CONTENTS
SECTION 3

RETURN TO SECTION TOC

RETURN TO SECTION TOC

RETURN TO SECTION TOC

RETURN TO SECTION TOC

RETURN TO SECTION TOC

RETURN TO SECTION TOC

CHAPTER

B1

Overview of MB86932

1.1 General Description

The MB86932 is a member of the SPARClite family whose function set is a superset of that of the MB86930. It is pin-compatible with the 208-pin version of the MB86930 processor, and is capable of running at 40MHz. In addition to all the features of the MB86930 processor, the MB86932 contains the following:

- **Address Translation:** A 16-entry Translation Lookaside Buffer (TLB) on the MB86932 provides the mechanism to translate virtual to physical addresses. Both virtual and physical address spaces are 4GB in size. Page sizes of 4K-bytes, 256K-bytes, and 16M-bytes are supported. Protection at the 1K-byte sub-page level is supported. Up to 64 independent concurrent processes (“contexts”) are supported, with protection against memory encroachment by any process on any other.
- **Instruction Cache:** The MB86932 has an 8K-byte, 2-way set associative, sectored instruction cache with 8-word lines. Each line is individually lockable. Tags for each line contain the address tag, a supervisor/user bit, and 8 “valid” flags, one for each word of the line. The instruction cache is a physical cache; that is, it is accessed with a physical, not a virtual, address. When code is to be removed from the cache, the cache can be invalidated in a single cycle; likewise, “locked” code in the cache can be unlocked in a single cycle.
- **Data Cache:** The MB86932 has an 2K-byte, 2-way set associative, sectored data cache with 4-word lines. Each line is individually lockable. Tags for each line contain the address tag, a supervisor/user bit, and 4 “valid” flags, one for each word

of the line. The data cache is a physical cache; that is, it is accessed with a physical, not a virtual, address. When data is to be removed from the cache, the cache can be invalidated in a single cycle; likewise, "locked" data in the cache can be unlocked in a single cycle.

- **On-Chip DMA:** The MB86932 has two DMA channels. Each channel supports two transfer types: contiguous block and chained block transfers. The DMA also supports three transfer protocols: single-datum transfer, block transfer, and demand transfer (where data moves continue as long as an external device requests it). Four data types are supported: byte, halfword, word, and quad-word. For byte and halfword, the DMA does all the required packing/unpacking. Each channel also supports either fly-by or flow-thru transfer modes, and each can be started by either software or external hardware requests. The addressing convention for accesses is "big_endian."
- **Configurable External Data Bus:** The MB86932 includes a data bus that can be configured at Reset as 8, 16, or 32 bits wide (when in the address space selected by chip select 0). This enables the MB86932 to boot from a single by-8 or by-16 ROM.
- **Burst Mode:** The MB86932 supports two data- and instruction-accessing modes to external memory: normal and burst. In normal mode, it accepts a single datum per address, driven externally. In burst mode, it accepts 4 words per address, driven externally. Burst mode stores are supported only as part of DMA requests, and no burst mode transfers are supported in 8/16 bit mode.

1.2 Programmer's Model of the MB86932

1.2.1 User-visible Registers

All the special-purpose registers and ASR registers defined on the MB86930 exist also on the MB96832. Note that the version number in the PSR register is 5 for the MB86932.

All on-chip control/status/data registers which exist in alternate address spaces in the MB86930, with one exception, exist also on the MB86932 in backwards-compatible format. The one exception is the Instruction Tags, whose format has changed.

The increase in cache and the addition of new peripherals in the MB86932 have made it necessary to add new registers, accessible through alternate address spaces; these are described below in 3.4.1.1. All on-chip memory-mapped control/status registers for these new features are mapped into ASI=0x01, 0x02, 0x03, 0x0C, 0x0D, 0x0E, or 0x0F. The BIU recognizes that these ASI's are mapped to internal registers rather than memory, and does not assert the external ASI pins (or any other pins) when doing accesses in these ASI spaces. Since the address calculated by the IU for any register of

this class is its physical address, no address translation is necessary, and the TLB is not involved

In the lists that follow, an appended asterisk (*) = “new in MB86932”; a double asterisk (**) = “changed from equivalent in MB86930.”

Cache/BIU control/status registers:

ASI: 0x01

Address range: 0x00000000-0x000000FF

0x00000000	ASI=0x1	Cache/BIU Control Register** (TLB enable bit added)
0x00000004	ASI=0x1	Lock Control Register
0x00000008	ASI=0x1	Lock Control Save Register
0x0000000C	ASI=0x1	Cache Status Register
0x00000010	ASI=0x1	Restore Lock Control Register
0x00000020	ASI=0x1	Bus Control Register*
0x00000080	ASI=0x1	System Support Control Register** (DMA priority; even/odd paritybits added)

Peripheral control/status registers:

ASI: 0x01

Address range: 0x00000100-0x000001FF

0x00000120	ASI=0x1	Same Page Mask Register
0x00000124	ASI=0x1	Address Range Specifier Register 1
0x00000128	ASI=0x1	Address Range Specifier Register 2
0x0000012C	ASI=0x1	Address Range Specifier Register 3
0x00000130	ASI=0x1	Address Range Specifier Register 4
0x00000134	ASI=0x1	Address Range Specifier Register 5
0x00000140	ASI=0x1	Address Mask Register 0
0x00000144	ASI=0x1	Address Mask Register 1
0x00000148	ASI=0x1	Address Mask Register 2
0x0000014C	ASI=0x1	Address Mask Register 3
0x00000150	ASI=0x1	Address Mask Register 4
0x00000154	ASI=0x1	Address Mask Register 5
0x00000160	ASI=0x1	Wait State Specifier Register** (SGL cycle/parity bit added)
0x00000164	ASI=0x1	Wait State Specifier Register** (SGL cycle/parity bit added)
0x00000168	ASI=0x1	Wait State Specifier Register** (SGL cycle/parity bit added)
0x00000174	ASI=0x1	Timer Register
0x00000178	ASI=0x1	Timer Preload Register
0x00000180	ASI=0x1	Source/Destination ASI Register (DMA0)*
0x00000184	ASI=0x1	Current Source Address Register (DMA0)*
0x00000188	ASI=0x1	Current Destination Address Reg (DMA0)*
0x0000018C	ASI=0x1	Current Byte Count Register (DMA0)*
0x00000190	ASI=0x1	Descriptor Pointer (DP) Register (DMA0)*
0x00000194	ASI=0x1	Channel Control Register (DMA0)*
0x00000198	ASI=0x1	Channel Status Register (DMA0)*
0x000001A0	ASI=0x1	Source/Destination ASI Register (DMA1)*
0x000001A4	ASI=0x1	Current Source Address Register (DMA1)*

0x000001A8	ASI=0x1	Current Destination Address Reg (DMA1)*
0x000001AC	ASI=0x1	Current Byte Count Register (DMA1)*
0x000001B0	ASI=0x1	Descriptor Pointer (DP) Register (DMA1)*
0x000001B4	ASI=0x1	Channel Control Register (DMA1)*
0x000001B8	ASI=0x1	Channel Status Register (DMA1)*

*ASSP Control/status registers: **

ASI: 0x01

Address range: 0x00000200-0x000002FF

Note: This space is reserved for additional control/status registers for possible future derivatives of the SPARClite family of products.

*TLB Entries: **

ASI: 0x01

Address range: 0x00000300-0x000003FF--

Note: This allows up to 32 entries in the TLB, although only 16 entries are used in the MB86932. The TLB can be read or written by the "lda" or "sta" instructions.

0x00000300	ASI=0x1	TLB RAM Entry 1
0x00000304	ASI=0x1	TLB CAM Entry 1
...		
...		***other TLB entries***
...		
0x00000378	ASI=0x1	TLB RAM Entry 16
0x0000037C	ASI=0x1	TLB CAM Entry 16

*TLB Status/Control Registers: **

ASI: 0x01

Address range: 0x00000400-0x000004FF

Note: The TLB enable bit is in the Cache/BIU Control Register.

0x00000400	ASI=0x1	ITLB Register "RAM" Entry
0x00000404	ASI=0x1	ITLB Register "CAM" Entry
0x00000408	ASI=0x1	Context Register
0x0000040C	ASI=0x1	Context Table Pointer Register
0x00000410	ASI=0x1	TLB Control Register
0x00000414	ASI=0x1	Data Fault Status Register
0x00000418	ASI=0x1	Instruction Fault Status Register
0x0000041C	ASI=0x1	TLB Most Recently Used Register
0x00000420	ASI=0x1	TLB Data Fault Address Register

Emulation Registers:

ASI: 0x01

Address range: 0x0000FF00-0x0000FFFF

0x0000FF00	ASI=0x1	Instruction Address Descriptor Register 1
0x0000FF04	ASI=0x1	Instruction Address Descriptor Register 2
0x0000FF08	ASI=0x1	Data Address Descriptor Register 1
0x0000FF0C	ASI=0x1	Data Address Descriptor Register 2
0x0000FF10	ASI=0x1	Data Value Descriptor Register 1
0x0000FF14	ASI=0x1	Data Value Descriptor Register 2 or Mask Register
0x0000FF18	ASI=0x1	Debug Control Register **
0x0000FF1C	ASI=0x1	Debug Status Register
0x0000FF20	ASI=0x1	Context Compare Register **

*Instruction Cache Lock Registers: ***

ASI: 0x02

Address range: 0x00000000-0x00000FFF (Bank 1)

0x80000000-0x80000FFF (Bank 2)

Note: Writing to every eighth *word* address in this space can be used to initialize the lock bit for each line in the instruction cache. This differs from the MB86930, where every *fourth* word location is accessed.

Data Cache Lock Registers:

ASI: 0x03

Address range: 0x0000FF00-0x000003FF (Bank 1)

0x8000FF00-0x800003FF (Bank 2)

Note: Writing to every fourth *word* address in this space can be used to initialize the lock bit for each line in the data cache. This is unchanged from the MB86930

*Instruction Cache Tag RAM: ***

ASI: 0x0C

Address range: 0x00000000-0x00000FFF (Bank 1)

0x80000000-0x80000FFF (Bank 2)

Note: Writing to every eighth *word* address in this space can be used to initialize the tags for each line in the instruction cache. This differs from the MB86930, where every *fourth* word location is accessed.

*Instruction Cache Invalidate Registers: **

ASI: 0x0C

Note: These registers are in addition to the Instruction Cache Tags which are accessed using ASI 0x0C.

0x00001000 Bank 1 Instruction Cache Invalidate (write only)

0x80001000 Bank 2 Instruction Cache Invalidate (write only)

*Instruction Cache Data RAM: ***

ASI: 0x0D

Address range: 0x00000000-0x00000FFF (Bank 1)

0x80000000-0x80000FFF (Bank 2)

Note: Writing to *word* addresses in this space can be used to initialize the values in the instruction cache.

Data Cache Tag RAM:

ASI: 0x0E

Address range: 0x00000000-0x000003FF (Bank 1)

0x80000000-0x800003FF (Bank 2)

Note: Writing to every fourth *word* address in this space can be used to initialize the tag bit for each line in the data cache. This is unchanged from the MB86930

*Data Cache Invalidate Registers: **

ASI: 0x0E

Note: These registers are in addition to the Data Cache Tags which are accessed using ASI 0x0E.

0x00001000 Bank 1 Data Cache Invalidate (write only)

0x80001000 Bank 2 Data Cache Invalidate (write only)

Data Cache Data RAM:

ASI: 0x0F

Address range: 0x00000000-0x000003FF (Bank 1)

0x80000000-0x800003FF (Bank 2)

Note: Writing to *word* addresses in this space can be used to initialize the data RAM. This is unchanged from the MB86930

1.3 Internal Architecture of the MB86932

Figure B1-1, shows the general block diagram of the MB86932. Figure B1-2 shows in more detail the major units and buses connecting them. The solid lines show the bus connections that are used when the accesses are within the user or supervisor data/ instruction ASI spaces (ASI 08, 09, 0A, and 0B). The dashed lines show the additional connections required to access the control/status or data registers through the alternate ASI spaces. The major buses are:

- Data Data Bus (DD)—A 32-bit bus used to transfer data to and from MB86932 functional units. In general, when a load is executed, data is transferred to the Integer Unit (IU) from one of the other units, and when a store is executed, data is transferred from the IU to one of the other units. When loads/stores to user or supervisor data space are performed, the DD gives the IU access to the Data Cache, the BIU (if the data is not in the cache), or the DSU (if the data is to be accessed out of DSU memory).

  When doing Load Alternates or Store Alternates, the DD bus can access all units except the Instruction Cache and Instruction Tags, which can be accessed only through the ID bus. In such a case, the IU can read data (load alternate) or write data (store alternate) to the control/status/data registers of all units. Since the TLB can be accessed by the IU only through alternate space, their connection is shown as a dashed line.

- Virtual Data Address bus (VDA)—This 32-bit bus connects the IU, where the virtual address is generated, to the TLB, where the virtual address is translated to a 32-bit physical address. It also connects to the Debug Support Unit.

- Physical Data Address bus (PDA)—This 32-bit bus carries the physical address generated by the TLB. During loads/stores to user or supervisor data space, it is used

both to access the Data Cache, and to compare against the Data Tags. The PDA bus also goes to the BIU for use when data is not cached, and has to be accessed from external memory.

For load alternates and store alternates, the PDA goes to all units (except for the I_cache and the I_tags), so that control/status/data registers can be accessed.

- Instruction Data bus (ID)—This 32-bit bus normally transfers instructions from either the Instruction Cache, the Bus Interface Unit, or the DSU (when code is being run out of DSU memory).

  Note: When a store alternate is being performed to the I_cache or the I_tags (during cache initialization, for example), the data are first transferred from the IU to the BIU on the DD bus. The BIU then transfers the data on the ID bus to the I_cache or the I_tags. When a load alternate from the I_cache or the I_tags to the IU occurs, the reverse operation takes place. This obviates the need to extend both the ID and the DD busses to the I_cache and I_tags. (In the figure below, the connections for reading/writing the tags through alternate space are shown as dashed lines.)

- Virtual Instruction Address bus (VIA)—This 30-bit wide bus carries the 30-bit virtual address generated by the IU to the TLB for translation into a Physical Instruction Address (PIA). (Since instructions must fall on word boundaries, their addresses need only specify a full word address, for which 30 bits suffices.) The VIA also goes to the Debug Unit.
- Physical Instruction Address bus (PIA)—This carries the translated address from the TLB to the I_cache and I_tags, and to the BIU for use when the instruction is not cached, and access must be made to off-chip memory. The PIA can also be driven by the BIU when doing a store or load alternate to the I_cache or I_tags. In this case, the item to be stored is first sent to the BIU over the DD bus, with the address on the PDA bus. The BIU accepts this item, and drives it back on the ID bus, with the address on the PIA bus. This obviates the need to connect both the ID and DD buses to the I_cache and I_tags.
- Alternate Space Identifier address bus (ASI)—This 8-bit bus is driven by the IU, and indicates which address space a load/store is transferring data from/to. Load- and Store-Alternate instructions are used to read/write status/control/data registers in the various units of the MB86932.
- DMA Data Data bus (DDD)—This 32-bit local bus goes from the DMA to the BIU, and is used to send/receive data to/from the BIU during a DMA operation.
- DMA Data Address bus (DDA)—This 30-bit local bus goes from the DMA to the BIU, and is used to send the source or destination address to the BIU during a DMA operation.
- DMA ASI bus (DDASI)—This 8-bit local bus goes from the DMA to the BIU, and is used to send the ASI value to the BIU for cases where the DMA is addressing an alternate address space.

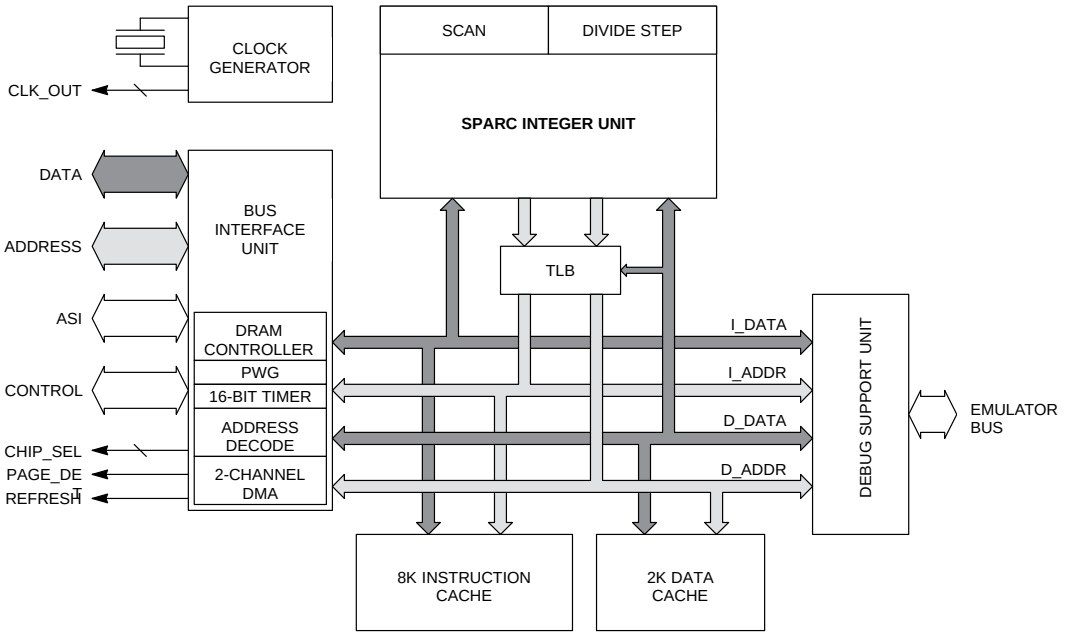


Figure B1-1. MB86932 Block Diagram

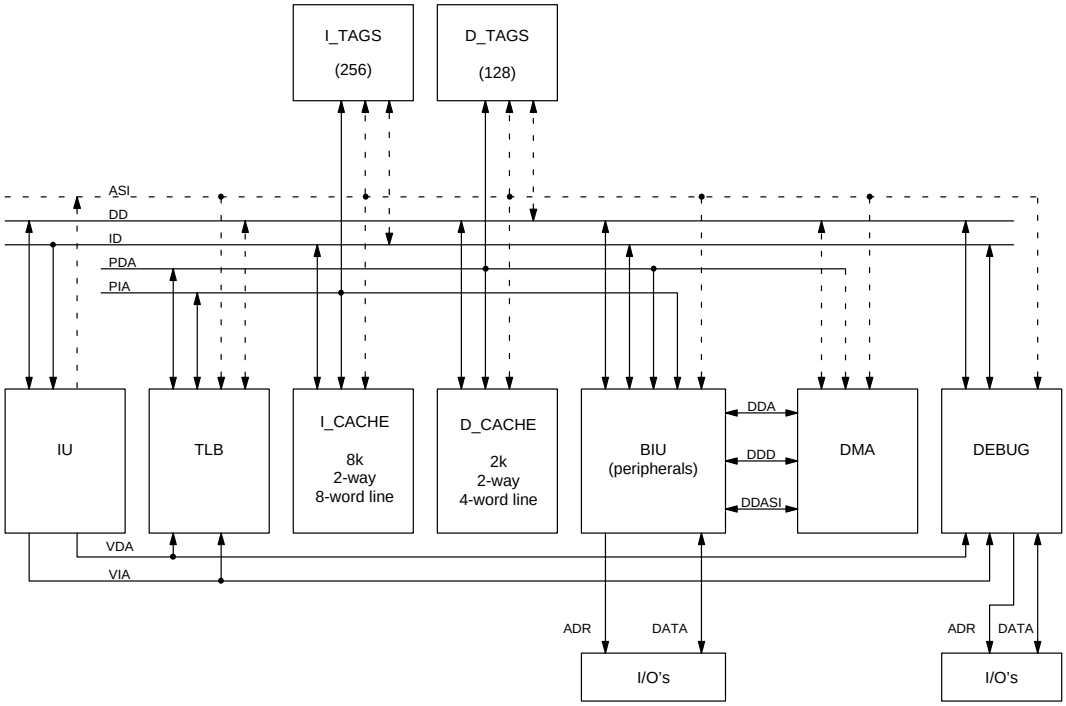


Figure B1-2. MB86932 Detailed Block Diagram

CHAPTER B2

MB86932 Memory Management Unit

2.1 Overview

The MB86932 provides hardware support for the implementation of an on-chip Memory Management Unit (MMU). No particular MMU architecture is determined for the MB86932. Rather, the hardware has been designed so that it can support a wide range of MMU architectures. In particular, it is possible to implement the SPARC Reference MMU using the hardware provided on-chip. For further information on compatibility with the SPARC Reference MMU, please see **See Section 2.5**.

The features provided by the MB86932 hardware are:

- A 16-entry Translation Lookaside Buffer (TLB)
- 32-bit virtual and physical address formats
- Support for pages/regions of different sizes (4K, 256K, 16M, 4G)
- Support for up to 64 processes (or contexts)
- Support for either single level or multi-level page tables, and
- TLB-miss processing initiated by hardware traps.

2.1.1 Memory Management Units: A General Description

This section provides a general description of MMU's, their function and benefits, for users that may be unfamiliar with them. It also defines terms that will be used throughout this chapter.

Figure B2-1 shows a block diagram of how an MMU fits with the CPU, cache, and memory. The MMU is responsible for doing address translations of the "virtual address" coming from the CPU to the "physical address" going to the cache and main memory. The "virtual address" is the address that the running program generates (from 0 up to 2^{32} for the SPARC Architecture). The "physical address" is the address that the hardware cache and memory receives. The CPU, as it runs code, produces virtual addresses which are dynamically translated to the physical address translation provided by the MMU. The benefits of virtual to physical address translation provided by the MMU are the following:

- Supports Virtual Memory
- Supports Multiple executing processes
- Supports Memory Protection

2.1.2 Virtual Memory

"Virtual memory" is the memory space that the program can address. For example, for the SPARC Architecture the virtual memory is 2^{32} bytes. "Physical memory" is the actual amount of memory (RAM, ROM, etc.) that is implemented in the system. Usually the physical memory is significantly smaller that the virtual memory.

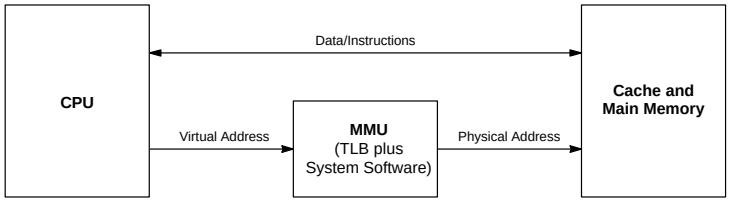


Figure B2-1. MMU's Role in Address Translation

Because this is true, it is necessary to dynamically allocate sections of this physical memory to code and data which is accessed by the program virtual address.

This is accomplished in the following way. The virtual memory space is broken into segments called "pages" (4k bytes in the SPARC Reference MMU). Similarly, the

physical address space is broken into equivalent sized segments called “page frames”. The complete program and data can be stored in mass storage (e.g. disk) until requested by the running program. When the program requests data not in physical memory, the required page needs to be retrieved from mass storage and put into an available page frame in physical memory. The MMU keeps track of where each page is placed in physical memory through the use of an “address translation table”, also called a “page table”. In the most general case, the address translation table contains, for each virtual memory page address, either a corresponding physical memory page frame address or a pointer to mass storage where that virtual page can be found. As long as the page is in physical memory the MMU uses this table to translate virtual addresses to physical addresses as the program executes. When the page is not found in physical memory the MMU is responsible for retrieving pages in mass storage and placing them in physical memory so that they can be accessed by the program.

