FUJITSU

SPARClite

Embedded Processor
User's Manual

MB86934 Addendum

January 1996, Edition 1.0

MB86934 ADDENDUM, EDITION 1.0

SPARC is a registered trademark of SPARC International based on technology developed by Sun Microsystems, Inc.

SPARClite is a trademark of SPARC International, Inc. based on technology developed by Sun Microsystems, Inc.

SPARCstation is a trademark of SPARC International, Inc. Products bearing the SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc.

NICE is a trademark of Fujitsu Microelectronics, Inc.

Circuit diagrams utilizing Fujitsu products are included as a means of illustrating typical semiconductor applications. Consequently, complete information sufficient for design purposes is not necessarily given.

Fujitsu Limited and its subsidiaries reserve the right to change products or specifications without notice. **Fujitsu advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.**

The information contained in this document does not convey any license under copyrights, patent rights or trademarks claimed and owned by Fujitsu Limited or its subsidiaries. Fujitsu assumes no liability for Fujitsu applications assistance, customer's product design, or infringement of patents arising from use of semiconductor devices in such systems' designs. Nor does Fujitsu warrant or represent that any patent right, copyright, or other intellectual property right of Fujitsu covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

Fujitsu Microelectronics, Inc.'s Semiconductor Division's products are not authorized for use in life support devices or systems. Life support devices or systems are device or systems which are:

1. Intended for surgical implant into the human body.
2. Designed to support or sustain life; and when properly used according to label instructions, can reasonably be expected to cause significant injury to the user in the event of failure.

The information contained in this document has been carefully checked and is believed to be entirely accurate. However, Fujitsu Limited and Fujitsu Microelectronics, Inc. assume no responsibility for inaccuracies.

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

List of Figures

List of Figures continued

List of Tables

List of Tables continued

CHAPTER

D1

Overview of MB86934

D1.1 General Description

The MB86934 is a member of the SPARClite family whose function set is a superset of that of the MB86930. It is available in a 256-pin package, and is capable of operating at 60 MHz. In addition to all the features of the MB86930 processor, the MB86934 contains the following:

- **Floating Point Unit:** The MB86934 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 except for the Nonstandard FP (NS=1) mode implementation. The FPU contains thirty-two 32-bit floating-point f registers, designated f[0] to f[31], and six vector-type f registers called FIFOs or vector registers to support floating-point operations. Newly-defined Enhanced Floating-Point operations allow access to the FIFOs.
- **Instruction Cache:** The MB86934 has an 8K-byte, 2-way set associative, sectored instruction cache with 8-word lines. Each line is individually lockable. Tags for each line contain the address tag, a supervisor/user bit, and 8 “valid” flags, one for each word of the line. 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 MB86934 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 MB86934 has two DMA channels. Each channel supports two transfer types: contiguous block and chained block transfers. The DMA also supports three transfer protocols: single-datum transfer, block transfer, and demand transfer (where data moves continue as long as an external device requests it). Four data types are supported: byte, halfword, word, and quad-word. For byte and halfword, the DMA does all the required packing/unpacking. Each channel also supports either fly-by or flow-thru transfer modes, and each can be started by either software or external hardware requests. The addressing convention for accesses is "big_endian."
- **SDRAM Interface:** A high-performance synchronous DRAM interface is integrated on-chip. This memory interface is 64-bits wide, and can achieve a peak bandwidth of 480 Mbytes/second.
- **FIFOs Mapped to Floating-Point Register File:** The MB86934 has six FIFOs. The on-chip FIFOs can be loaded from memory or stored to memory through the DMA. This allows the FPU to be decoupled from external memory latency. Using the FIFOs, DMA, and the SDRAM interface, the MB86934 can achieve up to 60 MFLOPS.
- **Configurable External Data Bus:** The MB86934 includes a data bus that can be configured at Reset as 8, 16, or 32 bits wide (when in the address space selected by chip select 0). This enables the MB86934 to boot from a single by-8 or by-16 ROM.
- **Burst Mode:** The MB86934 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 MB86934 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.
- **Power Down Modes:** The MB86934 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.

D1.2 Programmer's Model of the MB86934

D1.2.1 User-visible Registers

All the special-purpose registers and ASR registers defined on the MB86930 exist also on the MB96834.

All on-chip control/status/data registers which exist in alternate address spaces in the MB86930, with one exception, exist also on the MB86934 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 MB86934 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.

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)
0x00000174	ASI=0x1	Timer Register
0x00000178	ASI=0x1	Timer Preload Register
0x00000180	ASI=0x1	Source/Destination ASI Register (DMA0)
0x00000184	ASI=0x1	Current Source Address Register (DMA0)
0x00000188	ASI=0x1	Current Destination Address Reg (DMA0)
0x0000018C	ASI=0x1	Current Byte Count Register (DMA0)
0x00000190	ASI=0x1	Descriptor Pointer (DP) Register (DMA0)
0x00000194	ASI=0x1	Channel Control Register (DMA0)
0x00000198	ASI=0x1	Channel Status Register (DMA0)
0x000001A0	ASI=0x1	Source/Destination ASI Register (DMA1)
0x000001A4	ASI=0x1	Current Source Address Register (DMA1)
0x000001A8	ASI=0x1	Current Destination Address Reg (DMA1)
0x000001AC	ASI=0x1	Current Byte Count Register (DMA1)
0x000001B0	ASI=0x1	Descriptor Pointer (DP) Register (DMA1)
0x000001B4	ASI=0x1	Channel Control Register (DMA1)
0x000001B8	ASI=0x1	Channel Status Register (DMA1)

Emulation Registers:

ASI: 0x01

Address range: 0x0000F00-0x0000FFFF

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

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

D1.3 Internal Architecture of the MB86934

Figure D1-1 shows a block diagram of the MB86934. 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 MB86934 functional units. In general, when a load is executed, data is transferred to the Integer Unit (IU) 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 DD bus can access all units except the Instruction Cache and Instruction Tags, which can be accessed only through the ID bus. In such a case, the IU can read data (load alternate) or write data (store alternate) to the control/status/data registers of all units.

