
  An ownership-based protocol will be employed. A simplified state transition diagram is shown below in Figure 3.

- Homogeneous Modules

  All modules using the same MBus will play the same consistency game. The modules will appear identical as far as the MBus is concerned, yet they may differ internally.

- Second-Level Caches

  It is assumed that all second-level caches will be physically addressed.



SOFTWARE FLUSH OR MBUS COHERENT OPERATIONS WITH INVALIDATE (CI, CWI, CRI) CAUSE ANY STATE TO GO TO INVALID. FLUSH AND VICTIMIZATION WILL INVOLVE COPY-OUT IF OWNED.

Figure 3 - Simplified cache block state diagram.

Chapter 2

2. Signal Definition

2.1. Physical Signal Summary

Table•1 summarizes all the MBus Module physical signals. An asterisk (*) following the signal name indicates that the signal is active low (true when '0').

Physical Signals					
Signal Name	Signal Description	Output	Input	Line Type	Sig Type†
MCLK	MBus clock	clock buffer	Mast./Slv./Arb.	dedicated	BS
MAD[63:0]	Address/Control/Data	Master/Slave	Master/Slave	bussed	TS
MAS*	Address strobe	Master	Slave	bussed	TS
MRDY*	Data ready indicator	Slave	Master	bussed	TS
MRTY*	Xaction retry indicator	Slave	Master	bussed	TS
MERR*	Error indicator	Slave	Master	bussed	TS
MSH*	Shared(level-2 only)	Bus Watcher	Master	bussed	OD
MIH*	Inhibit(level-2 only)	Bus Watcher	Master/Memory	bussed	TS
MBR*	Bus request	Master	Arbiter	dedicated	BS
MBG*	Bus grant	Arbiter	Master	dedicated	BS
MBB*	Bus busy indicator	Master	Arbiter/Master	bussed	TS
IRL[3:0]	Interrupt Level	Interrupt Logic	CPU Modules	dedicated	BS
ID[3:0]	Module Identifier	System	MBus Modules	dedicated	BS
AERR*	Asynchronous error out	Module	Interrupt Logic	bussed	OD
RSTIN*	Module reset in signal	Reset Logic	Master/Slave	impl depen	BS
INTOUT*	Interrupt Out	I/O Modules	Interrupt Logic	dedicated	BS
SCANDI	Scan Data In	System	Modules	dedicated	BS
SCANDO	Scan Data Out	Modules	System	dedicated	BS
SCANCLK	Scan Clock	System	Modules	dedicated	BS
SCANTMS1	Scan Tap Control 1	System	Modules	dedicated	BS
SCANTMS2	Scan Tap Control 2	System	Modules	dedicated	BS

Table 1 – Physical Signal Summary

† BS = bi-state, TS = tri-state OD = open drain

2.2. Physical Signal Descriptions

MCLK	MBus master clock. The distribution of the MCLK signal in a system is implementation dependent. For example, depending on the connector, each module on the MBus may be given one or more identical MCLK lines which could originate from a single clock generator.
MAD[63:0]	Memory Address and Data. During the address phase, MAD[35:0] contains the physical address (PA[35:0]). The remaining signals (MAD[63:36]) on the bus contain the transaction-specific information which will be described in section 2.4. During the data phase, MAD[63:0] contains the data of the transfer. The bytes are organized as shown in Figure 4. For transactions involving less than a double word (8-bytes), the data must be aligned. For example, all even-addressed words will be aligned on MAD[63:32] whereas all odd-addressed words will be aligned on MAD[31:0]. As another example, byte address 2 of an odd-addressed word will be carried on MAD[15:8] i.e. byte 6 on the MBus. Unused data lines during the data phase are undefined.

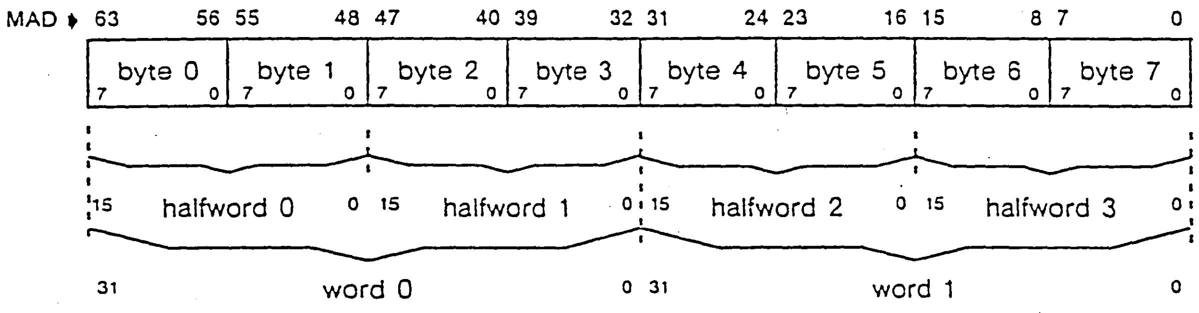


Figure 4 - Byte Organization

MAS*	Memory Address Strobe. This signal is asserted by the bus master during the very first cycle of a bus transaction. The cycle in which it is asserted is referred to as the "address cycle" or "address phase" of the transaction. For transactions that receive a Relinquish and Retry or a Retry acknowledgment, MAS* will be asserted again until the transaction gets a normal or error acknowledgment. Other cycle timing is discussed with respect to MAS*. e.g. A+2 indicates the second cycle after MAS* assertion. i.e. MAS* assertion is A+0.
MRDY*	MBus ready transaction status bit. This bit is one of the three bits used to encode the transaction status as shown in Table 2. The encoding with MRDY* asserted alone indicates that valid data has been transferred. The three status bits (MRDY*, MRTY*, and MERR*) are normally asserted by the addressed slave. The ERROR2(Timeout) acknowledgement will be asserted by the bus monitor.

MERR*	MRDY*	MRTY*	Meaning
H	H	H	idle cycle
H	H	L	Relinquish and Retry
H	L	H	Valid Data Transfer
H	L	L	reserved
L	H	H	ERROR1 => Bus Error
L	H	L	ERROR2 => Timeout
L	L	H	ERROR3 => Uncorrectable
L	L	L	Retry

Table 2 – Transaction Status Bit Encoding

MRTY* MBus retry transaction status bit. This bit is one of the three bits used to encode the transaction status as shown in Table 2. The encoding with MRTY* asserted alone indicates that the slave wants the master to abort the current transaction immediately and start over. The master will *relinquish* bus ownership upon this type of a retry acknowledgment. Note that if any type of acknowledgement other than "Valid Data Transfer" is issued, the cycle it is issued is the last cycle, regardless of how many further acknowledgement cycles would normally occur. The three status bits (MRDY*, MRTY*, and MERR*) are normally asserted by the addressed slave.

MERR* MBus error transaction status bit. This bit is one of the three bits used to encode the transaction status as shown in Table 2. The encoding with MERR* asserted alone indicates that a bus error (or other system implementation specific error) has occurred. Note that if any type of acknowledgement other than "Valid Data Transfer" is issued, the cycle it is issued is the last cycle, regardless of how many further acknowledgement cycles would normally occur.The three status bits (MRDY*, MRTY*, and MERR*) are normally asserted by the addressed slave.

MBR* MBus Request signal. This signal is asserted by an MBus master to acquire bus ownership. There is one unique MBR* signal per master.

MBG* MBus Grant signal. This signal is asserted by the external arbiter when the particular MBus master is granted the bus. There is one unique MBG* signal per master.

MBB* MBus busy signal. This signal is asserted as an output during the entire transaction, from and including the assertion of MAS* to the assertion of the last MRDY* or first other acknowledgment which terminates the transaction (such as an error acknowledgment). If a master wishes to keep the bus and perform several transactions without releasing the bus between them, it keeps MBB* asserted until the last MRDY* of the last transaction of the group. The potential master device samples this signal in order to obtain the bus ownership as soon as the current master releases the bus. MBB* locks *arbitration* on a particular MBus. A master is allowed to assert MBB* prior to the assertion of MAS*,(to hold the bus). It is also allowed to keep MBB* asserted after the assertion of the last acknowledgment in a few special cases for performance reasons. This continued assertion of MBB* should only occur while MBG* is still parked on the current master. The MAS* of the transaction prompting the continued assertion of MBB* should be generated quickly (2 cycles is the recommended maximum delay). For more details on arbitration see section 4.2.

MIH* Memory InHibit signal. This signal is only present in level-2 MBus modules. It is asserted by the owner of a cache block at the beginning of the second cycle after it receives the address (its A+2 cycle)† to inform the Main Memory that the current Coherent Read or Coherent Read and Invalidate request should be ignored. This is because the owner, not the Memory, will be responsible for delivering the cache data block. If no device asserts MIH* during its A+2 cycle†, main memory will be responsible for delivering the data. If main memory starts delivering data and MIH* is asserted, the memory delivery shall be aborted immediately. Any data which was received from main memory in this case should be ignored. It should be noted that because of the restriction on MIH* either occurring simultaneously with or before MRDY*, there will be at most two cycles worth of data to ignore. While MIH* is sourced by a module (for one cycle) two cycles† after receiving MAS* it may be observed by a cache in the interval from its A+2 until it observes an acknowledgement. This variation in where MIH* can be observed is due to the possibility for MBus repeaters and modules that do not meet the A+2 timing.

MSH* Cache block SHared signal (wired-or, open-drain). This signal is only present in level-2 MBus modules. Whenever a Coherent Read transaction appears on the bus, the bus monitor of each processor module should immediately search its cache directory. If a valid copy is found, the MSH* signal should be asserted in the second cycle after the address is received (its A+2 cycle)†. The MSH* signal is also sampled (observed) by external caches. It is asserted as an output (for a single cycle) if there is a cache hit in the snooping directory. While MSH* is sourced by a module two cycles † after receiving MAS* it may be observed by a cache from its A+2 until it observes an acknowledgement †. This variation in where MSH* may be observed is due to the possibility for MBus repeaters and modules that do not meet the A+2 timing. Signals are sourced at the beginning and observed or sampled at the end of a cycle. Due to the open drain nature of MSH* and the associated slow rise time, while the signal is driven active low for one cycle, it can be observed active low for up to two cycles and the trailing or rising edge of the signal is considered asynchronous.

RSTIN* Module reset input signal. This signal should reset all logic on a module to its initial state, and ensure that all MBus signals are inactive or tri-state as appropriate. The minimum assertion time of RSTIN* will be system implementation dependent, although the default assertion time will be 1024 MBus clock (MCLK) periods(25.6 microseconds). RSTIN* should be treated as asynchronous.

AERR* Module asynchronous error detect out signal. This signal is asserted by the module as a level to indicate that an asynchronous error was detected by the module. It remains asserted until a software initiated action resets a bit that is maintaining the signal assertion. AERR* is open drain because several modules could assert it simultaneously. AERR* may be asynchronous.

† See Appendix B.5. for notes to designers who wish to avoid the A+2 requirement.

INTOUT*	Module Interrupt out signal. This signal is asserted by an I/O module as a level to indicate an interrupt request to the system. It remains asserted until a software initiated action resets a bit that is maintaining the signal assertion. INTOUT* may be asynchronous. This signal is **optional**.
IRL[3:0]	These pins carry the Interrupt Request level inputs to a SPARC Integer Unit. They are only used by processor modules. IRL[3:0] may be asynchronous. Each processor module receives a dedicated set of IRL[3:0].
ID[3:0]	These pins carry the Module Identifier. These signals are not needed by Level 1 processor modules, which have a default ID of 0xF, and are **optional** for other modules who may obtain this information by other means. ID[3:0] is reflected as MID[3:0] during the address phase of every transaction, and is also used to identify a unique address range for Module identification, initialization and configuration. If modules do not have ID[3:0] input pins, the system must provide a function that allows each module to obtain a unique ID[3:0] value in an internal ID[3:0] register. One way this can be accomplished is to have a logic function with a known MBus address attached to the MBus arbiter. There are unique MBR*and MBG* lines per module and so this function when addressed, would return a unique ID[3:0] value on MAD, based on which module's MBG* was asserted just prior to the beginning of the ID Read transaction when MBB* was de-asserted.
SCANDI	This signal corresponds to the signal TDI as described in IEEE P1149.1 (here after referred to as P1149) . It is an input to the module and is used for receiving scan data from the system. This is the input of the scan ring and should not be inverted or gated. This signal changes on the falling edge of SCANCLK and should be sampled on the rising edge of SCANCLK. Scan is **optional**.
SCANDO	This signal corresponds to the signal TDO as described in P1149. It is an output of the module and is used for sending the scan data to the system. This is the output of the scan ring and should not be inverted or gated. SCANDO should be driven on the falling edge of SCANCLK and will be sampled on the rising edge of SCANCLK. Scan is **optional**.
SCANCLK	This signal is used to supply the clock to the scan ring on the module. (Typically 5 MHz) Scan is **optional**.
SCANTMS1	This signal is an input to the module. It is used to control the TAP controller state machine. It corresponds to the TMS signal as described in P1149. Scan is **optional**.
SCANTMS2	This signal is an input to the module. It is used to reset the TAP controller state machine. It corresponds to the TRST* signal as described in P1149. Scan is **optional**.

2.3. Multiplexed Signal Summary

Table 3 summarizes the multiplexed MBus signals. All multiplexed signals are active high (true when '1').

Multiplexed Signals (valid during address phase)		
Signal Name	Physical Signal	Signal Description
PA[35:0]	MAD[35:0]	Physical address for current transaction
TYPE[3:0]	MAD[39:36]	Transaction type
SIZE[2:0]	MAD[42:40]	Transaction data size
C	MAD[43]	Data cacheable (advisory)
LOCK	MAD[44]	Bus lock indicator (advisory)
MBL	MAD[45]	Boot mode / local bus (advisory) (optional)
VA[19:12]	MAD[53:46]	Virtual address (optional) (level-2)
reserved	MAD[58:54]	reserved for future expansion
SUP	MAD[59]	Supervisor Access Indicator(advisory) (optional)
MID[3:0]	MAD[63:60]	Module Identifier

Table 3 - Multiplexed Signal Summary

2.4. Multiplexed Signal Descriptions

PA[35:0] Physical address of current transaction which is multiplexed on MAD[35:0].

TYPE[3:0] The transaction types are encoded in bits MAD[39:36] as shown below in Table 4. Most of the transaction types are reserved.

TYPE[3]	TYPE[2]	TYPE[1]	TYPE[0]	Data Size	Transaction Type
H	H	H	H	-	reserved
H	H	H	L	-	reserved
H	H	L	H	-	reserved
H	H	L	L	-	reserved
H	L	H	H	-	reserved
H	L	H	L	-	reserved
H	L	L	H	-	reserved
H	L	L	L	-	reserved
L	H	H	H	-	reserved
L	H	H	L	-	reserved
L	H	L	H	32B	Coherent Read & Invalidate(CRI)
L	H	L	L	any†	Coherent Write & Invalidate(CWI)
L	L	H	H	32B	Coherent Read(CR)
L	L	H	L	32B	Coherent Invalidate(CI)
L	L	L	H	any†	Read(RD)
L	L	L	L	any†	Write (WR)

Table 4 - TYPE Encodings

† as defined by the SIZE signals in Table 5

SIZE[2:0] The transaction data SIZE information is encoded in bits MAD[42:40]. The size field is encoded as $\log_2$ [number of data bytes transferred]. The encoding of the SIZE bits is shown in Table 5.

SIZE[2]	SIZE[1]	SIZE[0]	Transaction Size
L	L	L	Byte
L	L	H	Half-word (2 bytes)
L	H	L	Word (4 bytes)
L	H	H	Double-word (8 bytes)
H	L	L	16-byte Burst
H	L	H	32-byte Burst
H	H	L	64-byte Burst
H	H	H	128-byte Burst

Table 5 - SIZE Encodings

* WRAPING *

For transactions with SIZE greater than 8 bytes, more than one MRDY* will be needed. For read transactions, MBus supports a feature called "wrapping". During the address phase, MAD[2:0] are don't care, and MAD[35:3] defines the 8 bytes to be returned with the first MRDY*. This means that the address defines which data to return first, and this will vary. Data returned on subsequent MRDY*s will be for the address associated with incrementing MAD[n:3] where n = 3 for SIZE = 16-bytes, n = 4 for SIZE = 32-bytes, up to n = 6 for SIZE = 128-bytes. As the address is (conceptually) incremented, the MAD[n:3] field will wrap around without incrementing MAD[35:n+1], which is static for the duration of transaction.

Wrapping affects all modules that have to deliver data with SIZE greater than 8 bytes. An exception is Level 2 cache controllers, which may choose not to support wrapping for snoop read hits, when they supply the data. This means that Level 2 processor modules that don't support wrapping on the snoop port cannot share an MBus with Level 2 MBus processors that issue "wrapped" requests. To ensure maximum compatibility, Memory and I/O modules should all support wrapped requests to their slave ports, but should only issue aligned requests from their master ports.

The wrapping feature is not supported for writes. Write transactions with Size greater than 8 bytes should have address bits MAD[n:3] be zero, and MAD[2:0] are undefined as for reads.

For transactions with SIZE less than or equal to 8 bytes, unneeded address lines are undefined. e.g. for SIZE = 8 bytes, MAD[2:0] are undefined.

VA[19:12] Virtual Address 19 through 12 (multiplexed on MAD[53:46]). This field only applies to MBus level-2 Coherent transactions. It is used to carry the virtual address bits 19 through 12 associated with the physical address of a Coherent Read transaction. These bits are used by virtually indexed caches that desire to index into the dual directories via the virtual "superset" bits to avoid synonym problems. This assumes a minimum page size of 4K in the system and maximum cache size of 1MB. Modules that choose not to provide this function nor to support non-coherent transactions (such as a level-1 device) should drive these lines *high*.

MID[3:0] Module identifier signals (multiplexed on MAD[63:60]). This field is sourced by all MBus modules and reflects the value input into the module on the ID[3:0] input signals. For level-1 processor modules this field is driven *high(0xF)*. This field is observed by slave ports that wish to issue a Relinquish and Retry acknowledgment(see section 3.4), so that they can identify the master with which to re-connect in a multi-master system configuration.

C Cacheable indicator (multiplexed on MAD[43]). When this signal is asserted, it indicates the state of the cacheable bit for the address of the transaction in the module MMU (if there is one). If a module has insufficient information to determine the level of this bit for a transaction, it should leave the bit de-asserted. This is an advisory bit, not used by MBus transactions, but possibly of use to the slave device.

An example use of C would be to inform a second level cache of the cacheability state of the address of a transaction with SIZE less than 32 bytes.

LOCK Lock indicator signal (multiplexed on MAD[44]). If the MBus master intends to lock access to a device residing on MBus (main memory is one MBus device) or some other bus connected to MBus, and perform N indivisible MBus transactions to the device, this bit needs to be asserted during the address cycles of all N MBus transactions. The locking master must keep MBB* asserted during each locked cycle and not de-assert it until the end of the final locked transaction (however MBB* may be suspended for a time by an R&R acknowledgment). The de-assertion of MBB* signals the MBus arbiter to release the MBus. It is the final de-assertion of MBB* after possible intervening R&R acknowledgments which tells the device to release its lock. LOCK is an advisory bit, not used by MBus transactions directly, but possibly of use to the slave device or bus interface.

An example use of LOCK would be to "lock" an MBus master to a particular slave. If an MBus processor performed an atomic operation to a resource arbitrated externally to MBus, such as a dual-ported memory device or another bus, then the external arbiter could prevent any other (non MBus) device from accessing that resource by locking arbitration. The referenced slave device in a LOCKed transaction could be, in essence, dedicated to the requesting master. The MBus slave port interface interprets an assertion of the MBus LOCK bit as saying "become locked" and a final de-assertion of MBB* at the end of the locked sequence as saying "become unlocked", and reports this information to the arbiter for the "locked" device (or bus). If the slave port supports R&R acknowledgments, it must know not to clear the locked state when MBB* is removed due to an R&R.