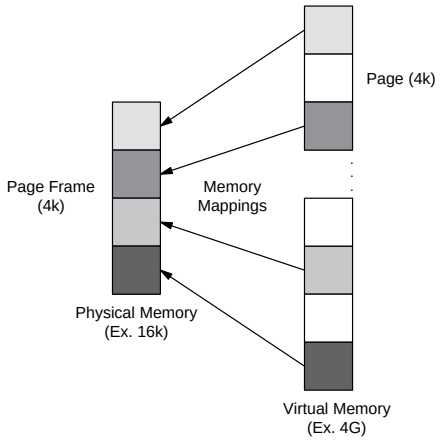


Figure B2-2. Memory Mappings

As an example, Figure B2-2 shows a physical address space of 16k bytes and a virtual address space of 4G bytes. The page size and the page frame size are both 4k bytes. The figure shows four virtual pages residing in physical memory. Other parts of the program would reside in mass storage (e.g. on disk).

Conceptually, both the virtual and physical address can be thought of as having two fields—the msb’s making up the page number and the lsb’s making up the offset (within the page). Effectively, the MMU’s does address translation by taking the page number from the virtual address and replacing it with the corresponding physical page number from the address translation table. The offset remains the same.

2.1.3 Multiple Processes

A process is an "executing" program. At any time a process can be "running" on the CPU or "waiting" (e.g., waiting for I/O). Multiple processes can be executing at the same time but there can be only one running process. Each process may be using a number of physical page frames in memory. For example, the four page frames in Figure B2-2 could be holding 4 pages each of which could be associated with a different process.

Each executing program (or process) sees its own 2^{32} virtual address space. To support multiple processes there must be a way to translate between a process's virtual addresses and the physical addresses of that process's pages in memory. To accomplish this the MMU uses a "context register" the value of which is used to identify the process which is currently running. Also, required is a "context table pointer register". The context table pointer register contains a pointer of the head of a table which in turn contains pointers to address translation tables for each process. The context register is used as an offset into this table. Thus, when a particular process is running the MMU must add the context to the context table pointer register to get the head of the address translation table for that process. See Figure B2-3. Once the table is found the virtual to physical address translation can complete as described in section 2.1.2

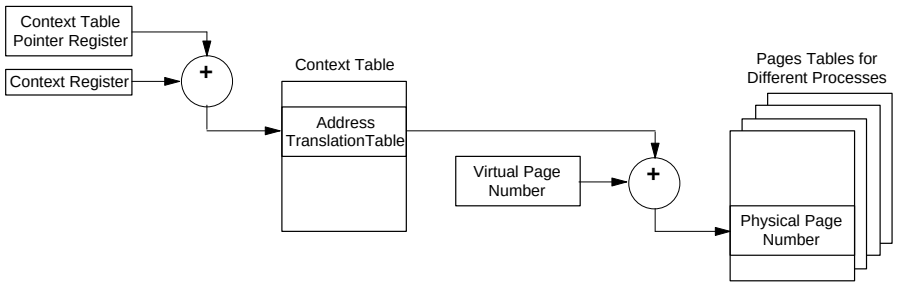


Figure B2-3. Schematic of Address Translation

2.1.4 Memory Protection

Memory protection can occur at two levels: the process level and the page level.

Memory protection at the process level is supported by the context register and the context table pointer register. Since each process can only go through its own address translation table when doing a virtual to physical address translation (as shown in Figure B2-3) one process can be prevented from accessing another process's instruction and data.

Since memory is segmented into pages, it is possible to associate with each page a protection field which can give permissions to the running program. These permissions can include whether the page can be read, written, executed, etc. by the program. For each page the permissions allowed are stored in the address translation table in the entry corresponding to that page.

2.1.5 How MMU's are Constructed

MMU's are constructed from a combination of hardware and software.

Software:

On the software side, the MMU is composed of the address translation table(s). There is one table for each process although this table can be either single level or multi-level as is defined in the SPARC Reference MMU (see Figure B2-3). These address translation tables reside in physical memory.

The MMU also is composed of the system software needed to search these tables to do virtual to physical address translations. When the running program generates a virtual address the MMU must conceptually translate that address by adding the context table pointer register to the context register to get a pointer which is added to the virtual page number to finally get the physical page number. This physical page number replaces the virtual page number to generate the physical address.

Finally, MMU software is required to move pages of instructions/data between mass storage and main memory as different pages of the running program are accessed.

Hardware:

The price of virtual addressing is that virtual addresses must be translated into physical addresses on the fly, at the time they are needed during execution. If each translation of virtual to physical addresses required a table lookup, as described above, the processor would run exceedingly slowly. Fortunately, instruction and data accesses exhibit a property known as "locality" – that is, they tend to occur not at random locations, but near each other on one or more recently-used pages.

To take advantage of this property, the MB86932 stores in one on-chip structure, the Translation Lookaside Buffer (TLB), 16 recently-used virtual page numbers, together with their corresponding physical page numbers. In another structure, the Instruction Translation Lookaside Buffer (ITLB), it stores the virtual and physical page numbers of the instruction page currently being accessed. Together these structures act as small cache of the most recently used virtual/physical address translations. Because of locality of address translations, most of the time the translation is done using the TLB.

Only rarely does the CPU have to go to the address translation table to find a physical page number. Since the TLB translations occur in parallel with cache/memory access there is no time penalty as long as the translation pair is in the TLB.

Hardware is also provided to cause a trap whenever a requested virtual address is not found in the TLB. The trap software can be written to use the context register and context pointer register to find the head of the current process address translation table in physical memory. The virtual page number can be used as an offset in this table to find the physical page number. The trap software can then store this virtual page number/physical page number pair in the TLB for future use.

2.2 Programmer's Model

This section describes the user visible relations and their functions. Many of the registers depicted below are very similar to those provided in the MB86930, except for a few bits or fields that support the MMU in the MB86932.

2.2.1 Cache/Bus Interface Unit Control Register

The cache/bus interface unit control register is identical to that on the MB86930 except for the addition of the "TLB Enable Bit" (TE), bit 6 of the register. When cleared, the TLB is disabled, and translations from virtual to physical addresses do not occur. When set, translations are enabled, contingent on the state of the TLB. The TE bit is cleared on reset.

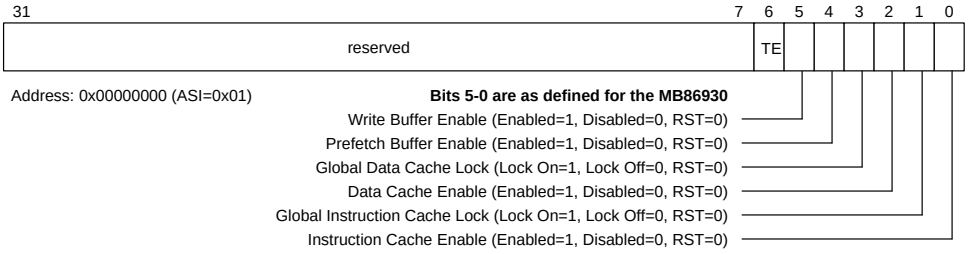
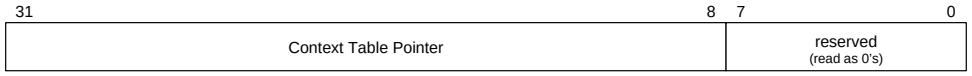


Figure B2-4. Cache/Bus Interface Unit Control Register

2.2.2 Context Table Pointer Register

This register holds the physical address of the base of the context table, which resides in main memory. When a software table walk is being done, the lower 8 bits of the context

register can be added to the context table pointer register to create an offset into the context table in memory.



Address: 0x0000040C (ASI=0x01)

Figure B2-5. Context Table Pointer Register

2.2.3 Context Register

Bits 7 through 2 of the context register are implemented. This register has two functions: first, it provides protection between processes. During a TLB access, the context field is compared against the corresponding field in the TLB entry. If the two match—or if the global bit is set to show that context is irrelevant in this case— a virtual-to-physical address translation occurs. Second, it can be used during a software table walk, when the context field is used as a word offset into the context table in main memory. This is done by adding the context register to the context table pointer register, producing the physical address of the desired root pointer in the context table in main memory.

Address: 0x00000408 (ASI=0x01)

Figure B2-6. Context Register

2.2.4 TLB Exceptions

There are only two kinds of faults that can be caused by a TLB access: the *instruction_access_exception* and the *data_access_exception.*, resulting respectively from an instruction address translation fault and a data address translation fault.

The MB86932 uses two of the existing traps defined in the SPARC (version 8) instruction set to support the TLB. The traps used are:

1. Instruction_access_exception

   Version 8 → Priority=5; trap type=0x01

   (a) A TLB miss occurred on an instruction access.

   (b) A blocking error such as “protection violation” occurred on an instruction access.

(c) A first reference to the instruction page was made, and the RT bit in the TLB Control Register was set.

(d) An external mexc signal occurred during an external instruction fetch.

(e) A parity error was detected on an external instruction fetch.

The cause of the instruction_access_exception is indicated by the Instruction Fault Status Register.

2. Data_access_exception

   Version 8 → Priority=13; trap type=0x09

   (a) A TLB miss occurred on a data access.

   (b) A blocking error such as "protection violation" occurred on a data access.

   (c) A first reference or first modification to this data page was made, and the RT or DMT bit in the TLB Control Register was set.

   (d) An external mexc signal occurred during an external read or write.

   (e) A parity error was detected on an external data read.

The cause of the data_access_exception is indicated by the Data Fault Status Register.

Since these two exceptions can be generated by several different causes, both TLB– and non-TLB related, two registers, described below, have been included to indicate the source of the exceptions.

2.2.5 Instruction Fault Status Register

There can be multiple causes for the instruction_access_exception; in particular, the TLB can cause this exception for a number of reasons. The Instruction Fault Status Register exists to indicate the exact reason for the fault, whether TLB-related or not. If the instruction_access_exception occurred, the address of the faulting instruction is in r[17].

The instruction Fault Status Register is a read-only register. The bits in this regis–ter are set by hardware when an instruction_access_exception occurs and indicate the cause of the instruction_access_exception. This register is cleared when either

The instruction Fault Status Register is a read-only register. The bits in this register are set by hardware when an instruction_access_exception occurs and indicate the cause of the instruction_access_exception. This register is cleared when either the instruction_access_exception. This register is cleared when either the Instruction Fault Status Register or the Data Fault Status Register is read by software.

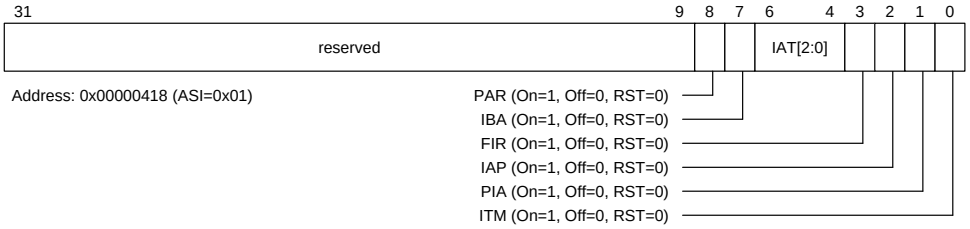


Figure B2-7. Instruction Fault Status Register

Bits 31-9: Reserved

Bit 8: Parity bit (PAR)—If IBA bit set, a set PAR indicates parity error; if PAR is cleared, “mexc” pin strobed, but no parity error detected.

Bit 7: Instruction Bus Access exception (IBA)—Set when either external “mexc” or parity error occurs during external instruction fetch.

Bits 6-4: Instruction Access Type (IAT[2:0])—This is the ACC field (from TLB) for the instruction causing the exception.

Bit 3: First Instruction Reference (FIR)—Set when first reference is made to so-far “unreferenced” instruction page; this causes a trap only if the “RT” bit of “TLB Control Register” is set.

Bit 2: Instruction Access Protection violation (IAP)—Set when an instruction lacks access permission sought.

Bit 1: Privileged Instruction Access violation (PIA)—Set when user-mode instruction seeks access to supervisor-mode area.

Bit 0: Instruction TLB Miss (ITM)—Set when address translation not in TLB or ITLB.

2.2.6 Data Fault Status Register

There can be multiple causes for the data_access_exception; in particular, the TLB can cause this exception for a number of reasons. The Data Fault Status Register exists to indicate the exact reason for the fault, whether TLB-related or not. If the data_access_exception occurred, the address of the datum causing the fault is held in the Data Fault Address Register.

The Data Fault Status Register is a read-only register. The bits in this register are set by hardware when a data_access_exception occurs and indicate the cause of the data_ access_exception. This register is cleared when either the Instruction Fault Status Register or the Data Fault Status Register is read by software.

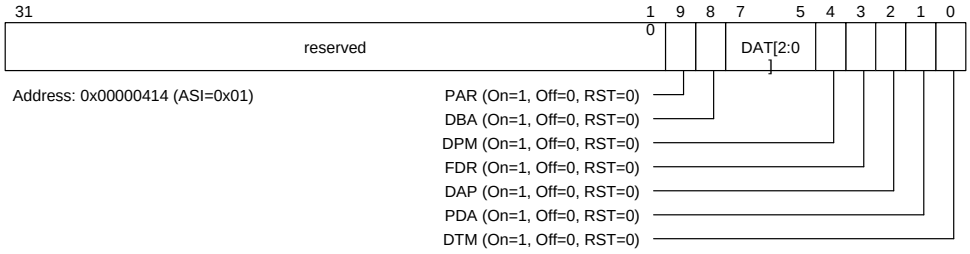


Figure B2-8. Data Fault Status Register

Bits 31-9: Reserved

Bit 9: Parity bit (PAR)—If IBA bit set, a set PAR indicates parity error; if PAR is cleared, "mexc" pin strobed, but no parity error detected.

Bit 8: Data Bus Access exception (DBA)—Set when either external "mexc" or parity error occurs during external data read.

Bits 7-5: Data Type (DAT[2:0])—This is the ACC field (from TLB) for the data causing the exception.

Bit 4: Data Page Modification (DPM)—Set when first store is done to so-far unmodified data page; causes trap only if DMT bit in TLB Control Register is set.)

Bit 3: First Data Reference (FDR)—Set when first reference is made to so-far unreferenced data page; causes trap only if RT bit in TLB Control Register is set.

Bit 2: Data Access Protection violation (DAP)—Set when data access is attempted without permission for type of access sought.

Bit 1: Privileged Data Access violation (PDA)—Set when data access sought to supervisor area when in user mode.

Bit 0: Data TLB Miss (DTM)—Set when data address translation not in TLB.

2.2.7 TLB Control Register

One bit in this register is used to control whether a fault can occur on the first write to an unmodified page, and the other bit is used to control whether a fault can occur on the first reference to a previously "unreferenced" page. These bits can be used to support different page-replacement schemes; they are cleared to 0 on reset.

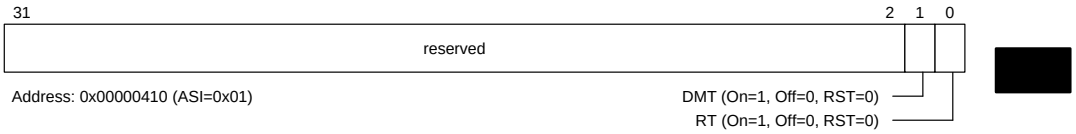


Figure B2-9. TLB Control Register

Bits 31-2: Reserved

Bit 1: Data Modify Trap (DMT)—Control bit, enables trapping on first modification of a datum in a so-far unmodified data page.

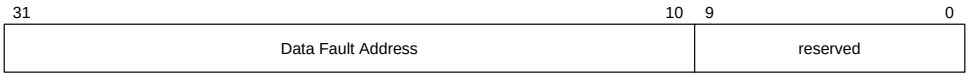
Bit 0: Reference Trap (RT)—control bit, enables trapping on first reference to so-far “unreferenced” page; cleared on reset.

2.2.8 TLB Data Fault Address Register

When a TLB fault occurs, it is necessary for the trap code to have access to the virtual address of the faulting instruction or data access. This allows the trap handler to know what address caused the fault. In the case of a TLB miss, this address is needed to perform a software table walk in main memory to find the correct translation. The instruction address is also necessary so that the access can be retried once the reason for the fault has been corrected.

In the case of a TLB fault during an instruction access, the virtual address of the faulting instruction is held in r[17] of the trap handler window. Thus, no special register is needed for this address. For data addresses the situation is different, since the effective data address is not saved during a trap; the TLB Data Fault Address Register is used instead to hold this address value. When a TLB fault is recognized during the memory stage of the pipeline, the address on the Data Address Bus is latched and held. This register can be read by the trap software. The TLB Data Fault Address Register contains the 22 most significant bits (MSB’s) of the faulting data address, which is sufficient information for the table walk. The format is shown in Figure B2-10.

The TLB Data Fault Address Register is a read-only register. When a TLB data_access_exception occurs the virtual data address is captured and held. This register is cleared when read by software.



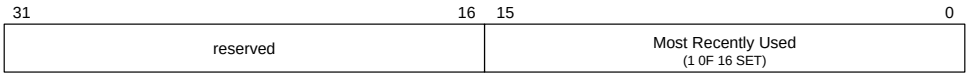
Address: 0x00000420 (ASI=0x01)

Figure B2-10. TLB Data Fault Address Register

2.2.9 Most Recently Used Register

Bits 0 through 15 of the Most Recently Used Register correspond to the 16 entries of the TLB. The register is updated every time a TLB match occurs: the bit corresponding to the matched entry is set, and all others are cleared. On a TLB miss, this register can be read, and supports replacement algorithms whose policy is to leave the most recently used address in the TLB.

The Most Recently Used Register is a read-only register. This register is cleared when read by software.



Address: 0x0000041C (ASI=0x01)

Figure B2-11. Most Recently Used Register

2.2.10 The TLB Entry

The TLB has 16 fully associative entries, each of them consisting of an entry in the CAM (content-addressible memory) array and the corresponding RAM array.

2.2.11 The TLB CAM Entry

The CAM-array entry is shown in the diagram below. Each CAM entry consists of a 20-bit virtual page number (VPN) that contains three index fields, a 2-bit fragment index, and a 6-bit context number (process identifier) that are compared against the virtual page address from the Integer Unit (IU) and the content of the context register. In addition, a global bit (G), two level bits (1:0), and a fragment enable bit (FE) are included. The CAM provides simultaneous comparison of all 16 TLB entries against the current virtual page address and context number. If a CAM entry matches the virtual page address, the corresponding RAM entry in the TLB provides a physical page number (PPN) to generate a physical address.

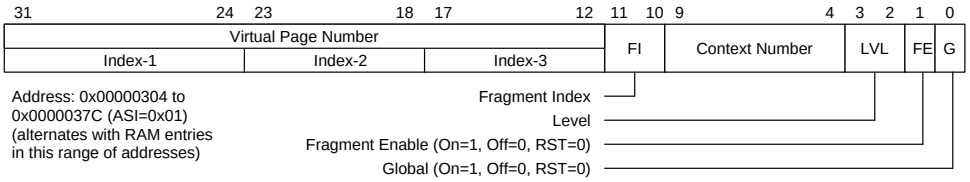


Figure B2-12. CAM Entry Format

Bits 31-12: Virtual Page Number (VPN)—Some page table subfields may be masked by Index-3, -2, and -1. Index-1, -2 and -3 support different page sizes.

During translation, the context field in the CAM is compared against the value in the context register. Only when these two values match can a RAM entry be selected. The only exception is when the corresponding Global bit is set, indicating that context is irrelevant. The Global bit is, in effect, an enabling switch for the Context field; if set, it masks out that field.

The two Level bits determine what page sizes are part of the MMU architecture. They work by acting as (encoded) masks, excluding from the comparison process one or more of the index subfields, as detailed in the following table:

Table B2-1: Level-bits Decoding Table

LVL[1:0]	Address Mapping	M3M2M1	TLB Field masked
11	4-kbyte	0 0 0	None
10	256-kbyte	1 0 0	Index 3
01	16-Mbyte	1 1 0	Indexes 2 and 3
00	4-Gbyte	1 1 1	Indexes 1, 2 and 3

As summarized in the table, the mask bits M3, M2 and M1 are used to exclude from comparison index 3 (bits 17-12), index 2 (bits 23-18), and index 1 (bits 31-24), respectively. If M3 is set, index 3 is masked out, and a RAM entry is selected based on the match of indexes 1 and 2, and the context number. The 6-bit index 3 combined with the 12-bit page offset can provide an index to a 256-Kbyte linear addressing region. If both M3 and M2 are set, the RAM entry is selected based on the match of the index 1 and the context number. The 12-bit index (bits 23-12) combined with the 12-bit page offset can provide an index to a 16-Mbyte addressing region. If all three masks are set, the RAM is selected based on the context number match alone, and the RAM entry provides an index to a 4-Gbyte addressing region.

2.2.12 The TLB RAM Entry

The RAM-array entry is shown in the diagram below. It consists of a 20-bit (maximum) physical page number (PPN), a 3-bit access-level protection, a cacheable bit, a modify bit, and a valid bit. The mask bits (M1, M2, and M3) are a decoded version of the LVL field in the corresponding CAM entry.

The mask bits allow the TLB to generate a correct index into a page for different page sizes. The index fields that are excluded from the CAM comparison process are used as part of this index into the page—used as part of the offset into the selected page instead of part of the PPN. If all mask bits are clear, the 20-bit PPN drives the upper 20 bits of the physical address, and the 12-bit offset drives the lower 12 bits. If M3 is set, the lower 6-bits of the PPN are replaced by the bits 17-12 of the virtual address. Therefore, the physical address contains a 18-bit untranslated address (page offset) and a 14-bit page number. If both M2 and M3 are set, the lower 12 bits of the PPN are replaced by bits 23-12 of the virtual address. The physical address then contains a 24-bit untranslated address (page offset), and a 8-bit page number. If all mask bits are set, the virtual address outputs to the physical address.

Associated with each virtual page are coded values that indicate what kind of accesses (read, write, execute, or none) may be made to this page by this program. When loaded into the TLB these values are stored in the ACC field [5:2], and are compatible with the specification given in the *SPARC Reference MMU Architecture.* Note that entries in the TLB make no explicit reference to ASI spaces; this information is implicit in the access bits of the TLB.

Table B2-2: Access Protection available through PTE[4:2]

ACC Field Value	Accesses Allowed	
	User Access (ASI=0x8 or 0xA)	Supervisor Access (ASI=0x9 or 0xB)
0	Read Only	Read Only
1	Read/Write	Read/Write
2	Read/Execute	Read/Execute
3	Read/Write/Execute	Read/Write/Execute
4	Execute Only	Execute Only
5	Read Only	Read/Write
6	No Access	Read/Execute
7	No Access	Read/Write/Execute

Note: ASI=Alternate Space Identifier; an 8-bit value that indicates whether a load/store instruction transfers data to/from external units, or registers on the chip.

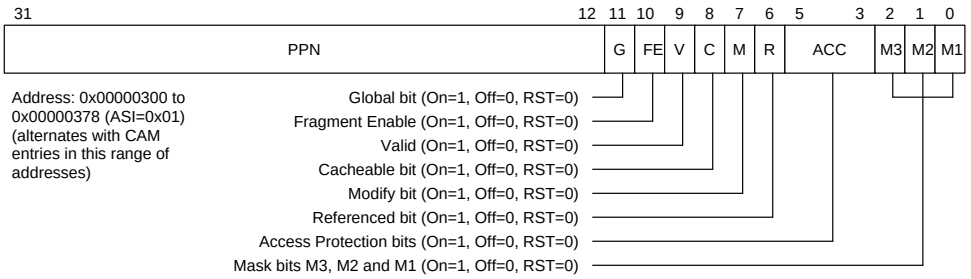


Figure B2-13. RAM Entry Format

Bits 31-12: (Maximum) Physical Page Number (PPN)—Some page table subfields may be masked by M3, M2, and M1)

Bit 11: Global bit (G)—This bit and the fragment enable bit (FE) are duplicates of the same bits in the corresponding RAM entry.

