Chapter 2
Data Transfers

VMEbus data is transferred over the Data Transfer Bus (DTB). Masters use the DTB to move data to and from slaves, interrupters use it to pass STATUS/ID words to handlers, and location monitors use it to initiate local activity in response to certain types of cycles.

The Data Transfer Bus was non-multiplexed under revisions A, B, C, C.1, IEC 821 and IEEE 1014-1987 of the VMEbus specification. However, the proposed VME64 bus specification allows multiplexed 64-bit address and data transfers. This enhancement doubles the theoretical bandwidth of the bus.

2.1 VME64

During the initial development stage of VMEbus in 1981, 16-bit microprocessors (such as the MC68000) with 24-bit addressing had only just hit the market, and eight bit peripheral chips were the state of the art. 32-bit processors had not been developed. The original bus architects, however, had the vision to define an architecture capable of handling 8, 16 and 32 bit data widths, as well as 16, 24 and 32-bit addressing modes.

However, as the speed of microprocessors, memory and I/O have increased, the theoretical bandwidth of VMEbus has become a significant factor in overall system capability. The theoretical VMEbus data transfer rate of 40 Mbytes per second has become a reality, and in some applications is a severe limitation of the bus. Typical applications that require higher bus bandwidth include FDDI disk controllers, image processing, bus-to-bus links and closed loop machine controls.

Some users feel that 64-bit data transfers are not necessary, especially if 16 or 32-bit CPUs are used. However, 64-bit data transfers are very useful in message passing architectures. In these systems message packets are assembled on each CPU (or I/O) module, and passed in a burst mode across the bus. The 64-bit bus is especially effective when large message packets are used.

To keep up with the need for ever faster transfer rates, a variety of schemes were proposed. These ranged from changes in setup-and-hold timing, to secondary buses to off load main VMEbus traffic. While these techniques can improve system performance, they either don't conform to the VMEbus standard, or they create radical changes to system architecture.

In 1988 Performance Technologies (East Rochester, N.Y. - USA) developed a superset to VMEbus which almost doubled the theoretical bandwidth of the bus. The Performance Technologies approach modified the block transfer cycle so that the address lines (unused after the first data transfer) carried an additional 32-bits of data. This extended the data transfer width from 32 to 64-bits, and has since become popularly known as VME64.

The attractiveness of the VME64 operating mode is its simplicity and compatibility with previous VMEbus specifications. The VME64 block transfer cycles (D64BLT) were defined using the identical timing and operating parameters of the similar 32-bit block transfer cycle (D32BLT). The physical implementation was straightforward and required no special backplane modifications.

VME64 has since been adopted in a proposed revision to the VMEbus specification. As part of the revision, the VITA technical committee included other enhancements to the specification such as several types of 64-bit addressing modes, standard CSR register scheme, auto slave ID and auto system controller functions.

2.2 Bus Cycles

The Data Transfer Bus allows several types of bus cycles. These include the read/write, block transfer, read-modify-write and address-only cycles. Chapter 4 covers an additional cycle called the interrupt acknowledge cycle. Not all bus modules are compatible with all types of bus cycles. When evaluating or designing VMEbus modules, be sure that slaves are compatible with the cycles generated by the masters. For example, a master may generate a read-modify-write cycle, but not all slaves respond to it.

2.2.1 Read/Write Cycle

The read/write cycle is the most commonly used of all the bus cycles. It is used to pass data between masters and a slaves 8, 16, 24 or 32 bits at a time.

2.2.1.1 Addressing

A master addresses a slave during every read/write cycle. This is done with address lines A01-A31, a six bit address modifier code AM0-AM5, and two control signals IACK* and LWORD*. All of these signals are qualified by the falling edge of address strobe AS*. In addition, the two data strobes DS0* and DS1* determine which byte location within a four byte group data should be accessed. There is no address line A00.

An alternative design approach permits addresses to be qualified on the falling edge of the data strobes DS0* / DS1* rather than address strobe AS*. This simplifies the design of slave boards.

2.2.1.2 Address Sizing

There are four possible address widths as shown in Table 2-1. These are called *short I/O*, *standard*, *extended* and *long* addressing modes, and correspond to 16, 24, 32 and 64-bits respectively. The size can be changed on every bus cycle. The most obvious advantage to this scheme is to allow older microprocessors to share the bus with newer ones. For example, a CPU module with a 68000 microprocessor capable of generating 24-bit addresses may share the bus with a 32-bit 68020.

Table 2-1. Dynamic address sizes.

Address Modifier Type	Address Bits	Active Address Lines	Mnemonic
Short I/O	16	A01 - A15	A16
Standard	24	A01 - A23	A24
Extended	32	A01 - A31	A32
Long (†)	64	A01 - A31 D00 - D31	A64

(†) Denotes proposed VMEbus enhancement

A64 addressing was first permitted under the proposed VME64 bus specification. The A64 concept arose from the development of the D64 block transfer cycle.

2.2.1.3 Address Modifier Code

A six bit address modifier code AM0-AM5 accompany each address. The address modifier is decoded as shown in Table 2-2. During a bus cycle the slave monitors the address modifier to determine which address lines to decode. Short I/O addresses are decoded from A01-A15, standard addresses from A01-A23, extended addresses from A01-A31 and long addresses from A01-A31, LWORD* and D00-D31. The address lines are routed as shown in Table 2-3.

The address modifier also indicates the type of bus cycle. It discriminates between instruction fetches, data read/write cycles, block transfer cycles and whether the cycle is generated by supervisory or non-privileged programs.

The function of the IACK* signal is closely associated with the address modifier codes. IACK* is asserted by *interrupt handlers* to show that the current cycle is an interrupt acknowledge cycle, and negated by *masters* to show that it is a data transfer cycle. It is useful to think of IACK* as a seventh address modifier bit. When IACK* is asserted, AM0-AM5 is ignored by slaves.

Table 2-2. Address modifier codes.

