FUJITSU

SPARClite

Embedded Processor
User's Manual

MB86936
Addendum

June 1995, Edition 1.1

FUJITSUMICROELECTRONICS, INC.

This document is published by the marketing department of Fujitsu Microelectronics, Inc., Semiconductor Division, 3545 North First Street, San Jose, California, U.S.A. 95134–1804.

Table Of Contents

CHAPTER

E1

Overview of MB86936

E1.3 General Description

The MB86936 is a member of the SPARClite family whose function set is a superset of that of the MB86930. It is available in a 208-pin package, pin compatible to MB86932, and is capable of operating at 60 MHz. In addition to all the features of the MB86930 processor, the MB86936 contains the following:

- **Floating Point Unit:** The MB86936 features a floating-point unit that fully conforms to the ANSI/IEEE Standard 754-1985, the SPARC Architecture Version 8 specification, and the SPARC IEEE754 Implementation Recommendation with the Nonstandard FP (NS=1) mode enabling “flush to zero” treatment of denormalized operands or results as permitted by the recommendation. The FPU contains thirty-two 32-bit floating-point f registers, designated f[0] to f[31].
- **Instruction Cache:** The MB86936 has an 4K-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. 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 MB86936 has a 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. 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 MB86936 has three 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."
- **DRAM Interface:** A high-performance DRAM interface is integrated on-chip.
- **Configurable External Data Bus:** The MB86936 includes a data bus that can be configured as 8, 16, or 32 bits wide. This enables the MB86936 to execute from a single by-8 or by-16 Memory.
- **Burst Mode:** The MB86936 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.
- **Bus Interface Unit:** The MB86936 BIU is capable of running at half the frequency of the core. This facilitates system design for users who want to run the core at 60 MHz to achieve high performance, while running external bus peripherals at 30MHz for economy.
- **Power Down Modes:** The MB86936 supports several power down modes. These modes allow the user to turn off the clocks to various parts of the chip that may not be in use, reducing power consumption.
- **Video Interface:** The MB86936 has a built–in interface to directly connect to a laser printer engine or high resolution scanner.
- **Timers:** The MB86936 features two independent general purpose 24–bit timers (a 16–bit counter with a 8–bit prescaler) that can be independently programmed to operate in one of the three different modes. TIMER0 and TIMER1outputs are connected to bit 5 and bit 3, respectively of internal interrupt request controller. TIMER0 output is also available on a MB86936 pin.
- **IRC:** An advanced Interrupt Request Controller provides a flexible way of handling interrupt request in an encoded or decoded request. The Interrupt Request Controller also allows a programmable interrupt priority. Interrupts from MB86936 peripheral controllers are included internally.
- **Emulation Mode Support:** A new feature in the MB86936 DSU allows the chip to emulate the MB86935. The version field of the Processor State Register (PSR) is 8 with the MB86936 in "native" mode and 9 when it is emulating the MB86935. In addition, emulating the MB86935 causes the hardware floating point unit to be disabled. The emulation mode feature is described in detail in section 9.3.

E1.4 Programmer's Model of the MB86936

1.4.1 User-visible Registers

All the special-purpose registers and ASR registers defined on the MB86930 exist also on the MB96836.

All on-chip control/status/data registers which exist in alternate address spaces in the MB86930, with one exception, exist also on the MB86936 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 MB86936 have made it necessary to add new registers, accessible through alternate address spaces. 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.

The registers are reset to 0 by default.

Cache/BIU Control/Status Registers:

ASI: 0x01

Address range: 0x00000000-0x000000FF

0x00000000	ASI=0x1	Cache/BIU Control Register
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
0x00000060	ASI=0x1	Power Down Register
0x00000080	ASI=0x1	System Support Control Register (DMA priority; even/odd parity bits 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)
0x0000016C	ASI=0x1	Bus Width and Cacheable
0x00000174	ASI=0x1	DRAM Refresh Timer
0x00000178	ASI=0x1	DRAM Refresh Preload Timer
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)
0x000001C0	ASI=0x1	Source/Destination ASI Register (DMA2)
0x000001C4	ASI=0x1	Current Source Address Register (DMA2)
0x000001C8	ASI=0x1	Current Destination Address Reg (DMA2)
0x000001CC	ASI=0x1	Current Byte Count Register (DMA2)
0x000001D0	ASI=0x1	Descriptor Pointer (DP) Register (DMA2)
0x000001D4	ASI=0x1	Channel Control Register (DMA2)
0x000001D8	ASI=0x1	Channel Status Register (DMA2)

Interrupt Controller Registers:

0x00000200	ASI=0x1	Trigger Mode Register 0
0x00000204	ASI=0x1	Trigger Mode Register 1
0x00000208	ASI=0x1	Request Sense Register
0x0000020C	ASI=0x1	Request Clear Register

0x00000210	ASI=0x1	IRC Mask Register
0x00000214	ASI=0x1	IRC Latch Clear Register
0x00000218	ASI=0x1	IRC Mode Select

Timer Registers:

0x00000240	ASI=0x1	Prescaler Register 0
0x00000244	ASI=0x1	Timer Control Register 0
0x00000248	ASI=0x1	Reload Register 0
0x0000024C	ASI=0x1	Count Register 0
0x00000250	ASI=0x1	Prescaler Register 1
0x00000254	ASI=0x1	Timer Control Register 1
0x00000258	ASI=0x1	Reload Register 1
0x0000025C	ASI=0x1	Count Register 1

Video Controller Registers:

0x00000280	ASI=0x1	Top Margin Register
0x00000284	ASI=0x1	Left Margin Register
0x00000288	ASI=0x1	Block Height Register
0x0000028C	ASI=0x1	Line Width Register
0x00000290	ASI=0x1	Start Bit Register
0x00000294	ASI=0x1	Video Control Register 1
0x00000298	ASI=0x1	Video Control Register 2
0x0000029C	ASI=0x1	Video Status Register
0x000002A0	ASI=0x1	Transmit FIFO
0x000002A4	ASI=0x1	Receive Buffer

DRAM Control Registers:

0x000007D0	ASI=0x1	DRAM Bank Configuration (Bank 0)
0x000007D4	ASI=0x1	DRAM Bank Configuration (Bank 1)
0x000007D8	ASI=0x1	DRAM Bank Configuration (Bank 2)
0x000007DC	ASI=0x1	DRAM Bank Configuration (Bank 3)
0x000007E0	ASI=0x1	DRAM Timing Register 1
0x000007E4	ASI=0x1	DRAM Timing Register 2

Emulation Registers:

ASI: 0x01

Address range: 0x0000F000-0x0000FFFF

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

Test Registers:

ASI: 0x01

Address range: 0x00010000-0x0001FFFF

0x00010000	ASI=0x1	Global Test Register
0x00010008	ASI=0x1	IRC Test 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

E1.5 Internal Architecture of the MB86936

Figure E1-1 shows a block diagram of the MB86936. The two major buses shown in the diagram are as follows:

- Data Data Bus (DD)—A 64-bit bus used to transfer data to and from MB86936 functional units. In general, when a load is executed, data is transferred to the Integer Unit (IU) or Floating Point Unit (FPU) from one of the other units, and when a store is executed, data is transferred from the IU or FPU 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 IU can read data (load alternate) or write data (store alternate) to the control/status/data registers of all units, except the Instruction Cache and Instruction Tags, via DD bus. The Instruction Cache and Instruction Tags can be accessed only through the ID bus.

- 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.)

- DMA Bus (FD)—This 32-bit bus allows transfer of data from the BIU to the DMA.

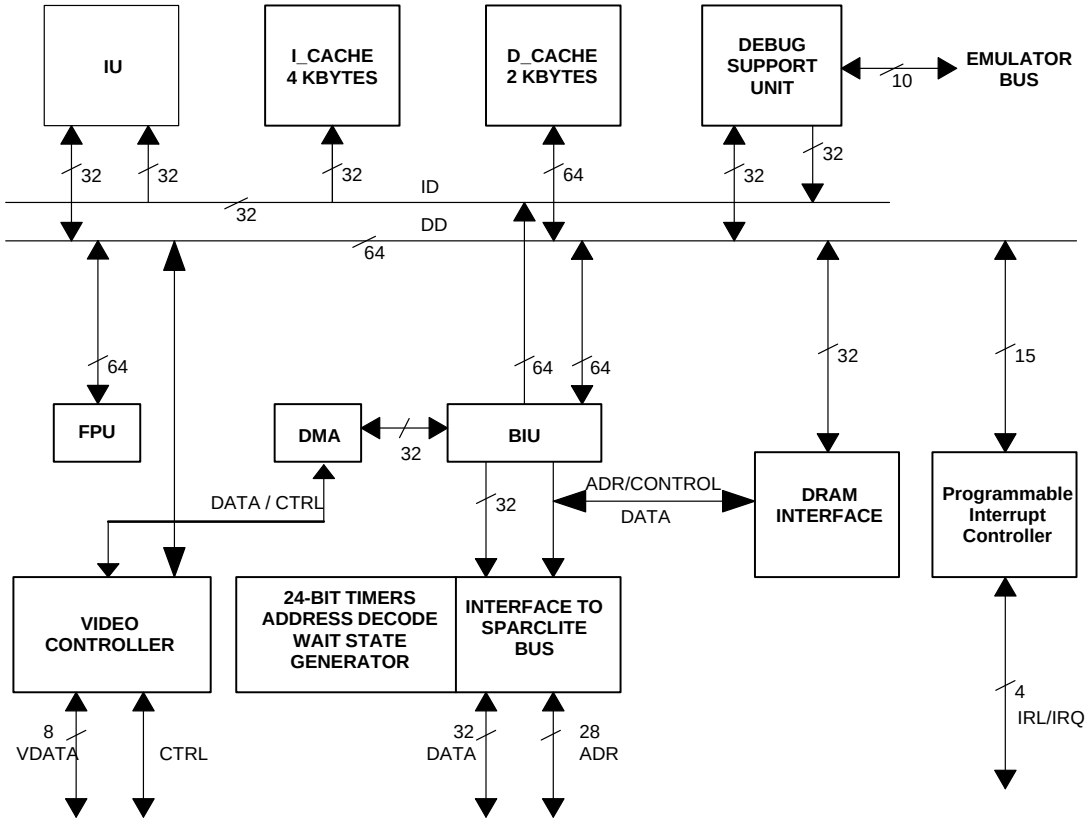


Figure E1-1. MB86936 Block Diagram

CHAPTER

E2

MB86936 Caches

E2.1 Overview of MB86936 Caches

The MB86936 offers enhanced support for cacheing: its instruction cache is 4K-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 MB86936 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).

E2.2 Programmer's Model

The cache control/status registers of the MB86936 form a superset of those in the MB86930. The registers common to the two chips are as follows:

0x00000000	ASI=0x1	Cache/BIU Control Register
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 MB86936 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.

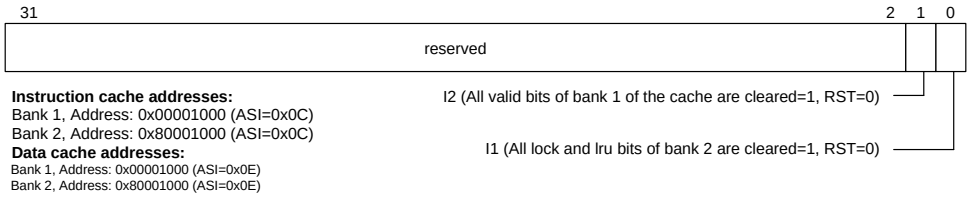


Figure E2-1. Cache Invalidate Register Format

E2.2.1 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.

E2.2.2 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.

E2.3 Internal Architecture of MB86936 Caches

Figure E2-2 shows cache operation (in the example shown, the Instruction Cache):

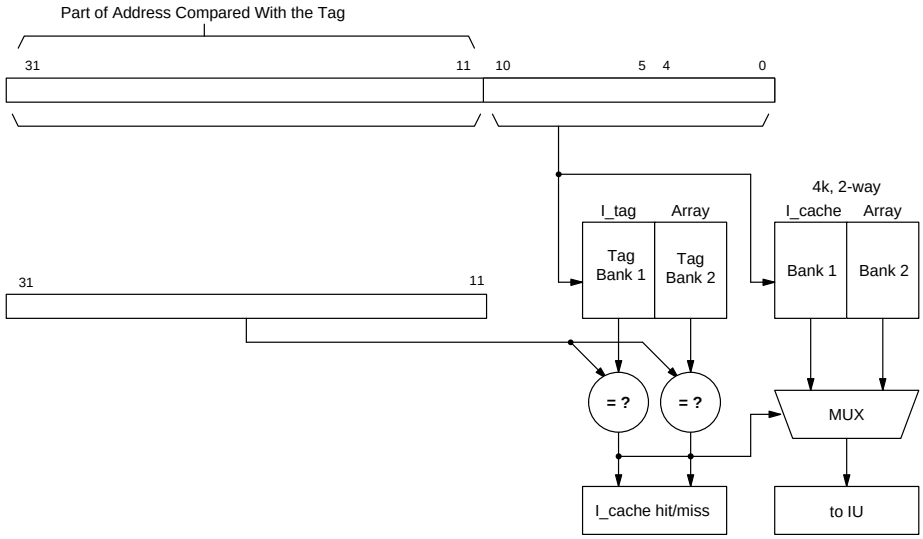


Figure E2-2. Cache Operation

E2.3.1 Instruction Cache

The instruction cache is an 4K-byte, 2-way associative, sectored cache, with 8-word lines. The basic operation of the cache is as follows: the IU sends the address to the I_cache, and I_cache tags. The lower 11 bits of the address are used to access the tag array and the I_cache. The tag read from the tag array is compared to bits 31-11 of the address to determine hit or miss.

The address coming out of the IU goes to the I_cache and tags. Bits 31-11 go to the tag array for comparison. Bits 10-5 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 4K I_cache. Bits 4-2 select a word out of the 8-word line.

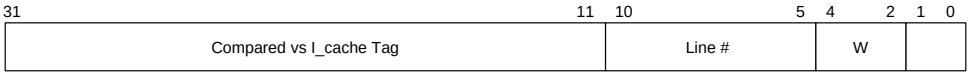


Figure E2-3. Address to I_cache and Tag Array

The instruction cache tag format is shown below. Twenty–one 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 MB86930 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 MB86936 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

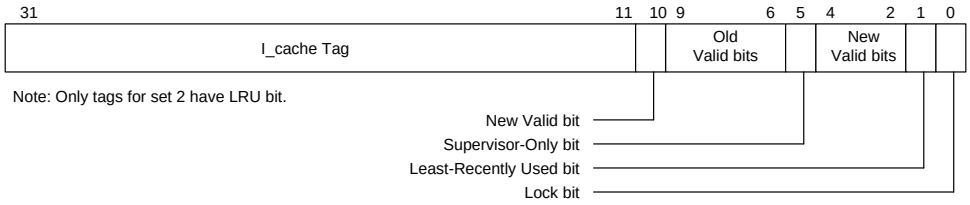


Figure E2-4. I_cache Tag Format

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.

E2.3.2 Read Hit

On an instruction fetch, the tag and the instruction are accessed in parallel, using the lower 12 bits of the address. If bits 31-11 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.”

E2.3.3 Miss Processing

If the address field in the tag does not match the address bits (31-11) 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 address bits (31-11), 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 address bits (31-11), 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.”

E2.3.4 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 address to the D_cache, and D_cache tags. The lower 12 bits of the address are used to access the tag array and the D_cache. Once this is completed, the tag read from the tag array is compared to bits 31-10 of the address to determine hit or miss.

Bits 31-10 of the address go to the tag array for comparison. Bits 9-4 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.

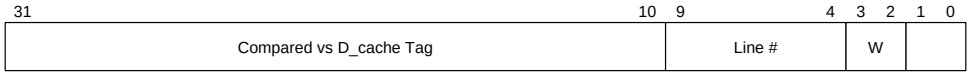


Figure E2-5. Address to D_cache and Tag Array

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.

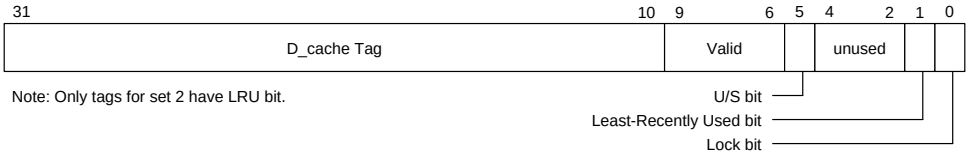


Figure E2-6. D_cache Tag Format

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.

E2.3.5 Read Hit

On a load, the tag and the data are accessed in parallel, using the lower 12 bits of the address. If bits 31-10 of the address field coming from the IU match 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.” There is a 64-bit data path between the cache and the FPU.

E2.3.6 Write Hit

On a store, if a hit is detected, 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.

E2.3.7 Miss Processing

If the address field in the tag does not match the address bits (31-10) coming from the IU, 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.”

E2.3.8 Atomic Load and Store

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

E2.3.9 Non–cacheable Memory Space

The MB86936 also supports non–cacheable memory region in DCACHE. There is no write in DCACHE if the request is from this region. The non–cacheable region can be programmed or from the external pin. Please refer to section E3–11 in BIU.

CHAPTER

E3

MB86936 Bus Interface Unit

E3.1 Overview of Bus Interface Unit

The BIU on the MB86936 includes all the features of the MB86930, and in addition offers the following:

- Option to run core at double the frequency of the Bus Interface Unit,
- 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 processor bus request feature that enables the MB86936 to request access to external address and data buses,
- A peripheral-to-DRAM interface,
- Glueless interface to ROM, EEPROM,
- Handshaking signals for external bus masters,
- 8-bit/16-bit/32-bit read and write between MB86936 and memory,
- Four deep buffered writes to increase bus access throughput,
- One deep instruction prefetching.

E3.2 Burst Mode

E3.2.1 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," Figure E3–1) 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.

E3.2.2 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 for a data transfer request for a quad word or more data, regardless of the status of the DBE bit.

Burst mode in DRAM memory space using the internal DRAM controller is another exception. The DRAM burst enable bit in the System Support Control Register is enabled to support DRAM burst (see DRAM Controller Chapter). If the bit is set, MB86936 will assert –BMREQ upon cache miss in DRAM space to indicate a burst mode in DRAM, but does not need –BMACK signal before starting the burst cycle.

E3.2.3 Burst Mode Fetch Sequence

In burst-mode accesses, the cache 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 table below shows the four possible sequences of words, in terms of their address LSBs and ADR[3:2], 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. ADR[3:2] change in subsequent three cycles to indicate the respective address fetched.

Table E3-1: 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

E3.2.4 Burst Mode Control Bits

Two bits in the Bus Control Register (ASI=0x0000 0020) are used to control burst mode for instruction fetches and data loads.

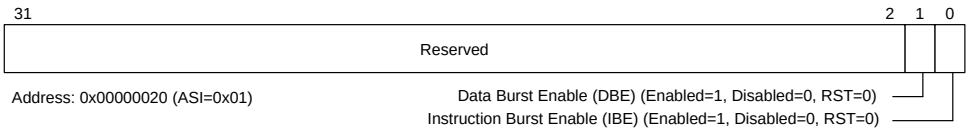


Figure E3-1. Bus Control Register

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.

E3.2.5 Non 32-bit Address Space

Burst mode access from the non 32-bit address space is not supported for 8- or 16-bit bus mode. If burst mode is enabled, and the address lies within the 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.

E3.2.6 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.

E3.2.7 Cache Off

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

E3.2.8 Bus Request

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

E3.2.9 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 Integer 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.

E3.2.10 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.

E3.2.11 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.

E3.2.12 Interface Timing

Figure E3-2 shows the timing of a burst mode transaction for an instruction fetch, data load, or DMA read. To start the transaction, the MB86936 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 –RDY for 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 E3-3.

Figure E3-4 shows the timing for the write portion of a DMA burst operation. The timing is identical to that in Figure E3-2, 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 changes through the burst based on sequence given in Table E3–1.

E3.3 Parity

The MB86936 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.

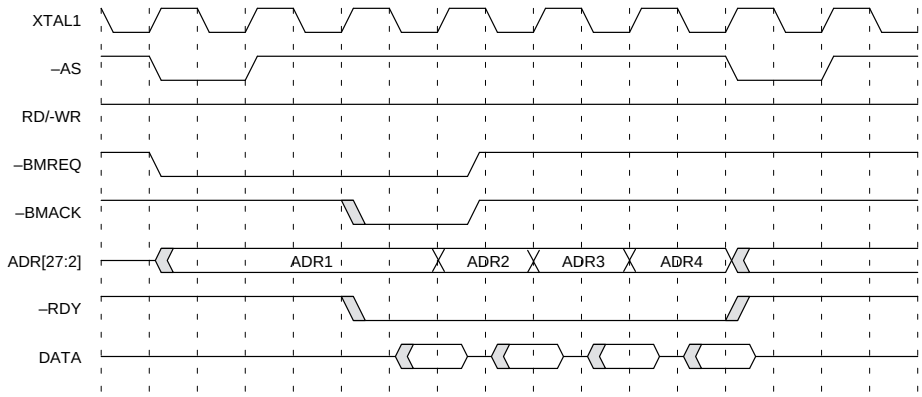


Figure E3-2. Burst Mode (0 wait state)

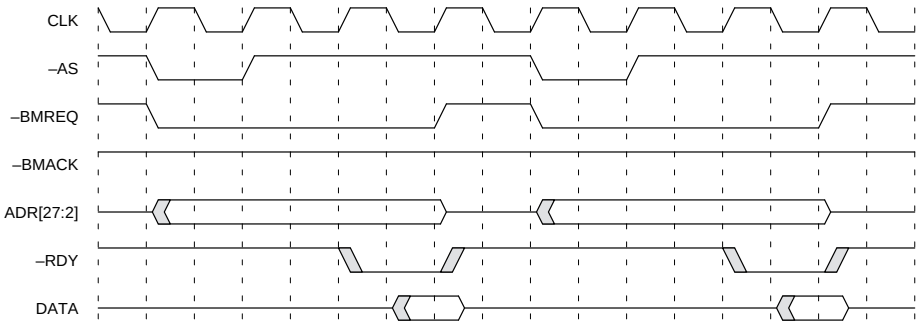


Figure E3-3. Terminated Burst Mode Due to –BMACK=1

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. If a parity error is detected on a data fetch, a data_memory_exception occurs. The parity bits will have a longer setup/delay time than the other data bits.

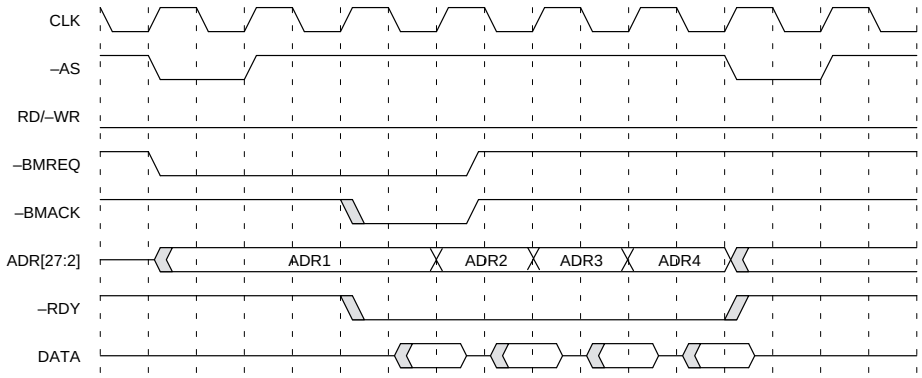


Figure E3-4. DMA Burst Mode, Write Portion

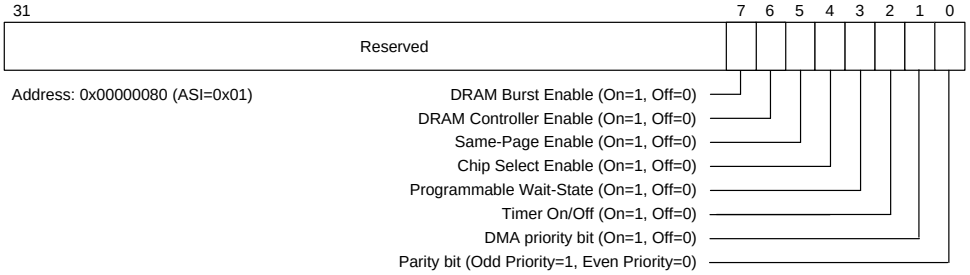


Figure E3-5. System Support Control Register

E3.4 Non Volatile/Flash Memory Support Signals

The MB86936 has two new signals, –NVWE (non-volatile RAM write enable) and –OE (output enable), that control non-volatile memory, such as EEPROM and/or Flash Memory.

–NVWE is used during writes to non-volatile memory to allow sufficient data hold time for the memory. It is asserted one cycle after –AS is asserted, and is released immediately when –READY is asserted, as shown in Figure E3-6. Therefore, at least three cycles must be implemented when the –NVWE signal is used.

–OE is used during reads from non-volatile memory to enable the memory output drivers. It is asserted one cycle after –AS is asserted, and is released at the end of the data transfer operation, as shown in Figure E3-7.

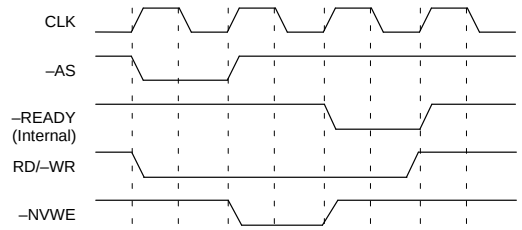


Figure E3-6. Non-Volatile Memory Write Timing

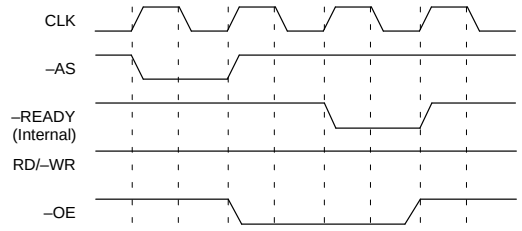


Figure E3-7. Non-Volatile Memory Read Timing

–NVWE and –OE operate only if both the Programmable Wait-State Enable bit in the System Support Control Register and the Wait Enable bit in the Wait-State Specifier Register are set to 1.

E3.5 External Bus Master Support

An external bus master requests for the memory bus by asserting –BREQ. The processor responds to an external bus master request by asserting the –BGRNT (Bus Grant) signal. The external bus master can drive RD/–WR, the address and data bus only during the –AS cycle. The BIU asserts –$\overline{CS}$ (Chip Select) for the external bus master one cycle after asserting –AS, and the wait state control logic asserts –RDYOUT to terminate the operation, as shown in Figure E3-8. The external bus master deasserts –BREQ when the bus is no longer required.

E3.5.1 DRAM Interface

If the bus request is for access to the DRAM through the internal DRAM controller, the BIU acts as an interface between the external bus master and the DRAM. In this case, –AS, ASI, RD/–WR, and ADR signals are I/O signals.

For DRAM reads, the external bus master asserts –AS, ASI, and ADR and asserts RD/–WR high to read. The signals are driven for one cycle only. The ASI and ADR are processed. –CS4 and processed ADR are sent out to the external DRAM. Internal DRAM ready is asserted to tell BIU that the corresponding data is available. BIU read in the data and resent it out. –RDYOUT is asserted to indicate the completion of the request.

For DRAM writes, the external bus master asserts –AS, sets the ASI, ADR, and D[31:0], and deasserts RD/–WR to write. The signals are also asserted for one cycle only. ASI and ADR are processed. –CS4, DRAM address and word data D[31:0] are driven out by BIU. Internal DRAM ready triggers –RDYOUT to indicate the completion of DRAM write.

Only word access is supported for external bus request to/from DRAM using the internal DRAM controller.

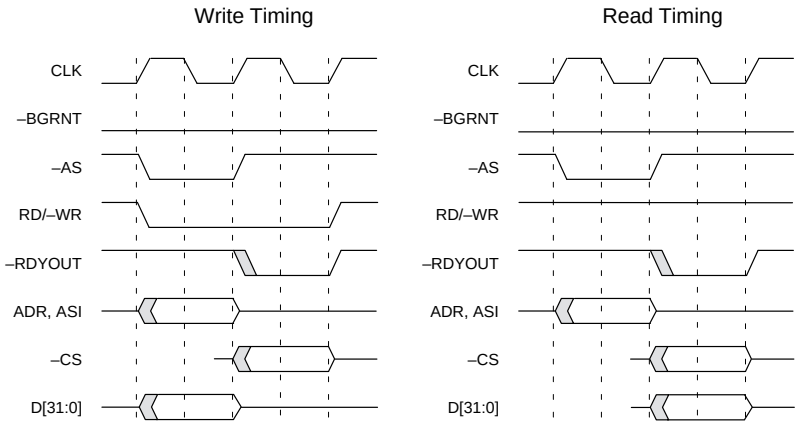


Figure E3-8. External Bus Master Signal Timing

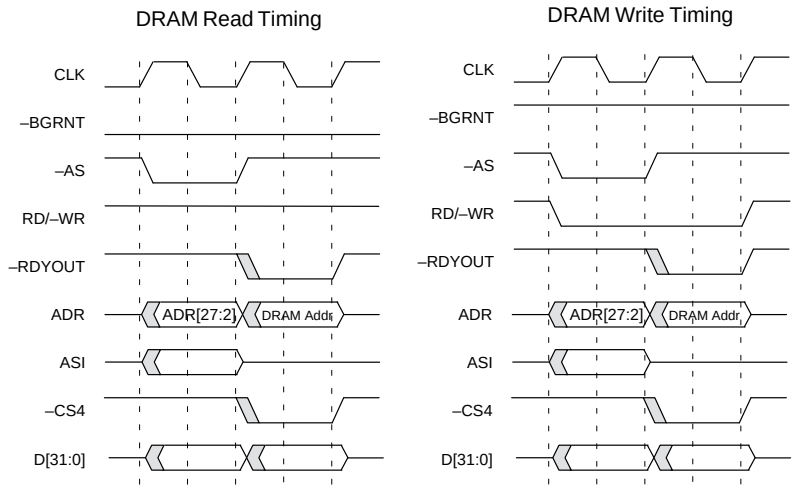


Figure E3-9. External Bus Master Signal Timing for DRAM Access

E3.6 Processor Bus Request

E3.6.1 Purpose

When the bus is released in response to an external device's request for the bus (by asserting –BREQ), the MB86936 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.

E3.6.2 Features

–PBREQ pin: An external pin is used to output the processor bus request signal, –PBREQ. The –PBREQ will be asserted whenever the MB86936 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. When the internal write buffer is enabled, BIU does not assert –PBREQ until the write buffer is completely filled. An example of the –PBREQ timing is shown in the figure on the following page:

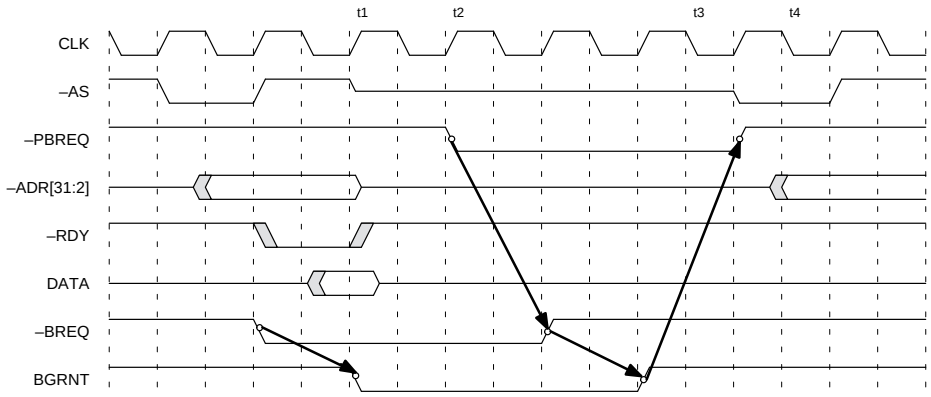


Figure E3-10. Example of –PBREQ timing

In the example above, after the current bus transaction is completed, 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.

E3.7 Same Page Support

The MB86936 supports same memory page operation only in the chip select 4 address range by asserting the SAMEPAGE signal when the current address is in the same memory page as the previous address. To use the SAMEPAGE signal, the memory *must* be located in the chip select 4 address range, and the internal DRAM controller must be disabled. When the internal DRAM controller is enabled, the SAMEPAGE signal is used exclusively by the internal DRAM controller to generate –RAS signals.

E3.8 Chip Selects

Chip selects of MB86936 behave as MB86930 for –CS0 to –CS3. Two other chip selects, –CS4 and –CS5, are assigned to the internal DRAM controller.

E3.8.1 Chip Select 4, –CS4

When the internal DRAM controller is used, ASSR[4] and AMR[4] are assigned to support DRAM address space. The internal DRAM controller, once enabled, responds to –CS4 only. Wait state generation and internal REARY generation for –CS4 are programmed in DRAM control registers, described in DRAM controller chapter. Therefore WSSR[2] has to be disabled. In addition, DRAM memory space referred by

–CS4 is always cacheable, with a memory data bus width of either word or halfword only.

When the internal DRAM controller is disabled, –CS4 can be used as the fifth chip select for other address space.

E3.8.2 Chip Select 5, –CS5

When the internal DRAM controller is used, ASSR[5] and AMR[5] are assigned to non-cacheable DRAM address space. If all DRAM space is to be cacheable, memory space referred to by –CS5 has to lie outside of DRAM space. Chip select 5 is also not seen as a pin when the internal DRAM controller is enabled.

The chip select signal is sent out on –RAS1 pin when the internal DRAM controller is disabled. Chip select 5 can be used to point to the sixth address space.

E3.9 Wait State Specifier Register

E3.9.1 Purpose

The Wait-State Specifier Register (WSSR) format on the MB86936 has been changed from that on the MB86930 to accommodate the burst mode bus transaction using internal –READY and Parity generation/checking.

Since the wait–state of the DRAM cycle is determined by the internal DRAM controller when it is enabled, the WSSR for –CS4 should be disabled when the internal DRAM controller is used.

E3.9.2 Format

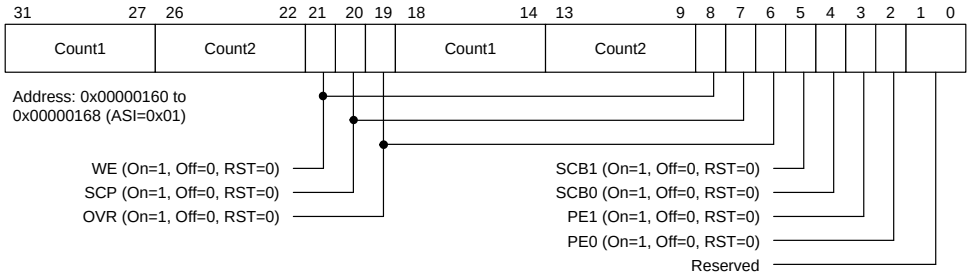


Figure E3-11. Wait State Specifier Register

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

E3.9.3 Wait State in –CS0 to –CS3, –CS5

Count1: Count1 +1 is the number of wait states inserted before internal –READY is asserted during write operations when SCP=0.

Count2: Count2 +1 is the number of wait states inserted before internal –READY is asserted during read operations when SCP=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: Unused; should be 0.

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

Note:
Wait state in –CS5 is meaningful only when the internal DRAM controller is disabled when the internal DRAM Controller is enable, WE bit of –CS5 should be 0.

E3.9.4 Wait State in –CS4

WE: Wait Enable, enables or disables the internal wait state generation for the individual address range. If WE is 1, SCP must be 0. This bit should be 0 if the internal DRAM Controller is enabled.

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.

a) In Burst Mode:

Burst mode enabled and –BMACK is asserted.

Count1: For –CS4, Count1 +1 is the number of wait states inserted before internal _READY is asserted for the first access of a burst mode transfer.

Count2: For –CS4, 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.

b) Not in Burst Mode:

b1) Burst mode enable and –BMACK is not asserted.

Count1 + 1: Count1 + 1 is the number of wait states inserted before internal _READY is asserted.

b2) Burst mode disable.

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.

Table E3-2: 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

E3.9.5 Wait State Generation

The MB86936 Wait-State Specifier Register (WSSR) format is the same as the MB86932 Wait-State Specifier Register format. MB86936 wait state generation, however, differs as follows:

(1) For –CS[5] and –CS[3:0], wait state generation differs for read and write operations. For read operations, the number of wait states is Count2 +1; for write operations, the number of wait states is Count1 +1.

(2) For –CS[4], wait state generation is the same as in the MB86932, when the internal DRAM controller is disabled. Wait state generation for –CS[4] with WSSR is invalid with internal DRAM Controller.

Note:
The wait state counter is clocked by the BIU clock, which is the external system clock

E3.10
8/16 Bit Bus Mode

The MB86936 processor supports 8/16-bit Bus Mode in the same way as the MB86930 processor. The MB86936 also supports 8/16-bit Bus Mode write operations as follows:

(1) In 8-bit Bus Mode, {ADR<27:2>, –BE2, –BE3} is the store address. BIU stores as many cycles as required only. For example, a store byte requires only one cycle, store halfword requires two cycles and store word requires four cycles on 8-bit Bus. 8–bit DRAM access is not supported using the internal DRAM Controller.

(2) In 16-bit Bus Mode, {ADR<27:2>, –BE2} is the store address, and –BE[1:0] are the byte enables. –BE1 enables the upper byte (D[15:8]), and –BE0 enables the lower byte (D[7:0]). For a store byte or store halfword, BIU executes one cycle. MB86936 takes two cycles to store word in 16-bit Bus Mode.

E3.10.1 Purpose

The data bus of the MB86936 can be configured to 8- and 16-bit bus modes as well as the standard 32-bit mode. This flexibility accommodates those cases in which code or data resides in memories organized as blocks of bytes or halfwords.

E3.10.2 Features

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

- 8-bit mode: D[7:0], Byte 3
- 16-bit mode: D[15:0], Byte 2-3
- 32-bit mode: D[31:0], Byte 0-3

E3.10.3 Bus Configuration

Upon reset, two external pins, –BMODE16 and –BMODE8 are used to determine the bus configuration of memory space referred by –CS0. 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

Bus width for memory space referred by –CS0 is determined by –BMODE16, –BMODE8 value when –RESET is asserted. Bus width for memory space referred by

–CS1 to –CS5 is programmed by writing to the Bus width and Cacheable Control Register (ASI = 0x1, ADR = 0x0000016C).

Table E3–3: –CS0 Bus Width Configuration

–BMODE16	–BMODE8	–CS0 Bus Width
0	0	Illegal
0	1	16-bit Memory Bus
1	0	8-bit Memory Bus
1	1	32-bit Memory Bus

In –CS1 to –CS5, the bus configuration is determined by the Bus Width and Cacheable Control Register (ASI=0x1, ADR=0x16c). Bus width for memory space referred by –CS1 to –CS5 can be programmed by writing to the Bus Width and Cacheable Control Register.

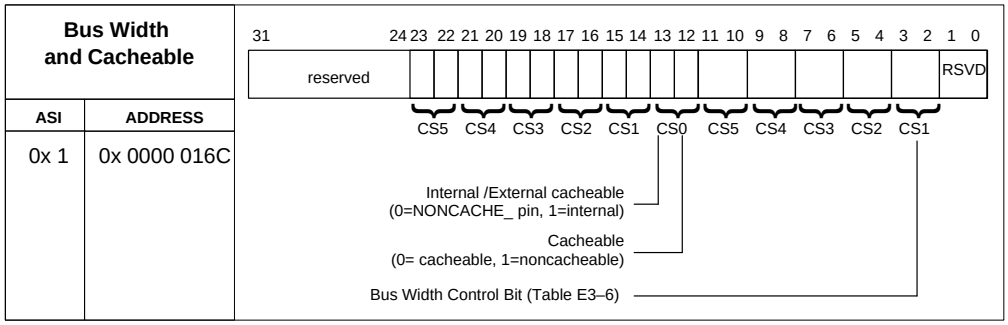


Table E3-4. Bus Width Control Bits of –CS1 to –CS5

BW1	BW	Bus Width
0	0	32-bit Memory Bus
0	1	8-bit Memory Bus
1	0	16-bit Memory Bus
1	1	Illegal

Memory space referred by each chip select of –CS1 to –CS5 can be individually programmed for 8/16/32 bit bus mode. Upon reset, 32-bit bus mode is the default.

BW1 and BW0 are activated high while –BMODE16, –BMODE8 are asserted low.

If the internal DRAM controller is enabled for –CS4 and –CS5, 8-bit bus is meaningless since the internal DRAM controller only supports 16- and 32-bit mode. If the internal DRAM controller is disabled, –CS4 and –CS5 do support 8-bit memory bus.

E3.10.4 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[27:2] and selected –BE bits can be concatenated to form a complete address for a non-32 bit bus mode.

Table E3-5: 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[27:2], –BE[2], –BE[3]}
16-bit bus mode address={ADR[27:2], –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 define and interface to the boot memory address space. 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.

E3.10.5 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 the boot memory, loading data from memory mapped I/O, and storing data to non-boot memory address space. All loads from any 8/16-bit memory 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. In order to meet the –BE AC timing, the –BE bits on the MB86936 need to be all 0’s for all types of loads—word, halfword, and byte—from the 32-bit memory 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.

A summary of the –BE[0:3] bit behavior for loads from the 8/16-bit bus 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 E3-6: Load –BE[0:3] Bit Sequence in Load Operation

Bus Mode	–BE[0:3] Sequence
8-bit bus	0000=>0001=>0010=>0011
16-bit bus	0000=>0010
32-bit bus	0000

E3.10.6 Burst Mode

Since speed is not a critical issue when executing out of 8/16-bit memory space, burst-mode is not supported for accesses to non 32-bit address space. When the system has a 8/16 bit memory being used, it should not assert –BMACK for any accesses.

E3.10.7 Memory Exception

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

E3.10.8 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.