MBL MBus boot mode / local bus indicator (multiplexed on MAD[45]). This signal is asserted by processor modules during the address phase of boot mode transactions, or during local bus transactions (SPARC processor accesses with ASI = 0x1) . It is system implementation dependent whether or not local bus transactions are employed in a system. This is an advisory bit, not used by MBus transactions, but possibly of use to the slave device. This bit is optional. If unused by an implementation it should remain de-asserted.

SUP	Supervisor access indicator (multiplexed on MAD[59]). This signal is asserted by processor modules and indicates that the MBus transaction is a processor Supervisor access. This is an advisory bit, not used by MBus transactions, but possibly of use to the slave device. This bit is optional. If unused by an implementation it should remain asserted. An example use of SUP would be to enable more state to be captured on processor asynchronous write errors.
reserved	This 5-bit field (multiplexed on MAD[58:54]), is reserved for future MBus expansion. The lines should be driven *high* if not used.

Chapter 3

3. MBus Transactions

3.1. Semantics

(1) Basic cycle is read/write(size), where size is from 1 to 128 bytes.

(2) Bus cycles greater than 8 bytes are performed as bursts.

(3) Data rate is controlled by the slave. Master must be able to accept a burst read, or source a burst write, of requested size, at maximum transfer rate.

(4) Master starts bus cycle with MAS* (address transfer phase). Master also asserts MBB* at (or before) this time.

(5) In the case of burst mode, multiple acknowledgments are used and the bus cycle ends with the acknowledgment of the last data transfer. The master de-asserts MBB* at end of the last cycle (except for locked bus cycles).

(6) Locked bus cycles are an indivisible sequence of basic bus cycles. Locked cycles are also terminated immediately by error acknowledge and individual transactions in the sequence may be suspended by retry or relinquish and retry acknowledgments.

Note: In the transaction semantic diagrams that follow, optional wait states are indicated as x, y, or z cycles. x, y, and z can be zero.

3.2. Level 1 Transaction Types

3.2.1. Read

A Read operation can be performed on any size of data transfer which is specified by the SIZE bits in Table 5. Read transactions support wrapping as defined in section 2.4. Read transactions involving less than 8 bytes will have undefined driven data on the unused bytes. The minimum MBus Read transaction will take 2 cycles . This minimum time is for the cases when no data is returned on MAD, such as during R&R or error acknowledgements. If data is being returned, an extra cycle is required to avoid bus contention. The arbitration protocol creates a dead cycle between transactions which ensures there will be no bus contention between back-to-back reads from different masters. If a module locks the bus and performs back-to-back reads, it is its responsibility to ensure a dead cycle to avoid contention Note that the protocol means that a master *must* be able to receive data at the maximum rate of the MBus for the duration of the transaction, i.e. 8 bytes on every consecutive clock. Figure 5 depicts the semantics of a Read operation. Refer to Chapter 8 for details pertaining to cycle waveforms.

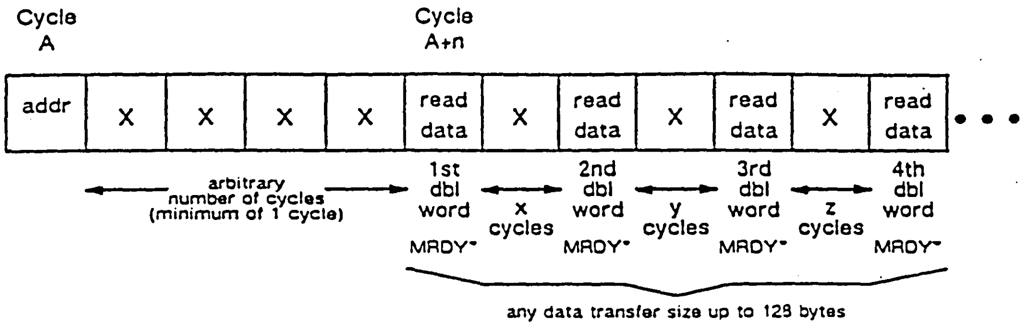


Figure 5 – Read Semantics

3.2.2. Write

A Write operation can be performed on any size of data transfer which is specified by the SIZE bits in Table 5. Write transactions involving less than 8 bytes will have undefined data on the unused bytes. The writing master will immediately drive the data in the period after the address phase of the transaction, and immediately after receipt of each MRDY* in transactions with SIZE greater than 8 bytes. Note that the protocol means that a master *must* be able to source data at the maximum rate of the MBus for the duration of the transaction, i.e. 8 bytes on every consecutive clock. The minimum MBus Write operation will take 2 cycles(minimum is actually 3 cycles if different masters are performing back-to-back writes). Figure 6 shows the basic semantics of a Write operation.

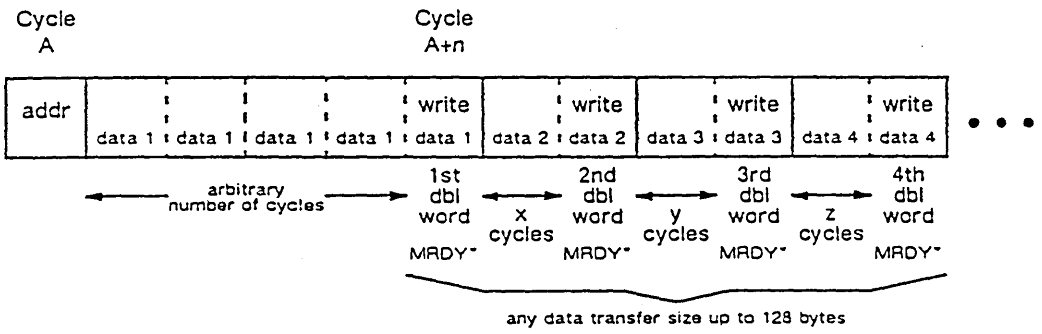


Figure 6 – Write Semantics

Due to the nature of the cache-consistency protocol, the Write transaction works equally well in level 1 and Level 2 MBus implementations. Writes can be used for non-cacheable accesses as well as for write-backs of dirty (sub-)blocks. Write transactions do not need to be snooped and the MIH* and MSH* signals must not be asserted during the operation.

R&R acknowledgements issued to Block Write transactions to cacheable locations introduce a detailed design problem, in that the write back buffer in this case may be the only source of the most up to date data. This introduces the prospect of having to snoop the write back buffer. To simplify the design of processor modules, the MBus specification eliminates the need for processor modules to snoop write back buffers and places the burden of handling this case of R&R acknowledgement on the module that issues the R&R. Modules that issue R&R acknowledgements to cacheable block write transactions must capture the address(es) of the cache line(s) until they complete the transaction to which they issued R&R. Should other modules attempt to read the line(s) during this interval the R&R issuing module must detect this and issue R&R to the intervening Coherent Read(CR) or Coherent Read and Invalidate(CRI) transaction(s). In general it should be possible for most modules to avoid the need to issue R&R to cacheable block write operations and hence avoid this complexity. The only likely exception is a coherent bus adaptor.

3.3. Additional Transaction Types for Level 2

3.3.1. Coherent Read

A Coherent Read operation is a block read transaction that maintains cache consistency. The participants in the transaction are the requesting cache, the other caches which snoop, and memory (or a second level cache). There are three possible read scenarios which the caches that snoop can experience:

a) For a snooping cache which does not have a copy of the requested block, it simply ignores this transaction.

b) For a snooping cache which does have a copy of the requested block but does not *own* it, it simply asserts MSH* for one cycle during its cycle A+2†. It will mark its copy as shared (if not already marked as such).

c) For a snooping cache which owns the requested block, it will assert both the MSH* and MIH* signals for one cycle during bus cycle A+2† and start shipping the requested data no sooner than its cycle A+6, (4 cycles after it issued MIH*). If its own copy of the block was labeled exclusive, it will be changed to shared, else no status change will take place for its own copy.

Upon receiving the data block, the requesting master shall label the block *exclusive* if no one asserts MSH* during its cycle A+2† and *shared* if the signal MSH* is asserted during its cycle A+2†. Figure 7 depicts the semantics of a Coherent Read operation where memory latency is long.

Case (c) above needs further elaboration. This is the only case where MIH* is asserted. This signal affects three parties. It is sourced by the snooping (intervening) cache and observed by both memory (or a second level cache) and the requesting cache. It tells the requesting cache that it may have received stale data from memory, and to ignore that data and data it may receive on the next clock and wait until the fourth or later clock for the correct data. It tells memory to stop sending data *immediately*, which means memory may send one more MRDY* before it can stop. The delay of 4 clocks at the requesting cache and the snooping (intervening) cache serve two related purposes. The first is to allow time for MRDY* and MAD from the memory to be turned off before the snooping cache asserts MRDY* and MAD, and so avoid bus contention. The second is to allow for implementations that buffer the MBus.

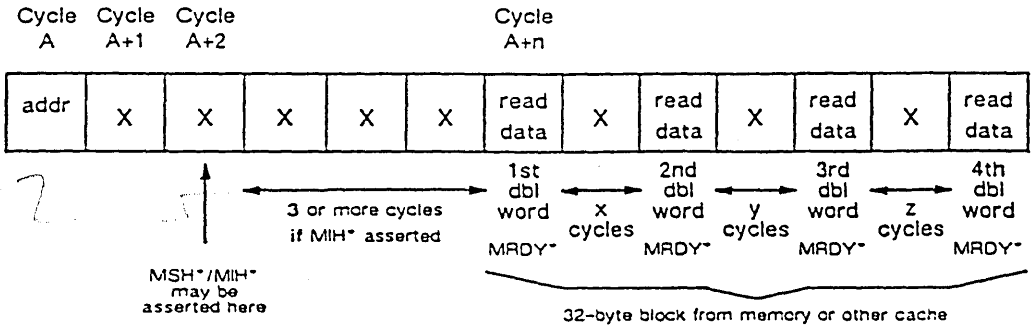


Figure 7 - Coherent Read Semantics

† See Appendix B.5. for notes to designers who wish to avoid the A+2 requirement.

The earliest that memory (or a second level cache) is allowed to issue MRDY*(or any acknowledgment) is its cycle A+2†. This ensures that acknowledgments never occur before MIH*. Figure 7 shows the semantics of a Coherent Read transaction.

3.3.2. Coherent Invalidate

An Invalidate operation can only be performed on a block basis. All Invalidate operations will be snooped. If an Invalidate operation hits in a cache, then that copy will be invalidated immediately, regardless of its state. **One module (normally a Memory controller) is responsible for the acknowledgment of the Coherent Invalidate transaction.** This is accomplished on its cycle A+2†, or later. All acknowledgment types are possible. Memory will only ever issue normal acknowledgments(MRDY*) to Coherent Invalidates, but coherent bus adaptors may issue other acknowledgments, particularly R&R. It should also be noted that a Coherent Invalidate transaction will have SIZE = 32B during the address phase, but it will only be expecting one MRDY* for the acknowledgment. Also the address may not be 32-byte aligned. Memory (or coherent bus adaptor) designers should take note of this. If in a particular system, caches cannot guarantee to complete their invalidation before their A+2† cycle, the memory controller for that system should delay the acknowledgment as appropriate. This implies that memory controllers should have a feature that allows the time to acknowledge invalidates to be varied to some extent, either hard wired or through a programmable register. A recommended range for the programmable delay is A+2 to A+10. This programmable delay is the MBus flow control technique to guarantee that invalidates can be completed at any rate they are issued.

This Coherent Invalidate MBus transaction is issued when a write is being performed into a cache line that is Shared. Before the write can actually be performed, all the other systems caches must have their local copies invalidated (write-invalidate cache-consistency protocol). Snooping caches will not assert MSH* during A+2†. The MAD lines will contain undefined data during the data phase cycles. If a Coherent Invalidate transaction should receive an R&R acknowledgement there is a possibility that the line which is about to be written becomes invalidated by an intervening invalidation transaction on the bus. This means that when the cache regains the bus it should issue a Coherent Read and Invalidate transaction, not a Coherent Invalidate transaction, to once again allocate the (sub-)block. Figure 8 shows the basic semantics of an Invalidate operation.

For any particular system, selecting which module will be responsible for acknowledging Coherent invalidates introduces some issues for memory controller designers. In most systems a single memory controller will be responsible. In systems with a coherent bus adaptor, the adaptor will be responsible. If it is desired to use a memory controller in a system that also has a coherent bus adaptor, it is then required to be able to tell the memory controller not to respond to invalidates. This should be accomplished during system initialization prior to enabling any caches, preferably by writing a bit in a register in a memory controller.

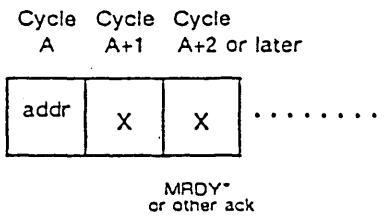


Figure 8 - Coherent Invalidate Semantics

† See Appendix B.5. for notes to designers who wish to avoid the A+2 requirement.

3.3.3. Coherent Read and Invalidate

Since the MBus supports a write-invalidate type of cache-consistency protocol, a special Coherent Read and Invalidate transaction, which combines a Coherent Read transaction with a Coherent Invalidate transaction, was included to reduce the number of MBus transactions. Caches that are performing Coherent Reads with the knowledge that they intend to immediately modify the data can issue this transaction.

Each Coherent Read and Invalidate transaction will be snooped by all system caches. If the address hits and the cache does not own the block, then that cache will immediately invalidate its copy of this block, no matter what state the data was in. If the address hits and the cache owns the block, then it will assert MIH* and supply the data. When the data has been successfully supplied, the cache will then invalidate its copy of this block. Figure 9 shows the basic semantics of a Coherent Read and Invalidate operation. Note that it is identical to the Coherent Read operation, except that the system caches will invalidate the block. All of the detailed comments concerning MIH* for the Coherent Read transaction apply equally to the Coherent Read and Invalidate transaction. MSH* is not driven during the Coherent Read and Invalidate transaction.

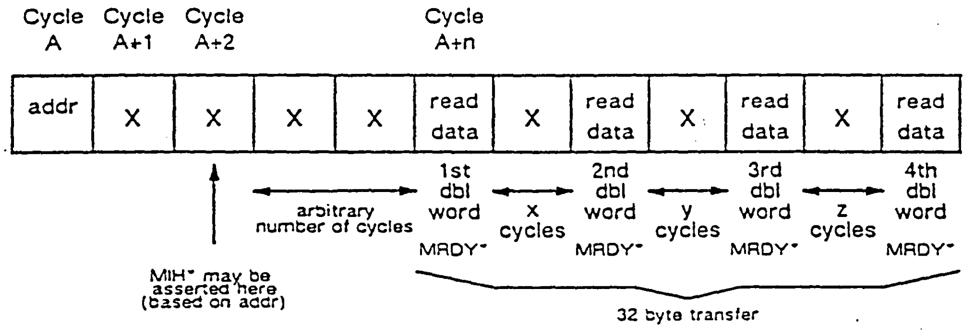


Figure 9 – Coherent Read and Invalidate Semantics.

3.3.4. Coherent Write and Invalidate

A Coherent Write and Invalidate transaction combines a Write transaction with a Coherent Invalidate transaction. The Coherent Write and Invalidate transaction is intended to reduce the number of MBus transactions. This transaction can be used by Level-2 modules that wish to support a write through cache, a degree of functionality beyond the requirements for level-2 MBus. (Supporting write through caches is useful for implementing simple 2nd level caches that support inclusion.) CWI is also of use to block copy and block fill mechanisms.

Each Coherent Write and Invalidate transaction will be snooped by all system caches. If the address hits, then caches will invalidate their copies of this block no matter what state the data was in. Figure 10 shows the basic semantics of a Coherent Write and Invalidate operation. Note that it is identical to the Write operation, except that the system caches will invalidate the block. All SIZE values are allowed and a single 32-byte block is invalidated regardless of the value of SIZE. Due to the nature of the cache coherency protocol neither MIH* nor MSH* are asserted.

All SIZE values are allowed in order to better accommodate write through caches. Systems with only either write through or write back caches work naturally, but a system with both a write through and a write back cache are very unlikely to work in a way that preserves cache consistency. This mixed system is not anticipated or recommended as a real MBus configuration.

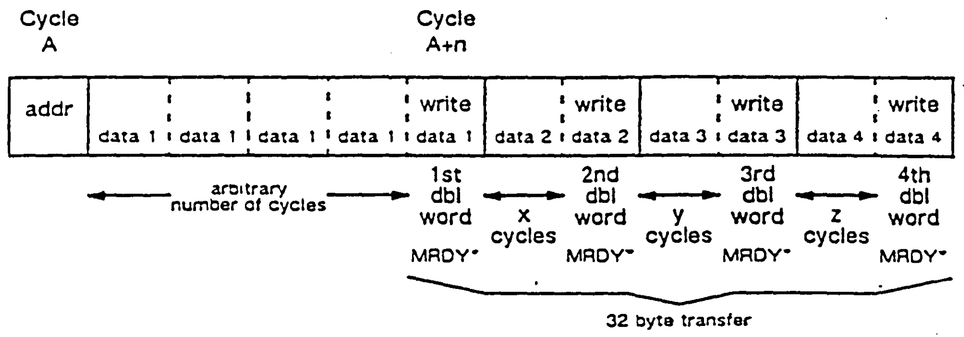


Figure 10 – Coherent Write and Invalidate Semantics

3.4. Acknowledgment Cycles

It is a requirement that any transaction once issued *must* correctly accept *any* acknowledgment type. This applies to all Level 1 and Level 2 transactions. The earliest that an acknowledgment can be issued is A+1 for Read and Write and A+2 for all Coherent Transactions. Processor caches that are supplying data as part of a Coherent Read transaction may only issue either normal or error acknowledgements. They may not issue R&R or Retry acknowledgements.

3.4.1. Idle Cycles

When there is no bus activity or when it is necessary to insert wait states in between the address cycle and the data cycle or between consecutive data cycles, an addressed slave can simply refrain from asserting any transaction status bits (MERR*, MRDY*, and MRTY*). The number of wait cycles which can be inserted is arbitrary, as long as it does not exceed the system timeout interval (see section 3.4.5. for timeout details).

3.4.2. Relinquish and Retry(R&R)

When a slave device cannot accept or supply data immediately, it can perform a *relinquish and retry* acknowledgment cycle by asserting MRTY* for only one bus cycle. This will indicate to the requesting master that it should release the bus immediately so that the bus can be re-arbitrated and possibly used by another MBus master. This involves at least one dead cycle until the suspended transaction can be performed in the case when the bus is still granted to the re-trying master. When the bus is no longer granted to the master in question, then the suspended transaction must wait until bus ownership is once again attained. When a transaction that receives an R&R acknowledgment regains bus mastership it must issue the same transaction over from the beginning. An exception to this is when a Coherent Invalidate turns into a Coherent Read and Invalidate (see section 3.3.2.). For Level 1 modules, for all transactions with SIZE greater than 8 bytes, a relinquish and retry acknowledgment can be asserted on any data transfer. For Level 2 modules, for all transactions with SIZE greater than 8 bytes, (including the Level-1 READ and WRITE transactions) R&R can only be issued on the first acknowledgement. It is the responsibility of the slave port to time the duration of the transaction that is causing it to issue R&R, and return an ERROR2 acknowledgment to the correct master when its device specific timeout interval (200 micro-seconds is recommended) has passed and the master has re-connected to the slave.

There are two different cases that cause slaves to issue R&R acknowledgments. The first is slow devices. If a device is slow to respond, the slave interface should wait a short interval (around one microsecond is recommended), and then issue an R&R acknowledgment. It should also capture the ID of the master from the MAD lines during the address phase (MID[3:0] field) and enter a "port busy" state while waiting for the device attached to the slave to respond. The master will eventually re-connect and the R&R process will be repeated until either the device responds or the slave timeout interval (which should be much greater than the short interval, 200 micro-seconds is recommended) is exceeded. The slave will then issue the normal or error acknowledgment respectively and exit the "port busy" state.

In systems with multiple masters, the slave that issues R&R must capture the ID of the master who's transaction is being postponed in order to know which master should receive the normal or error acknowledgment when the slave can complete the transaction. If a master with an ID other than that captured by the slave port should access the slave port while it is in the "port busy" state, it should simply be given an R&R acknowledgment.