Address Modifier (Hex)	IACK*	No. Address Bits	Transfer Type
3F	1	24	Standard supervisory block transfer
3E	1	24	Standard supervisory program access
3D	1	24	Standard supervisory data access
3C (†)	1	24	Standard supervisory 64-bit block transfer
3B	1	24	Standard non-privileged block transfer
3A	1	24	Standard non-privileged program access
39	1	24	Standard non-privileged data access
38 (†)	1	24	Standard non-privileged 64-bit block transfer
2D	1	16	Short supervisory access
29	1	16	Short non-privileged access
10 - 1F	1	--	User defined
0F	1	32	Extended supervisory block transfer
0E	1	32	Extended supervisory program access
0D	1	32	Extended supervisory data access
0C (†)	1	32	Extended supervisory 64-bit block transfer
0B	1	32	Extended non-privileged block transfer
0A	1	32	Extended non-privileged program access
09	1	32	Extended non-privileged data access
08 (†)	1	32	Extended non-privileged 64-bit block transfer
07 (†)	1	64	Long supervisory block transfer
06 (†)	1	64	Long supervisory program access
05 (†)	1	64	Long supervisory data access
04 (†)	1	64	Long supervisory 64-bit block transfer
03 (†)	1	64	Long non-privileged block transfer
02 (†)	1	64	Long non-privileged program access
01 (†)	1	64	Long non-privileged data access
00 (†)	1	64	Long non-privileged 64-bit block transfer
XX	0	3	Interrupt acknowledge cycle (uses A01-A03)

Note: All codes other than those shown are reserved for future use.
Don't care state = (XX), undefined = (-), low level signal = (0), high = (1),
(†) denotes proposed VMEbus enhancement.

Table 2-3 Address routing during various addressing modes.

Active Portion of Data Transfer Bus - Address Routing								
D24-D31	D16-D23	D08-D15	D00-D07	A24-A31	A16-A23	A04-A15	A01-A03	Address Modifier Codes (Hex)
A63 — During address broadcast portion of bus cycle only — A01								Long (64-bit) (†) 00 - 07
				A31 — A01				Extended (32-bit) 08 - 0F
					A23 — A01			Standard (24-bit) 38 - 3F
						A15 — A01		Short I/O (16-bit) 29, 2D
							A03-A01	Interrupt Acknowledge

= Unused portion of address bus
= Used to pass data
(†) Proposed VMEbus enhancement

Address modifier codes also simplify the design of many VMEbus modules. Modules can be as simple or complex as the application requires. Slaves, such as a serial I/O modules, require only several bytes of address space and can use short I/O addressing. This reduces part count by decreasing the number of comparators and control logic. This lowers the board cost and conserves space. More complex modules such as memory or graphic controllers require standard, extended or long addresses because of large memory requirements.

The address modifier also makes single (3U) and double height (6U) modules compatible. Single height modules use only the P1 connector, they can only monitor address lines A01-A23. This limits these modules to the short I/O and standard cycles. Double height modules, however, can monitor an additional eight or sixteen address lines on the P2 connector, and can perform 32 and 64-bit address transfers. Without the address modifier, the simple P2 expansion bus would be awkward.

2.2.1.4 Address Mnemonics

A series of mnemonics have been developed to help users select compatible modules. A series of standard address mnemonics are shown in Table 2-4. A16, A24, A32 and A64 mnemonics correspond to the number of address lines used in a transfer. For example an A16 slave will respond to bus cycles generated by an A16 master.

When selecting modules with master capability, make sure that they will generate all the cycles required by the slaves. This is not guaranteed by all versions of the VMEbus specification.

The IEEE-1014-1987 VMEbus specification requires that A32 masters generate A24 and A16 cycles. Similarly, A24 masters must also generate A16 cycles. Slaves do not have any such requirements.

The A64 addressing scheme was first introduced in the proposed VME64 version of the VMEbus specification. 64-bit addresses were not allowed under earlier versions. A64 masters must include the A32 and A24 capabilities, but not necessarily the A16.

2.2.1.5 Typical Read/Write Cycle

During a typical read/write cycle the master first addresses a slave, and then transfers data. The data is transferred using data lines D00-D31, WRITE*, data transfer acknowledge (DTACK*) and bus error (BERR*). The data lines and the WRITE* signal are qualified using data strobes DS0* and DS1*.

The timing waveform for a typical read cycle is shown in Figure 2-1. Here a master addresses a slave by driving A01-A31, AM0-AM5, IACK* and LWORD*. These are qualified by the falling edge of AS*. The master also negates WRITE* and asserts data strobes DS0* and/or DS1*. The slave decodes the address, places data onto D00-D31 and asserts data transfer acknowledge DTACK*. When the master has latched the data, it informs the slave by negating the data strobe(s). The slave then negates DTACK* and the cycle is terminated.

Table 2-4. Mnemonics that describe the various addressing modes.

Mnemonic	Description
A16	Generates (master) or accepts (slave, location monitor) bus cycles with short I/O (16-bit) addresses.
A24	Generates (master) or accepts (slave, location monitor) bus cycles with standard (24-bit) addresses. A24 masters must also be A16 compatible.
A32	Generates (master) or accepts (slave, location monitor) bus cycles with extended (32-bit) addresses. A32 masters must also be A16 and A24 compatible.
A64 (†)	Generates (master) or accepts (slave, location monitor) bus cycles with long (64-bit) addresses. A64 masters must also be A24 and A32 compatible.

(†) Proposed VMEbus enhancement

A write cycle is similar to a read cycle as Figure 2-2 shows. The main difference is that data is placed onto the data lines before the data strobes are asserted. Once the slave asserts DTACK* or BERR*, the master *can* immediately negate WRITE* and change the data lines. For this reason a slave must latch the data before it asserts DTACK*.

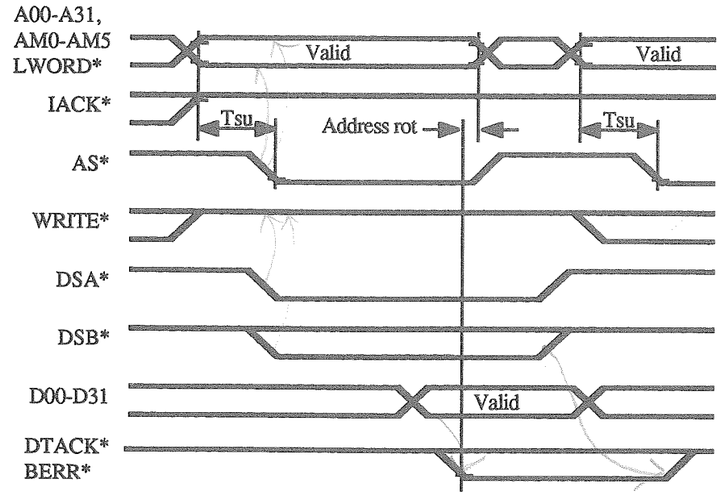

Figure 2-1. Read cycle with address pipelining.