- Instruction Data Bus (ID)—This 64-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.)

- FIFO, DMA Bus (FD)—This 64-bit bus allows transfer of data from the BIU to the FIFO or DMA.

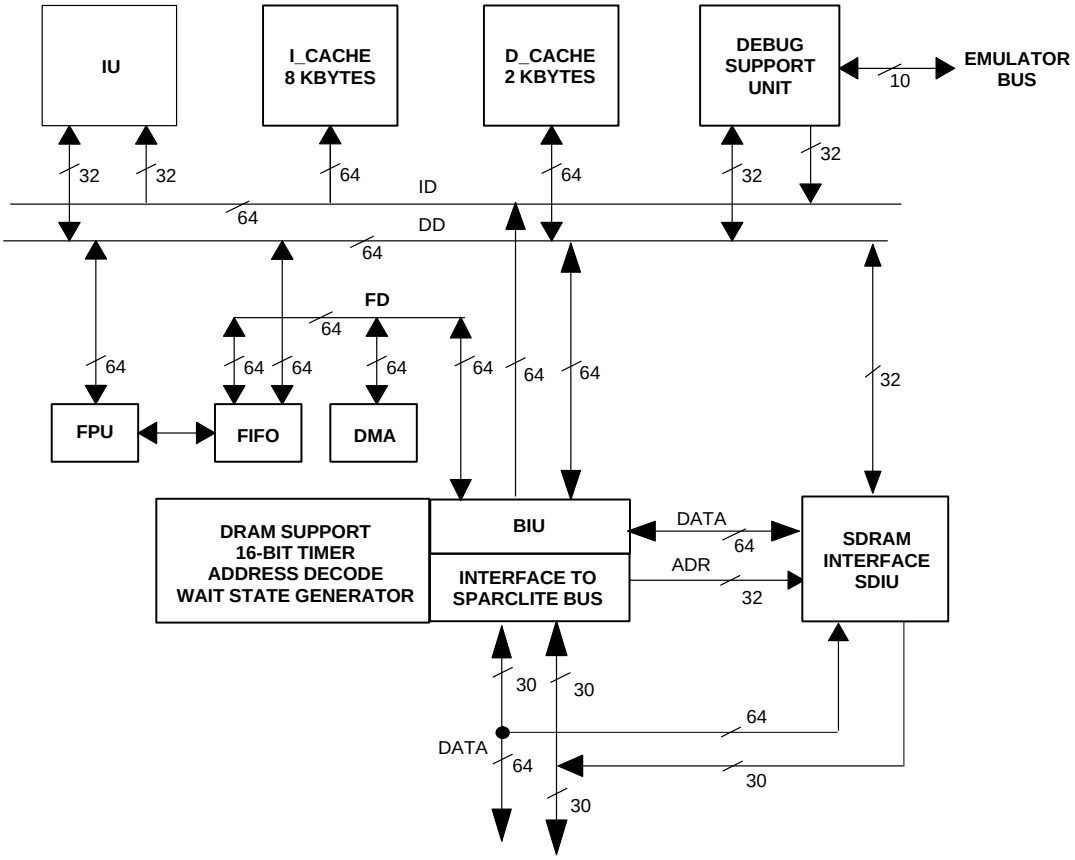


Figure D1-1. MB86934 Block Diagram

CHAPTER

D2

SDRAM Interface Unit

D2.1 Introduction

The MB86934 features an SDRAM-interface unit (SDIU) that supports BIU-SDRAM data transfers. The SDIU also allows bus masters to access single words in SDRAM.

The SDIU supports most of the SDRAM commands, can be configured for 32-bit or 64-bit bus mode, and has a 3-cycle fixed CAS latency and a 4-word burst size.

SDRAM is accessed using CS5 which is not an external pin in MB86934. Signals –SCAS, –SRAS and –SWE are generated during SDRAM operation only.

D2.2 SDIU Registers

The SDIU features three registers: the SDIU Mode Register, the SDIU Configuration Register, and the Auto Refresh Timer Register.

D2.2.1 SDIU Mode Register

This register is used to select the SDIU mode of operation including CAS latency, burst type, and burst length parameters.

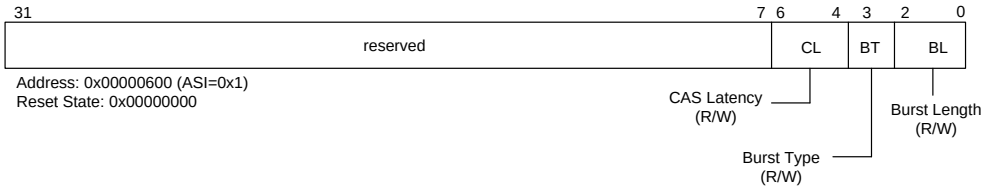


Figure D2––2. SDIU Mode Register

Bits 31-7: Reserved

Bits 6-4: Cas Latency (CL)

The CL field controls CAS Latency as follows:

Table D2–1. CAS Latency

CL	CAS Latency
000	Reserved
001	Reserved
010	Reserved
011	3
100	Reserved
101	Reserved
110	Reserved
111	Reserved

Bit 3: Burst Type (BT)

The BT bit selects the type of burst as follows:

Table D2–2. BT Burst Type

BT Bit	Burst Type
0	Reserved
1	Interleave Mode

Bits 2-0: Burst Length

The BL field selects the data burst length as follows:

Table D2–3. BL Field Data Burst Length

BL	Burst Length
000	Reserved
001	Reserved
010	4
011	Reserved
100	Reserved
101	Reserved
110	Reserved
111	Reserved

D2.2.2 SDRAM Configuration Register

This register configures the SDIU for the SDRAM type and bus size that it will support.