The second cause of R&R acknowledgments is the resolution of **deadlock** situations where there is a master and a slave port sharing an MBus interface and simultaneous transactions on both ports requires one transaction to back off. R&R requires the current owner of MBus to relinquish ownership in order to resolve the deadlock. R&R's used to resolve deadlocks are inherently stateless and do not require a "port busy" state.

A detail of significance is that R&R can be issued to a transaction that is part of a locked sequence of transactions. By definition, all transactions in a locked sequence are addressed to the same device, e.g. main memory(or second level cache) or an I/O adapter. There is only one "port busy" state per device, so there is only one source of R&R for a locked sequence.

Normally, main memory will not issue R&R. Multiple R&R sources from main memory would restrict locked sequences to addresses within one memory bank, Also, some aspects of coherent cache design are simplified by locking some MBus sequences such as fills and their associated write-back(if any). These sequences rely on either a memory system that does not issue R&R or an appropriately designed second level cache or coherent bus adaptor that does. See section 3.2.2. for more details.

It should be noted that processor caches which assert MIH* and then supply data cannot issue R&R acknowledgements.

3.4.3. Valid Data Transfer

A valid or ready data transfer is indicated by a responding slave with the assertion of the MRDY* transaction status bit for only one cycle. This signal needs to be asserted on reads to indicate to the requesting master that valid data has just arrived. On writes, MRDY* indicates to the writing master that the data has been accepted and that the writing master shall stop driving the accepted data. The next doubleword, if a write burst were being performed, will be driven onto the bus in the cycle immediately following the assertion of MRDY*.

3.4.4. ERROR1 => Bus Error

When the responding device asserts only the MERR* transaction status bit, the requesting master will interpret this as an external bus error just having taken place. The meaning of "Bus Error" is implementation dependant.

3.4.5. ERROR2 => Timeout

This acknowledgment is expected to be generated by some sort of watchdog timer logic in the system that primarily detects transactions that are not acknowledged. This is accomplished by timing the shorter of, either the assertion of MBB*, or the time since the last MAS* assertion, as follows.

A timeout counter should start on the assertion of MBB* and count until the de-assertion of MBB*, or until the timer has counted the timeout interval. If the counter counts to the timeout limit, a timeout error acknowledgement should be generated by the timeout monitor circuitry. When counting ceases, the counter should be re–initialized to its initial condition. If MAS* is asserted during the time that counting is enabled (MBB* assertion) the counter should be re–initialized, but continue counting. The number of cycles for the timeout interval is system implementation dependent. An interval of 200 micro-seconds is recommended. This error code can also be used to indicate a system implementation dependent error. Timeout is the suggested interpretation of an ERROR2 acknowledgment.

3.4.6. ERROR3 => Uncorrectable

This acknowledgment is mainly used by the addressed memory controller to inform the requester that in the process of accessing the data some sort of uncorrectable error has been encountered (like parity, uncorrectable ECC, etc). This error code can also be used to indicate a system implementation dependent error. Uncorrectable error is the suggested interpretation of an ERROR3 acknowledgment.

3.4.7. Retry

This acknowledgment differs from the *Relinquish and Retry* acknowledgment in that the master will not, in this case, release bus ownership if it is no longer granted the bus, but rather the transaction will immediately begin again with an address phase (MAS*, etc) as soon as the retried master is ready to do so. Retry errors can occur on any acknowledgment of a transaction. This type of acknowledgment can be useful when a correctable ECC error has occurred in the main memory subsystem.

Should a Retry acknowledgement occur on other than the first acknowledgement cycle, the issue of "Data Correctness" arises. Modules that use delivered data prior to completion of the transaction may not be able to tolerate delivery of bad data. They may choose to treat Retry acknowledgements as equivalent to ERROR3 acknowledgements. This assumes that the Retry is "stateless" and the slave device issuing it will not hang or otherwise malfunction if the transaction is not retried. This is a detail at the discretion of system implementors.

3.4.8. Reserved

This acknowledgement is reserved for future use. Should a master receive a Reserved acknowledgement its behavior is undefined.

→ N.B. MBB* in general is asserted between MAS* + MACK* — however it may be asserted outside [illegible] this region but never for more than a few cycles.
(200µsecs is 3,000 cycles at 40MHz)

Chapter 4

4. Arbitration

4.1. Arbitration Principles

- The Arbiter is a separate unit from both the slave(s) and master(s).
- Arbitration is overlapped with current bus cycle.
- Back-to-back transactions by different masters are not allowed. There must be at least one dead cycle in between each transaction during which MBB* is de-asserted.
- Arbitration algorithm is implementation dependent. (fair bandwidth allocation should be maintained)
- Bus parking will be employed. (current master keeps the bus until it is taken away by another request)
- Locked cycles are accommodated to handle indivisible operations.

4.2. Arbitration Protocol

The MBus arbitration scheme assumes a central arbiter. The exact algorithm used by the arbiter (like round robin, etc) is implementation dependent.

4.2.1. Bus requestors

A requesting module requests the MBus by asserting its dedicated MBR* signal and then waits for assertion of its dedicated grant signal MBG* by the arbiter. Upon receiving its dedicated grant signal (MBG*), the requesting master can start using the bus by asserting MAS* and MBB* as soon as MBB* is released (de-asserted) by the previous master. It is not necessary for the requesting master to assert MAS* immediately, but it is necessary to assert MBB* to acquire and hold the bus. A requesting master is not guaranteed to gain bus ownership if it does not immediately assert MBB* upon detecting the condition of its MBG* asserted and MBB* de-asserted because if some other master is requesting the bus, the arbiter will then assert the grant to the new requestor.

The requestor, upon receiving its dedicated grant signal (MBG*) should immediately remove its dedicated request (MBR*) on the next clock edge. It is allowed to assert MBR* in anticipation of needing the bus, and then de-assert it prior to receiving MBG*. However, this may waste bus cycles and should be avoided.

4.2.2. Bus arbiter

The arbiter uses only the MBRn*, MBGn*, signals from each master and the common bussed MBB* signal. The arbiter receives the requests (MBR*n) and resolves which grant (MBG*n) to assert. A grant remains asserted until at least one cycle after the current master has de-asserted MBB* when it becomes "parked", and may be removed at any time after this in response to assertion of further requests. If no other requests (MBR*n) are asserted, the grant (MBG*n) remains asserted (parked). Only one grant (MBG*) is asserted at any time. A dead cycle between successive transactions of different masters will always occur with the MBus arbitration scheme. Timing diagrams for example transactions are shown in Chapter 8.

Figure 11 below shows a block diagram and a state diagram of an arbiter that resolves two requests and issues the two associated grants. In the state diagram, the state names are indicated in bold text within the state circles. The signal outputs associated with each state are indicated in normal text within the state circles. In the state transition equations, ! indicates NOT, & indicates AND and # indicates OR. The signals are represented in their positive logic form. The reset signal MRST is omitted from the state transition equations for clarity. For larger arbiters, the state transition diagram becomes more complex. There will always be two states per MBG signal, one for the case where a grant has been issued but not accepted, and the other for the case where the grant has been (potentially) accepted and the arbiter remains parked on that grant.

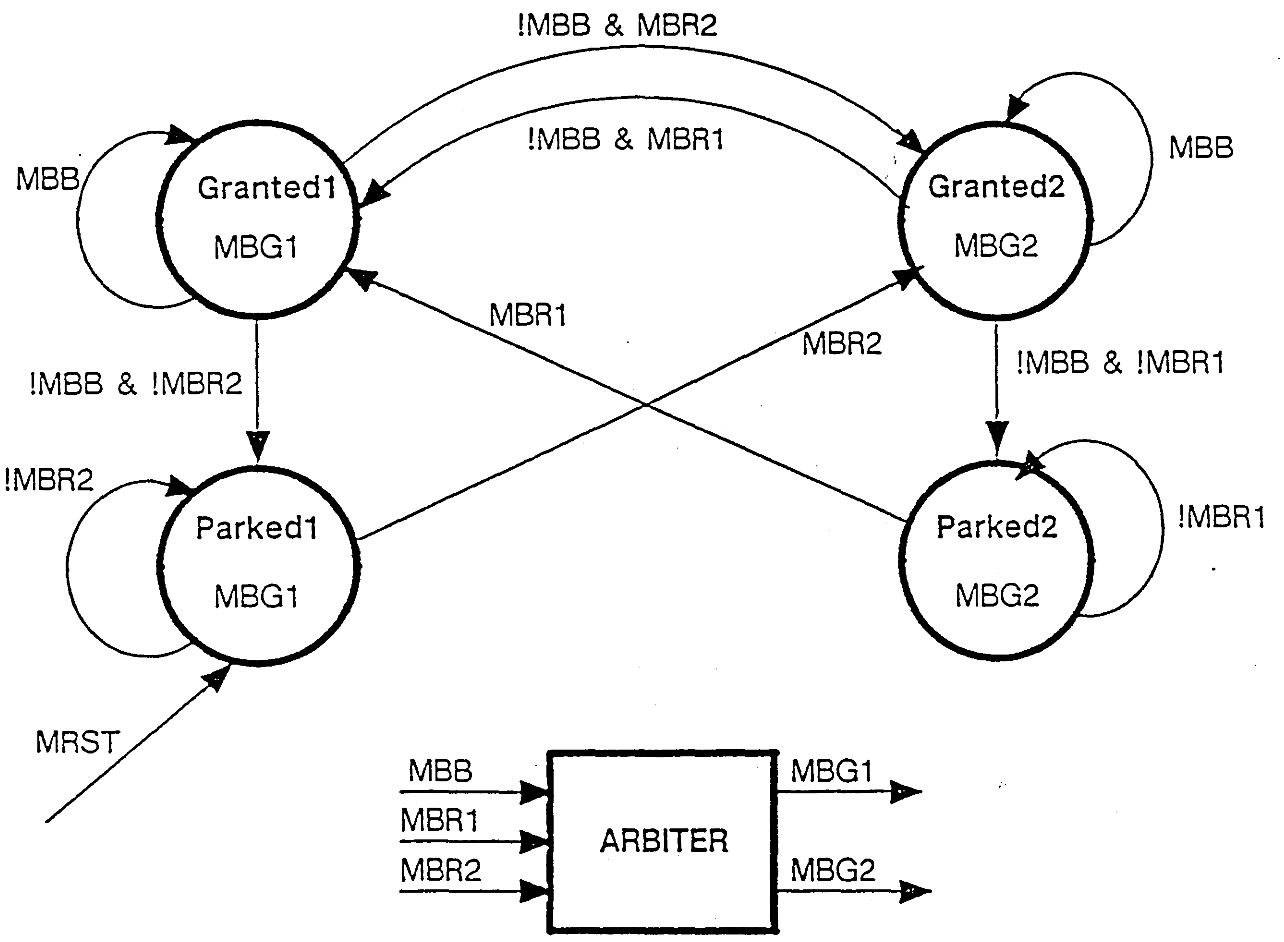


Figure 11 – Example of a two grant MBus arbiter.

Chapter 5

5. MBus Configuration Address Map

A small portion of the MBus memory space has been pre-allocated to each potential MBus module, to allow for a uniform method of system configuration. There is an individual space per MBus ID, 16 spaces in total. The ID of a module is determined by the value on the ID[3:0] pins. Level 1 processor modules do not have slave interfaces and so will not respond at the configuration address map locations. Table 6 shows the configuration address spaces of MBus.

Configuration Spaces	Mbus Identifier
0xFF0000000 to 0xFF0FFFFFF	Range for ID=0x0**
0xFF1000000 to 0xFF1FFFFFF	Range for ID=0x1
0xFF2000000 to 0xFF2FFFFFF	Range for ID=0x2
. . . .	. . . .
0xFFF000000 to 0xFFFFFFFFF	Range for ID=0xF

**reserved for "boot prom"

Table 6 – MBus Configuration Address Map

One 32-bit location in each space (0xFFnFFFFFC where n=ID) is fixed and should contain the Implementation Number and Version Number of the MBus module in an **MBus Port Register (MPR)** which is shown in Figure 12. A processor accessing the 16 possible MPRs can determine what ID slots are present(through timeout) and what devices they contain from the contents of the MPR. Other than this one address, the use of the address range is implementation specific, and specified by module vendors. Examples of items located in these configuration address ranges are registers that determine the address ranges which memory and I/O modules respond to. Similarly, implementation specific registers and memories necessary to configure and test modules would reside at locations within the device specific configuration address space.

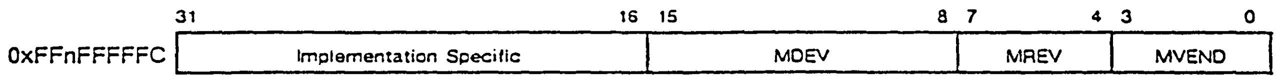


Figure 12 – MBus Port Register Format

MDEV MBus Device number. This field contains a unique number which indicates the vendor specific MBus device that is present at the referenced MBus port. Refer to Vendors for their MDEV assignments.

MREV Device Revision number. This field contains a number that can be interpreted as a revision number or some other variable of a device. Refer to Vendors for their MREV assignments, if any.

MVEND MBus vendor number. This field contains a unique number which indicates the vendor of the device present at the referenced MBus port. Refer to Appendix A for current MVEND assignments.

On coming out of reset, processor modules will fetch instructions from MBus address 0xFF000000 and subsequent memory locations. This means that the configuration address space for ID=0x0 is special and always needs to be present. MBus ID=0x0, then, can be considered as the "boot PROM" MBus module.

Chapter 6

6. MBus Electrical Characteristics

6.1. MBus Electrical Principles

Bus Protocol The MBus is a fully synchronous bus. The frequency of operation is 40Mhz.

Sampling All signals are changed and sampled on rising clock edges.

Driver Overlap No bussed signal is driven in the same cycle by more than one source. All bussed control signals (except MSH* and AERR* which are open drain) follow a sequence from tri-state to active low to driven high to tri-state, normally on successive rising clock edges. Since MBB* can be driven by two sources with only one dead cycle in between, the drive high followed by tri-state must occur within a single cycle. The multiplexed address data bus signals (MAD) follow a sequence from tri-state, to active high or low, to tri-state. MAD lines should always be high or low before tri-stating to ensure MAD lines are always at a known logic level.

Noise To avoid noise problems due to "floating" signals or very slow rise time signals, high impedance holding amplifiers are required on the MAD lines. Bussed control lines require high impedance pull up resistors. These amplifiers and/or resistors are a centralized function, provided by the system.

Driver Turn On There are many cases in the protocol where it is necessary to turn on and drive a signal in the same clock cycle. It is a general assumption that this is also done even for cases where it is not absolutely necessary. Care needs to be taken where a driver is turned on before the signal is driven as the protocol does not define the cases where this may be possible. This is particularly true for CR and CRI which are three party transactions.

6.2. Signal Grouping

For timing purposes, signals are divided into four groups, MAD[63:0], bussed control, point to point control and misc. Bussed control includes MAS*, MRDY*, MERR*, MRTY*, MBB*,MSH*, and MIH*. Point to point control includes MBR*and MBG*. Misc includes RSTIN*, IRL[3:0], INT-OUT*, and AERR*

SIGNAL CLASS	SIGNAL
BUSSED CONTROL:	MAS*, MRDY*, MRTY*, MERR*, MBB*, MSH*, MIH*
POINT TO POINT CONTROL:	MBR*, MBG*
MAD:	MAD[63:0]
MISC:	RSTIN*, AERR*, IRL[3:0], INT-OUT*

Figure 13 – MBus Signal Grouping for timing purposes

6.3. Definitions

17.5 **tcod:** Clock to output delay time for the bussed control lines. ~~17.5~~ 16

2.5 **tcoh:** Clock to output hold time for the bussed control lines. ~~2.5~~ 4

18.5 **tmod:** Clock to output delay time for the MAD lines. ~~18.5~~ 18

2.5 **tmoh:** Clock to output hold time for the MAD lines. ~~2.5~~ 4

14.5 **tpod:** Clock to output delay time for the point to point control lines. 16.5

2.5 **tpoh:** Clock to output hold time for the point to point control lines. 2.5

6 **tcis:** Input set up time for bussed control lines. ~~6.0~~ 3

1 **tcih:** Input hold time for bussed control lines. ~~1.0~~ 2

5 **tmis:** Input set up time for MAD lines. 20 3

1 **tmih:** Input hold time for MAD lines. 22 2

9 **tpis:** Input set up time for point to point control lines. 9.0

1 **tpih:** Input hold time for bussed control lines. 1.0

Tcp: Clock period. 25

Tpwh: Clock High period. ~~8~~ 14 11

Tpwl: Clock Low period. 14, 11

Tskur: Rising edge clock skew. 1.5 1.0

Tskuf: Falling edge clock skew. 1.5 1.0

6.4. Timing Reference Diagram

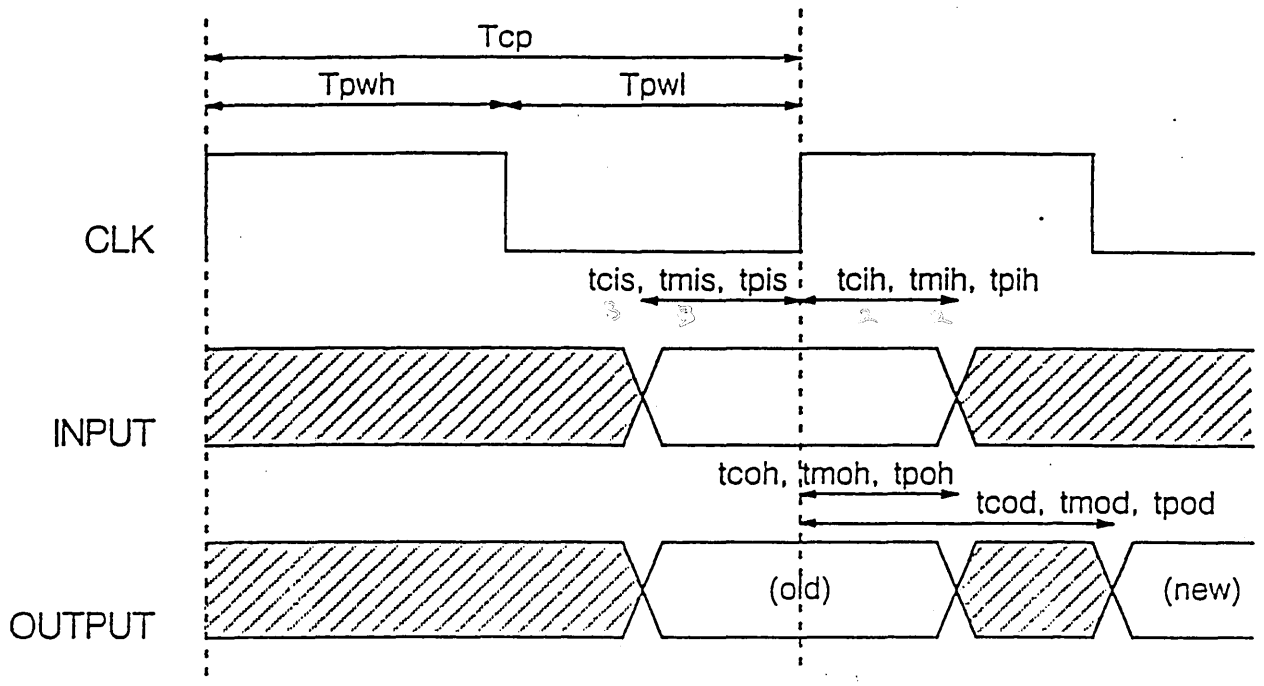
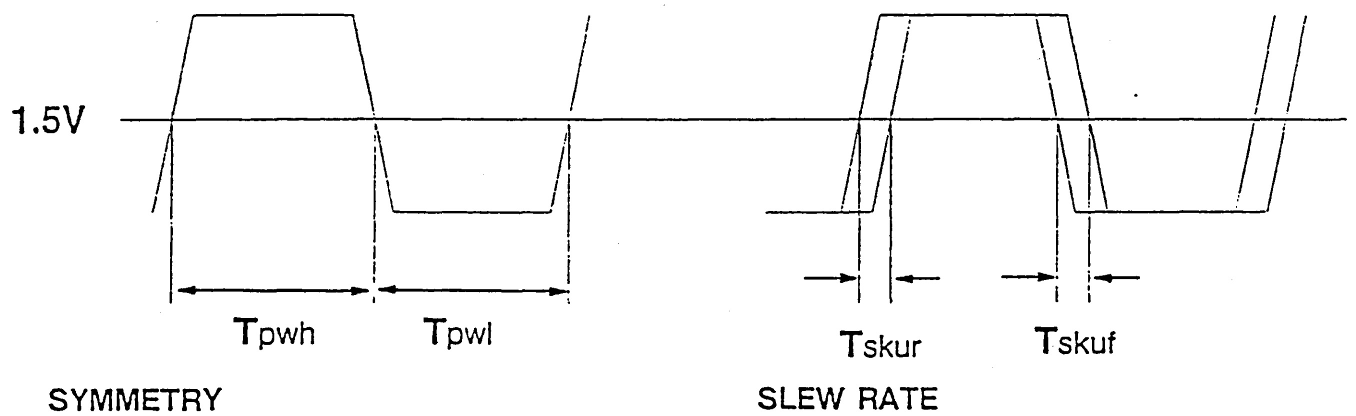


Figure 14 – Reference Timing Diagram

6.5. Clocks

The clocks provided to the module by the system will have the characteristics detailed in Figure 15.