E3.10.9 Read Timing

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

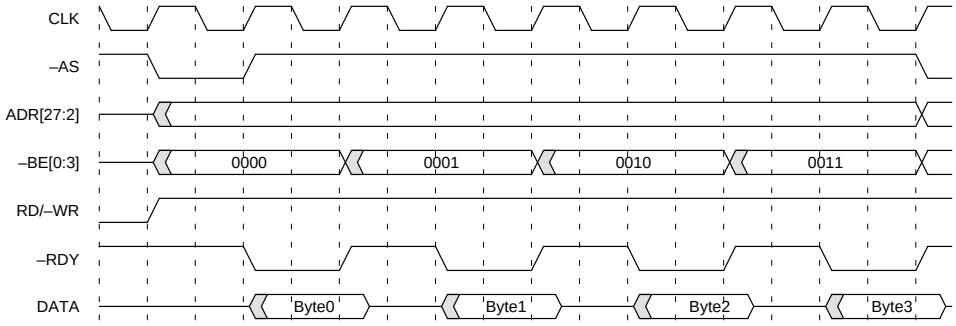


Figure E3-12. 8-bit Bus Mode Read (1 Wait State)

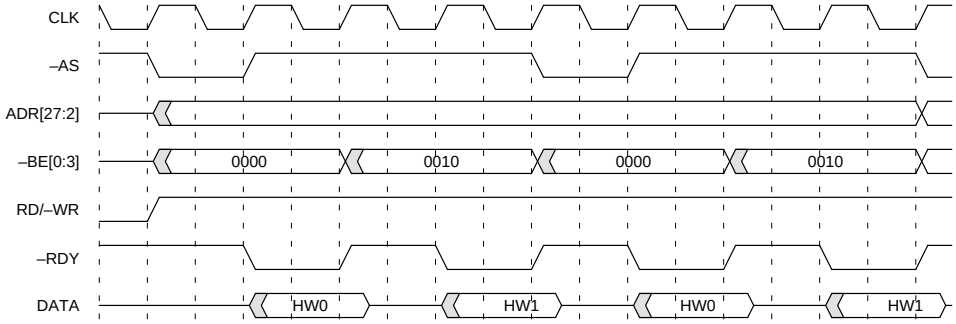


Figure E3-13. 16-bit Bus Mode Read (1 Wait State)

E3.10.10 Write Timing

Timing examples for the 8- and 16-bit bus mode write with 1 wait-state memory are shown below. Note that –AS is asserted for every cycle of write. The order of byte/halfword write is the reverse of the order of byte/halfword read.

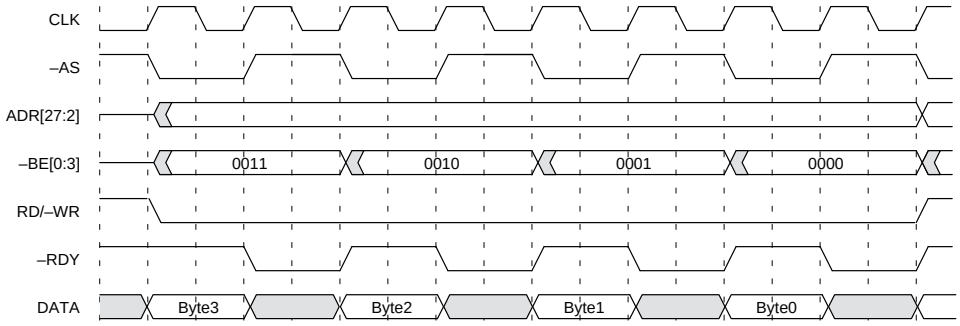


Figure E3-14. 8-bit Bus Mode Write (1 Wait State)

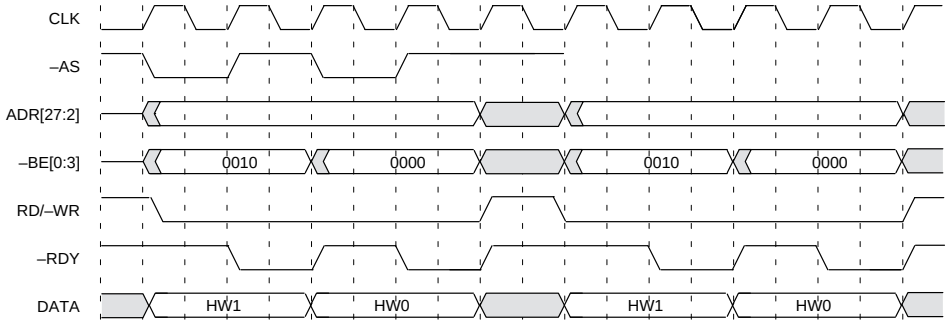


Figure E3-15. 16-bit Bus Mode Write (1 Wait State)

E3.11Boot Code Address Space

The boot code address space is defined by the –CS0 address-range specifier. On reset, the –CS0 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 –CS0. An example of the supervisor address space (ASI=0x9) memory map is shown below:

Figure E3-16. Supervisor Address Space (ASI=0x9) Memory Map

Any memory access from the boot-code 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.

E3.12 Non–cacheable Memory Access

Non–cacheability of MB86936 has been improved from MB86930. The user of MB86936 has the flexibility of setting different data memory space to be non–cacheable through either software programming or hardware control.

If the user decides to use the non–cacheable feature, bit7 of Cache/Bus Interface unit Control Register (ASI=0x01, ADR=0x0) should be set to 1. The bit is 0 on reset indicating that the feature is disabled.

The software or hardware control of non–cacheability is determined for each chip–select by programming the appropriate bit(s) in Bus–width and Cacheability Control Register (ASI=0x01, ADR=0x16C).

Table E3–7: Non–cacheability Control Bits

[12]	0 = Cacheable, 1 = Non–cacheable in –CS0 memory space
[13]	0 = Hardware, 1 = Software for –CS0 region
[14]	0 = Cacheable, 1 = Non–cacheable in –CS1 memory space
[15]	0 = Hardware, 1 = Software for –CS1 region

[16]	0 = Cacheable, 1 = Non–cacheable in –CS2 memory space
[17]	0 = Hardware, 1 = Software for –CS2 region
[18]	0 = Cacheable, 1 = Non–cacheable in –CS3 memory space
[19]	0 = Hardware, 1 = Software for –CS3 region
[20]	0 = Cacheable, 1 = Non–cacheable in –CS4 memory space
[21]	0 = Hardware, 1 = Software for –CS4 region
[22]	0 = Cacheable, 1 = Non–cacheable in –CS5 memory space
[23]	0 = Hardware, 1 = Software for –CS5 region

Note that non-cacheability control bits for –CS4 and –CS5 is valid only when the internal DRAM controller is disabled. When the internal DRAM controller is enabled, –CS4 is always cacheable and –CS5 is always non-cacheable.

E3.12.1 Hardware Non–cacheability

The advantage of hardware control is that the non–cacheability memory space can be smaller than the memory space referred by a chip–select. The correct logic value of –NONCACHE signals has to be available when the signal is expected to be valid.

To relax the timing, bit 8 and bit 9 of Cache/Bus Interface Unit Control Register indicates the cycle, after –AS is asserted, the –NONCACHE signal should be valid. –NONCACHE signal of logic 1 indicates cacheable, and logic 0 indicates non–cacheable, and last only one cycle.

Table E3–8: Cacheability Valid Timing

[9:8]	Cycle after –AS
00	same cycle
01	1 cycle
10	2 cycles
11	3 cycles

The –NONCACHE signal has to be valid at least one cycle before –READY is asserted. With the above statement, the fastest memory access is 1 wait state with hardware configuration.

E3.12.2 Software Programming of Non–cacheability.

If the entire memory space is pre–determined to be non–cacheable, the user can program the appropriate bit(s) in Bus–Width and Cacheability Register (see table E3–7).

E3.12.3 Internal DRAM Controller Enabled

When the internal DRAM controller is enabled, –CS4 is always cacheable, and –CS5 is always non–cacheable. If the entire DRAM is to be cacheable, memory space referred by –CS5 has to be outside the –CS4 memory space. This mode is valid only when the cacheability feature is enabled (bit 7 of Cache/Bus Interface Unit Control Register is set to logic value 1)., and bit 8 in the Instruction Fault Status Register is set (see TLB section)

E3.13 Write Buffers

To reduce the external memory bus traffic, MB86936 includes four deep write buffers, allowing the processor to write up to four word/double word data, while external bus is busy. The write buffers request a store as soon as the external bus is available and the buffer is not empty.

To maintain data consistency, any instruction fetch or data load from external memory is preceded by write buffer empty

E3.13.1 Programming the Write Buffer

Write buffer is utilized only when both data cache, instruction cache and write buffer are enabled. On write bit with data cache locked, the data is not written into the write buffer.

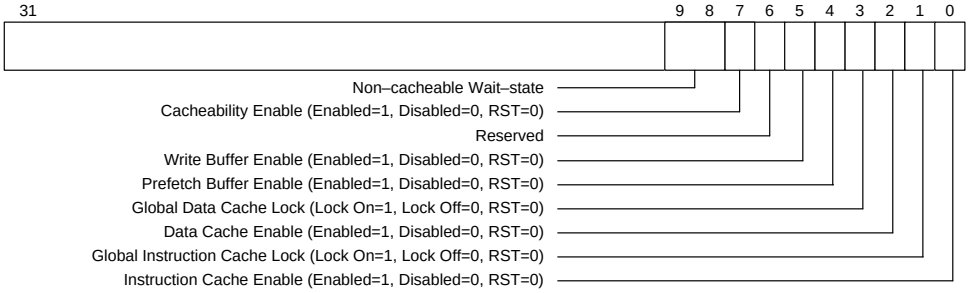


Figure E3-17. Cache/Bus Interface Unit Control Register

Bits 31–10: Reserved

Bits 9–8: Non–cacheable wait state. These two bits set the wait state cycle when –NONCACHE pin is valid.

Bit 7: Setting of this bit selects cacheability feature.

Bit 6: Reserved

Bit 5: Write Buffer Enabled—When set to 1, enables the write buffer of the BIU only if both the instruction and data caches are enabled. At reset, this bit is 0. This bit should be changed only when the instruction and data caches are off.

Bit 4: Prefetch Buffer Enabled—When set to 1, enables the prefetch buffer of the BIU only if both the instruction and data caches are enabled. At reset, this bit is 0. This bit should be changed only when the instruction and data caches are off.

Bit 3: Global Data Cache Lock—Locks the current entries into the on-chip data cache; with this bit set to 1, no valid entry in the data cache will be replaced. To insure the best performance with the cache locked, invalid words in allocated cache locations will be updated. On write hits, with the data cache locked, the data is not written to external memory, allowing the locked cache to be used as scratchpad RAM or a run-time stack, independent of main memory. When the Data Cache Lock bit is 0, the cache operates normally. At reset, this bit is 0.

Bit 2: Data Cache Enable—Turns the on-chip data cache on (1) and off (0). At reset, this bit is 0.

Bit 1: Global Instruction Cache Lock—Locks the current entries into the on-chip instruction cache; with this bit set to 1, no valid entry in the instruction cache will be replaced. To insure the best performance with the cache locked, invalid words in allocated cache locations will be updated. When this bit is 0, the cache operates normally. Writes to the Instruction Cache Lock bit do not affect cache operation for the following three instructions. At reset, this bit is 0.

Bit 0: Instruction Cache Enable—Turns the on-chip instruction cache on (1) and off (0). Writes to the Instruction Cache Enable bit do not affect cache operation for the following three instructions. At reset, this bit is 0.

E3.14 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 enabled and not empty, the store will occur.
- 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 bit 1 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

E4

MB86936 DRAM Controller

E4.1 Overview

The primary function of a DRAM controller is to convert the memory access signals generated by the CPU into signals which the DRAM requires in order to read or write data. To reduce pin count, the DRAM address is multiplexed.

Memory is arranged in a rectangular matrix where individual elements are located at a specific row and column address. The row and column addresses are derived directly from the address issued by the CPU through an address multiplexer. The DRAM uses the row address strobe (–RAS) to capture the row address. In page–mode DRAMs, the data in an entire row is then latched as a single vector or 'page' in the DRAM. Once in this 'page', the individual 'column' elements can be accessed faster than when accessing them in a large array. Subsequent accesses to the same page are called 'page–mode' accesses. Only the column address is needed to perform a page–mode access. The DRAM uses the column address strobe (–CAS) to capture the column address. An additional signal (–DWE) is used to distinguish between a read and a write access.

Since DRAMs are volatile, they must be refreshed periodically. This means that a timer is needed to request refresh at programmed intervals.

The MB86936's integrated DRAM controller provides an address multiplexer, a refresh timer, a page comparator and a programmable state machine to govern the timing relationships of the multiplexed row and column address and the DRAM control

signals. It can be used with DRAMs having different access times in a wide range of operating frequencies.

The DRAM controller is targeted at supporting the most common type and the lowest cost DRAM: the page–mode DRAM with CAS before RAS refresh. CAS before RAS refresh uses the DRAM's internal address counter to specify the row to be refreshed and avoids the added cost of an external counter. It does not support interleaving of memory banks.

E4.2 Registers

The registers used in configuring and activating the DRAM controller are described below.

E4.2.1 System Support Control Register

This register is used to enable the DRAM controller. It also controls burst support for DRAM accesses and enables the refresh timer. DRAM wait states are set by the DRAM Timing Registers. If the Programmable Wait–State Enable bit is set then the Wait Enable bit of the Wait State Specifier Register for CS4 and CS5 must be cleared.

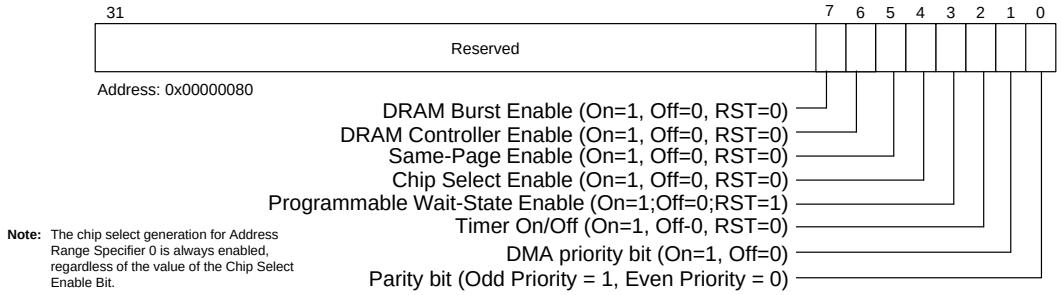


Figure E4–1 System Support Control Registers

Bits 31-6: Reserved

Bits 6-7:

Bit 5: Same-Page Enable—Enables (1) and disables (0) the same-page detection logic. When this bit is 1, the –SAME_PAGE signal is asserted whenever the address of an external access is on the same page as the previous access. The page size is controlled by the Same-Page Mask Register (see above). When this bit is 0, –SAME_PAGE is never asserted. The Same-Page Enable bit is cleared to 0 on reset.

Bit 4: Chip Select Enable—Enables (1) and disables (0) the generation of chip-select signals for external accesses in address ranges 1 through 5. Regardless of the state of this bit, however, –CS0 is always asserted when the current address lies in address range 0. The Chip Select Enable bit is cleared to 0 on reset.

Note: Before enabling chip selects all chip select Address Mask and Address Range registers should be initialized so that two chip selects are never selected at the same time.

Bit 3: Programmable Wait-State Enable—Enables (1) and disables (0) the programmable wait-state generators for all address ranges. The Programmable Wait-State Enable bit is set to 1 on processor reset. The DRAM Controller does not use this bit to set wait states.

Bit 2: Timer On/Off—Enables (1) and disables (0) the timer. This bit is cleared to 0 on reset.

Bit 1: When this bit is set, the BIU is shared equaly between the DMA and BIU. If both units are requesting the bus, they will alternate bus accesses. When this bitis cleared, the DMA has exclusive use of the bus for as long as DMA is requesting the bus.

Bit 0: 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.

Table E4-1: System Support Register Summary

Chip Selects	Affected by Chip-Select Enable?	Address Range Specifier		Address Mask		Wait-State Specifier	
		Address (ASI=0x01)	Value at Reset	Address (ASI=0x01)	Value at Reset	Address (ASI=0x01)	Value at Reset
0	No	N/A	ASI=0x09 ADR<31:10>=0	0x0000 0140	All mask bits 0 except ADR<14:10> = 1	0x0000 0160 (low halfword)	Count 1,2 = 31 Wait Enable=1 Single Cycle =0 Override=1

E4.2.2 DRAM Bank Configuration Register

This register configures the DRAM Controller for the DRAM type and bank size that it will support.

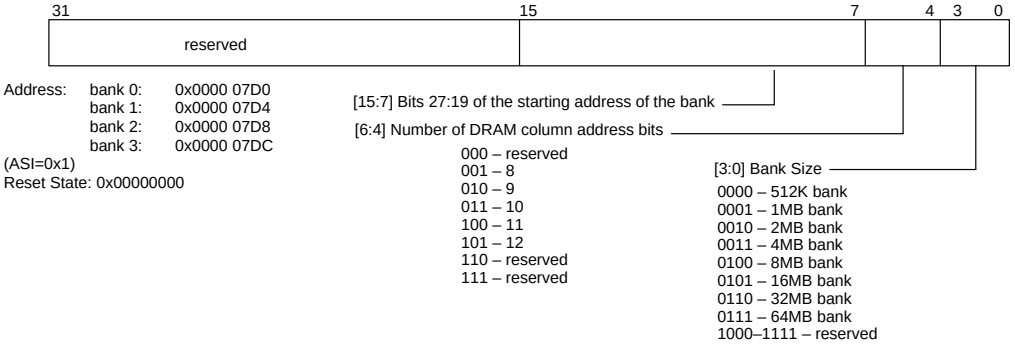


Figure E4–2 DRAM Bank Configuration Registers

DRAMs are specified by bit density, ie. the total number of bits in the chip. For a given bit density, a DRAM is available in different depth and data width combinations. For example, a 16Mbit DRAM can be found in a 2Mx8, 4Mx4, or 16Mx1 arrangement. To form a 32–bit memory bank, the number of chips required are four, eight and thirty two, respectively. The corresponding bank sizes will be 8MB, 16MB, and 64MB. The number of bits used for the row and column addresses given in the DRAM specification reflects its unternal structure.

The bank size, number of column bits, and starting address of the bank are written to the DRAM Configuration Registers, one for each bank and up to four banks. The number of DRAM column address bits will be used to control how the physical address is divided into row and column address bits. The DRAM must have at least eight column bits and can go up to 12 column address bits. The DRAM can have up to 12 address pins. Any page–mode DRAM from 256Kxn to 16Mxn (where n is the number of data bits) can be used.

The DRAM address space is defined by the Address Range Specifier Register and Address Mask Register for CS4. This address space is further subdivided into banks of addresses. The starting address of a bank and the bank size will determine the address range of a particular bank. Addresses falling in the range of addresses defined for a bank will activate the corresponding –RAS. Each –RAS corresponds to a particular bank.

For example, a 32 bit bank of memory using 1Mxn DRAMs will yield 4 megabytes. If it is desired that this bank start at address 0x0000000, and the DRAM has 10 column and 10 row bits, then bits 27:19 of the DRAM Bank Configuration Register will be set to '000000000', bits 6:4 set to '011' to indicate 10 column bits, and bits 3:0 set to '0011' to indicate a 4MB bank size.

The DRAM address space may be divided into the different banks in any order. The largest bank does not have to be bank 0. However it is important to observe the restriction that the largest bank occupy the lowest addresses in the DRAM space.

E4.2.3 DRAM Timing Register 1 & 2

Together, these two registers contain a set of parameters which control the timing relationships of the DRAM Controller signals. They affect all of the banks uniformly. Once these registers are written, the timing relationships of the multiplexed address, –RAS, –CAS, and –DWE are the same regardless of which bank is accessed. These registers are cleared during reset.

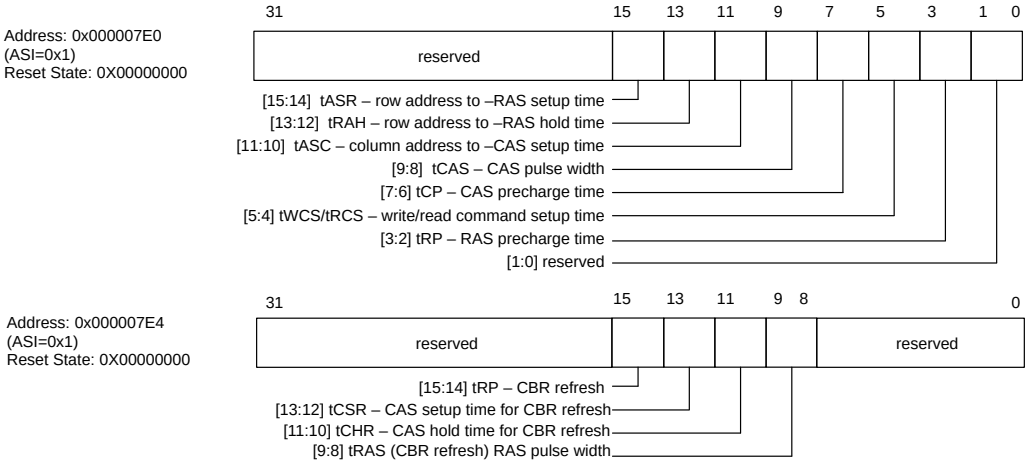


Figure E4–3 DRAM Timing Registers

Table E4–2: Timing Register Parameter Settings

$[x_1:x_0]$	# Internal Clock Cycles
00	1 cycle
01	2 cycles
10	3 cycles
11	4 cycles

The internal clock cycle is the unit of reference for all parameters in the Timing Registers. The default for these parameters is one internal clock cycle. If no clock doubling is used, the internal clock cycle time is identical to the external bus cycle time. These parameters specify the number of wait cycles incurred between the state transitions in Figure E2–3.

The BIU will latch the data on D[31:0] at the end of phase 2 of the external bus clock. The DRAM controller asserts an internal DATA–READY signal on the last cycle that –CAS is active.

Table E4–3: Timing Register Parameters

tASR	row address to –RAS setup time. –RAS will be asserted one to four internal clock cycles after the row address change to provide the address to –RAS setup time.
tRAH	row address hold time. The row address to column address switch will occur one to four internal clock cycles after –RAS is asserted to satisfy the row address hold time requirement.
tASC	column address setup time. –CAS is asserted one to four internal clock cycles after the row to column address change to provide the address to –CAS setup time.
tCAS	–CAS pulse width. –CAS will be deasserted one to four internal clock cycles after –CAS is asserted
tCP	–CAS precharge time. –CAS will remain deasserted a minimum of one to four internal clock cycles after it is deasserted. Note that –CAS will remain deasserted if there are no DRAM accesses.
tWCS,tRCS	Write/Read command setup time. –CAS is asserted from one to four internal clock cycles after –DWE is changed.
tRP	–RAS precharge time. –RAS will remain deasserted for at least one to four internal clock cycles after it is deasserted. It will remain deasserted until the –RAS active state is reached. (See State Diagram)
tRP (CBR)	–RAS precharge time for CBR refresh. The transition from the IDLE state and the CBR –CAS active state can be one to four internal clock cycles. This can be used to lengthen the –RAS precharge time.
tCSR (CBR)	–CAS to –RAS setup time for –CAS before –RAS refresh. –RAS will be asserted from one to four internal clock cycles after –CAS is asserted.
tCHR (CBR)	–CAS to –RAS hold time for CBR refresh. –CAS will be deasserted from one to four internal clock cycles after –RAS is asserted.
tRAS (CBR)	–RAS will be deasserted from one to four internal clock cycles after –CAS is deasserted. This can be used to lengthen the –RAS pulse width during CBR refresh. The –RAS pulse width during CBR refresh is the sum of tCHR(CBR) and tRAS(CBR).

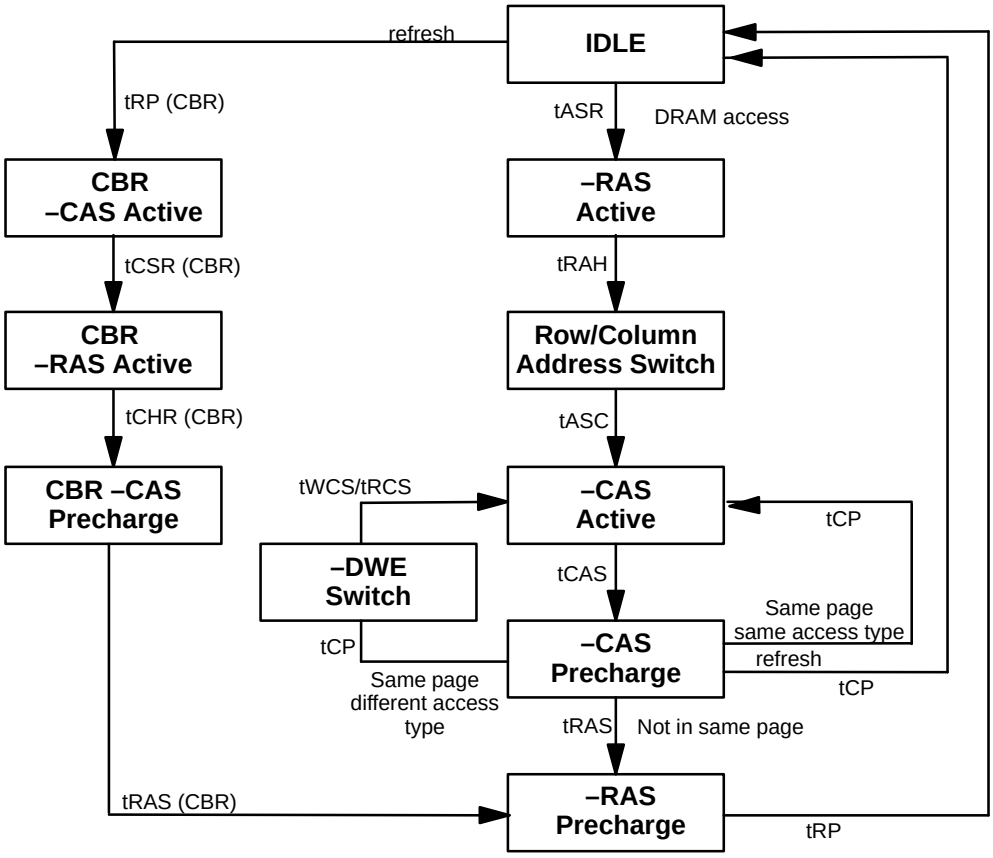


Figure E4-4. State Diagram

E4.2.4 Same Page Mask Register

If the current DRAM access and the previous DRAM access share the same DRAM row address, then these accesses are said to be in the 'same' page . These accesses need not be sequential. Accesses to the same page in DRAM are faster than accesses to different pages since –RAS remains asserted and only the column address needs to be changed ie. –CAS will be reasserted. The Same Page Mask Register sets the size of a page. A page refers to the number of column locations in a given row. The number of column address bits in the DRAM determines the page size. For example, a 1Mx4 DRAM has 10 column address bits and its page size is 1K. If more than one bank is used and the page sizes differ between banks, then the Same Page Mask Register should be programmed for the smallest page size.

This register controls which bits of the current address and ASI will be compared with the previous address and ASI. If the unmasked bits in the current address and ASI match with the bits in the previous address and ASI then the current access is in the same 'page' as the previous access. The DRAM address range must not be accessed during the three cycles after this register pair is written.

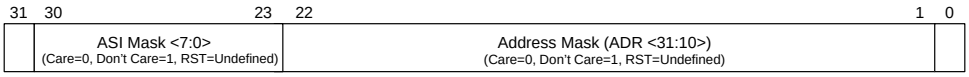


Figure E4–5. Same-Page Mask Register

Bit 31: Reserved

Bits 30-23: ASI Mask—Specifies which bits in the ASI of the current external access are to be compared with the corresponding bits in the ASI of the previous access. Only those bits are compared for which the mask bit is 0. Mismatches in any other bits do not prevent the two accesses from being recognized as "on the same page." The bits of this field are cleared to 0 on reset.

Bits 22-1: Address Mask—Specifies which of the 22 most significant bits in the address of the current external access are to be compared with the corresponding bits in the address of the previous access. Only those bits are compared for which the mask bit is 0. Mismatches in any other bits do not prevent the two accesses from being recognized as "on the same page." The bits of this field are cleared to 0 on reset.

Bit 0: Reserved

Table E4–4 Same Page Mask Register Values with ASI not masked

# Column bits	16–bit	32–bit
8	not used	0x00000000
9	0x00000000	0x00000002
10	0x00000002	0x00000006
11	0x00000006	0x0000000e
12	0x0000000e	0x0000001e

E4.2.5 Address Range Specifier Register 4 and Address Mask Register 4

Address Range Specifier Register 4 is used in conjunction with Address Mask Register 4 to select which portion of the 4GB address space is assigned as the DRAM address space. Chip Select 4 (CS4) will be asserted whenever an access is made to an address which falls in the range specified by this register pair. The starting address of a bank must be on a bank size boundary (eg. a 4MB bank can start at 0x00000000, 0x00400000, etc.) and the largest bank must occupy the lowest addresses followed by the next largest bank and so on. The DRAM address range must not be accessed during the three cycles after this register pair is written.

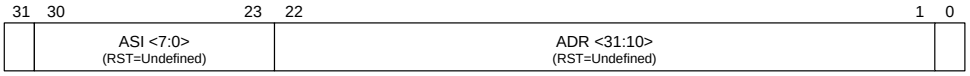


Figure E4–6. Address Range Specifier Registers

Bit 31: Reserved

Bits 30-23: ASI[7:0]—Specifies the ASI of a target address range. The value of this field is undefined on reset.

Bits 22-1: ADR[31:10]—Specifies the 22 most significant bits of a target address range. The value of this field is undefined on reset.

Bit 0: Reserved

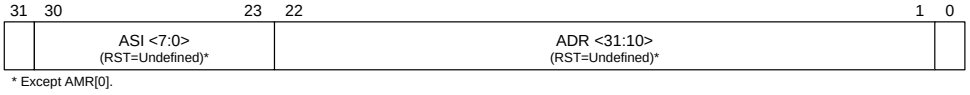


Figure E4–7. Address Mask Registers

Bit 31: Reserved

Bits 30-1: Mask—Specifies which bits in the ASI and address of the current external access are to be compared with the corresponding bits in the address-range specifier. Only those bits are compared for which the mask bit is 0. See Table E2-3 for reset value.

Bit 0: Reserved

Table E4–5: Programming the Address Mask Register

DRAM address space	Value in Address Mask Register 4
1MB	0x0000 07fe
2MB	0x0000 0ffe
4MB	0x0000 1ffe
8MB	0x0000 3ffe
16MB	0x0000 7ffe
32MB	0x0000 fffe
64MB	0x0001 fffe

E4.2.6 Timer Register and Timer Preload Register

These registers are used by the DRAM controller to set the refresh interval. The Timer Register contains the current count of a 16–bit timer. The Timer Preload Register contains a value which is loaded into the timer when the timer overflows. The timer overflows when its count decrements to zero. The processor will then assert the –TIMER_OVF signal externally. The DRAM controller will detect this internally and begin a refresh cycle. The minimum value written to the Timer Register is 0x01. Typically, the same value is written to both registers.

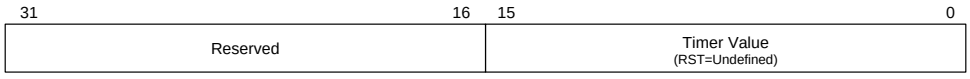


Figure E4–8. Timer Register

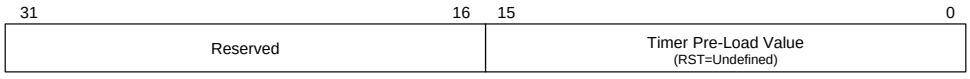


Figure E4-9. Timer Pre-Load Register

E4.2.7 Bus Width and Cacheability Register

The DRAM controller supports 32–bit and 16–bit wide memory. It does not support 8–bit wide memory. The data bus width is specified in this register.

Bits 9:8 and bits 11:10 specifies the bus width for CS4 and CS5, respectively. These settings must be the same. Table E4-6 defines bits 11:8 of this register.

Table E4–6: Bus Width Settings

Bus Width	bits [11:8]
32–bit	0000
16–bit	1010

E4.3 Burst Mode

Burst mode read operation is supported by the DRAM controller through the Bus Interface Unit (BIU). The BIU generates the necessary burst addresses and sends them to the DRAM controller in sequence. The DRAM controller will treat burst accesses like any other access to DRAM. During a burst operation no other memory access will be interleaved with the burst accesses. To enable DRAM burst, bit 7 of the System Support Control Register must be set. In addition, burst-mode must be enabled for either the D-cache or the I-cache, or both, by programming the Bus Control Registers (0X00000020 ASI=0X01). Burst-mode write is not supported.

E4.4 –CAS Behavior during Word, Halfword, and Byte Accesses

The DRAM controller has four –CAS output signals. Each –CAS controls a byte in a 32 bit memory system. –CAS0 controls byte0(the most significant byte in a Big Endian architecture such as SPARC, and –CAS<1:3> control the least significant three bytes, respectively. In a 16 bit system, –CAS2 controls the most significant byte (byte 2) and –CAS3 controls the least significant byte (byte 3). –CAS0 and –CAS1 are not used in a 16 bit system.

In a 32 bit system, a word access will cause all four –CAS signals to be asserted simultaneously during the –CAS active phase. An access to the most significant halfword will cause –CAS<0:1> to be asserted. An access to the least significant halfword will cause –CAS<2:3> to be asserted.

E4.5 Address Multiplexing

The multiplexed DRAM row and column address appear on ADR[13:2]. These pins should be connected to A12:A0 of the DRAMs. If ADR13, ADR12,ADR11 or ADR10 are unused they should be left unconnected. ADR[27:14] are unaffected by the DRAM controller and will reflect bits 27:14 of the physical address.

For 32–bit systems, the least significant address bit is A2. It is also the least significant column address bit. For 16–bit systems, the least significant address bit and the least significant column address bit is A1.

During the row address phase,the row address appears on ADR[13:2] with the least significant bit at ADR[2]. The row address' least significant bit will vary according to the number of column address bits in the DRAM. The column bits will appear on ADR[13:2] during the column address phase with the least significant bit (A2/A1 for 32/16 bit systems) going through ADR[2].

For example, given a DRAM with 11 row address bits and 10 column address bits to be used in a 32–bit memory system. The DRAM has 11 address pins and will be connected to ADR[12:2]. During the row address phase, A[22:12] will appear on ADR[12:2] and will be latched by –RAS. A[23] will appear on ADR[13], but since it is not connected to the DRAM, it is ignored.

During the column address phase, A[11:2] will appear on ADR[11:2]. A[13:12] will appear on ADR[13:12]. Both will be ignored since ADR[13] is unconnected and A[12] is not needed by the DRAM during the column address phase.

In general, all DRAM address pins should be connected to ADR[13:2] with ADR[2] connected to A[0] of the DRAM. If there are fewer than 12 address pins on the DRAM, then some of the ADR pins will be unconnected, as in the example above. The proper row and column addresses will appear on ADR[13:2] in sequence and only the relevant address bits will be latched by the DRAM.

Table E4–7: Address Multiplexing in a 32–bit System

# column addr bits	Row address	Column address	Output pins
8	A[21:10]	A[13:2]	ADR[13:2]
9	A[22:11]	A[13:2]	ADR[13:2]
10	A[23:12]	A[13:2]	ADR[13:2]
11	A[24:13]	A[13:2]	ADR[13:2]
12	A[25:14]	A[13:2]	ADR[13:2]

Table E4–8: Address Multiplexing in a 16–bit System

# column addr bits	Row address	Column address	Output pins
8	A[20:9]	A[12:1]	ADR[13:2]
9	A[21:10]	A[12:1]	ADR[13:2]
10	A[22:11]	A[12:1]	ADR[13:2]
11	A[23:12]	A[12:1]	ADR[13:2]
12	A[24:13]	A[12:1]	ADR[13:2]

E4.6 16–bit Operation

The smallest memory data bus width supported by the DRAM controller is 16 bits. The DRAM controller does not support 8 bit wide memory. When using a 16–bit data bus the BIU will make two accesses to load or store a word and one access to load or store a halfword. Instruction fetches involve two accesses.

E4.7 Refresh

–CAS before –RAS refresh is scheme used by the internal DRAM controller. All four –CAS signals are asserted while –RAS is deasserted. After appropriate setup and hold times, all four –RAS signals are asserted. –DWE is deasserted during refresh. Care must be taken to insure that sufficient power and ground are supplied to the DRAMs.

E4.8 Programming the DRAM Controller

The internal DRAM Controller is disabled after reset. The user completes the following initialization sequence before making accesses to DRAM through the internal DRAM controller.

- Allocate the CS4 and CS5 address spaces by writing the Address Range Specifier Registers and Address Mask Registers for CS4 and CS5.
- Further subdivide the CS4 address space into bank address spaces by writing the appropriate values in the DRAM Bank Configuration Registers. Insure that the largest bank occupies the lowest address range and that all banks are aligned on bank boundaries, eg. a 4MB bank can start at address 0x00000000, 0x00400000(4MB), 0x00800000(8MB), etc. The starting address of the bank, the number of column bits used by the DRAM and the size of the bank are programmed into the DRAM Bank Configuration Register.

- Program the refresh interval in the Timer Register and in the Timer Preload Register. The same value may be used for both registers. Refresh cycles will not occur until the DRAM Controller is enabled. Most DRAMs require eight CAS– before RAS– cycles to occur before this mode of refresh is recognized by the DRAMs. The DRAM space should not be accesses until eight refresh cycles has occurred.
- Program the Same Page Mask Register.
- Disable Wait State Generation for CS4– and CS5– by setting the WE bit to '0' in the Wait State Specifier Register for CS4– and CS5–, 0x00000168 ASI 0x01.
- Program the DRAM data width, either 16–bit or 32–bit into register 0x0000016C.
- Enable the DRAM controller, the Refresh Timer, the Chip Selects, and the Same–Page logic by writing the System Support Control Register 0x00000080 ASI 0x01.