2.2.1.6 Data Strobes

The data strobes DS0* and DS1* serve a dual function. As a level sensitive signal they select which bytes are accessed. As an edge sensitive signal they are used to qualify data.

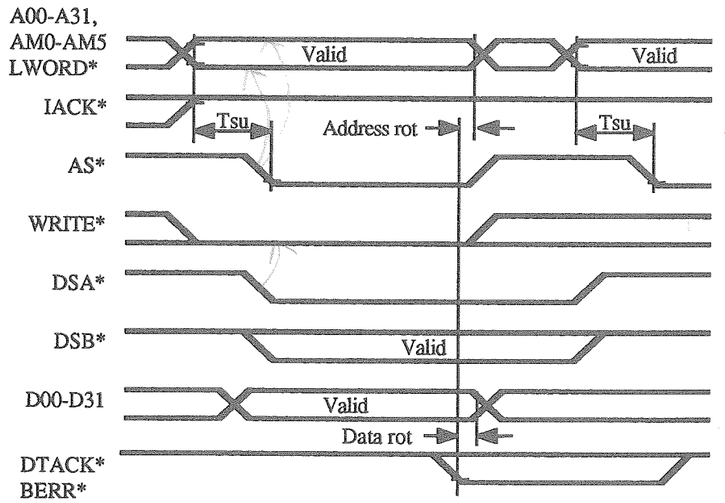

Figure 2-2. Write cycle with address pipelining.

The VMEbus specification does not use the terms DS0* and DS1* in its timing diagrams. Instead it refers to DSA* and DSB*. This notation was introduced to prevent confusion in cases where bus skew (propagation delay) causes one data strobe to fall or rise before the other.

Propagation delay times through transceivers at the master may also cause these signals to be sent at slightly different times.

2.2.1.7 Cycle Termination by DTACK*, BERR*

Slaves terminate all bus cycles by asserting DTACK* or BERR*. DTACK* is the normal way to end the cycle. During read cycles the slave asserts DTACK* after driving the data bus, and during write cycles the slave asserts it after it has latched the data.

BERR* can be asserted by a slave or a bus timer. When it is asserted by a slave it indicates that an error has occurred during the cycle. The VMEbus specification does not say what may have caused the error nor what should be done in response to it. For example, a memory module may assert BERR* in response to a parity error.

The bus timer asserts BERR* if the bus has locked up. Bus lock-ups can be caused either by system failures or out of range addressing. One popular use for this feature is for self configuration of the system. During power-up initialization the number of cards can be counted by a master in the system, as well as the amount of memory installed.

VMEbus uses a fully interlocked handshaking mechanism with data strobes DS0* and DS1*, DTACK* and BERR*. At the beginning of a cycle a master must not assert either data strobe until DTACK* and BERR* (from the last bus cycle) have been negated. Failing to do so may cause data to be corrupted on the current or the previous cycle.

Care must be taken when evaluating or designing masters that use the 680XX microprocessor family. These microprocessors don't use a fully interlocking bus cycle, and external circuitry must be provided to make them totally compliant.

2.2.1.8 Cycle Termination by RETRY*

The VME64 Specification designated the (former) RESERVED pin on the P2 connector (P2-B3) as a RETRY* signal. RETRY*, together with BERR*, can be driven low by slaves to indicate that a requested data transfer cannot take place, but should be attempted again by the master in a future bus cycle. The retry provision was made to prevent deadlock (deadly embrace) conditions in bus-to-bus links and secondary buses.

BERR* is used in conjunction with RETRY* so that upward compatibility from earlier designs can be maintained. For example Revision C.1 VMEbus masters, which do not monitor the RETRY* signals, would simply terminate the cycle as if a bus error took place. VME64 masters would terminate the cycle, but would re-attempt the cycle at a future time. Without the concurrent assertion of RETRY* and BERR*, the Revision C.1 master would not terminate the cycle and the bus could potentially get locked-up.

After receiving RETRY* low, the master must relinquish control of the bus. This feature prevents the deadlock conditions in bus-to-bus links and secondary buses. After relinquishing the bus, however, the master should wait a short time before attempting to access the bus again. The amount of time that a master waits before requesting the bus depends on the bus arbitration method and system architecture. By waiting, the master allows other processors to access the bus.

The RETRY* signal pin is fully compliant with backplanes designed under previous bus specifications (before VME64). Earlier specifications required that the RESERVED pin be bused

and terminated, so the RETRY* function shouldn't be a problem. Unfortunately, some backplane manufacturers did not bus and terminate this pin, even though they should have. System integrators should verify that pin P2-B3 is bussed and terminated before specifying or purchasing a backplane.

2.2.1.9 Address Pipelining

Some microprocessors use address pipelining to speed up data transfers. During address pipelining the bus master broadcasts the address of the next bus cycle before the current cycle is completed. As shown in Figures 2-1 and 2-2, immediately after the slave asserts DTACK* the master may negate AS* and place a new address on the bus. Data transfers can be speeded up by overlapping the address broadcast with the previous cycle. Slaves must be designed to function properly in the presence of an address pipeline cycles.

The removal of a valid address before DTACK* or BERR* are negated is sometimes called address rot. Similarly, changing the data lines after DTACK* or BERR* are asserted during a write cycle is called data rot.

When designing or evaluating slave interfaces, be sure they latch all addresses and address modifiers before asserting DTACK* or BERR*. The address is often latched on the falling edge of either data strobe. Failing to do so may cause the slave to change its data lines before the end of a read cycle because its on-board address has changed. During a write cycle the slave's internal timing may be disrupted.

When evaluating or designing slaves that do not utilize address pipelining (all slaves must function properly in the presence of these cycles), care should be taken when latching the address on the falling edge of AS*. On read cycles, the master can negate and re-assert AS* immediately after a slave asserts DTACK* or BERR*. If the address is latched on the falling edge of AS*, the slave could be presented with a new address before it negates DTACK* or BERR*. This could corrupt the data. On slaves which do not implicitly participate in address pipelining cycles, it is better to latch the addresses on the falling edge of the data strobes. The data strobes can be 'or'ed together and used to latch the addresses.