SYMMETRY

Tpwh = 11.0 ns min 14.0 ns max

SLEW RATE
(.8V to 2V) = 0.63 V/ns min
(2V to .8V) = 0.75 V/ns min

Figure 15 – CLOCK specification

6.6. Level-1 and Level-2 Clock Characteristics (Ta = 0-70C)

Paramter	min	max	Unit
Tcp	25	25	ns
Tpwh	11	14	ns
Tpwl	11	14	ns
Tskur	0	1.5	ns
Tskuf	0	1.5	ns

Table 7 - Level-1 and Level-2 Clock Characteristics

6.7. Level-1 A.C. Characteristics(MAD and Control) (Ta = 0-70C)

Paramter	min	max	Unit
tmis		3	ns
tmih		2	ns
tmod	-	18	ns
tmoh	4	-	ns
tcis		3	ns
tcih		2	ns
tcod	-	16	ns
tcoh	4	-	ns

* All times are for a Capacitive load of 100pF

Table 8 - Level-1 AC Characteristics

6.8. Level-2 A.C. Characteristics(MAD and Control) (Ta = 0-70C)

Parameter	SPEC	Load
tcod max	17.5	System-max
tcoh min	2.5	System-min
tmod max	18.5	System-max
tmoh min	2.5	System-min
tpod max	14.5	System-max
tpoh min	2.5	System-min
tcis max	6.0	
tcih max	1.0	
tmis max	5.0	
tmih max	1.0	
tpis max	9.0	
tpih max	1.0	

NOTE: All times in nano-seconds

Table 9 - MBus Level-2 A.C. Characteristics (MAD and CNTRL)

NOTE: All Level-2 timing values are relative to the pads on the DIE.

It should be noted that the timing specification listed here is for a reference load represented by the SPICE models System-max and System-min(available from SPARC International). Conformance with this Level-2 AC specification is achieved by chip vendors guaranteeing that these timing numbers have been achieved when performing a SPICE analysis using accurate models of chip internal circuitry driving the reference loads System-max and System-min. This gives a more complete degree of compatibility between modules from different vendors than the simple RC load usually used in IC specifications.

The reference loads, System-max and System-min, include elements of a typical MBus system, including IC packages, MBus connectors, and printed wiring board traces.

The spec listed is not to be used as the timing criteria a chip tester would be programmed with. It is expected that data sheets from different vendors will be quoted for different test environments and so will differ from the numbers in this table. These numbers also represent the minimum requirements. Chip vendors can exceed these specifications.

It is recommended that MBus systems designers perform SPICE or similar analysis on their particular printed wiring board layout. In order to facilitate this, it is recommended that MBus chip vendors provide accurate SPICE models of their I/O circuitry to their customers.

The timing listed in Table 9 is for a 40 MHz MBus system. It is recommended that systems be designed with a TOTAL clock skew budget of less than or equal to 1.5ns across all MBus devices. That is to say the difference between the arrival time of the earliest and latest clocks (measured at 1.5V) to MBus modules should at most be 1.5ns.

6.9. Level-1 and Level-2 A.C. Characteristics(Scan) (Ta = 0-70C)

SCAN (SCANDI, SCANDO SCANTMS1, SCANTMS2)

Signal	SPEC	Comments
Clk to output max	25.0	Measured Relative to SCANCLK Falling Edge
Clk to output min	2.5	Measured Relative to SCANCLK Falling Edge
Input set up max	20.0	Measured Relative to SCANCLK Rising Edge
Input Hold max	20.0	Measured Relative to SCANCLK Rising Edge

NOTE: All times in nano-seconds, timing values are relative to the pads on the DIE.

Table 10 - MBus Level-1 and 2 A.C. Characteristics (Scan signals)

6.10. Level-1 and Level-2 DC Characteristics (Ta = 0-70C)

Symbol	Signal Description	Conditions	min	max	unit
VCC	Supply Voltage		4.75	5.25	V
Vih	Input High Voltage level		2.0	VCC	V
Vil	Input low Voltage level		VSS	.8	V
Iil	Input Leakage			+- 1.0	μA
Iol	Output Leakage			+- 1.0	μA
Iih	Input High Current			20	μA
Iilo	Input Low Current			-10	μA
Voh	Output high Voltage	Ioh = -2mA	2.4	VCC	V
Vol	Output Low Voltage	Iol = 2mA	0.0	0.4	V
Vol	Output Low Voltage(MSH*)	Iol = 8mA	0.0	0.4	V
Vol	Output Low Voltage(AERR*)	Iol = 3mA	0.0	0.4	V
Cin	Input Capacitance			10	pF
Cout	Output Capacitance			12	pF
Ci/o	Input/Output Capacitance			15	pF

Table 11 - Level-1 and Level-2 DC Characteristics

The DC specification represents the minimum MBus requirements. MBus components may exceed the minimum requirements. It should not be inferred that meeting the DC specifications means that this will guarantee compliance with the AC specifications.

6.11. General Electrical Topics

6.11.1. A.C. Threshold Assumptions

Inside the chip

The threshold point used to evaluate timing inside of a chip will vary from vendor to vendor. Consult with your vendor to establish the threshold point of the MBus driver input..

Outside the chip (System level)

All MBus levels are TTL and should be measured as follows: Low=.8V; High=2.0v. For the clocks however, 1.5V should be used as the threshold for the purposes of making timing measurements and calculations.

6.11.2. Asynchronous signals

An IRL[3:0] level should be treated as an asynchronous input group by a processor module. All four IRL lines should transition between valid logic levels within 10ns to avoid spurious interrupts on fast modules.

RSTIN* should be treated by a module as an asynchronous input signal. It may be asserted for several reasons. When asserted for power-on reset it is recommended that it be asserted for a minimum of100ms. For other reset events it is recommended that it be asserted for a minimum of 25uS.

AERR* should be treated as an asynchronous input signal to the interrupt logic. It must remain asserted until de-asserted by software action.

MID[3:0] are hard-wired stable signals.

6.11.3. Reset and Initialization

As explained under the signal description of RSTIN*, modules should tri-state all bussed lines and drive all point to point signals inactive high when RSTIN* is asserted. This must be accomplished within the default assertion time of RSTIN*. It is the responsibility of the system to ensure that all bussed control signals(CNTRL) are high and all MAD lines are at stable logic levels before RSTIN* is de-asserted to any module. After coming out of reset, processor modules may take some time to respond to transactions. System designers need to be aware of the characteristics of the processor modules they intend to support in order to ensure they do not time out when accessing modules coming out of reset. The SCANTMS2 signal must be asserted during power-up reset to modules that implement P1149 (JTAG).

6.11.4. Pull Ups and Holding Amplifiers

High impedance pull-up resistors are required on MAS*, MRDY*, MRTY*, MERR*, MBB* and MIH*. 10 Kohms is recommended. A 1.5K ohm pull up resistor is recommended for AERR*. A 619 ohm pull up resistor is recommended for MSH*. MAD[63:0] signals require holding amplifiers. An example of a holding amplifier is shown below. The recommended high output impedance driver Iol and Ioh max, at Vol and Voh respectively, is 100μA. It is the responsibility of the system to provide pull ups and holding amplifiers.

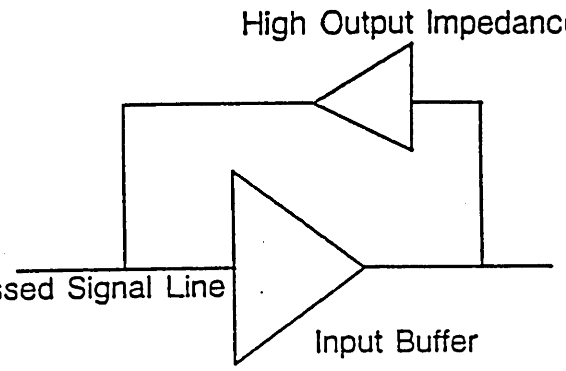


Figure 16 – Holding amplifier

Chapter 7

7. Mechanical Specifications

The mechanical specification consists of the connector and its associated pin-out, and the mechanical drawings for the module printed wiring boards.

7.1. MBus Connector

The MBus Module concept of field-upgradeable performance requires a connector with high reliability, ease of installation, and a fool-proof keying scheme. The aggressive MBus cycle time goal of 25ns requires a connector with excellent electrical performance, i.e., transmission line characteristics with low capacitance and inductance. Finally, promoting MBus as a physical, as well as logical standard requires a connector which is widely available in the marketplace. Based on the above requirements, the AMP "Microstrip"(part # 121354-4) has been selected. This is the module part (shrouded male pins) of a mating pair of connectors.

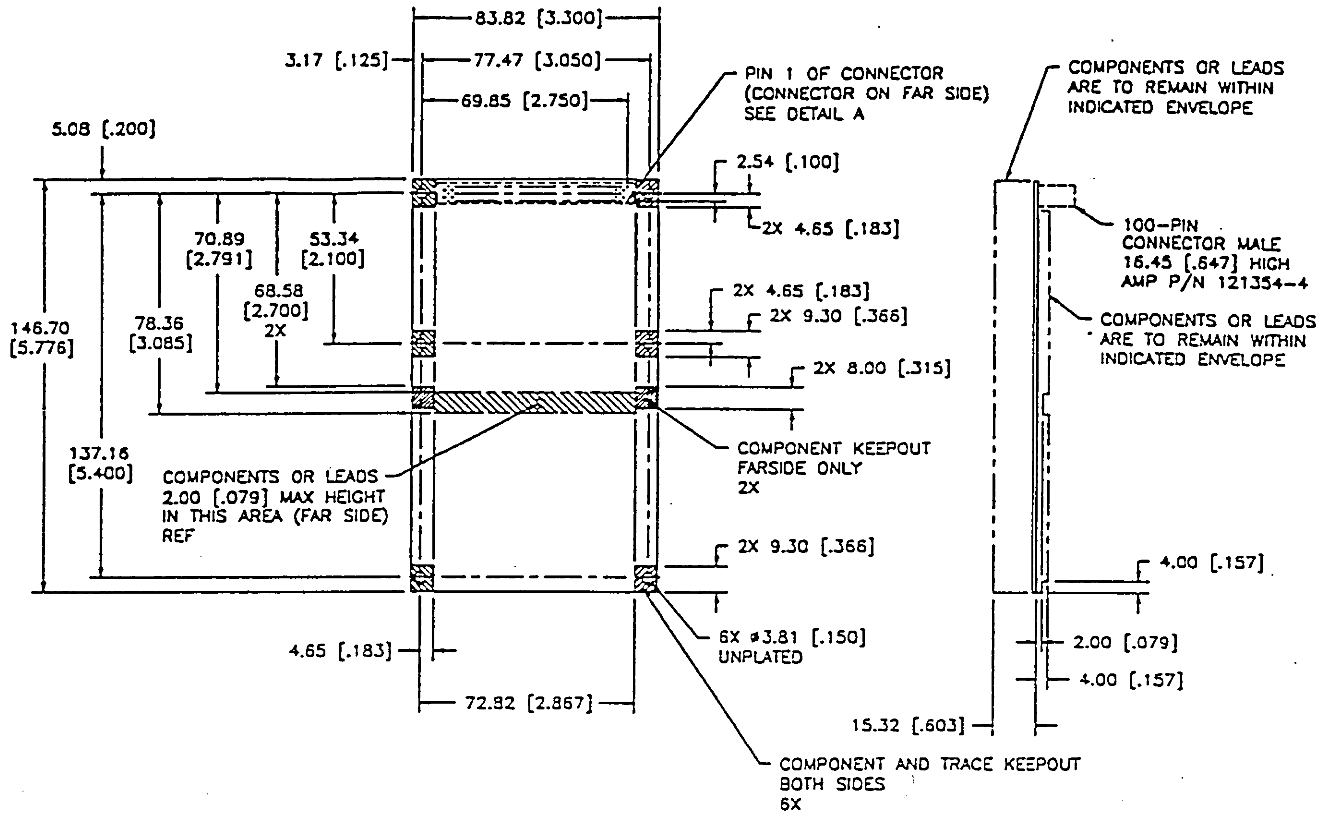
CAUTION: Make sure the system the module is being designed for provides standoffs on either side of the connector. This is required to help prevent breakage of the connector if by accident the module is "over rotated" while "walking" the module out. That is to say, if one end of the connector stays mated while the other is de-mated it is very likely that the end of the connector that stays mated will break once the module has swung up by 30 - 40 degrees.

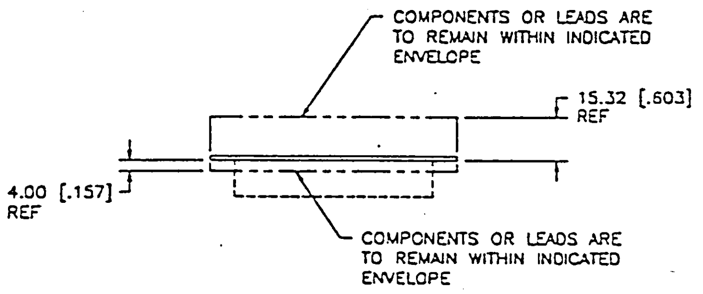
7.2. MBus Connector pin-out

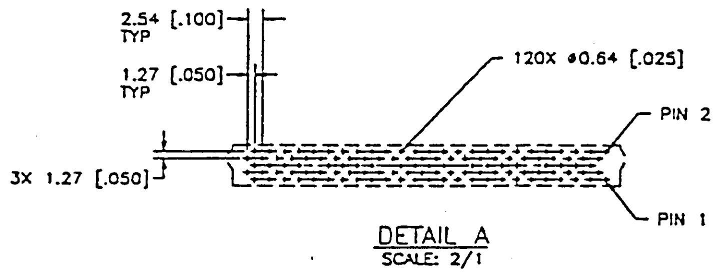
For a description of the signals listed in NO TAG refer to Chapter 2. It should be noted here that the signals provided at the connector are sufficient to support two processors per module. Therefore you will find two sets of bus request, bus grant and interrupt lines. The clock signals MCLK0-3 are identical clocks that must be provided by all systems. Four clocks are provided to allow for modules that have many clock loads.

1.	SCANDI	Blade1		2.	SCAN TMS1
3.	SCANDO			4.	SCANTMS2
5.	SCANCLK		GND	6.	IRL0[1]
7.	IRL0[0]			8.	IRL0[3]
9.	IRL0[2]		GND	10.	INTOUT*
11.	MAD[0]			12.	MAD[1]
13.	MAD[2]		GND	14.	MAD[3]
15.	MAD[4]			16.	MAD[5]
17.	MAD[6]		GND	18.	MAD[7]
19.	MAD[8]			20.	MAD[9]
21.	MAD[10]	Blade2		22.	MAD[11]
23.	MAD[12]			24.	MAD[13]
25.	MAD[14]		+5V	26.	MAD[15]
27.	MAD[16]			28.	MAD[17]
29.	MAD[18]		+5V	30.	MAD[19]
31.	MAD[20]			32.	MAD[21]
33.	MAD[22]		+5V	34.	MAD[23]
35.	MAD[24]			36.	MAD[25]
37.	MAD[26]		+5V	38.	MAD[27]
39.	MAD[28]			40.	MAD[29]
41.	MAD[30]	Blade3		42.	MAD[31]
43.	MBR0*			44.	MSH*
45.	MBG0*		GND	46.	MIH*
47.	MCLK0			48.	MRTY*
49.	MCLK1		GND	50.	MRDY*
51.	MCLK2			52.	MERR*
53.	MCLK3		GND	54.	MAS*
55.	MBR1*			56.	MBB*
57.	MBG1*		GND	58.	RESERVED1
59.	MAD[32]			60.	MAD[33]
61.	MAD[34]	Blade4		62.	MAD[35]
63.	MAD[36]			64.	MAD[37]
65.	MAD[38]		+5V	66.	MAD[39]
67.	MAD[40]			68.	MAD[41]
69.	MAD[42]		+5V	70.	MAD[43]
71.	MAD[44]			72.	MAD[45]
73.	MAD[46]		+5V	74.	MAD[47]
75.	MAD[48]			76.	MAD[49]
77.	MAD[50]		+5V	78.	MAD[51]
79.	MAD[52]			80.	MAD[53]
81.	MAD[54]	Blade5		82.	MAD[55]
83.	MAD[56]			84.	MAD[57]
85.	MAD[58]		GND	86.	MAD[59]
87.	MAD[60]			88.	MAD[61]
89.	MAD[62]		GND	90.	MAD[63]
91.	RESERVED2			92.	IRL1[0]
93.	IRL1[1]		GND	94.	IRL1[2]
95.	IRL1[3]			96.	AERR*
97.	RSTIN*		GND	98.	ID[1]
99.	ID[2]			100.	ID[3]

Table 12 - MBus pin out


83.82 [3.300]
3.17 [.125]
77.47 [3.050]
69.85 [2.750]
5.08 [.200]
PIN 1 OF CONNECTOR (CONNECTOR ON FAR SIDE) SEE DETAIL A
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
2.54 [.100]
2X 4.65 [.183]
70.89 [2.791]
53.34 [2.100]
68.58 [2.700] 2X
146.70 [5.776]
78.36 [3.085]
2X 4.65 [.183]
2X 9.30 [.366]
2X 8.00 [.315]
100-PIN CONNECTOR MALE 16.45 [.647] HIGH AMP P/N 121354-4
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
137.16 [5.400]
COMPONENTS OR LEADS 2.00 [.079] MAX HEIGHT IN THIS AREA (FAR SIDE) REF
COMPONENT KEEPOUT FARSIDE ONLY 2X
2X 9.30 [.366]
4.00 [.157]
4.65 [.183]
6X ø3.81 [.150] UNPLATED
2.00 [.079]
4.00 [.157]
72.82 [2.867]
15.32 [.603]
COMPONENT AND TRACE KEEPOUT BOTH SIDES 6X