Bit 10: Fragment Index bit (FE)—If set, this bit asserts that the Fragment Index bits are to be included in the comparison. If so, the result is to establish access protection at the sub-page 1K level, the “fragment” level. When the FE bit is set, the Fragment Index bits are compared against virtual address bits 10 and 11 (which do not themselves go through translation). A RAM entry is then selected only when there is a VPN match *and* a FI match in the CAM entry. Since each TLB entry can set its own access level or protection, protection at the 1-Kbyte level is thus available. When the FE bit is not set, the FI bits are excluded from the comparison, and access protections apply only to the full 4-Kbyte (or larger) whole pages.

Bit 9: Valid bit (V)—This bit reports the current validity of the TLB entry. The V bit of each entry should be cleared by software to invalidate those entries before the TLB is enabled.

Bit 8: Cacheable bit (C)—This bit indicates whether the memory addressed by the TLB is cacheable or not.

Bit 7: Modify bit (M)—This bit in the TLB is set when the memory page is modified by a write operation.

Bit 6: Referenced bit (R)—This bit is set by the TLB when the page in question is accessed. This bit can be used in TLB replacement algorithms.

Bits 5-3: Access Protection bits (ACC)—The access-level protection for the address region mapped by the RAM entry. Access-level protection is checked during TLB access. If a TLB match occurs, but access-level protection is violated, the TLB will generate a trap.

Bits 2-0: Mask bits (M3, M2 and M1)—A decoded version of the LVL field in the corresponding CAM entry.

2.2.13 ITLB Description

Fully static implementation of a one-entry Instruction Translation Lookaside Buffer (ITLB) allows immediate access to the physical address of the last page entry stored there. The registers associated with the ITLB are in locations 0x00000400 ("RAM" entry) and 0x00000404 ("CAM" entry). These two registers are the same as the TLB RAM and CAM entries. The instruction cache hit/miss and access permissions are determined by the PTE in the ITLB, so there is no performance penalty in using the ITLB. If an access-level violation is detected, the ITLB generates an instruction_access_exception trap.

2.2.14 TLB Lookup

If an instruction address translation is not found in the ITLB, an ITLB hold is asserted, and the virtual instruction address is looked up in the TLB on the next (the second) cycle, preempting any data address translation. On a match in the TLB, the TLB entry is output to the ITLB. On the third cycle, the translation is retried using the ITLB, with guaranteed success. If the translation is not found in the TLB, an instruction_access_exception trap is asserted, and a software routine to access the address translation table in main memory can be executed. This is known as a "software table walk". Due to the locality of instructions, and the availability of the TLB in case of ITLB miss, the performance price of instruction address translation is minimal.

For data address translation, the virtual data address is used directly to compare against each entry in the TLB. If a TLB match occurs, the TLB outputs the physical address, the cacheable bit, and the access protection bits. If the translation is not found in the TLB, or if an access-level violation is detected, a data_access_exception trap is asserted. If the trap occurred because of a TLB miss, a software table walk can be initiated.

2.2.15 TLB Software Notes

Note 1: When the TLB is enabled, a lda or sta (load alternate or store alternate) instruction which accesses the TLB CAM/RAM to read or modify the contents of the TLB cannot be followed immediately by an instruction which uses the TLB for translation. Some other (non load/store) instruction or a nop must follow the lda or sta.

Note 2: When reading or writing TLB registers, only sta and lda instructions should be used. stda and ldda are not supported for the CAM/RAM entries in the TLB.

2.3 Internal Architecture

2.3.1 Details of TLB Logic

Because of the normally sequential nature of instruction addresses, it is likely that the next required instruction is on the current page. Accordingly, the virtual instruction address is sent directly to the ITLB for translation; only if the desired address is not found there is it sent to the TLB. If it is found in the TLB, it is loaded into the ITLB, and instruction address translation proceeds. If the physical address is not found in the TLB either, an “instruction_access_exception” trap is asserted, and a software table walk can be performed. The physical address found in the Page Tables is entered into the ITLB and TLB for use in later instruction address translations.

Because data is more widely scattered, data address translations go directly to the TLB. If the desired physical address is not found in the TLB, a “data_access_exception” trap is asserted, a table walk is performed, and the physical address when found in the Page Tables is entered into the TLB for later data address translations.

The net result for overall system performance is that few instruction or data references in a normally structured program need be translated by accessing the Page Tables in memory; the great majority of the physical addresses needed are found on-chip in the TLB/ITLB, and two physical addresses—one data address and one instruction address—can be acquired simultaneously. The details of the process of translating a virtual into a physical address are illustrated in the diagram and flowchart in Figures B2–14 and B2–15.

2.3.2 Address Translation: Logical and Physical Steps

The diagram below illustrates the registers and fields involved in TLB-based translation of a virtual address into a physical address. The flowchart supplements it by correlating the physical steps taken with their meaning from the user's point of view.

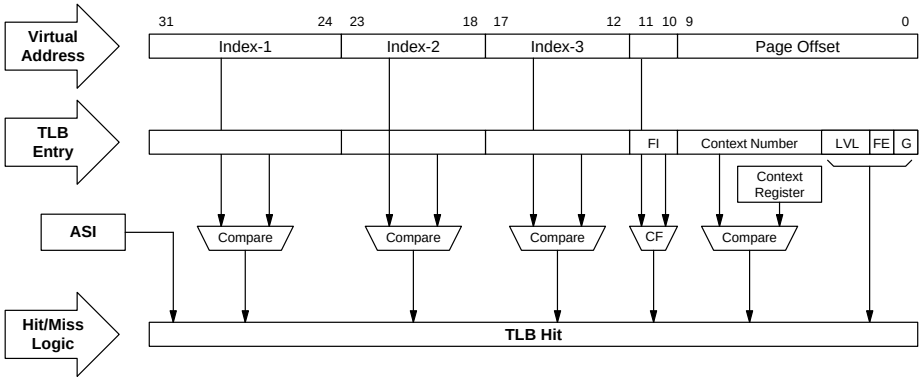


Figure B2-14. Address Translation by TLB: Fields and Registers

In this flowchart, the logic of each translation step is given in the left-hand box, and its physical realization in the corresponding right-hand one.

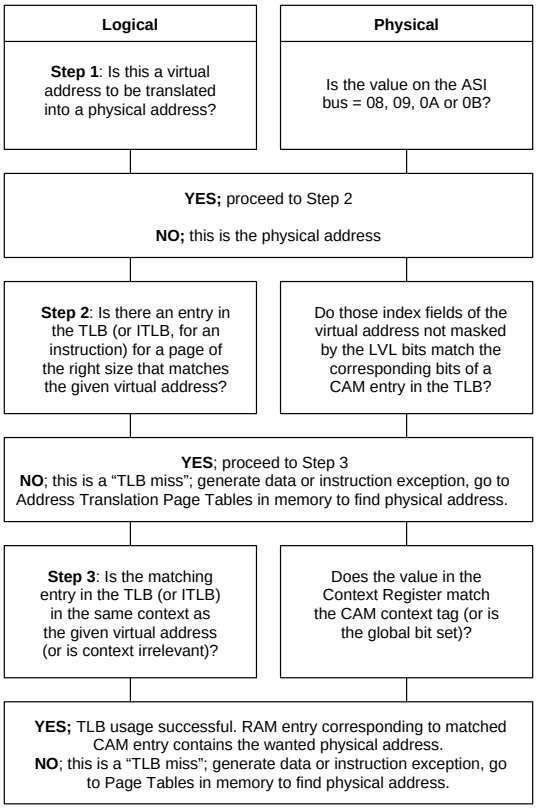


Figure B2-15. Flowchart of TLB Address Translation

2.3.3 Basic TLB Exception Timings

The diagram below indicates at what stage of the pipe the various TLB exceptions occur, and when the processor recognizes these exceptions. Instruction-access TLB faults occur during the fetch stage, while data-access TLB faults occur during the memory stage. All of the exceptions are recognized at the end of the memory stage.

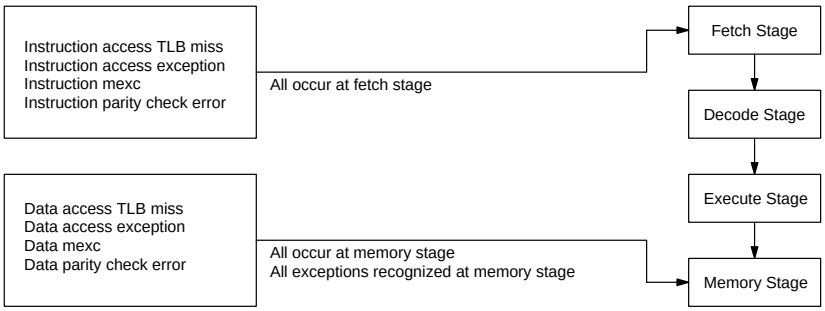


Figure B2-16. TLB Exception Timings

2.3.4 TLB Timing Considerations

The TLB does the instruction/data address translation in parallel with accessing the instruction/data caches (see Figure B3-2 in Chapter 3 *"MB86932 Caches"*). Thus, there is no additional cycle penalty when executing with the TLB enabled if the virtual/physical address translation is in the TLB. There are two exceptions to this rule:

1. If the virtual/physical address pair being accessed is in the TLB but not in the ITLB, the translation will require an additional two cycles. The first of these is used to access the TLB for the translation pair and load it into the ITLB; the second is used to retry the translation with the ITLB.
2. If either cache is disabled and the TLB is enabled, the TLB translation requires an extra cycle be inserted before the address can be driven on the Address pins.

Figure B2-17 shows the timing for TLB virtual-to-physical address translations. The "ITLB Status" indicates whether an ITLB hit or miss occurs for the address of the instruction in the fetch stage. The "TLB Status" normally indicates whether a TLB hit or miss occurs for the data address of the instruction in the memory stage.

Four situations are indicated in Figure B2-17. Cycle 0 is an example of both an ITLB hit for the instruction address of INST4, and a TLB hit for the data address of INST1.

Cycles 1 through 3 show what occurs when INST5 misses in the ITLB in cycle 1. The INST5 instruction translation is retried using the TLB in cycle 2. If a hit occurs, the ITLB is updated with the value from the TLB. Finally, the instruction address of INST5 goes through translation using the ITLB in cycle 3.

Cycles 4 and 5 indicate what happens when the instruction address translation for INST6 is in neither the ITLB nor the TLB. After accessing the ITLB and missing, the

TLB is accessed with the same instruction address. When a miss is detected here, this causes an instruction memory exception to occur. Note that this does not cause a trap until this instruction reaches its memory stage in cycle 8.

Cycle 8 shows a data address TLB miss. (The data address from the instruction in the memory stage always goes to the TLB except in the cases when the instruction address preempts it after an ITLB miss.) This data-address TLB miss causes a data_memory_exception which that is recognized in the same cycle in which it occurs.

For the last two cases, the exceptions cause the processor to vector to the trap routine (INST20) in cycle 9. The instructions in the decode, execute, and memory steps are “squashed” at that time. The faulting instruction (INST6 in both cases) does not write back a result to the register file. Instead, the PC (virtual address) of the faulting instruction is written to the register file. In addition, if a data address translation caused the fault, the value of the faulting virtual data address is written into the data fault address register.

2.3.5 TLB Emulation Support Logic

When the MB86932 chip is executing from the In-Circuit Emulation (ICE) port, all accesses, including those to supervisor instructions and data, should be untranslated. This is required because the ICE logic has a fixed memory map, and will not be able to handle translated addresses. The ICE code will have the responsibility of doing the table walk before accessing any location. It should access everything using physical addresses.

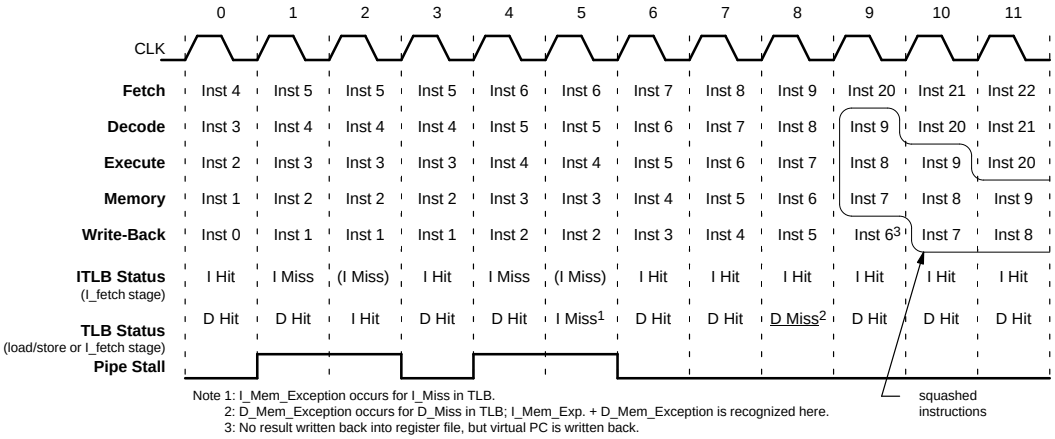


Figure B2-17. Sequence of Events for ITLB and TLB Misses

2.4 Programming Considerations

2.4.1 MMU Architecture Example: the SPARC Reference MMU

One MMU architecture that the MB86932 can support is the SPARC Reference MMU. This architecture can be implemented as follows.

The information the MMU required to perform virtual-to-physical address translation is put in a hierarchy of physically-addressed structures, the Page Tables, that reside in main memory. In the SPARC Reference MMU architecture, with three Page Tables, Page Tables 1 and 2 would contain two kinds of entries:

- *Page Table Pointers*, which contain the physical address of the logically next-lower table, and thereby link the tables together as a hierarchy (Note that PTPs are never found in the TLB.)
- *Page Table Entries*, which contain the physical address of a page of the size associated with the table (along with other page-specific information). Since the TLB caches PTEs, the information *content* of PTEs in main memory should be compatible with that of PTEs in the TLB, although the exact format may differ.

Page Table 3, can contain only Page Table Entries (PTEs), since there is no next-lower table for it to point to. SPARC reference compatible formats of the PTP and PTE are:

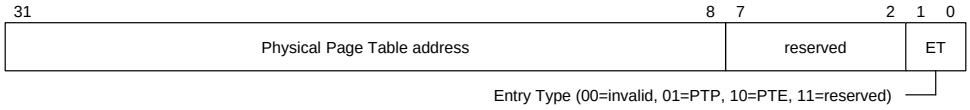


Figure B2-18. Page Table Pointer (PTP)

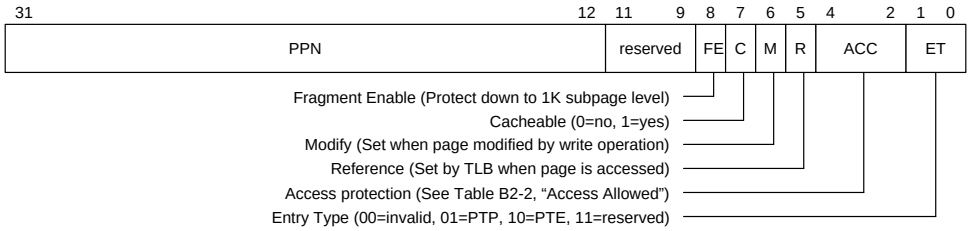


Figure B2-19. MB86932 Page Table Entry (PTE)

Note that the FE field is not part of the SPARC Reference MMU Architecture, but is introduced by the MB86932. (The reservation of bits [11:9] is similarly an MB86932 feature.)

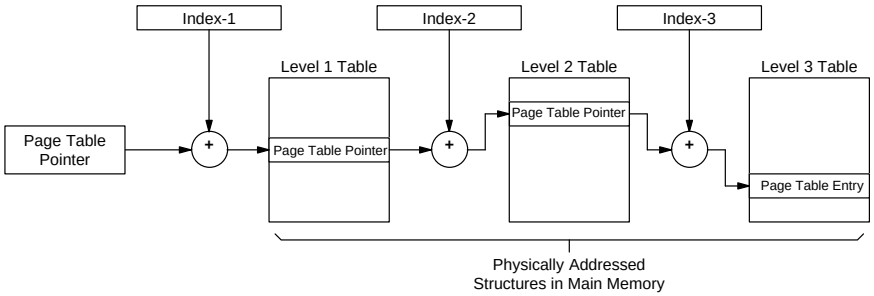


Figure B2-20. Pointer chaining from Page Table to Page Table

The table walk, or search from table to table for a matching virtual address, follows a simple logic: after the root pointer locates the Level-1 table in memory, and index-1 of the virtual address picks out a particular entry in that table, each succeeding table (if necessary) is located by the PTP just found, and the entry in that new table is picked by the next index field of the virtual address.

The reason for dividing the page pointers among three tables is to support sparse addressing efficiently; the root and the PTEs in the three tables point respectively to pages of 4 gigabytes, 16 megabytes, 256 kilobytes, and 4 kilobytes, so memory can be used in block sizes appropriate to an application's routines and data structures.

2.4.2 Virtual Address format

The format of a virtual address as generated by the Integer Unit (IU) and passed for translation to the TLB is as follows: the Page Offset field for the maximum (four page sizes) configuration is nominally 12 bits wide (bits 11-0) and specifies a particular byte within a 4K-byte page; the three index fields, collectively known as the Virtual Page Number (VPN) field, enable the TLB to identify the correct page. Each index field is an offset into the correspondingly-numbered page table.

If the page whose physical address is sought is larger than 4K-bytes, a 12-bit offset field is insufficient to identify a specific byte within it. But the PTE of a page size larger than 4K will not be in table 3, and will not require that all three Page Tables be walked during the translation process, so one or more of the index fields becomes available for use in forming a bigger page offset field, exactly as required. In effect, then, the offset field is always as big as needed for the current address translation process.

For example, if the information sought is in a 256K-byte page, the index-3 field, needed only when the PTE is in Table 3, is effectively made part of the Page Offset field for

that translation, giving an Offset field of 18 bits; this supports the identification of an individual byte in a 256K-byte page. Similarly, a 16M-byte page will not require index fields 2 or 3, yielding the effective 24-bit Offset field that is needed for addressing a 4M-byte address space.

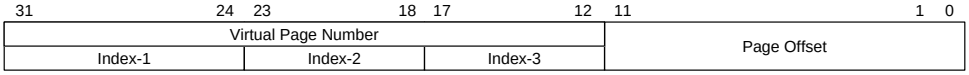


Figure B2-21. Virtual Address Format

2.4.3 Physical Address format

The MB86932 physical address format differs from that specified in the *SPARC Reference MMU Architecture* in being 32 bits wide rather than 36. It is otherwise as specified in that document: Since all pages begin on 4K-byte boundaries, the low-order 12 bits of the physical address are the same as those of the virtual address, and do not require translation.

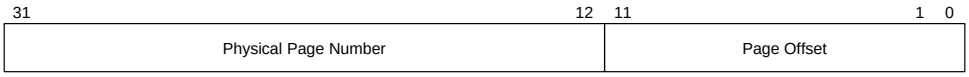


Figure B2-22. Physical Address format

2.5 Conformity to SPARC Reference MMU Architecture

The MMU architecture of the MB86932 is as specified in *The SPARC Reference MMU Architecture* (Sun Microsystems, Revision 1.4, 23 Jan 1989), with the following exceptions:

- The physical address format is 32 bits wide rather than the 36 bits specified in the Reference.Architecture.
- As a consequence of the difference in physical address format, the Physical Page Number (PPN) portion of the Page Table Entry (PTE) is 20 bits wide rather than the 24 bits specified in the Reference.Architecture.
- Bit 8 of the PTE, the Fragment Enable (FE) bit, has been reserved in the 932 to support protection down to 1K sub-page boundaries.
- A few of the registers and bits specified in the Reference Architecture are not implemented, or not implemented in full. Specifically: only bits 7-2 of the Context Register are implemented.

- The *instruction_address_MMU_miss* and the *data_address_MMU_miss* exceptions defined in the *Reference* are not implemented; on the occurrence of one of these misses, the TLB will instead vector to the instruction_access_exception trap routine, or the data_access_exception trap routine, respectively.

CHAPTER

B3

MB86932 Caches

3.1 Overview of MB86932 Caches

The MB86932 offers enhanced support for cacheing: its instruction cache is 8K-bytes in size, and has 8-word lines. (The corresponding values for the MB86930 are 2K-bytes and 4-word lines.) The data cache of the MB86932 remains the same as the MB86930's at 2K-bytes and 4-word lines. The increased instruction cache size is reflected in a new format for the Instruction Cache Tag, which has four new "valid" bits to control the four new words per cache line (the other four valid bits remain in the same positions they occupy in the I_Cache Tag in the MB86930, making for backward compatibility).

Both caches are "physical" caches; that is, the cache arrays are accessed with physical, not virtual, addresses. The addresses stored in the tag arrays are also physical, not virtual, addresses. Since the caches are accessed with physical addresses, the reading and writing of the caches is expedited by the MB86932's restriction of the minimum page size to 4K-bytes. This allows the lower 12 bits of the physical address to be identical to the lower 12 bits of the virtual address, which in turn means that, given 2-way set associativity, the cache can be up to 8K bytes without requiring any address translation when being accessed. The MB86932 uses this full 8K-byte space in its instruction cache, while in the data cache only 2K-bytes (of the possible 8K-bytes) are implemented.

3.2 Programmer's Model

The cache control/status registers of the MB86932 form a superset of those in the MB86930. The registers common to the two chips are the:

0x00000000	ASI=0x1	Cache/BIU Control Register** (TLB enable bit added)
0x00000004	ASI=0x1	Lock Control Register
0x00000008	ASI=0x1	Lock Control Save Register
0x0000000C	ASI=0x1	Cache Status Register
0x00000010	ASI=0x1	Restore Lock Control Register

To this set (all in the ASI=0x01 space) the MB86932 adds two Instruction_Cache_Invalidate Registers, one for each bank of the instruction cache, and two Data_Cache_Invalidate Registers, one for each bank of the data cache. All four are write-only; their format is shown below.

Bank 1 of the instruction cache is controlled by the register at address 0x00001000, while bank 2 is controlled by the register at address 0x80001000 both in ASI space 0x0C. Bank 1 of the data cache is controlled by the register at address 0x00001000, while bank 2 is controlled by the register at address 0x80001000, both in ASI space 0x0E.

Invalidating the cache, and clearing lock and lru bits, is an easy way to remove old code/data from the caches when a new page is brought into physical memory, or after a DMA has been made to cacheable locations in main memory. Clearing only the lock and lru bits is an easy way to allow locked code to be replaced after use. Note that the invalidate bits are written during the M stage of the instruction; thus, their effect is not felt until the fourth instruction after the instruction that writes to these registers.