When evaluating or designing modules that support block transfer cycles (discussed below), special care should be taken to insure that address pipelining will work. During the block transfer cycle the master may change A01-A31 and LWORD* after the first falling edge of DTACK*, but the address modifier code AM0-AM5 must remain stable until the last falling edge of DTACK*.

Address pipelining also reduces the overhead needed to change bus masters (see Chapter 3 on Multiprocessing). During bus arbitration, a new master can assume ownership of the data transfer bus (assert BBSY*) after the current master negates address strobe AS*. Arbitration occurs in parallel to the current master's final data transfer cycle.

Special consideration to address pipelining must be given in A64 and D64BLK cycles. These cycles are permitted under the proposed VME64 Specification. For example, during a D64BLK read cycle the address bus is used to transfer data. After the last falling edge of DTACK* or BERR*, the master must assert AS* until DTACK* or BERR* has been negated. The prevents another master from driving the address bus (and the data on it) before the current master has latched the data.

2.2.1.10 Data Sizing

The data bus is dynamically configured (just like the address bus). Data transfers of 8, 16, 24, 32 and 64-bits can be made without any software overhead whatsoever. This makes VMEbus products compatible over wide ranges of technology.

Data bus sizing is achieved by splitting the data lines into four byte-wide banks: D00-D07, D08-D15, D16-D23 and D24-D31. The master signals the type and size of transfer with strobes DS0* and DS1*, address line A01 and LWORD*.

VMEbus uses a BYTE(n) convention to specify how data is stored in memory, where (n) is the address offset from an even 32-bit boundary. Table 2-5 shows this convention.

Table 2-5. Data organization in memory.

Categories of Byte Locations			
4-Byte Group		8-Byte Group (†)	
Operand	Byte Address (Binary)	Operand	Byte Address (Binary)
BYTE(0)	XXXX....XX00	BYTE(0)	XXXX....X000
BYTE(1)	XXXX....XX01	BYTE(1)	XXXX....X001
BYTE(2)	XXXX....XX10	BYTE(2)	XXXX....X010
BYTE(3)	XXXX....XX11	BYTE(3)	XXXX....X011
		BYTE(4)	XXXX....X100
		BYTE(5)	XXXX....X101
		BYTE(6)	XXXX....X110
		BYTE(7)	XXXX....X111
Key: (X) Don't care (†) Proposed VMEbus enhancement			

During a data transfer the master asserts DS0*, DS1*, A01 and LWORD* depending upon where it expects to read or write data. The level of these signals and the associated data paths are shown in Table 2-6.

Dynamic data sizing allows older and newer technologies to work together. For example, a CPU module with a 68000 microprocessor (8 or 16-bit data path) can share VMEbus with a 68020 based CPU (8, 16, 24 or 32 bit data path).

VMEbus offers five styles of bus interface. These are classified using the mnemonics D08(O), D08(EO), D16, D32 and D64.

The D08(O) slave transfers data on lines D00-D07. Transfers of eight bits can be made at odd addresses (e.g. $0002FF01 or $0002FF03). D08(O) masters are not specifically allowed under the VMEbus specification since these are simply a subset of the D08(EO) master. An example of a D08(O) slave would be an 8-bit serial I/O module.

The D08(EO) master or slave allows bytes to be transferred at even or odd addresses. Transfers to or from these modules must be done eight bits at a time, and only one data strobe may go low at any time. An example of a master with this interface would be a CPU module with an 8-bit processor (such as an MC68008).

The D16 master or slave allows 16-bit data transfers over data lines D00-D16. The D32 master or slave allows 24 or 32-bit transfers over data lines D00-D31. D16 or D32 modules that locate data at other than two or four byte boundaries are said to support unaligned transfers (see below).

The D64 interface monitors or drives data lines D00-D31 as well as address lines A01-A31 and LWORD*. This interface was first permitted under the proposed VME64 bus Specification. The specification also requires that D64 masters, slaves and location monitors include the D08(EO), D16 and D32 capabilities.

The IEEE-1014-1987 version of the bus specification requires that D16 masters, slaves and location monitors include the D08(EO) capability. It also requires that D32 masters, slaves and location monitors include the D16 and D08(EO) capabilities. These were both optional under the Revision B, C, C.1 and IEC 821 versions of the bus specification.

The VMEbus specification does not provide slaves with the ability to acknowledge data port size during a transfer. Some popular microprocessors (like the 68020 and 68030) require slaves to do so. This requires the use of memory mapping or mode bit techniques.

In memory mapped systems the data port size is selected by the address. For example, a D16:D32 CPU module may configure its bus interface as a D16 master during bus accesses between $00000000 and $00FFFFFF, and 32-bits between $01000000 and $01FFFFFF.

Selection of a data port size may also be accomplished with a mode bit. The CPU sets a bit indicating the type of required access. For example, the bit could be set when the master generates a D32 cycle, and cleared for a D16. The advantage of this method is simplicity of design. The disadvantage is that system software needs to continuously toggle the mode bit.

2.2.1.11 Unaligned Data Transfers

Unaligned data transfers are allowed under the VMEbus specification. VMEbus modules can place two or four bytes of data at other than two or four byte boundaries. These are called unaligned transfers. Unaligned transfers can speed up a VMEbus system by allowing 32-bits of data to be transferred at odd addresses in two bus cycles instead of three.

When a master reads or writes data it can do so in a variety of ways. For example, consider case B of Figure 2-3. Here a four byte transfer takes place at an unaligned boundary. The master can transfer the data using one of two methods. By one method the master transfers a Single Byte(1), a Double Byte(2-3) and a Single Byte(0). This means that the whole transfer requires three bus cycles. Using a second method the master performs an Unaligned Byte(1-3) and a Single Byte(0) transfer. This second method takes only two bus cycles. Unaligned transfers can substantially reduce the number of bus cycles.

The VMEbus specification does not stipulate the order in which data is transferred to or from memory. In the example above, the Single Byte(0) transfer could take place before or after the Unaligned Byte(1-3) transfer. This can be important in multiprocessor systems where another master may be granted a bus cycle between two consecutive transfers. Software engineers should design flags accordingly.

Figure 2-4 shows four ways that 16-bit words may be stored in memory.

Table 2-6. Active portions of bus during data transfers.