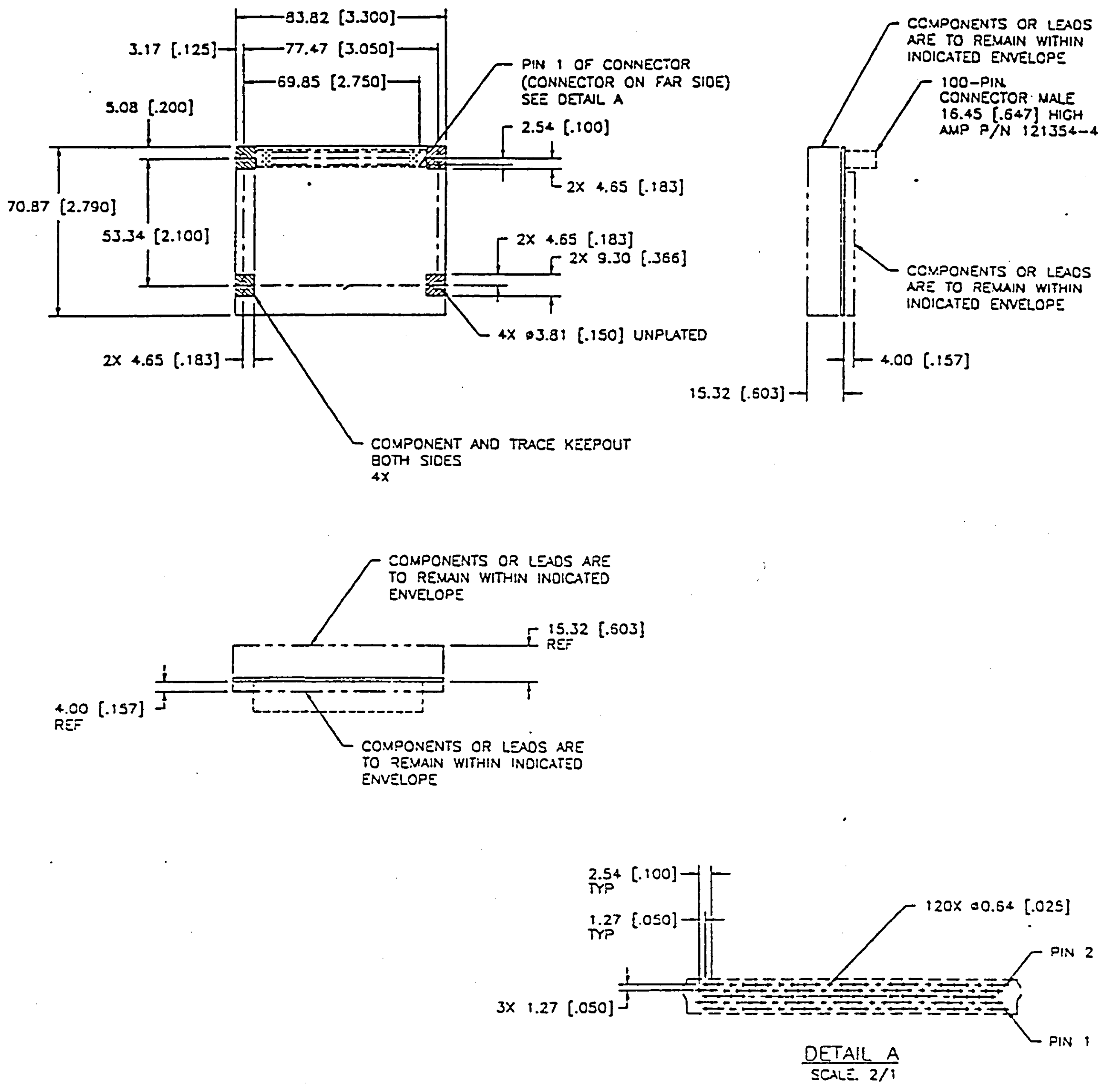

COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
15.32 [.603] REF
4.00 [.157] REF
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE



NOTES, UNLESS OTHERWISE SPECIFIED:

1. METRIC DIMENSIONS ARE PRIMARY DIMENSIONS. ENGLISH DIMENSIONS, ENCLOSED IN BRACKETS, ARE REFERENCE ONLY.
2. ALL HOLES TO BE PLATED THRU. HOLE DIAMETERS REFER TO FINISHED HOLE SIZE.

Figure 17 – Full Size MBus module


83.82 [3.300]
3.17 [.125]
77.47 [3.050]
69.85 [2.750]
5.08 [.200]
PIN 1 OF CONNECTOR (CONNECTOR ON FAR SIDE) SEE DETAIL A
2.54 [.100]
2X 4.65 [.183]
70.87 [2.790]
53.34 [2.100]
2X 4.65 [.183]
2X 9.30 [.366]
4X ⌀3.81 [.150] UNPLATED
2X 4.65 [.183]
COMPONENT AND TRACE KEEPOUT BOTH SIDES 4X
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
100-PIN CONNECTOR MALE 16.45 [.647] HIGH AMP P/N 121354-4
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
4.00 [.157]
15.32 [.603]
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
15.32 [.603] REF
4.00 [.157] REF
COMPONENTS OR LEADS ARE TO REMAIN WITHIN INDICATED ENVELOPE
2.54 [.100] TYP
1.27 [.050] TYP
120X ⌀0.64 [.025]
PIN 2
3X 1.27 [.050]
PIN 1
DETAIL A
SCALE 2/1

NOTES, UNLESS OTHERWISE SPECIFIED:

1. METRIC DIMENSIONS ARE PRIMARY DIMENSIONS. ENGLISH DIMENSIONS, ENCLOSED IN BRACKETS, ARE REFERENCE ONLY.
2. ALL HOLES TO BE PLATED THRU. HOLE DIAMETERS REFER TO FINISHED HOLE SIZE.

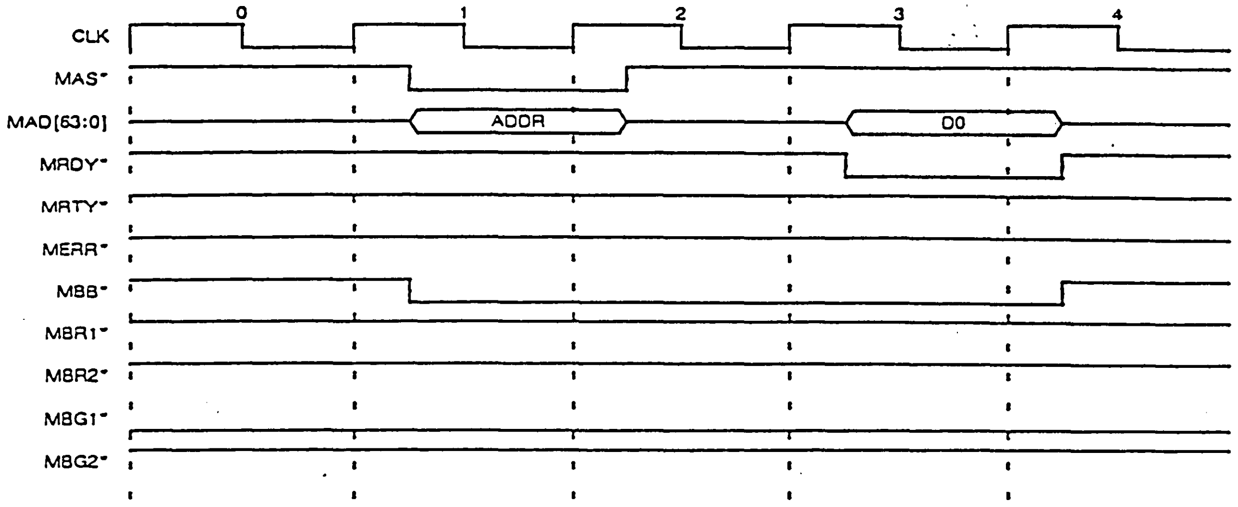
Figure 18 - Half Size MBus module

Chapter 8

8. Timing Diagram Examples

The following idealized wave-form diagrams are included in order to assist in understanding the operations of the MBus. In most of the waveforms, the bus is shown already granted to one of the masters. All waveforms also show MBB* asserting with MAS* and de-asserting immediately after the last acknowledgment is received. This is not a requirement of the MBus protocol. It should also be noted that only two masters are depicted using the MBus (MBR1*, MBR2*). However, this can be extended to as many masters as can be supported electrically (this was discussed in Chapter 6). A line intermediate between high and low is shown on MAD[63:0]. This indicates the time that the MAD lines are driven by holding amplifiers and does not indicate that an indeterminate voltage level is permitted on MAD[63:0]. The cross hatched areas indicate the bus is being driven to valid logic levels but the data is indeterminate.

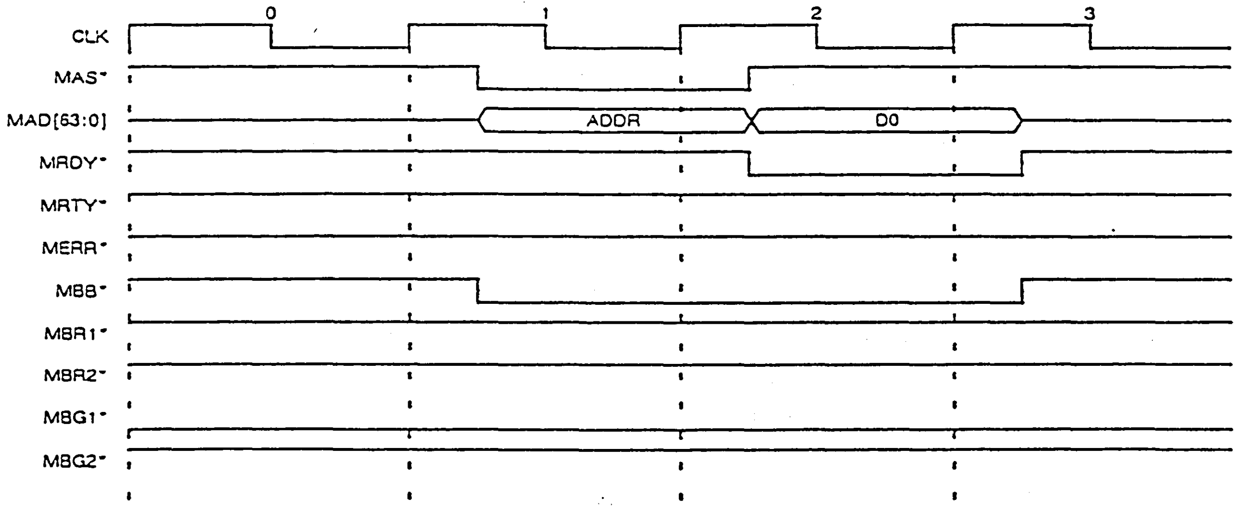
8.1. Word Read



Waveform 1 - Word Read

Waveform 1 depicts a simple word read transaction by a master who has already been granted the bus (master 1). The number of cycles between the address cycle and the data cycle(s) of all the waveforms in this section is implementation dependent. Waveform 1 shows a minimum memory latency of 2 cycles.

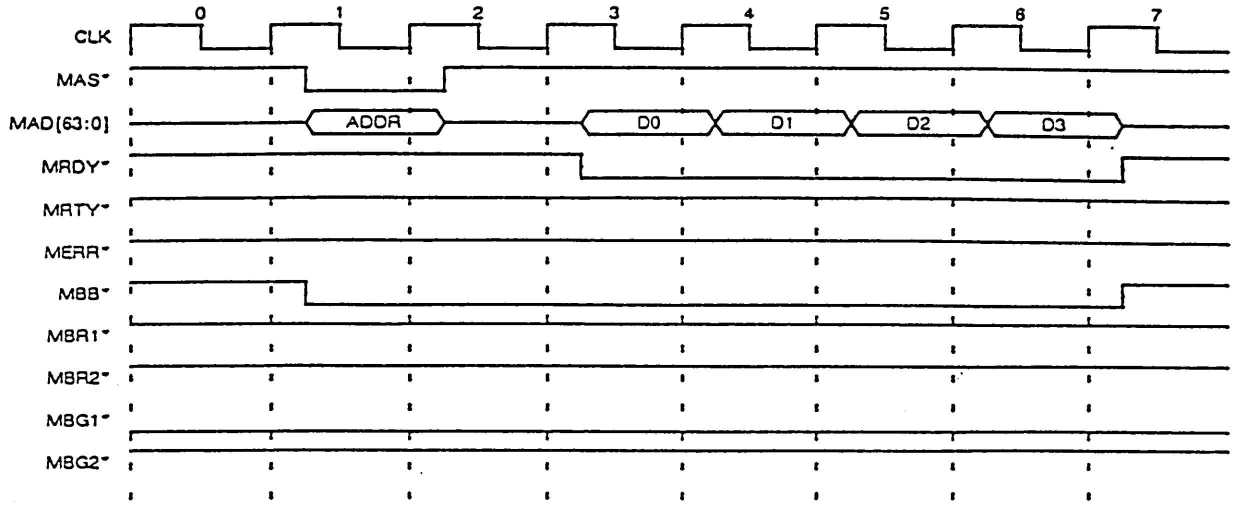
8.2. Word Write



Waveform 2 - Word Write

Waveform 2 depicts a simple word write transaction by a master who has already been granted the bus (master 1). Note how the data is immediately driven onto the MAD lines after the address cycle and that the bus has been granted to master 1 prior to the beginning of the transaction. The slave response time is implementation dependent. Waveform 2 depicts the minimum write time of 2 cycles.

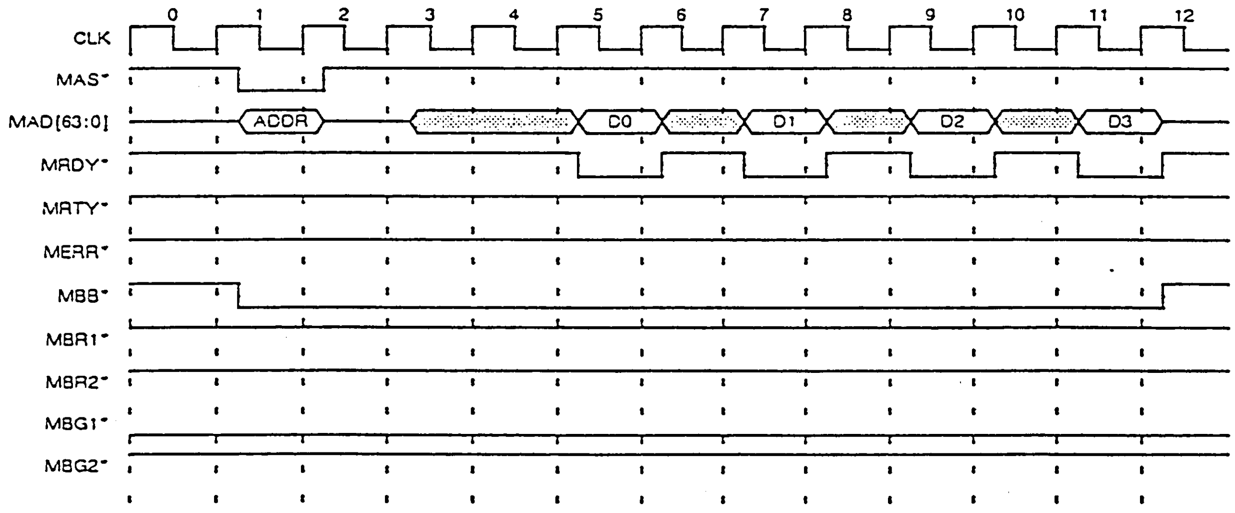
8.3. Burst Read with No Delays



Waveform 3 – Burst Read with No Delays

Waveform 3 depicts a burst read operation of 32 bytes where the slave device supplies the data at the maximum rate possible. Note again that the bus has been granted to master 1 prior to the beginning of the transaction.

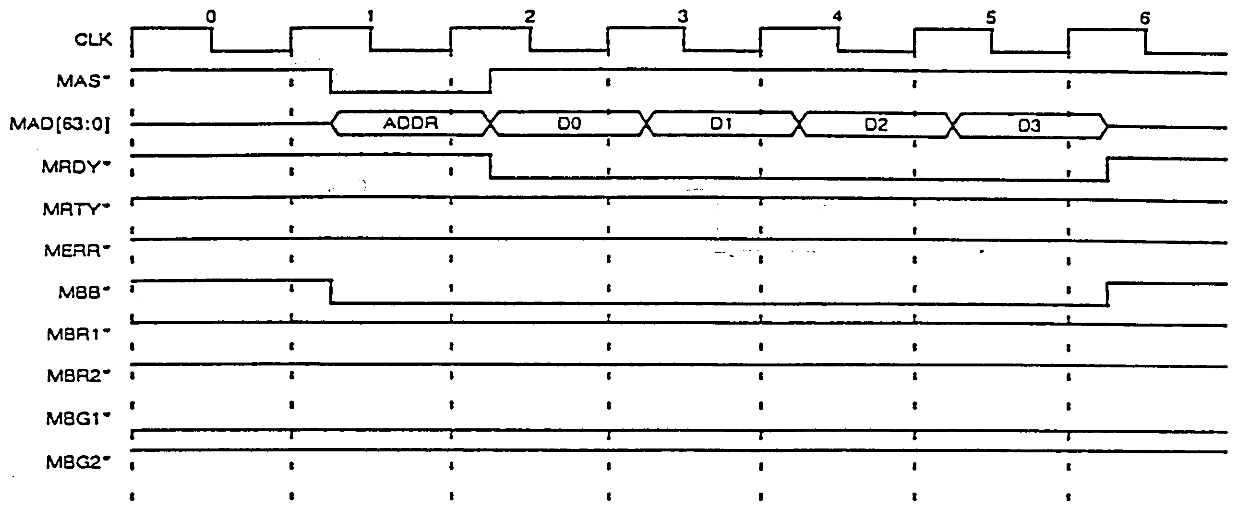
8.4. Burst Read with Delays



Waveform 4 – Burst Read with Delays

Waveform 4 depicts a burst read operation of 32 bytes where the slave device cannot supply the data at the maximum rate possible. Wait states are inserted between each 8-byte quantity by simply de-asserting MRDY* for an arbitrary number of cycles (Waveform 4 shows 1 cycle of delay inserted between each 8-byte transfer as well as 3 wait states before the burst response begins). Note again that the bus has been granted to master 1 prior to the beginning of the transaction.

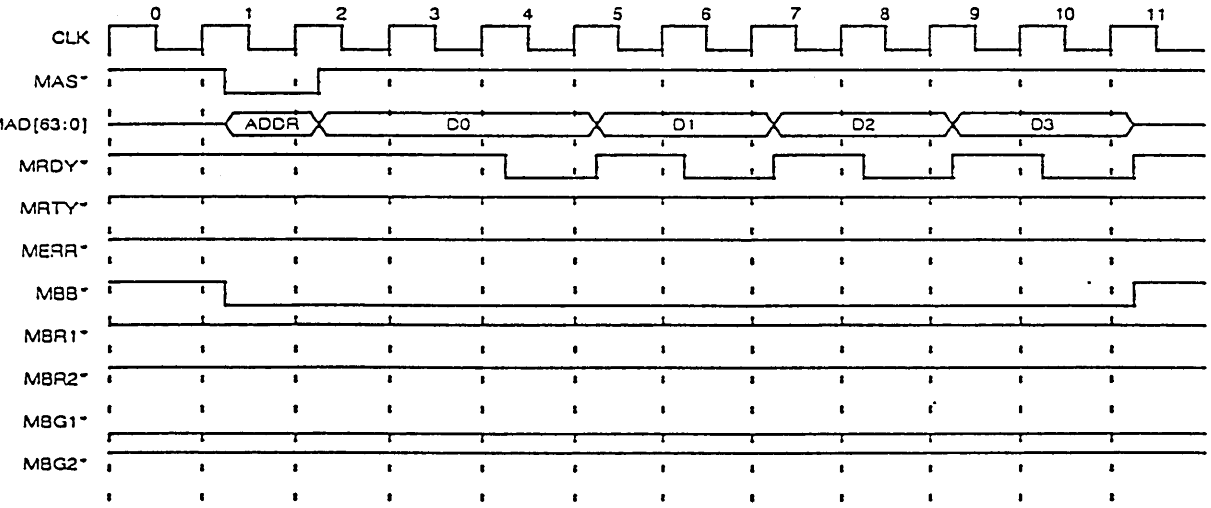
8.5. Burst Write with No Delays



Waveform 5 - Burst Write with No Delays

Waveform 5 depicts a burst write operation of 32 bytes where the slave device accepts the data at the maximum rate possible. Note again that the bus has been granted to master 1 prior to the beginning of the transaction. Note also how the next double word to be written is driven in the cycle immediately after MRDY* asserts.