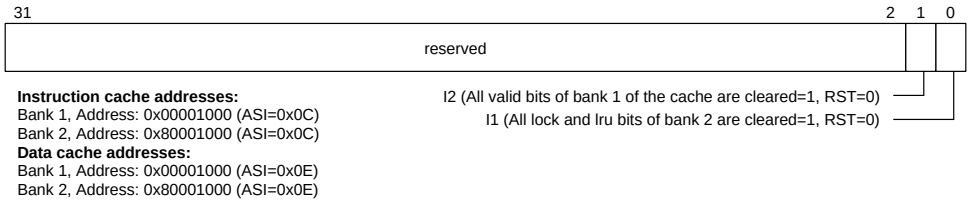


Figure B3-1. Cache Invalidate Register Format

3.2.1 Operation of the Instruction Cache

At reset the cache is turned off, and the valid bits, lock bits, and LRU bits are set to 0. Initialization of the cache to particular values can be done by doing stores to an alternate address space 0x0C. When the cache is off, all requests are sent to the external memory. After the cache is initialized, the user writes a 1 to the cache-on bit to turn on the cache.

3.2.2 Operation of the Data Cache

At reset, the cache is turned off, and the valid bits, lock bits, and LRU bits are set to 0. Initialization of the cache to particular values can be done by doing writes to alternate address space 0x0E. When the cache is off, all requests are sent to the external memory. After the cache is initialized, the user writes a 1 to the cache-on bit to enable the caches.

Accesses to the ASI's corresponding to user and supervisor data space are cached. No loads or stores from any other ASI are cached.

3.3 Internal Architecture of MB86932 Caches

Figure 3-Cache-1, below, shows how the TLB works with the caches (in the example shown, the Instruction Cache):

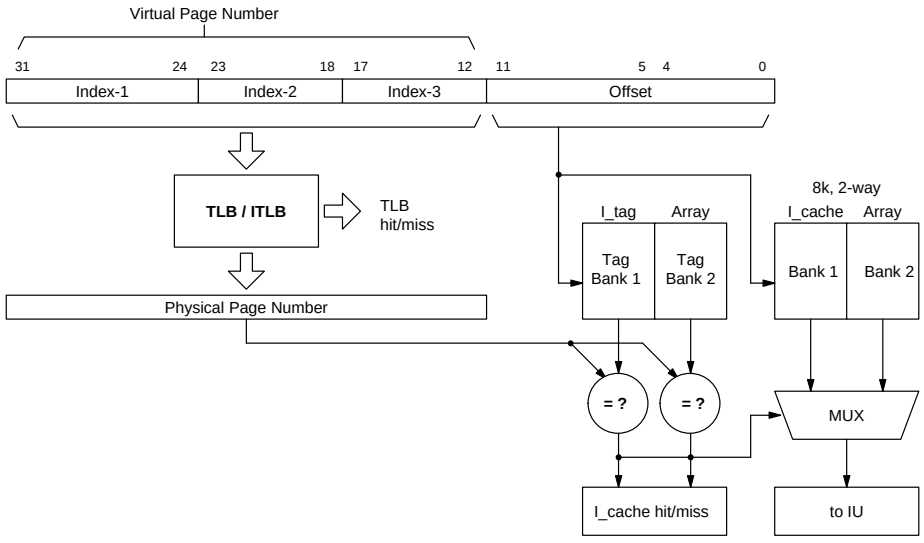


Figure B3-2. TLB/Cache Interaction

3.3.1 Instruction Cache

The instruction cache is an 8K-byte, 2-way associative, sectored cache, with 8-word lines. The basic operation of the cache is as follows: the IU sends the virtual address to the TLB, I_cache, and I_cache tags. Since the lower 12 bits of the virtual address are not translated, they are available immediately at the I_cache and tag array. Thus, the tag array can be accessed and the I_cache address can be decoded simultaneously with the TLB translation of the virtual page number to the physical page number. Once this is completed, the tag read from the tag array can be compared to bits 31-12 of the translated physical address to determine hit or miss.

The virtual instruction address format is shown below. The virtual page number has three index fields that are conditionally translated by the TLB, based on the mapped memory region size. The address coming out of the TLB is the physical address, and goes to the I_cache and tags. Bits 31-12 go to the tag array for comparison. Bits 11-5, which do not go through translation, select two tags (one for each bank) out of the 256-entry tag array, and also choose two lines (one for each bank) out of the 8K I_cache. Bits 4-2 select a word out of the 8-word line. In each of the diagrams below, bits 0-11 are the untranslated part of the address.

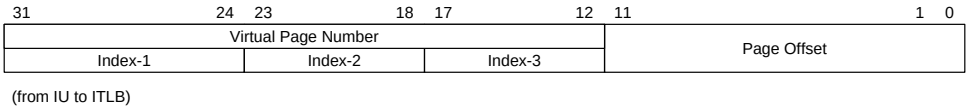


Figure B3-3. Virtual Instruction Address

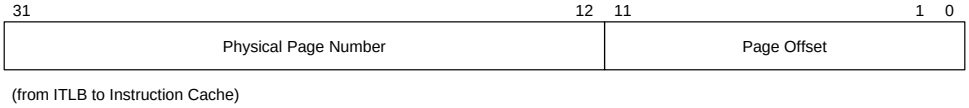


Figure B3-4. Physical Instruction Address

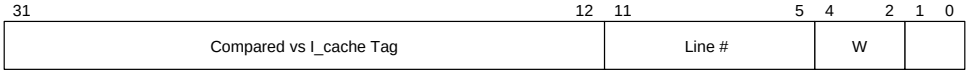


Figure B3-5. Address to I_cache and Tag Array

The instruction cache tag format is shown below. Twenty bits make up the address tag. Four bits, 9-6, are Valid bits for four of the words of the 8-word line. These bits are in the same location as the valid bits of the MB89630 I_cache tag array. Four additional

Valid bits have been added for the other four words of the 8-word line. Bit 5 is used to indicate whether the line can be accessed by supervisor only. Bit 1 is the least-recently used bit, which is used when doing a line replacement in the I_cache. Note that because of the increase in cache size and line size, the tag format of the MB86932 differs from that of the MB86930.

How the valid bits in a tag correspond to the words in the corresponding line is shown below:

Word Address [4:2]	000	001	010	011	100	101	110	111
Valid Bit Location	6	7	8	9	2	3	4	10

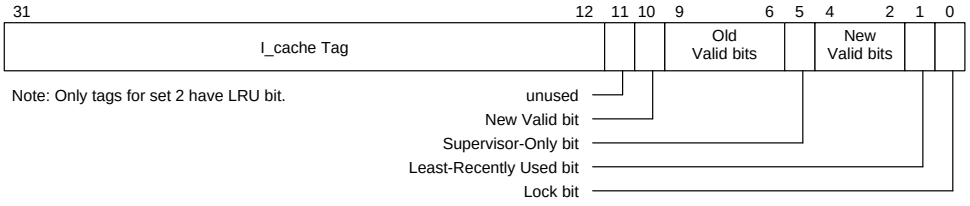


Figure B3-6. I_cache Tag Format

Note that any access that competes with a currently locked entry in the cache is treated as non-cacheable. In addition to the lock bits in the tag array, there is a global cache lock bit for each of the caches. Whenever these global lock bits are set, all accesses that do not result in a hit in the cache are treated as non-cacheable.

Writes to the instruction address space are not supported. The tag and instruction memory can be updated by doing writes to alternate address spaces 0x0C and 0x0D.

3.3.2 Read Hit

On an instruction fetch, the tag and the instruction are accessed in parallel, using the untranslated lower 12 bits of the address. If the translated bits of the address match one of the accessed tags, and the U/S fields match, and the “valid” bit corresponding to the word being accessed is set, then the required instruction is in the cache. The instruction is returned to the IU, and the LRU bit is updated. The lock bit may be updated, based on the value of the Instruction lock bit in the “lock control register.”

3.3.3 Miss Processing

If the address field in the tag does not match the translated address bits (31-12) coming from the TLB, or the U/S bit does not correspond to the ASI indicated by the IU, or the

corresponding "valid" bit is not set, the result is a cache miss. In this case, the "hold" signal to the IU, and the "miss" signal, are asserted. This freezes the IU pipeline. The request is sent to external memory via the BIU.

If the address field in the tag matches the translated address bits (31-12), and the U/S bit corresponds to the ASI indicated by the IU, and at least one of the valid bits is set (but the valid bit for the requested word is not set), it implies that an entry has already been allocated for this word. There is no need to select an entry to be replaced.

If the miss is due to the address field in the tag not matching the translated address bits (31-12), or the U/S bit does not correspond to the ASI indicated by the IU, or none of the valid bits is set, then an entry needs to be selected for replacement (or allocation). The LRU bit for this entry is checked, and the least-recently used entry is chosen to be replaced (or allocated).

The entry that is chosen for replacement will also depend on the "lock" bits. Consider two sets, A and B. If the lock bit for a given entry in A is set, and the corresponding bit of B is clear, then the entry in B will be replaced regardless of the value of the LRU bit. The LRU bit will be updated to show the entry in A to be the least-recently used. If the lock bit for both entries, or the lock bit for the whole cache, is set, then the access will be treated as a non-cacheable access.

In the case of an instruction fetch, when the required instruction is accessed from main memory, it is returned to the IU and stored in the cache. The "hold" signal freezing the IU is deasserted. If a line was replaced or allocated because of the cache miss, the valid bit for the accessed word is set, and the other valid bits are reset. If the word being accessed is part of an already allocated line, then only the "valid" bit for the accessed word is set. All other bits remain unchanged. The lock bit may also be updated based on the value of the Instruction lock bit in the "lock control register."

3.3.4 Data Cache

The data cache is a 2K-byte, 2-way associative, sectored cache, with 4-word lines. The basic operation of the cache is as follows: the IU sends the virtual address to the TLB, D_cache, and D_cache tags. Since the lower 12 bits of the virtual address are not translated, they are available immediately at the D_cache and tag array. Thus, the tag array can be accessed and the D_cache address can be decoded simultaneously with the TLB translation of the virtual page number to the physical page number. Once this is completed, the tag read from the tag array can be compared to bits 31-10 of the translated physical address to determine hit or miss.

The virtual data address format is shown below. The virtual page number has three index fields that are conditionally translated by the TLB, based on the mapped memory region size. The address coming out of the TLB is the physical address, and goes to the

D_cache and tags. Bits 31-10 go to the tag array for comparison. Bits 9-4, which do not go through translation, select two tags (one for each bank) out of the 128-entry tag array, and also choose two lines (one for each bank) out of the 2K D_cache. Bits 3-2 select a word out of the 4-word line. In each of the diagrams below, bits 11-0 are the untranslated part of the address.

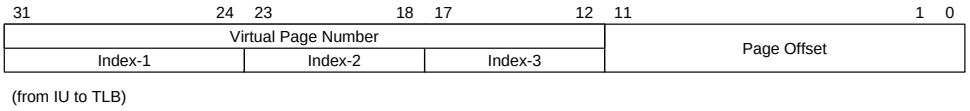


Figure B3-7. Virtual Data Address

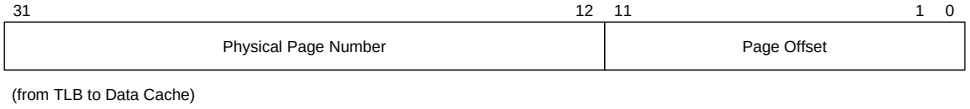


Figure B3-8. Physical Data Address from TLB

Figure B3-9. Address to D_cache and Tag Array

The data cache tag format is shown below. Twenty-two bits make up the address tag. Four bits, 9-6, are valid bits for each word of a D_cache line. Bit 5 is used to indicate whether the line can be accessed by supervisor only. Bit 1 is the least-recently used bit, which is used when doing a line replacement in the D_cache. Finally, bit 0 is used to lock the entry into the cache. Note that this format is identical to that of the MB86930.

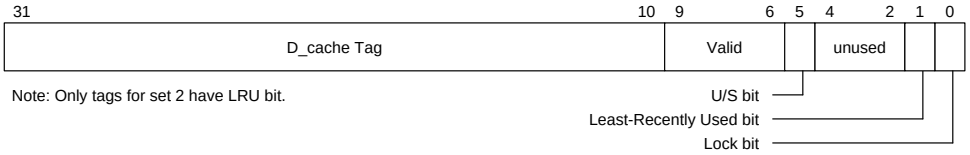


Figure B3-10. D_cache Tag Format

The data cache follows a write-through update policy. On a write hit, the data is written to both the cache and main memory. If there is a write miss, the data is written only to

the external memory. A different write policy is followed if the write is to a locked location.

The lock bit in the data cache has the effect of locking the current data in the cache. Any access that does not result in a hit in the cache, and maps to a location that is currently locked, is treated as non-cacheable. Any writes to locked data cache entries are not written to main memory. Only the data in the cache is updated.

3.3.5 Read Hit

On a load, the tag and the data are accessed in parallel, using the untranslated lower 12 bits of the address. If the translated portion of the address field coming from the TLB matches the tag, and the U/S bit corresponds to the ASI indicated by the IU, and the "valid" bit corresponding to the word being accessed is set, then the required data is in the cache. Since a hit is detected, the data is returned to the IU, and the "hold" signal to the IU is not asserted. The LRU bit is updated. The lock bit may be updated, based on the value of the Data lock bit in the "lock control register."

3.3.6 Write Hit

On a store(ST, STB, STH), if a hit is detected, the IU hold signal is not asserted. The LRU bit is updated. The lock bit may be updated, depending on the value of the Data lock bit in the "lock control register." If the lock bit for this entry is not set, or the Data lock bit in the "lock control register" does not indicate that the entry is to be locked, then the transaction is also sent to the BIU to be completed in external memory.

3.3.7 Miss Processing

If the address field in the tag does not match the translated address bits (31-10) coming from the TLB, or the U/S bit does not correspond to the ASI indicated by the IU, or the corresponding "valid" bit is not set, the result is a cache miss.

In the case of a write miss, the cache is left unchanged, and the request is sent to the BIU to be completed in external memory.

A read miss is processed in exactly the same way as a miss for an instruction fetch, except that the lock bit may be updated depending on the value of the Data lock bit in the "lock control register."

3.3.8 Atomic Load and Store

All atomic load and store transactions are treated as non-cacheable transactions.

CHAPTER

B4

MB86932 Bus Interface Unit

4.1 Overview of Bus Interface Unit

The BIU on the MB86932 includes all the features of the MB86930, and in addition offers the following:

- A four-word burst mode for instruction fetches and data loads,
- Byte-based parity generation/checking for the external data bus,
- A modified Wait State Specifier Register that supports burst mode and parity generation/checking on specified address ranges,
- A ROM/PROM interface that allows the MB86932 to boot from either 8-bit wide or 16-bit wide ROM/PROM,
- A processor bus request feature that enables the MB86932 to request access to external address and data buses,
- Modified timing on the external address bus when the TLB is enabled while caches are off.

4.2 Burst Mode

4.2.1 Overview

The Bus Interface Unit (BIU) supports the fetching of instructions and data from external memory to the appropriate cache in ’bursts’ of four words at a time. A burst

mode transfer is initiated either by a cache miss or by a DMA request. For a cache miss, burst mode is supported only for instruction fetches and data loads, not for stores. The IU is held until all four words are fetched. For DMA burst access, both data burst reads and data burst writes are supported. (Note, however, that the DMA does not support movement of data to/from cache.)

When burst mode is triggered by a cache miss, it replaces four words in the cache line where the miss occurred. Such a burst-mode transfer can take place only if (a) the enabling bit (see "Bus Control Register," below) is set, and (b) the external memory supports burst mode. In the case of an i_cache miss, only half the line is replaced, since i_cache lines are eight words long. In the case of a d_cache miss, the entire four-word line is replaced by a burst-mode fetch. The four-word sequence fetched in burst mode starts with the word that caused the miss, followed by three more words in a standard order.

4.2.2 Burst Mode Interface Pins

Two pins are dedicated to burst mode:

–BMREQ: Output pin to inform the memory system that the current bus transaction is a burst mode.

–BMACK: Input pin to inform the processor that the memory system can support burst mode.

Note: When a cache miss occurs, –BMREQ will be asserted only if the corresponding bit of the Bus Control Register (DBE for data, IBE for instructions) is set. However, for a DMA transaction, –BMREQ is asserted whenever a quad word transfer is requested, regardless of the status of the DBE bit.

4.2.3 Burst Mode Fetch Sequence

In burst-mode accesses, the BIU automatically uses the two least significant bits (LSBs) of the address of the requested word, ADR[3:2], to determine the sequence in which the other three words will be fetched. (The sequence is optimized for a 2-way interleaved memory.) The table below shows the four possible sequences of words, in terms of their address LSBs, depending on the LSBs of the word causing the miss. Note that the first word accessed in a burst is always the one requested by the IU and that during a burst access, bits ADR[3:2] do not change.

Table B4-1: Sequence of Words Fetched in Burst Mode

LSBs of Missed Word	SEQUENCE OF WORDS TRANSFERRED(in terms of their LSBs)			
	1st word	2nd word	3rd word	4th word
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

4.2.4 Bus Mode Control Bits

Two bits in the Bus Control Register are used to control burst mode for instruction fetches and data loads.

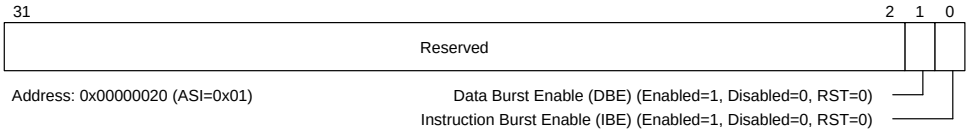


Figure B4-1. Bus Control Register

On reset, burst mode for both instruction and data misses is disabled. The user must explicitly enable one or both after reset. Bus operations already in progress are not affected by modification of the burst-enable bits.

4.2.5 PROM Address Space

Burst mode access from the PROM address space is not supported for 8- or 16-bit bus mode. If burst mode is enabled, and the address lies within the PROM address space for a non-32-bit bus mode transfer, the burst mode request output signal (–BMREQ) will still be asserted, but the burst acknowledge signal (–BMACK) should not be asserted by the external memory. If –BMACK is asserted under these conditions, the BIU operation is undefined.

4.2.6 Prefetch Buffer

The prefetch buffer is not used when burst-mode instruction fetches are enabled, and is automatically disabled if the IBE bit is set, regardless of the state of the Prefetch Buffer Enable bit in the Cache/BIU Control Register. If the external memory system cannot handle burst mode operations, the instruction burst mode should be left disabled, so that the prefetch buffer can be used.

4.2.7 Cache Off

Instruction and data burst mode is automatically disabled if the corresponding cache is turned off.

4.2.8 Bus Request

The bus will be released to service another request only after the completion of the burst mode transaction.

4.2.9 Memory Exception (Instruction fetches or Data loads)

All four word accesses of a burst mode access will be completed even if a memory exception occurs on any of the word accesses. During a burst access, word accesses that cause an external memory exception (–MEXC asserted) are not written into the cache, while any words that do not cause a memory exception are written to cache. Note that the Interger Unit will recognize a memory exception only when it is accessing the specific word with which the memory exception is associated.

For example, if the IU requested word 00, the BIU would burst-read 00, 01, 10 and 11. If an external memory exception occurred only on word 10, this word would not be written to the cache; the other three words, however, would be written to the cache. The IU would not vector to the memory_exception trap handler, since there was no memory exception on the specific word it requested.

If, however, the IU ever tried to access word 10, which was not written into the cache because of the memory exception, a miss would occur which would cause the BIU to fetch that word from memory again. If a –MEXC were asserted on this access of word 10, the processor would vector to the memory_exception trap handler, since this was the word specifically requested by the IU.

4.2.10 Memory Exception (DMA)

When a memory exception (–MEXC strobed) occurs on any word of a DMA burst read, the DMA will complete all four reads. The corresponding four writes, needed to complete the transaction, will not occur.

When a memory exception occurs on any word of a DMA burst write, the DMA will continue, completing all four writes.

A memory exception on a DMA transfer will not cause the IU to vector to the data_memory_exception trap routine.

4.2.11 Non-cacheable Accesses

Burst mode fetches from a non-cacheable address space are not supported. The burst request signal (–BMREQ) will not be asserted, and only a single-word fetch will be performed.

4.2.12 Interface Timing

Figure 3-BIU-1 below shows the timing of a burst mode transaction for an instruction fetch, data load, or DMA read. To start the transaction, the MB86932 outputs a burst

mode request signal (–BMREQ) to the memory system. The memory system asserts the burst mode acknowledge signal (–BMACK) to the processor when the first word is fetched, indicating that a burst mode request can be handled. The –BMACK should be asserted only in the cycle when the –RDYfor the first access is asserted. The memory latency involved in the first word fetch is the same as in a non-burst access, and subsequent fetches are usually shorter; as in the Figure, a single cycle. This does not mean that each fetch following the first will occur in one cycle; subsequent fetches can take any number of cycles, depending on the –RDY assertion. The –BMREQ signal is deasserted after the completion of the first word fetch.

If the memory system cannot handle a burst mode transaction, –BMACK will remain deasserted Once the burst mode logic detects an inactive –BMACK, the burst mode access will terminate. The burst mode logic will not attempt to complete the fetch of the remaining words in the cache line. However, –BMREQ will be asserted again for any subsequent misses. Therefore, for a certain address segment in which the memory system cannot handle a burst mode operation, the –BMACK signal can remain deasserted. An example is shown in Figure B4-3.

Figure B4-4 shows the timing for the write portion of a DMA burst operation. The timing is identical to that in Figure B4-2, except that the RD/–WR line is low, indicating a write operation is in progress.

Note that ADR[31:2] is the address of the first word fetched. This address remains constant through the burst.

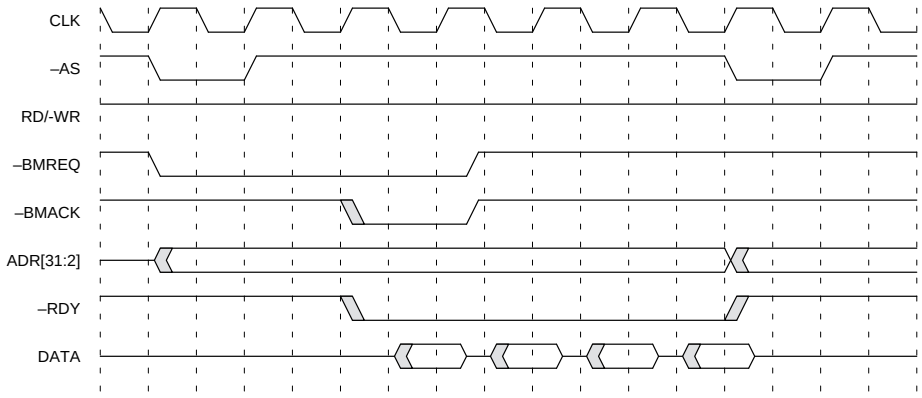


Figure B4-2. Burst Mode (0 wait state)

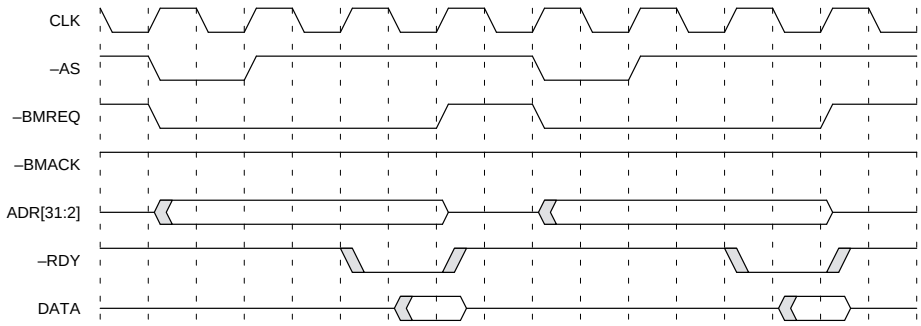


Figure B4-3. Terminated Burst Mode Due to –BMACK=1

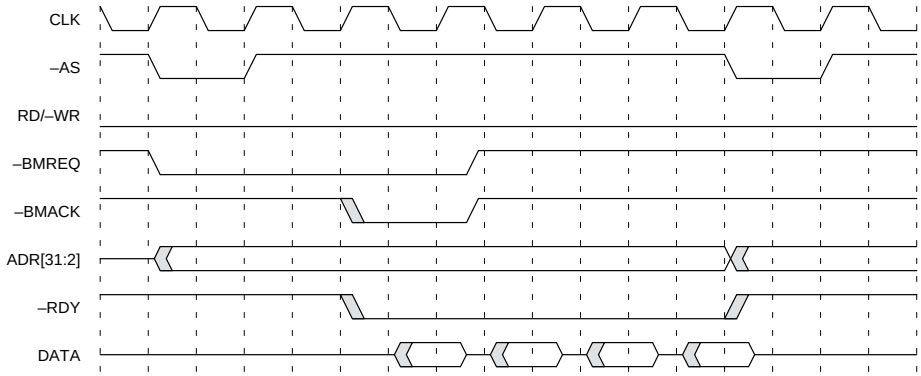


Figure B4-4. DMA Burst Mode, Write Portion

4.3 Parity

The MB86932 provides parity generation/checking for the 32-bit external data bus. Parity can be enabled/disabled for specified address ranges by setting/clearing bits in the Wait-State Specifier Register (see section on that register, below). Parity can be set even or odd by setting bit 0 in the System Support Control Register: set to 1, odd parity is generated/checked; set to 0, even parity is generated/checked. On reset, the value of this bit is cleared to 0.