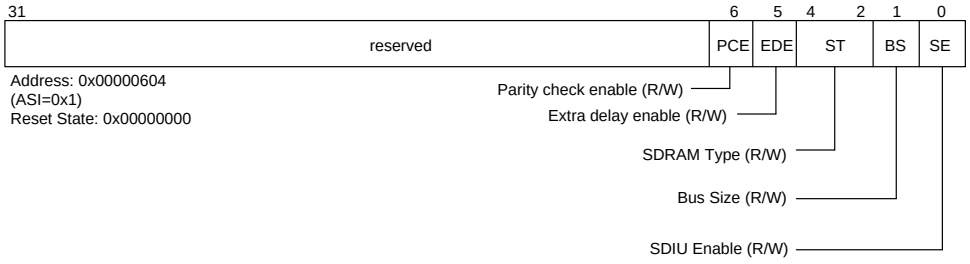


Figure D2––1. SDRAM Configuration Register

Bits 31–7: Reserved

Bit 6: Parity check enable (PCE)

The PCE enables parity-check during SDRAM read command as follows:

Table D2–4. Parity Check Enable

PCE	Remark
0	Parity error will not cause exception
1	Parity error will cause exception

Bit 5: Extra delay enable (EDE)

The EDE bit when set to 1 will allow single stage pipelining for ADR, –CAS, –RAS, –SWE, –SCS [3:0], SDQM [1:0] signals. If EDE bit is set to 0 these signals will can not be pipelined.

Bits 4-2: SDRAM Type (ST)

The ST field configures the SDIU for the type of SDRAM in use as follows:

Table D2–5. ST Configuration of SDRAM

ST	SDRAM Type
000	4 MB (X 8) (default)
001	16 MB (X 4)
010	16 MB (X 8)
011	4 MB (X16)
100	16 MB (X16)
101	Reserved
110	Reserved
111	Reserved

Bit 1: Bus Size (BS)

The BS bit selects the bus size as follows:

Table D2–6. Bus Size

BS Bit	Bus Size
0	64-bit Bus (default)
1	32-bit Bus

Specifications for the 64-bit and 32-bit bus modes are as follows:

Table D2–7. 64–Bit D–Bus Configuration

SDRAM Type	Memory Size	SDIU Page Size (Column Address X2)	Number of Chips	Column Address (934 ADR Pins)	Row Address (934 ADR Pins)	Bank Address (934 ADR Pin)
4 MB (X 16)	2 M Bytes	4 K Bytes	4	A10–A3	A11–A3	A20
4 MB (X 8)	4 M Bytes	8 K Bytes	8	A11–A3	A11–A3	A21
16 MB (X 16)	8 M Bytes	4 K Bytes	4	A10–A3	A13–A3	A20
16 MB (X 8)	16 M Bytes	8 K Bytes	8	A11–A3	A13–A3	A23
16 MB (X 4)	32 M Bytes	16 K Bytes	16	A12–A3	A13–A3	A24

Table D2–8. 32–Bit D–Bus Configuration

SDRAM Type	Memory Size	SDIU Page Size (Column Address X2)	Number of Chips	Column Address (934 ADR Pins)	Row Address (934 ADR Pins)	Bank Address (934 ADR Pin)
4 MB (X 16)	1 M Bytes	2 K Bytes	2	A9–A2	A10–A2	A19
4 MB (X 8)	2 M Bytes	4 K Bytes	4	A10–A2	A10–A2	A20
16 MB (X16)	4 M Bytes	2 K Bytes	2	A9–A2	A12–A2	A19
16 MB (X 8)	8 M Bytes	4 K Bytes	4	A10–A2	A12–A2	A22
16 MB (X 4)	16 M Bytes	8 K Bytes	8	A11–A2	A12–A2	A23

Table D2–9. –SCS [3:0] Address Map

SDRAM Type	–SCS	32-Bit D-Bus Configuration (934 ADR Pins)	64-Bit D-Bus Configuration (934 ADR Pins)
4 MB (X 16)	–SCS [3] –SCS [2] –SCS [1] –SCS [0]	~ (ADR 20 & ADR 21) ~ (ADR 20 & ~ ADR 21) ~ (~ ADR 20 & ADR 21) ~ (~ ADR 20 & ~ ADR 21)	~ (ADR 22 & ADR 21) ~ (ADR 22 & ~ ADR 21) ~ (~ ADR 22 & ADR 21) ~ (~ ADR 22 & ~ ADR 21)
4 MB (X 8)	–SCS [3] –SCS [2] –SCS [1] –SCS [0]	~ (ADR 22 & ADR 21) ~ (ADR 22 & ~ ADR 21) ~ (~ ADR 22 & ADR 21) ~ (~ ADR 22 & ~ ADR 21)	~ (ADR 23 & ADR 22) ~ (ADR 23 & ~ ADR 22) ~ (~ ADR 23 & ADR 22) ~ (~ ADR 23 & ~ ADR 22)
16 MB (X 16)	–SCS [3] –SCS [2] –SCS [1] –SCS [0]	~ (ADR 23 & ADR 22) ~ (ADR 23 & ~ ADR 22) ~ (~ ADR 23 & ADR 22) ~ (~ ADR 23 & ~ ADR 22)	~ (ADR 24 & ADR 23) ~ (ADR 24 & ~ ADR 23) ~ (~ ADR 24 & ADR 23) ~ (~ ADR 24 & ~ ADR 23)
16 MB (X 8)	–SCS [3] –SCS [2] –SCS [1] –SCS [0]	~ (ADR 24 & ADR 23) ~ (ADR 24 & ~ ADR 23) ~ (~ ADR 24 & ADR 23) ~ (~ ADR 24 & ~ ADR 23)	~ (ADR 25 & ADR 24) ~ (ADR 25 & ~ ADR 24) ~ (~ ADR 25 & ADR 24) ~ (~ ADR 25 & ~ ADR 24)
16 MB (X 4)	–SCS [3] –SCS [2] –SCS [1] –SCS [0]	~ (ADR 25 & ADR 24) ~ (ADR 25 & ~ ADR 24) ~ (~ ADR 25 & ADR 24) ~ (~ ADR 25 & ~ ADR 24)	~ (ADR 26 & ADR 25) ~ (ADR 26 & ~ ADR 25) ~ (~ ADR 26 & ADR 25) ~ (~ ADR 26 & ~ ADR 25)