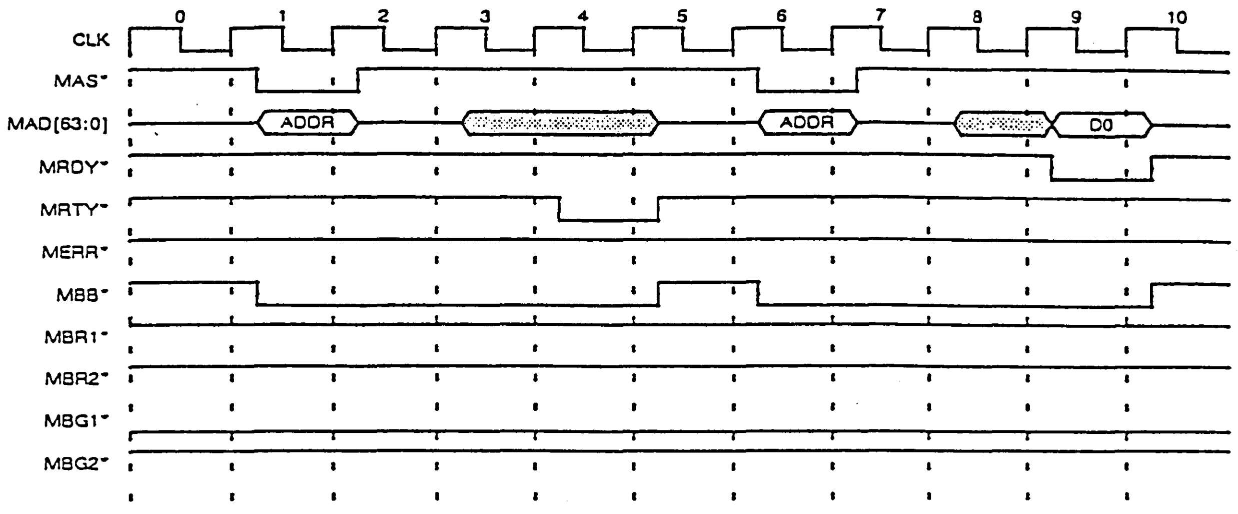
8.6. Burst Write with Delays



Waveform 6 - Burst Write with Delays

Waveform 6 depicts a burst write operation of 32 bytes where the slave device cannot accept the data at the maximum rate possible. Wait states are inserted between each 8-byte quantity by simply de-asserting MRDY* for an arbitrary number of cycles (Waveform 6 shows 1 cycle of delay inserted between each 8-byte transfer as well as 2 wait states before the beginning of the burst). Note again that the bus has been granted to master 1 prior to the beginning of the transaction. Also note how the next double word to be written is driven in the cycle immediately after MRDY* asserts.

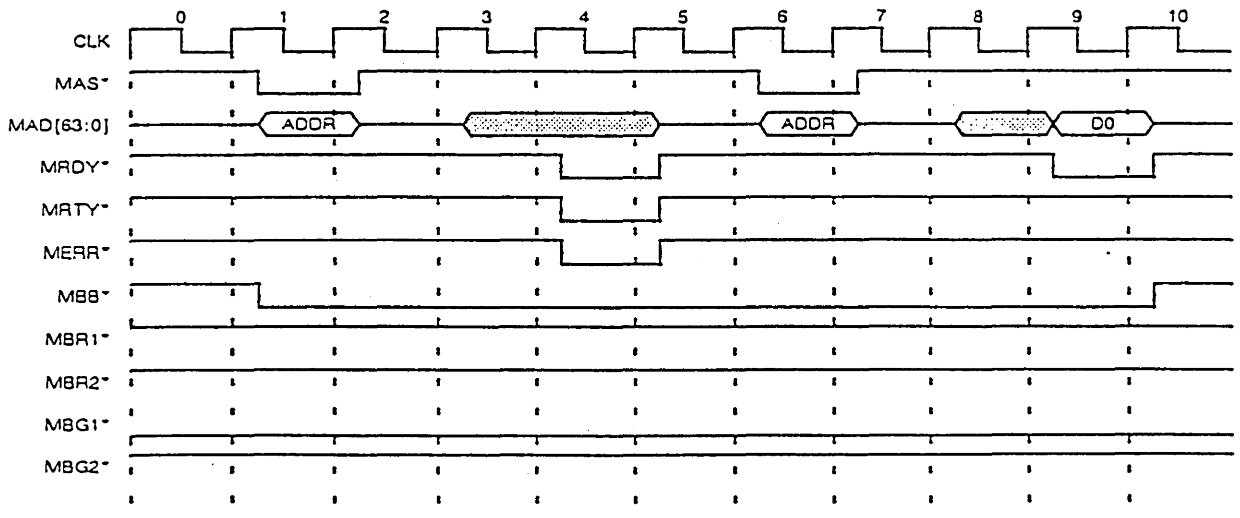
8.7. Relinquish and Retry



Waveform 7 - Relinquish and Retry

Waveform 7 depicts a Read operation in which the addressed slave needed to inform the master that it should relinquish the bus and retry the operation again once bus ownership has been attained. In the case shown above, the master in question maintained ownership as no one else wanted the bus. Thus, only one dead cycle is present between the Read transactions.

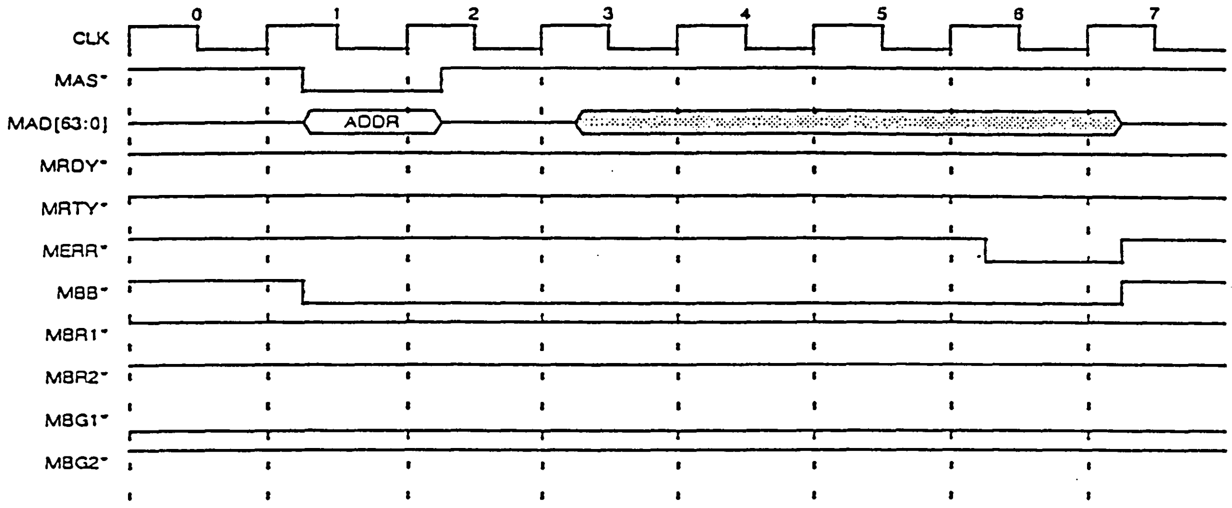
8.8. Retry



Waveform 8 - Retry

Waveform 8 depicts a Read operation in which the addressed slave needed to inform the master that it should retry the operation immediately. The master must always keep MBB* asserted upon a Retry acknowledgment. However, there must be at least one dead cycle between the Retry acknowledgment and the ensuing MAS* as is shown in Waveform 8. This dead cycle must be present for both read and write operations which were Retried.

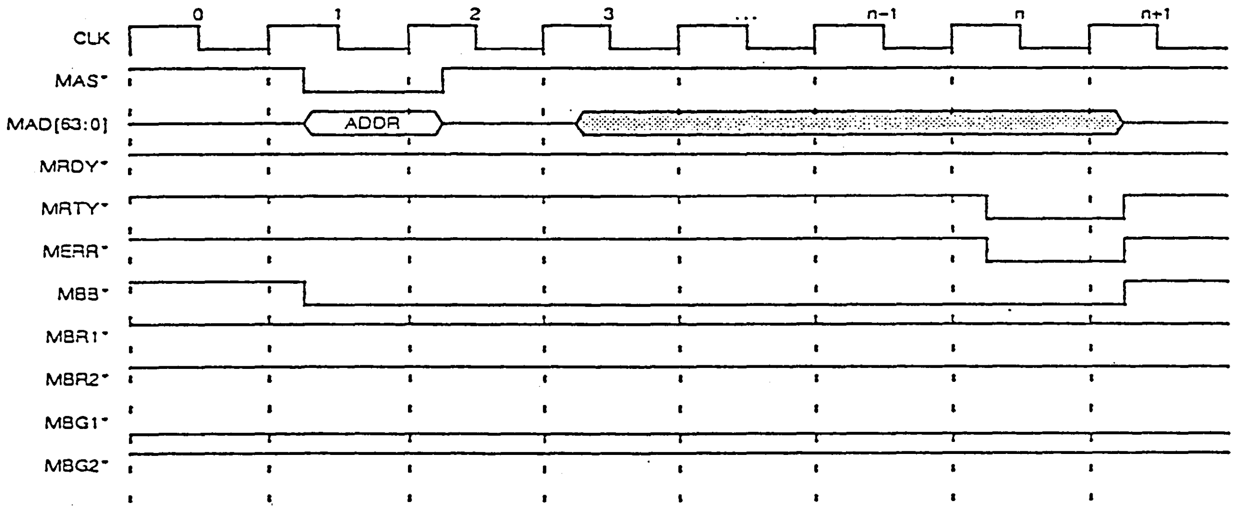
8.9. ERROR1 (Bus Error)



Waveform 9 - ERROR1 (Bus Error)

Waveform 9 depicts an operation in which the addressed slave detected some sort of a bus error or other system implementation dependent error.

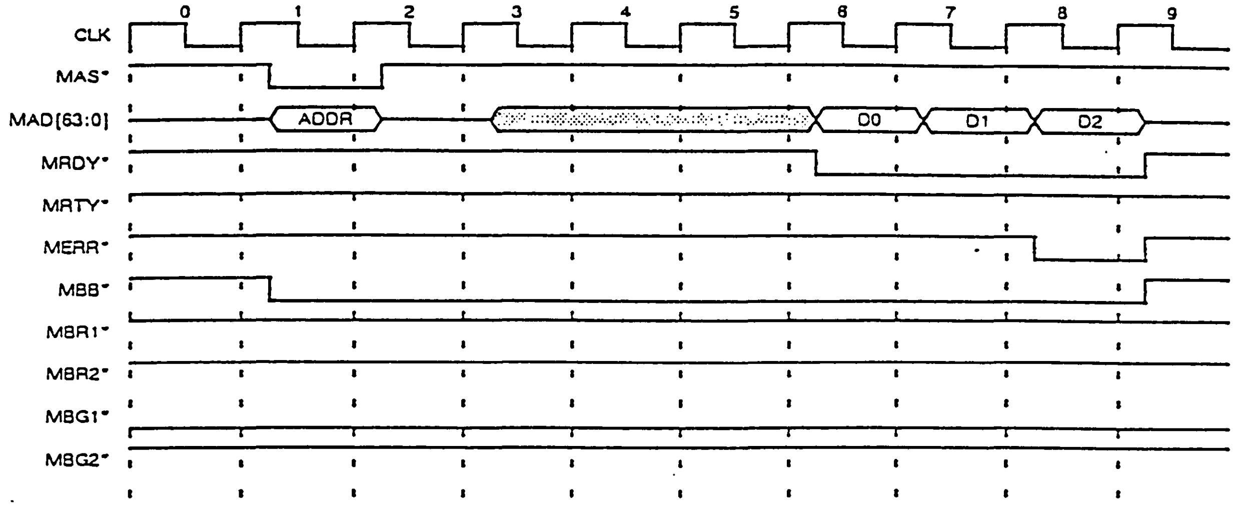
8.10. ERROR2 (Timeout)



Waveform 10 - ERROR2 (Timeout)

Waveform 10 depicts an operation in which a timeout or other system implementation dependent error occurred.

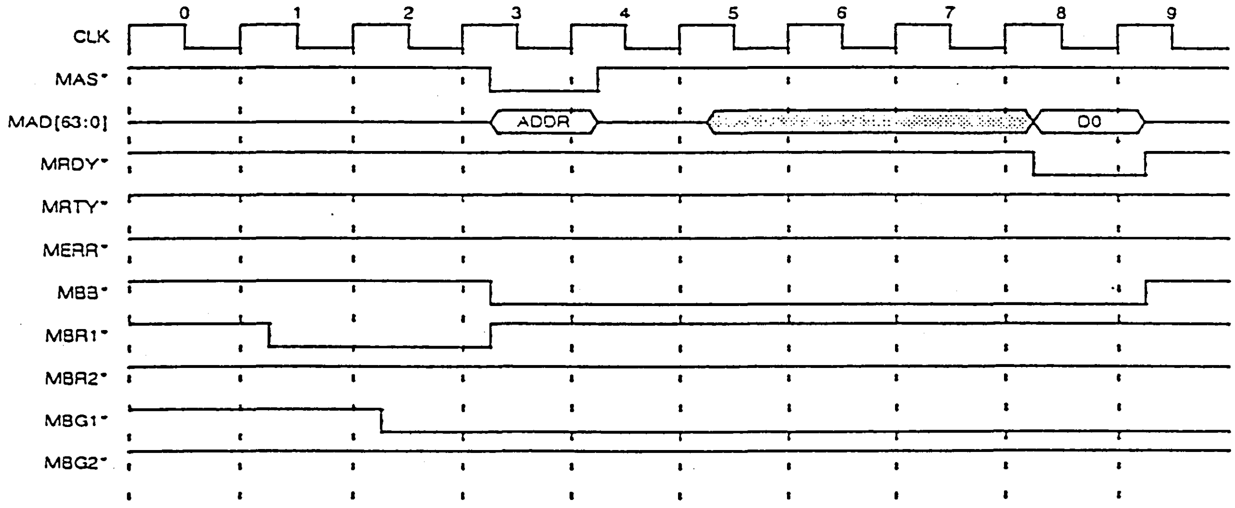
8.11. ERROR3 (Uncorrectable)



Waveform 11 - ERROR3 (Uncorrectable)

Waveform 11 depicts an operation in which an uncorrectable or other system implementation dependent error occurred. Note how the Uncorrectable acknowledgment came on the third data transfer of a burst operation. The transaction must be immediately aborted upon detection of an error acknowledgment. This also applies to ERROR1 and ERROR2 acknowledgements to burst transactions.

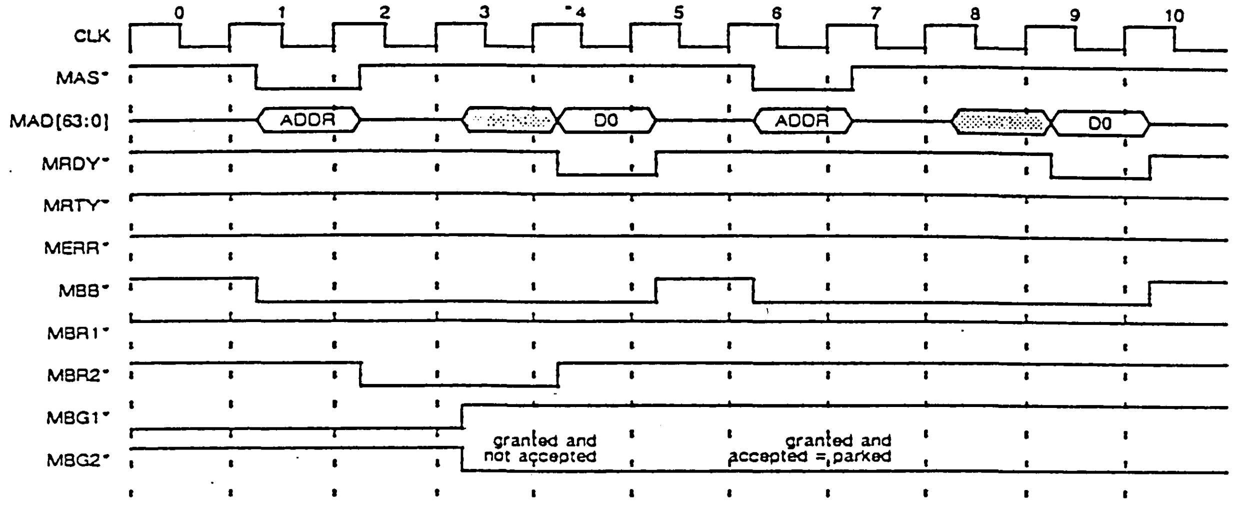
8.12. Initial Bus Arbitration



Waveform 12 - Initial Bus Arbitration

Waveform 12 depicts a master requesting an idle bus in order to perform a word read. This actually depicts the first transaction after reset, as there is no grant initially, a condition that can only occur at that time.

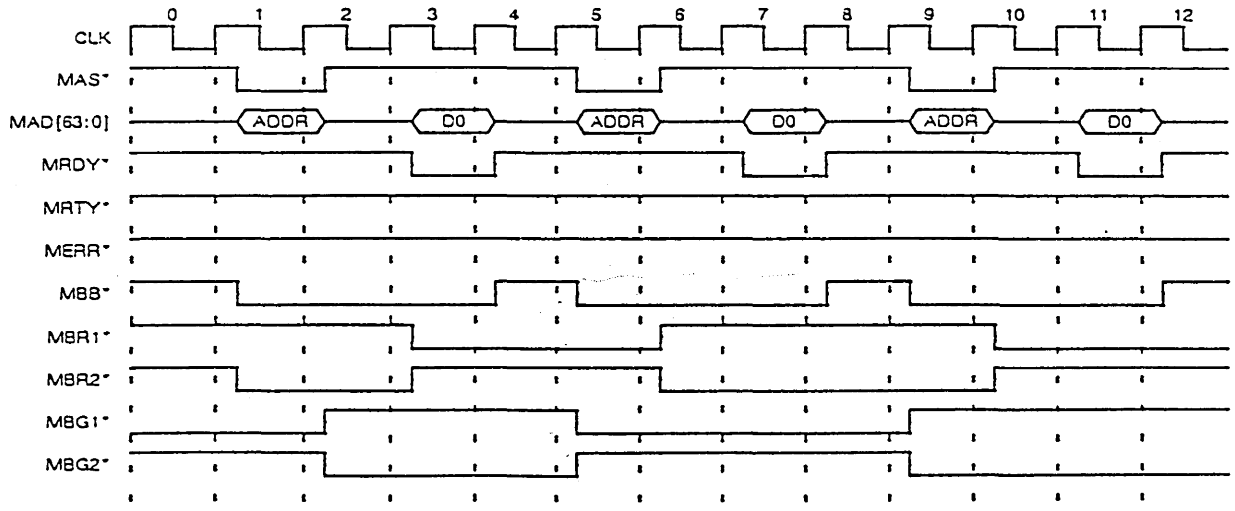
8.13. Arbitration from Master 1 to Master 2



Waveform 13 – Arbitration from Master 1 to Master 2

Waveform 13 depicts how the bus ownership is turned over to another master. It was assumed that the bus was already parked on master 1 and module 1 decides to start a cycle during cycle 1 by asserting MBB* and MAS*. Later Master 2 requests and is granted the bus. After MBB* is de-asserted Master 2 drives MAS* and MBB*. At this time the grant is parked on master 2.