Active Portion of Data Transfer Bus - Byte Lanes and Byte Routing								Control Signal Levels
A24-A31	A16-A23	A08-A15	LWORD* A01-A07	D24-D31	D16-D23	D08-D15	D00-D07	DS1* DS0* A01 LWORD* A02
(†) EIGHT BYTE(0-7) BYTE(0)	BYTE(1)	BYTE(2)	BYTE(3)	BYTE(4)	BYTE(5)	BYTE(6)	BYTE(7)	(Address Broadcast Phase Only) 0 0 0 0 0
				QUAD BYTE(0-4) BYTE(0)	BYTE(1)	BYTE(2)	BYTE(3)	0 0 0 0 X
				UNALIGNED(0-2) BYTE(0)	BYTE(1)	BYTE(2)		0 1 0 0 X
					UNALIGNED(1-3) BYTE(1)	BYTE(2)	BYTE(3)	1 0 0 0 X
					UNALIGNED(1-2) BYTE(1)	BYTE(2)		0 0 1 0 X
						DBL BYTE(2-3) BYTE(2)	BYTE(3)	0 0 1 1 X
						DBL BYTE(0-1) BYTE(0)	BYTE(1)	0 0 0 1 X
							SINGLE BYTE(3)	1 0 1 1 X
						SINGLE BYTE(2)		0 1 1 1 X
							SINGLE BYTE(1)	1 0 0 1 X
						SINGLE BYTE(0)		0 1 0 1 X
ILLEGAL								1 0 1 0 X 0 1 1 0 X

= Unused portion of data bus
= Used to pass address
(X) = Don't care
(†) Proposed VMEbus enhancement

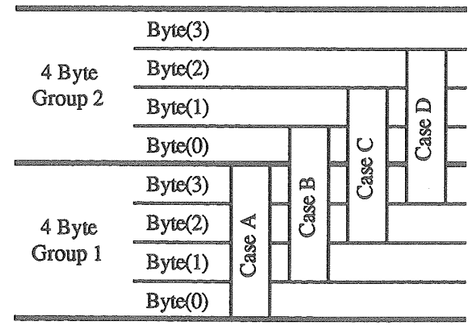

Figure 2-3. Four ways that 32-bits of data can be saved in memory.

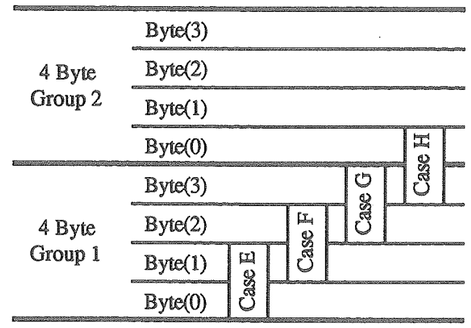

Figure 2-4. Four ways that 16-bits of data can be saved in memory.

A special UAT mnemonic specifies whether an unaligned transfer can be generated by a master, accepted by a slave or monitored by a location monitor.

The IEEE-1014-1987 version of the bus specification requires that D32 slaves and location monitors must accept unaligned (UAT) data transfers. D32 masters are not required to generate these cycles, however. The UAT function was optional for slaves and location monitors under earlier versions of the bus specification.

There is no provision for D64 unaligned data transfers.

Some software compilers can be set to generate code only on even boundaries. This reduces the number of unaligned transfers and therefore speeds up the system. Some compilers with this feature will not prevent data transfers at unaligned boundaries, only instructions.

2.2.2 Block Transfer Cycle

The block transfer cycle moves blocks of data at high speed across the bus. The block transfer cycle is sometimes called the *burst mode*.

Consider a CPU module fetching program instructions: if an instruction is read at address $100, it will probably read the next one at $102 or $104. Some computers take advantage of this by 'thinking ahead' and reading data from the next few bytes of memory while the CPU is busy decoding the current instruction. Circuits which do this are often called instruction prefetch or pipelining circuits. Special dynamic memories, called page mode and nibble mode memories, are sometimes used to simplify instruction prefetch. These memories can be placed into a high speed 'dump' mode which allow fast burst transfers. In dump mode these memories are often double or triple the speed of normal memories.

When software message passing schemes are used, the messages are often assembled on a CPU and 'burst' across the bus. The block transfer cycle can streamline message passing architectures.

In multiprocessing systems the block transfer mode can be used to transfer data in small bursts, reducing the overhead of bus arbitration. Disk intensive applications use the block transfer cycle to move data at high speed between CPUs and disk controllers.

The block transfer cycle is faster than read/write cycles because the master presents an address only once during the cycle. The extra overhead of computing and changing addresses does not take place. The timing diagram for a block transfer cycle is shown in Figure 2-5.

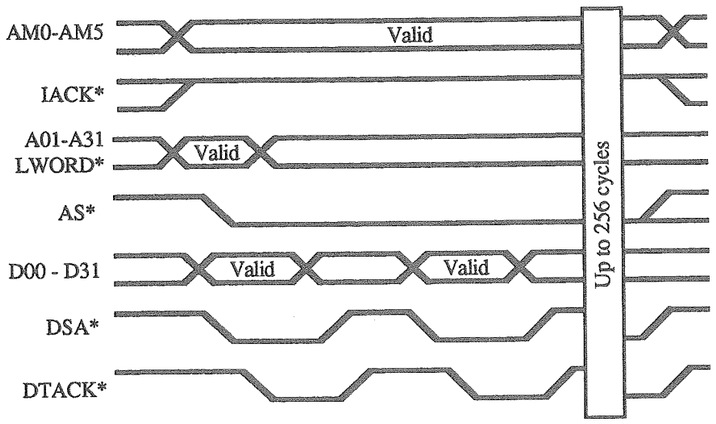

Figure 2-5. Block transfer cycle (write).

During the cycle both an address and a block transfer address modifier is presented by the master to the slave. Once the slave is addressed, multiple bytes of data can be read or written (in ascending order) by toggling the data strobes. Block transfer counters on the slave automatically increment their on-board addresses. This relieves the master from computing and changing the address during every cycle.

The ability to generate or accept a block transfer cycle is optional. To prevent incompatibility between modules, a mnemonic called BLT is used. If a master can generate a block transfer cycle it is called a BLT master. If a slave can accept a block transfer cycle it is called a BLT slave. When integrating a VMEbus system make sure that compatible modules are used.

The BLT mnemonic is not to be confused with the bacon, lettuce and tomato cycle (which is not supported by VMEbus).

2.2.2.1 Block Transfer Cycles Under Early VMEbus Revisions

To reduce the complexity of block transfer slaves, a rule was introduced in the revision C VMEbus specification which forbids block transfers from crossing 256 byte boundaries. This rule was also included in the IEEE 1014-1987 version of the bus specification.