Table D2–10. MB86934 Connection To MB81116820 (SDRAM)

Configuration	64–bit D–bus	32–bit D–bus	SDRAM Type 16MB (x8)
Chip	MB86934 (pin names)		MB81116820 or equivalent (pin names)
Address	A13–A3	A12–A2	A10–A0
Bank Select	A23	A22	A11
Clock Enable	SCKE	SCKE	CKE
Input Mask/Output Enable for D[63:32] Bus	SDQM0	unused	DQM +
Input Mask/Ouput Enable for D[31:0] Bus	SDQM1	SDQM1	DQM ^
Row Address Select	–SRAS	–SRAS	RAS
Column Address Select	–SCAS	–SCAS	CAS
Write Enable	–SWE	–SWE	WE
Chip Select	–SCS[3:0]	–SCS[3:0]	$\overline{CS}$ *

Note: * = –SCS[3] is connected to $\overline{CS}$ pin of SDRAM chips of highest memory-set
–SCS[0] is connected to $\overline{CS}$ pin of SDRAM chips of lowest memory-set
\+ = for SDRAM chips connected to D[63:32} bus in only "64-bit D-bus mode"
^ = for SDRAM chips connected to D[31:0] bus in "32-bit or 64-bit D-bus modes"

Table D2–11. MB86934 Connection To MB81116420 (SDRAM)

Configuration	64–bit D–bus	32–bit D–bus	SDRAM Type 16MB (x4)
Chip	MB86934 (pin names)		MB81116420 or equivalent(pin names)
Address	A13–A3	A12–A2	A10–A0
Bank Select	A24	A23	A11
Clock Enable	SCKE	SCKE	CKE
Input Mask/Output Enable for D[63:32] Bus	SDQM0	unused	DQM +
Input Mask/Ouput Enable for D[31:0] Bus	SDQM1	SDQM1	DQM ^
Row Address Select	–SRAS	–SRAS	RAS
Column Address Select	–SCAS	–SCAS	CAS
Write Enable	–SWE	–SWE	WE
Chip Select	–SCS[3:0]	–SCS[3:0]	$\overline{CS}$ *

Note: * = –SCS[3] is connected to $\overline{CS}$ pin of SDRAM chips of highest memory-set
–SCS[0] is connected to $\overline{CS}$ pin of SDRAM chips of lowest memory-set
\+ = for SDRAM chips connected to D[63:32} bus in only "64-bit D-bus mode"
^ = for SDRAM chips connected to D[31:0] bus in "32-bit or 64-bit D-bus modes"

Table D2–12. MB86934 Connections To MB81116162X (SDRAM)

			SDRAM Type
SDRAM Type	64–bit D–bus	32–bit D–bus	16MB (x16)
Chip	MB86934 (pin names)		MB81116162X or equivalent (pin names)
Address	A13–A3	A12–A2	A10–A0
Bank Select	A20	A19	A11
Clock Enable	SCKE	SCKE	CKE
Input Mask/Output Enable for D[63:32] Bus	SDQM0	unused	UDQM + LDQM +
Input Mask/Output Enable for D[31:0] Bus	SDQM1	SDQM1	UDQM ^ LDQM ^
Row Address Select	–SRAS	–SRAS	RAS
Column Address Select	–SCAS	–SCAS	CAS
Write Enable	–SWE	–SWE	WE
Chip Select	–SCS[3:0]	–SCS[3:0]	$\overline{CS}$ *

Note: * = –SCS[3] is connected to $\overline{CS}$ pin of SDRAM chips of highest memory-set
–SCS[0] is connected to $\overline{CS}$ pin of SDRAM chips of lowest memory-set
\+ = for SDRAM chips connected to D[63:32} bus in only "64-bit D-bus mode"
^ = for SDRAM chips connected to D[31:0] bus in "32-bit or 64-bit D-bus modes"

Table D2–13. MB86934 Connections To MB81141623 (SDRAM)

			SDRAM Type
SDRAM Type	64–bit D–bus	32–bit D–bus	4MB (x16)
Chip	MB86934 (pin names)		MB81141623 or equivalent (pin names)
Address	A11–A3	A10–A2	A8–A0
Bank Select	A20	A19	BS
Clock Enable	SCKE	SCKE	CKE
Input Mask/Output Enable for D[63:32] Bus	SDQM0	unused	DQML + DQMU +
Input Mask/Output Enable for D[31:0] Bus	SDQM1	SDQM1	DQML ^ DQMU ^
Row Address Select	–SRAS	–SRAS	RAS
Column Address Select	–SCAS	–SCAS	CAS
Write Enable	–SWE	–SWE	WE
Chip Select	–SCS[3:0]	–SCS[3:0]	$\overline{CS}$ *

Note: * = –SCS[3] is connected to $\overline{CS}$ pin of SDRAM chips of highest memory-set
–SCS[0] is connected to $\overline{CS}$ pin of SDRAM chips of lowest memory-set
\+ = for SDRAM chips connected to D[63:32} bus in only "64-bit D-bus mode"
^ = for SDRAM chips connected to D[31:0] bus in "32-bit or 64-bit D-bus modes"

Table D2–14. MB89634 Connections To MB8114823 (SDRAM)