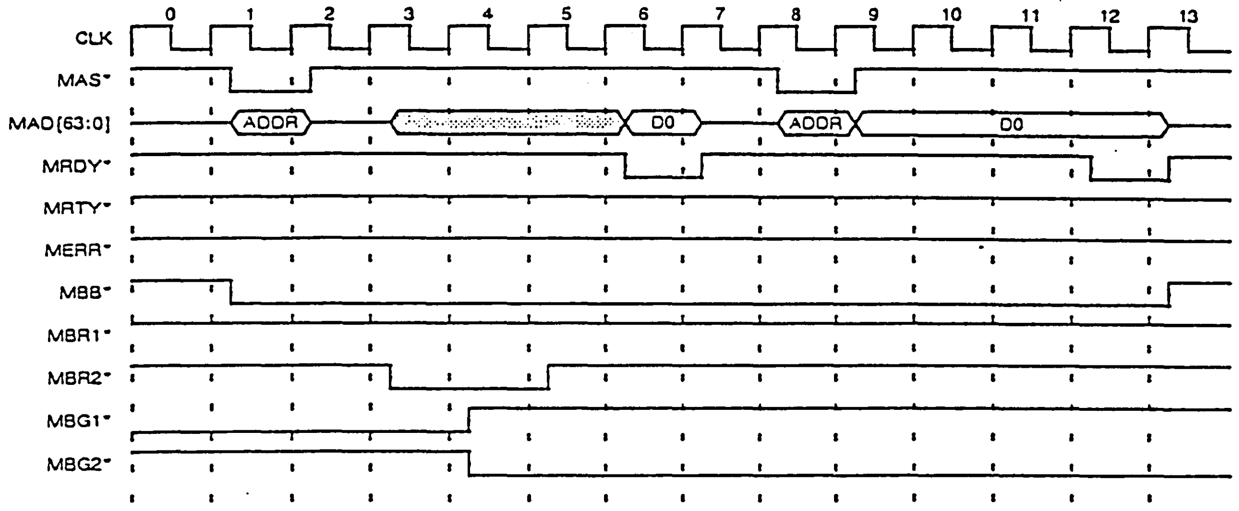
8.14. Arbitration with Multiple Requests



Waveform 14 – Arbitration with Multiple Requests

Waveform 14 depicts arbitration when two masters are continually requesting bus ownership. Initially the bus has been parked on master 1 who then performs a transaction by asserting MBB* and MAS*. Master 2 then acquires the bus followed by master 1 again. Note how the requests must be de-asserted once the grants are obtained.

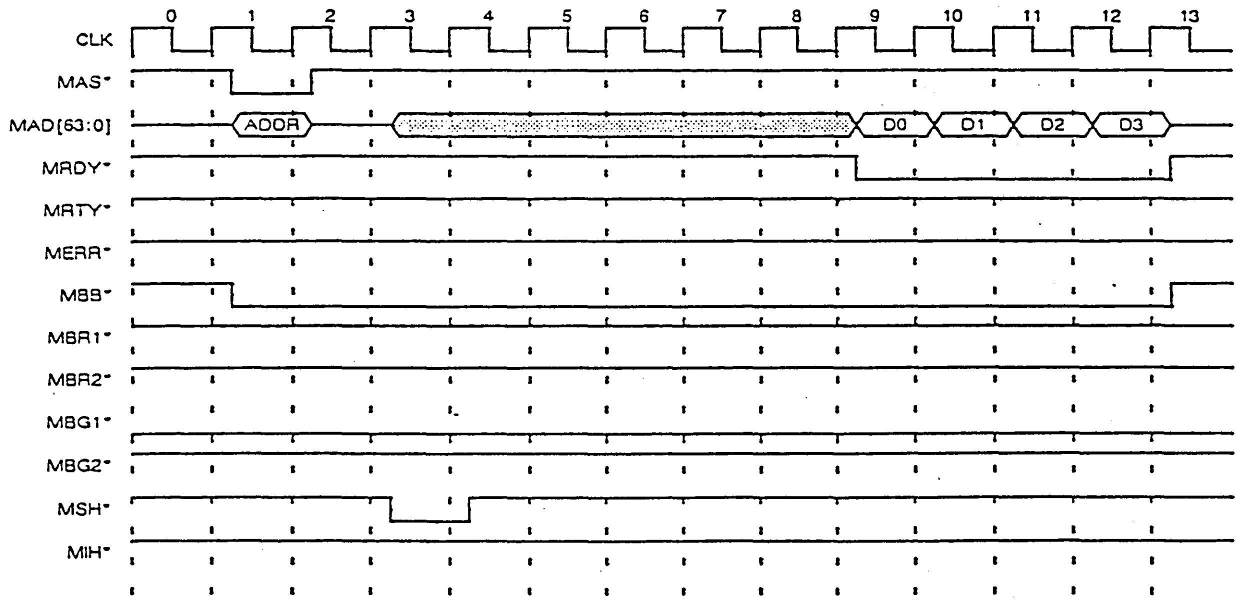
8.15. Locked Cycles



Waveform 15 - Locked Cycles

Waveform 15 depicts a read cycle followed by a locked write cycle. Note how MBB* does not de-assert until the completion of the write operation, despite re-arbitration of the bus as in the above case.

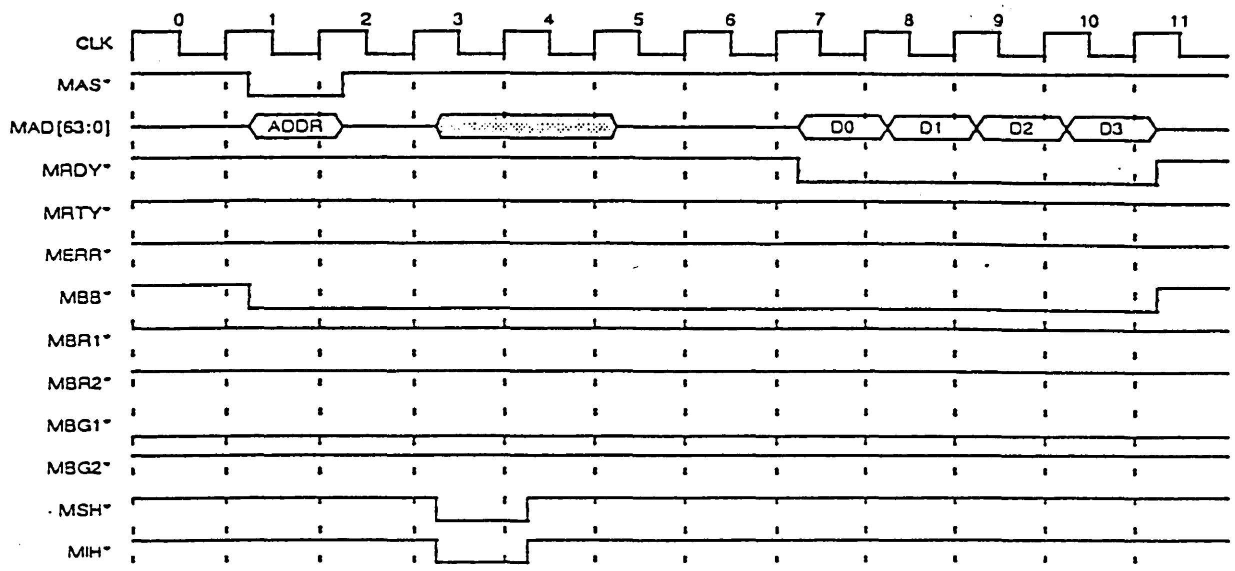
8.16. Coherent Read of Shared Data



Waveform 16 - Coherent Read of Shared Data

Waveform 16 depicts a Coherent Read in which the requested data actually exists in one (or more) other cache(s) in the system, but is not owned by any cache(s). These caches will assert MSH* on cycle A+2 (or A+7 – refer to Appendix B) as shown.

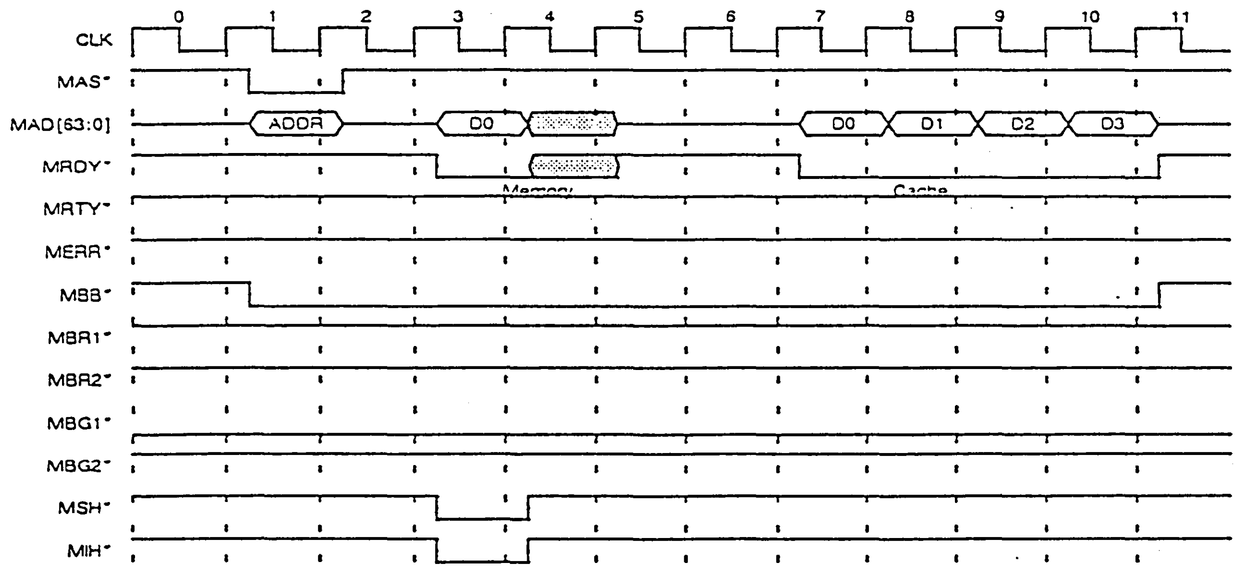
8.17. Coherent Read of Owned Data (long-latency memory)



Waveform 17 - Coherent Read of Owned Data (long-latency memory)

Waveform 17 depicts a Coherent Read operation in which the requested data is owned by another cache in the system. The owning cache (as well as any other cache with the same data) will assert MSH* during cycle A+2 (or A+7 - refer to Appendix B). Only the owning cache will assert MIH*.

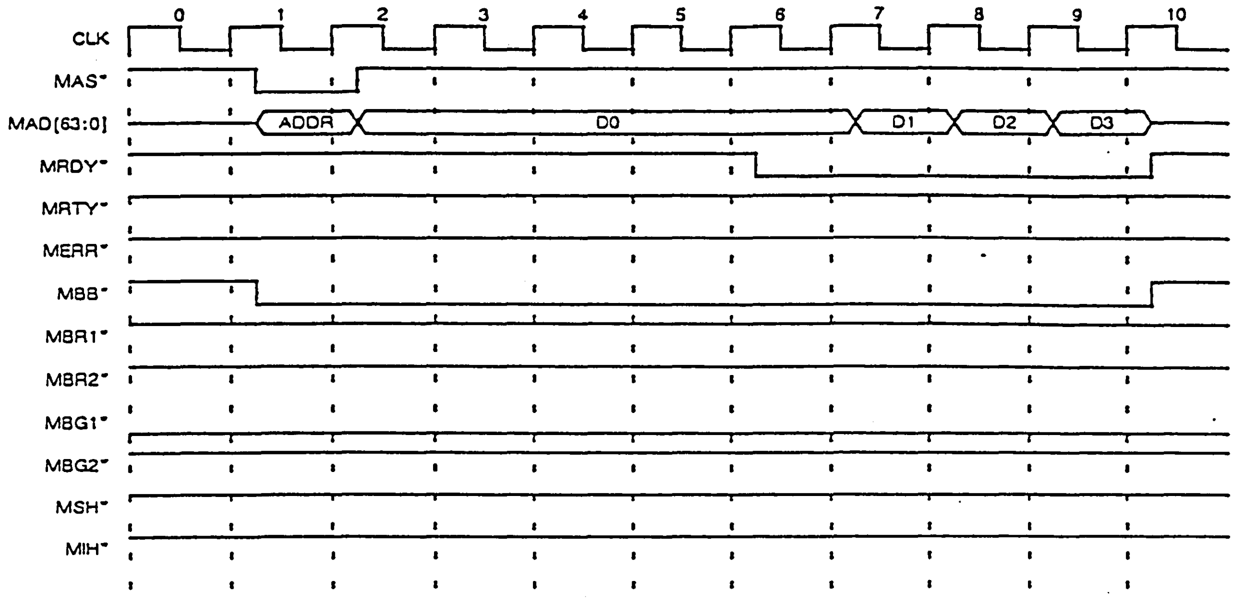
8.18. Coherent Read of Owned Data (fast memory)



Waveform 18 - Coherent Read of Owned Data (fast memory)

Waveform 18 depicts a Coherent Read operation in which the requested data is owned by another cache in the system. The owning cache (as well as any other cache with the same data) will assert MSH* during cycle A+2 (or A+7 - refer to Appendix B). Only the owning cache will assert MIH* and then supply the data. In this case above, memory has already begun to respond and thus must get off the bus immediately to allow the cache which owns the data to drive the bus.

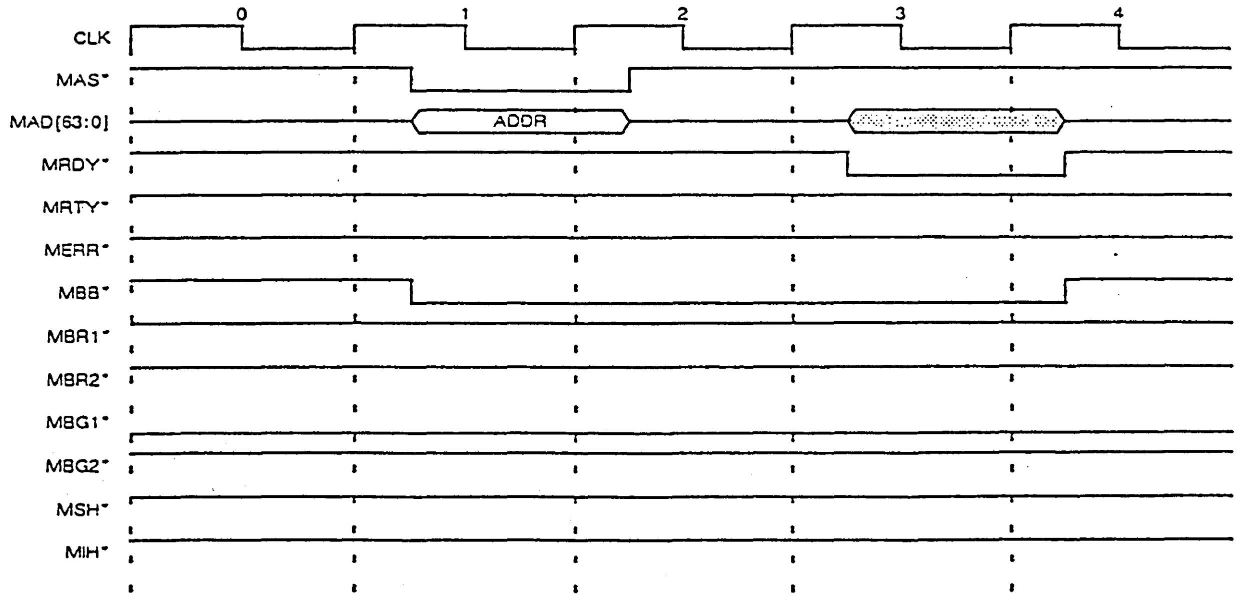
8.19. Coherent Write and Invalidate



Waveform 19 - Coherent Write and Invalidate

Waveform 19 depicts a Coherent Write and Invalidate operation in which one or more other caches in the system actually contained the data. The other cache(s) will not assert MSH* during this transaction, but will always invalidate the block.

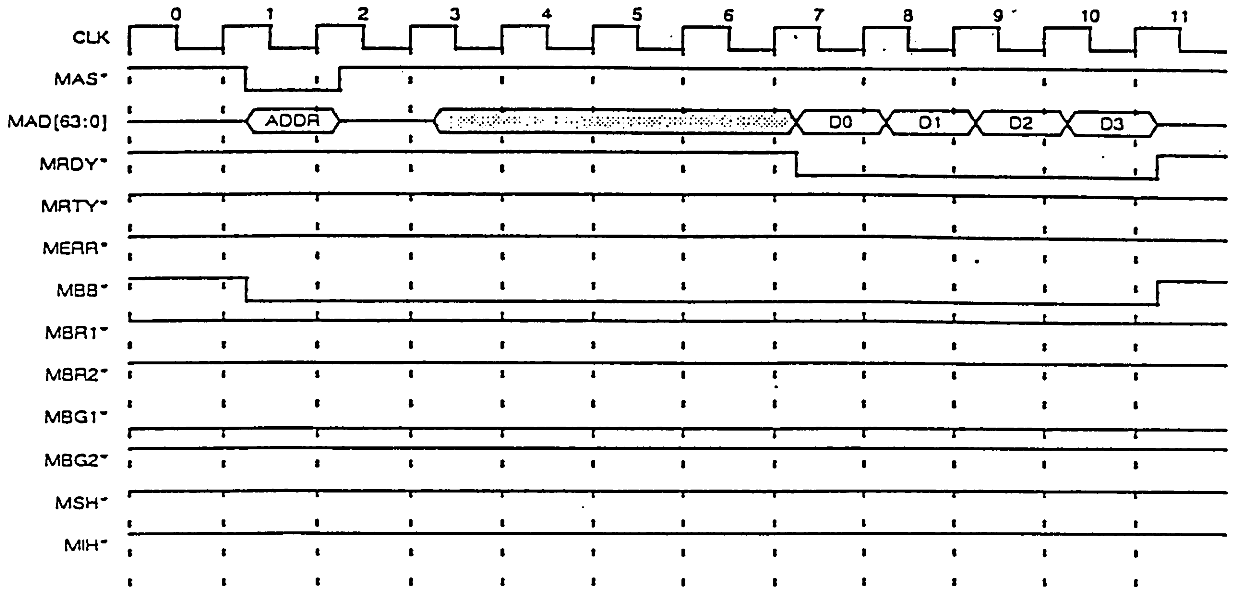
8.20. Coherent Invalidate



Waveform 20 - Coherent Invalidate

Waveform 20 depicts a Coherent Invalidate operation. Memory (or second level cache), in this case, will assert MRDY* during A+2 (or later - refer to Appendix B). System caches which contain the data being invalidated will not assert MSH* during this transaction.

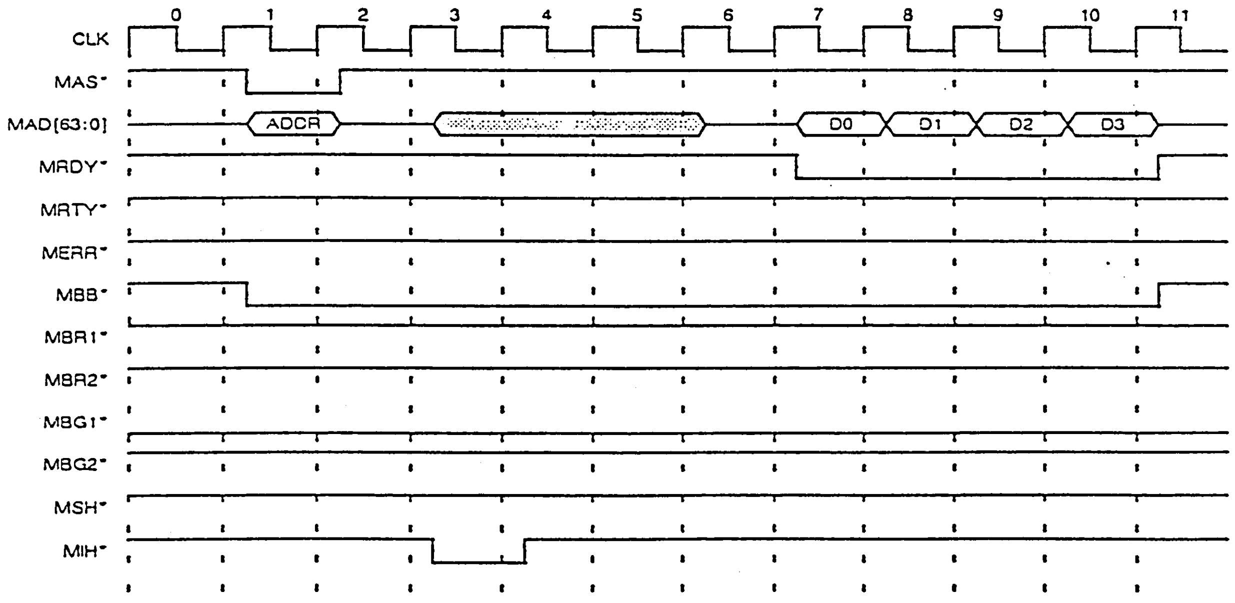
8.21. Coherent Read and Invalidate (of shared data)



Waveform 21 - Coherent Read and Invalidate (of shared data)

Waveform 21 depicts a Coherent Read and Invalidate operation in which no system cache owned the piece of data. All caches in the system will invalidate their copies of the block upon detection of a Coherent Read and Invalidate transaction. Note how the MSH* line does not assert for CRI.