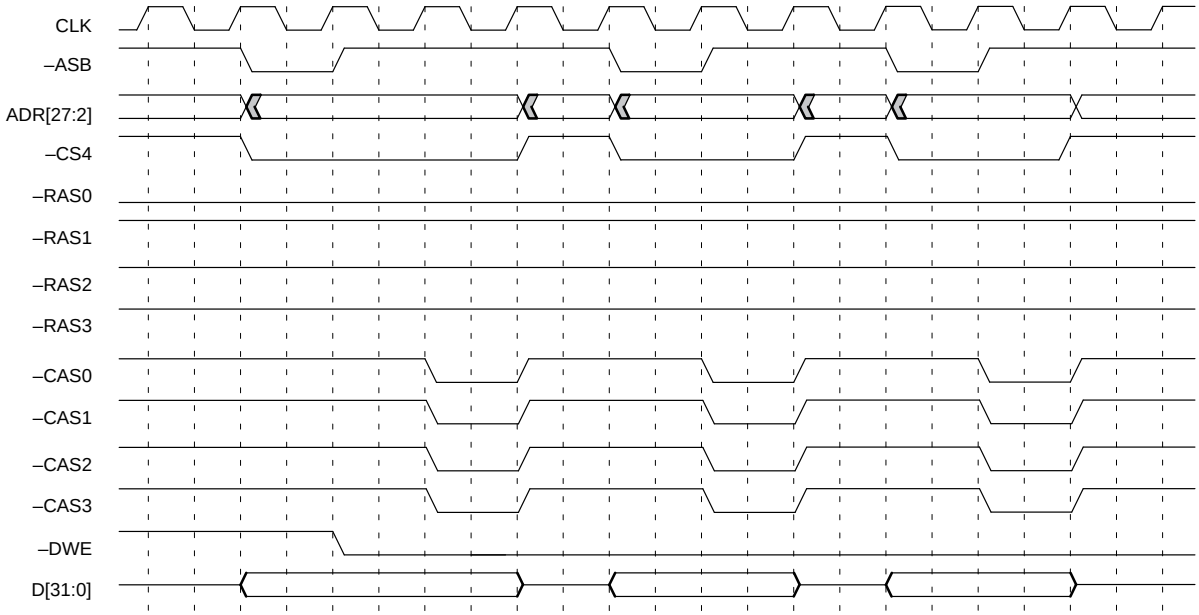


Figure E4–2 Back to Back Page_Mode Writes

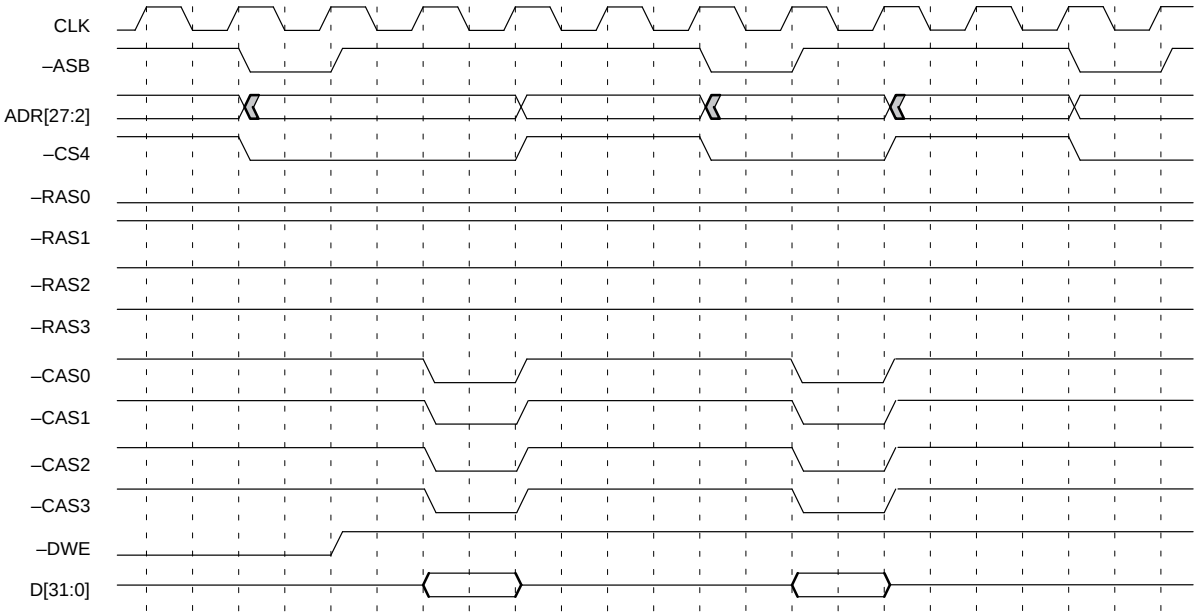


Figure E4–3 Back to Back Page_Mode Reads

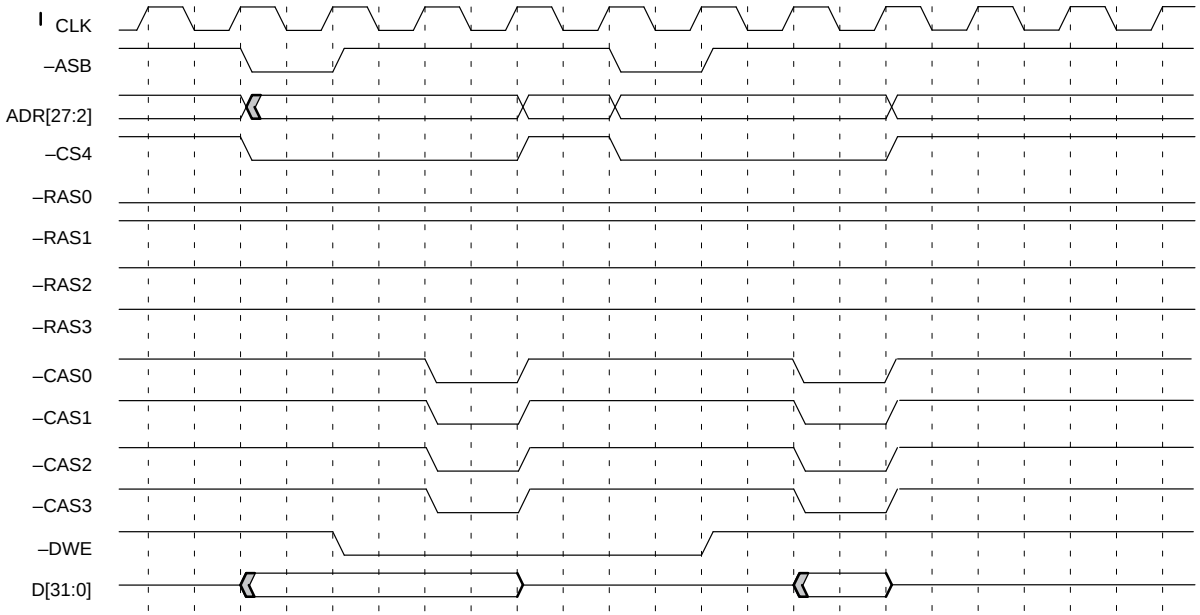


Figure E4–4 Page_Mode Write followed by a Page_Mode Read

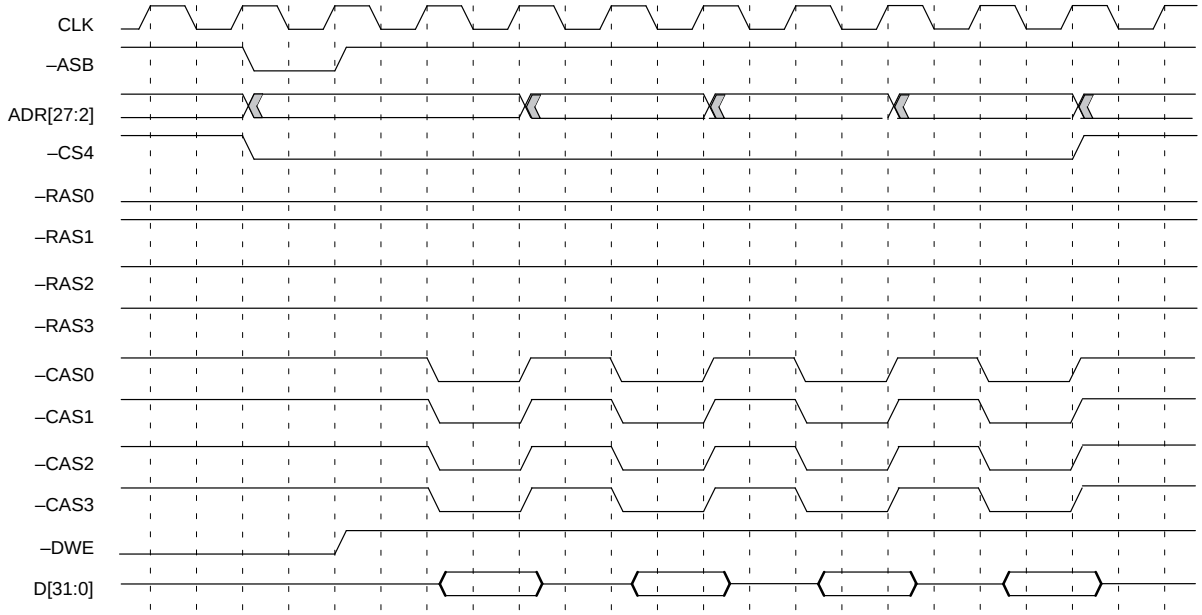


Figure E4–5. Burst Mode DRAM Read –– ICACHE and DCACHE Enabled

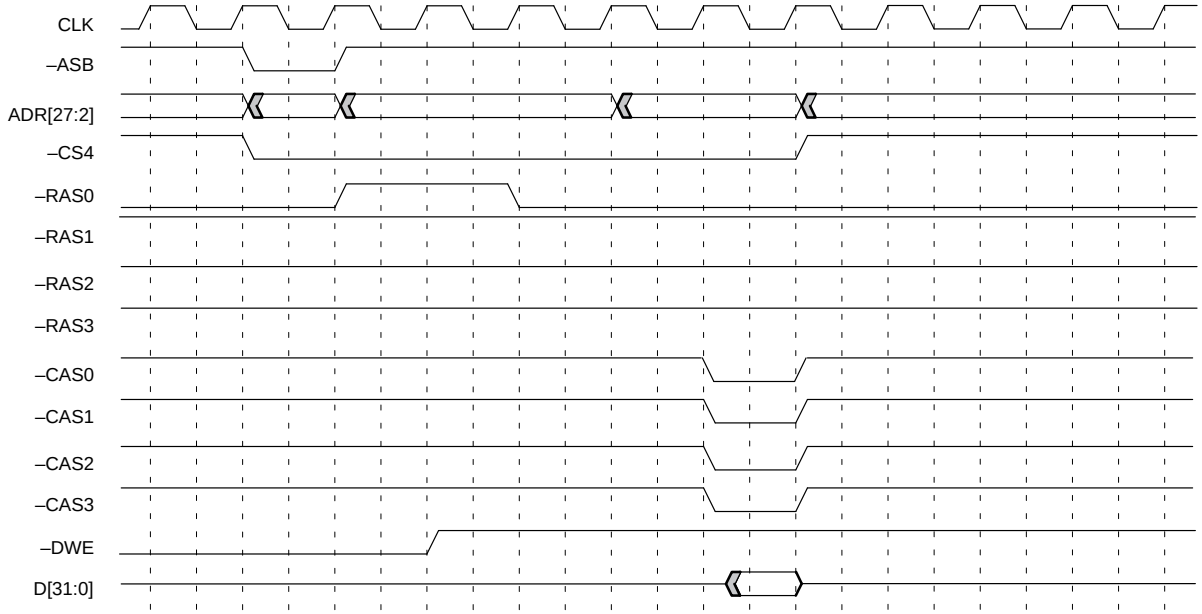


Figure E4–6. Non Page_Mode Read on Bank0 following another access to Bank0

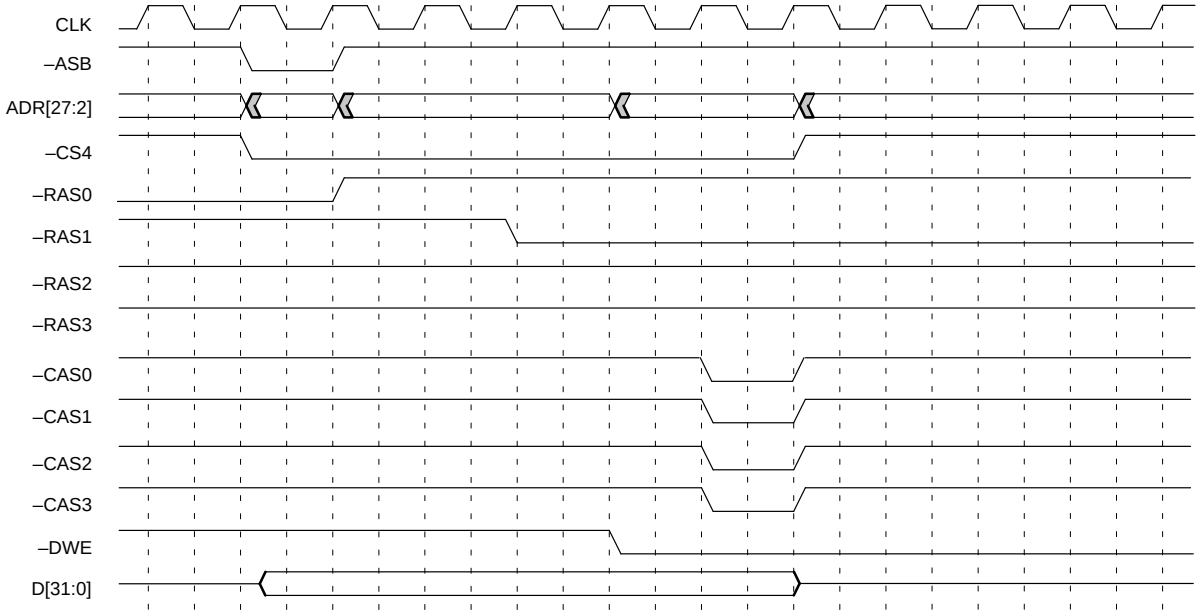


Figure E4–7. Non Page_Mode Write on Bank1 following an access to Bank0

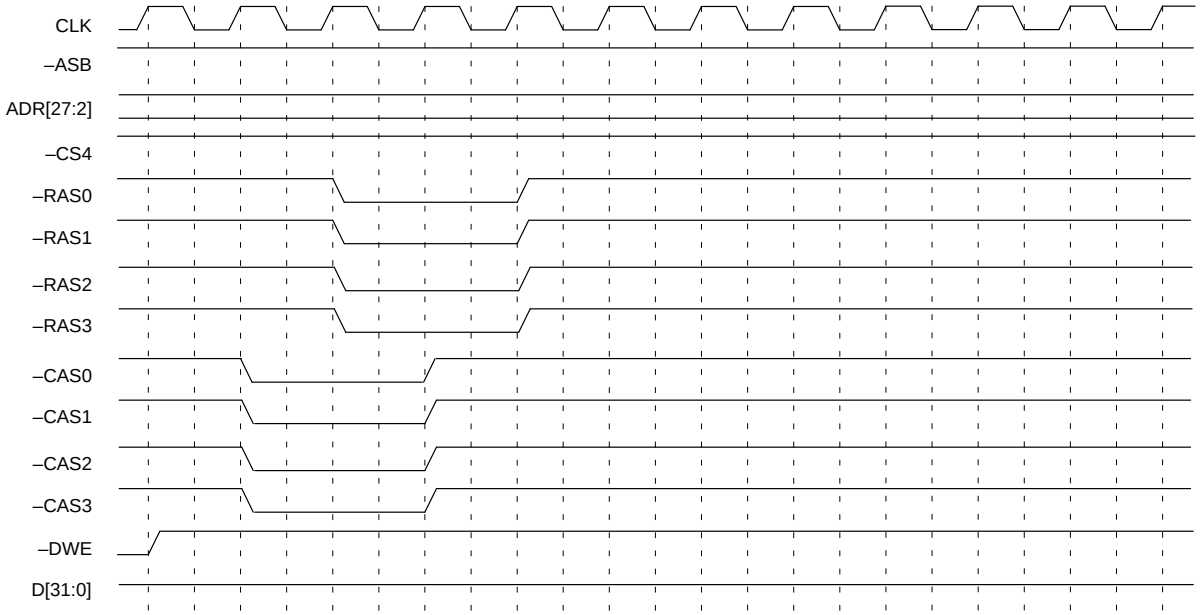


Figure E4–8. CAS before RAS refresh

CHAPTER

E5

MB86936 DMA

E5.1 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 three 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. At least one of the source or destination has unchanging address. “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 MB86936, and a detailed picture of the flow within the DMAC.

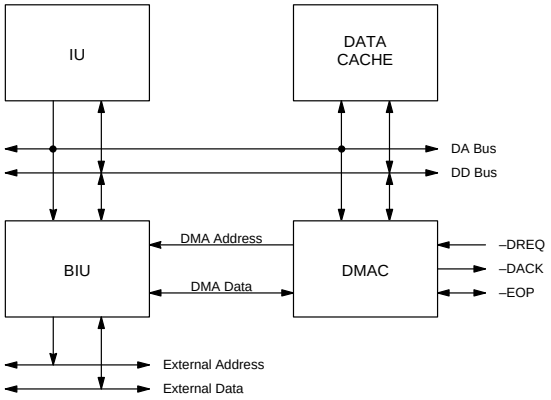


Figure E5-1. Relation of DMAC to Other Major Components

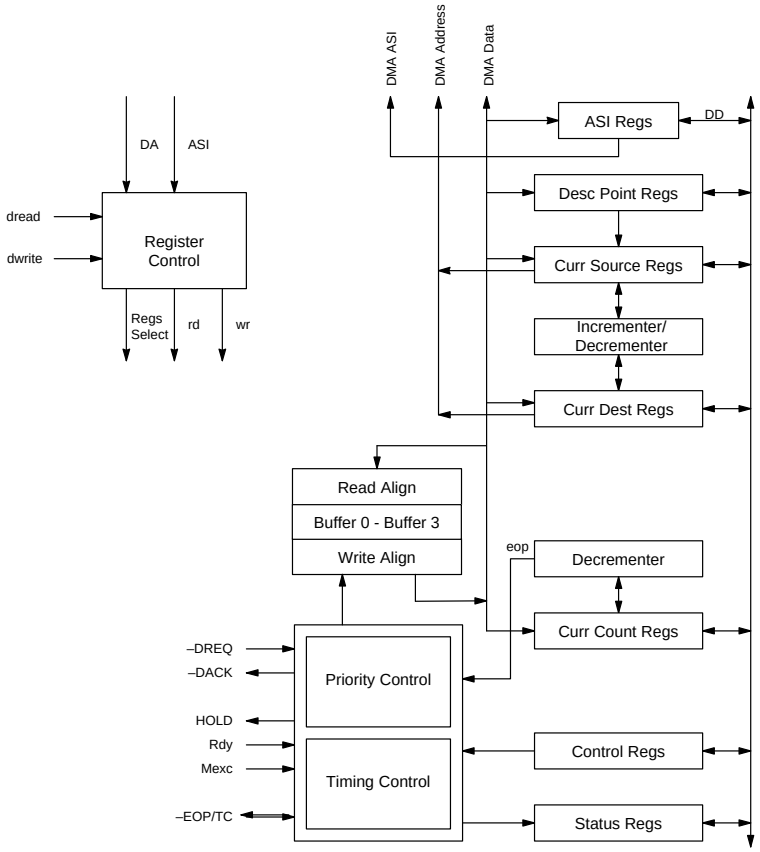


Figure E5-2. DMA Block Diagram

E5.2 Programmer's Model

Table E5-1: DMA Signal Descriptions

Signal	Function
–DREQ2 –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.
–DACK2 –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.
–EOP2 –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.

Nine 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.

E5.2.1 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 BIU is shared equally between DMA and the IU. If both units are requesting the bus, they will alternate bus accesses. When this bit is cleared, the DMA has exclusive use of the bus for as long as DMA is requesting the bus.

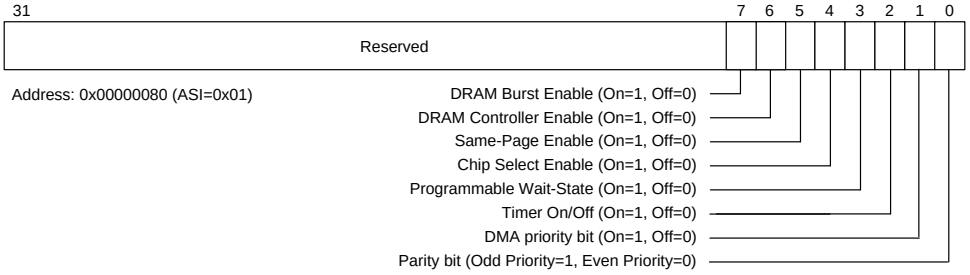
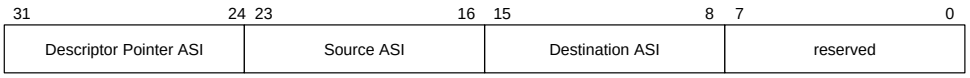


Figure E5-3. System Support Control Register

E5.2.2 DP/Source/Destination ASI Register



Address:0x00000180 (DMA0) (ASI = 0x01)
0x000001A0 (DMA1)
0x000001C0 (DMA2)

Figure E5-4. DP/Source/Destination ASI Register

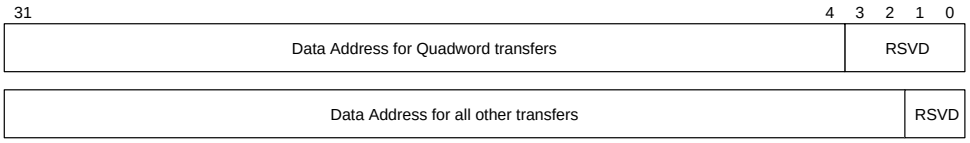
Bits 31-24: Descriptor Pointer ASI (DP ASI)—ASI of the Descriptor Pointer, a register used in buffer-chaining mode. It points to the next element of the linked list whose elements describe the source and destination of the DMA transfer.

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

E5.2.3 Current Source Address Register

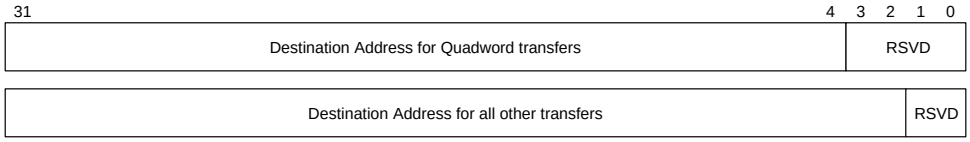


Address:0x00000184 (DMA0) (ASI=0x01)
0x000001A4(DMA1)
0x000001C4 (DMA2)

Figure E5-5. 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.

E5.2.4 Current Destination Address Register



Address:0x00000188 (DMA0)
(ASI=0x01)
0x000001A8 (DMA1)
0x000001C8 (DMA2)

Figure E5-6. 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.

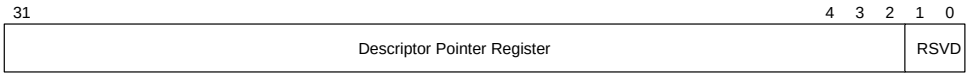
E5.2.5 Current Byte Count Register

Address:0x0000018C (DMA0)
(ASI=0x01)
0x000001AC (DMA1)
0x000001CC (DMA2)

Figure E5-7. 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.

E5.2.6 Descriptor Pointer Register



Address:0x00000190 (DMA0)
(ASI=0x01)
0x000001B0 (DMA1)
0x000001D0 (DMA2)

Figure E5-8. Descriptor Pointer (DP) Register

Used in Chaining Mode, the descriptor pointer points to the first element in the linked-list of chaining descriptors. When using buffer-chaining, there is no need to setup the source address, destination address, or byte count as they are loaded from the first chaining descriptor.

E5.2.7 Channel Control Register

Bits 31:24, 20–18 are reserved, should be written 0's only, and read unknown values. 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.

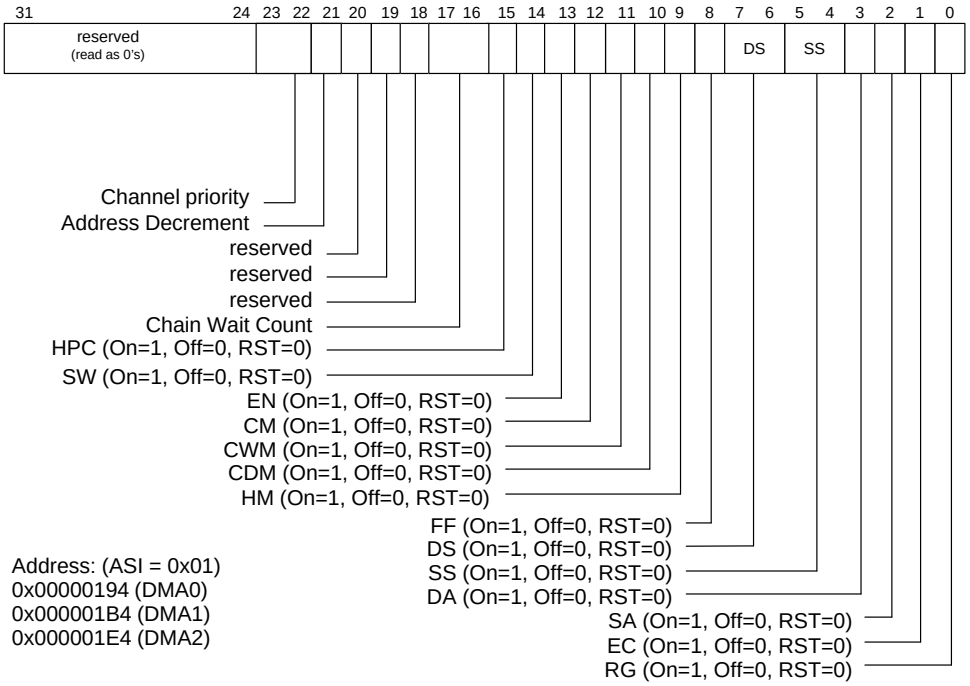


Figure E5-9. Channel Control Register

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 all three channels are disabled.

The channel status can also be accessed through ASR 18 (channel 0) and ASR 19 (channel 1). This allows a program to read and write the DMA status register without entering supervisor mode. The channel 2 Status Register cannot be read as an ASR.

Bits 23-22 Channel Priority.— Sets the priority of this DMA channel. 3 is the highest priority and 0 is the lowest priority. If all channels have priority 0, round robin arbitration is used. Note: either all channels must be priority zero, or each channel must be assigned a unique priority from 1 to 3.

Bit 21 Address Decrement Mode.— Setting this bit to 1 causes the source/destination addresses to be decremented during a DMA transfer. This bit overrides bits 2 and 3 (source/dest. address inc./hold). This is primarily used for DMA to the Video Interface in Video Duplex Mode. Note: this mode is not compatible with quad–word transfers.

Bit 20–18: Reserved. Should be set to 0.

Bits 17-16: Chain Wait Count (CWC)—Used in chain-wait mode to set the number of chaining descriptors that are loaded before entering the chaining-wait state. A value of 0 in this field causes DMA to wait after each chaining descriptor is fetched. A value of 1 causes DMA to wait after every other chaining descriptor is fetched. A value of 2 causes DMA to wait after every three chaining descriptors are fetched. The value 3 is not valid.

EOP is asserted whenever the DMA controller enters the wait state.

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.

E5.2.8 Channel Status Register

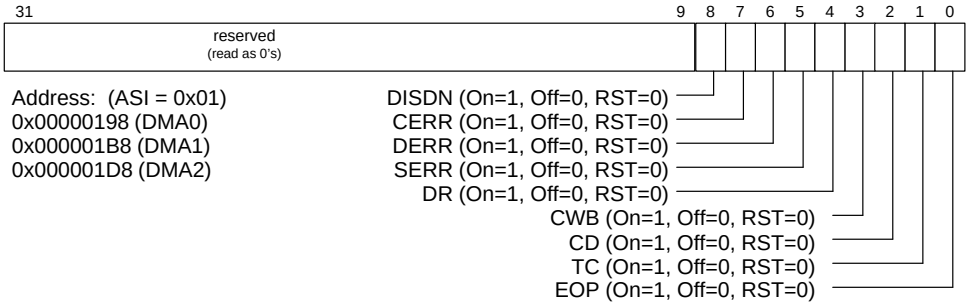


Figure E5-10. Channel Status Register

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.

E5.2.9 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 only the Descriptor-pointer, the ASI register, and the Channel Control register. The values for the address registers and the Current Byte Count Register will be loaded from the chaining descriptor. In DMA chaining mode, the chaining descriptors are loaded before the DMA actually starts. After the channel is enabled, it will perform five read cycles to load the first chaining buffer. Next, the actual DMA will occur. When the DMA completes (transfer count reaches –1), if the most recently loaded descriptor-pointer (DP) is not zero, the next chaining descriptor will be loaded. If the last DP loaded was zero, then that buffer was the last in the chain.

After each chaining-descriptor load and DMA transfer operation, the chain-wait counter decrements. If chain-wait mode is enabled and this counter reaches –1, EOP will be asserted and the DMAC will suspend itself until the chaining-wait bit in the status register is cleared. While the DMAC is suspended, any of the registers can be safely inspected or modified before re-activating the channel.

When Terminal Count (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.

E5.2.10 DMA Channel Arbitration

Whenever a channel reaches Terminal Count (or after every transfer in single transfer mode), the DMA controller re–arbitrates use the DMA controller. This provides fair access to the DMA controller if more than one DMA channel is simultaneously active.

When multiple DMA channels are active and requiring service, there must be a mechanism to decide which DMA channel is allowed to use the DMA controller. This is the job of the DMA arbiter.

The DMA arbiter provides two modes of arbitration: round robin and fixed priority mode.

Round robin mode is activated when all DMA channels have their channel priority bits set to zero and the SWM bit is set (bit 14). If more than one channel is active and

requesting service, then the DMA channels will equally share the DMA controller in a round robin fashion (first channel 0, then channel 1...).

Fixed priority mode is activated if the DMA channels are given different priorities and the SWM bit is set to zero (bit 14). When this mode is used, each channel must have a unique priority from 1 to 3 (no channel should be given the same priority and none should have the priority of zero). In fixed priority arbitration, the DMA controller always services the highest priority active channel (3 is the highest priority, 1 is the lowest).

This arbiter is also backward compatible to earlier SPARClite DMA arbiters: if all of the DMA channel priority bits are set to zero, the SWM and HPC bits select switching/ fixed priority mode and which channel is high–priority (in fixed mode). Note that in this mode, channel 2 cannot be set to the high–priority channel.

E5.2.11 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.

E5.2.12 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.

E5.2.13 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.

E5.2.14 Error Conditions

Memory Access Exceptions:

- Source Transfer Exception
- Destination Transfer Exception
- Chaining Exception

Transfer errors are signalled when –MEXC occurs during a DMA transfer. 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.

E5.3 External Interface

E5.3.1 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 is reached, 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 E5-11. 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 E5-13 below, showing a quadword transfer taking four data cycles. The last DMA transfer is accompanied by EOP.

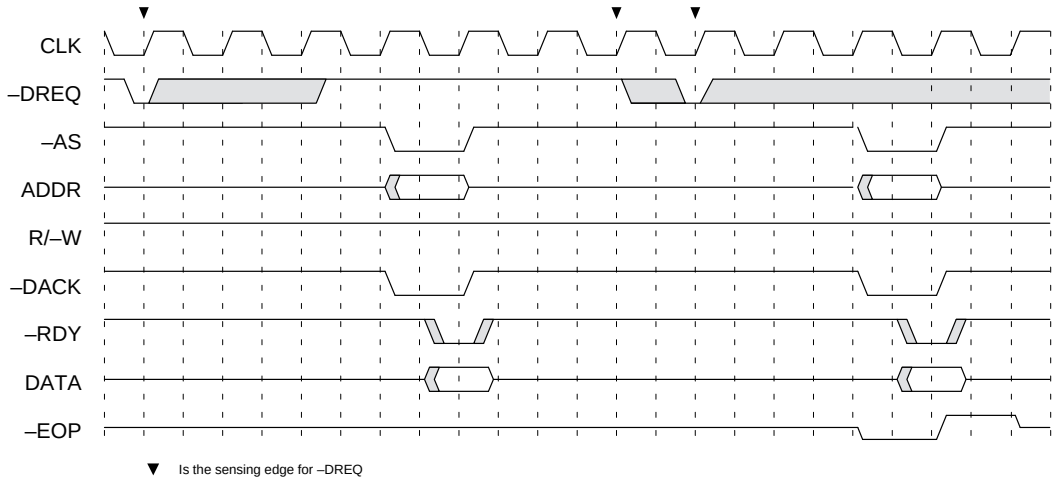


Figure E5-11. Single Transfer, Edge-Sensitive, Flyby (R/–W high)

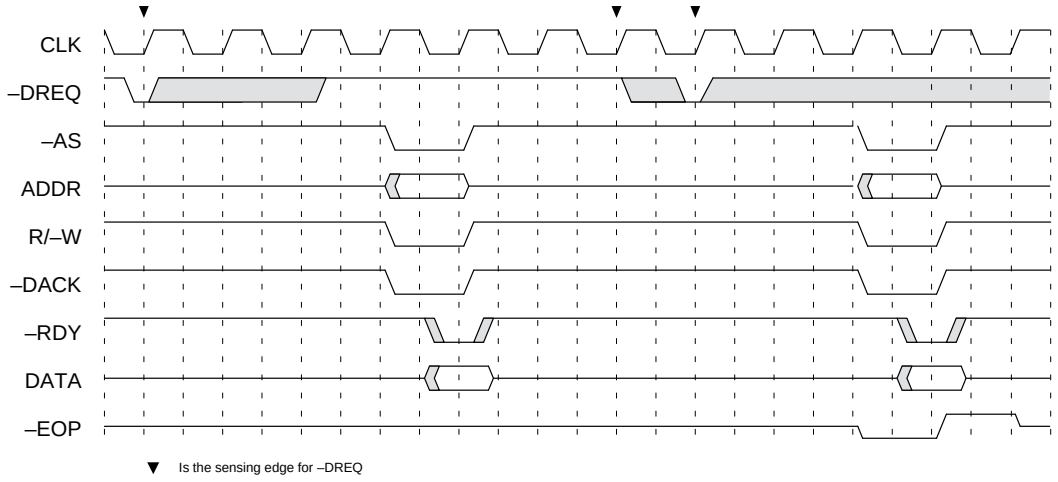


Figure E5-12. Single Transfer, Edge-Sensitive, Flyby (R/–W low)

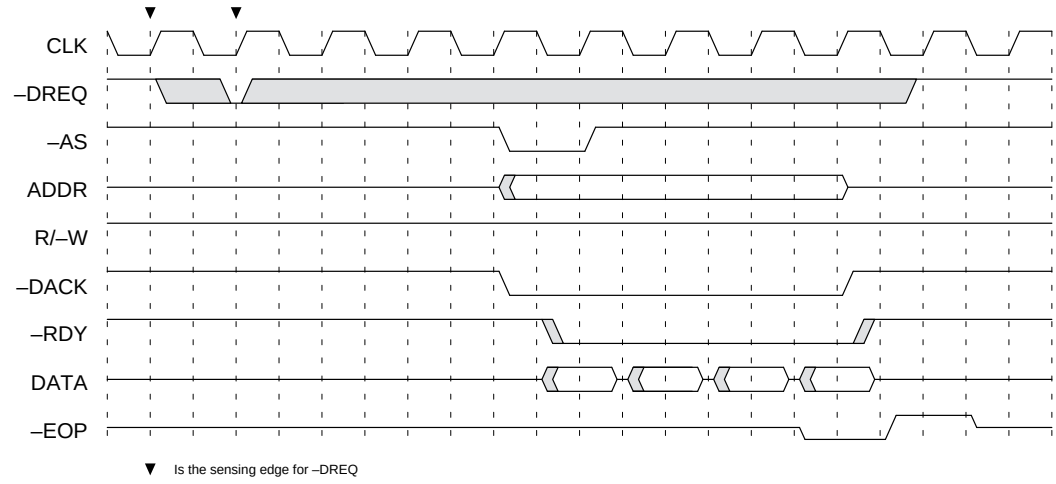


Figure E5-13. Single Transfer, Edge-Sensitive, Flyby, Quadword (R/–W high)

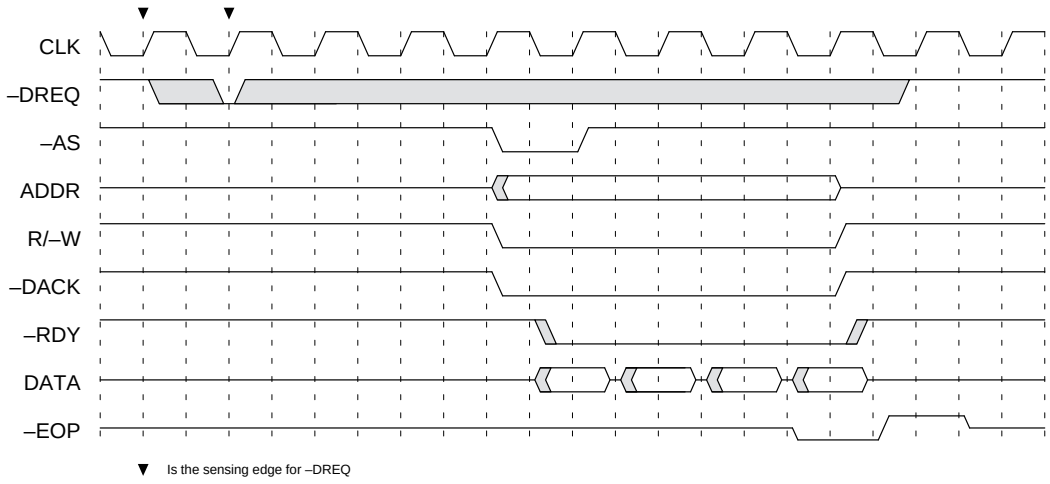


Figure E5-14. Single Transfer, Edge-Sensitive, Flyby, Quadword (R/–W low)

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 E5-15 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.

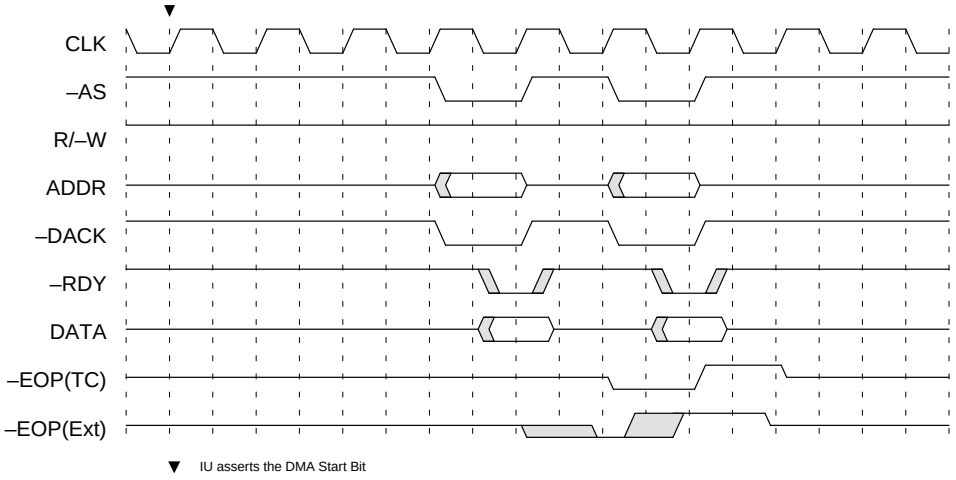


Figure E5-15. Block Transfer, Flyby (R/–W high)

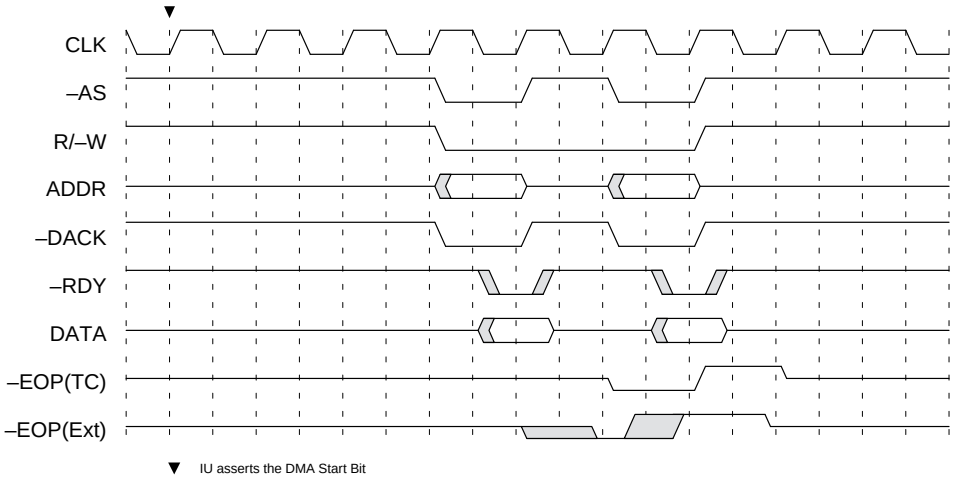


Figure E5-16. Block Transfer, Flyby (R/–W low)

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 E5-17. When a request for a demand transfer is made, the DMAC will look at the –DREQ to see if any request is pending.

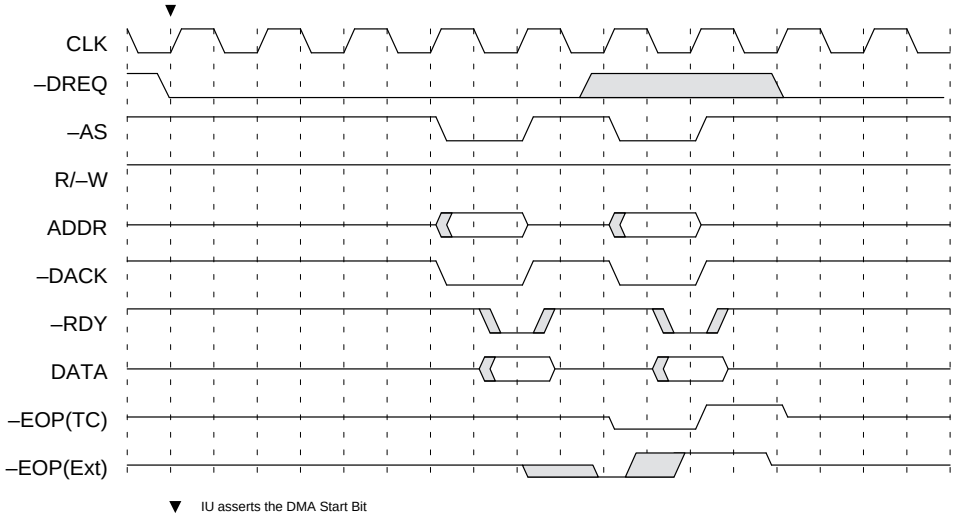


Figure E5-17. Demand Transfer, Flyby (R/–W high)

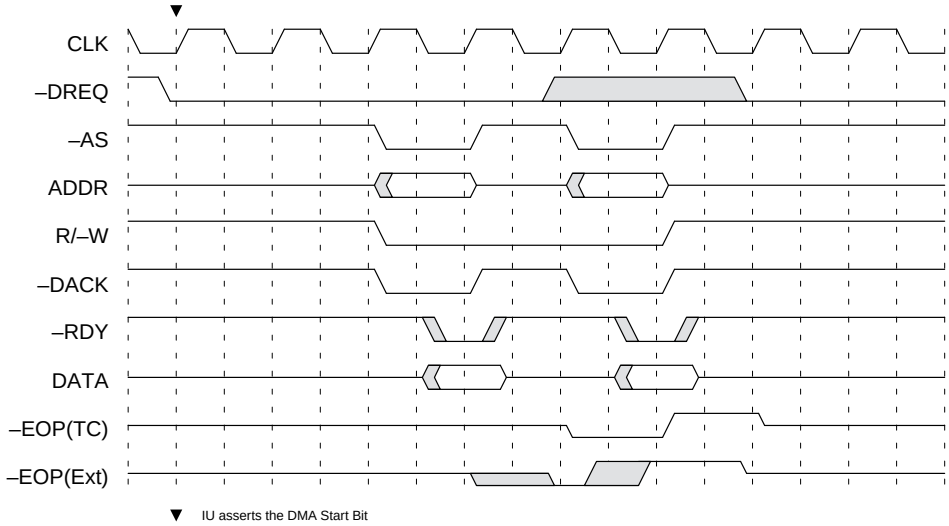


Figure E5-18. Demand Transfer, Flyby (R/–W low)

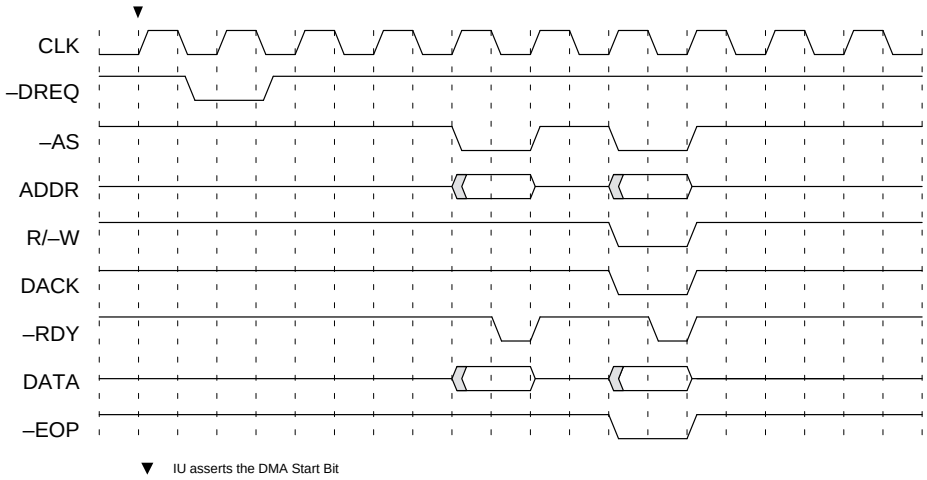


Figure E5-19. Single Transfer, Edge Sensitive Flow Through, Destination Request

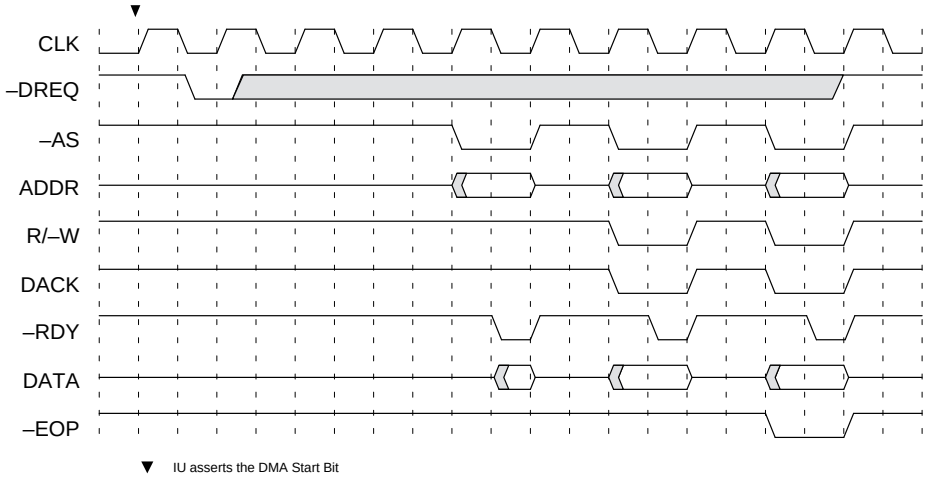


Figure E5-20. Demand Transfer, Flow Through, Word to Half–Word, Destination Request

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. Burst sizes of up to 64 words are allowed in flyby mode.
- *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. Burst sizes of up to 4 words (quad-word) are allowed in flowthru mode.

Source/Destination Size

The source and destination size can be byte, half-word, word, or quad-word. For flyby transfer, the source and destination size must be the same. For flowthru mode, if the source and destination size 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 half-word, word, and quad-word sizes.