Parity is generated/checked for every byte of data (resulting in four parity bits). If parity is odd, the parity bit is set to 1 when there are an odd number of 1's in the data; if parity is even, the parity bit is set to 1 when there are an even number of 1's in the data. When

enabled, parity is generated for all writes to external memory. Incoming parity is checked only for the address ranges for which the “PE” bit in the corresponding Wait-State Specifier Register is set to 1. If a parity error is detected on an instruction fetch, an instruction_memory_exception occurs, and bit 8 in the Instruction Fault Status Register is set (see TLB section). If a parity error is detected on a data fetch, a data_ memory_exception occurs, and bit 9 in the Data Fault Status Register is set (see TLB section). The parity bits will have a longer setup/delay time than the other data bits.

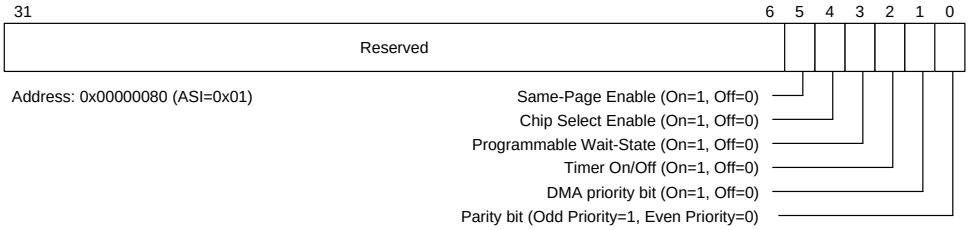


Figure B4-5. System Support Control Register

Note:			
	PARITY [0]	Correspond to	D [7:0]
	PARITY [1]	Correspond to	D [15:8]
	PARITY [2]	Correspond to	D [23:16]
	PARITY [3]	Correspond to	D [31:24]

4.4 Wait State Specifier Register

4.4.1 Purpose

The Wait-State Specifier Register (WSSR) format on the MB86932 has been changed from that on the MB86930 to accommodate the burst mode bus transaction using internal –READY and Parity generation/checking.

4.4.2 Format

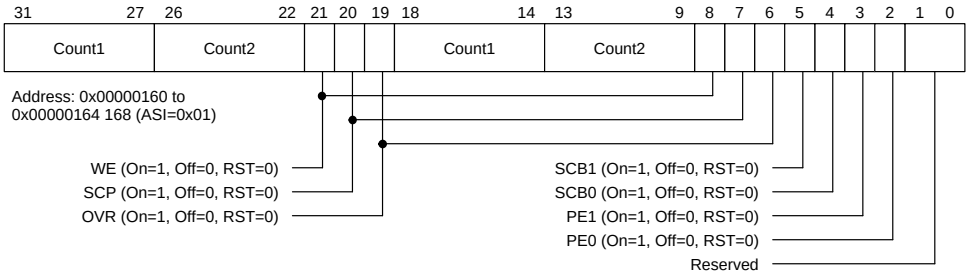


Figure B4-6. Wait State Specifier Register

The bits in the WSSR can have two different meanings depending on whether burst mode is enabled or disabled.

4.4.3 Same Page Mode

Burst mode disabled or burst mode enabled and –BMACK not asserted for this region.

Count1: Count1 +1 is the number of wait states inserted before internal –READY is asserted, under the following conditions: SCP=0, and current access is not in the same page as the previous access.

Count2: Count2 +1 is the number of wait states inserted before internal –READY is asserted, under the following conditions: SCP=0, and current access is in the same page as the previous access.

WE: Wait Enable, enables or disables the internal wait state generation for the individual address range. IF WE is 1 SCP must be 0.

SCP: If this bit is 1 the internal –READY is generated in the same cycle when an access is started. All accesses to external memory in this address range will be single cycle. IF SCP is 1 WE must be 0.

OVR: Allows the system to terminate the memory operation before the internally specified time. If the OVR bit is set to 1, and the external hardware asserts external –READY signal, the wait state generator will stop counting and will wait for the next transaction.

SCB: Unused; should be 0.

PE: Enable checking of Parity. PE1, PE0 correspond to address ranges for WSSR[31:19] and WSSR[18:16] respectively.

4.4.4 Burst Mode

Burst mode enabled and –BMACK is asserted.

Count1: Count1 +1 is the number of wait states inserted before internal –READY is asserted, for the first access of a burst mode transfer.

Count2: Count2 +1 is the number of wait states inserted before internal –READY is asserted, for the 2nd, 3rd and 4th access of a burst mode access if SCB=0.

WE: Wait Enable, enables or disables the internal wait state generation for the individual address range. If WE is 1, SCP must be 0.

SCP: If this bit is 1, the internal –READY is generated in the same cycle when an access is started. All accesses to external memory in this address range will be single cycle. If SCP is 1, WE must be 0.

OVR: Allows the system to terminate the memory operation before the internally specified time. If the OVR bit is set to 1, and the external hardware asserts external –READY signal, the wait state generator will stop counting and will wait for the next transaction.

SCB: If this bit is 1, in the burst mode all accesses after the first access take a single cycle. If this is 1, Count2 is ignored. SCB1 and SCB0 correspond to address ranges for WSSR[31:19] and WSSR[18:6] respectively.

PE: Enable checking of Parity. PE1, PE0 correspond to address ranges for WSSR[31:19] and WSSR[18:6] respectively.

Table B4-2: RESET State

WSSR reset state for –CS[1] to –CS[5]:	WSSR reset state for –CS[0]:
Count2=0	Count2=31
Count1=0	Count1=31
WE=0	WE=1
SCP=0	SCP=0
SCB=0	SCB=0
OVR=0	OVR=1
PE=0	PE=0

4.5 ROM Interface

4.5.1 Purpose

The data bus of the MB86932 can be configured upon reset to 8- and 16-bit bus modes as well as the standard 32-bit mode. This flexibility accommodates those cases in which boot code resides in PROMs organized as blocks of bytes or halfwords.

4.5.2 Features

Bus Configuration: the data bus configurations are fixed to specific segments of the bus:

- 8-bit mode: D[7:0]
- 16-bit mode: D[15:0]
- 32-bit mode: D[31:0]

4.5.3 Bus Configuration on Reset

Two external pins, –BMODE16 and –BMODE8 are used to determine the bus configuration. The two bus configuration pins have weak pull-ups, so that if unconnected, the bus configuration will default to a 32-bit bus.

(reserved): –BMODE16=0, –BMODE8=0

8-bit mode: –BMODE16=1, –BMODE8=0

16-bit mode: –BMODE16=0, –BMODE8=1

32-bit mode: –BMODE16=1, –BMODE8=1

4.5.4 System Interface

In order to minimize external "glue logic" required for interfacing to the 8- or 16-bit bus, the BE bits are encoded to reflect the two LSBs of a byte address or the LSB of a halfword address. Therefore, the ADR[31:2] and selected –BE bits can be concatenated to form a complete address for a non-32 bit bus mode.

Table B4-3: System Interface BE Bits

Bus Mode	Byte	BE[0:3]
8-bit bus	0	0000
	1	0001
	2	0010
	3	0011
16-bit bus	0 & 1	0000
	2 & 3	0010

8-bit bus mode address= {ADR[31:2], –BE[2], –BE[3]}
16-bit bus mode address={ADR[31:2], –BE[2]}

–CS[0], which is enabled on reset, and the internal –READY generation logic, can be used to minimize any glue logic required to interface to the PROM. On reset, the wait state generator, corresponding to –CS[0] for internal –READY generation, is set to 32 cycles. Later on in the boot code, the wait state generator can be changed to a more appropriate value.

4.5.5 PROM Address Space

The PROM address space is defined by the –CS[0] address-range specifier. On reset, the –CS[0] address range defaults to 32K bytes (starting address=0x0), and the ASI is initialized to 0x9. The PROM address range can be changed later using the mask bit

register associated with –CS[0]. An example of the supervisor address space (ASI=0x9) memory map is shown below:

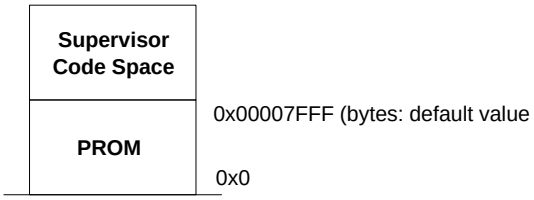


Figure B4-7. Supervisor Address Space (ASI=0x9) Memory Map

Any memory access from the PROM address space, in a non-32 bit mode, will make the –BE bit encodings reflect the LSBs of a byte/halfword address. Furthermore, the fetched bytes/halfwords will be assembled into a 32-bit word. On the other hand, any access from the non-PROM address range will result in a normal, 32-bit memory access.

4.5.6 Load/Stores

One of the functions of the boot code is to set the processor and system configuration. This might involve loading system parameters from PROM, loading data from memory mapped I/O, and storing data to non-PROM address space. All loads from the PROM address space behave the same way as instruction fetches, in that, for a non-32 bit bus mode –BE , bit encoding and word assembly are done. Loads from a non-PROM address space behave in the normal (32-bit) manner. In order to meet the –BE AC timing, the –BE bits on the MB86932 need to be all 0's for all types of loads—word, halfword, and byte—from the non-PROM address space. This requires a functional change from the current specification of the MB86930's –BE bits, which reflect the byte information for loads. This change does not cause a problem, since the processor fetches a full 32-bit word on a load, and the IU selects the byte appropriately. As on the MB86930 –BE bits should be ignored for 32-bit loads.

Since stores to the PROM will never occur, for all stores, regardless of address space, the –BE bits will reflect the byte information of the store. Therefore, byte and halfword stores to the PROM address space becomes meaningless, since the –BE[2] and –BE[3] bits no longer reflect the byte address. Furthermore, store word operations to the PROM address space will not result in a dis-assembly process for a non-32 bit bus mode. Since stores to PROM address space are not disabled, the user would have to qualify –CS[0] with the R/–W signal to use it as a PROM chip select signal. This will not be necessary if the user can be sure that a store to PROM space never occurs.

A summary of the –BE[0:3] bit behavior for loads from the PROM address space is shown below. For all load instructions (byte, halfword, word), a full 32-bit fetch occurs.

For example, in the 8-bit bus mode, four bytes will be fetched for all loads, and the BE bits will sequence with the proper 2 LSBs of the byte address.

Table B4-4: Load –BE[0:3] Bit Behavior

Bus Mode	Operation	BE[0:3] in PROM space
8-bit bus	Loads (all)	0000=>0001=>0010=>0011
16-bit bus	Loads (all)	0000=>0010
32-bit bus	Loads (all)	0000

4.5.7 Burst Mode

Since speed is not a critical issue when executing boot code out of PROM, and because there is no industry-wide standard for a burst-mode EPROM interface, burst-mode interface is not supported for accesses from PROM address space. When the system has a 8/16 bit memory being used for boot code, it should not assert –BMACK for any accesses to –CS0.

4.5.8 Memory Exception

Any memory exception that occurs during a fetch from the PROM address space in a non-32 bit bus mode will be held off until the entire word is fetched.

4.5.9 Bus Request

Any bus request happening during the non-32 bit bus mode fetch will not be recognized until the end of the complete 32-bit fetch operation.

4.5.10 Timing

Timing examples for the 8- and 16-bit bus modes with 1 wait-state memory are shown below. Note that –AS is asserted at the beginning for one cycle.

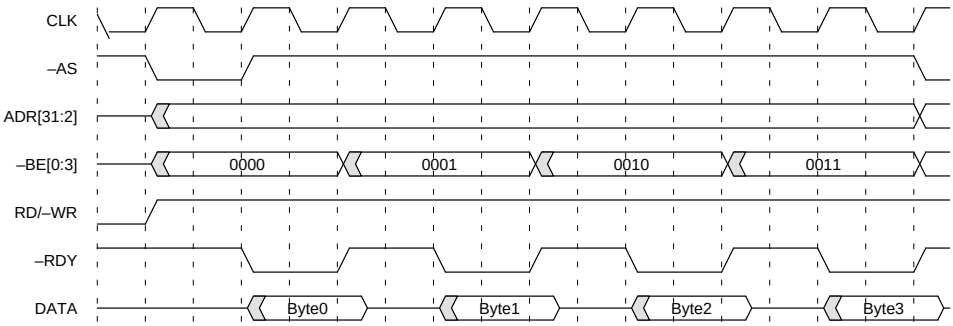


Figure B4-8. 8-bit Bus Mode (1 Wait State)

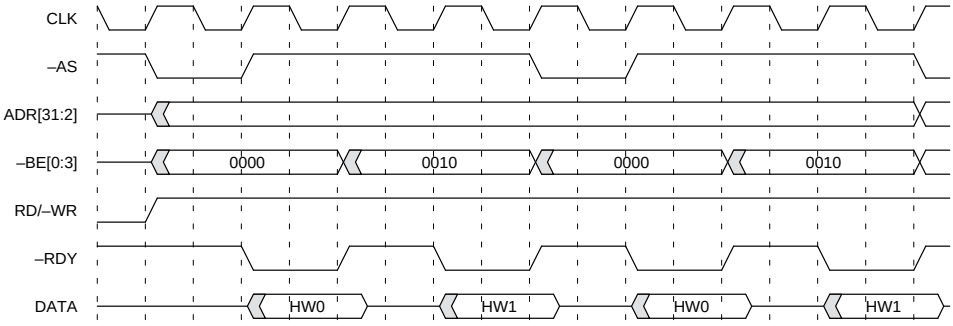


Figure B4-9. 16-bit Bus Mode (1 Wait State)

4.6 Processor Bus Request

4.6.1 Purpose

When the bus is released in response to an external device's request for the bus (by asserting –BREQ), the MB86932 processor cannot access the bus as long as the bus request signal remains asserted. An external bus arbiter may never be aware that the processor needs the bus back. To remedy this problem, a processor bus request signal is

asserted whenever the external bus is required by the processor. The external bus arbiter then can release the bus to the processor requesting it. Also, in a bus-based multiprocessor system, a processor bus request signal is useful to the external bus arbiter in deciding which processor requires the bus.

4.6.2 Features

–PBREQ pin: An external pin is used to output the processor bus request signal, –PBREQ. The –PBREQ will be asserted whenever the MB86932 requires the bus while the bus is granted to an external device. The external device using the bus can monitor the –PBREQ signal, and remove the –BREQ signal at an appropriate time. An example of the –PBREQ timing is shown in the figure below:

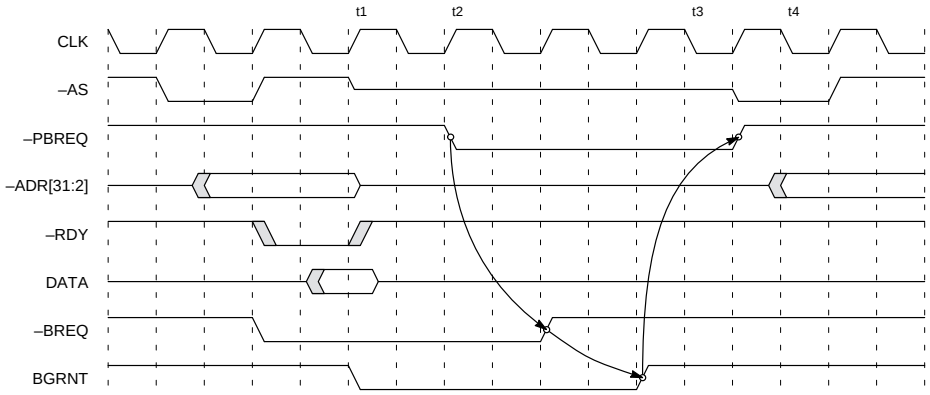


Figure B4-10. Example of –PBREQ timing

In the example above, the bus is released at the beginning of cycle t1 in response to an external bus request. At t2, –PBREQ is asserted because of a pending bus cycle in the processor. The external bus arbiter de-asserts –BREQ, and returns the bus to the processor. –PBREQ remains asserted until the end of the cycle t3. At t4, the processor drives the bus.

4.7 BIU Timing

4.7.1 Effect of TLB

Since the TLB can be used with the cache turned off, a one-cycle delay is introduced at the beginning of each memory operation to complete the TLB translation. For cache-on

cases, the TLB does not introduce an additional delay since address translation occurs during the one-cycle already available for cache hit/miss detection. The first figure below shows the timing for the cache-off, TLB-off case; the second figure shows the timing for the cache-off, TLB-on case. Note in the second figure the one-cycle delay for each new memory operation.

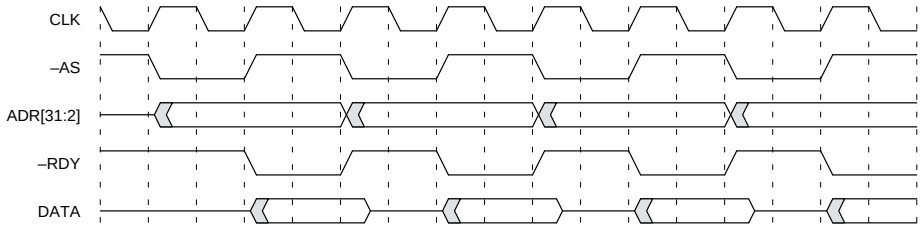


Figure B4-11. Cache=off, TLB=off (1 wait state)

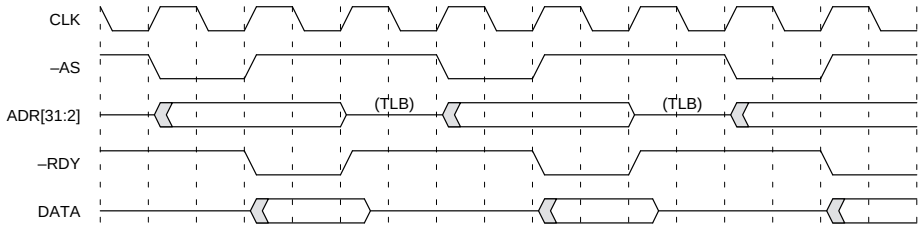


Figure B4-12. Cache=off, TLB=on (1 wait state)

4.8 BIU Priorities

In general the following hierarchical rules apply when multiple requests are made to the bus interface unit:

- The bus cycle currently in progress will complete.
- If there is a pending external bus request, the bus will be granted to the external requestor.
- If there is a pending DMA request, the bus will be granted to the DMA controller.
- If the write buffer is full, the buffer will be emptied.
- If there is a pending load or store operation it will be serviced.
- If there is a pending request for an instruction it will be fetched.

- If the prefetch buffer is empty, a prefetch cycle will be initiated.

Note that bit1 in the System Support Control Register can be used to allow the IU to "steal" cycles from the DMA. When this bit is set the DMA will de-assert its request after each datum is moved. When cleared the DMA will keep the bus until the whole DMA transaction has completed.

CHAPTER

B5

MB86932 DMA

5.1 Overview

The Direct Memory Access Controller (DMAC) module provides high-speed memory-to-memory and memory-to-peripheral data transfers. The DMAC executes independently of the CPU, making it possible for the processor to execute from cache while DMA transfers are taking place. The DMAC operates on physical addresses.

The DMAC supports two independent DMA channels concurrently. It supports byte, half-word, word and quad-word transfers. The DMA mechanism provides three different methods of performing DMA transfers: Single transfer, Demand transfer, and Block transfer. Single transfer and Demand transfer use the DMA request (–DREQ) and DMA acknowledge (–DACK) signals to synchronize transfers with external devices. Block transfers do not use –DREQ and –DACK, they are typically used to transfer data from memory to memory.

“Fly-by” transfer mode is supported for high speed DMA transfers. In this mode, a single bus transaction transfers the data from source to destination. “Flow-Thru” transfer mode is also supported. In this mode, two bus transactions, a read followed by a write, need to be performed to complete the transfer of data from source to destination.

The DMA channels can be configured to perform a single buffer transfer, or to operate in the buffer-chaining mode. The buffer-chaining mode is provided to simplify operations such as scatter/gather. In this mode, the DMAC is configured with a series of

descriptors in memory. Each descriptor describes a single buffer transfer, which is part of the complete DMA transfer.

The two figures that follow give, respectively, an overall picture of the relationship of the DMAC to other major functional components of the MB86932, and a detailed picture of the flow within the DMAC.