8.22. Coherent Read and Invalidate (of owned data)



Waveform 22 - Coherent Read and Invalidate (of owned data)

Waveform 22 depicts a Coherent Read and Invalidate operation in which a system cache owned the piece of data and so supplied it. It then invalidated its copy. Other caches also invalidated their copies.

Chapter 9

9. Revision History

Revision	Date	Comments
0.7	1-20-89	First Revision of document. Preliminary copy release externally.
0.7.1	3-20-89	LEVEL 1: changed to a "SPARC" MBus. fixed block size to 32B distinguished between LOCK and MBB added more waveforms LEVEL 2: incorporated into one physical document w/ L1 added invalidate and excl rd and replace xaction changed to a write-invalidate protocol split flush into TLB and cache flush added more waveforms
0.8	7-5-89	Temporary Document. Never made it to release.
0.9	7-9-89	Interspersed both Level 1 and Level 2 information throughout entire document as opposed to separate chapters for each. Added several new chapters. Added more transaction types and changed some transaction type names. Removed all bus-based flushing from the document. Refined the electrical specs for Level 1 systems. Added address map and concept of MPR. Added MID lines. SUP bit. Removed dblwd wrapping support.
1.0	7-31-89	Put dblwd wrapping back in. Separated IRL/ID lines. Added more definitions. implementation notes (Appendix B). clarified MBB* semantics as well as LOCK. General clarification addition.
1.1	3-29-90	Constrained R&R to first ack in L-2 systems. Only MBB* needs to go from driven high to tri-state in one cycle. Waveforms were re-done. Added TI to appendix A. Miscellaneous clarifications throughout document.
1.2	1-18-91	Modified timeout mechanism to include MAS*. Defined Invalidate acknowledgement more accurately. Covered R&R on write-back case. Added Scan and INTOUT* signals. General clarifications. Extended Electrical Spec and added Mechanical Spec.

Table 13 - Revision History

Appendix A

A. MBus Port Register Assignments

MVEND	Vendor
0x0	Fujitsu
0x1	Cypress/Ross
0x2	Reserved
0x3	LSI Logic
0x4	TI
0xF	reserved for systems

Table 14 – MPR Vendor Assignment

Appendix B

B. Notes to Implementors

There are many things of significance to the design of MBus components scattered throughout the specification or implied by the specification. These notes are to help clarify some of these issues for implementors.

B.1. Memory Controllers

Memory controllers will probably only be slave devices and so should not need to connect MBR*, MBG* or MBB*. Memory controllers will not issue R&R acknowledgments, although Retry acknowledgments may prove useful to ECC memory controllers. Memory controllers must accommodate wrapped requests and it is recommended that for compatibility they accommodate all transfer sizes allowed for by the specification. The only signals multiplexed on MAD during the address phase of interest to most controllers are TYPE and SIZE.

To enhance compatibility, it is recommended that memory controllers accommodate Level 2 functionality. This means they should recognize all Coherent transactions. Generally this does not add complexity beyond identifying the transaction, as most Coherent transactions are simple reads and writes from the memory controllers perspective. Beyond this, Level 2 compatibility involves two significant details. First, during Coherent Read and Coherent Read and Invalidate transactions the MIH* signal may be asserted. Memory controllers interpret this signal as telling them to immediately abort the transaction. Secondly, the Coherent Invalidate Transaction must be acknowledged by memory. This means that if multiple memory controller configurations or systems with coherent bus adaptors and memory controllers are allowed, there must be a means to disable memory controllers as the source of acknowledgment. This might be through a pin or a bit in a configuration register. Also the acknowledgment should occur on A+2 or later. Memory Controllers may choose to provide a programmable cycle count for the invalidate acknowledgments in order to accommodate a wider range of modules and system configurations. There is no need for memory controllers to observe MSH*. Write and Coherent Write and Invalidate transactions are identical to Memory controllers i.e. all sizes are supported. Also, to support modules with non-standard MIH* and MSH* timing and coherent bus adaptors, memory controllers should consider providing a programmable means to vary the minimum acknowledgement timing to Coherent Read and Coherent Read and Invalidate Transactions.

A detail of memory controller design is how errors are handled, and how data is delivered in the presence of errors. MBus does not specify system details of this nature. A conservative memory design will always ensure that incorrect data is never transferred across the MBus. This may cost performance as generally it takes time to detect errors. Less conservative designs may report errors after incorrect data is delivered, either synchronously with an error acknowledgment, or asynchronously via the AERR* signal. This approach may not be acceptable to many computer vendors. Processor Modules that are using the "wrapping" feature to re-start the processor early will have no error recovery mechanism with either the late synchronous or asynchronous error reporting approach.

B.2. I/O Adapters

An MBus I/O adapter can be broken down into a Master section and a Slave section , the MBus I/O adapter slave port being the processor to I/O bus connection for programmed I/O, and the MBus I/O adapter master port being the I/O bus to memory connection for DMA. Most of the complexity is in the I/O adapter MBus master section. A generic MBus I/O master port requires an I/O MMU (probably a SPARC reference MMU or a derivative) and, possibly an I/O cache.

A central issue in I/O adapter design is I/O consistency i.e. ensuring that both I/O and processors do not obtain stale data. I/O consistency must be handled by software cache flushing in Level 1 MBus systems, as there is no Invalidate transaction for level 1 MBus. For level 2 systems, data consistency can be handled completely by hardware, or by a combination of hardware and software, depending on the sophistication of the I/O adapter MBus master interface. Complete hardware handling of data consistency can be accomplished in several ways, depending on the performance and complexity goals. The highest performance (and highest complexity) design uses an I/O cache that has control logic and dual directories similar to a Level 2 processor module. A simpler design does not use dual directories and implements consistency by using locked read modify write sequences for DMA write transfers with SIZE other than 32-bytes. This design provides efficient cache consistent transfer for 32-byte DMA transactions and less efficient cache consistent transfer for DMA transactions of less than 32-bytes.

The MBus I/O adapter slave port will generally turn MBus transactions into equivalent I/O bus transactions. A typical I/O bus might be VMEbus or SBus. I/O adapter slave ports will probably issue R&R acknowledgments to deal with slow devices and "deadlocks", and so will need the circuitry to handle R&R time-out and ID capture. I/O adapter slave ports will also probably need to observe the LOCK bit on MAD in conjunction with MBB*, and forward a LOCK indication to the "other" bus interface. MBus I/O adapter slave ports may also choose to buffer MBus write transactions (i.e. return an immediate acknowledgment to the MBus master before completing the write transaction). If they do, it is desirable that there be a means to turn off this feature, and also a means to flush the write buffers.

B.3. Reflective Memory Support

Reflective Memory operations, where memory is updated with the new data that appears on MBus during Coherent Read transactions that assert MIH*, are permitted but not required. It is recommended that caches have the ability to perform their part of Reflective transactions as an option that can be enabled when they are installed in a system where the memory supports this feature. (This implies the ability of the cache asserting MIH* during a Coherent Read transaction to ensure the cache line be marked as clean after it has supplied the data, because memory will be updated with the most recently modified data). Memory controllers, by observing MIH*, and waiting for the subsequent data, can obtain the most recent data and update memory. The memory controller design will probably need a queue, and the system designer should ensure that this queue can never overflow. i.e. the maximum rate at which reflected data is delivered to memory should not exceed the memory system's ability to absorb it. MBus has no mechanism for memory to control the arrival rate of data transactions when caches are supplying the data.

— back to back reflective writes
will overflow simple write buffer
if page miss occurs! (Hence BR/BG
mechanism on multi-chip module)

B.4. Second Level Cache Issues

Second level caches are implicitly supported by MBus transactions. There are several kinds of systems with second level caches envisaged using MBus. The most generic system uses the Level 2 protocols and can support several CPU modules at the first level sharing a common second level cache as shown in Figure 19. This requires a complex second level cache which has two competing ports and must resolve deadlock situations. There are several complexities that these designs must deal with. . R&R on Write (write-back) and R&R on Coherent Invalidate both produce a similar problem, in that a temporary state is created where there is no "owner". Memory is assumed the default owner, which causes a failure of consistency. A uniform solution is for interfaces that issue R&R on coherent transactions to assume temporary ownership of the line until the R&R is resolved. This circuitry would detect accesses to the line the R&R was outstanding against, and R&R those accesses. Another problem concerns Coherent Read and Invalidate. Should a Coherent Read and Invalidate transaction result in MIH* being asserted from a cache on the same bus, the second level cache must capture the Coherent Invalidate in a queue and forward it to other caches when appropriate.

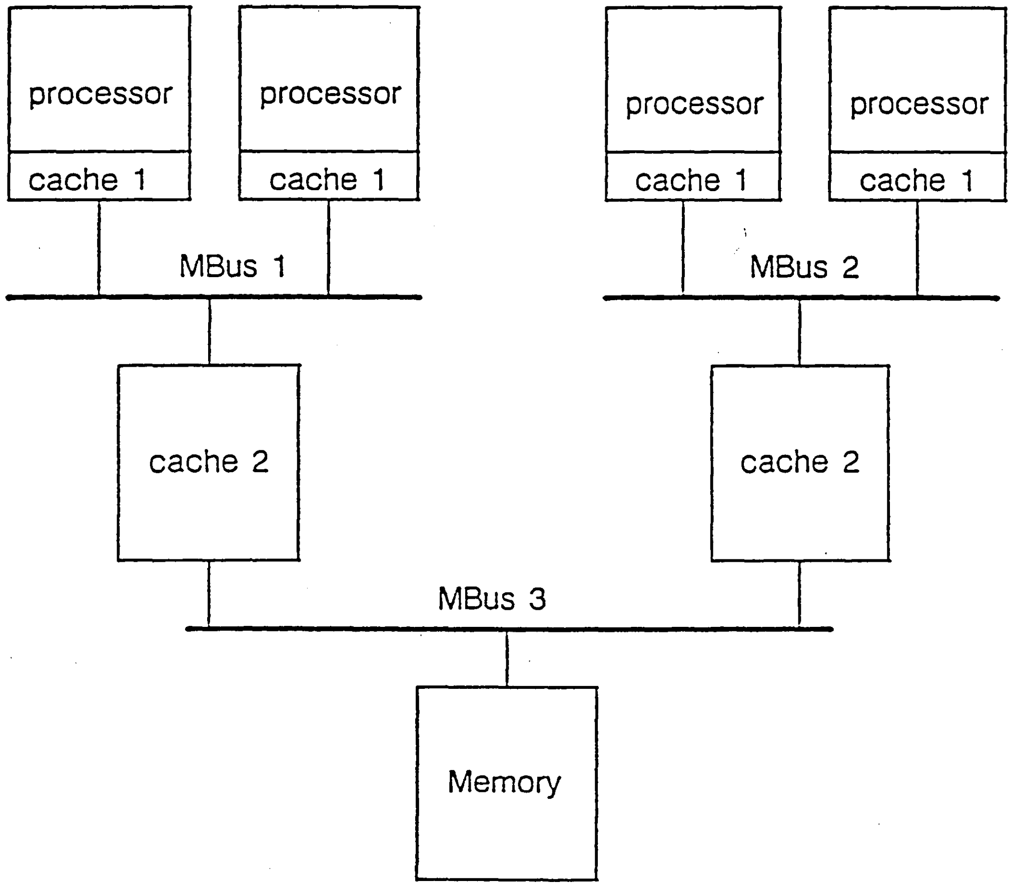


Figure 19 - Generic Multi-Level Cache System

A simpler use of second level caches has one processor per second level cache and does not use the complete level 2 protocol. This type of system is shown in Figure 20. It requires MBus CPU modules that can support a "write through" cache mode and accept an Invalidate transaction. With this level of support it is possible to ensure that lines that are in the First level cache are always present in the second level cache (inclusion), and so all logic associated with coherence will be included at the second level cache. The Invalidate exceeds Level 1 requirements and the write through capability is additional to the minimum requirements of Level 2.This design point is useful where the size of the first level cache is constrained and the system performance would otherwise be limited by the resulting miss rates and a saturated MBus.

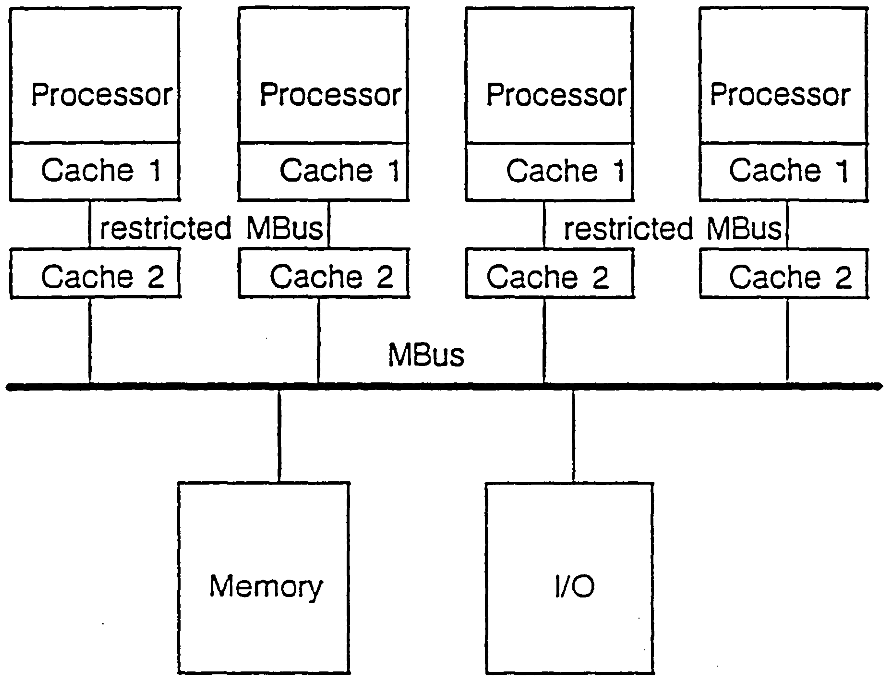


Figure 20 - Simple Second-Level Cache System

Another simple use of a second level cache is shown in Figure 21. Because MBus is a circuit switched bus, its ultimate performance is limited by memory latency. When designing large memory systems, it is difficult to achieve low latency. A solution is to use a large second level cache in front of memory that essentially acts as a buffer to memory. This allows latencies close to the minimum MBus latency to be achieved.

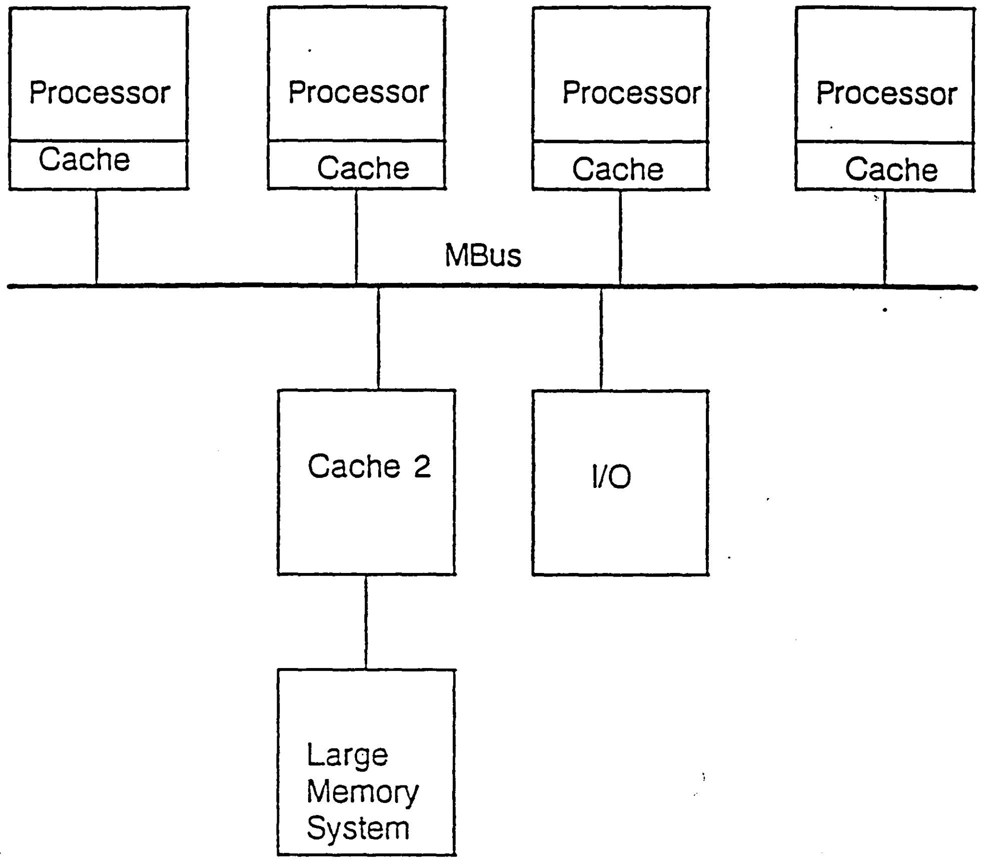


Figure 21 – Simple Second-Level Cache Example 2

B.5. Timing of MSH* and MIH*

MBus timing is specified with MSH* and or MIH* assertion on A+2 i.e. 2 cycles after MAS* is received. This assumes a "dual directory" structure where there is a dedicated duplicated set of tags as part of the MBus "snoop" logic, and operation synchronous to the MBus clock. If it is desired not to have a dual directory or there is a need to synchronize from a clock other than the MBus clock, then the A+2 timing need not be met as long as some restrictions apply. The basic restriction is that memory acknowledgements should never occur before MIH*. This restriction limits the minimum latency of MBus memory systems to the MIH* timing. Systems that wish to accommodate modules with non-standard MIH* timing need to guarantee minimum memory latencies relative to these modules via a programmable minimum memory CR or CRI latency mechanism implemented in the memory controller(s).

In general, modules with variable MSH*/MIH* timing will also need to restrict the maximum rate at which Coherent Invalidates might arrive. This is accomplished via a programmable delay on acknowledgements to CI transactions, implemented in the memory controller(s).

Supporting variable timing condenses to a few rules. If you are a cache performing a CR or CRI, then source MIH* and MSH* for a single cycle as soon as you can and at the same time. Specify the worst case MIH*/MSH* timing from MAS* (e.g. A+7) in order that system designers know what minimum read latency to program memory controllers with. If you are a snooping cache(either intervening or non-intervening), then observe MIH*/MSH* in the interval from MAS*+2, to when you observe your first MRDY*. If you are memory, do not issue MRDY* (or any acknowledgement) to CR and CRI transactions before you can observe MIH* from the slowest module in a system.

B.6. Compatibility Issues

A number of compatibility issues are covered in the specification. Here are just a few compatibility issues which might easily be overlooked:

- **Wrapping:** Generally, compatibility on wrapping is an issue between a module and memory or I/O devices. Memory should support wrapped requests and this resolves most compatibility issues. An exception is Level 2 Processor Modules that do not issue wrapped requests and do not wish to deliver data to wrapped Coherent Read transactions when they assert MIH*. Modules that issue wrapped Coherent Read transactions will clearly not mix in a system with modules of this nature.

- **MSH* and MIH* Timing:** For modules that generate MIH* and MSH* on other than A+2, Memory must have a minimum latency greater than or equal to the MIH* and MSH* timing. Modules with A+2 timing should mix with modules with variable timing.

- **Virtual Address bits in MAD:** Modules that do not generate the Virtual Address "super-set" bits on Coherent transactions cannot mix with modules that use direct mapped virtual addressed caches.