To take advantage of the burst transfer supported by the BIU, the DMAC offers quad-word transfers. Quad-word transfers are fastest when both the source and the destination sizes are quad-word, but the DMAC can support any combination of source and destination sizes (from byte through quad-word). All transfers must be address-aligned on their size boundary. For example, if the source size is quad-word and the destination size is word, then the source address must be quad-word aligned and the destination address must be word aligned.

The DMAC provides full packing and unpacking for sources and destinations of differing sizes (in flowthru mode only). The DMAC will never read or write a different size than what is programmed, so some transfers may be padded with unknown data to fill out the transfer size. The DMAC can mix any of the flow-thru sizes (byte through quad-word). The DMAC can also mix any combination of byte-counts for DMA transfers. If the byte count is less than one transfer unit, the DMAC will always transfer one full unit and pad the rest of the data with unknown values. For example, if the DMAC was set up to transfer three bytes from a word size to a byte-size device, the DMAC would read one word and then write three bytes (ignoring the 4th byte, which was read as part of the word). In the other direction, if three bytes were to be read from a byte-wide device and written to a word-wide device, the DMAC would read three bytes and then write one word to the destination device (the 4th byte would contain unknown data). It is up to software to allocate a large-enough destination buffer to hold this extra padding-data.

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 by configuring the Interrupt controller to use the –EOP signal as an interrupt source. When –EOP causes an interrupt, the service routine should read the DMA status register to determine the type of DMA interrupt which occurred.

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

E6

MB86936 Interrupt Request Controller

E6.1 Overview of Interrupt Controller

The Interrupt Controller (IRC) has 4 interrupt input pins for external (off–chip) interrupt sources. The IRC also accepts internal (on–chip) interrupts. The IRC runs at the internal clock frequency for fast interrupt response. It has 15 channels of interrupts with priority 15 having the highest priority and priority 1 having the lowest priority.

The IRC has 3 modes:

Mode 0: compatible with MB86930.

Mode 1: operates similar to that of MB86931; in addition, this mode has 3 groups of internal interrupt channels that have programmable priority.

Mode 2: external interrupts dispersed in priorities with internal interrupts, and has 5 groups of internal interrupt channels that have programmable priority.

The IRC has filter circuits at inputs to increase noise resistance to prevent false interrupts: external interrupt– has to be asserted for 2 or more consecutive external bus cycles to trigger an interrupt. Internal interrupt needs to be asserted for only 1 or more internal clock cycle to trigger an interrupt.

The trigger modes for triggering an interrupts are:

Mode 0: a high–level triggers interrupt.

Mode 1, Mode 2: there are 2 programmable trigger modes for each interrupt channel: high–level and low–level.

E6.2 Memory–mapped Control Registers

The memory–mapped control registers for IRC are described below:

Table E6–1 Memory–mapped Control Register for IRC.

asi	address	register	access
0x1	0x200	TM0 Control Reg	read/write
0x1	0x204	TM1 Control Reg	read/write
0x1	0x208	Request Sense Reg	read/NA
0x1	0x20c	Request Clear Reg	NA/write
0x1	0x210	Mask Reg	read/write
0x1	0x214	IRL Latch/IRL clear	Note (1)
0x1	0x218	IRC mode & program-mable priority Reg	read/write
0x1	0x21c	reserved	———

NA : Not Applicable

note(1): IRL Latch is read only and IRL clear bit is write only.

note: Request Clear Reg is an addressing location only.

In addition to the control registers, the IRC has a memory–mapped test register. The purpose of the test register is to test internal interrupt channels through programming (bypassing the internal interrupt sources). In normal operation, this test register should not be used.

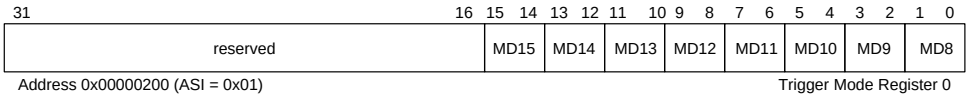
A register related to IRC testing is the Global Test Enable Control (GTEC) register. It needs to be programmed to enable testing of any peripheral. Set bit[0] to enable testing. This register is provided as a double safety feature– its bit[0] must be set and the corresponding peripheral's test enable bit must also be set in order to go into test mode for the peripheral. At reset, this register is cleared.

Table E6–2 Memory–mapped Test Register for IRC.

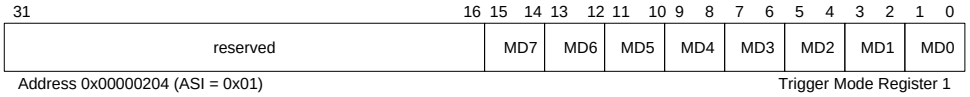
asi	address	register	access
0x1	0x10008	IRC test Reg	read/write
0x1	0x1000c	reserved	——
0x1	0x10000	Global test enable Reg	— /write

E6.2.1 Trigger Mode Control Register

The Trigger Mode registers control the trigger mode for each interrupt channel. Trigger Mode Register 0 controls trigger modes for interrupt channels 8-15; Trigger Mode Register 1 controls trigger modes for interrupt channels 1-7.



Bits 15-0: Trigger Mode Selects - Select trigger modes for channels 8-15.



Bits 15-2: Trigger Mode Selects - Select trigger modes for channels 1-7.

Bits 1-0: Reserved.

Two-bit fields in the registers select one of two trigger modes for each channel as follows:

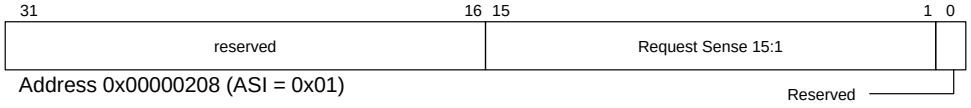
MDx Value*	Trigger Mode
0	High Level
1	Low Level
2	Reserved
3	Reserved

Reset clears the Trigger Mode registers, resulting in high level triggering for each interrupt channel.

Note: An interrupt channel should be masked before its trigger mode is changed, or a false interrupt may occur.

E6.2.2 Request Sense Register

In Mode 1 and Mode 2, after an interrupt passes through the filter circuit, if an interrupt is recognized, the Request Sense register holds the interrupt for prioritizing. In Mode 0 this register is not used. After servicing an interrupt, the register must be cleared to accept new interrupts. At reset, all bits are cleared.



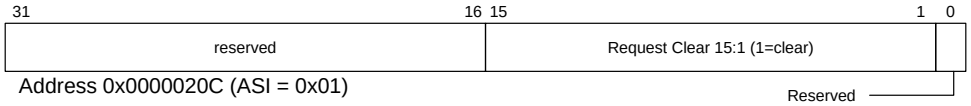
Bits 15-1: Sense IRQ Latch - Correspond to interrupt channels 15-1 and indicate, when high, that the corresponding interrupts are latched and pending.

Bit 0: Reserved.

Reset clears the Request Sense Register.

E6.2.3 Request Clear Register

In Mode 1 and Mode 2, the processor writes to the Request Clear Register to clear the interrupt requests in Request Sense Reg. This register is not used in Mode 0. At reset, all bits are cleared.



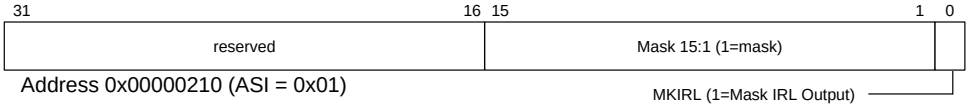
Bits 15-1: Clear IRQ Latch - Correspond to interrupt channels 15-1, and writing the bits to 1 clears the corresponding interrupt latches.

Bit 0 Reserved.

Reset clears the Request Clear Register.

E6.2.4 Mask Register

In Mode 1 and Mode 2, the processor sets the bits [15] to [1] in this register to mask interrupt request to the corresponding channels (if a channel is masked, the interrupt asserted in the channel is excluded from prioritization). Set the bit[0] to mask prioritized interrupts. This register is not used in Mode 0. At reset, all bits are cleared.



Bits 15-1: Interrupt Request Mask - Correspond to interrupt channels 15-1, and writing them to 1 masks the corresponding interrupt requests.

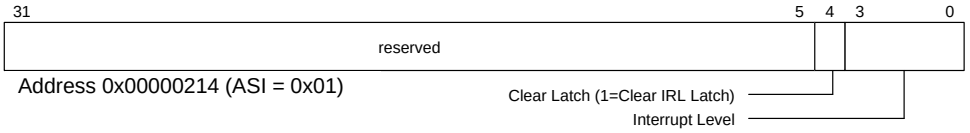
Bit 0: Mask IRL - Masks the output of the IRL Latch. When MKIRL is set to 1, the IRL Latch output is masked, and no interrupt is served. When MKIRL is 0, the encoded interrupt level number in the IRL latch is asserted on the IRL<3:0> bus to interrupt the processor. MKIRL is typically set to 1 (mask enabled) in systems that poll interrupt requests.

Reset clears the Mask register.

E6.2.5 IRL Latch/IRL Clear

In Mode 1 and Mode 2, bits[3:0] hold the prioritized interrupt. After servicing an interrupt, set bit[4] to clear this latch to accept new interrupt. This latch is not used in Mode 0. At reset, all bits are cleared.

The processor uses the IRL Latch/Clear register to clear and read the IRL Latch.



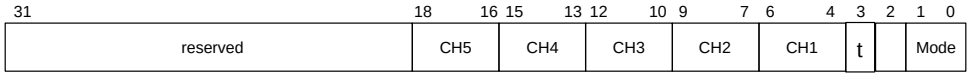
Bit 4: Clear IRL Latch - Clears the IRL Latch when written to 1.

Bits 3-0: Interrupt Level - Holds the value of the IRL Latch. The processor typically reads IRL to identify the highest-priority interrupt level in systems that poll the interrupts.

Reset clears the IRL Latch/Clear Register. The “Clear Latch” bit is only writeable while the interrupt level bits are only readable. Writes do not affect them.

E6.2.6 IRC Mode and Priority Register

The interrupt mode of IRC is selected by programming the IRC Mode and Priority Register. Reset clears all bits in the register and sets the IRC to Mode 0.



Address 0x0000 0218 IRC Mode and Priority Register

Bits 31–19: Reserved

Bits 18–4: Programmable priority bits. Programmable priority is valid only in Mode 1 and Mode 2. Priority Channel 5 has the highest priority, followed by Priority Channel 4, Priority Channel 3 and so on. 3 bits in each priority channel allows the setting of group 1 to group 5 (see section E6–3 for grouping).

On reset, the value of bit [18:4] = 101 100 011 010 001

Bit 3: Test Enable bit. This bit lets the user test the interrupt priority by programming.

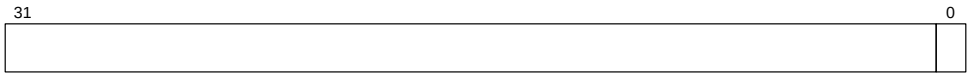
Bit 2: Reserved

Bits 1–0: Mode programming. The user select the IRC interrupt mode by programming the bits.

00	Mode 1 (default)
01	Mode 2
10	Mode 2
11	Reserved. Operation is unpredictable

E6.2.7 Global Test Register

The Global Test Enable Register is programmed to enable peripheral testing.



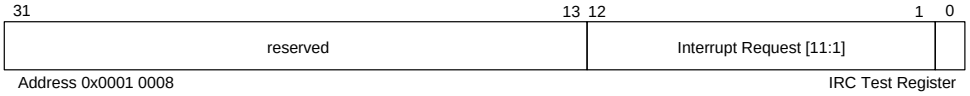
Address 0x0001 0000

Bits 31–1: Reserved

Bit 0: Test enable. This bit is cleared upon request.

E6.2.8 IRC Test Register

The user can simulate/test the interrupt request and priority of IRC by programming. Internal interrupt request (IRQ[11:1]) can be set by programming, absolutely bypassing the corresponding interrupt sources.



Bits 31–12: Reserved

Bits 11–1: Programmable Interrupt Request bits. Every bit position simulates each interrupt request.

Note: This feature exists only in MB86936 and may not be implemented in any future product

E6.3 Interrupt Mode and Priority Programming

The Interrupt Controller (IRC) has 3 modes:

Mode 0– Supports external encoded interrupts IRL[3:0] (as in MB86930). IRL[3:0] are asserted high for interrupt. The external encoded interrupts IRL[3:0] bypass the priority encoder in IRC and go directly to the IU control.

Mode 1– Supports 4 external decoded interrupts IRQ[15:12] and 11 internal decoded interrupts IRQ[11:1] (similar to MB86931). In addition, this mode has 3 groups of internal interrupt channels that have programmable priority request 15 (IRQ[15]) has the highest priority, followed by IRQ[14], IRQ[13] and IRQ[12]. The priority of each group is programmed IRC Mode and Priority Register (ASI=0x1, ADR=0x218). Each group has 2 interrupt channels; the priority of channels within the group is fixed with higher priority given to higher interrupt request. In this mode all interrupts go through the priority encoder in IRC. The interrupt request with the highest priority is encoded and sent to the IU control.

Mode 2– Supports both 4 external decoded interrupts and 11 internal decoded interrupts. The external interrupt channels are dispersed in priority with 5 groups of interrupt channels that are programmable. Each group of internal interrupts that is programmable consists of 2 channels; the priority of the channels within a group cannot be changed.

All interrupts go through the priority encoder in IRC, and the encoded interrupts go to the IU control.

Table E6–3 and Table E6–4 describe the grouping of internal interrupt channels and programmable priority assignment of external and internal channels for both Mode 1 and Mode 2. Group 4 and 5 are available only in mode 2.

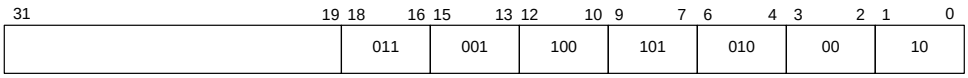
Table E6–3 Group Assignment in Mode 1 and Mode 2.

Group	Interrupt Channel
1	3 and 2
2	6 and 5
3	9 and 8
4	7 and 4
5	11 and 10

Table E6–4 Interrupt Request Priority in Mode 1 and Mode 2.

Priority	Mode 1	Mode 2
15	IRQ [15]/IRL[3] pin	Higher interrupt channel in group referred by Priority Channel 5
14	IRQ [14]/IRL[2] pin	Lower interrupt channel in group referred by Priority Channel 5
13	IRQ [13]/IRL[1] pin	IRQ[15]/IRL[3] pin
12	IRQ [12]/IRL[0] pin	Higher interrupt channel in group referred by Priority Channel 4
11	N/A	Lower interrupt channel in group referred by Priority Channel 4
10	N/A	IRQ[14]/IRL[2] pin
9	Higher Interrupt Request Priority Channel 3	Higher interrupt channel in group referred by Priority Channel 3
8	Lower Interrupt Request Priority Channel 3	Lower interrupt channel in group referred by Priority Channel 3
7	N/A	IRQ[13]/IRL[1] pin
6	Higher Interrupt Request Priority Channel 2	Higher interrupt channel in group referred by Priority Channel 2
5	Lower Interrupt Request Priority Channel 2	Lower interrupt channel in group referred by Priority Channel 2
4	N/A	IRQ[12]/IRL[0] pin
3	Higher Interrupt Request Priority Channel 1	Higher interrupt channel in group referred by Priority Channel 1
2	Lower Interrupt Request Priority Channel 1	Lower interrupt channel in group referred by Priority Channel 1
1	N/A	N/A

Below is an example of how to program the IRC Mode and Priority Register (ASI=0x01, ADR=0x218)



Address 0x0218 IRC Mode and Priority Register

Priority Channel 5 is assigned to interrupt request group 3 (interrupt request channel 9 and 8), followed by an external interrupt pin IRQ[15]/IRL[3]. Priority Channel 4 is assigned to group 1 (interrupt request channel 2 and 3) followed by an external interrupt pin IRQ[14]/IRL[2], and so on. Bit [3] disabled IRC test. Bit [2] is reserved and should be written as 0. Bit [1:0] are programmed for mode 2.

The above program results in the following priority in mode 2 interrupt priority sequence.

Priority	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Interrupt Channel	9	8	15	3	2	14	7	4	13	11	10	12	6	5	2

E6.4 Usage of Interrupt Controller in MB86936

In MB86936, several internal interrupt sources are connected to the IRC. The table below shows the interrupt channel assignment for them.

Table E6–5

interrupt channel	interrupt sources
2	DMA2– DMA generated eop2 ORed with external input pin –EOP2
3	Timer1– timer 1 time out
5	Timer0– timer 0 time out
6	DMA0– DMA generated eop0 ORed with external input pin –EOP0
8	DMA1– DMA generated eop1 ORed with external input pin –EOP1
9	Video Controller interrupt (see Video section)

This configuration allows the users to use –EOP0, –EOP1, –EOP2 as extra external interrupt pin. Since interrupt request channel 2 is connected to internal–EOP source as well as external –EOP2 pin, when DMA channel 2 is disabled, the user can take advantage of –EOP2 as interrupt request channel 2. The same happens to DMA channel 1 (–EOP1 = IRQ[8]) and DMA channel 0 (–EOP0 = IRQ[6]). This effectively gives the user three extra interrupt request channels with programmable priority.

E6.5 IRC Operation

The interrupt controller operation in each of the 3 modes are described below:

E6.5.1 Mode 0

When an external interrupt is asserted high for 2 or more external bus cycles, the interrupt is recognized. The interrupt goes directly to the IU.

E6.5.2 Mode 1 and 2

When an external interrupt is asserted for 2 or more external bus cycles or an internal interrupt is asserted for 1 or more internal clock cycle, the interrupt is recognized, and the interrupts are latched in the Request Sense Register. The interrupts which are not masked by the Mask Register are prioritized, and the prioritized interrupt vector is latched in the IRL Latch. If the IRL Latch is not masked by the IRL mask bit (bit 0 in Mask Reg), then the interrupt vector is sent to the IU for interrupt.

After servicing an interrupt, the IU must clear the corresponding interrupt bit in the Request Sense Register and clear the IRL Latch. Then the new prioritized interrupt vector is generated from the remaining or new interrupts latched in the Request Sense Register.

E6.5.3 Polling

The processor can poll interrupts by reading either the IRQ Latch via the Request Sense register, or the IRL Latch via the IRL Latch/Clear register.

The processor may mask interrupts that it polls via the Request Sense register by masking either the IRQ Latch or the IRL Latch. The processor then periodically reads the IRQ Latch and clears interrupts from the latch when they are serviced. The IRL Latch may remain unmasked to allow interrupt-driven servicing of some interrupts if the polled interrupts are masked with the IRQ Latch mask.

The processor may mask all interrupts when it polls interrupts via the IRL Latch/Clear register by masking the IRL Latch. The processor then periodically reads the IRL Latch for the highest-level pending interrupt and clears both the IRL Latch and the interrupt from the IRQ Latch once the interrupt is serviced.

1) Request Sense Register polling

Set the bits in the Mask Register to mask interrupts. Then the IU polls the interrupts recognized in the Request Sense Register at predefined intervals under software control and services the interrupt.

2) IRL Latch polling

Set the IRL mask bit to mask the prioritized interrupt vector. Then the IU polls the IRL Latch at predefined intervals under software control to service the interrupt.

E6.5.4 Initialization

All IRc Registers are cleared to logic value 0 by –RESET. This results in high–level trigger mode for all interrupts under Mode 0 (decoded IRL[3:0] mode), and all masks disabled.

After reset, the interrupt trigger mode should be changed after the interrupts are masked with the IRQ mask to eliminate false interrupts. The masks can then be disabled.

E6.6 IRC Control Programming Considerations

To prevent false interrupts when changing trigger modes, the following steps should be taken in programming the trigger mode register.

1) Mask the interrupt bit whose trigger mode is to be changed.

2) Change the trigger mode by writing to the Trigger Mode Control Register.

3) Clear the corresponding interrupt bit Sense Register.

4) Clear the IRL latch.

5) Remove the mask bit.

The following example shows an assembly language program on programming the IRC Trigger Mode control register. The program changes the interrupt channel 15 trigger mode to low–level trigger mode, leaving the rest of the interrupt channel trigger modes to high–level.

```
   ....
   ....
! define control register ASI address space
#define CASI   0x1
! define Interrupt Controller registers
!
! Trigger Mode control registers
#define TM0    0x200
#define TM1    0x204
! Request Sense reg.
#define RQS    0x208
```

```
! Request Clear reg.
#define RQC    0x20c
! Request Mask reg.
#define RQM    0x210
!IRL latch
#define IRLAT  0x214
! set channel 15 interrupt to low–level interrupt trigger,
! the rest of the channels are high–level interrupt triggers
#define tm15L  0x4000
!define mask for channel 15
#define mask15 0x8000
! define clear for channel 15
#define clr15  0x8000
! define mask reset for all channels
#define mskrst 0x0
! define IRL latch clear
#define clrirl 0x10
   ....
   ....
! the following program segment changes the trigger mode for
! interrupt channel 15
! set mask for channel 15
    set   RQM, %l2
    set   mask15, %l3
    sta   %l3, [%l2] CASI
! change trigger mode for channel 15
    set   TM0, %l2
    set   tm15L, %l3
    sta   %l3, [%l2] CASI
! clear Request Sense reg.
    set   RQC, %l2
    set   clr15, %l3
    sta   %l3, [%l2] CASI
! clear IRL latch
    set   IRLAT, %l2
    set   clrirl, %l3
    sta   %l3, [%l2] CASI
! clear the mask bit
    set   RQM, %l2
    set   mskrst, %l3
    sta   %l3, [%l2] CASI
   ....
   ....
```

CHAPTER

E7

MB86936 Video Interface

E7.1 Overview

The video Interface provides direct connection to a number of laser-beam marking engines. It may also be used to receive data from a raster input device such as a scanner or to serialize/deserialize a data stream.

General functions:

- Internal or external VCLK with programmable polarity and programmable 4-bit clock division.
- Suspend the operation when VCLK is inactive (for external VCLK only).
- Internal or external PSYNC with programmable polarity.
- External LSYNC with programmable polarity.
- Either DMA (channel 0) or interrupt request on transmit FIFO threshold or receive buffer full.
- Programmable Line-Width, Image-Address, Block-Height, Top-Margin and Left-Margin.
- Programmable interrupt sources.

- During DMA transfers, video transfer progress may be monitored by reading the DMA channel's registers.

Transmit functions:

- 8-word deep 32-bit wide FIFO with programmable threshold.
- Programmable blank level and width (1, 4 or 8) for VDAT<7:0>. When 8-bit video is enabled, ASI<3:0> pins will be used for VDAT<7:4>.
- Reverse mode transmission.
- Programmable word or quad word DMA to load the FIFO.

Receive functions:

- 1-word 32-bit wide holding buffer.
- Status to indicate receive buffer full (data available).

E7.1.1 Video Interface Overview

The serial data transfer timing is controlled through the PSYNC, LSYNC and VCLK pins. Data transfers to/from main memory and the video interface can be handled either through DMA or interrupt driven means. The active level of PSYNC and LSYNC is programmable, as is the active edge of VCLK (rising or falling). The VDAT levels are also programmable: inverted or non-inverted data, and 0 or 1 for the Blank-Data level.

Both PSYNC and VCLK can be either internally or externally generated signals. The default is externally generated signals. If VCLK is internally generated, it will be a free-running divided-down version of the internal clock. If VCLK is externally generated, VCLK will be divided down internally by a factor of 1 to 16. External VCLK may be stopped at any time to suspend the video interface's operation.

E7.2 Video Transmit

The page cycle begins when PSYNC becomes active (either as an input or an output). At the beginning of a page cycle, four count-down registers are loaded from the top margin, left margin, block height and line width registers. When the top margin counter reaches zero the first line will be transmitted. Blanks will be transmitted until the left margin is zero. The actual video data is transmitted until the line width is zero. The values of the previously mentioned registers never change – only internal copies of these registers change. This allows for easily restarting the video operation – the line and margin registers do not need to be written if the page layout is the same.

After the video-interface has been initialized, it waits for PSYNC to become active to indicate the beginning of a page. If PSYNC is a level signal, PSYNC is normally held active until the end of the page.

LSYNC being asserted indicates the beginning of a line. Top Margin number of lines (LSYNCs) are skipped before sending any data. During this time, Blank-level data is present on the VDAT pins. After LSYNC is asserted, LeftMargin bits (VCLKs) are skipped before sending the serial data. During that time, Blank-Level data is present on the VDAT lines. After the left margin is passed, data is shifted out until the end of the printed line (LineWidth VCLK's). Blank-level data is sent out on the VDAT lines until LSYNC is de-asserted and re-asserted again (next line).

The data to be transmitted does not need to be word-aligned in memory, but must be at least byte-aligned. The StartBit register is used to start the shifting at one of 4 bit positions in the first word of each line.

The video interface also has a blank-page feature which forces the VDAT output to the blank level for the entire page. To print a blank page, set the blank-page bit in the video control register to 1, and set up a normal video transmit (DMA may also be initialized, but the video interface will ignore the data). This feature is normally used in interrupt-driven transfer mode. No data needs to be written to the video transmit register (FIFO).

E7.2.1 Video Transmit FIFO Control

The Video Interface contains an 8–word, 32–bit wide FIFO to buffer video data during video transmit operations. The FIFO can be filled either by using the IU to write data to the Transmit FIFO register, or by setting up a DMA transfer to fill the FIFO. The video interface must be enabled in order for the FIFO to accept data.

The status of the FIFO (empty, half–empty or full) can be determined by reading the video status register's FIFO–empty (threshold reached) bit. The meaning of the FIFO empty/half–empty/full bit is controlled by the video control register's FIFO–threshold bit. If the FIFO–threshold bit is set to 0, the FIFO threshold is "completely empty" which means that the status bit indicates FIFO–empty only when the FIFO is completely empty. If the FIFO–threshold control bit is set to 1, then the status register's FIFO–empty bit get set when the FIFO has 4 or fewer words left in the FIFO.

E7.2.2 Data Flow during Video Transmit

There are three possible methods for monitoring and controlling the flow of data into the video interface during a video data transmit operation: polling, interrupt driven and DMA.

In polled or programmed mode, the control program monitors the video FIFO empty (threshold reached) bit in the status register. When this bit indicates that there is space available in the FIFO, the program writes video data into the Transmit FIFO register port.

In interrupt mode, the control program receives a video interrupt whenever the transmit FIFO is half–empty or completely–empty (programmable by user). To enable interrupt mode, set the ”IRQ on FIFO empty” bit in the video control register. With this bit set, an interrupt will occur whenever the FIFO needs more data.

In DMA mode, the user sets up a DMA operation to fill the transmit FIFO whenever it needs data. To set up a DMA to video operation, perform the following steps:

1. disable the Video Interface and DMA channel 0.

2. Set up DMA channel 0 in Flyby, single transfer mode, destination–request, external DMA Request, and either word or quadword transfer size. If printing in Duplex (reverse) mode, set the DMA address mode to ”address decrement” mode and use only word–sized transfers.

3. Set up all of the Video registers (margins, start bit ...) except video control.

4. Clear the video status register, set video control register 2, and last, set video control register 1 to enabled, dma–enable and other options as applicable. This enables video and starts the transmit operation.

Note: DMA to/from the video interface can only occur on DMA channel 0.

E7.2.3 Duplex Mode — Reverse Data Transmission

The video interface has the ability to transmit data in either forward or reverse bit order. This feature is useful for two–sided printing (duplex). Normally Duplex mode is used for printing the reverse side of the page.

In normal mode, the video–data bits are shifted out with the most significant bit first (normally bit 31). In duplex mode, the video data is shifted out starting with the least significant bit (normally bit 0). In duplex mode, the user should send words to the Transmit FIFO in reverse order (reverse with respect to a normal page). If DMA is used, then the ’Address Decrement Mode’ of DMA should be used to copy words to the video FIFO in reverse order.

E7.2.4 Aborting and Restarting Video Transmit

If a page which is being printed needs be aborted and restarted (as in the case of a paper-jam), the re-print operation can be started by writing just a few registers.

All of the video registers retain their values until they are changed by the user (or by reset), therefore, to restart a page, only the video control register will need to be written (to re-enable the video interface). However, if DMA is used, then the DMA registers will need to be re-programmed (DMA registers do not hold their value). If a chained DMA operation was used for example, this would simply require programming the DMA descriptor-pointer and the DMA control register. It is the responsibility of the software to retain the values for reprogramming the DMA registers and the video control register.

E7.3 Video Receive

PSYNC must be asserted in order to receive data. When LSYNC is active, the line count determines the number of bits to receive on VDAT. For high performance, the received data is buffered in a 32-bit receiving buffer. Either an interrupt or DMA is requested when the holding register is full.

The video receive operation is similar to the transmit operation, except TopMargin is not used. Serial data is collected in to a de-serializing buffer as long as PSYNC is asserted. Each line begins with LSYNC and continues for LineWidth VCLK's. When the serial data buffer becomes full (32 bits of data have been collected), the data is copied to the Receive Buffer and the receive-buffer-full flag is set in the video status register.

If PSYNC is configured to be an output, then PSYNC is used to indicate that the Video Interface is ready to accept data. PSYNC will be asserted 4 VCLK's after the video interface is enabled, and remain active until BlockHeight lines have been read in. Duplex mode and the Start-bit register have no effect in receive mode. The direction is always assumed to be forward.

E7.4 Video Interrupts

The video interface is capable of generating interrupts for a number of different conditions. The user can select which conditions can cause an interrupt through use of the enable-mask bits in the video control register. Detailed descriptions of the interrupt

sources can be found in the Video Control register description. The potential interrupt sources are:

- PSYNC de-asserted (only available for level PSYNC)
- End of Page
- Transmit FIFO empty/half–empty.
- Receive buffer full.

Note: In order to receive interrupts after each chain block completes, the DMA channel 0 should be set into chaining test mode. (DMA Control Register bit 10).

E7.5 Video Registers

LineWidth

The width of the printed image in VCLK's. When Line-Width is decremented down to zero, VDAT outputs blank-level from then to the end of the line. If any data remains in the shift-register when LineWidth reaches zero, that data is ignored. The LineWidth must be at least 1.

BlockHeight

The number of lines in the printed image. This register is decremented for each LSYNC (after the top margin is skipped). When BlockHeight reaches zero, VDAT outputs blank-level data from that point to the end of the page. The BlockHeight must be at least 1.

TopMargin

The number of lines to skip from the top of the page to the first line of the printed image. This value is decremented for every LSYNC received.

LeftMargin

The number of dots/VCLK's to skip from the edge of the page to the beginning of the printed image. The LeftMargin must be at least 1 for receiving and 2 for transmitting. There must be at least 2 VCLK's after the video data for the line is transmitted (at least a 2 VCLK right margin)

StartBit

This register defines which bit to start with when printing a line. The shift register is 32 bits wide, and this register specifies which bit will be the first bit shifted out. This start-bit applies to the beginning of each line. The valid settings are: in normal mode: 31, 23, 15, 7. in duplex mode: 0, 8, 16, 24.

Example (not duplex mode, 1 bit VDAT):

value = 31 means shift out all of the data in the shift register starting from bit 31 and continuing through bit 0.

value = 23 means start shifting from bit 23, through bit 0.

In duplex (reverse) mode:

value = 0 means shift out all of the data in the shift register starting from bit 0 and continuing through bit 31.

value = 8 means start shifting from bit 8, through bit 31.

Transmit FIFO register

Read data from or write data to the transmit FIFO directly (without DMA). This register can be used to access the 8 word deep video transmit FIFO. Normally, the CPU would write 32-bit words to this buffer if DMA was not being used to send data to the print-engine. Data is read out from the FIFO in the same order in which it was written. There is no protection against overrun (reading too much data) if the user is using the IU to read from the FIFO.

Note: Reading from this register is only allowed while the video interface is disabled. Otherwise, the video interface and CPU would be competing for the data from the FIFO and the results would be unpredictable. Reading from the FIFO is primarily used for debugging purposes.

ReceiveBuffer register

Read data from or write data to the receive buffer directly (without DMA). The receive buffer is 1 word deep (32 bits). When the shift register receives 32 bits of data, the data is transferred into this register to be read by the CPU. The status flag 'Receive Buffer Full' indicates that there is data to be read from this register.

Video Control Register 1

Bits: 15–12 VCLK Divisor – This value is the amount to divide–down the internal VCLK from the source clock (either the internal IU clock or the VCLK pin). The valid range is 1–16 (setting to 0 means divide by 1; setting this to 1 means divide by 2, ...)

Bit: 11 VCLK Drive Out – Setting this bit to 1 causes the internal video interface's VCLK to be driven onto the VCLK pin. Normally bits 10 and 11 would never be set to 1 simultaneously.

Bit: 10 VCLK Source External – This bit indicates whether the source for VCLK generation is the VCLK pin or the internal IU clock. If this bit is 1, the VCLK pin is used. Note that if the internal IU clock is used, VCLK must be divided down by 2 or more (see bits 15:12).

Bit: 9 VCLK Invert – Setting this bit causes the Video Interface's VCLK to be inverted with respect to the VCLK pin.

Bit: 8 RESERVED

Bit: 7 IRQ on Receive – Setting this bit causes the video interface to generate an interrupt when a word has been received by the video interface.

Bit: 6 IRQ on FIFO Empty – Setting this bit causes the video interface to generate an interrupt when the FIFO is either half–empty or completely empty (based on the value of the 'FIFO–threshold' bit in video control register 2).

Bit: 5 IRQ on End Of Page – Setting this bit causes the video interface to generate an interrupt when the last line of the page has been printed.

Bit: 4 IRQ on PSYNC Negated – Setting this bit causes the video interface to generate an interrupt when the PSYNC pin changes from asserted to de–asserted.

Bit: 3 DMA Enable – Enables DMA to/from the video interface. Must be set for DMA–video transmit/receive operations.

Bit: 2 Duplex Mode – When set to 1, video data is transmitted in reverse bit order. This is intended for reverse/duplex printing operations. This bit should be cleared during video–receive operations.

Bit: 1 VDAT Output – Direction of Video Data. If this bit is 1, video data moves out from the video interface (transmit); otherwise, VDAT is an input.

Bit: 0 Video Enable – When this bit is 1, the video interface is enabled for transmitting and receiving data. Setting this bit to 0 disables the video interface, disables interrupts and clears part of the status register

Video Control Register 2

Bits: 15–11 RESERVED

Bit: 10 Blank Page – Setting this bit to 1 causes all VDAT (video data) output to be set to the blank level. Useful for printing a blank page or blank side of a page in duplex mode. This has no effect in receive mode.

Bit: 9 Blank Level – Determines the VDAT (video data) value which represents a blank. This is the data value used for margins and blank pages.

Bit: 8 FIFO Threshold Half – This bit determines the meaning of the video status register's FIFO threshold reached bit. If this bit is 1, then the status register indicates when the FIFO has fewer than 4 words left in the transmit FIFO (half empty). If this bit is 0, then the status register bit indicates when the FIFO is completely empty (currently transmitting the last word—the user program may write up to 8 words of data).

Bits: 7–6 Video Data Width – Sets the width of the video data. 00 = 1 bit data, 01 = 4 bit data, 10 = 8 bit data, 11 = reserved.

Bit: 5 VDAT Invert – If this bit is set, all video data is inverted before being sent or received.

Bit: 4 LSYNC Pulse mode – Selects pulse or level mode for the LSYNC input pin. If this bit is 1, Pulse mode is selected, otherwise level mode is selected.

Bit: 3 LSYNC Invert – Selects whether the LSYNC pin is active high or active low. If this bit is 0, LSYNC is treated as an active high input; if this bit is 1, LSYNC is treated as an active low input.

Bit: 2 PSYNC Pulse mode – Selects pulse or level mode for the PSYNC input pin. If this bit is 1, Pulse mode is selected, otherwise level mode is selected.

Bit: 1 PSYNC Invert – Selects whether the PSYNC pin is active high or active low. If this bit is 0, PSYNC is treated as an active high input; if this bit is 1, PSYNC is treated as an active low input.

Bit: 0 PSYNC Output – Selects whether the PSYNC pin is an input or an output. If this is set to 0, PSYNC is an input; otherwise PSYNC is an output. Note: PSYNC output is always level mode.

Video Status Register

Bit: 7 End Of Page – This bit indicates that the end of the transmitted page has been reached. Specifically, all lines of video data have been transmitted. This bit can be cleared by writing a 0 to the video status register.

Bit: 6 PSYNC Status – This bit is 1 when PSYNC is active, 0 otherwise.

Bit: 5 LSYNC Status – This bit is 1 when LSYNC is active, 0 otherwise.

Bit: 4 Video Interrupt Request Active – This bit indicates that the video interface is currently generating an interrupt.

Bit: 3 DMA Request Active – This bit indicates that the video interface is currently requesting a DMA transfer.

Bit: 2 Receive Buffer Full – This bit indicates that the receive buffer contains a valid word of data. This bit is automatically cleared when the data is read.

Bit: 1 FIFO Threshold Reached – This bit indicates that the FIFO is either half empty or completely empty (transmitting the last word).

Bit: 0 Page Active – This bit indicates that a page of data is currently printing. This bit is 1 while the video interface is enabled and the line and page counters have not yet reached zero.

Note: To clear ‘End of Page’ and ‘DMA Int’, ‘0’ should be written to this register. Writing ‘0’ will not affect the other bits in this register.

E7.6 Video Signal Timing

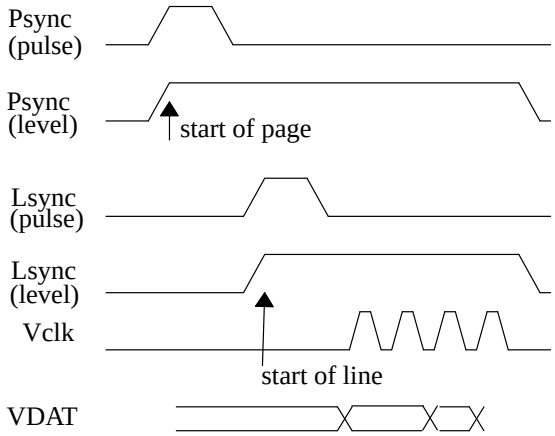
PSYNC and LSYNC are asynchronous inputs. PSYNC and LSYNC can be programmed to be treated as pulse-mode signals or level-mode signals. They can also be programmed to be active-high or active-low. Note: since these signals are asynchronous, they must be low noise signals.

In pulse mode, the positive edge of the signal is used as the active edge (unless the sync invert bit is set, in which case the negative signal edge is used). In level mode, LSYNC must be held active for the entire line, and PSYNC must be held active for the entire page (except in the case of video transmit abort and restart mode). PSYNC as an output is only available as a level signal.

External VCLK may be stopped (held low) at any time effectively suspending the video interface. VCLK must run at a lower frequency than the IU frequency. If VCLK is generated by the internal clock, the VCLK divisor must be set to at least 2.

LSYNC and PSYNC (as inputs) must provide at least 10nS setup and hold time with respect to the active edge of VCLK.

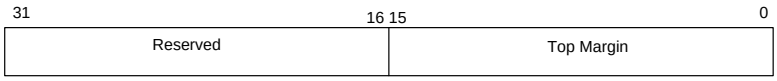
Video Sync Signals



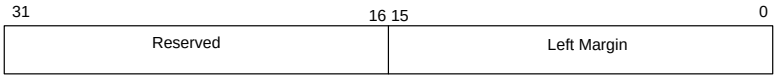
Note: Active high signals shown. Active polarity of the sync signals is programmable.