			SDRAM Type
SDRAM Type	64–bit D–bus	32–bit D–bus	4MB (x8)
Chip	MB86934 (pin names)		MB8114823 or equivalent (pin names)
Address	A11–A3	A10–A2	A8–A0
Bank Select	A21	A20	BS
Precharge-Command Bit	A9	A8	PC
Clock Enable	SCKE	SCKE	CKE
Input Mask/Output Enable for D[63:32] Bus	SDQM0	unused	DQM +
Input Mask/Output Enable for D[31:0] Bus	SDQM1	SDQM1	DQM ^
Row Address Select	–SRAS	–SRAS	RAS
Column Address Select	–SCAS	–SCAS	CAS
Write Enable	–SWE	–SWE	WE
Chip Select	–SCS[3:0]	–SCS[3:0]	$\overline{CS}$ *

Note: * = –SCS[3] is connected to $\overline{CS}$ pin of SDRAM chips of highest memory-set
–SCS[0] is connected to $\overline{CS}$ pin of SDRAM chips of lowest memory-set
\+ = for SDRAM chips connected to D[63:32} bus in only "64-bit D-bus mode"
^ = for SDRAM chips connected to D[31:0] bus in "32-bit or 64-bit D-bus modes"

Bit 0: SDIU Enable (SE)

The SE enables and disables the SDRAM as follows:

Table D2–15. SDRAM Enable/Disable

SE Bit	Enable/Disable
0	Disable (default)
1	Enable

D2.2.3 Auto Refresh Timer Register

The Auto Refresh Timer (ART) Register is written with a value that generates an appropriate refresh interval for the SDRAM that is in use (see *SDRAM Auto Refresh* in Section D2.3).

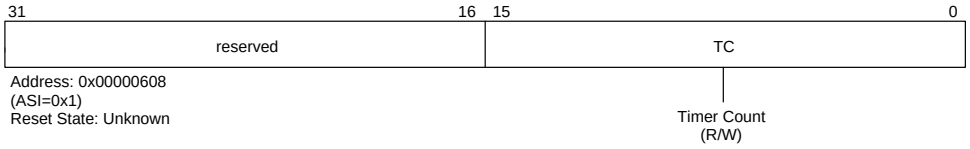


Figure D2–0. Auto Refresh Timer Register

Bits 31-16: Reserved
Bits 15-0: Timer Count (TC)

D2.3 SDIU Operation

When SCKE (SDRAM clock enable) is active (high), combinations of the –SRAS, –SCAS, and –SWE SDIU outputs, in conjunction with the –SCE signal at the rising edge of the clock, determine SDRAM operation. Signal 'CLK' in waveforms refer to MB86934 chip's internal clock.

Tables D2-16 shows the SDRAM commands and associated signals. Figure D2-4 shows a simplified SDRAM state diagram. This table and figure should be referenced for an understanding of the SDRAM commands and SDRAM operation.

Table D2–16. SDIU Commands (Ref: MB81116820 SDRAM)

COMMAND	SCKE	SDQM	A11	A10*	A9	A8-A0	–SCS	–RAS	–CAS	–SWE	Mnemonic
Mode Register Set (5)(6)	H	X	V	V	V	V	L	L	L	L	MRS
Auto Refresh(6)	H	X	X	X	X	X	L	L	L	H	REFR
Self-Refresh Entry(6)	L	X	X	X	X	X	L	L	L	H	SREN
Self-Refresh Exit	H	X	X	X	X	X	L	H	H	H	SREX
Precharge All Banks	H	X	X	H	X	X	L	L	H	L	PALL
Bank Active (RAS)(7)	H	X	V	V	V	V	L	L	H	H	ACTV
Write(8)	H	X	V	L	X	V	L	H	L	L	WR
Read(8)	H	X	V	L	X	V	L	H	L	H	RD
Data Write/Output Enable	H	L	X	X	X	X	X	X	X	X	ENBL
Data Mask/Output Disable	H	H	X	X	X	X	X	X	X	X	MASK

Notes:

(1) V = Valid, L = Logic Low, H = Logic High, X = Either Low or High.

(2) All commands are assumed to be valid state transitions.

(3) All inputs are latched on the rising edge of clock.

(4) Required after power-up.

(5) The MRS, REFR, SREN, and PD commands should be issued only after all banks have been precharged (PALL command). Refer to the State Diagram in Figure D2-4.

(6) The ACTV command should be issued only after the corresponding bank has been precharged (PALL command).

(7) The WR and RD commands should be issued only after the corresponding bank has be activated (ACTV command). Refer to the State Diagram in Figure D2-4.

SDIU After Reset

The SDIU is disabled after reset. Therefore, the user program must complete the following SDIU initialization sequence before executing SDRAM read or write commands following reset:

(1) Allow at least 2 ms of idle with a program loop.

(2) Write the SDIU Mode Register (ASI 0x1, addr. 0x600).

(3) Write the Auto Refresh Timer Register (ASI 0x1, addr. 0x608) with the auto-refresh timer count based on the SDRAM requirement. This count determines SDRAM refresh signal timing.