This provision solved several problems that existed with the block transfer cycle. It prevented board-to-board crossings during a cycle, it reduced the address counter requirements to eight bits, and allowed the use of commonly available nibble or page mode memories (many standard memories require that block transfers don't cross 256 byte boundaries). If it must cross the boundary the master can stop the cycle, re-submit a new address, and do another block transfer.

The 256 byte rule also prevents a master from hogging the bus with large numbers of cycles. Since VMEbus arbitration cannot take place with AS* asserted (see Chapter 3) bus ownership cannot change during a block transfer cycle.

2.2.2.2 Block Transfer Cycles Under VME64

Several 64-bit block transfer cycles are included in the VME64 bus enhancements. All are 100% upward compatible with modules designed to previous VMEbus specifications. New timing parameters are 'inside' the cycle and will not affect modules which do not decode the address modifier.

Two basic block transfer cycles were added under VME64: SSBLT (source-synchronized block transfer cycle) and MBLT (multiplexed block transfer cycle).

At press time (September, 1992) the SSBLT cycle is being defined. SSBLT is a 64-bit multiplexed data transfer cycle with special handshaking provisions. Data transfer occurs at the edge of every data strobe. DTACK* acknowledgements are not required by the slave. The normal VMEbus handshaking protocols are abandoned, and the master synchronized every transfer: hence the name source-synchronized block transfer cycle. SSBLT transfer rates should be between 100 and 160 Mbytes/second.

A second block transfer cycle is the MBLT. This cycle is also referred to as a D64BLT cycle. The D64BLT cycle is a multiplexed version of the D32BLT. As Figures 2-6 and 2-7 show, the major difference is that the first transfer of a D64BLT cycle transfers an address, but not data. The second and subsequent transfers of a D64BLT cycle are identical to D32BLT except that the address lines are used in conjunction with the data lines to transfer 64-bits of data with each data strobe (DS0* & DS1*). D64BLT always transfers eight bytes per transfer to a double longword aligned address. Figure 2-8 shows a D64BLT bus cycle.

During 64-bit data transfers, the LWORD* pin is used for data bit D32. This is because there are only 31 address lines A01-A31. Timing difficulties do not occur because the LWORD* timing relationship is identical to the address lines.

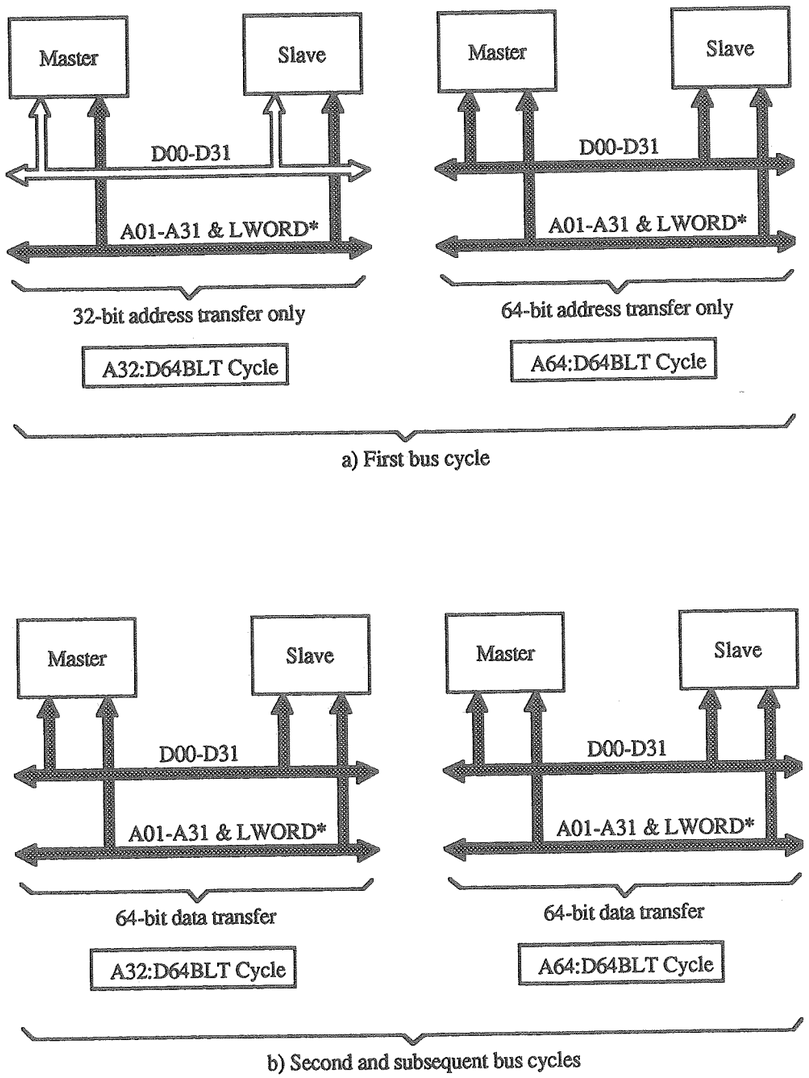

Figure 2-6. Comparison between A32:D64BLT and A64:D64BLT cycles.