TABLE E7–3.Video Control Registers

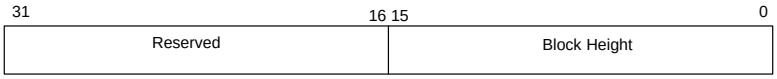
Top Margin	
ASI	**ADDRESS**
0x 1	0x 0000 0280



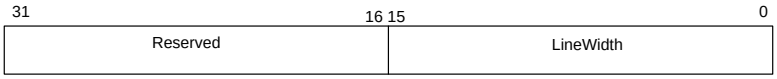
Left Margin	
ASI	**ADDRESS**
0x 1	0x 0000 0284



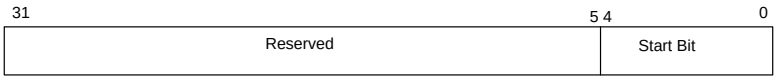
Block Height	
ASI	**ADDRESS**
0x 1	0x 0000 0288



Line Width	
ASI	**ADDRESS**
0x 1	0x 0000 028C



Start Bit	
ASI	**ADDRESS**
0x 1	0x 0000 0290



Video Control 1	
ASI	**ADDRESS**
0x 1	0x0000 0294

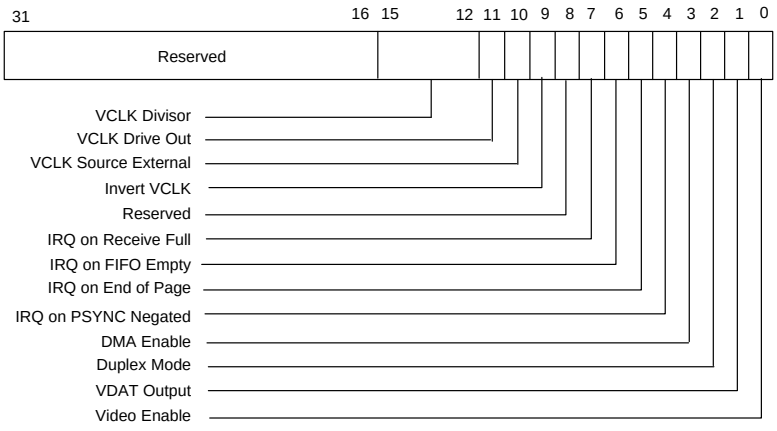
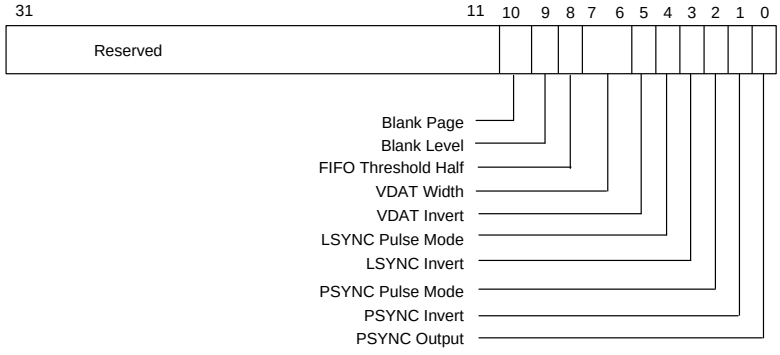
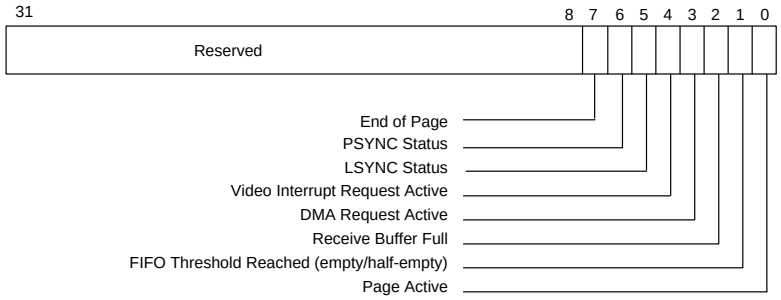


TABLE E7–3. MB86936 Video Control Registers (continued)

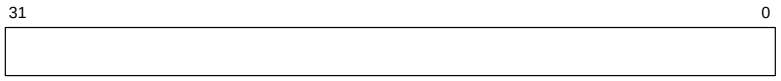
Video Control 2	
ASI	**ADDRESS**
0x 1	0x0000 0298



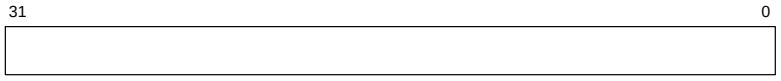
Video Status	
ASI	**ADDRESS**
0x 1	0x0000 029C

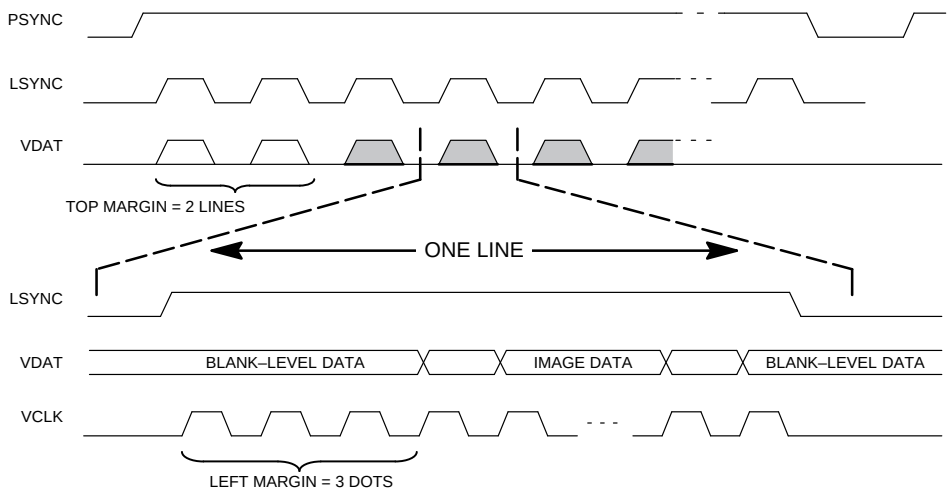


Transmit FIFO	
ASI	**ADDRESS**
0x 1	0x0000 02A0

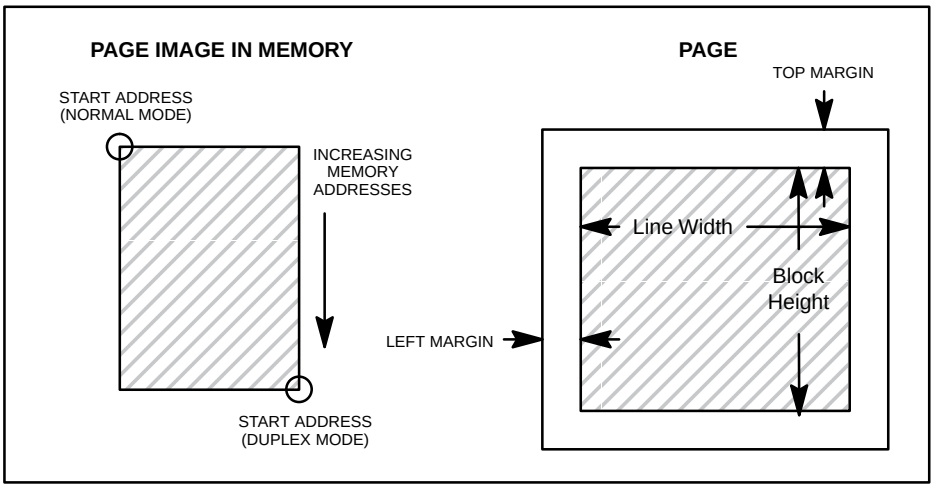


Receive Buffer	
ASI	**ADDRESS**
0x 1	0x0000 02A4

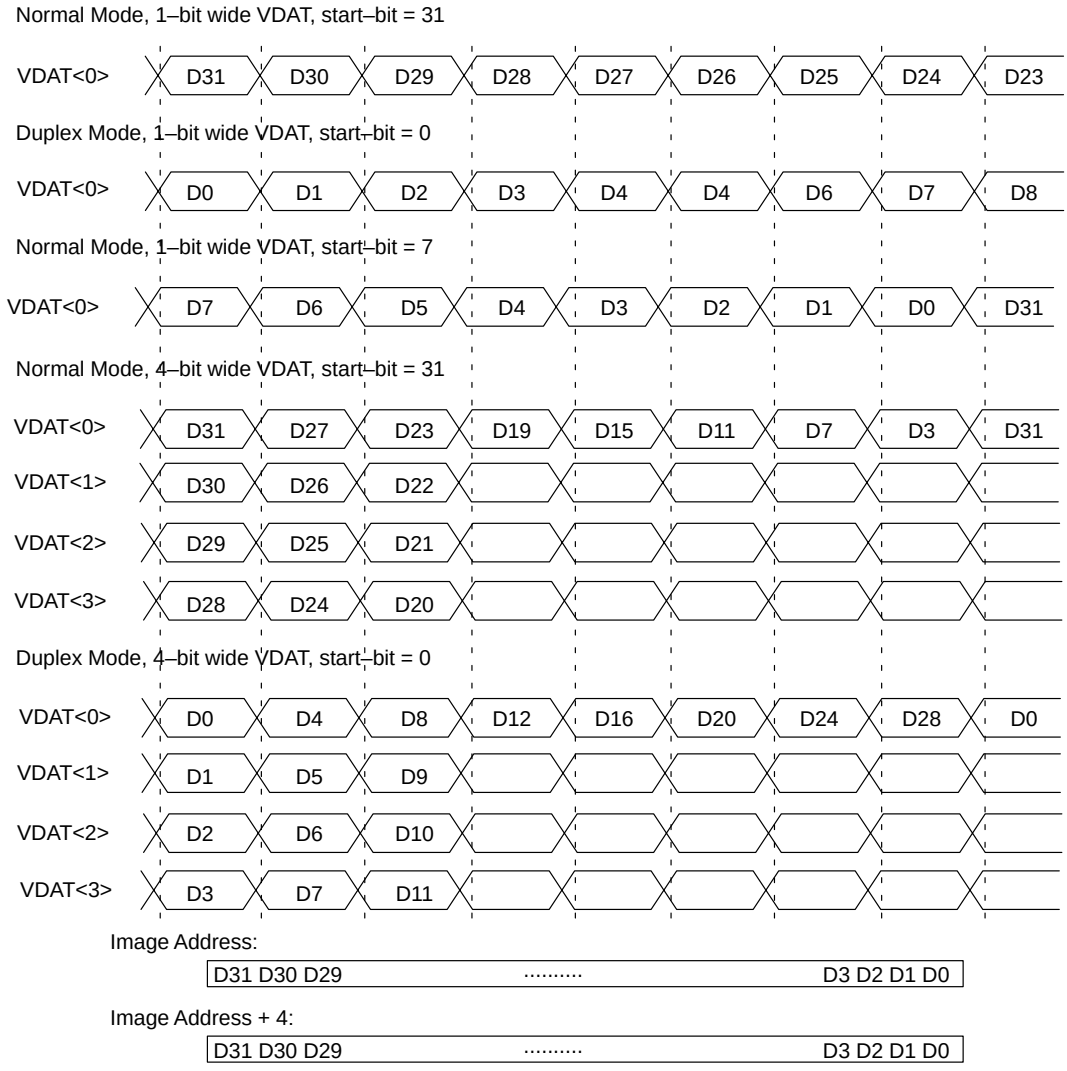




Video Transmit with Level Mode Sync Signals
Top Margin = 2, Left Margin = 3



Video Transmit Page Organization



Note: D31 is the most significant bit.

Timing 21. Video Data Order

CHAPTER

E8

MB86936 Timers

The MB86936 features two independent general-purpose 24–bit Timers (a 16–bit counter with an 8 bit prescaler). Each timer can be independently programmed to operate in one of the following three modes:

- Mode 0 – Periodic Interrupt Mode
- Mode 1 – Timeout Interrupt Mode
- Mode 2 – Square Wave Generator Mode

Timer 0 and Timer 1 have clock prescalers that can be independently clocked by CLK. The timers themselves can be independently clocked by CLK, or by the prescaler output clock (PRSCKx).

Figure E8–1 shows a block diagram of the timers and prescalers and their clock options. The prescaler output clocks are labeled PRSCKx, the internal clock is labeled CLK. CLK runs at half of the processor clock frequency.

E8.1 Timer Registers

Each timer has a Timer Control register, a Reload register, and a Count register for timer configuration and control. Timers have Prescaler registers for prescaler control. Table E8–1 shows the timer register map.

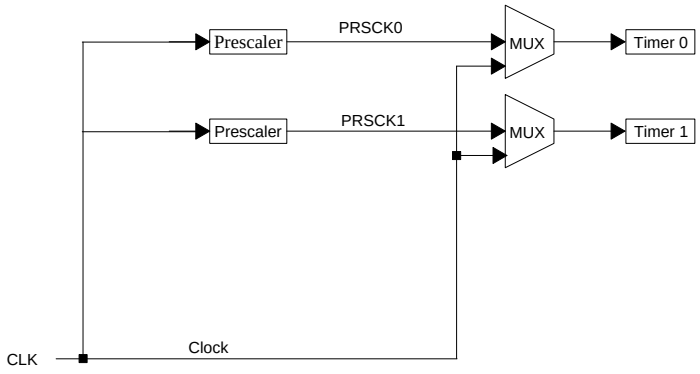


Figure E8–1. Timer Prescaler Block Diagram

Table E8–1. Timer Register Map

Address	Functional Unit	Register Name	Access	Reset State
0x00000240	Prescaler 0	Prescale Register 0	R/W	0x01
0x00000244	Timer 0	Timer Control Register 0	R/W	0
0x00000248		Reload Value 0	R/W	0
0x0000024C		Count Value 0	R	0
0x00000250	Prescaler 1	Prescale Register 1	R/W	0x01
0x00000254	Timer 1	Timer Control Register 1	R/W	0
0x00000258		Reload Value 1	R/W	0
0x0000025C		Count Value 1	R	0

E8.1.1 Prescaler Register

The Prescaler register allows selection of the prescaler clock, the prescaler output, and the prescaler counter value.

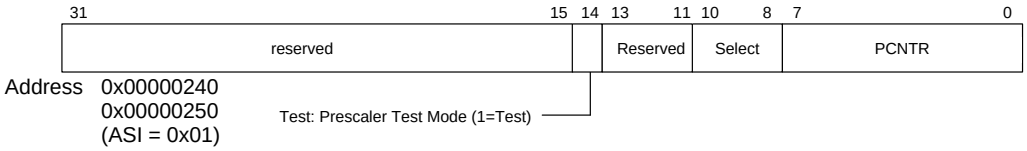


Figure E8–2. Prescaler Register

Bit 15: reserved

Bit 14: Prescaler Test Mode - The prescaler test mode is intended for factory use only, and Test should therefore remain 0 during normal operation. To enable test mode, this bit and global test bit* needs to be written 1.

Bits 13-11: Reserved.

Bits 10-8: Prescaler Output Select - Selects one of the eight prescaler outputs for PRSCKx, the prescaler clock output. Each selection is one half the frequency of the previous selection. A 0 in this field selects the prescaler counter output; a 1 selects one half the frequency of the prescaler counter output, etc.

Bits 7-0: Prescaler Counter Value - Determines the prescaler counter output frequency. The value in this field is loaded into the prescaler counter when it is written, and when timeout occurs. The prescaler counter value must be 1 or greater; a value of 1 forces the prescaler output clock (PRSCKx) low.

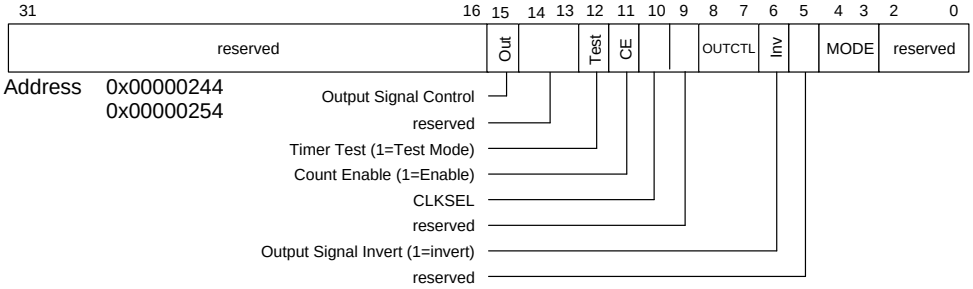
Reset initializes the Prescaler registers to 0x01. This initial state selects internal prescaler clock, the highest prescaler output clock frequency, and a Prescaler value of 1 (PRSCK forced low).

The reserved fields should be written “0” for future software compatibility.

*Global test bit is set to 1 by writing 0x1 at global test register (address 0x10000, asi=0x1)

E8.1.2 Timer Control Registers (TCR)

The TCR enables and disables the timer and allows selection and control of the timer In and Out signals, clock sources, and operation modes.



Bit 15: Output Signal Level - A read-only status bit for reading the current Out signal level. When the Out signal level is high, the OUT status bit is 1.

Bit3 14–13: Reserved.

Bit 12: Timer Test Mode - The timer test mode is intended for factory use only, and should therefore remain 0. This bit and global test bit needs to be written 1.

Bit 11: Count Enable - Enables the timer when set to 1; disables the timer when cleared to 0. The timer and its prescaler should be configured for desired operation when the timer is enabled.

Bits 10: Clock Select - Selects the timer clock source as follows:

Figure E8–3. Timer Control Registers

Table E8–2. Clock Source

CLKSEL	Clock Source
0	Internal Clock
1	Prescaler Output Clock

Bit 9: Reserved.

Bits 8-7: Out Signal Control - Selects the state of the Out pin while the timer is stopped as follows:

Table E8–3. Out State

OUTCTL	Out State
0	Remains in the current state
1	Asserted high
2	Asserted low
3	Reserved.

Bit 6: Invert - Inverts the timer Out signal when set to 1.

Bit 5: Reserved

Bits 4-3: Mode Select - Selects the timer mode of operation as follow:

Table E8–4. Timer Operating Mode

Mode	Timer Operating Mode
0	Periodic Interrupt Mode
1	Timeout Interrupt Mode
2	Square Wave Generator Mode
3	Reserved

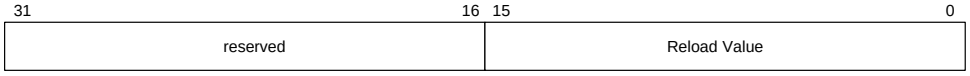
Bits 2:0: Reserved.

Reset initializes the Timer Control register to 0. The reserved fields should be written “0” for future software compatibility.

*Global test bit is set to 1 by writing 0x1 at global test register (address 0x10000, asi=0x1)

E8.1.3 Reload Register

The Reload register holds the initial value of the timer counter.



Address 0x00000248, 0x00000258, (ASI = 0x01)

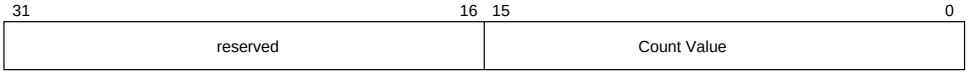
Bits 15-0: Timer Reload Value - In Modes 0 and 2, the Timer Reload Value is automatically loaded into the counter when a timeout occurs. In Mode 2, the Timer Reload Value is compared with the Count Register value to control the Out signal.

Figure E8–4. Reload Register

Reset initializes the Reload register to 0. The reserved field should be written “0” for future software compatibility.

E8.1.4 Count Register

The Count register is a read-only register that holds the current timer counter value.



Address 0x0000024C, 0x0000025C, (ASI = 0x01)

Bits 15-0:Timer Count Value - The current timer count value.

Figure E8–5. Count Register

Reset initializes the Count register to 0.

E8.2 Prescaler Operation

Figure E8–6 shows a prescaler block diagram consisting of an 8-bit counter, cascaded divide-by-two flip-flops, and selector logic.

Once the prescaler counter is loaded, the counter decrements at its clocked frequency and generates an output to the cascaded flip-flops. The flip-flops successively divide by two to provide eight frequencies for selection by the selector logic. The selector logic selects the output of the counter or one of the divided outputs as the prescaler clock output according to the value in the Prescaler register Select field. The clock output, PRSCKx, may be used to clock the timer.

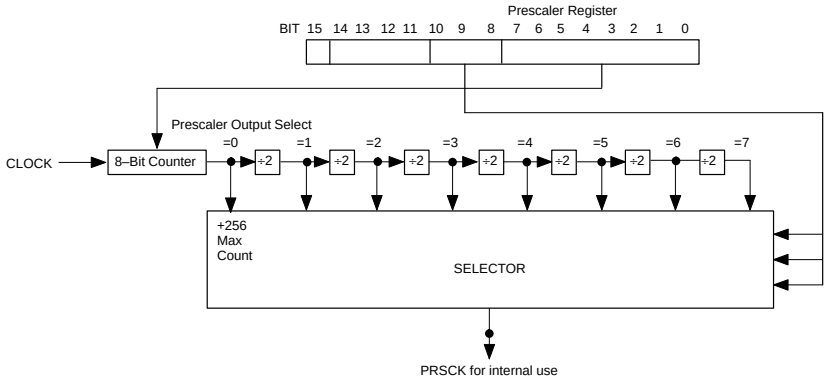


Figure E8–6. Prescaler Block Diagram

E8.2.1 Output Clock Duty Cycles

The clocks generated by the cascaded flip-flops have 50% duty cycles when selected with 1-7 in the Prescaler register Select field.

The clock generated directly by the prescaler counter, selected with 0 in the Prescaler Select field, is not a 50% duty cycle clock. The clock is asserted high until the counter reaches 1, and is then asserted low for one internal clock cycle. The clock is then asserted back to the high level while the counter reloads and counts down to 1 again. The clock is therefore low for one internal clock cycle during the countdown period.

The timer operation is independent of the prescaler clock duty cycle.

E8.2.2 Counter Loading

The 8-bit prescaler counter is loaded with the value in the Prescaler Register PCNTR field in three ways as follows:

– When the 8-bit prescaler counter decrements to 0.

– By writing to the PCNTR field.

– When the timer reload value is loaded or reloaded into the companion timer if the timer is clocked by the prescaler output clock, and the prescaler is clocked by CLK. CLKSEL must be 1 in the companion timer's Timer Control register (prescaler output clock selected to clock the timer).

The cascaded flip-flops in the divide chain are cleared when the prescaler counter is loaded.

E8.3 Timer Operation

Figure E8–7 shows a block diagram of a timer.

Timer 0 and Timer 1 can be clocked with the internal clock, or a prescaler clock. Timer clock selection is controlled by the CLKSEL field in the TCR.

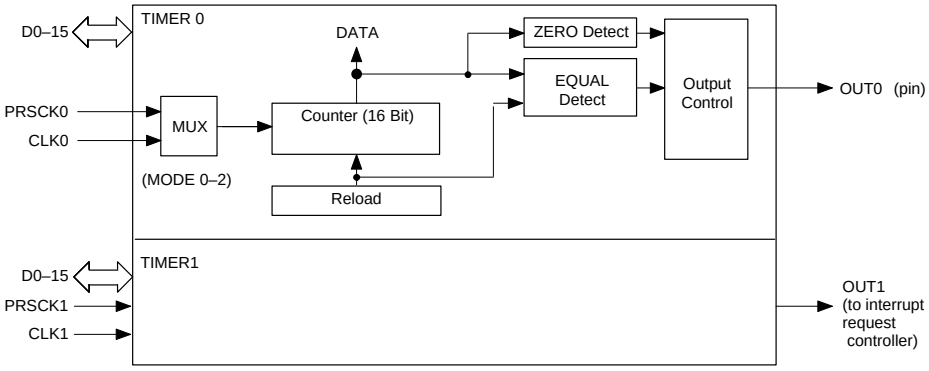


Figure E8–7. Timer Block Diagram

E8.3.1 Out Signal

The Out signal is used to indicate timeout, or the half-value of the Reload Value register during countdown. The Out signal active level is controlled by the Out Signal Control field and the Invert control bit in the TCR. The Out Signal Control field controls the state of the Out signal while the timer is stopped. The Invert bit inverts the Out signal when set to 1.

The Out signal is typically tied to an interrupt request controller to generate a processor interrupt at timeout in Modes 0, 1, and is used as a square wave in Mode 2.

TIMER0 and TIMER1 outputs are connected to bit 5 and bit 3, respectively of internal interrupt request controller. TIMER0 output is also available on MB86936 pin.

The following are the conditions for resetting and setting the Out signal level for the various timer modes during timer operation, with the TCR Invert bit cleared to 0:

Table E8–5. Out Signal

Mode	Out Signal Reset	Out Signal Set
0	Writing Reload Register; Reading Count Value Register	Timeout
1	Writing Reload Register; Reading Count Value Register	Timeout
2	When the half-value of the Reload Register is reached.	Timeout

The Out signal is inverted when the Invert bit is set to 1.

E8.3.2 Starting and Stopping the Timer

The timers are stopped following reset. Timer operation is initiated in all modes by first writing the timer mode in the TCR Mode field and setting the Count Enable control bit in the TCR to 1.

Timer operation in Modes 0, 1, 2, begins when the Reload register is written. The Reload register value is transferred to the timer counter when the Reload register is written, and the counter begins decrementing.

Once operating, each timer is stopped in the various operating modes as follows:

- Modes 0: Writing the TCR CE bit to 0.
- Mode 1: Writing the TCR CE bit to 0 or timeout.
- Mode 2: Writing the TCR CE bit to 0.

Note that the timers can be halted in all operating modes by writing to the TCR CE bit to 0.

E8.3.3 Timer Operating Modes

Each timer supports three operating modes: periodic interrupt mode (Mode 0), timeout interrupt mode (Mode 1), square wave generator mode (Mode 2). The timer operating mode is controlled by the Mode field in the TCR.

Periodic Interrupt Mode (Mode 0)

The TCR register is written so that the Out signal is initially set to the high or low state, depending on the OUTCTL field in the TCR, the timer is enabled (CE=1), and mode 0 selected. The counter then begins decrementing and the Out signal is driven low when the Reload register is written with the reload value.

When timeout occurs (counter = 0), the timer Out signal transitions to the high level. The Reload register value loads into the counter at timeout, and the counter continues decrementing. The Out signal remains at the high level until the Counter register is read or the Reload register is written.

The Out levels are inverted if Inv = 1 in the TCR.

Timeout Interrupt Mode (Mode 1)

This mode differs from Mode 0 at timeout. In Mode 1, the timer halts at timeout instead of reloading and decrementing the counter.

The TCR register is written so that the Out signal is initially set to the high or low state, depending on the OUTCTL field in the TCR, the timer is enabled (CE=1), and mode 1 selected. The counter then begins decrementing and the Out signal is driven low when the Reload register is written with the reload value.

When timeout occurs (counter = 0), the timer Out signal transitions to the high level, and the counter halts. The Out signal remains at the high level and the counter remains halted until the Count register is read or the Reload register is written. When the Count register is read or the Reload register is written, the Out signal is asserted low, the Reload register value loads into the timer counter, and the counter decrements.

The Out levels are inverted if Inv = 1 in the TCR.

Square Wave Generator Mode (Mode 2)

This mode differs from Mode 0 in the transition of the Out signal.

The TCR register is written so that the Out signal is initially set to the high or low state, depending on the OUTCTL field in the TCR, the timer is enabled (CE=1), and mode 2 selected. The counter then begins decrementing when the Reload register is written with the reload value.

When the counter decrements to half of the reload value, the Out signal is driven to the low level. When timeout occurs (counter = 0), the timer Out signal transitions to the high level. The counter reloads at timeout, and continues decrementing, repeating the Out level changes. The Out signal is therefore a square wave.

The following are the square wave high and low times for various Reload register values represented by "N":

Table E8–6. Square Wave Generator

N	**Period** (N+1)	**High Level** (N+1)/2+1	**Low Level** N/2
0	—	—	
1	2	1	
2	3	2	1
3	4	3	1
4	5	3	2
5	6	4	2
6	7	4	3

For N ≥ 2, the period of the square wave is *N+1*, the high level width is *(N+1)/2+1*, and the low level is *N/2*. N = 0 and N = 1 are special cases, as shown in the table.

The Out levels are inverted if Inv = 1 in the TCR.

Table E8–7. Timer Operating Mode Summary

	Go/Halt		Initial Value Loading	Out Signal Control	
	Go	**Halt**		**Reset**	**Set**
Mode0 Periodic Interrupt	Reload Reg Write after Mode Set and CE=1	TCR Write	Reload Reg Write, Timeout	Reload Reg Write, Count Reg Read	Timeout
Mode1 Timeout Interrupt	Reload Reg Write After Mode Set and CE=1	TCR Write Timeout	Reload Reg Write	Reload Reg Write, Count Reg Read	Timeout
Mode2 Square Wave Generator	Reload Reg Write After Mode Set and CE=1	TCR Write,	Reload Reg Write, Timeout	Equality Detection (1/2 Reload Value)	Timeout

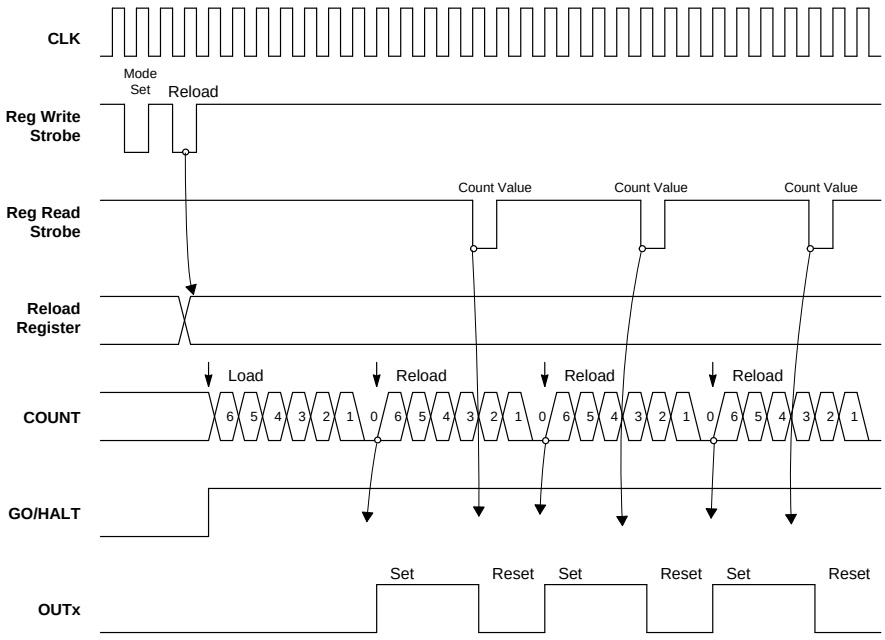


Figure E8–8. Periodic Interrupt Timing (Mode 0) Using the Internal Peripheral Clock

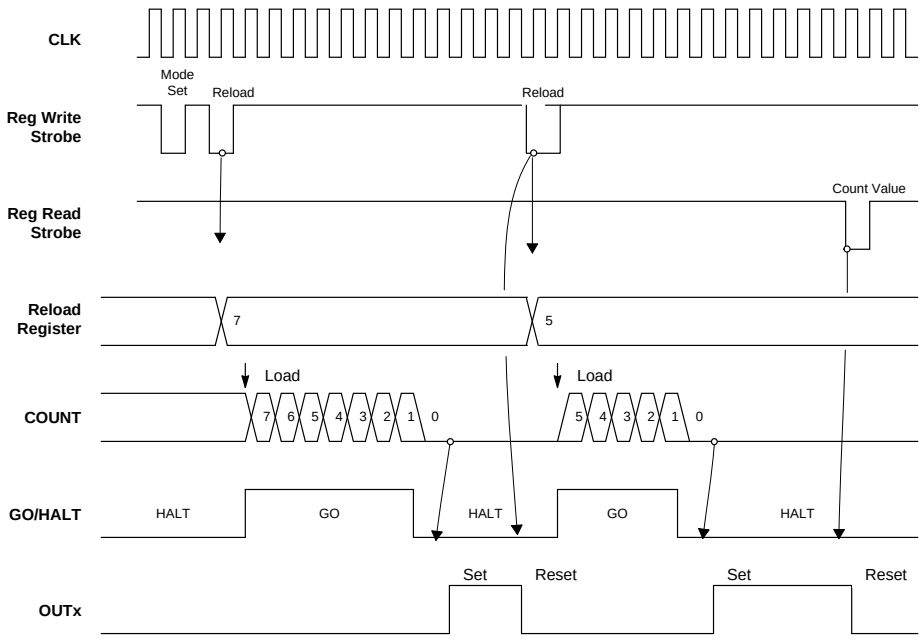
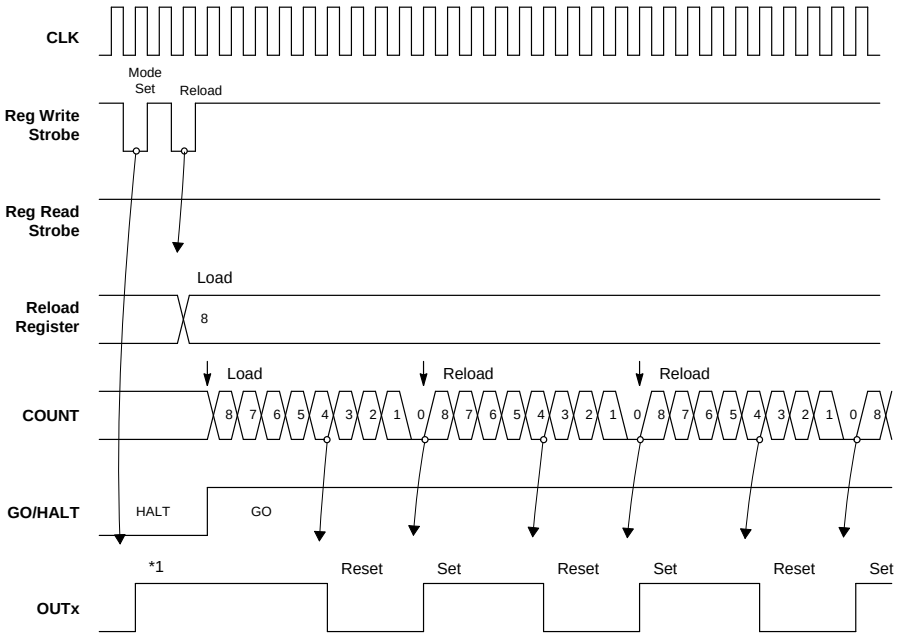


Figure E8–9. Timeout Interrupt Timing (Mode 1) Using the Internal Peripheral Clock



*1. When OUT control bit of TCR is set.

Figure E8–10. . Square Wave Generator Timing (Mode 2) Using the Internal Peripheral Clock

CHAPTER

E9

MB86936 Debug Support Unit (DSU)

The MB86936 DSU is mostly identical to that of the MB86930. Certain Floating–Point instructions (LDDF, STDF, STDFQ and all FPops) have special breakpoint behavior which is described in sections 9.1 and 9.2. In addition, the MB86936 DSU contains a feature which allows he chip to emulate the function of the MB86935. This capability is described in section 9.3.

E9.1 Data Breakpoints Immediately Before FPop

In the MB86936, the FPop1, FPop2, and instructions are not immediately trapped by the Data Address Breakpoint trap request or by the Data Value Breakpoint trap request. These traps are suspended by the DSU until they are accepted by the IU or until the processor is reset, as shown in the following code fragment example:

```
Assume Data_Address_Descriptor_Register_1 = 0x100.

st     %i0, [0x100]  ! st raises the Data Address Breakpoint trap request.
fadds  %f0, %f1, %f2 ! fadds is an FPop instruction, so it is not trapped.
fsubs  %f3, %f4, %f5 ! fsubs is an Fpop instruction, so it is not trapped.
and    %i1, %i2, %i3 ! and is trapped because it is not an Fpop
                     ! instruction.
```

E9.2 Data Breakpoints For LDDF/STDF/STDFQ

When the Data Address Breakpoint trap request or the Data Value Breakpoint trap request is used for the LDDF, STDF, and STDFQ instructions in the MB86936, the Data Address Descriptor register must have an even word address (i.e., DA[2:0] = 000), and the Data Value Descriptor register must have the breakpoint value for the least-significant word of the data (i.e., DD[31:0]).

The following code fragment shows breakpoint operation for the LDD and LDDF instructions.

```
Assume Data_Address_Descriptor_Register_1 = 0x100.
Assume Data_Value_Descriptor_Register_1   = 0x89abcdef.
Assume Memory [0x100] = 0x01234567
Assume Memory [0x104] = 0x89abcdef
ldd   [0x100], %i0   ! ldd_reg does not raise the Data Value Breakpoint
                     ! trap request.
ldd   [0x100], %f0   ! ldd_freg raises the Data Value Breakpoint trap
                     ! request
nop                  ! nop is trapped by the breakpoint trap request.
```

The instruction *ldd [0x100], %i0* does not trap because the IU has only a 32-bit data bus, and the double word load is therefore executed as two single-word loads (*ld [0x100], %i0* and *ld [0x104], %i1*) as follows:

(1) %i0 ← Memory [0x100] = 0x01234567 (ld [0x100], %i0)
(2) %i1 ← Memory [0x104] = 0x89abcdef (ld [0x104], %i1)

The first load does not trap because the data is incorrect for the breakpoint. The second load does not trap because the address is incorrect for the breakpoint.

The instruction *ldd [0x100], %f0* traps because the FPU, unlike the IU, has a 64-bit data path, so the load is executed as one double-word load. The DSU has only a 32-bit Data Value Descriptor register, so it checks the least-significant word of the data (i.e., DD[31:0] = 0x89abcdef) for the Data Value breakpoint. Both the address and the data are therefore correct for the breakpoint.

%f0 - %f1 ← Memory [0x100] = 0x01234567-89abcdef (64 bits)

Correct Address Correct Data

E9.3 Emulation Mode Control

The MB86936 has the ability to emulate the MB86935 which differs in that it does not have hardware floating point support. This emulation mode is under the control of a new register, the Emulation Mode Control Register. It is configured as follows:

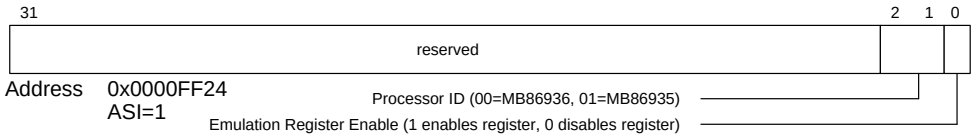


Figure E9–1. Emulation Mode Control Register

Bits 31–3: Reserved

Bits 2–1: Processor ID – These are the two least significant bits of the version number for each processor. This value is 8 (binary 1000) for the MB86936 and 9 (binary 1001) for the MB86935.

Bit 0: Emulation register enable – Setting this bit high enables the emulation mode. If the bit is low the processor will assume its “hard–wired” ID value.

Upon reset, the contents of the Emulation Control Register are cleared, thereby initializing the chip without emulation capability. As a result of putting the chip into MB86935 emulation mode, the hardware floating point unit will be disabled. And reading the Processor State Register (PSR) will yield a value of 9 instead of 8 for the version field (bits 27–24).

CHAPTER

E10

Floating-Point Unit

E10.1 Overview of the MB86936 Floating-Point Unit

The MB86936 FPU fully conforms to the ANSI/IEEE Standard 754-1985, the SPARC Architecture Version 8 specification, and the SPARC IEEE754 Implementation Recommendation except for the Nonstandard FP (NS=1) mode implementation.

Quad-precision Floating-Point operations in the MB86936 FPU cause the unimplemented_FPop Trap, and are then emulated in software. Floating-Point operations with Subnormal Number(s) cause the unfinished_FPop Trap (if NS=0), and are then emulated in software. The FPU executes all other Floating-Point operations.

E10.2 FPU Data Formats

The MB86936 architecture recognizes three floating-point data formats:

- Floating-Point Single hardware.
- Floating-Point Double hardware.
- Floating-Point Quad – NOT in hardware – by trap software only.

The Floating-Point data formats conform to the IEEE Standard for Binary Floating-Point arithmetic, ANSI/IEEE Standard 754-1985.

Figure E10-1 shows the floating-point data formats and the subwords within each format. Table E10-1 shows the subformat arrangements in memory, and in the processor registers. Tables E10-2 through E10-4 define the formats.

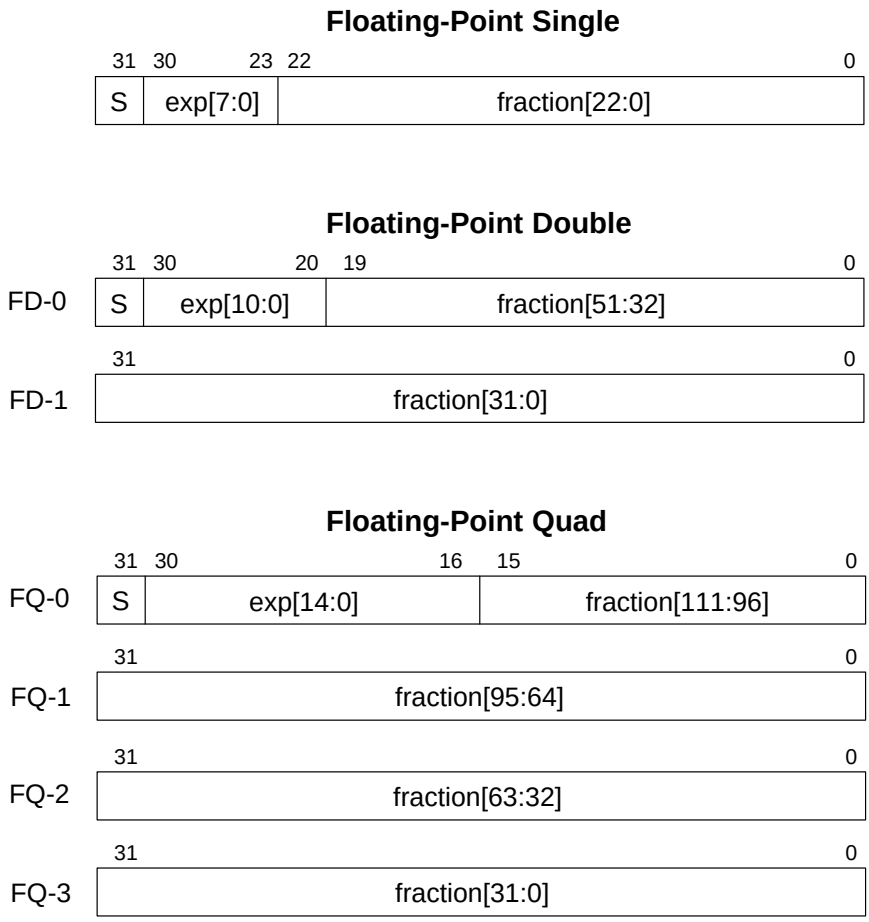


Figure E10-1. Data Formats

Table E10-1: Doubleword and Quadword Arrangement in Memory and Registers

Sub-Format Name	Sub-Format Field	Memory Address Alignment	Memory Address (byte)	Register Number Alignment	Register Number (word)
FD-0	s:exp[10:0]:fraction[51:32]	0 mod 8	n	0 mod 2	r
FD-1	fraction[31:0]	4 mod 8	n+4	1 mod 2	r+1
FQ-0	s:exp[14:0]:fraction[111:96]	0 mod 16	n	0 mod 4	r
FQ-1	fraction[95:64]	4 mod 16	n+4	1 mod 4	r+1
FQ-2	fraction[63:32]	8 mod 16	n+8	2 mod 4	r+2
FQ-3	fraction[31:0]	12 mod 16	n+12	3 mod 4	r+3

Table E10-2: Floating-Point Singleword Format Definition

s = sign (1 bit)	
e = biased exponent (8 bits)	
f = fraction (23 bits)	
u = undefined	
normalized value (0<e<255):	$(-1)^s \times 2^{e-127} \times 1.f$
subnormal value (e=0):	$(-1)^s \times 2^{-126} \times 0.f$
zero (e=0):	$(-1)^s \times 0$
signaling NaN:	s = u; e = 255 (max); f = .0uu – uu (At least one bit of the fraction must be nonzero.)
quiet NaN:	s = u; e = 255 (max); f = .1uu – uu
–∞ (negative infinity)	s = 1; e = 255 (max); f = .000 – 00
+∞ (Positive Infinity)	s = 0; e = 255 (max); f = .000 – 00

Table E10-3 Floating-Point Doubleword Format Definition

s = sign (1 bit) e = biased exponent (11 bits) f = fraction (52 bits) u = undefined	
normalized value (0<e< 2047):	$(-1)^s \times 2^{e-1023} \times 1.f$
subnormal value (e=0):	$(-1)^s \times 2^{-1022} \times 0.f$
zero (e=0):	$(-1)^s \times 0$
signaling NaN:	s = u; e = 2047 (max); f = .0uu – uu (At least one bit of the fraction must be nonzero.)
quiet NaN:	s = u; e = 2047 (max); f = .1uu – uu
–∞ (negative infinity)	s = 1; e = 2047 (max); f = .000 – 00
+∞ (Positive Infinity)	s = 0; e = 2047 (max); f = .000 – 00

Table E10-4: Floating-Point Quadword Format Definition

s = sign (1 bit) e = biased exponent (15 bits) f = fraction (112 bits) u = undefined	
normalized value (0<e<32767):	$(-1)^s \times 2^{e-16383} \times 1.f$
subnormal value (e=0):	$(-1)^s \times 2^{-16382} \times 0.f$
zero (e=0):	$(-1)^s \times 0$
signaling NaN:	s = u; e = 32767 (max); f = .0uu – uu (At least one bit of the fraction must be nonzero.)
quiet NaN:	s = u; e = 32767 (max); f = .1uu – uu
–∞ (negative infinity)	s = 1; e = 32767 (max); f = .000 – 00
+∞ (Positive Infinity)	s = 0; e = 32767 (max); f = .000 – 00

E10.3 FPU Registers

The FPU contains a set of 32 general purpose registers, a Floating–Point deferred trap queue (FQ) and a floating–point state register (FSR). Two flags in the Processor State Register (PSR) control the enabling and disabling of the FPU.

E10.3.1 Floating-Point State Register (FSR)

The Floating-Point State Register (FSR) is the FPU control and status register. The register contains FPU control and status information.