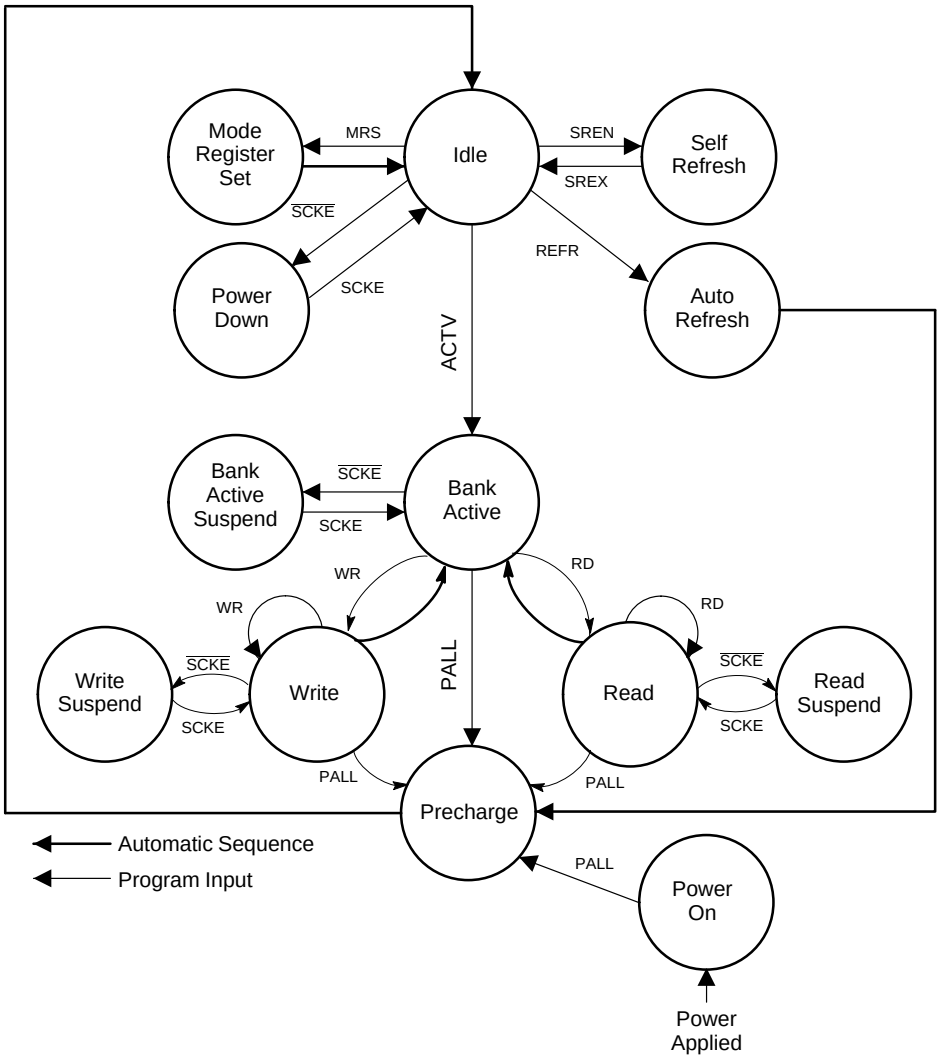


Figure D2–1. State Diagram

(4) Write the SDRAM Configuration Register (ASI 0x1, addr 0x604) to enable the SDIU, and to select the bus size and SDRAM type.

(5) Wait for a minimum of 8 auto refresh (REFR) commands.

The SDIU is then initialized for SDRAM read and write operations.

Figure D2-5 shows SDIU "mode register set" command timing.

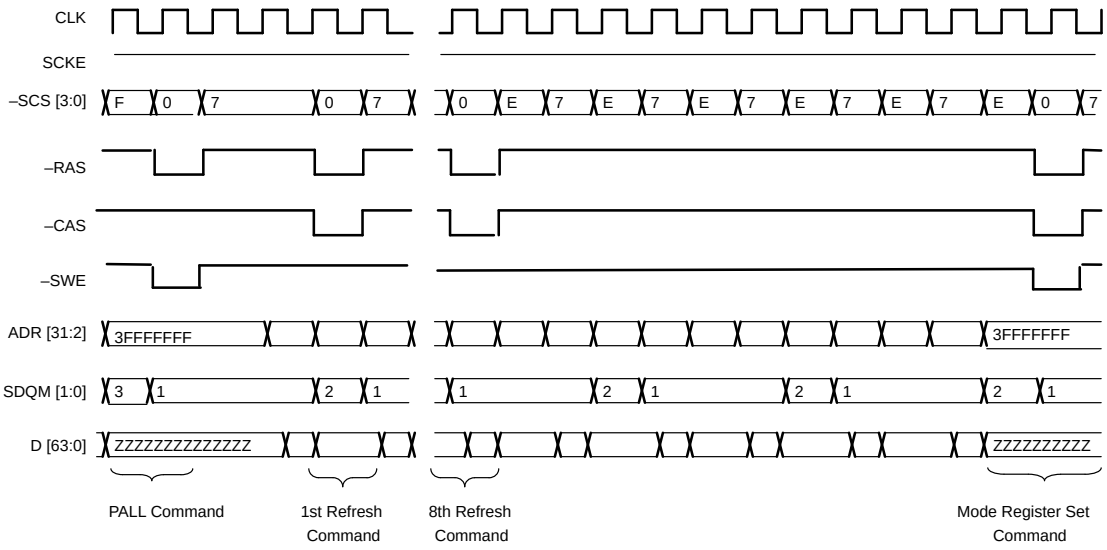


Figure D2–2. SDIU "Mode Register Set" Timing

SDRAM Auto Refresh

The SDRAM requires periodic refresh to avoid loss of data. For example, 16 MB SDRAM requires a refresh no less often than every 16 µs, or at least 4096 refreshes during each 65.6 ms period.

The Auto Refresh timer can be programmed to provide periodic refresh by writing the appropriate timer count for the SDRAM into the Auto Refresh Timer (ART) register. When the ART register is written, the counter loads the refresh count value from the ART register, then decrements the count with chips internal clock. The refresh signal for the SDRAM is asserted when the count reaches 0, at which time the counter reloads the refresh count value from the ART and repeats the refresh cycle.

SDRAM Power-Down

The SDIU places the SDRAM in the power-down mode (the self-refresh state) when bit 0 (SE bit) of the SDRAM Configuration Register is cleared to 0. The SDIU issues the Self Refresh Entry (SREN) command to place the SDRAM in the power-down mode. The SDRAM refreshes itself in the power-down mode.