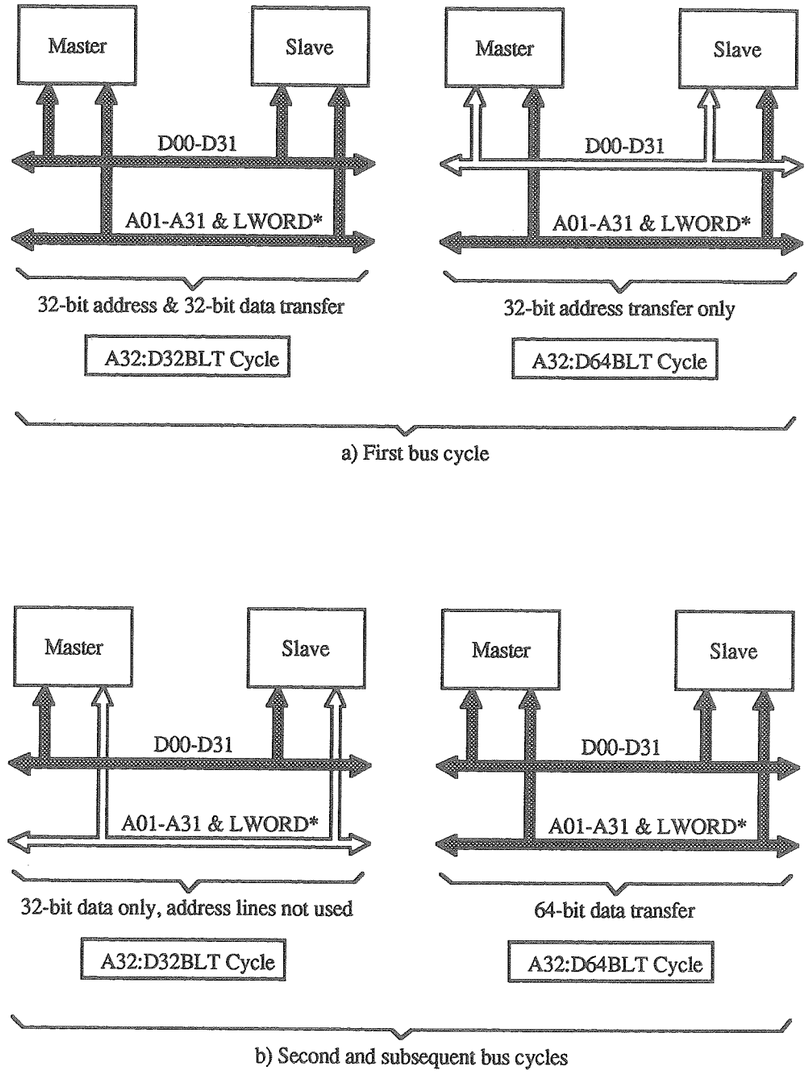

Figure 2-7. Comparison between A32:D32BLT and A32:D64BLT cycles.

Specific changes in VME64 include:

- The D64BLT cycle is differentiated from a D32BLT cycle by the inclusion of address modifier codes that were previously reserved. Refer to Table 2-2 for a listing of the new address modifier codes.
- D64BLT also specifies the order in which data is transferred.
- Specifications of the rules and timing parameters that govern the switching of address lines over to data lines. This is of particular concern for read cycles where a master must relinquish control of the address lines to the slave. For more information, refer to the section of this book on address pipelining.
- Specification of the maximum number of transfers allowed for a D64BLT cycle.

D64BLT cycles cannot cross 2K-byte boundaries. The purpose of this rule is the same for the 256-byte rule for non-multiplexed D32BLT, D16BLT and D08(EO)BLT transfers.

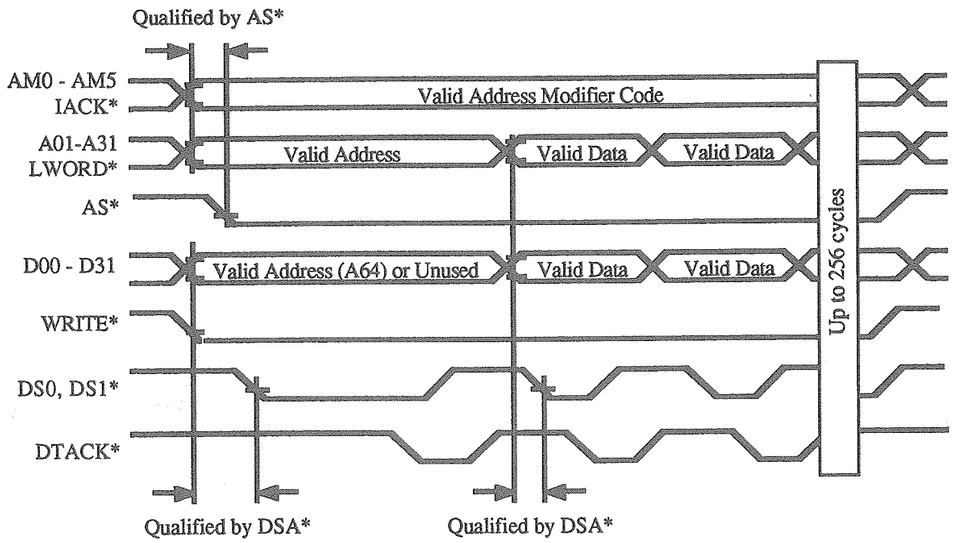

Figure 2-8. D64 block transfer write cycle. The read cycle is similar, except that data is valid on the falling edge of DTACK*. D64 block transfers were introduced in the proposed VME64 version of the VMEbus Specification.

2.2.3 A64 Addressing

Like D64BLT, all A64 cycles are 100% compatible with existing VMEbus backplanes and modules. All A64 cycles are two or more transfers in length. As Figures 2-7 and 2-8 show, the first transfer is an address broadcast and is common to all A64 cycles.

A64:D64, A64:D32, A64:D16, A64:D08(EO) and A64:D08(O) cycles have a single data transfer following the address broadcast. A64:D64BLT, A64:D32BLT, A64:D16BLT and A64:D08(EO)BLT cycles have one or more data transfers following the address broadcast.

Specific changes to the VMEbus specification included in VME64 are:

- Specification of address modifiers for A64 (long) cycles.
- Specification of how address bits are assigned to the data lines during A64 address broadcast cycles.
- Specification of the rules and timing parameters that govern the change over of the data lines from the address transfer role to the data transfer role. This is of particular concern for read cycles where the master must relinquish control of the data lines to the slave. For more information, refer to the section of this book on address pipelining.

2.2.4 Read-Modify-Write Cycle

The read-modify-write cycle is used in multiprocessor and multitasking systems. This special cycle allows many processes to share common resources such as disk controllers, serial ports or memory. As the name implies, the read-modify-write cycle reads and writes data to a memory location in a single bus cycle. It prevents the allocation of a common resource to two or more processes. The read-modify-write cycle is sometimes called an indivisible cycle or a test-and-set cycle.

One possible application for the read-modify-write cycle is an airline ticket reservation system. Consider two people (at different locations) reserving seats on a flight from New York to London. Unless specific care is taken, it is possible for both passengers to be allocated the same seat (if both reservations are made at the same time). The read-modify-write cycle can be used to prevent this situation.