The FSR is read and written with the STFSR and LDFSR instructions, respectively. The RD, TEM, NS, fcc, aexc, and cexc fields are readable and writable, but the ver, ftt, and qne fields are read-only. The qne field is cleared by reset; the ftt field is cleared by reset and by the STFSR instruction.

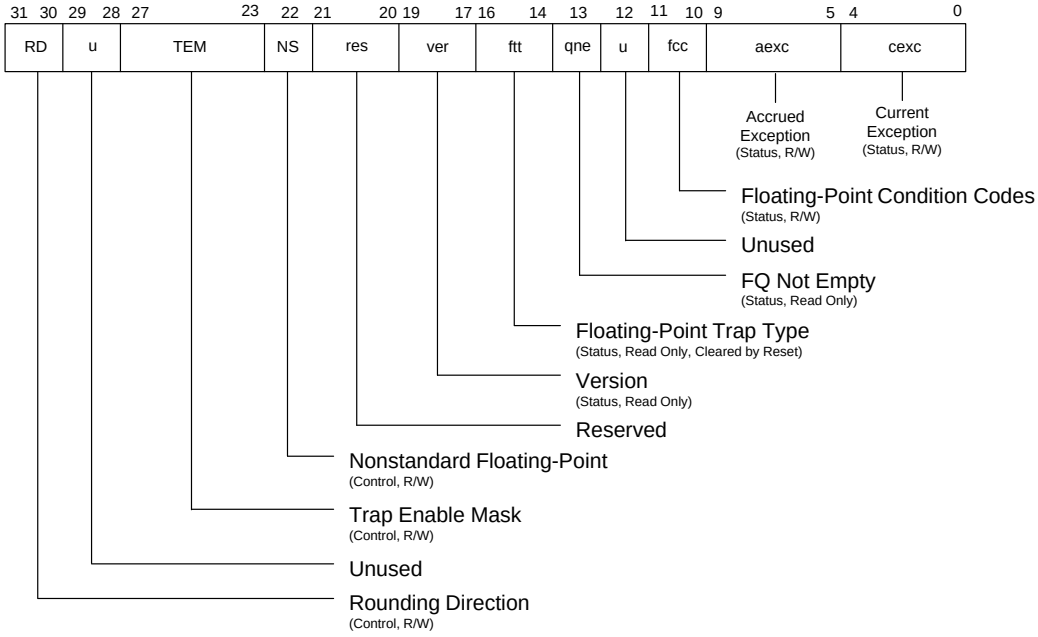


Figure E10-2. Floating-Point State Register

Bits 31-30: Rounding Direction (RD) [*Control Field, Readable and Writable*]

This field selects the rounding direction for floating-point results according to ANSI/IEEE Standard 754-1985 as follows:

RD	Rounding Direction
0	Nearest (even if tie)
1	Zero
2	+ Infinity
3	– Infinity

RD is read and written with the STFSR and LDFSR instructions, respectively.

Bits 29-28: Unused (U)

This field is undefined when read with the STFSR instruction. To ensure future compatibility, the field should be written 0 when the LDFSR instruction is used.

Bits 27-23: Trap Enable Mask (TEM) [*Control Field, Readable and Writable*]

This field selectively enables and disables assertion of an fp_exception trap in response to one or more floating-point exceptions that are indicated in the cexc field of the FSR. A 1 in the TEM field enables an fp_exception trap for the corresponding floating-point exception; a 0 disables an fp_exception trap for the corresponding floating-point exception. (See Section 10.4.3, *IEEE 754 Exception*, for details).

The TEM field floating-point exception masks are as follows:

Bit	Exception
27	NVM
26	OFM
25	UFM
24	DZM
23	NXM

TEM is read and written with the STFSR and LDFSR instructions, respectively.

Bit 22: Nonstandard FP (NS) [*Control Bit, Readable and Writable*]

This bit sets the FPU in the Nonstandard mode. The Nonstandard mode is also called the Fast mode and the Abrupt Underflow mode. The other (standard) mode is called IEEE Underflow mode, or the Gradual Underflow Mode.

When the NS bit is 1, a subnormal (denormalized) floating-point number in each source f register is considered to be zero by the FPU. The FPU replaces a positive subnormal operand with +zero, and a negative subnormal operand with –zero. The FPU does not assert an exception (including inexact (nv) exception) following a replacement, and does not indicate that a replacement has occurred.

The FPU does not produce any subnormal numbers as FPop results, even if underflow occurs. Instead, the FPU outputs the underflow default results (± zero or ± the smallest

normalized number), depending on rounding mode and its sign. Unlike the IEEE 754 underflow handling, this underflow handling maintains consistency with the overflow handling. (See Section 10.5.2, *Overflow, Underflow, and Inexac*t, for details.)

When the NS bit is 0, subnormal (denormalized) floating-point operand(s) or result(s) invoke the unfinished_FPop trap. The FPop is emulated in software to conform to ANSI/IEEE Standard 754-1985. (See Section 10.5.4, *Emulation for Subnormal Number, Invoked by the Unfinished_FPop Trap*, for details.)

NS is read and written with the STFSR and LDFSR instructions, respectively.

Programming Notes:

(1) The NS bit does not affect the FMOVs, FNEGs, FABSs, STfp, or LDfp instructions. These instructions are not affected by the precision type (single, double, or quad) or numbers (NaN, Zero, Subnormal Number, etc.). They just transfer contents as data between f registers or between an f register and memory whether the NS bit is 1 or 0. Therefore, they never raise any fp_exception, and they never have a subnormal number replaced by zero.

(2) The NS bit is defined as implementation-dependent in the SPARC Architecture Manual (Version 8). This definition is only for SPARClite. Other SPARC devices may have other definitions. (The SPARClite definition of the NS bit is not the same as the definition given in the SPARC IEEE 754 Implementation Recommendation section of the SPARC V8 manual.)

Bits 21-20: Reserved

This field reads 0 when read with the STFSR instruction. To ensure future compatibility, the field should be written 0 when the LDFSR instruction is used.

Bits 19-17: Version (ver) [*Status Field, Read Only*]

This field Identifies the FPU version. The MB86936 FPU version is 6. The ver field can be read with the STFSR instruction, but is not affected by the LDFSR instruction.

Programming Note:

Software identifies the FPU as belonging to the SPARClite MB86936 processor by reading “0” in the PSR implementation (impl) field (identifies Fujitsu Microelectronics, Inc. implementation), by reading “6” the PSR version (ver) field (identifies processor as MB86936), and by reading “6” in the FSR version field (identifies the FPU version).

Bits 16-14: Floating-Point Trap Type (ftt) [*Status Field, Read Only*]

This field identifies the floating-point exception trap types. The ftt field is a read-only field that identifies the type of floating-point exception that occurs as follows (see Section 10.4.2, *Floating-Point Exception Trap Types*, for details):

ftt	Trap Type
0	none
1	IEEE_754_exception
2	unfinished_FPop
3	unimplemented_FPop
4	sequence_error
5	hardware_error (not implemented in the MB86936)
6	invalid_fp_register
7	reserved

The ftt field can be read with the STFSR instruction. Reset, execution of the STFSR instruction, and execution of an FPop with no floating-point exceptions clear the ftt field. The LDFSR instruction does not affect ftt.

Programming Note:

The SPARC Architecture Manual (Version 8) specifies that clearing of the ftt field to 0 following execution of the STFSR instruction is implementation-dependent. The MB86936 FPU clears the ftt field following execution of the STFSR instruction, but other SPARC FPUs may not.

Bit 13: FQ Not Empty (qne) [*Status Bit, Read Only*]

This bit indicates whether the floating-point deferred-trap queue (FQ) contains any FPop instruction. If qne=0, the FQ is empty; if qne=1, the FQ is not empty. Reset and execution of successive STDFQ instructions empties the FQ, resulting in qne=0.

The qne bit can be read with the STFSR instruction. The LDFSR instruction does not affect qne.

Bit 12: Unused (u) - This bit is undefined when read with the STFSR instruction. To ensure future compatibility, the bit should be written 0 when the LDFSR instruction is used.

Bits 11-10: FP Condition Codes (fcc) [*Status Field, Readable and Writable*]

The fcc field is updated only by a floating-point compare instruction such as FCMP or CMPE as follows:

fcc	Relation
0	frs1 = frs2
1	frs1 < frs2
2	frs1 > frs2
3	frs1 ? frs2 (unordered)

If either frs1 or frs2 is a signaling NaN (SNaN) or a quiet NaN (QNaN), the fcc field becomes 3 (unordered).The fcc field is unchanged if a floating-point compare instruction generates any fp_exception.

The FBfcc instruction bases its control transfer on the fcc field. The field can be read and written with the STFSR and LDFSR instructions, respectively.

Programming Note:

> The FBfcc instruction can branch based on the fcc field which was changed by the STFSR, not by FCMP instructions. In the MB86936 FPU, the STFSR can be followed by the FBfcc without any instructions between. However, other SPARC FPUs may require three instructions between the STFSR and the FBfcc.
>
> Similarly, the SPARClite FPU does not require any instructions between the FCMP/FCMPE instructions and a following FBfcc, but some SPARC FPUs require one non-FPop2 instruction between these instructions.

Bits 9-5: Accrued Exception (aexc) [*Status Field, Readable and Writable*]

This field accumulates IEEE_754 floating-point exceptions that occur while their traps are disabled using the TEM field as follows:

FSR bit	Exception
5	nxa
6	dza
7	ufa
8	ofa
9	nva

The aexc field is unchanged if an FPop generates an IEEE_754_exception trap or other fp_exception trap.

After an FPop is executed without any fp_exception traps except an IEEE_754_exception trap, the TEM and cexc field are logically ANDed together. If the result is nonzero, an IEEE_754_exceptions trap is generated; otherwise, the new cexc field is ORed into the aexc field.

The aexc field is read and written with the STFSR and LDFSR instructions, respectively. (See Section 10.4.3, *IEEE 754 Exception*, for details.)

Bits 4-0: Current Exception (cexc) [*Status Field, Readable and Writable*]

This field identifies IEEE_754 floating-point exceptions that were generated by the most recently executed FPop instruction as follows:

FSR bit	Exception
0	nxc
1	dzc
2	ufc
3	ofc
4	nvc

The cexc field is updated either when an FPop is completed without a trap, or when an FPop causes an IEEE_754_exception trap. Only one IEEE_754 exception is selected for the IEEE_754_exception trap; i.e., only one bit in the cexc field becomes 1, and the rest become 0's. The cexc field is unchanged if an FPop generates an fp_exception trap except the IEEE_754_exception trap.

The cexc field is read and written with the STFSR and LDFSR instructions, respectively. (See Section 10.4.3, *IEEE 754 Exception*, for details.)

Programming Note:

> The cexc field can be changed with the STFSR instruction. However, this change does not generate new fp_exception traps. The cexc is evaluated for fp_exception traps only when an FPop is executed; not when an STFSR is executed.

E10.3.2 FPU Register Set

The MB86936 FPU contains thirty-two 32-bit floating-point f registers that are designated f[0] to f[31]. The FPU f registers are not windowed as are the IU r registers. Each floating-point instruction therefore has access to all 32 f registers. The f registers can be read and written with FPop instructions and with load/store floating-point instructions (particularly LDF, LDDF, STF, and STDF).

A single f register, such as f[0] or f[1], can hold one single-precision operand. A double-precision operand requires an aligned pair of f registers, such as f[0]-f[1] or f[2]-f[3]. A quad-precision operand requires an aligned quadruple of f registers, such as f[0]-f[1]-f[2]-f[3] or f[4]-f[5]-f[6]-f[7]. The f registers can therefore hold a maximum of 32 single-precision, 16 double-precision, or 8 quad-precision operands.

The floating-point instructions that access floating-point double-precision data in the f-registers assume double alignment. The least-significant bit of a double-word f register number must be zero (i.e., f[0], f[2]...; not f[1], f[3]...). Similarly, the least-significant two bits of a quad-word f register number must be zeros (i.e., f[0], f[4]...; not f[1], f[2], f[3], f[5], f[6], f[7]...).

E10.3.3 Floating-Point Deferred-Trap Queue (FQ)

The Floating-Point Deferred-Trap Queue (FQ) is a queue of three double-word entries. Each entry holds an FPop instruction, and the Program Counter (PC) address from which it was fetched. The instructions remain in the queue until executed by the FPU, which can execute the instructions concurrently. When a floating-point trap occurs, the FQ holds the FPop instructions that are pending completion by the FPU.

The FQ is a first-in-first-out queue. The FPU therefore cannot change the order of completion of the instructions in the FQ. The number of entries of the FQ is implementation-dependent, so FQs in other SPARC devices may hold a different number of entries.

Figure E10-5 illustrates FQ operation. An FPop instruction enters the FQ when dispatched by the IU to the FPU. The first instruction is stored in the first (front) FQ entry and remains there until executed. The next instruction is stored in the second FQ entry if the first instruction has not executed, or in the first FQ entry if the first instruction has executed. The next instruction is stored in the third FQ entry if neither of the previous two instructions has executed, in the second entry if only the first instruction has executed, or in the first entry if both of the preceding instructions have executed.

The FPop instruction in the first entry exits the FQ when it executes without a floating-point exception, and the instructions that remain in the queue move up one entry towards the front of the queue. If the instruction causes a floating-point exception, it stays in the front entry, other instructions in the FQ do not move toward the front of the queue, and the FPU changes from the fp_execution state to the fp_exception_pending state.

When a floating-point exception occurs, the trap handler reads the contents of the FQ with the Store-Double Floating-Point Queue (STDFQ) instruction, which stores the contents of the front entry of the FQ into memory. The PC address part of the entry is stored into memory at the effective address, and the instruction code part of the entry is stored at the effective address + 4. All remaining instructions move up one entry.

Each instruction exits the FQ when it is stored to memory. When an STDFQ instruction empties the FQ, the qne bit is cleared to 0. (See Section 10.3.1, *Floating-Point State Register (FSR)*, for a description of the qne bit.)

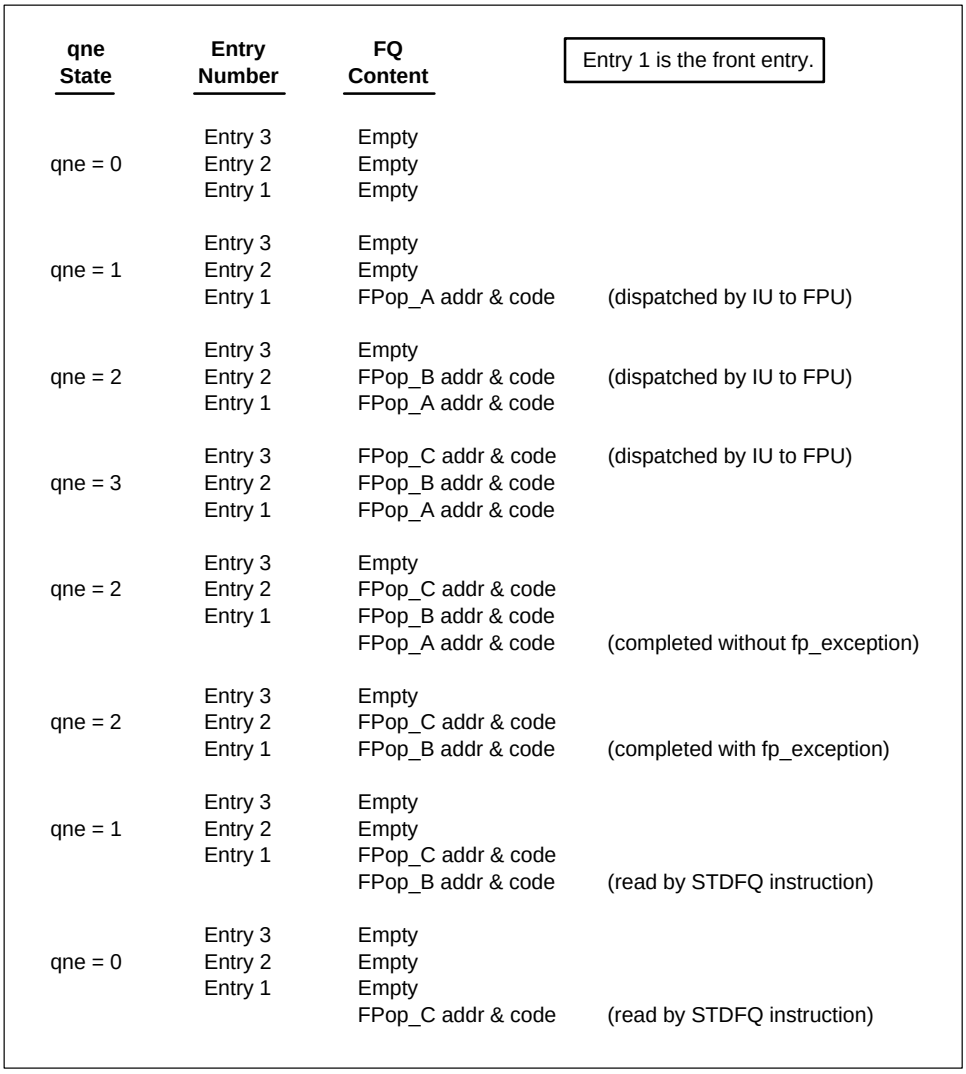


Figure E10-5. Floating-Point Deferred-Trap Queue Operation

Programming Note:

The floating-point trap handler uses STDFQ instructions to access the FPop instructions in the FQ; that is, the instruction in the first FQ entry that caused the floating-point exception, and the remaining instructions in the FQ that are pending execution. The handler may emulate the FPops in software, may re-execute the FPops in the FPU, or may discard the FPops and invoke an error handler.

E10.3.4 EF bit in the PSR

The Enable Floating-Point (EF) bit is Bit 12 of the Processor State Register (PSR).

Table E10-7 shows the effect of the EF bit state on the FPop, LDfp, STfp, and FBfcc instructions. When EF = 1, these instructions can be executed. When EF = 0, these instructions cause the fp_disabled trap, and the FPop1/FPop2 instructions are not dispatched to the FPU.

Table E10-7: EF Bit Effect on Instruction Execution

EF Bit State	Effect on FPop1/FPop2/LDfp/STfp/FBfcc
0	Causes fp_disabled Trap
1	Executed by the FPU

Although the EF bit controls whether a floating-point instruction is trapped and whether an FPop instruction is dispatched to the FPU, it does not control the FPU directly. The FPU continues to execute FPop instructions in the FQ even when the FPU is disabled.

Programming Note:

An Operating System (OS) can use the EF bit to determine whether a particular process uses the FPU. If a process does not use the FPU, the FPU registers (f-reg/FSR/FQ) do not have to be saved and restored across context switches. The OS just sets the EF bit to 0, and switches to the process.

If the next process uses the FPU, the OS must wait until the FPU finishes all instructions in the FQ. The STFSR instruction can be used in this situation because STFSR waits for completion of all instructions in the FQ before executing. If one of the instructions in the FQ requests a floating-point trap, the STFSR is trapped so that the OS can handle the exception before switching the processes. Once the FQ is empty, the OS saves the 32 f registers and the FSR for later restoration. The FSR ftt and qne fields are not writable, but both must be 0 across context switches.

E10.4 Floating-Point Traps and FPU States

This section describes traps associated with floating-point instructions, and the fp_execute, fp_exception_pending, and fp_exception FPU states.

E10.4.1 Traps Associated with Floating-Point Instructions

Floating-point instructions consist of FPop (FPop1, FPop2), LDfp (LDF, LDDF, LDFSR), STfp (STF, STDF, STFSR, STDFQ), and FBfcc instructions. There are 4 traps associated with the floating-point instructions: fp_disabled trap, fp_exception trap, mem_address_not_aligned trap, and data_access_exception trap.

fp_disabled trap

If the EF bit of the Processor State Register (PSR) is 0 an attempt to execute a floating point instruction will cause an fp_disable trap.

fp_exception trap

An fp_exception trap has an IU trap type (tt) of 8, and its priority is 9.

The fp_exception trap also has 6 floating–point trap types (ftt=1 to ftt=6). One of the trap types, the IEEE_754_exception trap(ftt=1), has 5 exception types: nv, of, uf, dz, nx. The sequence_error trap (ftt=4) is a precise trap. The rest are deferred traps.

deferred fp_exception trap

The IEEE_754_exception (ftt=1), unfinished_FPop (ftt=2), unimplemented_FPop (ftt=3), and invalid_fp_register (ftt=6) traps can be generated only by an FPop instruction, not by an LDfp, STfp, or FBfcc instruction. When the dispatched FPop is completed in the FPU with an fp_exception, the FPU requests the fp_exception trap to the IU. Such traps are called deferred traps.

The deferred trap request is accepted by the IU when it executes another floating-point instruction, which is then trapped. An fp_exception trap handler can find the FPop with the deferred trap request in the front entry of the FQ, which is why the queue is called the Floating-Point Deferred-Trap Queue (FQ).

precise fp_exception trap

Unlike the deferred traps that can be generated only by FPop instructions, the sequence_error (ftt=4) trap can be generated by all floating-point instructions except STF, STDF, and STFSR. The trap is always generated when a floating-point instruction is in the IU, not in the FPU. As a result, the floating-point instruction itself is trapped. Such traps are called precise traps. When an FPop instruction is trapped, it is not dispatched to the FPU.

mem_address_not_aligned and data_access_exception traps

The LDfp and STfp instructions may cause mem_address_not_aligned traps and data_access_exception traps, as do the LD_integer and ST_integer instructions. (Note that the LDfp and STfp instructions are executed by the IU, not by the FPU.)

Programming Notes:

(1) In SPARClite, when the LDfp has the data_access_exception trap, its destination register (f register or FSR) remains unchanged. However, other SPARC processors may fill the register with a predetermined constant value (such as all 1's).

(2) In the SPARC Version 8 specification, it is recommended that the LDDF/STDF instruction have the fp_exception trap with ftt=6 (invalid_fp_register) when its operand (frd) is misaligned (i.e., odd number in the frd field). In the MB86936, however, the LDDF/STDF instruction with misaligned operand does not cause any traps. The LSB of the frd field of the LDDF/STDF instruction is ignored (i.e., forced to 0 internally).

Furthermore, the LDD/STD instruction with misaligned operand has no trap in the MB86936, and the LSB of the frd field is ignored (i.e., forced to 0 internally).

E10.4.2 Floating-Point Exception Trap Types

The Floating-Point Trap Type (ftt 7) is reserved in the SPARC Version 8 specification for future expansion. The Hardware Error Trap Type (ftt 5) is not implemented in the MB86936 FPU. The rest of the Floating-Point Trap Types are implemented in the MB86936, including a new trap type defined in the SPARC Version 8 specification, ftt 6 (invalid_fp_register).

The MB86936 FPU uses the ftt 2 (unfinished_FPop) trap type to handle subnormal numbers. (See Section 10.5.4, *Emulation for Subnormal Number Invoked by the Unfinished_FPop Trap*, for details.) The FPU also uses the ftt 3 (unimplemented_FPop) trap type to handle quad precision floating-point operations. (See Section 10.5.5, *Emulation for Quad-precision Operation, Invoked by the Unimplemented_FPop Trap*, for details.)

An FPop instruction may have more than one cause, and each cause is assigned an ftt. For example, both ftt 2 and ftt 3 apply when an operand of a quad precision FPop contains a subnormal number, and both ftt 4 and ftt 6 apply when an FPop having an invalid fp register is executed in the fp exception mode. However, only one ftt can be asserted in each of these cases.

To resolve these conflicts, each ftt is assigned a unique priority in the FPU, as shown in Table E10-9. Therefore, when an operand of a quad precision FPop contains a subnormal number, ftt 3 (unimplemented_FPop) is asserted because it has a higher priority than ftt 2 (unfinished_FPop); and when an FPop having an invalid fp register is executed in the fp exception mode, ftt 4 (sequence_error) is asserted because it has a higher priority than ftt 6 (invalid_fp_register).

Table E10-9: Floating-Point Trap Types

ftt	Priority	Trap Type	Implementation in MB86936 FPU
0	–	none	no fp_exception trap
1	5	IEEE_754_ exception	IEEE 754 exceptions (nv, of, uf, dz, nx)
2	4	unfinished_FPop	subnormal number in operand(s) or result
3	3	unimplemented_ FPop	quad-precision floating-point operation
4	1	sequence_error	fp instruction in fp exception mode
5	–	hardware_error	not implemented in the MB86936 FPU
6	2	invalid_fp_register	misaligned f register(s) (frs1/frs2/frd)
7	–	reserved	reserved for future expansion

ftt=1, IEEE_754_exception

An IEEE_754_exception indicates that the FPU had the floating-point exception which conforms to the ANSI/IEEE Standard 754-1985. The IEEE_754 exception type is encoded in the cexc field. However, the destination f register, aexc, and fcc are not affected by the IEEE_754_exception trap.

ftt=2, unfinished_FPop (subnormal number in operand(s) or result)

An unfinished_FPop indicates that the FPU was unable to generate correct results or exceptions as defined by ANSI/IEEE Standard 754-1985. In the MB86936, this trap arises when subnormal number(s) are in operand(s) or the result, and when NS=0.

ftt=3, unimplemented_FPop (quad precision floating-point operation)

An unimplemented_FPop indicates that the FPU has decoded an FPop that is not implemented. This trap arises in the MB86936 when the quad precision floating-point operation is in the FQ.

Programming Note:

In the case of an unfinished_FPop or unimplemented_FPop floating-point trap type, software should emulate or re-execute the exception-causing instruction, and update the FSR and destination f register.

ftt=4, sequence_error

A sequence_error indicates abnormal error conditions in the FPU. It is caused when:

(1) An attempt is made to execute an STDFQ instruction when the floating-point deferred-trap queue (FQ) is empty.

(2) An attempt is made to execute a floating-point instruction (such as FPop, LDfp, and FBfcc; except STfp) when the FPU is in the fp_exception state.

ftt=5, hardware_error (not implemented in the MB86936 FPU)

A hardware_error indicates that the FPU has detected a catastrophic internal error, such as an illegal state or a parity error during an f register access.

Programming Note:

> If a sequence_error or hardware_error occurs during execution of user code, it may not be possible to recover sufficient state information to continue execution of the user application.

ftt=6, invalid_fp_register

An invalid_fp_register indicates that one or more register(s) of an FPop is (are) misaligned; i.e., a double-precision register number is not 0 mod 2, or a quadruple-precision register number is not 0 mod 4.

Programming Note:

> This ftt is newly-defined in the SPARC Version 8 specification. The MB86936 FPU supports it. However, other SPARC processors may generate an illegal_instruction trap instead.

ftt=7, reserved

The Floating-Point Trap Type 7 is reserved for future expansion.

E10.4.3 IEEE 754 Exception

The IEEE 754 exception has five exception types: invalid, overflow, underflow, division-by-zero, and inexact. Figure E10-6 shows the FSR fields affected by the exceptions, which are generated as follows:

invalid (nv) exception [cexc.nvc, aexc.nva, TEM.NVM]:

An operand is improper for the operation to be performed. For example, (0/0), and (infinity-infinity) are invalid. 1=invalid, 0=valid.

overflow (of) exception [cexc.ofc, aexc.ofa, TEM.OFM]:

The infinitely precise correct result is larger in magnitude than the largest normalized number in the specified format, and smaller in magnitude than infinity. 1:overflow, 0:no overflow.

underflow (uf) exception [cexc.ufc, aexc.ufa, TEM.UFM]:

If NS=1: The infinitely precise correct result is smaller in magnitude than the smallest normalized number in the indicated format, and larger in magnitude than zero.

If NS=0 and UFM=1: The nonzero result is tiny. Tininess may be detected before or after rounding.

If NS=0 and UFM=0: The nonzero result is tiny, and a loss of accuracy occurs. Tininess may be detected before or after rounding. Loss of accuracy may be either a denormalization loss, or an inexact result. 1:underflow, 0:no underflow. (See Section 10.5.4, *Emulation for Subnormal Number, Invoked by the Unfinished_FPop Trap* for details.)

division-by-zero (dz) exception [cexc.dzc, aexc.dza, TEM.DZM]:

X/0, where X is subnormal or normalized. Note that 0/0 does not set the dz bit. 1:division-by-zero, 0:no division-by-zero.

inexact (nx) exception: [cexc.nxc, aexc.nxa, TEM.NXM]

The rounded result of an operation differs from the infinitely precise correct result. 1:inexact result, 0:exact result.

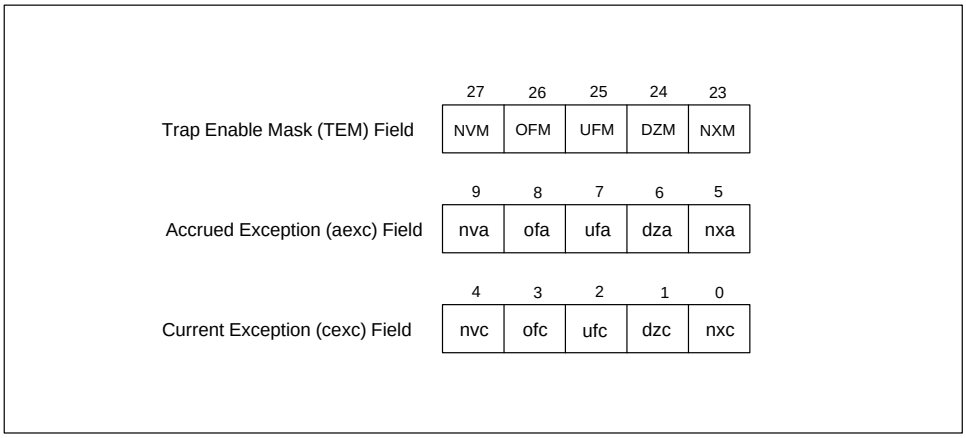


Figure E10-6. FSR TEM, aexc, and cexc Fields

When an FPop generates an IEEE 754 exception, the FPU behaves as follows;

```
FPop_generates_IEEE_754_exception {
 cexc'=IEEE_754_exception_type(s)_generated_by_the_FPop;
 if (TEM & cexc')==0
 No_Trap (ftt=0; cexc=cexc'; fcc=fcc_result; aexc=(aexc|cexc'); f[frd]=result);
 else
 IEEE_754_Exception_Trap (ftt=1; cexc=selected_one_IEEE_754_exception_type);
 }
```

The IEEE 754 exception has multiple exception types in only two cases.

(1) of and nx: whenever the overflow exception arises, the inexact exception also arises.

(2) uf and nx: whenever the underflow exception arises, the inexact exception also arises.

Exception: If NS=0 and UFM=1 (*tininess,* and not *loss_of_accuracy*), then the underflow exception arises without the inexact exception.

When an IEEE 754 exception invokes the fp_exception trap, only one IEEE 754 exception type is selected to be 1 in the cexc field, even if the IEEE 754 exception has a multiple exception type. The selection is based on the value of TEM, and the priority of each exception type (the priority of *uf* and *of* is higher than the priority of *nx*). (See Section 10.5.2, *Overflow, Underflow, and Inexact,* for details.)

E10.4.4 Floating-Point Trap Handlers

When a floating-point trap occurs, the results are as follows:

(1) The ftt field is updated.

(2) The fcc field is unchanged.

(3) The aexc field is unchanged.

(4) The cexc field is unchanged, except for an IEEE_754_exception. When an IEEE_754_exception occurs, the cexc field contains exactly one bit that is 1, which corresponds to the exception that caused the trap. The remaining bits are 0's.

(5) The value of the destination f register (frd) is unchanged.

The sequence_error, hardware_error, and invalid_fp_register trap types are unlikely to arise in the normal course of computation. They are essentially unrecoverable from the point of view of user applications.

In contrast, IEEE_754_exception, unfinished_FPop, and unimplemented_FPop are likely to arise occasionally in the normal course of computation, and must be recoverable by software. Software (such as emulator software) should define the values of the fcc, aexc, and cexc fields, and generate the value of the destination f register, as appropriate.

Programming Note:

If an unfinished_FPop or unimplemented_FPop trap handler invokes a user's IEEE 754 trap handler that is designed to be invoked by an IEEE_754_exception trap, the unfinished_FPop and unimplemented_FPop trap handler must produce the results as if hardware produced them. (i.e., the fcc and aexc fields, and destination f register are unchanged. Only one bit of the cexc field is 1.)

Such user's IEEE 754 trap handler may require the address and code of the FPop instruction that caused the exception. Furthermore, the user's handler expects that the FQ has been analyzed and emptied (qne=0), and that the ftt field has been analyzed and cleared (ftt=0). (The ftt field must not be referred in user's IEEE 754 handler because the handler is designed to handle the IEEE 754 exception.)

Figure E10-7 summarizes Floating-Point trap handling.

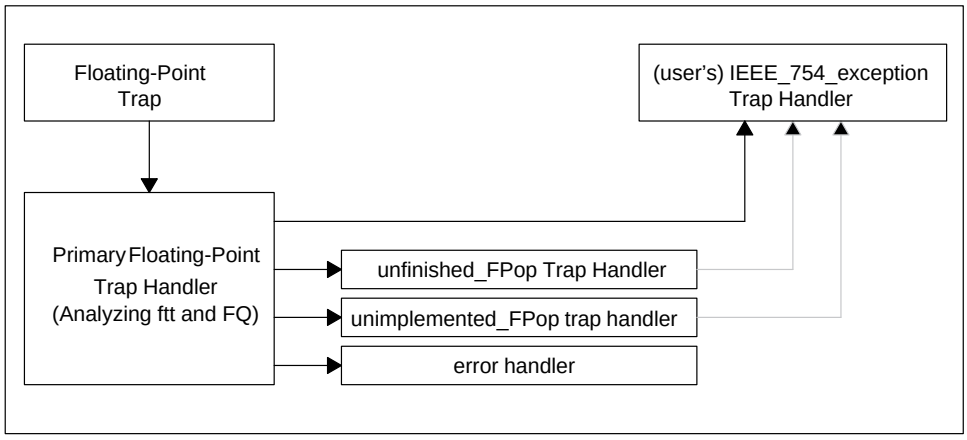


Figure E10-7. Floating-Point Trap Handling

E10.4.5 FPU States (fp_execute, fp_exception_pending, fp_exception)

The FPU is always in one of three states: the fp_execute state, the fp_exception_pending state, and the fp_exception state. These FPU states are not directly visible to software.

The FPU is in the fp_execute state following reset, and normally stays in this state. The FPU can execute FPop instructions only when the FPU is in the fp_execute state.

When an FPop generates a floating-point exception (such as IEEE_754_exception, unfinished_FPop, unimplemented_FPop, or invalid_fp_register; except sequence_error), the FPU requests the IU to service the floating-point exception trap, and moves from the fp_execute state to the fp_exception_pending state. The IU does not accept FPU's fp_exception trap request while it is executing non-floating-point instructions, but accepts the request when it attempts to execute any floating-point instruction (such as FPop, LDfp, STfp, or FBfcc). As a result, the floating-point instruction being executed in the IU is trapped. If the instruction is an FPop, it is not dispatched to the FPU. (FPops that are trapped for any reason are not dispatched to the FPU.)

Such a trap is called a deferred trap. The instruction requesting the trap is not trapped because it is not in the IU. It stays in the front entry of the floating-point deferred-trap queue (FQ) of the FPU. Another instruction (floating-point instruction) in the IU is trapped instead.

While the FPU is in the fp_exception state, only floating-point store instructions (STfp, such as STF, STDF, STFSR, or STDFQ) and non-floating-point instructions can be executed by the IU. The other floating-point instructions (FPop, LDfp, and FBfcc) cause sequence_error exceptions.

In the fp_exception state, the fp_exception trap handler uses the STFSR and STDFQ instructions to collect information from the FSR and the FQ of the FPU. The fp_exception state ensures that the handler can get the information before it is modified or changed.

The FPU moves from the fp_exception state to the fp_execute state when the FQ is emptied by the STDFQ instruction(s). In the fp_execute state, the fp_exception trap handler can use any floating-point instructions (such as FPop, LDfp, STfp and FBfcc), so that it can re-execute or emulate FPop(s) in the FQ. The FQ has one FPop instruction causing the fp-exception in the front entry, and may have other FPop instruction(s) dispatched, but not completed in the FPU.

Table E10-10 summarizes the FPU states.

Table E10-10: FPU States

State	FQ	FP Instruction	Sequence Error
fp_execute	Empty or Not	Executed	STDFQ + FQ_Empty
fp_exception_pending	Not Empty	Trapped	No Sequence Error
fp_exception	Not Empty	Only STfp is executed	FPop, LDfp, FBfcc

Programming Note:

This definition of the FPU states is for the MB86936 FPU. Other SPARC FPUs may have different definitions.

E10.4.6 Sequence_error Trap

Unlike the other fp_exception traps that are defined as deferred traps, the sequence_error trap is defined as the precise trap (i.e., the sequence_error trap is generated by a floating-point instruction that is in the IU, not in the FPU).

When the IU attempts to execute an FPop, LDfp, or FBfcc instruction in the fp_exception state, the instruction is trapped by the sequence_error trap. When the IU attempts to execute the STDFQ instruction with the FQ empty (meaning that the FPU is in the fp_execute state), the instruction is trapped by the sequence_error trap.

The sequence_error trap is the precise trap, so the FPU does not move to fp_exception_pending state. The sequence_error trap does not change the FPU state. If it occurs in the fp_exception state, the FPU stays in fp_exception state. If it occurs in the fp_execute state, the FPU stays in the fp_execute state.

When the IU attempts to execute the STFSR/STDFQ instruction in the fp_exception state, the IU can execute the instruction immediately. However, when the IU attempts to execute the STFSR/STDFQ instruction in the fp_execute state with the FQ not empty, the IU must wait for the completion of all FPops in the FQ (i.e., wait for the FQ to empty).

If all FPops in the FQ are completed without an fp_exception, the IU stops waiting and attempts to execute the STFSR/STDFQ. As a result, the STFSR is executed. Unlike the STFSR, the STDFQ is trapped, this time by the sequence_error trap because the FQ is empty.

Programming Notes:

(1) The FQ is empty when the sequence_error trap occurs in the fp_execute state, so the FPU does not generate the other fp_exception trap request following that sequence_error trap. This makes fp_exception trap handler programming simpler.

(2) This definition of the Sequence_error Trap is for the MB86936 FPU. Other SPARC FPUs may have different definitions.

E10.5 Results of FPop Instructions

When the FPop instructions are executed, most of the expected results are precisely specified in the ANSI/IEEE Standard 754-1985 specification. The MB86936 FPU conforms to the specification. However, some results are left to be defined by each implementation. The SPARC Architecture Version 8 specification has recommendations for implementations. The MB86936 FPU incorporates the recommendations, except for the Nonstandard (NS) mode recommendation.

E10.5.1 FPop Results with NaN Operands

The usage of the sign bit and bits from fraction [MSB-1] to fraction[LSB] in a NaN is implementation-dependent in the IEEE 754 specification. An implementation may hide some additional information in those bits of a NaN, and can make rules so that such information in the NaN can propagate from operand(s) to the result. The SPARC IEEE 754 Implementation Recommendation defines the rules as follows;

(1) NaN as calculated result (NaN generation)

If the result of an FPop from no NaN operand(s) is a quiet NaN (i.e., a NaN is newly generated), the sign bit must be 0, the exponent bits must be all 1's, and the fraction bits must be all 1's. The sign bit is not generated from the sign bit of operand(s). For example, (+0)/(+0) and (+0)/(-0) produce the same quiet NaN with sign bit = 0.

When the Calculated result is QNaN, the QNaN is made as follows: Calc QNaN : frd.s="0"; frd.e="11...11"; frd.f="111...111".

It is assumed in the SPARC Recommendation that all floating-point data areas in storage are initialized to all 1's (i.e., one representation of quiet NaNs). Therefore, by reading the sign bit, software can distinguish the generated quiet NaN (sign=0), from the initialized quiet NaN (sign=1).

Assumed QNaN initialization is as follows: Init QNaN : frd.s="1"; frd.e="11...11"; frd.f="111...111".

(2) NaN propagation

The SPARC Recommendation defines the rules for propagation of a NaN. A signaling NaN's priority in the propagation is higher than the priority of a quiet NaN. If both operands are quiet NaNs (or signaling NaNs), the priority of the source f register 2 (frs2) is higher than the priority of the source f register 1 (frs1).

If one operand is a NaN and another operand is a number (not a NaN), the NaN should propagate to the result without being affected by its operation. For example, in the operation frs1:QNaN * frs2:-1, the frs1:QNaN just moves to the

destination f register (frd), i.e., frd=frs1. The sign bit of the quiet NaN in the result must not be negated by -1.

(3) Signaling NaN to quiet NaN transformation

When a signaling NaN propagates to the result, the invalid exception arises (which is why the NaN is called a *signaling* NaN). If no trap occurs, the signaling NaN is transformed into a quiet NaN by setting the MSB of its fraction field (quiet bit) to 1, and is saved in the destination register without losing its hidden information.

frs2's SNaN is converted to a QNaN without losing its information as follows: frd=frs2, except frd.f[MSB]="1" (making QNaN).

(4) NaN's precision transformation

When a NaN propagates to the result in a different format (i.e., precision) from its operand (i.e., F[sdq]TO[sdq], FsMULd, or FdMULq with a NaN operand), the NaN is transformed as follows:

Converting to a narrower format: Excess low-order bits of the operand fraction are discarded (some information may be lost). The exponent field is shrunk for the narrower format. The sign bit is copied from the operand to the result without modification.

frs2's Double QNaN is converted to Single QNaN as follows: frd.s=frs2.s; frd.e="11111111"; frd.f=frs2.f[MSB:MSB-22].

Converting to a wider format: Excess low-order bits of the result fraction are set to 0's. The exponent field is expanded for the wider format. The sign bit is copied from the operand to the result without modification.

frs2's Single QNaN is converted to Double QNaN as follows: frd.s=frs2.s; frd.e="111_11111111"; frd.f={frs2.f[MSB:LSB], "000000000_0000000000_0000000000"}.

If the NaN is a signaling NaN, the precision transformation and the signal to quiet transformation occurs simultaneously.

The following tables show FPop results from NaN operand(s)

[frd] {FADD/FSUB/FMUL/FDIV/FSQRT}

-- *frs1* --	-- *frs2* -- Number	QNaN	SNaN
None	Calc(1)	frs2	frs2Q(2)
Number	Calc(1)	frs2	frs2Q(2)
QNaN	frs1	frs2	frs2Q(2)
SNaN	frs1Q(2)	frs1Q(2)	frs2Q(2)

[frd] {FsTOd/FsMULd}

-- *frs1* --	-- *frs2* -- Number	QNaN	SNaN
None	Calc(1)	frs2'(3)	frs2Q'(3)
Number	Calc(1)	frs2'(3)	frs2Q'(3)
QNaN	frs1'(3)	frs2'(3)	frs2Q'(3)
SNaN	frs1Q'(3)	frs1Q'(3)	frs2Q'(3)

[frd] {FdTOs}

-- *frs1* --	-- *frs2* -- Number	QNaN	SNaN
None	Calc	frs2''(4)	frs2Q''(4)

(1) When the Calculated result is a QNaN, the QNaN is made as follows - Calc QNaN : frd.s="0"; frd.e="11...11"; frd.f="111...111". (c.f. Init QNaN : frd.s="1"; frd.e="11...11"; frd.f="111...111".)