The SDIU executes the Self Refresh Exit (SREX) command and exits the SDRAM power-down mode when bit 0 of the SDRAM Configuration register is set to 1.

D2.4 SDIU Data Transfer Operations

The SDIU accesses SDRAM through the Bus Interface Unit (BIU). It can transfer 32-bit single-word, 32-bit double-word, 64-bit single-word, and 64-bit double-word data. It also supports read-modify-write operations.

In 32-bit bus operations, data is transferred only on the lower 32 bits (D[31:0]) of the 64-bit data bus. The upper 32 bits (D[63:32]) are masked by the SDQM signals (SDQM[1:0] = 0x01). Only one data word is transferred on D[31:0] in single-word operations; two words are transferred sequentially on D[31:0] in double-word operations; four words are transferred sequentially during burst transfers.

In 64-bit operations, data can be transferred on the upper 32 bits of the data bus (D[63:32]), on the lower 32 bits of the data bus (D[31:0]), or on the entire data bus (D[63:0]). The upper 32 bits of data are masked by SDQM = 0x01, and the lower 32 bits of data are masked by SDQM = 0x10. Only one data word is transferred on either D[31:0] or D[63:32] in single-word operations; two words are transferred on D[63:0] in double-word operations, and during burst transfers.

Irrespective of type of bus configuration (32-bit or 64-bit) PARITY[3:0] bus is corresponding to D[63:48], D[47:32], D[31:16] and D[15:0] during SDRAM operations. In waveforms signal 'CLK' refers to MB86934 chip's internal clock.

D2.4.1 Page Hit/Miss Detection

Same Row Detection logic in the SDIU determines whether the SDRAM memory location being accessed is in the same row of the SDRAM as the location that was previously accessed. The system can use this feature to execute fast burst mode data transfers to/from locations that are in the same row of the SDRAM. If the current SDRAM access is in a different row (page), SDIU will detect page – miss.

D2.4.2 BIU Write Operations

The SDIU supports byte, half-word, single-word, and double-word writes. Byte- and half-word writes are executed with read-modify-write operations. Double-word writes occur when the BIU executes floating-point store double operations.

32-Bit Write Single — Page Hit

The operation begins with a valid address on the address lines. The WR command is asserted during cycle 0 to start the write cycle, and the data is asserted on the data bus. SDQM[1:0] is set to 0x01 in cycle 0 to mask the upper word (D[63:32]) and write the lower word (D[31:0]), and is set to 0x11 the following three cycles to disable SDRAM writes.

32-Bit Write Single — Page Miss

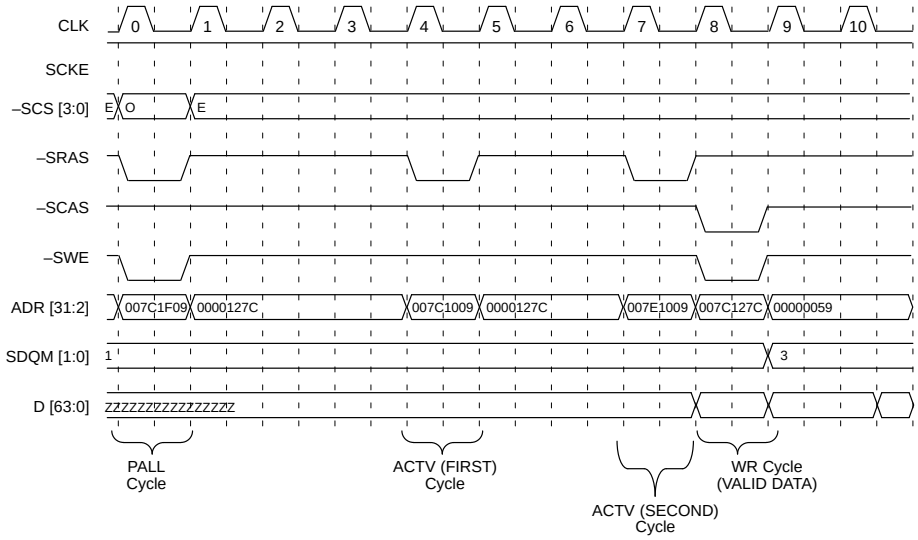


Figure D2–3. 32-Bit BIU Write Single–Page Miss (BIU to SDRAM)

The write operation begins with a PALL cycle, followed by two ACTV cycles and a WR cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and the bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with the row address and other bank select on the address lines. The WR command is asserted during cycle 8 to start the write cycle.

The rest of the operation is similar to the 32-bit write single page hit case.

32-bit Write Double — Page Hit

The write operation begins with a valid column address on the address lines. The WR command is asserted during cycle 0 to start the write cycle, and the data is asserted on the data bus. SDQM[1:0] is set to 0x01 during cycles 0 and 1 to mask the upper word (D[63:32]) and write the lower word (D[31:0]), and is set to 0x11 the following two cycles to disable SDRAM writes.

Figure D2-7 shows timing for a 32-bit double-word write with page hit.

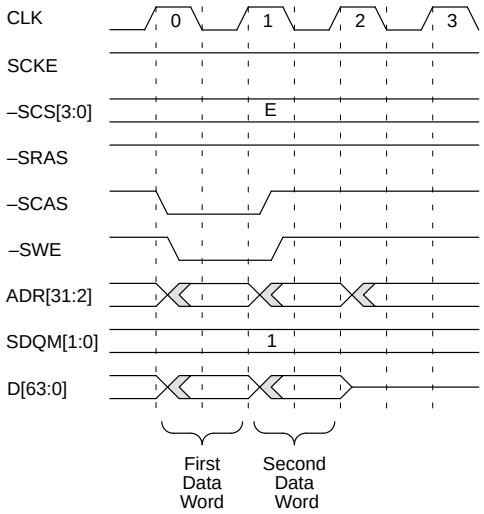


Figure D2–4. 32-Bit BIU Write Double — Page Hit (BIU to SDRAM)

32-bit Write Double — Page Miss

The write operation begins with a PALL cycle, followed by two ACTV cycles and a WR cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and the bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The WR command is asserted during cycle 8 to start the write cycle.