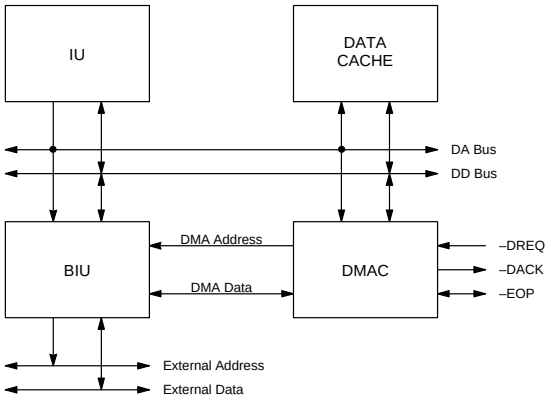


Figure B5-1. Relation of DMAC to Other Major Components

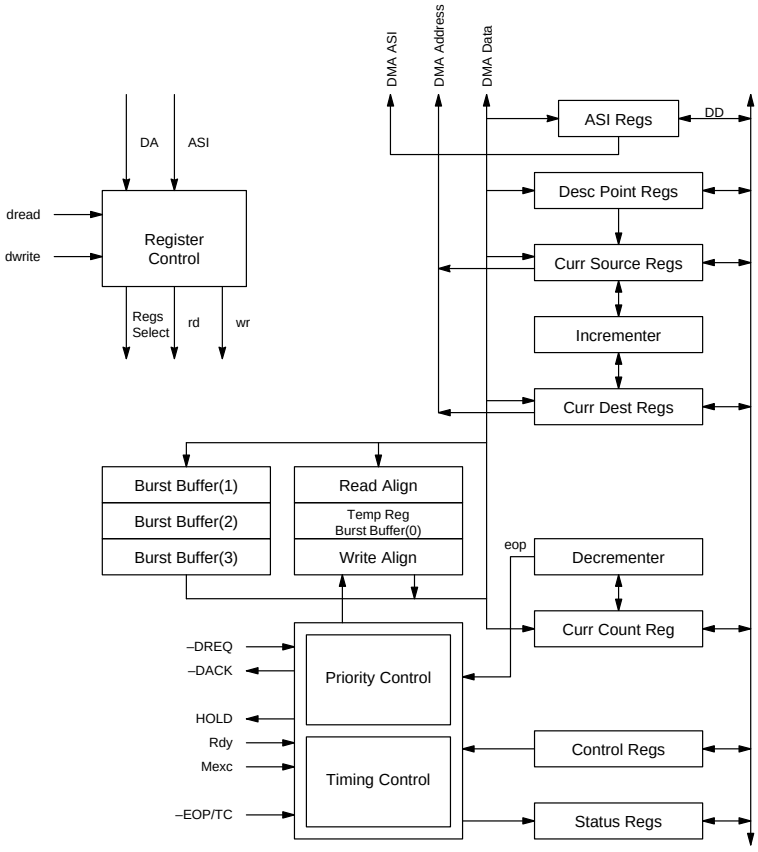


Figure B5-1. DMA Block Diagram

5.2 Programmer's Model

Table B5-1: DMA Signal Descriptions

Signal	Function
–DREQ1 / –DREQ0	**DMA REQUEST (I):** This input signal indicates that an external device is requesting DMA transfer. It is an edge-sensitive signal for single transfer, and a level-sensitive signal for demand transfer.
–DACK1 / –DACK0	**DMA ACKNOWLEDGE (O):** This output signal is sent to the external device to acknowledge the DMA request, and is active when the requesting device is accessed.
–EOP1 / –EOP0	**END OF PROCESS (I/O):** This pin is used as input when an external device wants to cause the DMA process to terminate. It functions as output when the byte count reaches zero. When not active, –EOP output will be tristated. For signalling the Terminal Count (TC) , –EOP will be pulled down, and then be pulled up for one cycle. A high impedance internal pull up is used to hold the signal high when –EOP is tristated. The –EOP issued by the DMAC can be used as input to the interrupt controller. If –EOPx is asserted by the external device, channel x will be disabled. Reprogramming is needed to enable a channel.

Six pins are dedicated to the DMAC, three for each channel. In the table above, the pin number corresponds to the channel number. For example, the –DREQ0 pin is the request pin for channel 0.

5.2.1 DMA Priority

The DMA Priority Bit in the System Support Control Register can be programmed to indicate whether the DMA is to release the bus for one clock cycle so that the IU can use it. When this bit is set, the DMAC will deassert the HOLD signal to the BIU for one clock cycle after a DMA entry has been transferred. In this way, the IU can steal a bus cycle when service is needed. When this bit is cleared, the DMA blocks the IU from using the BUS until the whole DMA transaction completes.

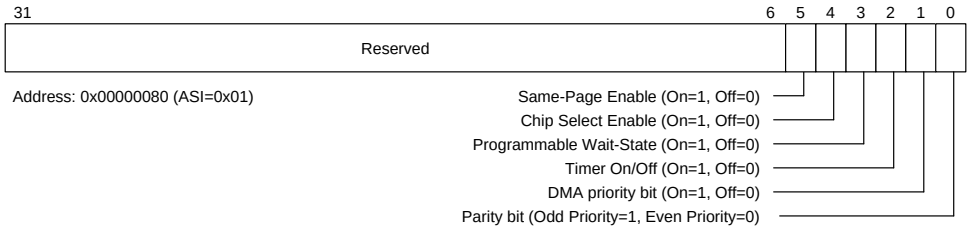
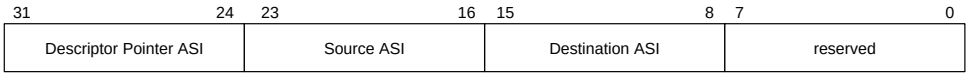


Figure B5-3. System Support Control Register

5.2.2 DP/Source/Destination ASI Register



Address:0x00000180 (DMA0) (ASI = 0x01)
0x000001A0 (DMA1)

Figure B5-4. DP/Source/Destination ASI Register

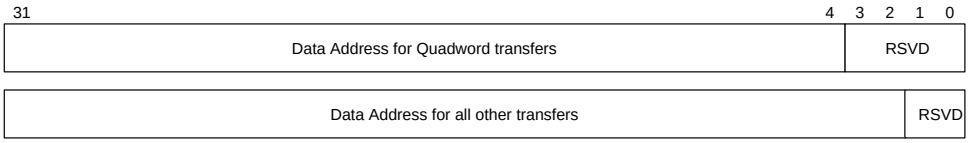
Bits 31-24: Description Pointer ASI (DP ASI)—ASI of the Descriptor Pointer, a register used in buffer-chaining mode. It points to the next element of the linked list whose elements describe the source and destination of the DMA transfer.

Bits 23-16: Source ASI—ASI of the Current Source Address Register, which is described below.

Bits 15-8: Destination ASI (Dest ASI)—ASI of the Current Destination Address Register, which is described below.

Bits 7-0: Reserved

5.2.3 Current Source Address Register



Address:0x00000184 (DMA0) (ASI=0x01)
0x000001A4(DMA1)

Figure B5-3. Current Source Address Register

The Current Source Address Register is used to address memory accesses in flyby mode, and to hold the source data address in flowthru mode. It contains one 30-bit (31:2) word-aligned address. For byte, halfword, and word transfers, all 30 bits (31:2) are used; for quadword transfers, only 28 bits (31:4) are used. Bits beyond the current address field are ignored. The CSA Register value is updated after a transfer in the read phase has been done, and points to the next location to be transferred. Note that in flyby mode, a DMA transfer has just one Read/Write phase; in flowthru mode, a DMA transfer has one read phase, one write phase, and an intervening idle clock cycle.

5.2.4 Current Destination Address Register

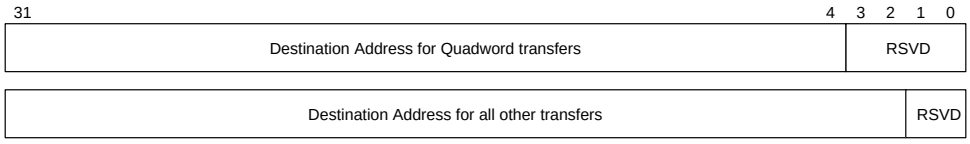


Figure B5-8. Current Destination Address Register

The Current Destination Address Register is not used in flyby mode; it holds the destination data address in flowthru mode. It contains one 30-bit (31:2) word-aligned address. For byte, halfword, and word transfers, all 30 bits (31:2) are used; for quadword transfers, only 28 bits (31:4) are used. Bits beyond the current address field are ignored. The CDA Register value is updated after a transfer in the write phase has been done.

5.3.5 Current Byte Count Register

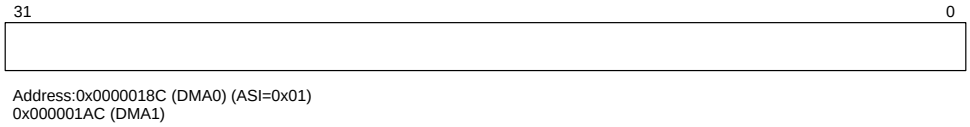
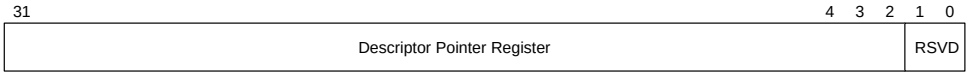


Figure B5-7. Current Byte Count (CBC) Register

The CBC register indicates the number of bytes of data still left to be transmitted. The value of the data should be programmed to be one less than the actual number of bytes to be transmitted. For example, to transfer two words, this register should be loaded with the value "7". The value will be decremented at the beginning of the DMA transfer cycle by the number of bytes involved in the transfer, regardless of the unit in terms of which the transfer is specified (half-word, word, etc.)**.** The Byte Count Register is updated only in the Read phase, not in the Write phase; it is updated at the beginning of the transfer.

5.2.6 Descriptor Pointer Register



Address:0x00000190 (DMA0) (ASI=0x01)
0x000001B0 (DMA1)

Figure B5-8. Descriptor Pointer (DP) Register

Used in Chaining Mode, the DP Register points to the next element of the linked list. Successive elements of the list describes the source and destination of successive buffers to be transferred.

5.2.7 Channel Control Register

Bits 31 to 16 are reserved, ignored on a Write, and Read as zero. The entire register is reset to zero. Note that the two channel control registers are not identical: the HPC and SW bits in the channel 0 register are global, while the same bits in the channel 1 register are reserved, and read as undefined.

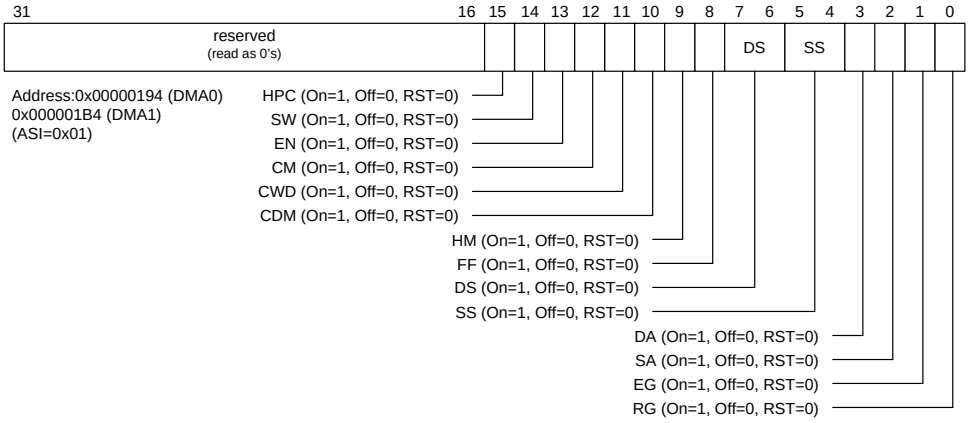


Figure B5-9. Channel Control Register

The Channel Priority Switch Mode bit “SW” and the High Priority Channel bit “HPC” of the channel 0 Control Register determine the priority setup of the DMA Controller. These two global bits should be programmed only when both channels are disabled.

Bits 31-16 Reserved

Bit 15: High Priority Channel (HPC)—0 if channel 0 has high priority; 1 if channel 1 has high priority. (The HPC should be programmed to specify the channel that has high priority at the outset; if SW=1, it will be updated to show the current high-priority channel as the DMA transfer progresses. Note that this bit exists only in the channel 0 control register; the corresponding bit in the channel 1 control register is reserved, and read as undefined.

Bit 14: Channel Priority Switch Mode (SW)—0 if fixed, 1 if switchable. (If 0, the HCP is fixed, and specifies a prechosen higher priority channel; if switchable, the HCP will be updated to whichever channel is not currently being serviced.) Note that this bit exists only in the channel 0 control register; the corresponding bit in the channel 1 control register is reserved, and read as undefined.

Bit 13: Enable [Start] DMA (EN)—0 if disable channel, 1 if enable. (The DMA channel can be enabled by writing 1 to this field, and is reset by the hardware when the channel enters the disabled state. In Internal Request mode (see RG field), a 1 here means Start DMA; in External Request mode, a 1 here means Accept External DMA request.)

Bit 12: Chaining Mode (CM)—0 if reprogramming, 1 if buffer chaining.

Bit 11: Chaining Wait Mode (CWM)—0 if Chaining Wait Function disable, 1 if enable. (Decides whether next chaining descriptor is to be read.)

Bit 10: Chaining Debug Mode (CDM)—0 if assert –EOP only after the whole Chaining transfer, 1 if assert –EOP after each buffer transfer

Bit 9: Transfer/Handshake Mode (HM)—0 if Single Transfer, 1 if Demand Transfer. (Applies only to external request; for internal program request, DMAC supports block transfer mode only.)

Bit 8: Flyby/Flowthru (FF)—0 if Flyby (single address), 1 if Flowthru (Dual Address).

Bits 7-6: Destination Size (DS)—00 if word, 01 if byte, 10 if halfword, 11 if quadword.

Bits 5-4: Source Size (SS)—00 if word, 01 if byte, 10 if halfword, 11 if quadword.

Bit 3: Destination Addressing (DA)—0 if increment, 1 if hold.

Bit 2: Source Addressing (SA)—0 if increment, 1 if hold.

Bit 1: External Control Option (EC)—0 if source request, 1 if destination request.

Bit 0: Request Generation (RG)—RG=0 if internal request, 1 if external request.

5.2.8 Channel Status Register

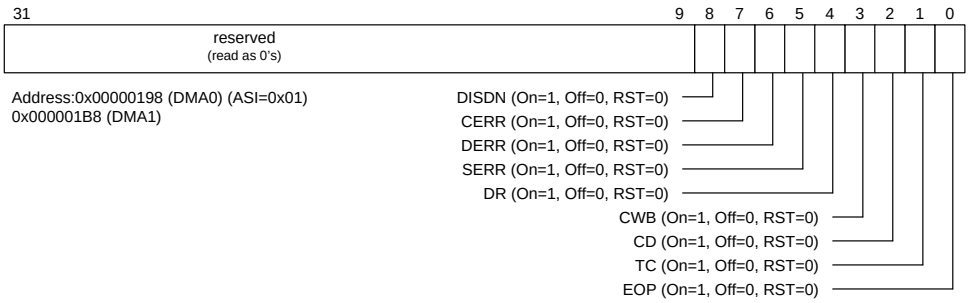


Figure B5-10. Channel Status Register

Bits 31-9: This register is shown as having only 9 bits because these bits are reserved, ignored on a Write, and Read as zero. The entire register is reset to zero.

Bit 8: Disable Done (DISDN)—the user can disable the DMA channel by writing 0 to the Enable bit of the Control Register. This bit will be set when the channel has been effectively software-disabled.

Bit 7: Chaining Error on DMA Transfer (CERR)

Bit 6: Destination Error on DMA Transfer (DERR)

Bit 5: Source Error on DMA Transfer (SERR)

Bit 4: DMA Request presented (DR)—A DMA request is pending.

Bit 3: Chaining Wait (CWB)—If the Chaining Wait Mode in the Control Register has been set, this status bit will be set after each buffer has been transferred. The Chaining Descriptor fetch will not be executed. After the program redoes the setup for this channel, and clears this status bit, the DMA will proceed with the new register setup.

Bit 2: Chaining Done (CD)—The whole chain of data buffers have been successfully transferred; set up in chaining mode.

Bit 1: Terminal Count (TC)—A data buffer has been successfully transferred. It will be set when termination of transfer is reached for nonchaining mode and chaining debug mode.

Bit 0: End of Process, external (EOP)—Channel transfer stop due to external –EOP signal.

5.2.9 Channel Initialization

The DMA Control has two transfer modes: 1) Single Buffer Transfer Mode, and 2) Buffer Chaining Mode. Each mode has its own programming requirements.

To initialize the DMA Channel for Single Buffer Transfer Mode, the user must program these registers:

- ASI Register
- Current Source Address Register
- Current Destination Address Register
- Current Byte Count Register
- Channel Control Register

After programming these registers, the user writes the start (enable) bit of the Channel Control Register to enable the Channel.

To initialize the DMA Channel for Buffer Chaining Mode, the user must program the registers listed above for Single Buffer Mode transfers, and in addition must program the Descriptor Pointer (DP) Register. The DP points to the next element of the chaining list for the buffers to be transferred. After the channel finishes transferring each block, it will spend five data access cycles to set up the DP/Source/ Destination ASI Register, the Current Source Address Register, the Current Destination Address Register, and the Current Byte Count Register. The last chaining cycle is used to get the pointer and put it in the Descriptor Pointer Register.

When TC happens, the DMA will load the chaining information pointed to by the DP, and the DMA process continues. An external –EOP will disable the channel.

In chaining mode, whether block or demand transfers are being carried out, a channel that has reached TC will load the chaining block descriptor, and the DMA Controller will see if a request from the high priority channel is outstanding. If it is, the DMAC will suspend the next transfer of the present sequence, and release the bus to the high priority channel. For example: assume that priority switching mode is in effect; channel 0, the original high priority channel, is in chaining mode; and channel 1 is in reprogramming mode. If both channels get –DREQ asserted, channel 0 will be serviced first. WHen TC is reached, DMAC will load the information for the next transfer block; the outstanding request from channel 1 will be noted, and—because channel 1 is the high priority channel—its request will be serviced now.

5.2.10 Buffer Chaining Data Structure

- PSDASI (Descriptor, Source, and Destination ASI)
- SA (Source Address)
- DA (Destination Address)

- BC (Byte Count)
- NPTR (Next Buffer Descriptor Pointer); a NULL pointer, 0000, indicates the end of the block buffer list.

5.2.11 DMA Initialization

DMA operations can be initiated by either software request or hardware request. A software request is made by clearing the Request Generation bit and setting the DMA Enable bit. A hardware request is made by setting the Request Generation bit and the DMA Enable bit, and then causing the assertion of an external –DREQ.

When the CPU clears the Request Generation bit and sets the DMA Enable bit, the software-initiated DMA starts immediately. A hardware request is started only when –DREQ is asserted while the DMA Enable bit is set. –DREQ is edge-sensitive for Single Transfer Mode, level-sensitive for Demand Transfer Mode. For Demand Mode to complete a whole buffer block, –DREQ must be asserted until –EOP is asserted. –EOP can be asserted by the DMA Controller or an external device.

5.2.12 Basic DMA Timing

1. For a single transfer, the DMAC will sample –DREQ for the next DMA request after –DACK is asserted. That is, DMAC will try to detect the edge that signals such a request; an edge asserted between that which caused the last transfer and the assertion of –DACK will be ignored. Even if an edge is detected before the DMAC releases the bus, the DMAC will still release the bus and then request it again.
2. –DACK will toggle during the read or write cycle to enable the peripheral device. Ready (from BIU) will be used to deassert the –DACK.
3. –DACK is used for handshaking with a peripheral device to deassert the –DREQ for single transfer mode. –EOP(TC) is used for handshaking with a peripheral device to deassert the –DREQ for demand transfer mode.
4. TC will be used to enable the reloading of the address/count to the current registers to initialize the set up for a buffer chaining transfer. External –EOP will disable the DMAC channel in chaining mode, and leave the state of the channel as it was.

5.2.13 Error Conditions

Memory Access Exceptions:

- Source Transfer Exception
- Destination Transfer Exception
- Chaining Exception

When an Error condition occurs, the relevant bits in the Status Register will be set up, and –EOP will be asserted.

When a memory-exception occurs, –EOP will be asserted one cycle later. This –EOP can be used as input to the interrupt controller. The –EOP due to a memory exception can be deasserted by clearing the status bit of the corresponding exception.

For quad-word transfers, if an exception occurs during the read phase, DMA will still finish all four reads, but will not go into the write phase. If an exception occurs during the write phase, DMA will complete all four writes.

For transfers other than quad-word, the DMA will stop immediately after the exception occurs.

5.3 External Interface

5.3.1 Transfer Protocols

Single Transfer Mode

In the Single Transfer Mode, one data entry transfer from source to destination is performed by the DMAC at a time. The –DREQ input is arbitrated according to the channel priority decisions made by the user. The channel with the DMA request will signal the BIU for bus service. After a DMA data entity has been transferred, control of the bus will be released. Transfers continue in this manner until the Byte Count expires, or until external –EOP is found active. Since the –DREQ is edge-sensitive for single transfers, a –DREQ pulse will cause only one transfer, no matter what its length. The channel will request the bus for each DMA transfer. Bus control is released between each transfer and the next. The DMAC will sample the next –DREQ edge for a DMA transfer request after –DACK is asserted. A new request edge coming before –DACK has been asserted will be ignored. A timing diagram for single transfer mode is given below in Figure B5-11 This diagram shows two consecutive DMA transfers. A sample High and then Low of –DREQ constitutes an edge request for a transfer. The last block transfer is accompanied by –EOP. –R/W is asserted High in flyby mode for a destination transfer—that is, one where data will flow from memory—and asserted Low for a source transfer, where data will flow to memory. In Figure B5-13 below, showing a quadword transfer taking four data cycles. The last DMA transfer is accompanied by EOP.