(2) frs1Q: frs1's SNaN is converted to a QNaN without losing its information. frd=frs1; except frd.f[MSB]="1" (making QNaN).

frs2Q: frs2's SNaN is converted to QNaN without losing its information. frd=frs2; except frd.f[MSB]="1" (making QNaN)

(3) frs1' : frs1's Single QNaN is converted to Double QNaN. frd.s=frs1.s; frd.e="111_11111111"; frd.f={frs1.f[MSB:LSB],"000000000_0000000000_0000000000"}.

frs1Q': frs1's Single SNaN is converted to Double QNaN. frd.s=frs1.s; frd.e="111_11111111"; frd.f={"1",frs1.f[MSB-1:LSB],"000000000_0000000000_0000000000"}.

frs2' : frs2's Single QNaN is converted to Double QNaN. frd.s=frs2.s; frd.e="111_11111111"; frd.f={frs2.f[MSB:LSB],"000000000_0000000000_0000000000"}.

frs2Q': frs2's Single SNaN is converted to Double QNaN. frd.s=frs2.s; frd.e="111_11111111"; frd.f={"1",frs2.f[MSB-1:LSB],"000000000_0000000000_0000000000"}.

(4) frs2'' : Double QNaN is converted to Single QNaN. frd.s=frs2.s; frd.e="11111111"; frd.f={frs2.f[MSB:MSB-22]}.

frs2Q'': Double SNaN is converted to Single QNaN. frd.s=frs2.s; frd.e="11111111"; frd.f={"1",frs2.f[MSB-1:MSB-22]}.

An IEEE 754 floating-point operation with SNaN(s) in its operand(s) causes an invalid exception. When an IEEE 754 floating-point operation generates a calculated QNaN, it causes an invalid exception (e.g. SQRT(-1), (+0)/(+0), (+Infinity)+(-Infinity)). An FCMPE instruction causes an invalid exception when its operand(s) are QNaN(s) or SNaN(s).

The following tables show invalid exception conditions.

[exc] {FADD/FSUB/FMUL/FDIV/FSQRT/F[sdq]TO[sdq]/FCMP}

	-- frs2 --		
-- frs1 --	**Number**	**QNaN**	**SNaN**
None	Calc(1)	--	nv
Number	Calc(2)	--	nv
QNaN	--	--	nv
SNaN	nv	nv	nv

[exc] {FCMPE}

	-- frs2 --		
-- frs1 --	**Number**	**QNaN**	**SNaN**
Number	--	nv	nv
QNaN	nv	nv	nv
SNaN	nv	nv	nv

(1) e.g., SQRT(-1)

(2) e.g., (+0)/(+0), (+Infinity)+(-Infinity)

NaN operands do not affect the FMOVs, FNEGs, and FABSs instructions because they are not affected by precision types (single, double, or quad) and numbers (NaN, Zero, Subnormal Number, etc.). They just transfer contents as 32-bit data between f registers. Therefore, they never cause fp_exceptions, including invalid exceptions.

The following table shows FMOVs, FNEGs, and FABSs instruction operation.

FPop	**frd[31:0]**
FMOVs	{frs2[31], frs2[30:0]}
FNEGs	{~frs2[31], frs2[30:0]}
FABSs	{0, frs2[30:0]}

E10.5.2 Overflow, Underflow, and Inexact

Overflow occurs when the rounded result of an FPop is larger in magnitude than the largest normalized number in the indicated format.

Underflow occurs in the NS mode when the rounded result of an FPop is smaller in magnitude than the smallest normalized number in the indicated format.

Inexact occurs when the final result of an FPop is not equal to the infinitely precise correct result. It happens when the rounded result differs from the infinitely precise correct result. It also happens when the rounded result overflows or underflows, and the default value is set to the final result instead of the rounded result. Even if the rounded result would be equal to the infinitely precise correct result in the broader exponential range, the overflow or underflow makes the final result inexact.

Programming Note:

> IEEE 754 specifies that the wrapped exponent results be delivered for trapped underflows and overflows. However, the SPARC architecture V8 specification states the following in the *Traps Inhibit Result* section: "The destination f register is unchanged when a floating-point trap occurs." Therefore, the MB86936 FPU does not provide this IEEE 754 feature. If software requires the feature, it must implement the feature in the software.

Overflow is handled as follows:

| largest_normalized_number | : MAXI;
| infinitely_precise_correct_result | : CORR;
| rounded_result | : ROUN;
| infinite_number | : INFI.

When MAXI < CORR < INFI:

Rounded Result	OFM=1 NXM=1	OFM=1 NXM=0	OFM=0 NXM=1	OFM=0 NXM=0	Overflow
ROUN == MAXI	nxc NX_trap	nxc nxa	nxc NX_trap	nxc nxa	NO
MAXI < ROUN < INFI	ofc OF_trap	ofc OF_trap	nxc NX_trap	nxa ofa, nxa	YES

Notes:

(1) The priority of the OFM bit is higher than the priority of the NXM bit.

(2) When the overflow trap occurs, only the ofc bit is set to identify the trap. Similarly, when the inexact trap occurs, only the nxc bit is set.

(3) When CORR < INFI, ROUN cannot be INFI.

The unrounded result is always normalized before being rounded, even if the unrounded result overflows in the given exponent range in the precision. The FPU can do this because the FPU has much wider and greater exponent range than the given exponent range, so internally the unrounded result never overflows.

If OFM=0 and NXM=0 when overflow occurs, the overflow default value is set to the final result. The default value depends on the rounding mode and the sign of the result as follows:

RD	Round Toward	+ Sign	– Sign
0	Nearest	+INFI	–INFI
1	Zero	+MAXI	–MAXI
2	+Infinity	+INFI	–MAXI
3	–Infinity	+MAXI	–INFI
Not Implemented(1)	±Infinity	+INFI	–INFI

(1) This Round Toward Infinity mode is not implemented because it is not specified in IEEE 754.

Underflow in the NS mode is handled as follows:

| smallest_normalized_number | : MINI;
| infinitely_precise_correct_result | : CORR;
| rounded_result | : ROUN;
| ZERO | : ZERO.

When ZERO < CORR < MINI:

Rounded Result	UFM=1 NXM=1	UFM=1 NXM=0	UFM=0 NXM=1	UFM=0 NXM=0	Underflow
ROUN == MINI	nxc NX_trap	nxc nxa	nxc NX_trap	nxc nxa	NO
ZERO < ROUN < MINI	ufc UF_trap	ufc UF_trap	nxc NX_trap	ufc, nxc ufa, nxa	YES

Notes:

(1) The priority of the UFM bit is higher than the priority of the NXM bit.

(2) When the underflow trap occurs, only the ufc bit is set to identify the trap. Similarly, when the inexact trap occurs, only the nxc bit is set.

(3) When ZERO < CORR, ROUN cannot be ZERO in the NS mode.

In the NS mode, the unrounded result is always normalized before being rounded, even if the unrounded result underflows in the given exponent range in the precision. The FPU can do this because the FPU has much wider and greater exponent range than the given exponent range, so internally the unrounded result never underflows.

> In the IEEE 754 Underflow specification (the Gradual Underflow), the unrounded result is not normalized for rounding if the unrounded result underflows in the given exponent range and precision. Instead, the exponent of the unrounded result is adjusted to the minimum, and the fraction of the unrounded result is shifted as much as the adjustment made in the exponent. (This is why the number is called a subnormal or denormalized number.) Then, the denormalized unrounded result is rounded to realize the Gradual Underflow.

If UFM=0 and NXM=0 when underflow occurs, the underflow default value is set to the final result. The default value depends on rounding mode and the sign of the result as follows:

RD	Round Toward	+ Sign	– Sign
0	Nearest	+ZERO	–ZERO
1	Zero	+ZERO	–ZERO
2	+Infinity	+MINI	–ZERO
3	–Infinity	+ZERO	–MINI
Not Implemented(1)	±Infinity	+MINI	–MINI

(1) This Round Toward Infinity mode is not implemented because it is not specified in IEEE 754.

E10.5.3 Integer Results

The FsTOi, FdTOi, and FqTOi instructions generate integer results. Unlike a floating-point overflow raising the overflow (of) and inexact (nx) exceptions, an integer overflow raises just the invalid (nv) exception. Unlike a floating-point result rounded based on the RD field, an integer result is always rounded toward zero (i.e., the RD is ignored). Furthermore, an integer result never underflows. The rounded result just becomes 0 if -1 < unrounded_result < 1.

If the source register contains a NaN, ±infinity, positive number >= +2147483648.0 (i.e., overflow at + side), or negative number <= -2147483649.0 (i.e., overflow at – side), the invalid (nv) exception arises. If no trap occurs and the sign bit of the operand is positive (frs2.MSB=0), the FPU outputs the positive default integer result, +2147483647 (i.e., 0x7fffffff). If no trap occurs and the sign bit of the operand is negative (frs2.MSB=1), the FPU outputs the negative default integer result, –2147483648 (i.e., 0x80000000).

Programming Notes:

(1) Even if +2147483648.0 > operand > +2147483647.0 (such as +2147483647.99...), or -2147483649.0 < operand < -2147483648.0 (such as -2147483648.99...), the result does not overflow because the result is rounded toward zero before the overflow detection.

(2) According to IEEE754's recommendation, a NaN does not have the concept of polarity. The sign bit does not mean more than the MSB of data in the NaN. However, if the operand is the NaN, the F[sdq]TOi instruction always generates the + or - integer result based on the sign bit of the operand. There is discrepancy in the recommendation.

For example, (+0.0)/(+0.0) and (+0.0)/(-0.0) result in the same quiet NaN with sign 0. Therefore,

F[sdq]TOi((+0.0)/(+0.0)) = +2147483647 (i.e., 0x7fffffff),
F[sdq]TOi((+0.0)/(-0.0)) = +2147483647 (i.e., 0x7fffffff).

(On the other hand, if a NaN has polarity, the question arises - whether (+Infinity)+(-Infinity) must be +NaN or -NaN.)

The following tables show the destination f register values (frd) and the IEEE 754 exceptions for the FsTOi and FdTOi instructions.

[frd] (destination f register)

ZERO	NORM	INFI	QNaN	SNaN
INT_ZERO	Calc(2)	INT_MAX(1) INT_MINI(1)	INT_MAX(1) INT_MINI(1)	INT_MAX(1) INT_MINI(1)

[exc] (floating-point exception)

ZERO	NORM	INFI	QNaN	SNaN
--	Calc(2)	nv	nv	nv

(1) if frs2.s=0 then frd=0x7fffffff (INT_MAXI) {even NaN} if frs2.s=1 then frd=0x80000000 (INT_MINI) {even NaN}

(2) calc may be INT with no IEEE exception; INT with nx (inexact: rounded always toward zero); INT_MAXI with nv (invalid: overflows at + side); INT_MINI with nv (invalid: overflows at - side).

When the nx and the nv occur at the same time (true only when FdTOi), the nx is ignored, and only the nv arises. For example,

If: frs2 = (+) Fra:1.f_ffff_ffff_ffff * Exp:36
Then: Int (frs2) = 1f_ffff_ffff.ffff
overflow (nv) rounded (nx)

As a result: frd=0x7fff_ffff with nv.

E10.5.4 Emulation for Subnormal Number, Invoked by the Unfinished_FPop Trap

When the NS bit is 0, if the source f register(s) of an FPop contain(s) a subnormal (i.e., denormalized) number(s), the FPop is trapped by the unfinished_FPop trap in the front entry of the FQ. The FPop is also trapped by the unfinished_FPop trap in the front entry of the FQ if the correct unrounded result (i.e., the infinitely precise correct result) of the FPop is smaller in magnitude than the smallest normalized number in the indicated format (i.e., if (ZERO < CORR < MINI) && (NS==0)→ unfinished_FPop trap).

In both cases, software should emulate the trapped FPop and update the destination f register(s) and the fcc, cexc, and aexc fields in FSR to conform to ANSI/IEEE Standard 754-1985.

Programming Note:

The emulator of an FPop_with_subnormal_number must conform to the SPARC IEEE 754 Implementation Recommendation in the SPARC Architecture Version 8 specification with regard to underflow as follows;

| smallest_normalized_number | : MINI;

| infinitely_precise_correct_result | : CORR;

| rounded_result | : ROUN;

| ZERO | : ZERO.

When ZERO < CORR < MINI:

Result Rounded for Subnormal Number	UFM=1 NXM=*	UFM=0 NXM=1	UFM=0 NXM=0	Underflow		Subnormal Result
				Trap	Flag	
ROUN == MINI (so ROUN != CORR)	ufc UF_trap	nxc NX_trap	ufc, nxc ufa, nxa	Yes	Yes	NO
ZERO < ROUN < MINI && ROUN ! = CORR	ufc UF_trap	nxc NX_trap	ufc, nxc ufa, nxa	Yes	Yes	YES
ZERO < ROUN < MINI && ROUN = CORR	ufc UF_trap	None	None	Yes	No	YES
ROUN == ZERO (so ROUN != CORR)	ufc UF_trap	nxc NX_trap	ufc, nxc ufa, nxa	Yes	Yes	NO

Notes:

(1) "Tininess detected before rounding" is true when ZERO < CORR < MINI. "Loss_of_accuracy detected as inexact" is true when ROUN != CORR. The underflow trap occurs if UFM==1 and Tininess. The ufa bit is set if UFM=0, MXM=0, Tininess, and Loss_of_accuracy.

(2) Even if ZERO < CORR, ROUN can be ZERO in the IEEE 754 Underflow mode. In the IEEE 754 underflow, the unrounded result is converted to subnormal (denormalized) number if the unrounded result underflows in the given exponent range and precision. Then, the denormalized unrounded result is rounded, so the rounded result can be zero.

Refer to the SPARC Architecture Manual (Version 8) and the ANSI/IEEE Standard 754-1985 for details.

E10.5.5 Emulation for Quad-precision operation, Invoked by the Unimplemented_FPop Trap

When any quad-precision FPop (including FqTO[isd], F[isd]TOq, and FdMULq) is dispatched to the FPU, the FPop is trapped by the unimplemented_FPop trap in the front entry of the FQ. Software should emulate the trapped FPop and update the destination f registers and the fcc, cexc, and aexc fields in FSR to conform to ANSI/IEEE Standard 754-1985.

Programming Note:

The priority of the unimplemented_FPop trap is higher than the priority of the unfinished_FPop Trap, but is lower than the priority of the invalid_fp_register trap in the MB86936 implementation. Table E10-11 shows the MB86936 trap priorities.

Table E10-11: Trap Priorities

Priority	Trap Type	ftt	
1	sequence_error	4	Not Dispatched→ Error
2	invalid_fp_register	6	Dispatched → Bad alignment → Error
3	unimplemented_FPop	3	Dispatched → Quad Precision → Emulation
4	unfinished_FPop	2	Dispatched → Subnormal Num.→ Emulation
5	IEEE_754_exception	1	Dispatched → Executed → IEEE Exception

Therefore, the emulator of a quad-precision FPop does not have to check the invalid_fp_register (bad alignment), but must be able to handle a subnormal number, and must be able to handle the IEEE_754_exception.

The emulator of an FPop_with_subnormal_number does not have to check the invalid_fp_register or the quad_precision, but must be able to handle the IEEE_754_exception.

Refer to the SPARC Architecture Manual (Version 8) and the ANSI/IEEE Standard 754-1985 for details.

E10.5.6 Result of FPop Instruction without NaN(s)/DNRM(s) in Operand(s)

The following tables show the results of the IEEE 754 floating-point operations when operands do not contain NaNs or subnormal (denormalized) numbers (DNRM). [frd] is the table of the destination f register (frd) value. [exc] is the table of the IEEE 754 exceptions.

In the tables,"calc" is a result that depends on the calculated value. The "calc" in the exception table can be either the "of", "uf", "nx", "of & nx", or "uf & nx" exception, or nothing.

FADD/FSUB Without NaN/DNRM Operands

If FADD, let FRS2=+frs2; then, frs1+frs2=frs1+FRS2.
If FSUB, let FRS2=-frs2; then, frs1-frs2=frs1+FRS2

[frd]

	-- FRS2 --		
-- frs1 --	**±ZERO**	**±NORM**	**±INFI**
±ZERO	±ZERO(3)	±NORM(1)	±INFI(1)
±NORM	±NORM(2)	±calc(6)	±INFI(1)
±INFI	±INFI(2)	±INFI(2)	±INFI(4), QNaN(5)

[exc]

	-- FRS2--		
-- frs1 --	**±ZERO**	**±NORM**	**±INFI**
±ZERO	--	--	--
±NORM	--	calc	--
±INFI	--	--	nv(5)

(1) [+] if FRS2.s=0; [–] if FRS2.s=1.

(2) [+] if frs1.s=0; [–] if frs1.s=1.

(3) +ZERO if +ZERO+ZERO or RD!=R– and (ZERO–ZERO or –ZERO+ZERO).
–ZERO if –ZERO–ZERO or RD==R– and (ZERO–ZERO or –ZERO+ZERO).
{RD==R– means rounding toward minus infinity.}

(4) +INFI if +INFI+INFI; –INFI if –INFI–INFI.

(5) QNaN if +INFI–INFI or –INFI+INFI. (Also, nv if so.)

(6) +ZERO if RD!=R– and (NORM–NORM=ZERO or –NORM+NORM=ZERO).
–ZERO if RD==R– and (NORM–NORM=ZERO or –NORM+NORM=ZERO).

FMUL Without NaN/DNRM Operands

[frd]

-- frs1 --	*-- frs2 --* ±**ZERO**	±**NORM**	±**INFI**
±**ZERO**	±ZERO(1)	±ZERO(1)	QNaN
±**NORM**	±ZERO(1)	±calc(1)	±INFI(1)
±**INFI**	QNaN	±INFI(1)	±INFI(1)

[exc]

-- frs1 --	*-- frs2 --* ±**ZERO**	±**NORM**	±**INFI**
±**ZERO**	--	--	nv
±**NORM**	--	calc	--
±**INFI**	nv	--	--

(1) (+)=(+)*(+); (+)=(–)*(–); (–)=(–)*(+); (–)=(+)*(–).

FDIV Without NaN/DNRM Operands

[frd]

-- frs1 --	*-- frs2 --* ±**ZERO**	±**NORM**	±**INFI**
±**ZERO**	QNaN	±ZERO(1)	±ZERO(1)
±**NORM**	±INFI(1)	±calc(1)	±ZERO(1)
±**INFI**	±INFI(1)	±INFI(1)	QNaN

[exc]

-- frs1 --	*-- frs2 --* ±**ZERO**	±**NORM**	±**INFI**
±**ZERO**	nv	--	--
±**NORM**	dz	calc	--
±**INFI**	--	--	nv

(1) (+)=(+)/(+); (+)=(–)/(–); (–)=(–)/(+); (–)=(+)/(–).

FSQRT Without NaN/DNRM Operand

[frd]

-- frs2 --					
+ZERO	**–ZERO**	**+NORM**	**–NORM**	**+INFI**	**–INFI**
+ZERO	–ZERO(1)	+calc	QNaN	+INFI	QNaN

[exc]

-- frs2 --					
+ZERO	**–ZERO**	**+NORM**	**–NORM**	**+INFI**	**–INFI**
--	--	calc	nv	--	nv

(1) SQRT(–0) is –0.

FsTOd/FdTOs Without NaN/DNRM Operand

[frd]

-- frs2 --					
+ZERO	**–ZERO**	**+NORM**	**–NORM**	**+INFI**	**–INFI**
+ZERO	–ZERO	+calc	– calc	+INFI	–INFI

[exc]

-- frs2 --					
+ZERO	**–ZERO**	**+NORM**	**–NORM**	**+INFI**	**–INFI**
--	--	calc	calc	--	--

FiTOs/FiTOd Without NaN/DNRM Operand

[frd]

-- frs2 --		
INT_ZERO	**+INT**	**–INT**
+zero(1)	+calc(1)	–calc(1)

[exc]

-- frs2 --		
INT_ZERO	**+INT**	**–INT**
--	calc(2)	calc(2)

(1) The result must be a normalized number or +zero, not be a NaN, infinity, a subnormal number, nor –zero.

(2) FiTOs may have "nx" exception because integer:31bits > single_f:24bits. FiTOd has no exception because integer:31bits < double_f:53bits.

The fcc Field Updated by FCMP/FCMPE

-- frs1 --	*-- frs2 --*					
	–INFI	**–NORM**	**±ZERO**	**+NORM**	**+INFI**	**QNaN/ SNaN**
–INFI	0:=	1:<	1:<	1:<	1:<	3:?
–NORM	2:>	calc(2)	1:<	1:<	1:<	3:?
±ZERO	2:>	2:>	0:=(1)	1:<	1:<	3:?
+NORM	2:>	2:>	2:>	calc(2)	1:<	3:?
+INFI	2:>	2:>	2:>	2:>	0:=	3:?
QNaN/SNaN	3:?	3:?	3:?	3:?	3:?	3:?

(1) (+0) is equal to (–0); (–0) is equal to (+0).

(2) If frs1=frs2, fcc=0; if frs1<frs2, fcc=1; if frs1>frs2, fcc=2; never fcc=3.

E10.6 Pipeline of FPU and Latency

The SPARC FPU pipeline structure and interlock mechanism is implementation dependent, and therefore differs with each SPARC FPU. The FPUs maintain enough in common to allow code that is generated by a “generic” SPARC compiler to run efficiently in the MB86936 FPU, but the highest performance is realized with code written specifically for the MB86936 FPU.

This section describes the MB86936 pipeline structure and interlock conditions. It is intended for the software engineer whose goal is to write software (such as an MB86936-specific compiler or key routines or libraries in assembly language) that has the highest-performance possible.

The MB86936 IU pipeline and FPU pipeline are complicated and cannot be fully described in this document. The following descriptions therefore focus on the key FPU design factors.

E10.6.1 FPU Pipeline

The SPARClite FPU consists of 4 pipeline stages: the A_stage, the B_stage, the C_stage, and the D_stage. They are also called the FPU_A, FPU_B, FPU_C, and FPU_D stages, respectively.

The SPARClite IU has 5 stages: the Fetch stage, the Decode stage, the Execution stage, the Memory stage, the Write-back stage. They are also called the IU_F, IU_D, IU_E, IU_M, and IU_W stages, respectively.

If an FPop instruction does not have a trap request by the Execution stage, the IU dispatches the FPop instruction to the FPU. If the FPop instruction does have a trap request by the Execution stage, the IU does not dispatch the FPop instruction. The FPop with the trap request moves to the Memory stage, where it is eventually trapped.

The FPop instruction flows through the IU and FPU pipelines as follows:

(1) When dispatched: IU_F → IU_D → IU_E → FPU_A → FPU_B → FPU_C → FPU_D

(2) When trapped: IU_F → IU_D → IU_E → IU_M → IU_W

The IU reads the FPop operand(s) from corresponding f register(s) in the Execution stage. The operand(s) are available for the FPU in the A_stage.

The A_stage, B_stage, and C_stage are the execution stages of the FPU. Unlike the IU, the FPU requires 3 stages for execution because each floating-point operation requires completion of many tasks. For example, the floating-point add operation requires swapping, adjusting (shifting), adding/subtracting, normalizing (shifting), and rounding of the fraction part of the floating-point number. It also requires handling of the exponential part and the sign part of the floating-point number.

When an FPop is in the C_stage and there is an fp_exception, the FPU asserts the fp_exception trap request to the IU. The FPU keeps asserting the request until it is accepted by the IU.

The D_stage is the write back stage of the FPU. When an FPop is in the D_stage and there is no fp_exception, the FPU updates the FSR and writes the result into the f register. If there is an fp_exception when the FPop is in the D_stage, the FPU updates the FSR but does not write the result into the f register.

The FPU pipeline can be summarized as follows:

FPU_A	→	FPU_B	→	FPU_C	→	FPU_D
(Execution 1)		(Execution 2)		(Execution 3)		(Write-back)

When the IU dispatches an FPop, the FPop moves into the FPU pipeline and into the FQ. When the FPop moves out of the FPU pipeline, it also moves out of the FQ. An FPop is in the FQ while it is in the FPU pipeline. However, the entries of the FQ (such as front, 2nd, 3rd) do not correspond to the stages of the FPU pipelines (such as FPU_A, FPU_B, FPU_C, FPU_D).

Programming Note:

There are two traps which may be detected (i.e., requested) in the Memory stage. One is the data_access_exception trap from the BIU. The other is the data_break_point trap from the DSU. If an FPop is dispatched to the FPU, both trap requests are ignored. (Note: The FPop does not access the memory, but the Write Buffer in the BIU may generate the data_access_exception trap request for the FPop in the Memory stage.)

Both requests can be ignored because the BIU (Write Buffer) and DSU keep asserting the trap requests to the IU until the requests are accepted. Both traps can be considered asynchronous traps.

E10.6.2 FPop Throughput and Latency

Each FPop except FMULd, FDIVs, FSQRTs, FDIVd, and FSQRTd, stays only 1 cycle at each stage of the FPU. Therefore, the throughput of these FPops is 1 cycle, and their latency with respect to the following FPop is 3 cycles. When there is a data dependency between an FPop at FPU_D and another FPop at FPU_A (i.e., FPU_D.frd == FPU_A.frs1 or FPU_D.frd == FPU_A.frs2), the result in the D_stage is bypassed to the operand(s) in the A_stage.

Table E10-12 shows floating-point instruction throughput and latency.

Table E10-12: Floating-Point Instruction Throughput and Latency

Instruction	Throughput	Latency
FDIVs/FSQRTs	13	14
FMULs/FsMULd	1	3
FADDs/FSUBs	1	3
All Other FPops_s	1	3
FDIVd/FSQRTd	28	29
FMULd	4	6
FADDd/FSUBd	1	3
All Other FPop_d	1	3

Programming Note:

> The IU and the FPU check the data dependency every time and assert the Data Hazard interlock if necessary; then, there is no data dependency between an FPop at FPU_D and another FPop at FPU_B; also, between an FPop at FPU_D and another FPop at FPU_C. A program does not need to ensure those conditions.

The FMULd instruction stays in the A_stage for 4 cycles; so its throughput is 4 cycles, and its latency is 6 cycles.

The FDIVs/FSQRTs instructions stay in the A_stage for 13 cycles, but skip over the B_stage; so their throughput is 13 cycles, and their latency is 14 cycles.

The FDIVd/FSQRTd instructions stay in the A_stage for 28 cycles, but skip over the B_stage; so their throughput is 28 cycles, and their latency is 29 cycles.

When an f register is written with a value that is then read, the written value and the read value are always the same; so the bypass technique works for the f register.

When there is a data dependency between an FPop at FPU_D and an FPop at IU_E (i.e., FPU_D.frd == IU_E.frs1 or FPU_D.frd == IU_E.frs2), the result in the D_stage can reach the operand(s) in the Execution stage by passing through the f register designated by frd - from the register's input port to its output port in the same cycle.

E10.6.3 IU Interlocks, IU Holds, FPU Interlocks, and FPU Hold

The IU has the IU holds and the IU interlocks. The FPU has the FPU hold and the FPU interlocks. Some IU interlocks are generated for the IU, and some are generated for the FPU. All FPU interlocks are generated for the FPU.

All IU holds and IU interlocks stop the IU pipeline but do not stop the FPU pipeline. All FPU interlocks and the FPU hold stop the FPU pipeline, but do not stop the IU pipeline directly.

The IU holds stop the entire IU pipeline and are usually generated by peripheral units (such as BIU) to "hold" the IU momentarily. Unlike the IU holds, the IU interlocks stop the IU pipeline partially. They are generated by the processor units (i.e., IU/FPU), and are used to stall instructions in the IU pipeline.

The FPU hold stops the entire FPU pipeline and is generated when the FPU is in the fp_exception_pending state or in the fp_exception state. The FPU hold is not asserted while the FPU is in the fp_execute state. Unlike the FPU hold, the FPU interlocks stop the FPU pipeline partially to stall instructions in the FPU pipeline.

The FPU interlocks are generated for the FMULd, FDIVs/FSQRTs, and FDIVd/FSQRTd instructions, which must stay in the A_stage of the FPU for several cycles to complete their executions. These interlocks are called the FMULd interlock, the FDIVs_FSQRTs interlock, and the FDIVd_FSQRTd interlock.

The IU interlocks are more complicated and varied than the FPU interlocks. Only the IU interlocks generated for FPU instructions are described in this section.

E10.6.4 FPU_full Interlock

One of the IU interlocks generated for the FPU is called the FPU_full interlock. If a FMULd, FDIVs/FSQRTs, or FDIVd/FSQRTd instruction occupies the A_stage of the FPU, the following FPop must wait in the Execution stage of the IU. To do so, the FPU_full interlock is asserted for no more than 3 cycles for a FMULd instruction, 12 cycles for a FDIVs/FSQRTs instruction, and 27 cycles for a FDIVd/FSQRTd instruction.

Some non-FPop instructions can be positioned between the FP0ps, the preceding FPop and the following FPop without changing the FPU execution time. There are at least 27 cycles between the FDIVd/FSQRTd and the following FPop, but this does not mean that 27 instructions can be positioned between them because if the IU pipeline is stopped by an IU hold, one instruction can require more than one cycle to complete. (e.g. If 3 waits are needed for one memory access, one load instruction has 3 held cycles; therefore, the load instruction requires total 4 cycles to execute.)

Like the FPU_full interlock, most IU interlocks generated for the FPU stall instructions in the Execution stage. When such interlocks are asserted, the Fetch stage, the Decode stage, and the Execution stage are stopped; but the Memory stage and the Write-back stage are not stopped.

Figure E10-8 shows an example of the FPU_full interlock.

E10.6.5 Data Hazard Interlocks

(1) An FPop following another FPop causes the RAW (Read After Write) Data Hazard interlock if there is a RAW dependency between the two FPops. The following FPop is stalled in the IU_E stage while the preceding FPop is in the FPU_A or FPU_B stage. When the preceding FPop moves to the FPU_C stage, the interlock is negated. When the preceding FPop moves to the FPU_D stage, the result is bypassed to the FPU_A stage if the following FPop is in the FPU_A.

    For example, if “FSUBs %f2,%f3,%f4” follows “FADDs %f0,%f1,%f2” without any instruction between them, the FSUBs instruction is stalled for 2 cycles by the RAW Data Hazard interlock in the IU_E stage because there is a RAW dependency in the %f2 register. The FSUBs instruction “reads” the %f2 register “after” the FADDs instruction “writes” the %f2 register.

(2) In the same way, a STF/STDF instruction that follows an FPop may have the RAW Data Hazard interlock. The STF/STDF instruction is stalled in the IU_E stage while the preceding FPop is in the FPU_A or FPU_B stage. When the FPop moves to the FPU_D stage, the result is bypassed to the IU_M stage if the STF/STDF instruction is in the IU_M stage.

1.		2.		3.	
IU_F: FSUBd		IU_F: FCMPd		IU_F:	
IU_D: FMULd		IU_D: FSUBd		IU_D: FCMPd	
IU_E: FADDd		IU_E: FMULd		IU_E: FSUBd	
IU_M:	FPU_A:	IU_M:	FPU_A: FADDd	IU_M:	FPU_A: FMULd
IU_W:	FPU_B:	IU_W:	FPU_B:	IU_W:	FPU_B: FADDd
	FPU_C:		FPU_C:		FPU_C:
	FPU_D:		FPU_D:		FPU_D:
4.		5.		6.	
IU_F:		IU_F:		IU_F:	
IU_D: FCMPd		IU_D: FCMPd		IU_D: FCMPd	
U_E: FSUBd		IU_E: FSUBd		IU_E: FSUBd	
IU_M:	FPU_A: FMULd	IU_M:	FPU_A: FMULd	IU_M:	FPU_A: FMULd
IU_W:	FPU_B:	IU_W:	FPU_B:	IU_W:	FPU_B:
	FPU_C:FADDd		FPU_C:		FPU_C:
	FPU_D:		FPU_D: FADDd		FPU_D:
<< Interlock >>		**<< Interlock >>**		**<< Interlock >>**	
7.		8.		9.	
IU_F:		IU_F:		IU_F:	
IU_D:		|U_D:		IU_D:	
IU_E: FCMPd		IU_E:		IU_E:	
IU_M:	FPU_A: FSUBd	IU_M:	FPU_A: FCMPd	IU_M:	FPU_A:
IU_W:	FPU_B: FMULd	IU_W:	FPU_B: FSUBd	IU_W:	FPU_B:FCMPd
	FPU_C:		FPU_C: FMULd		FPU_C: FSUBd
	FPU_D:		FPU_D:		FPU_D: FMULd

Figure E10-8. FPU_full Interlock Example

For example, if "ST %f2,[0]" follows "FADDs %f0,%f1,%f2" without any instruction between them, the STF instruction is stalled for 2 cycles. The STF instruction "reads" the %f2 register "after" the FADDs instruction "writes" the %f2 register.

(3) An FPop following an LDF/LDDF instruction causes the RAW Data Hazard interlock if there is a RAW dependency between them. The FPop is stalled in the IU_E stage while the LDF/LDDF is in the IU_M stage. When the LDF/LDDF instruction moves to the IU_W stage, the interlock is negated, and the loaded data moves from the IU_W stage to the IU_E stage via the designated f register.

For example, if "FSUBs %f2,%f3,%f4" follows "LD [4],%f2" without any instruction between them, the FSUBs instruction is stalled for at least 1 cycle (depending on the IU hold conditions) by the RAW Data Hazard interlock in

the IU_E stage because there is a RAW dependency in the %f2 register. The FPop “reads” the %f2 register “after” the LDF instruction “writes” the %f2 register.

(4) In the same way, an STF/STDF instruction that follows an LDF/LDDF instruction may cause the RAW Data Hazard interlock. The STF/STDF instruction is stalled in the IU_E stage while the LDF/LDDF instruction is in the FPU_M stage. When the LDF/LDDF instruction moves to the IU_W stage, the interlock is negated, and the loaded data moves from the IU_W stage to the IU_E stage via the designated f register.

For example, if “ST %f2,[4]” follows “LD [0],%f2” without any instruction between them, the STF instruction is stalled for at least 1 cycle (depending on the hold condition.). The STF instruction “reads” the %f2 register “after” the LDF instruction “writes” the %f2 register.

(5) An LDF/LDDF instruction that follows an FPop causes the WAW (Write After Write) Data Hazard interlock if there is a WAW dependency between them. The LDF/LDDF instruction is stalled in the IU_E stage while the FPop is in the FPU_A or FPU_B stage. When the FPop moves to the FPU_C stage, the interlock is negated. As a result, before the LDF/LDDF moves to the IU_W stage and writes the loaded data to the f register, the FPop moves to the FPU_D stage and writes the result to the designated f register.

For example, if “LD [0],%f2” follows “FADDs %f0,%f1,%f2” without any instruction between them, the LDF instruction is stalled for 2 cycles by the WAW Data Hazard interlock in the IU_E stage because there is a WAW dependency in the %f2 register. The LDF instruction “writes” the %f2 register “after” the FADDs instruction “writes” the %f2 register.

(6) An LDF/LDDF instruction that follows an FPop has the WAR (Write After Read) Data Hazard interlock if there is the WAR dependency between them. At that time the LDF/LDDF instruction must wait for the completion of the FPop because if the FPop is trapped by the fp_exception trap, its trap handler may have to read the source register(s) of the FPop. The LDF/LDDF instruction should not write the loaded data to the register before this happens.

The LDF/LDDF instruction is stalled in the IU_E stage while the FPop is in the FPU_A or FPU_B stage. When the FPop moves to the FPU_C stage, the interlock is negated. At that time, if the FPop in the FPU_C stage has an fp_exception trap request, the LDF/LDDF instruction in the IU_E stage is annulled; so the source register of the FPop is not updated by the LDF/LDDF instruction. When the LDF/LDDF instruction moves to the IU_M stage, the LDF/LDDF instruction is trapped for the fp_exception.

For example, if “LD [0],%f1” follows “FADDs %f0,%f1,%f2” without any instruction between them, the LDF instruction is stalled for 2 cycles by the WAR Data Hazard interlock in the IU_E stage because there is a WAR

dependency in the %f1 register. The LDF instruction "writes" the %f1 register "after" the FADDs instruction "reads" the %f1 register, and after the FADDs instruction has completed without any fp_exceptions.

E10.6.6 STFSR_LDFSR_STDFQ Interlock and FPop_Quad Interlock

Two STFSR_LDFSR_STDFQ interlocks are generated to ensure proper STFSR/LDFSR/STDFQ instruction execution. The first interlock stalls the STFSR/LDFSR/STDFQ in the IU_E stage while any FPop instruction is in the FQ. The other interlock stalls the following instruction in the IU_D stage while the STFSR/LDFSR/STDFQ instruction is in the IU_E or the IU_M stage. The interlock is negated when the STFSR/LDFSR/STDFQ instruction is in the IU_W stage.

The FPop_Quad interlock stalls an FPop/LDF/LDDF/STF/STDF instruction in the IU_E stage while any Quad precision FPop(s) is in the FPU_A, FPU_B, or FPU_C stage.

Programming Notes:

(1) The STFSR_LDFSR_STDFQ interlock and the FPop_Quad interlock are not generated frequently, so they have very little affect on performance. A programmer can ignore these two interlocks.

(2) When the following FPop is a Quad precision FPop, the RAW Data Hazard interlock may not be generated even if there is the RAW dependency. It should not cause any problem because the Quad precision FPop is eventually trapped.

E10.6.7 Latency of FCMP to FBfcc and FCMP_FBfcc Interlock

The latency of the FCMP to the FBfcc is 3 cycles in the best case.

Although the fcc field of the FSR is updated in the D_stage, the following FBfcc does not wait for the update. The FBfcc uses the “hot” fcc to make the branch decision instead of using the fcc in the FSR. The “hot” fcc is updated when an FCMP instruction moves to the B_stage of the FPU (i.e., 2 cycles earlier than FSR’s update).

The IU makes the FBfcc branch decision during its Decode stage and then waits there. The FCMP moves from Execution stage of the IU to the A_stage of the FPU in 1 cycle if there is no IU interlock or IU hold condition. After being dispatched to the FPU, the FCMP moves from the A_stage to the B_stage (updating the “hot” fcc) in 1 cycle. There are at least 2 cycles between the FCMP instruction and the FBfcc instruction, so the latency of the FCMP to the FBfcc is therefore 3 cycles in the best case.

If there is no IU interlock or IU hold, a program can have at most two instructions (FPop or non-FPop, except another FCMP) between the FCMP instruction and the FBfcc instruction without changing branch timing. When a program has one instruction or no instruction between the FCMP and the FBfcc (note: other SPARC FPUs may not allow “no” instruction), the FCMP_FBfcc interlock may be asserted to stall the FBfcc in the Decode stages as long as necessary.

Programming Note:

> When an FCMP instruction has an fp_exception, the “hot” fcc has an unknown value, and the following FBfcc instruction may therefore branch to the wrong location. This causes no problem, however, because the FBfcc instruction is trapped by the fp_exception trap that was generated by the FCMP instruction.

E10.6.8 Latencies of Interrupt, Trap, and Task Switch

Although an interlock may stop an FPop for many cycles (e.g., 27 cycles for FDIVd/FSQRTd) in the Execution stage:

(1) The interrupt latency is not increased by the interlock.

An interrupt request is detected in the Execution stage. Once an interrupt is detected, the interlock for an FPop is annulled, and the FPop is not dispatched to the FPU. The FPop then moves to the Memory stage with the interrupt trap request, and it is trapped there.

(2) The trap latency is not increased by the interlock.

When an instruction is trapped in the Memory stage, following instructions in the Fetch, Decode, and Execution stages are squashed. Even if an FPop is interlocked at the Execution stage, it is squashed by the trap, and the interlock is annulled. Therefore, an interlock at Execution (or Fetch or Decode) stage does not increase the latency of a trap.

(3) The task switch latency is minimized by the FPU_full interlock.

When the OS switches tasks, the OS must wait for the completion of all FPops in the FQ if both tasks use the FPU. If there is no FPU_full interlock, the three-entry FQ can have at most 3 FDIVd/FSQRTd instructions. Then, in the worst case, the OS must wait for the completion of 3 FDIVd/FSQRTd instructions for about 84 (28∗3) cycles. However, with the FPU_full interlock, the OS must wait only about 28 cycles for the completion of all FPops in the FQ, even in the worst case.

CHAPTER

E11

Floating-Point Instructions

This chapter describes all floating-point instructions that the MB86936 supports in hardware.

E11.1 Floating-point Operate (FPop) Instructions

opcode	op3	operation
FPop1	110100	Floating-point operate
FPop2	110101	Floating-point operate

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	rs1	opf	rs2
FPop2	10	rd	110101	rs1	opf	rs2

Description:

The Floating-point operate (FPop) instructions are encoded using two "type 3" formats: FPop1, FPop2. The particular floating-point instruction is determined by the instruction opf field. Note that the load/store floating-point instructions are not FPop instructions.

The FPop1 and instructions do not affect the floating-point condition codes, but the FPop2 and instructions may affect the floating-point condition codes.

The FPop instructions support operations between integer words and single-, double-, and quad-precision floating-point operands in f register(s). All FPop instructions operate according to ANSI/IEEE Standard. 754-1985 on single, double, and quad formats.

The least significant bit of an f register address is not used by double-precision FPop instructions, and the least significant 2 bits of an f register address are not used by quad-precision FPop instructions. These unused register address bits are reserved and should be written 0 by software to ensure future compatibility. If these address bit(s) are not 0 in an FPop instruction with a double- or quad-precision operand, an fp_exception trap occurs with FSR.ftt = invalid_fp_register.

If either the EF field of the PSR is 0 or no FPU is present, an FPop1 or FPop2 instruction causes an fp_disabled trap.

Floating-point exceptions may cause either precise or deferred traps.

Programming Note:

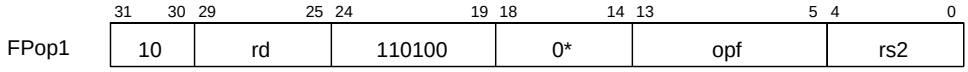
The following restriction does not apply to the MB86936, but may apply to other SPARC processors.

If an FPop2 instruction such as FCMP or FCMPE sets the floating-point condition codes, then at least one non-FPop2 (non-floating-point operate 2) instruction must be executed between the FPop2 instruction and a following FBfcc instruction. Otherwise, the result of the FBfcc instruction is undefined.

E11.1.1 Convert Integer to Floating-Point Instructions

opcode	opf	operation
FiTOs	011000100	Convert Integer to Single
FiTOd	011001000	Convert Integer to Double
FiTOq	011001100	Convert Integer to Quad

Format:



* Not used, must be 0.