For example, assume the ticket reservation software is set up so that each seat on the flight is represented by a bit in memory. If the bit is zero, the seat is empty. If the bit is one, then the seat is reserved. To reserve a seat, the ticketing software first reads the bit to see if the seat is occupied. If the bit is zero the software sets it to a one. If it the bit is set (the seat is already occupied), then the software looks for another seat.

The following software is an example of problematic 680XX assembly code that allows two seats to be allocated to the same passenger. This software does not generate a read-modify-write cycle:

```
*
* A0 = LOCATION OF MEMORY BIT REPRESENTING SEAT
*
            BTST.B #7,(A0)      * IS THE SEAT TAKEN?
            BEQ GETSEAT         * BRANCH IF SO
            .
            .                   * LOOK FOR ANOTHER SEAT
            .
GETSEAT:    BSET.B #7,(A0)      * RESERVE THE SEAT
```

The problem occurs between the time the bit is tested (BTST.B) and set (BSET.B). During this interval several instructions are performed (such as BEQ). In a multiprocessing or multitasking

system, a bus arbitration could take place between the time the bit is checked and set. If another processor is running the same code, at the same time, both could get the same seat because they both read the same bit as zero. The outcome would be two passengers booked on the same seat; an embarrassing problem for the airline.

This problem can be solved with a read-modify-write cycle. In the 680XX family, a CPU can generate this cycle using the TAS (test-and-set) instruction. This instruction reads a byte, tests the condition of bit #7, sets it to a one, and writes it back to memory. Rewriting the previous program using the TAS instruction:

```
*
* A0 = LOCATION OF MEMORY BIT REPRESENTING SEAT
*
            TAS (A0)        * TEST BIT AND SET IT
            BEQ GETSEAT     * BRANCH IF AVAILABLE
            .
            .               * ADDITIONAL CODE
            .
GETSEAT:                    * . . . AND CONTINUE
```

The airline reservation example is simplistic, but it does illustrate the use of the read-modify-write cycle. Many multiprocessing systems must arbitrate for system resources such as memory buffers and peripherals.

Not all VMEbus masters or slaves can participate in read-modify-write cycles. Modules that generate or accept the cycle are said to be RMW compatible. All modules must be able to tolerate RMW cycles, however. When evaluating or designing bus modules, look at the software requirements to find out if modules need to be RMW compatible.

The read-modify-write cycle is shown in the timing diagram of Figure 2-9. During the cycle, back-to-back read and write cycles are performed while AS* remains asserted. In the first half of the cycle WRITE* is negated, and data is read from memory. The master modifies the data, asserts WRITE*, and puts it back. Keeping AS* asserted prevents bus arbitration in the middle of the cycle. See Chapter 3 for more information on bus arbitration.

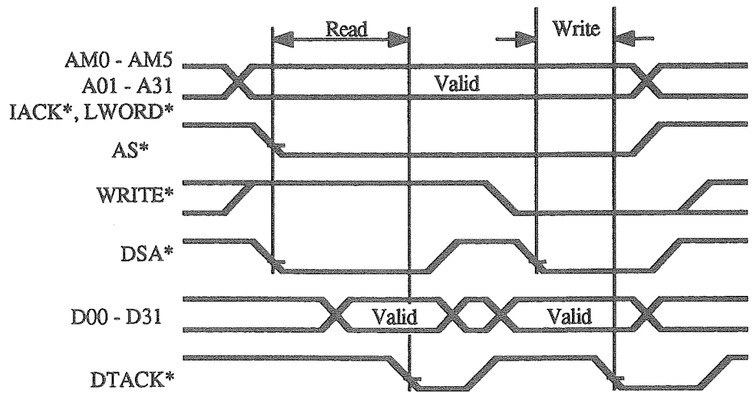

Figure 2-9. The read-modify-write cycle. Note that AS* remains asserted during the entire cycle.

When evaluating or designing VMEbus masters capable of read-modify-write cycles, make sure they do not change their address lines during the cycle. This can cause problems on processors such as the 68020 which utilizes a special RMC (Read-Modify-Control) pin. Under certain conditions the 68020 can change its address lines during the read-modify-write cycle. One possible solution is to latch the microprocessor address lines before the start of every cycle.

Problems can also happen on read-modify-write slaves. A common mistake is to use AS* to drive the slave's DTACK* generator. Once the slave has been selected, use data strobes DS0* and DS1* to assert and negate DTACK*. If this is not done, the module could lock up the bus.

2.2.5 Address-Only Cycle

The address-only cycle is used to broadcast an address. No data is transferred during the cycle. This cycle is differentiated from read/write cycles in that masters do not assert either data strobe. Since neither data strobe is asserted, the slave does not terminate the cycle with DTACK* or BERR*. Figure 2-10 shows the address-only cycle.

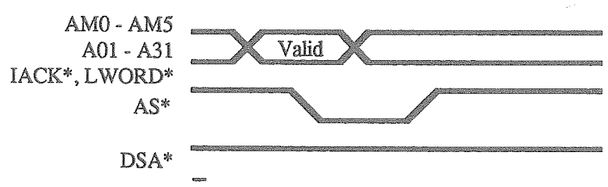

Figure 2-10. Address-only cycle.

The address-only cycle allows a master's local memory address decoder to work in parallel with a slave's, and can speed up the bus in some cases. It can also simplify the design of some masters. For example, a 68010 MPU with 68451 MMU may terminate a bus cycle after the 68451 asserts AS* (the MMU aborts the external cycle). The bus interface design is simplified if these cycles are leaked onto VMEbus.

The ADO mnemonic is used to describe modules that can initiate or tolerate address-only cycles. When evaluating or designing slave modules, make sure they tolerate ADO cycles. The IEEE-1014-1987 version of the VMEbus specification requires that all slaves support ADO cycles. Modules designed under earlier specifications are not required to do so.

2.3 Circuit Example - Simple 8-bit Parallel I/O Module

While most boards in a system can be purchased through established vendors, the need often arises for at least one custom VMEbus module. These are usually specialized I/O modules that are customized to the application. Often the resources dedicated to these custom projects are larger than those given to the rest of the system. Several circuits are presented here to aide the user in understanding and designing simple VMEbus interface circuits. All circuit examples presented in this book have been built and tested.

Figure 2-11 shows a simple circuit for an 8-bit parallel I/O module with an A16:D32:D16:D08(O) interface. This module illustrates some basic slave interfacing concepts including address decoding, bus timing and use of the control signals. It can be used as a building block for real time clocks, A/D converters, D/A converters and other simple I/O functions.