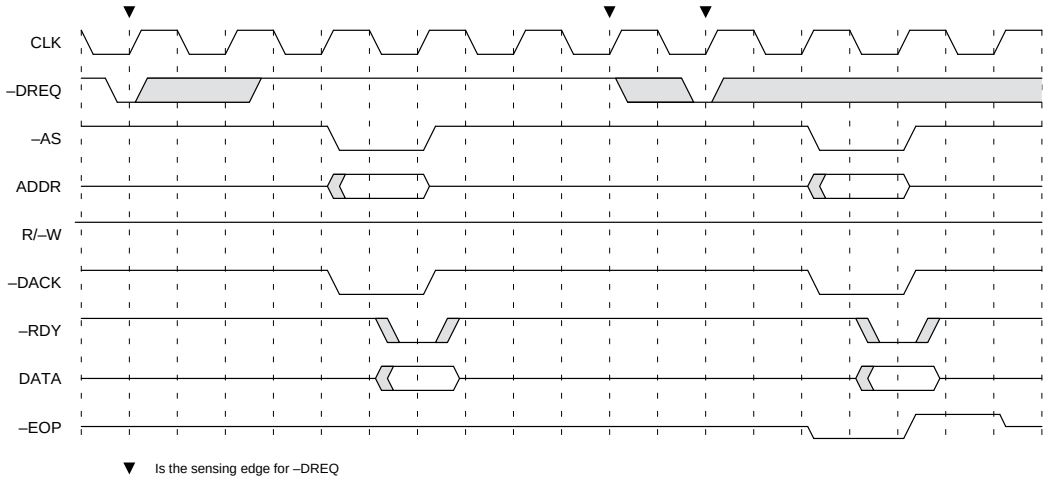


Figure B5-11. Single Transfer, Edge-Sensitive, Flyby (R/–W high)

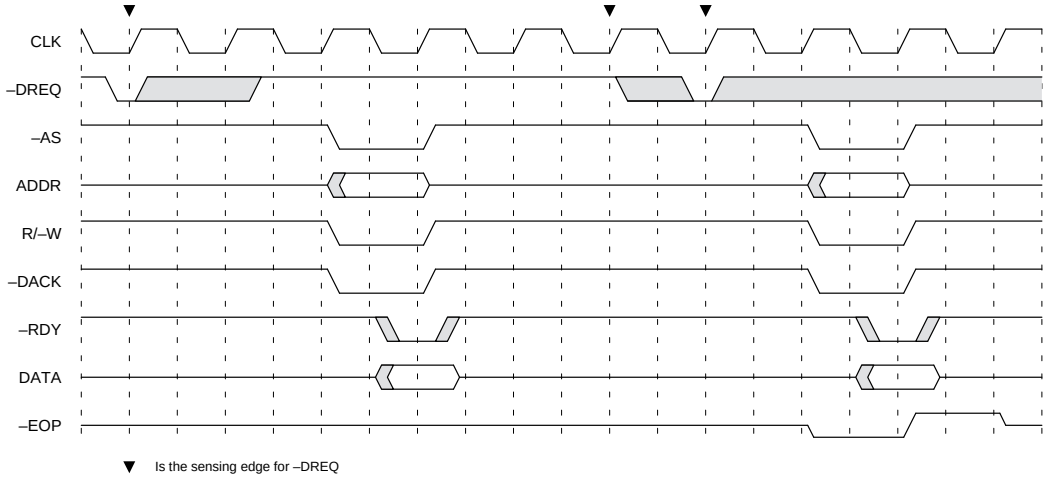


Figure B5-12. Single Transfer, Edge-Sensitive, Flyby (R/–W low)

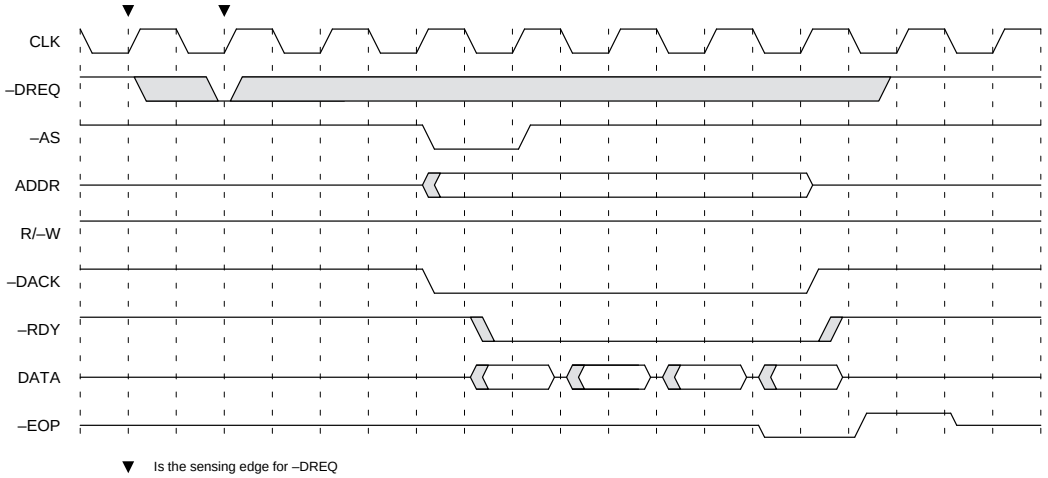


Figure B5-13. Single Transfer, Edge-Sensitive, Flyby, Quadword (R/–W high)

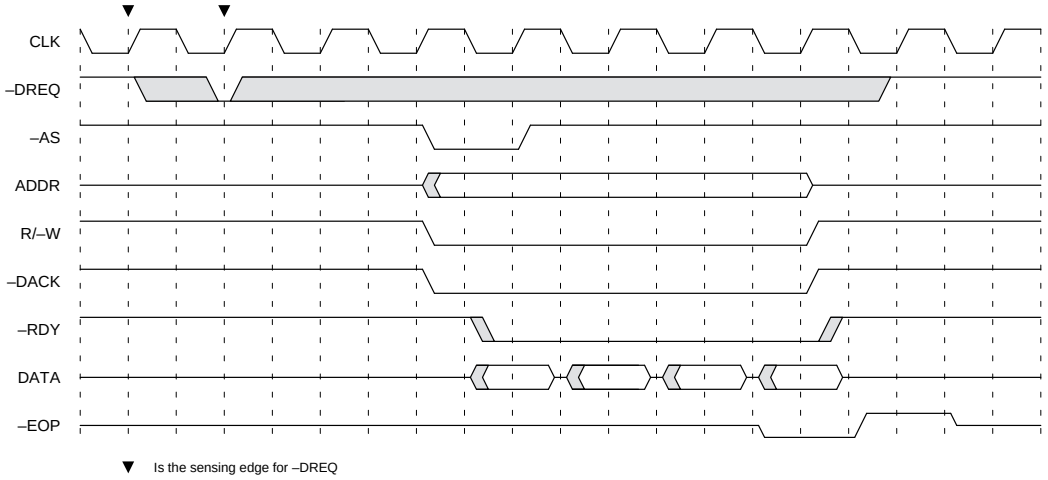


Figure B5-14. Single Transfer, Edge-Sensitive, Flyby, Quadword (R/–W low)

Block Transfer Mode

Block transfer is initiated by software request. In this mode, the CPU starts the DMA action by setting the Start bit of the control register. The transaction will continue until the Terminal Count (TC) happens, or until –EOP is asserted by the external device.

Block transfer mode can be used for either flowthru or flyby transactions. For flyby transactions, the DMAC will assert and then deassert the –DACK for each transferred datum.

A timing diagram for software-initiated block transfer is shown in Figure B5-15 below. The timing is the same as that for demand transfer mode, except that the request is set by software. The transfer will begin two cycles after the channel control register has been written.

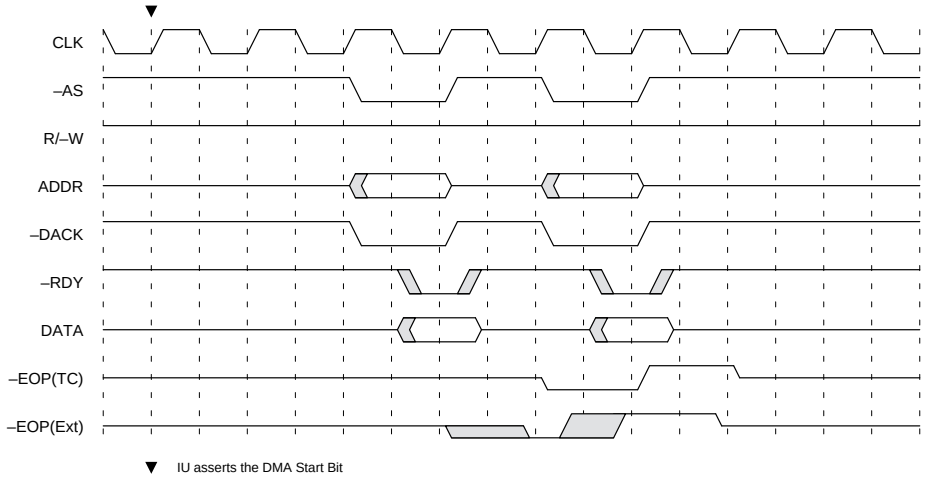


Figure B5-15. Block Transfer, Flyby (R/–W high)

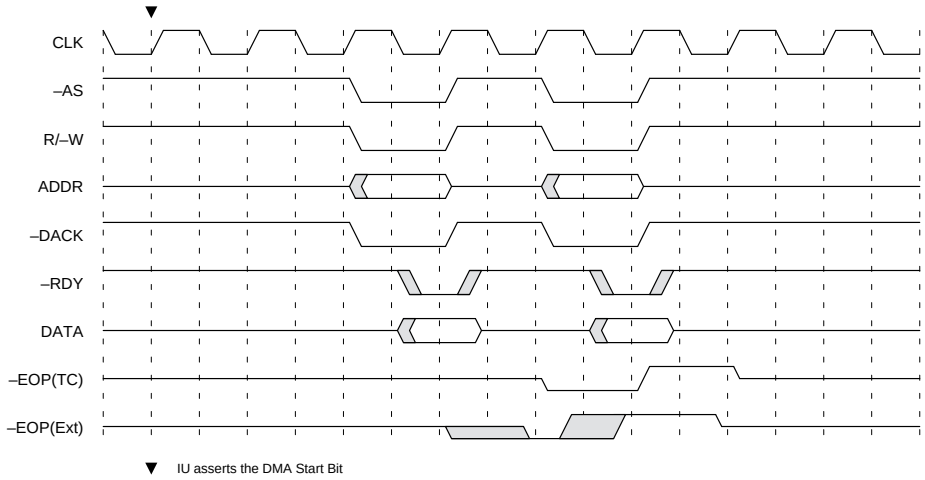


Figure B5-16. Block Transfer, Flyby (R/–W low)

Demand Transfer Mode

Demand Transfer Mode provides flexible handshaking procedures during the DMA process. A Demand Transfer is initiated by an external level-sensitive DMA request (–DREQ). The next request will be sampled after the preceding transfer request has been completed. The process continues until (a) the external device deasserts the –DREQ, (b) the byte count (TC) expires, or (c) an external –EOP is encountered. A timing diagram for demand transfer is shown below in Figure B5-17. When a request for a demand transfer is made, the DMAC will look at the –DREQ to see if any request is pending.

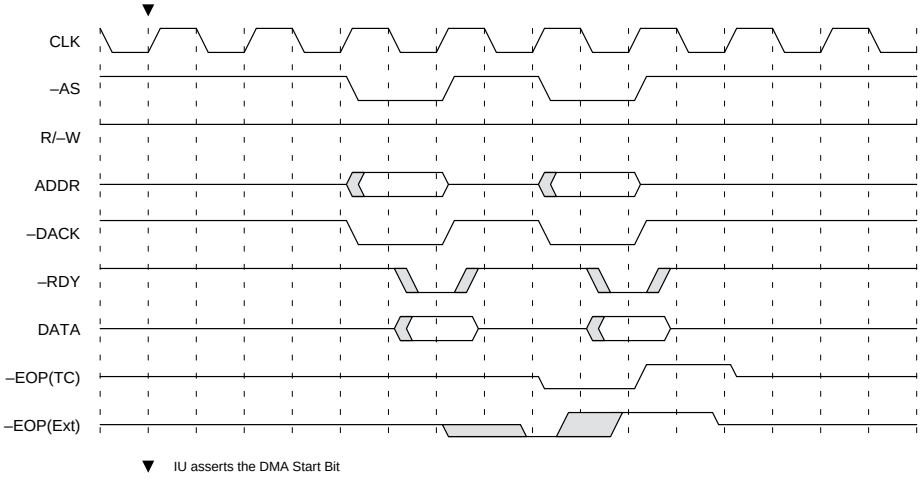


Figure B5-17. Demand Transfer, Flyby (R/–W high)

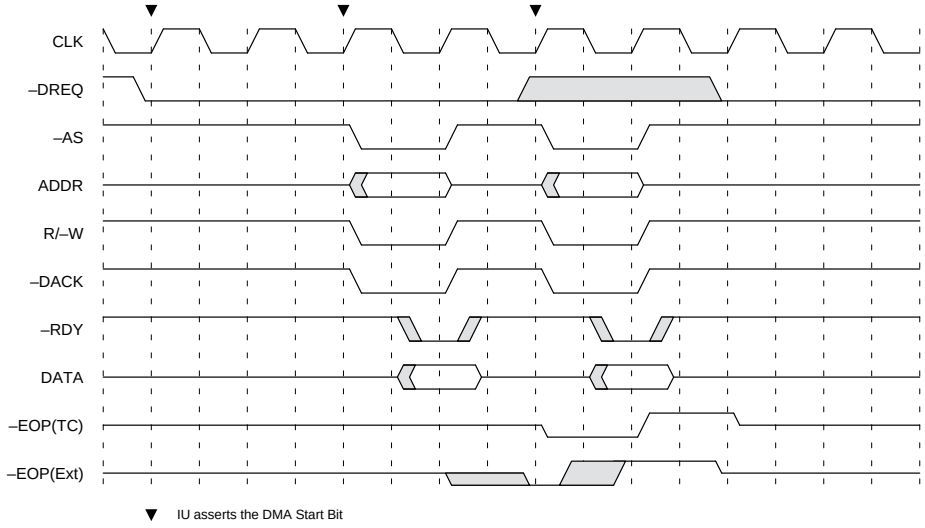


Figure B5-18. Demand Transfer, Flyby (R/–W low)

Transfer Addressing

- *Flyby*—Flyby mode is in effect when the source and destination have the same width, and flyby mode is enabled. –DACK is used to acknowledge the external DMA request, and to access the requestor's data. One bus cycle is needed for a byte, half-word, or word transfer; four bus cycles are needed for a quad-word flyby transfer. A single address is needed for this type of bus operation. The R/–W will signal the direction of data flow; for R/–W="1", the data flow is from the memory counterpart to the requesting device, and for R/–W="0" it is from the requesting device to the memory counterpart.

- *Flowthru*—For this bus operation, a read sequence is used to obtain the data from the source, and a write sequence is used to send the data to the destination. During read, the data will be assembled and put in a Temporary Register. During write, the data in the Temporary Register will be disassembled and sent to the destination. The DMA Controller will toggle the –DACK during the read or write session, depending on whether the External Control Option (EC) is set to Source or Destination Request. Whichever type of Request is specified by the EC, the other address is optional; for example, if EC=0 (Source Request), the provision of a destination address is unnecessary. The programmer can use the –DACK to enable a read or write to the external device whether the DMA request is internal or external.

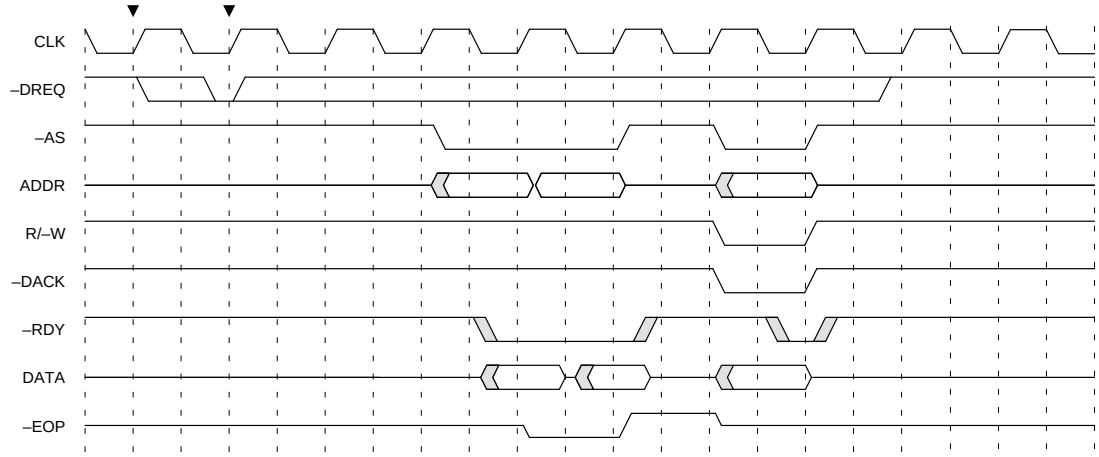


Figure B5-19. Single Transfer, Edge Sensitive, Flowthru, Dest Request Byte width for Source, Halfword width for Dest

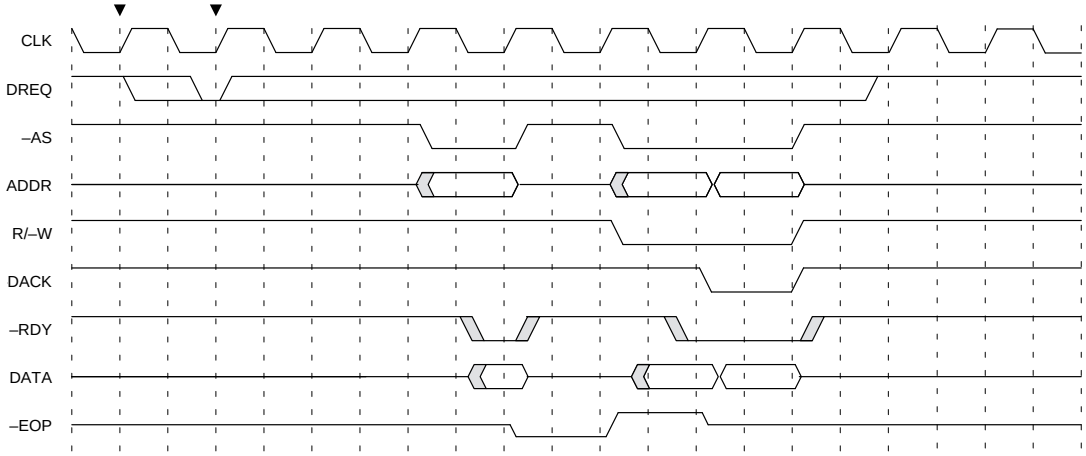


Figure B5-20. Single Transfer, Edge Sensitive, Flowthru, Source Request Word width for source, Halfword width for Dest

Source/Destination Data Length

The source and destination data length can be byte, half-word, word, or quad-word. For flyby transfer, the source and destination data length mast be the same. For flowthru mode, if the source and destination data lengths differ, the DMAC will automatically assemble the data during read to the bigger of the two sizes, and disassemble the data to the size of the destination during write. The assembly/disassembly applies only to the byte, half-word, and word sizes.

To take advantage of the burst transfer supported by BIU, the DMAC offers quad-word transfer. Quad-word transfer requires that both source and destination size be quad-word, and both source and destination addresses have to be aligned on quad-word boundaries. The DMAC will assert the quad-word address, and indicate to the BIU that quad-word transfer is needed; BIU will then decide when to proceed with burst-mode transfer.

For consistency with the memory mapping seen by the IU, address (31:2) is used as the byte address for byte transfers, as the halfword address for halfword transfers, and as the word address for either word or quad-word transfers.

Program/DMA Interaction

The –EOP issued by the DMAC can be used as an input to an interrupt controller.

A chaining wait mechanism is supported, enabling synchronization between the program and DMA buffer chaining. This chaining wait function provides a way for the user to modify the channel setup and/or modify the chaining descriptors while a chained DMA activity is in progress. The user can set the chaining wait function bit in the Control Register to enable this function. When this bit is set, and a buffer block has been transferred, the chaining wait bit in the Status Register will be set, and the corresponding DMA channel will go to chaining wait state, which is equivalent to the disabled state. The chaining wait bit set in the Control Register will block the loading of the next descriptor. The user can reprogram the channel, and then reset the chaining wait in the Status Register to restart the transfer. After the block has been transferred, –EOP will be issued as an input to the interrupt controller. The interrupt service routine may modify the channel setup registers and/or the chaining descriptors, and then clear the chaining wait bit in the Status Register. After the chaining wait bit in the Status Register has been cleared, the DMAC will start the DMA transfer using the modified channel setup.

```
–EOP will be asserted on these conditions:

Single buffer mode:TC (byte count expires)
                          Error on abnormal read/write transfer.
```

Chaining mode:	If only the chaining mode bit is set, and the whole chain transfer is completed
	Chaining wait function set in Control Register and the TC (byte count expires)
	Error on abnormal read/write transfer
	If chaining debug mode is set in the control register, –EOP will be asserted at the end of each transferred block.

Note: to use chaining wait, the user must set both chaining mode (CM) and chaining wait mode (CWM) in the control register. To use chaining debug, the user must set both CM and Chaining Debug Mode (CDM) in the control register.

–EOP can be used to interrupt the CPU, and the interrupt will be serviced based on the content of the Channel Status Register.

Memory Exception

Memory Exception (MEXC) is asserted by BIU to signal that an error condition was generated during transfer. The DMA channel will stop the transfer immediately, set up the relevant bit (Source/Destination/Chaining error) in the DMA channel Status Register, and assert the –EOP. The –EOP will be deasserted when the memory exception status bit is cleared by the program. For quad-word transfer (intended for burst mode), the DMA will finish all four read or write cycles before stopping and setting up the relevant bit in the Status Register.

CHAPTER

B6

MB86932 DSU

6.1 Overview

The MB86932 DSU offers several important features not found on the MB86930:

- Use of virtual address bus rather than physical address bus for breakpointing.
- “Context” is automatically taken into account in IA/DA Break Point comparisons.
- Readable ICE registers.

The MB86932 also includes an EMU pin interface that is fully compatible with the other members of the SPARClite family (MB86930 and MB86931).

6.2 Programmer’s Model

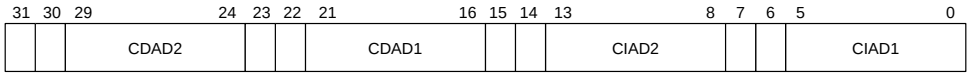
6.2.1 New Registers and Flags

Debug Control Register (augmented)

The Debug Control Register (see Figure 2-36) has been augmented by the addition of the Emulate_933 bit (DSU CR [13]). If that bit is set to 1, the MB86932 has six register windows, and the Implementation (impl) and Version (ver) fields of the Processor State Register (see Figure 2-3) identify the processor as an MB86933. If the Emulate_933 bit is 0, the MB86932 has eight register windows, and the impl/ver fields identify the processor as an MB86932.

This flag affects only the number of windows and the impl/ver fields in the PSR. The MB86932's cache and TLB can be enabled even if the flag is set to 1. Since the cache and TLB are disabled after Reset, the MB86932 will emulate the MB86930 correctly as long as the program being executed does not set the bits that enable the cache and the TLB.

Context Compare Register



Address: 0x0000FF20 to (DASI=0x01)

Figure B6-1. Context Compare Register

Bit 31: mask comparison of DASI[7:0] with dasid2[7:0]

Bit 30: Context_DA_Mask_2(CDAM2)

Bits 29-24 Context_DA_Description_2 (CDAD2)

Bit 23: mask comparison of DASI[7:0] with dasid1[7:0]

Bit 22: Context_DA_Mask_1 (CDAM1)

Bits 21-16: Context_DA_Description_1 (CDAD1)

Bit 15: mask comparison of ISUPER with isuperd2

Bit 14: Context_IA_Mask_2 (CIAM2)

Bits 13-8: Context_IA_Description_2 (CIAD2)

Bit 7: mask comparison of ISUPER with isuperd1

Bit 6: Context_IA_Mask_1 (CIAM1)

Bits 5-0: Context_IA_Description_1 (CIAD1)

Context IA/DA Mask Flags:

Each of the four Context Description fields is associated with a Mask field. If a Mask field is set to 1, its associated Context field is *not* compared. These flags are to be set when the ICE/Monitor logic does not recognize the concept of "context"; in such cases, a break will be invoked whenever the IA/DA matches the breakpoint address, regardless of context. Within the break routine, a check can be made to see if the context is the one for which the break was defined. This masking condition governs all the cases listed below.

6.2.2 Logic of Context Comparison

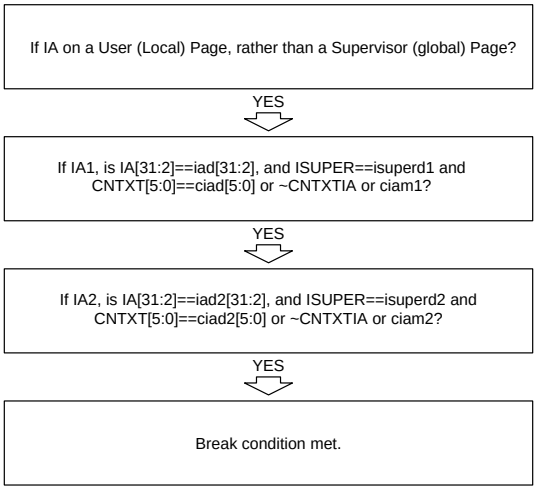


Figure B6-2. On an Instruction Address (IA) Match Break:

If no page in the TLB matches (that is, the IA's page is not found in there), the value of the CNTXTIA signal is undefined. This may cause an incorrect IA break request, but this will do no harm; the memory exception trap invoked by the TLB will cancel that incorrect break request.

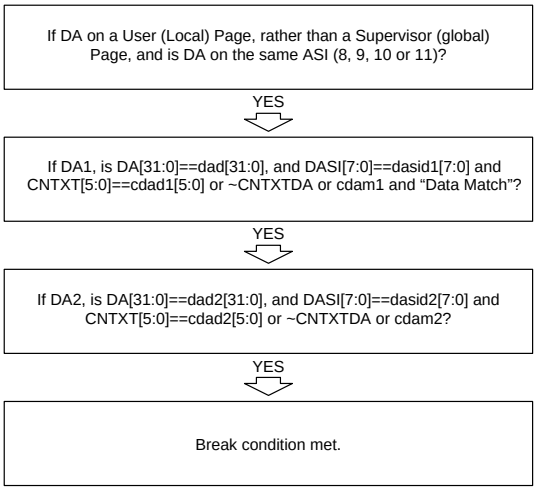


Figure B6-3. On a Data Address (DA) and Data Data (DD) Match Break:

If no page in the TLB matches (that is, the DA's page is not found in there), the value of the CNTXTDA signal is undefined. This may cause an incorrect DA break request, but this will do no harm; the TLB-miss or instruction-access exception trap invoked by the TLB will cancel that incorrect break request.