Syntax:

fitos *fregrs2 , fregrd*
fitod *fregrs2 , fregrd*
fitoq *fregrs2 , fregrd*

Description:

These instructions convert the 32-bit integer word operand in f[rs2] into a floating-point number in the destination format. They write the result into the f register(s) specified by rd.

FiTOs rounds according to the RD field in the FSR.

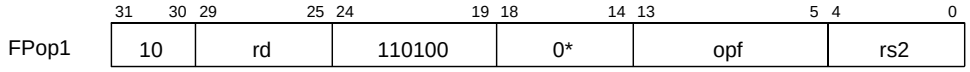
Traps:

fp_disabled
fp_exception (NX (FiTOs), Invalid_fp_register (FiTOd, FiTOq))

E11.1.2 Convert Floating-Point to Integer Instructions

opcode	opf	operation
FsTOi	011010001	Convert Single to Integer
FdTOi	011010010	Convert Double to Integer
FqTOi	011010011	Convert Quad to Integer

Format:



* Not used, must be 0.

Syntax:

fstoi *fregrs2 , fregrd*
fdtoi *fregrs2 , fregrd*
fqtoi *fregrs2 , fregrd*

Description:

These instructions convert the floating-point operand in the f register(s) specified by rs2 into a 32-bit integer word in f[rd]. The result is always rounded toward 0 (the RD field in the FSR is ignored).

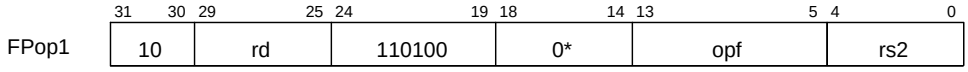
Traps:

fp_disabled
fp_exception (NV, NX, invalid_fp_register (FdTOi, FqTOi))

E11.1.3 Convert Between Floating-Point Formats Instructions

opcode	opf	operation
FsTOd	011001001	Convert Single to Double
FsTOq	011 001101	Convert Single to Quad
FdTOs	011000110	Convert Double to Single
FdTOq	011001110	Convert Double to Quad
FqTOs	011000111	Convert Quad to Single
FqTOd	011001011	Convert Quad to Double

Format:



* Not used, must be 0.

Syntax:

fstod $freg_{rs2}$, $freg_{rd}$
fstoq $freg_{rs2}$, $freg_{rd}$
fdtos $freg_{rs2}$, $freg_{rd}$
fdtoq $freg_{rs2}$, $freg_{rd}$
fqtos $freg_{rs2}$, $freg_{rd}$
fqtod $freg_{rs2}$, $freg_{rd}$

Description:

These instructions convert the floating-point operand in the f register(s) specified by rs2 to a floating-point number in the destination format. They write the result into the f register(s) specified by rd. Rounding is performed according to the RD field in the FSR.

FqTOd, FqTOs, and FdTOs (the “narrowing” conversion instructions) can result in OF, UF, and NX exceptions. FdTOq, FsTOq, and FsTOd (the “widening” conversion instructions) cannot. Any of these six instructions can trigger an NV exception if the source operand is a signaling NaN.

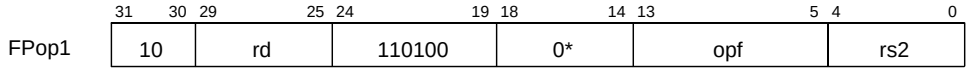
Traps:

fp_disabled
fp_exception (OF, UF, NV, NX, invalid_fp_register)

E11.1.4 Floating-Point Move Instructions

opcode	opf	operation
FMOVs	000000001	Move
FNEGs	000000101	Negate
FABSs	000001001	Absolute Value

Format:



* Not used, must be 0.

Syntax:

fmovs $freg_{rs2}$, $freg_{rd}$
fnegs $freg_{rs2}$, $freg_{rd}$
fabss $freg_{rs2}$, $freg_{rd}$

Description:

FMOVs copies the contents of f[rs2] to f[rd]. FNEGs copies the contents of f[rs2] to f[rd] with the sign bit complemented. FABSs copies the contents of f[rs2] to f[rd] with the sign bit cleared. These instructions do not round.

Programming Notes:

(1) One FMOVs instruction per word is required to transfer a multiple-precision value between f registers.

(2) If the source and destination registers ($freg_{rs2}$ and $freg_{rd}$) are the same, a single FNEGs (FABSs) instruction performs negation (absolute value) for any operand precision, including double- and quad- precisions. If the source and destination registers are different, an FNEGs (FABSs) and a following FMOVs instruction perform a double-precision negation (absolute value); an FNEGs (FABSs) and three following FMOVs instructions perform a quad-precision negation (absolute value).

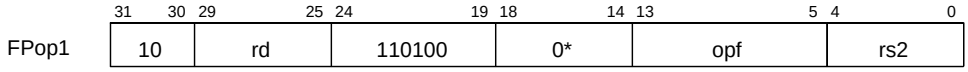
Traps:

fp_disabled

E11.1.5 Floating-Point Square Root Instructions

opcode	opf	operation
FSQRTs	000101001	Square Root Single
FSQRTd	000101010	Square Root Double
FSQRTq	000101011	Square Root Quad

Format:



* Not used, must be 0.

Syntax:

fsqrts $freg_{rs2}$, $freg_{rd}$
fsqrtd $freg_{rs2}$, $freg_{rd}$
fsqrtq $freg_{rs2}$, $freg_{rd}$

Description:

These instructions generate the square root of the floating-point operand in the f register(s) specified by the rs2 field, and place the result in the destination f register(s) specified by the rd field. Rounding is performed according to the rd field in the FSR.

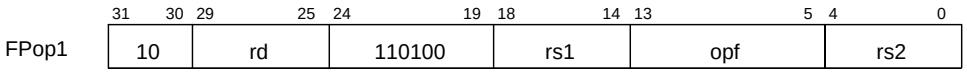
Traps:

fp_disabled
fp_exception (NV, NX, invalid_fp_register (FSQRTd, FSQRTq))

E11.1.6 Floating-Point Add and Subtract Instructions

opcode	opf	operation
FADDs	001000001	Add Single
FADDd	001000010	Add Double
FADDq	001000011	Add Quad
FSUBs	001000101	Subtract Single
FSUB d	001000110	Subtract Double
FSUBq	001000111	Subtract Quad

Format:



Syntax:

fadds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
faddd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
faddq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsubs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsubd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsubq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$

Description:

The floating-point add instructions add the f register(s) specified by the rs1 field and the f register(s) specified by the rs2 field, and write the sum into the f register(s) specified by the rd field.

The floating-point subtract instructions subtract the f register(s) specified by the rs2 field from the f register(s) specified by the rs1 field, and write the difference into the f register(s) specified by the rd field.

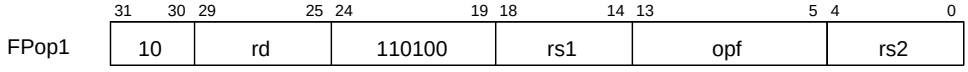
Traps:

fp_disabled
fp_exception (OF, UF, NX, NV (∞–∞), invalid_fp_register (all except FADDs and FSUBs)

E11.1.7 Floating-Point Multiply and Divide Instructions

opcode	opf	operation
FMULs	001001001	Multiply Single
FMULd	00100 1010	Multiply Double
FMULq	001001011	Multiply Quad
FsMULd	001101001	Multiply Single to Double
FdMULq	001101110	Multiply Double to Quad
FDIVs	001001101	Divide Single
FDIVd	001001110	Divide Double
FDIVq	001001111	Divide Quad

Format:



Syntax:

fmuls	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efmuls	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdivs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdivd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdivq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$

Description:

The floating-point multiply instructions multiply the f register(s) specified by the rs1 field by the f register(s) specified by the rs2 field, and write the product into the f register(s) specified by the rd field.

The FsMULd instruction provides the exact double-precision product of two single-precision operands without underflow, overflow, or rounding error. Similarly, FdMULq provides the exact quad-precision product of two double-precision operands.

The floating-point divide instructions divide the f register(s) specified by the rs1 field by the f register(s) specified by the rs2 field, and write the quotient into the f register(s) specified by the rd field.

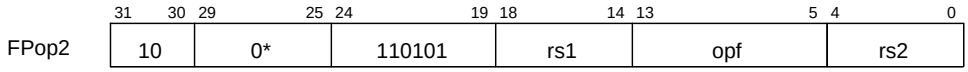
Traps:

fp_disabled
fp_exception (OF, UF, DZ (FDIV only), NV, NX, invalid_fp_register
(all except FMULs and FDIVs)

E11.1.8 Floating-Point Compare Instructions

opcode	opf	operation
FCMPs	001010001	Compare Single
FCMPd	001010 010	Compare Double
FCMPq	001010011	Compare Quad
FCMPEs	001010101	Compare Single and Exception if Unordered
FCMPEd	001010110	Compare Double and Exception if Unordered
FCMPEq	001010111	Compare Quad and Exception if Unordered

Format:



* Not used, must be 0.

Syntax:

fcmps $freg_{rs1}$, $freg_{rs2}$
fcmpd $freg_{rs1}$, $freg_{rs2}$
fcmpq $freg_{rs1}$, $freg_{rs2}$
fcmpes $freg_{rs1}$, $freg_{rs2}$
fcmped $freg_{rs1}$, $freg_{rs2}$
fcmpeq $freg_{rs1}$, $freg_{rs2}$

Description:

These instructions compare the f register(s) specified by the rs1 field with the f register(s) specified by the rs2 field, and set the floating-point condition codes as follows:

fcc	**Relation**
0	$freg_{rs1} = freg_{rs2}$
1	$freg_{rs1} < freg_{rs2}$
2	$freg_{rs1} > freg_{rs2}$
3	$freg_{rs1}$? $freg_{rs2\ (unordered)}$

The "compare and cause exception if unordered" instructions (FCMPEs, FCMPEd, and FCMPEq) cause an invalid (NV) exception if either operand is a signaling NaN or a quiet NaN. FCMP causes an invalid (NV) exception if either operand is a signaling NaN.

Programming Note:

The following restriction does not apply to the MB86936, but may apply to other SPARC processors.

A non-FPop2 (non-floating-point-operate2) instruction must be executed between an FPop2 (FCMP or FCMPE) instruction and a following FBfcc instruction. Otherwise, the result of the FBfcc is undefined.

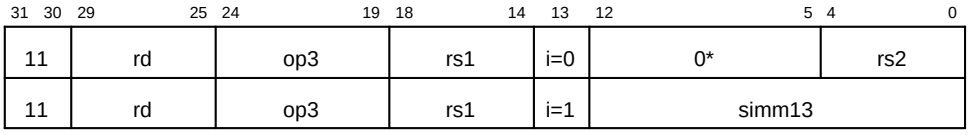
Traps:

fp_disabled
fp_exception (NV, invalid_fp_register (all except FCMPs and FCMPEs)

E11.2 Load Floating-Point (LDfp) Instructions

opcode	op3	operation
LDF	100000	Load Floating-Point Register
LDDF	100011	Load Double Floating-Point Register
LDFSR	100001	Load Floating-Point State Register

Format:



* Not used, must be 0.

Syntax:

ld [*address*] , *fregrd*
ldd [*address*] , *fregrd*
ld [*address*] , %fsr

Description:

The load single floating-point instruction (LDF) moves a word from memory into f[rd].

The load doubleword floating-point instruction (LDDF) moves a doubleword from memory into an f register pair. The most significant word at the effective memory address is moved into the even f register. The least significant word at the effective memory address +4 is moved into the following odd f register. The least significant bit of the rd field is unused and should always be set to 0 by software.

The load floating-point state register instruction (LDFSR) waits for all FPop instructions that have not finished execution to complete, then loads a word from memory into the FSR.

The effective address for the load instruction is “r[rs1] + r[rs2]” if the *i* field is 0, and “r[rs1] + sign_ext(simm13)” if the *i* field is 1.

LDF and LDFSR cause a mem_address_not_aligned trap if the effective address is not word-aligned; LDDF traps if the address is not doubleword-aligned. If the EF field of the PSR is 0 or if no FPU is present, a load floating-point instruction causes an fp_disabled trap.

Programming Notes:

(1) The MB86936 ignores the least-significant bit of the LDDF rd field. Other SPARC processors may cause an fp_exception_trap with FSR.ftt = invalid_fp_register if the bit is 1.

(2) If any of the three instructions that follow an LDFSR (in time) is an FBfcc, the value of the FSR fcc field that is seen by the FBfcc is undefined. This restriction does not apply to the MB86936, but may apply to other SPARC processors:

Implementation Note:

If a load floating-point instruction traps with a data access exception, the destination f register(s) remain unchanged.

Traps:

fp_disabled
fp_exception (sequence_error)
data_access_exception
mem_address_not_aligned

E11.3 Store Floating-Point (STfp) Instructions

opcode	op3	operation
STF	100100	Store Floating-Point
STDF	100111	Store Double Floating-Point
STFSR	100101	Store Floating-Point State Register
STDFQ†	100110	Store Double Floating-Point deferred-trap Queue.

† privileged instruction

Format:

31 30	29 25	24 19	18 14	13	12 5	4 0
11	rd	op3	rs1	i=0	0*	rs2
11	rd	op3	rs1	i=1	simm13	

*Not used, must be 0.

Syntax:

st *fregrd* , [*address*]
std *fregrd* , [*address*]
st %fsr , [*address*]
std %fq , [*address*]

Description:

The store single floating-point instruction (STF) copies f[rd] into memory.

The store double floating-point instruction (STDF) copies a doubleword from an f register pair into memory. The more-significant word (in the even-numbered f register) is written into memory at the effective address, and the less-significant word (in the odd-numbered f register) is written into memory at “effective address + 4”. The least-significant bit in the rd field is not used and should be written to 0 by software.

The store floating-point deferred-trap queue instruction (STDFQ) stores the front doubleword of the Floating-Point Queue (FQ) into memory. An attempt to execute STDFQ when the FQ is empty (FSR.qne = 0) should cause an fp_exception trap with FSR.ftt set to 4 (sequence_error).

The store floating-point state register instruction (STFSR) waits for any concurrently executing Fpop instructions to complete, then writes the FSR into memory. STFSR zeros FSR.ftt after writing the FSR to memory.

The effective address for a store instruction is "r[rs1] + r[rs2]" if the *i* field is 0, or "r[rs1] + sign_ext(simm13)" if the *i* field is 1.

STF and STFSR cause a mem_address_not_aligned trap if the address is not word-aligned, and STDF and STDFQ trap if the address is not doubleword aligned. If the EF field of the PSR is 0 or if the FPU is not present, a store floating-point instruction causes an fp_disabled trap.

Programming Note:

The MB86936 ignores the least-significant bit of the rd field of the STDF. Other SPARC processors may assert an fp_exception_trap with FSR.ftt = invalid_fp_register if the bit is 1.

Implementation Note:

The MB86936 implementation might cause a data_access_exception trap due to a "non-resumable machine-check" error during an "effective address + 4" memory access, even though the corresponding "effective address" access did not cause an error. Thus, memory data at the effective memory address may be changed in this case. (Note that this cannot happen across a page boundary because of the doubleword alignment restriction.)

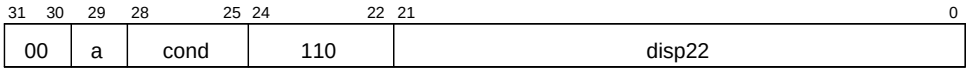
Traps:

fp_disabled
fp_exception (sequence_error (STDFQ))
privileged_instruction (STDFQ only)
mem_address_not_aligned
data_access_exception

E11.4 Branch on Floating-Point Cond. Codes (FBfcc) Instruc.

opcode	cond	operation	fcc test
FBA	1000	Branch Always	1
FBN	0000	Branch Never	0
FBU	0111	Branch on Unordered	U
FBG	0110	Branch on Greater	G
FBUG	010 1	Branch on Unordered or Greater	G or U
FBL	0100	Branch on Less	L
FBUL	00 11	Branch on Unordered or Less	L or U
FBLG	0010	Branch on Less or Greater	L or G
FBNE	0001	Branch on Not Equal	L or G or U
FBE	1001	Branch on Equal	E
FBUE	1010	Branch on Unordered or Equal	E or U
FBGE	1011	Branch on Greater or Equal	E or G
FBUGE	1100	Branch on Unordered or Greater or Equal	E or G or U
FBLE	1101	Branch on Less or Equal	E or L
FBULE	1110	Branch on Unordered or Less or Equal	E or L or U
FBO	1111	Branch on Ordered	E or L or G

Format:



Syntax:

```
fba {,a}     label
fbn {,a}     label
fbu {,a}     label
fbg {,a}     label
fbug {,a}    label
fbl {,a}     label
fbul {,a}    label
fblg {,a}    label
fbne {,a}    label    (synonym: fbnz)
fbe {,a}     label    (synonym: fbz)
fbue {,a}    label
fbge {,a}    label
fbuge {,a}   label
fble {,a}    label
fbule {,a}   label
```

fbo {,a} label

Note: To set the "annul" bit for FBfcc instructions, append " ,a" to the opcode mnemonic. For example, use "fbl ,a label". The braces ({}) in the preceding table indicate that the " ,a" are optional.

Description:

Unconditional Branches (FBA, FBN)

If its annul field is 0, an FBN (Branch Never) instruction executes as a "NOP". If its annul field is 1, the following (delay) instruction is annulled (not executed). In neither case does a transfer of control take place.

FBA (Branch Always) causes a PC-relative, delayed control transfer to the address "PC + (4 x sign_ext(disp22))," regardless of the value of the floating-point condition code bits. If the annul field of the branch instruction is 1, the delay instruction is annulled (not executed). If the annul field is 0, the delay instruction is executed.

Fcc-Conditional Branches

Conditional FBfcc instructions (all except FBA and FBN) evaluate the floating-point condition codes (*fcc*) according to the cond field of the instruction. Such evaluation produces either a "true" or "false" result. If "true," the branch is taken; that is, the instruction causes a PC-relative delayed control transfer to the address "PC + (4 x sign_ext(disp22))." If "false," the branch is not taken.

If a conditional branch is taken, the delay instruction is always executed regardless of the value of the annul field. If a conditional branch is not taken and the *a* (annul) field is 1, the delay instruction is annulled (not executed). (Note that the annul bit has a **different** effect on conditional branches than on unconditional branches).

An FBfcc should not be placed in the delay slot of a conditional branch instruction.

If the PSR's EF bit is 0, or if an FPU is not present, an FBfcc instruction does not branch, does not annul the following instruction, and generates an fp_disabled trap.

Programming Notes:

The following restrictions do not apply to the MB86936, but may apply to other SPARC processors.

(1) If the instruction executed immediately before an FBfcc is an FPop2 instruction, the result of the FBfcc is undefined. Therefore, at least one non-FPop2 instruction should be executed between an FPop2 and a following FBfcc.

(2) If any of the three instructions that follow (in time) an LDFSR is an FBfcc, the value of the fcc field of the FSR that is seen by the FBfcc is undefined.

Traps:

fp_disabled
fp_exception (sequence_error)

CHAPTER

E12

Power Down Mode

The MB86936 features a power-down mode to partially or fully power down the processor.

The processor is divided into six functional logic groups that can be powered down through the Power-Down Register. The six groups are categorized as *independent* or *dependent* as follows:

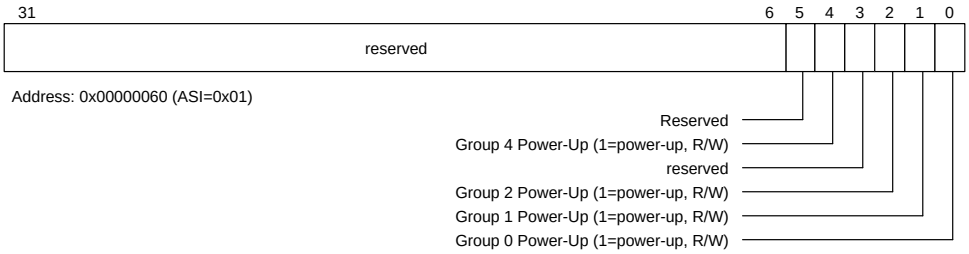
Group	Category	Processor Function
0	Independent	FPU
1	Independent	DMA
2	Dependent	Core (IU, BIU, I_cache, D_cache)
4	Dependent	ICE

Each independent group can be individually powered up or down independently of the other groups.

The dependent groups must not be powered down alone. If group 2 (processor core) or group 4 (ICE) is powered down, all other groups must be powered down.

E12.1 Power-Down Register

The Power-Down Register (also called the *Shadow Register*) contains bits that control power-down as follows:



Bits 31-6: Reserved

Bit 5: Reserved

Bit 4: Group 4 Power-Down (G4PD) — Controls power to the ICE logic. When set to 1, the ICE logic is powered-down; when cleared to 0, the ICE logic is powered-up.

Bit 3: Reserved

Bit 2: Group 2 Power-Down (G2PD) — Controls power to the processor core. When set to 1, the IU, BIU, and caches are powered-down; when cleared to 0, the IU, BIU, and caches are powered-up.

Bit 1: Group 1 Power-Down (G1PD) — Controls power to the DMA. When set to 1, the DMA is powered-down; when cleared to 0, the DMA is powered-up.

Bit 0: Group 0 Power-Down (G0PD) — Controls power to the FPU. When set to 1, the FPU is powered-down; when cleared to 0, the FPU is powered-up.

The register is written 0x3f to power down the entire processor. This is called a *global* power down.

The reset state of the Power-Down Register is 0x0.

E12.2 Power-Down Operation

A group power-down bit can be changed in the Power-Down Register by executing two consecutive store-alternate instructions. Each independent group can be selectively powered up by clearing the group bit in the Power-Down Register to 0 with store-alternate instructions.

Forcing the -PDRESET signal pin low for at least two system clock cycles clears the Power-Down Register, resulting in global power up. No group can be powered down while -PDRESET is held low.

Programming Note:

Two consecutive store-alternate instruction are required to power up or power down. The power-up/power-down state is undefined if the store-alternate instructions are not

executed consecutively. Interrupts should therefore be disabled before executing the store-alternate instructions, and should be re-enabled after the instructions have executed, as shown in the following example:

```
! disable traps to ensure that the sta instructions execute back-
! to-back.

set   ox10c0, %g1       ! enable EF, S, PS
mov   %g1, %psr
nop
nop

! the following back-to-back sta instructions result in global
! power down

sta   %g6, [%g5] 0x1    ! %g6=0x3f %g5=0x60
sta   %g6, [%g5] 0x1

! re-enable traps

set   0x10e0, %g1       ! enable EF, S, PS, ET
mov   %g1, %psr
nop
nop
```

CHAPTER

E13

MB86936 External Interface

E13.1 SIGNAL DESCRIPTIONS[1]

SYMBOL	TYPE	DESCRIPTION
–RESET	I	**SYSTEM RESET:** Asserting reset for at least 4 processor cycles after the clock has stabilized causes the MB86936 to be initialized.
XTAL1 (CLK_IN) XTAL2	I/O O G(Q) I (Q)	**EXTERNAL OSCILLATOR:** The frequency of the XTAL1 input determines the frequency of operation of the bus. The internal frequency of operation of the part is a function of the frequency of the XTAL1 signal and the –CLKDBL signal. The XTAL2 pin should be left floating.
CLKOUT1	O G(Q) I (Q)	**CLOCK OUTPUT 1:** This is an output signal against which MB86936 bus transactions can be referenced. The CLKOUT1 frequency is the same as the frequency applied to XTAL1. CLKOUT1 is in phase with CLK_IN.
CLKOUT2	O G(Q) I (Q)	**CLOCK OUTPUT 2:** This is an output signal against which MB86936 bus transactions can be referenced. The CLKOUT2 frequency is the same as the frequency applied to XTAL1. CLKOUT2 is out of phase with CLK_IN.

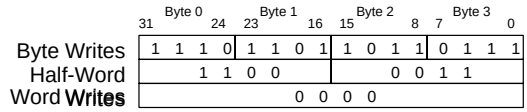
E13.1 SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–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.
–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 MB86936 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.
–NONCACHE	I	**NON-CACHEABLE:** Asserted by the memory system to indicate the data on the memory bus in the non-cacheable memory region. Logic 0 indicates non-cacheable and logic 1 indicates cacheable. This pin is ignored when the internal cacheability is used.
Symbol	**Type**	**Description**
IRL <3>/IRQ15 IRL <2>/IRQ14 IRL <1>/IRQ13 IRL <0>/IRQ12	I	**INTERRUPT REQUEST:** These are prioritized system requests. IRQ15 has the highest priority and IRQ1 has the lowest priority. IRQ11-1 are generated by the on-chip peripherals. IRL<3:0> are encoded interrupt inputs and IRQ15-12 are decoded interrupt inputs. The trigger for each IRQ interrupt can be programmed for a high level, a low level, a rising edge, or a falling edge. The external interrupt requests are sampled during two successive external bus clock periods to minimize false interrupts.
LSYNC	I	Video line sync.
PSYNC	I/O	Video page sync.
VDAT <3:0>	I/O	Video data input/output.
VCLK	I/O	Video clock.

E13.1 SIGNAL DESCRIPTIONS (Continued)[1]

<table>
<tr><th>SYMBOL</th><th>TYPE</th><th>DESCRIPTION</th></tr>
<tr><td>–CS0, –CS1,
–CS2, –CS3,
–CS4,</td><td>O
S(L)
G(1)
I (1)</td><td>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 five 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 MB86936 without the need for additional logic.</td></tr>
<tr><td>ADR <27:2></td><td>O
S(L)
G(Z)
I (1)</td><td>ADDRESS BUS: The 26-bit ADDRESS BUS (ADR<27:2>) 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. If the DRAM Controller is enabled, then MA<11:0> is output on ADR<27:16> during DRAM accesses.</td></tr>
<tr><td>ASI <3:0>/
VDAT<7:4></td><td>I/O
S(L)
G(Z)
I (1)
/
I/O</td><td>ADDRESS SPACE IDENTIFIERS: The ADDRESS SPACE IDENTIFIERS are outputs which indicate to which of 16 available spaces the current ADDRESS BUS value corresponds. The ASI values are defined as follows:
<table>
<tr><th>ASI</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>
</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 0x8 is cacheable. When 8-bit video is enabled, the ASI<3:0> pins are used for VDAT<7:4> I/O. However, the ASI is used internally.</td></tr>
</table>

E13.1 SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–BMODE8	I S(L)	**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 MB86936 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, 10, 11). 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.
–BMODE16	I S(L)	**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 MB86936 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.
–BE0-3	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 accesses as follows: Bus Mode / Byte / BE<2:3> 8-bit: 0 / 00; 1 / 01; 2 / 10; 3 / 11 16-bit: 0 & 1 / 00; 2 & 3 / 10
D <31:0>	I/O S(L) G(Z) I (Z)	**DATA BUS:** The bus interface has 32 bidirectional data pins (D<31:0>) 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 mode is enabled and D<15:0> are used when 16-bit mode is enabled.

E13.1 SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–AS	O S(L) G(Z) I (1)	**ADDRESS STROBE:** A control signal asserted by the MB86936 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.
–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 5 address ranges each with different transaction times can be programmed.
–DREQ0-2	I A(L)	**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 transfers. –DREQ0 corresponds to DMA channel 0, –DREQ1 corresponds to DMA channel 1 and –DREQ2 corresponds to DMA channel 2. If DMA Channel 0 is being used by the Video Interface, –DREQ0 is ignored.
–DACK0-2	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, –DACK1 corresponds to DMA channel 1 and –DACK2 corresponds to DMA channel 2.
–EOP0-2	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, –EOP1 corresponds to DMA channel 1 and –EOP2 corresponds to channel 2. A pull-up holds –EOP0-2 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.

E13.1 SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
CLK_ECB	I	**EXTERNAL CLOCK BYPASS:** Tying this signal high causes the CLK_IN signal to bypass the Phase Lock Loop (PLL). This signal is used for testing of the chip.
–CLKDBL	I	**CLOCK DOUBLER:** Tying this signal low causes the internal logic to run at twice the frequency of the clock input.
Symbol	**Type**	**Description**
–RAS3/ –SAME_PAGE	O O S(L) G(1) I (1)	**–RAS3 DRAM Row Address Strobe:** Can be connected directly to the corresponding –RAS pin of a DRAM. Typically, –RAS is used to select a DRAM bank. When the 936 DRAM controller is disabled, this pin will output **–SAME_PAGE.** **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. –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.
–RAS2/ –TIMER_OVF	O O S(L) G(Q) I (Q)	When the 936 DRAM controller is disabled, this pin will output **–TIMER_OVF** **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.
–RAS1/ –CS5, –RAS0	O	When the 936 DRAM controller is enabled –CS4, –CS5, represent the same DRAM space. This is to control which part of the DRAM is non-cacheable, When the 936 DRAM controller is disabled, –RAS1 pin will output –CS5. Please see the pin description of CHIP SELECTs –CS0-4.
–CAS0-3	O	**DRAM Column Address Strobe:** Can be connected directly to the corresponding –CAS pin of a DRAM. –CAS is used to select bytes within a 32-bit DRAM word.
–DWE	O	**DRAM Write Enable:** Can be connected directly to the corresponding –WE pin of a DRAM.
–NVWE	O	**WRITE ENABLE FOR NON-VOLATILE MEMORY:** This signal is asserted one cycle after –AS and stays asserted till one cycle before the end of the transaction for a write operation. The signal is generated only when internal wait state generation in enabled for current access.
–OE	O	**OUTPUT ENABLE:** The signal is asserted one cycle after –AS and stays asserted till the last cycle of a read operation. This signal is generated when internal wait state generation is enabled for the current access.
–READYOUT	O	Ready Out for External Bus Masters using Internal Ready Generation.
TIMEROUT0	O	Timer output pin. According to the mode, the output wave functions as (1) periodic interrupt signal output; (2) square wave output: (3) one-shot pulse output; This pin is low during reset.
PARITY 3-0	O	Parity3 corresponds to D<31:24> Parity2 corresponds to D<23:16> Parity1 corresponds to D<15:8> Parity0 corresponds to D<7:0>

E13.1 SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–PDRESET	I	Power Down Reset is asserted by the external system to get the part out of power down mode.
EMU_SD <3:0>	I/O	**EMULATOR STATUS/DATA BITS:** Bi-directional pins used by a hardware emulator to control and monitor MB86936 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 MB86936 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.
–EMU_ENB	I/O	**EMULATOR ENABLE:** Tied low while the MB86936 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.
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.

1. 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

E14

MB86936 JTAG

E14.1 MB86936 JTAG Pin List

The MB86936 JTAG cells are arranged in a shift register configuration (see Figure E11-1. When shifting in a JTAG pattern through TDI, the LSB should correspond to the JTAG cell value for -EMU_SD<3> pin whereas, the MSB of the pattern should correspond to the IRL<3> pin's JTAG cell. As far as JTAG output through TDO is concerned, the first bit out corresponds to –EMU_SD<3> JTAG cell value and the last output bit corresponds to the IRL<3> JTAG cell value. Table E14-1 lists the order of all of the JTAG cells.

Table E14-1: JTAG Pin Order

Order	JTAG Cell	JTAG Cell Type	Function
1	–DREQ2	input	DMA Channel 2 input Request
2	–DACK2	output	DMA Channel 2 output Acknowledge
3	RAS2/–TIMER_ OVF	output	RAS2 (DRAM controller enabled) –TIMER_OVF (DRAM controller disabled)
4	XTAL1	input	Crystal input
5	RAS0	output	RAS0 (DRAM controller enabled)
6	–NONCACHE	input	Non-cacheable address

Table E14-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
7	eopio2	output	Bidirectional control signal for –EOP2 eopio2 = 1, –EOP2 is an input eopio2 = 0, –EOP2 is an output
8	–EOP2_i	input	Input bit of –EOP2
9	–EOP2_o	output	Output bit of –EOP2
10	PARITY_i <2>	input	Input bit of PARITY<2>
11	PARITY_o<2>	output	Output bit of PARITY<2>
12	PARITY_i <3>	input	Input bit of PARITY<3>
13	PARITY_o<3>	output	Output bit of PARITY<3>
14	EMU_BRK	input	Emulator break input
15	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
16	EMU_SD_i<3>	input	Input bit 3 of EMU_SD<3:0> bus
17	EMU_SD_o<3>	output	Output bit 3 of EMU_SD<3:0> bus
• •	• •	• •	• •
22	EMU_SD_i<0>	input	Input bit 0 of EMU_SD<3:0> bus
23	EMU_SD_o<0>	output	Output bit 0 of EMU_SD<3:0> bus
24	EMU_D_i<3>	input	Input bit 3 of EMU_D<3:0> bus
25	EMU_D_o<3>	output	Output bit 3 of EMU_D<3:0> bus
• •	• •	• •	• •
30	EMU_D_i<0>	input	Input bit 0 of EMU_D<3:0> bus
31	EMU_D_o<0>	output	Output bit 0 of EMU_D<3:0> bus
32	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
33	–EMU_EN_i	input	Input bit of –EMU_ENB pin
34	–EMU_EN_o	output	Output bit of –EMU_ENB pin
35	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
36	D_i<31>	input	Input bit 31 of D<31:0> bus
37	D_o<31>	output	Output bit 31 of D<31:0> bus
• •	• •	• •	• •
62	D_i<18>	input	Input bit 18 of <31:0> bus
63	D_o<18>	output	Output bit 18 of D<31:0> bus
64	–BMODE16	input	

Table E14-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
65	D_i<17>	input	Input bit 17 of D<31:0> bus
66	D_o<17>	output	Output bit 17 of D<31:0> bus
67	D_i<16>	input	Input bit 16 of D<31:0> bus
68	D_o<16>	output	Output bit 16 of D<31:0> bus
69	D_i<15>	input	Input bit 15 of D<31:0> bus
70	D_o<15>	output	Output bit 15 of D<31:0> bus
71	–BMODE8	input	
72	D_i<14>	input	Input bit 14 of D<31:0> bus
73	D_o<14>	output	Output bit 14 of D<31:0> bus
• •	• •	• •	• •
84	D_i<8>	input	Input bit 8 of <31:0> bus
85	D_o<8>	output	Output bit 8 of D<31:0> bus
86	–DWE	output	DRAM Write Enable
87	D_i<7>	input	Input bit 7 of <31:0> bus
88	D_o<7>	output	Output bit 7 of <31:0> bus
89	D_i<6>	input	Input bit 6 of <31:0> bus
90	D_o<6>	output	Output bit 6 of <31:0> bus
91	–BMREQ	output	Burst mode request output signal
92	D_i<5>	input	Input bit 5 of D<31:0> bus
93	D_o<5>	output	Output bit 5 of D<31:0> bus
• •	• •	• •	• •
102	D_i<0>	input	Input bit 0 of <31:0> bus
103	D_o<0>	output	Output bit 0 of D<31:0> bus
104	–RESET	input	Chip reset pin
105	–BREQ	input	Bus request input
106	–MEXC	input	Memory exception input
107	–READY	input	External memory transaction complete signal
108	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.
109	–CAS0	output	DRAM Column 0
110	–BGRNT	output	Bus grant output signal
111	–ERROR	output	Error output signal
112	–LOCK	output	Bus lock output signal

Table E14-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
113	–BMACK	input	Burst mode acknowledge input signal
114	–RD/WR_i	input	Memory Read/Write input signal
115	–RD/WR_o	output	Memory Read/Write output signal
116	–AS_i	input	Start of memory transaction input signal
117	–AS_o	output	Start of memory transaction output signal
118	–PBREQ	output	Processor Bus Request output signal
119	CAS1	output	DRAM column 1
120	CAS2	output	DRAM column 2
121	–CS<0>	output	Chip select 0 output signal
122	–DREQ0	input	DMA Channel 0 request input signal
123	–CS<1>	output	Chip select 1 output signal
124	–CS<2>	output	Chip select 2 output signal
125	–CS<3>	output	Chip select 3 output signal
126	–CS<4>	output	Chip select 4 output signal
127	–DREQ1	input	DMA Channel 1 request input signal
128	RAS1/–CS<5>	output	DRAM RAS1 (DRAM Controller enabled) –CS5 (DRAM Controller disabled)
129	RAS3/ –SAMEPAGE	output	DRAM RAS3 (DRAM Controller enabled) –SAMEPAGE (DRAM Controller disabled)
130	–DACK0	output	DMA Channel 0 output acknowledge
131	BE<3>	output	Byte enable 3 output signal
132	BE<2>	output	Byte enable 2 output signal
133	BE<1>	output	Byte enable 1 output signal
134	BE<0>	output	Byte enable 0 output signal
135	atstatejo†	output	Bidirectional control signal for ASI<3:0>/VDAT<7:4> If atstatejo = 1, ASI<3:0>/VDAT<7:4> are inputs If atstatejo = 0, ASI<3:0>/VDAT<7:4> are inputs
136	ASI_i<0>/ VDAT_i<4>	input	ASI<0> input (4 bit video), VDAT<4> input (8 bit video)
137	ASI_o<0>/ VDAT_o<4>	output	ASI<0> output (4 bit video), VDAT<4> output (8 bit video)
138	CAS3	output	DRAM column 3
139	ASI_i<1>/ VDAT_i<5>	input	ASI<1> input (4 bit video), VDAT<5> input (8 bit video)
140	ASI_o<1>/ VDAT_o<5>	output	ASI<1> output (4 bit video), VDAT<5> output (8 bit video)

Table E14-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
141	ASI_i<2>/ VDAT_i<6>	input	ASI<2> input (4 bit video), VDAT<6> input (8 bit video)
142	ASI_o<2>/ VDAT_o<6>	output	ASI<2> output (4 bit video), VDAT<6> output (8 bit video)
143	ASI_i<3>/ VDAT_i<7>	input	ASI<3> input (4 bit video), VDAT<7> input (8 bit video)
144	ASI_o<3>/ VDAT_o<7>	output	ASI<3> output (4 bit video), VDAT<7> output (8 bit video)
145	–DACK1	output	DMA Channel 1 output acknowledge
146	RDYOUT	output	
147	–PDRESET	input	Powerdown Reset input signal
148	–NVWE	output	Non–volatile memory write enable
149	–OE	output	PROM output enable
150	ADR_i<2>	input	Input bit of ADR<2>
151	ADR_o<2>	output	Output bit of ADR<2>
⋮	⋮	⋮	⋮
156	ADR_i<5>	input	Input bit of ADR<5>
157	ADR_o<5>	output	Output bit of ADR<5>
158	eopio0	output	Bidirectional control for –EOP0 pin eopio0 = 1: –EOP0 is input eopio0 = 0: –EOP0 is output
159	ADR_i<6>	input	Input bit of ADR<6>
160	ADR_o<6>	output	Output bit of ADR<6>
161	ADR_i<7>	input	Input bit of ADR<7>
162	ADR_o<7>	output	Output bit of ADR<7>
163	–EOP0_i	input	Input bit for –EOP0
164	–EOP0_o	output	Output bit for –EOP1
165	ADR_i<8>	input	Input bit of ADR<8>
166	ADR_o<8>	output	Output bit of ADR<8>
167	ADR_i<9>	input	Input bit of ADR<9>
168	ADR_o<9>	output	Output bit of ADR<9>
169	eopio1	output	Bidirectional control for –EOP1 pin eopio1 = 1: –EOP0 is input eopio1 = 0: –EOP0 is output
170	ADR_i<10>	input	Input bit of ADR<10>
171	ADR_o<10>	output	Output bit of ADR<10>

Table E14-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
172	–EOP1_i	input	
173	–EOP1_o	output	
174	ADR_i<11>	input	Input bit of ADR<11>
175	ADR_o<11>	output	Output bit of ADR<11>
176	ADR_i<12>	input	Input bit of ADR<12>
177	ADR_o<12>	output	Output bit of ADR<12>
178	addenbjo†	output	Bidirectional for ADR<27:2> pin addenbjo = 1: ADR<27:2> are tri–stated addenbjo = 0: ADR<27:2> are outputs
179	ADR_i<13>	input	Input bit of ADR<13>
180	ADR_o<13>	output	Output bit of ADR<13>
• • •	• • •	• • •	• • •
185	ADR_i<16>	input	Input bit of ADR<16>
186	ADR_o<16>	output	Output bit of ADR<16>
187	PARITY_i<0>	input	Input bit of PARITY<0>
188	PARITY_o<0>	output	Output bit of PARITY<0>
189	ADR_i<17>	input	Input bit of ADR<17>
190	ADR_o<17>	output	Output bit of ADR<17>
191	TIMEROUT0	output	Timer 0 output
192	LSYNC	input	Video Line Sync input signal
193	ADR_i<18>	input	Input bit of ADR<18>
194	ADR_o<18>	output	Output bit of ADR<18>
195	ADR_i<19>	input	Input bit of ADR<19>
196	ADR_o<19>	output	Output bit of ADR<19>
197	ADR_i<20>	input	Input bit of ADR<20>
198	ADR_o<20>	output	Output bit of ADR<20>
199	PARITY_i<1>	input	Input bit of PARITY<1>
200	PARITY_o<1>	output	Output bit of PARITY<1>
201	ADR_i<21>	input	Input bit of ADR<21>
202	ADR_o<21>	output	Output bit of ADR<21>
203	–CLKDBL	input	Clock Double
204	ADR_i<22>	input	Input bit of ADR<22>
205	ADR_o<22>	output	Output bit of ADR<22>
• • •	• • •	• • •	• • •
210	ADR_i<25>	input	Input bit of ADR<25>

Table E14-1: JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
211	ADR_o<25>	output	Output bit of ADR<25>
212	psyncio†	output	Bidirectional control signal for PSYNC psyncio = 1: PSYNC is an input psyncio = 0: PSYNC is an output
213	PSYNC_i	input	Input bit of PSYNC
214	PSYNC_o	output	Output bit of PSYNC
215	vclkio†	output	Bidirectional control signal for VCLK vclkio = 1: VCLK is an input vclkio = 0: VCLK is an output
216	VCLK_i	input	Input bit of VCLK
217	VCLK_o	output	Output bit of VCLK
218	ADR_i<26>	input	Input bit of ADR<26>
219	ADR_o<26>	output	Output bit of ADR<26>
220	ADR_i<27>	input	Input bit of ADR<27>
221	ADR_o<27>	output	Output bit of ADR<27>
222	vdatio†	output	Bidirectional control signal for VDAT<3:0> vdatio = 1: VDAT<3:0> are inputs vdatio = 0: VDAT<3:0> are outputs
223	VDAT<3>_i	input	Input bit of VDAT<3>
224	VDAT<3>_o	output	Output bit of VDAT<3>
⋮	⋮	⋮	⋮
229	VDAT<0>_i	input	Input bit of VDAT<0>
230	VDAT<0>_o	output	Output bit of VDAT<0>
231	IRL<3>	input	
232	IRL<2>	input	
233	IRL<1>	input	
234	IRL<0>	input	
235	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.