CHAPTER

B7

MB86932 External Interface

7.1 SIGNAL DESCRIPTIONS†

Symbol	Type	Description
–RESET	I	**SYSTEM RESET:** Asserting reset for at least 4 processor cycles after the clock has stabilized, causes the MB86932 to be initialized.
XTAL1, (CLK_IN) XTAL2	I/O O G(Q) I (Q)	**EXTERNAL OSCILLATOR:** The crystal inputs determine execution rate and timing of the MB86932 processor. Connecting a crystal to these pins forms a complete crystal oscillator circuit. The crystal oscillator frequency is the same as the processor operating frequency. When driving the processor with an external clock, XTAL2 pin should be left floating.
CLKOUT1	O G(Q) I (Q)	**CLOCK OUTPUT 1:** This is an output signal against which MB86932 bus transactions can be referenced. The CLKOUT1 frequency is the same as the frequency applied to XTAL1 and is the same as the processor operating frequency. CLKOUT1 is in phase with CLK_IN.
CLKOUT2	O G(Q) I (Q)	**CLOCK OUTPUT 2:** This is an output signal against which MB86932 bus transactions can be referenced. The CLKOUT2 frequency is the same as the frequency applied to XTAL1 and is the same as the processor operating frequency. CLKOUT2 is out of phase with CLK_IN.
–LOCK	O S(L) G(Z) I (1)	**BUS LOCK:** This is a control signal asserted by the processor to indicate to the system that the current bus transaction requires more than one transfer on the bus. The Atomic Load Store instruction for example requires contiguous bus transactions which cause the assertion of the bus lock signal. The bus may not be granted to another bus owner as long as –LOCK is active. –LOCK is asserted with the assertion of –AS and remains active until –READY is asserted at the end of the locked transaction.

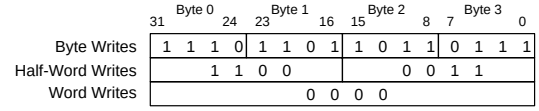
7.1 SIGNAL DESCRIPTIONS (Continued)†

Symbol	Type	Description
–BREQ	I S(L)	**BUS REQUEST:** Asserted by another device on the bus to indicate that it wants ownership of the bus. The request must be answered with a bus grant (–BGRNT) from the MB86932 before the device can proceed by driving the bus. Once the bus has been granted, the device has ownership of the bus until it de-asserts –BREQ. The user should ensure that devices on the bus cannot monopolize the bus to the exclusion of the CPU. Inputs to –BREQ while –RESET is active are valid and cause Bus Grant to be asserted.
–BGRNT	O S(L) G(0) I (Q)	**BUS GRANT:** Asserted by the CPU in response to a request from a device wanting ownership of the bus. The CPU grants the bus to other devices only after all transfers for the current transaction are completed. All bus drivers are three-stated with the assertion of the bus grant signal.
–ERROR	O S(L) G(Q) I (Q)	**ERROR SIGNAL:** Asserted by the CPU to indicate that it has halted in an error state as a result of encountering a synchronous trap while traps are disabled. In this situation the CPU saves the PC and nPC registers, sets the tt value in the TBR, enters into an error state and asserts the –ERROR signal. The system can monitor the –ERROR pin and initiate a reset under the error condition. This pin is high on reset.
–MEXC	I S(L)	**MEMORY EXCEPTION:** Asserted by the memory system to indicate a memory error on either a data or instruction access. Assertion of this signal initiates either a data or instruction access exception trap in the IU. The current bus access is invalidated by asserting the –MEXC in the same cycle as the –READY signal. The IU ignores the contents of the data bus in cycles where –MEXC is asserted.
IRL <3:0>	I A(L)	**INTERRUPT REQUEST BUS:** The value on these pins defines the external interrupt level. IRL<3:0>=1111 forces a non-maskable interrupt. IRL value of 0000 indicates no pending interrupts. All other values indicate maskable interrupts as enabled in the PIL field of the processor status register (PSR). Interrupts should be latched and prioritized by external logic and should be held pending until acknowledged by the processor. An interrupt controller is available on the MB86940.
–TIMER_OVF	O S(L) G(Q) I (Q)	**TIMER UNDERFLOW:** Asserted by the processor to indicate that the internal 16-bit timer has underflowed. This signal can be used to initiate a DRAM refresh cycle or a one cycle periodic waveform. On reset, the timer is turned off and –TIMER_OVF is high.
–SAME_PAGE	O S(L) G(1) I (1)	**SAME-PAGE DETECT:** The –SAME_PAGE is used to take advantage of fast consecutive accesses within Fast Page Mode DRAM page boundaries. This signal is an output asserted by the processor when the current address is within the same page as the previous memory access. The –SAME_PAGE signal is asserted with –AS and remains active for one processor cycle. –SAME_PAGE is never asserted in the first transaction following a transaction by another device on the bus. The page size is specified by writing the SAME-PAGE MASK register.
–CS0, –CS1, –CS2, –CS3, –CS4, –CS5	O S(L) G(1) I (1)	**CHIP SELECTS:** These outputs are asserted when the value on the address bus matches the address range in one of the corresponding ADDRESS RANGE registers. The signals are used to decode the current address into one of six address ranges. Address ranges should not overlap. Each address range has a corresponding wait specifier which is used to automatically assert the –READY signal after a user defined number of processor clock cycles. This allows a variety of memory and I/O devices with different access times to be connected to the MB86933 without the need for additional logic.

7.1 SIGNAL DESCRIPTIONS (Continued)†

<table>
<tr><th>Symbol</th><th>Type</th><th>Description</th></tr>
<tr><td>ADR <31:2></td><td>O
S(L)
G(Z)
I (1)</td><td>ADDRESS BUS: The 30-bit ADDRESS BUS (A31-A2) is an output which identifies the data or instruction address of a 32-bit word. Reads are always one word in size while byte, half-word, or word transaction sizes for writes is identified by separate byte-enable signals (–BE0-3). The address bus is valid for the duration of the bus transaction.</td></tr>
<tr><td>ASI <7:0></td><td>O
S(L)
G(Z)
I (1)</td><td>ADDRESS SPACE IDENTIFIERS: The ADDRESS SPACE IDENTIFIERS are outputs which indicate to which of 256 available spaces the current ADDRESS BUS value corresponds. The ASI values are defined as follows:
<table>
<tr><th>ASI <7:0></th><th>ADDRESS SPACE</th></tr>
<tr><td>0x1</td><td>Control Register</td></tr>
<tr><td>0x2</td><td>Instruction Cache Lock</td></tr>
<tr><td>0x3</td><td>Data Cache Lock</td></tr>
<tr><td>0x4 - 0x7</td><td>Application Definable</td></tr>
<tr><td>0x8</td><td>User Instruction Space</td></tr>
<tr><td>0x9</td><td>Supervisor Instruction Space</td></tr>
<tr><td>0xA</td><td>User Data Space</td></tr>
<tr><td>0xB</td><td>Supervisor Data Space</td></tr>
<tr><td>0xC</td><td>Instruction Cache Tag RAM</td></tr>
<tr><td>0xD</td><td>Instruction Cache Data RAM</td></tr>
<tr><td>0xE</td><td>Data Cache Tag RAM</td></tr>
<tr><td>0xF</td><td>Data Cache Data RAM</td></tr>
<tr><td>0x10 - 0xFD</td><td>Application Definable</td></tr>
<tr><td>0xFE - 0xFF</td><td>Reserved for Debug Hardware</td></tr>
</table>
The ASI values specified as “application definable” can be used by supervisor mode instructions such as Load Alternate and Store Alternate. The ASI value is available in the same cycle in which the corresponding address value is asserted on the address bus. The ASI pins are valid for the duration of the bus transaction. ASI values 0x8, 0x9, 0xA, and 0xB are cacheable.</td></tr>
<tr><td>–BMODE8</td><td>I
S(L)</td><td>8-BIT BOOT MODE: This signal is sampled during reset and causes read accesses, memory mapped to –CS0, to assume 8-bit ROM memory. The MB86932 generates four sequential fetches to assemble a complete instruction or data word before continuing. Bytes are fetched in sequence (0,1,2,3) as encoded by –BE[2] and –BE[3] (00, 01, 02, 03). Writes to –CS0 are unaffected by boot mode selection and if left unconnected, a weak pull-up on this pin (and –BMODE16 pin) causes the processor to default to 32-bit mode.
Note: BMODE8 and BMODE16 should not be asserted at the same time.</td></tr>
<tr><td>–BMODE16</td><td>I
S(L)</td><td>16-BIT BOOT MODE: This signal is sampled during reset and causes read accesses, memory mapped to –CS0, to assume 16-bit ROM memory. The MB86932 generates two sequential fetches to assemble a complete instruction or data word before continuing. Half words are fetched in sequence (0,1) as encoded by –BE[2]. Writes to –CS0 are unaffected by boot mode selection. If left unconnected, a weak pull-up on this pin (and –BMODE8 pin) causes the processor to default to 32-bit mode.
Note: BMODE8 and BMODE16 should not be asserted at the same time.</td></tr>
</table>

7.1 SIGNAL DESCRIPTIONS (Continued)†

Symbol	Type	Description
–BE3-0	O S(L) G(Z) I (O)	**BYTE ENABLES (O):** These pins indicate whether the current store transaction is a byte, half-word or word transaction. –BE0-3 signals are available in the same cycle in which the corresponding address value is asserted on the address bus and is valid for the duration of the bus transaction. This bus should be used only to qualify store transactions. For load transactions all sub-word requests are read (and replaced in the cache) as words and then the appropriate byte or half-word is extracted by the integer unit. Possible values for –BE3-0 are as follows:  BE<2:3> are also used in 8 and 16-bit ROM accesses as follows: Bus Mode / Byte / BE<2:3> 8-bit: 0 / 0 0; 1 / 0 1; 2 / 1 0; 3 / 1 1 16-bit: 0 & 1 / 0 0; 2 & 3 / 1 0
D <31:0>	I/O S(L) G(Z) I (Z)	**DATA BUS:** The bus interface has 32 bidirectional data pins (D31-D0) to transfer data in thirty-two bit quantities. D(31) corresponds to the most significant bit of the least significant byte of the 32-bit word. A double word is aligned on an 8-byte boundary, a word is aligned on a 4-byte boundary, and a half-word is aligned on a 2-byte boundary. If a load or store of any of these quantities is not properly aligned, a Not Aligned Trap will occur in the processor. In write bus cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If the preceding cycle was a write, data is driven in the cycle immediately following the cycle in which –READY was asserted. If the preceding cycle was a read, data is driven one cycle after the cycle in which –READY was asserted to minimize bus contention between the processor and the system. Pins D[7:0] are used when the 8-bit boot mode is enabled and D[15:0] are used when 16-bit mode is enabled.
–AS	O S(L) G(Z) I (1)	**ADDRESS STROBE:** A control signal asserted by the MB86930 or other bus master to indicate the start of a new bus transaction. A bus transaction begins with the assertion of –AS and ends with the assertion of –READY. –AS remains asserted for 1 clock cycle. During cycles in which neither the processor nor another bus master is driving the bus the bus is idle, and –AS remains de-asserted.
RD/–WR	O S(L) G(Z) I (1)	**READ/BUS TRANSACTION:** This signal specifies whether the current bus transaction is a read or a write operation. When –AS is asserted and RD/–WR is low, then the current transaction is a write. With –AS asserted and RD/–WR high, the current transaction is a read. RD/–WR remains active for the duration of the bus transaction and is de-asserted with the assertion of –READY.

7.1 SIGNAL DESCRIPTIONS (Continued)†

Symbol	Type	Description
–READY	I S(L)	**READY:** This is a control signal asserted by the external memory system to indicate that the current bus transaction is being completed and that it is ready to start with the next bus transaction in the following cycle. In case of a fetch from memory, the processor will strobe the value on the data bus at the rising edge of CLK_IN following the assertion of –READY. For the case of a write, the memory system will assert –READY when the appropriate access time has been met. In most cases, no additional logic is required to generate the –READY signal. On-chip circuitry can be programmed to assert –READY based on the address of the current transaction. The external system can override the internal ready generator to terminate the current bus cycle early. Up to 6 address ranges each with different transaction times can be programmed.
–DREQ0-1	A(L) I	**DMA REQUEST:** Indicates that an external device is requesting a DMA transfer. This signal is edge sensitive for single transfers and level sensitive for demand transfer. –DREQ0 corresponds to DMA channel 0, while –DREQ1 corresponds to DMA channel 1.
–DACK0-1	O	**DMA ACKNOWLEDGE:** This signal is asserted when an external device asserts –DREQ and the processor accesses the external device. –DACK1 corresponds to DMA channel 0, while –DACK1 corresponds to DMA channel 1.
–EOP0-1	I/O	**END OF PROCESS:** This signal is asserted by the external device when it wants to terminate a DMA transfer. Alternately, the processor drives this signal when the byte count reaches zero. –EOP0 corresponds to DMA channel 0, while –EOP1 corresponds to DMA channel 1. A pull-up holds –EOP0-1 high when it is not being driven.
–PBREQ	O	**PROCESSOR BUS REQUEST:** This signal is asserted by the processor to indicate to an external bus arbiter that it needs to regain control of the bus. This provides a handshake between the arbiter and the processor to allow the bus to be allocated based on demand.
–BMREQ	O	**BURST MODE REQUEST:** This signal is asserted by the processor to indicate to the external system that the processor's burst mode is enabled and the current transaction can be a burst. If the external system supports burst mode, it asserts –BMACK concurrently with –RDY to begin the burst mode transfer.
–BMACK	I	**BURST MODE ACKNOWLEDGE:** This signal is asserted by the system to indicate that it can support burst mode for the address currently on the bus. The system asserts –BMACK in response to the processor asserting –BMREQ.
CLK_ECB	I	**EXTERNAL CLOCK BYPASS:** Tying this signal high causes the CLK_IN signal to bypass the Phases Lock Loop (PLL). This signal is used for testing of the chip.
PARITY<3:0>	I/O	**PARITY:** When enabled, this signal provides even or odd parity checking for data bus accesses.
EMU_SD <3:0>	I/O	**EMULATOR STATUS/DATA BITS:** Bi-directional pins used by a hardware emulator to control and monitor MB86930 execution. These pins should be left unconnected.
EMU_D<3:0>	I/O	**EMULATOR DATA BITS:** Bi-directional pins used by a hardware emulator to control and monitor MB86930 execution. These pins should be left unconnected.
EMU_BRK	I	**EMULATOR BREAK REQUEST LINE:** Input used by a hardware emulator to request a trap when emulation is enabled. This pin should be left unconnected.

7.1 SIGNAL DESCRIPTIONS (Continued)†

Symbol	Type	Description
–EMU_ENB	I	**EMULATOR ENABLE:** Tied low while the MB86930 is being reset to enable hardware emulator mode on the chip. This pin should be left unconnected.
TCK	I	**TEST CLOCK:** JTAG compatible test clock input**.**
TMS	I	**TEST MODE:** JTAG compatible test mode select pin. Test is enabled when –TMS is low.
TDI	I	**TEST DATA IN:** JTAG compatible test data input.
TDO	O	**TEST DATA OUT:** JTAG compatible test data output.
–TRST	I	**TEST RESET:** Asynchronous reset for JTAG logic. If not using JTAG, this signal must be pulled low.

†. In the following descriptions, signal names preceded by a minus sign (–) indicate an active low state. Dual function pins have two names separated by a slash (/).

NOTES:

I = Input Only Pin
O = Output Only Pin
I/O = Either Input or Output Pin
- = Pins "must be" connected as described
A(L) = Asynchronous: Inputs may be asynchronous to CLKOUT.
S(L) = Synchronous: Inputs must meet setup and hold times relative to CLK_IN. Outputs are Synchronous to CLK_IN

G(…) = While the bus is granted to another bus master (–BGRNT=asserted), the pin is
G(1) is driven to V_{CC}
G(0) is driven to V_{SS}
G(Z) floats
G(Q) is a valid output

I (…) = While the bus is between bus cycles (or being reset) and is not granted to another bus master, the pin is
I (1) is driven to V_{CC}
I (0) is driven to V_{SS}
I (Z) floats
I (Q) is a valid output

CHAPTER

B8

MB86932 JTAG

8.1 MB86932 JTAG Pin List

The MB 86932 JTAG cells are arranged in a shift register configuration (see Figure B8-8). When shifting in a JTAG pattern through TDI, the LSB should correspond to the JTAG cell value for –TIMER_OVF pin whereas, the MSB of the pattern should correspond to the CLK_ENB pin's JTAG cell. As far as JTAG output through TDO is concerned, the first bit out corresponds to –TIMER_OVF JTAG cell value and the last output bit corresponds to the CLK_ENB JTAG cell value. Table B8–1 lists the order of all of the JTAG cells.

Table B8-1: JTAG Pin Order

Order	JTAG Cell	JTAG Cell Type	Function
1	–TIMER_OVF	output	Timer Overflow pin
2	XTAL1	input	Crystal input
3	–TEST	input	Factory test pin
4	PARITY<2>	in/out	
5	PARITY<3>	in/out	
6	EMU_BRK	input	Emulator break input

Table B8-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
7	icediojo†	output	Bidirectional control for EMU_D/EMU_SD buses icediojo = 1: EMU_D and EMU_SD buses are input icediojo = 0: EMU_D and EMU_SD buses are output
8	EMU_SD_i<3>	input	Input bit 3 of EMU_SD<3:0> bus
9	EMU_SD_o<3>	output	Output bit 3 of EMU_SD<3:0> bus
• • •	• • •	• • •	• • •
14	EMU_SD_i<0>	input	Input bit 0 of EMU_SD<3:0> bus
15	EMU_SD_o<0>	output	Output bit 0 of EMU_SD<3:0> bus
16	EMU_D_i<3>	input	Input bit 3 of EMU_D<3:0> bus
17	EMU_D_o<3>	output	Output bit 3 of EMU_D<3:0> bus
• • •	• • •	• • •	• • •
22	EMU_D_i<0>	input	Input bit 0 of EMU_D<3:0> bus
23	EMU_D_o<0>	output	Output bit 0 of EMU_D<3:0> bus
24	iceenblio†	output	Bidirectional control signal for –EMU_ENB pin iceenblio = 1: –EMU_ENB pin is an input iceenblio = 0: –EMU_ENB pin is an output
25	–EMU_EN_i	input	Input bit of –EMU_ENB pin
26	–EMU_EN_o	output	Output bit of –EMU_ENB pin
27	dbusiojo†	output	Bidirectional control signal for D<31:0>, Parity <3:0> dbusiojo = 1: D<31:0>, Parity <3:0> are inputs dbusiojo = 0: D<31:0>, Parity <3:0> are inputs
28	D_i<31>	input	Input bit 31 of D<31:0> bus
29	D_o<31>	output	Output bit 31 of D<31:0> bus
• • •	• • •	• • •	• • •
54	D_i<18>	input	Input bit 18 of <31:0> bus
55	D_o<18>	output	Output bit 18 of D<31:0> bus
56	–BMODE16	input	
57	D_i<17>	input	Input bit 17 of D<31:0> bus
58	D_o<17>	output	Output bit 17 of D<31:0> bus
59	D_i<16>	input	Input bit 16 of D<31:0> bus
60	D_o<16>	output	Output bit 16 of D<31:0> bus
61	D_i<15>	input	Input bit 15 of D<31:0> bus
62	D_o<15>	output	Output bit 15 of D<31:0> bus
63	–BMODE8	input	
64	D_i<14>	input	Input bit 14 of D<31:0> bus

Table B8-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
65	D_o<14>	output	Output bit 14 of D<31:0> bus
⋮	⋮	⋮	⋮
80	D_i<6>	input	Input bit 6 of <31:0> bus
81	D_o<6>	output	Output bit 6 of D<31:0> bus
82	–BMREQ	output	
83	D_i<5>	input	Input bit 5 of D<31:0> bus
84	D_o<5>	output	Output bit 5 of D<31:0> bus
⋮	⋮	⋮	⋮
93	D_i<0>	input	Input bit 0 of <31:0> bus
94	D_o<0>	output	Output bit 0 of D<31:0> bus
95	–RESET	input	Chip reset pin
96	–BREQ	input	Bus request input
97	–MEXC	input	Memory exception input
98	–READY	input	External memory transaction complete signal
99	tstatejo†	output	Three–state control signal for ADR, ASI, –BE, –AS, RD/WR and –LOCK If tstatejo = 1: signals are three–stated. If tstatejo = 0: signals are outputs.
100	–BGRNT	output	Bus grant output signal
101	–ERROR	output	Error output signal
102	–LOCK	output	Bus lock output signal
103	–BMACK	input	
104	–RD/WR	output	Memory Read/Write output signal
105	–AS	output	Start of memory transaction output signal
106	–PBREQ	output	
107	–CS<0>	output	
108	–DREQ0	input	
109	–CS<1>	output	
110	–CS<2>	output	
111	–CS<3>	output	
112	–CS<4>	output	
113	–DREQ1	input	
114	–CS<5>	output	
115	–SAMEPAGE	output	

Table B8-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
116	–DACK0	output	
117	BE<3>	output	
118	BE<2>	output	
119	BE<1>	output	
120	BE<0>	output	
121	ASI<0>	output	
122	ASI<1>	output	
123	ASI<2>	output	
124	ASI<3>	output	
125	–DACK1	output	
126	ASI<4>	output	
127	ASI<5>	output	
128	ASI<6>	output	
129	ASI<7>	output	
130	ADR<2>	output	
131	ADR<3>	output	
132	ADR<4>	output	
133	ADR<5>	output	
134	eopio0	output	Bidirectional control for –EOP0 pin eopio0 = 1: –EOP0 is input eopio0 = 0: –EOP0 is output
135	ADR<6>	output	
136	ADR<7>	output	
137	–EOP0_i	input	
138	–EOP0_o	output	
139	ADR<8>	output	
140	ADR<9>	output	
141	eopio1	output	Bidirectional control for –EOP1 pin eopio1 = 1: –EOP0 is input eopio1 = 0: –EOP0 is output
142	ADR<10>	output	
143	–EOP1_i	input	
144	–EOP1_o	output	
145	ADR<11>	output	
⋮	⋮	⋮	⋮

Table B8-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
150	ADR<16>	output	
151	PARITY_i<0>	input	
152	PARITY_o<0>	output	
153	ADR<17>	output	
154	ADR<18>	output	
155	ADR<19>	output	
156	ADR<20>	output	
157	PARITY_i<1>	input	
158	PARITY_o<1>	output	
159	ADR<21>	output	
⋮	⋮	⋮	⋮
169	ADR<31>	output	
170	IRL<3>	input	
171	IRL<2>	input	
172	IRL<1>	input	
173	IRL<0>	input	
174	CLK_ENB	input	

†. These are internal I/O control signals. Therefore, there are no corresponding external pins.
1. The following pins are not three-statable. –SAME_PAGE, –CS<5:0>, –BGRNT, TIMER_OVF, –ERROR.
2. The following pins have no corresponding JTAG cells: CLKOUT1, CLKOUT2, XTAL2, –TRST, TCK, TMS, TDI, TDO.