The rest of the operation is similar to the 32-bit write double page hit case.

64-bit Write Single — Page Hit

The write operation begins with a valid column address on the address lines. The WR command is asserted in cycle 0 to start the write cycle, and the data is asserted on the data bus. SDQM[1:0] is set to either 0x01 to mask the upper word (D[63:32]) and write the lower word (D[31:0]), or to 0x10 to write the upper word and mask the lower word. SDQM[1:0] is set to 0x11 during the following three cycles to disable SDRAM writes.

64-bit Write Single — Page Miss

The write operation begins with a PALL cycle, followed by two ACTV cycles and a WR cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The WR command is asserted during cycle 8 to start the write cycle.

The rest of the operation is similar to the 64-bit write single page hit case.

64-bit Write Double— Page Hit

The write operation begins with a valid column address on the address lines. The WR command is asserted in cycle 0 to start the write cycle, and the data is asserted on the data bus. SDQM[1:0] is set to 0x00 in cycle 0 to write the double word on D[63:0], and is set to 0x11 the following three cycles to disable SDRAM writes.

64-bit Write Double — Page Miss

The write operation begins with a PALL cycle, followed by two ACTV cycles and a WR cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and the bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and the other bank select on the address lines. The WR command is asserted during cycle 8 to start the write cycle.

The rest of the operation is similar to the 64-bit write double page hit case.

Figure D2-8 shows six words of write in 64-bit mode where the first three words are written in one page and the last three words are written in a different page.

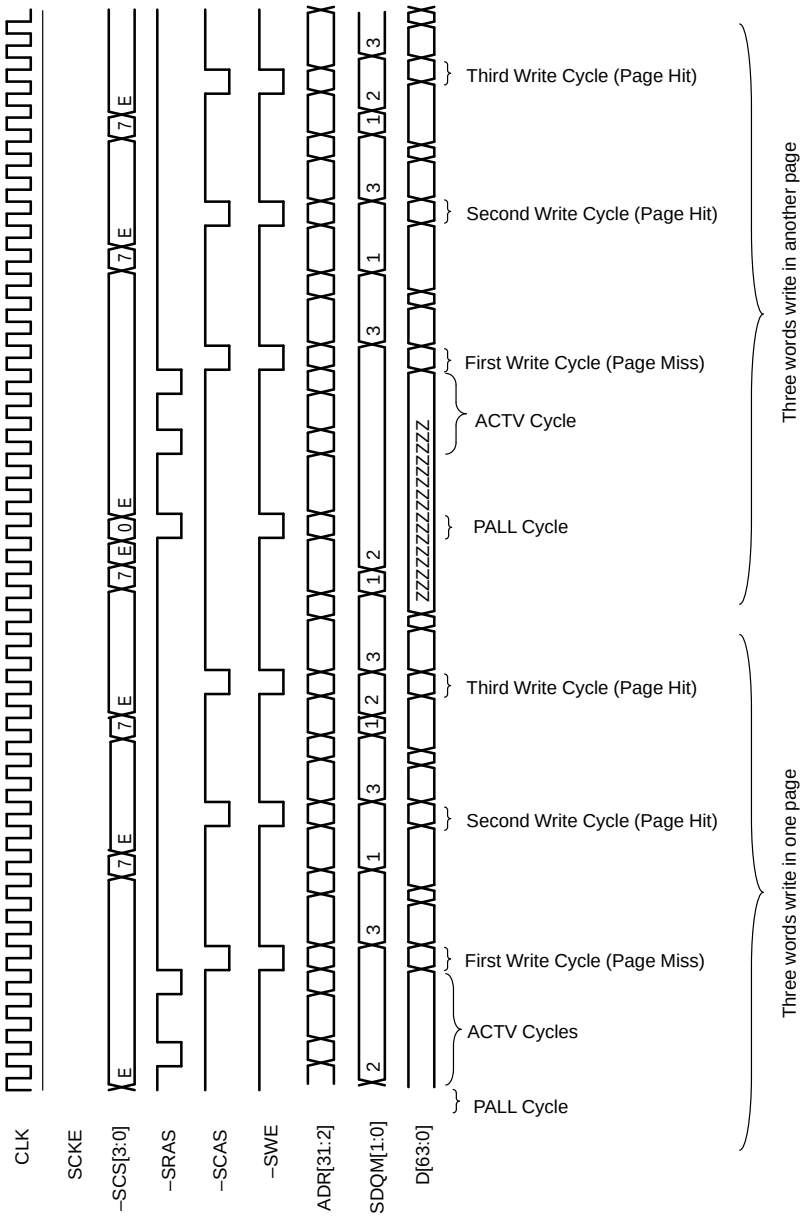


Figure D2–5. Three Words SDRAM Write in 64-Bit Mode in One Page Followed by Three Words SDRAM Write in 64-Bit Mode in Different Page

D2.4.3 BIU Read Operations

The SDIU supports one-word, two-word, and four-word reads. When the BIU executes a four-word read, the SDIU fetches four words from the SDRAM. The two-word read occurs when the BIU executes floating-point load double operations.

32-Bit Read Single — Page Hit

The operation begins with a valid address on the address lines. The RD command is asserted during cycle 0 to start the read cycle. SDQM[1:0] is set to 0x01 during cycle 1 to mask the upper word (D[63:32]) and read the lower word (D[31:0]), which is asserted on the data bus during cycle 3. SDQM is set to 0x11 to disable the SDRAM output drivers during cycles 4-6.

Figure D2-9 shows timing for a 32-bit single-word read with page hit.

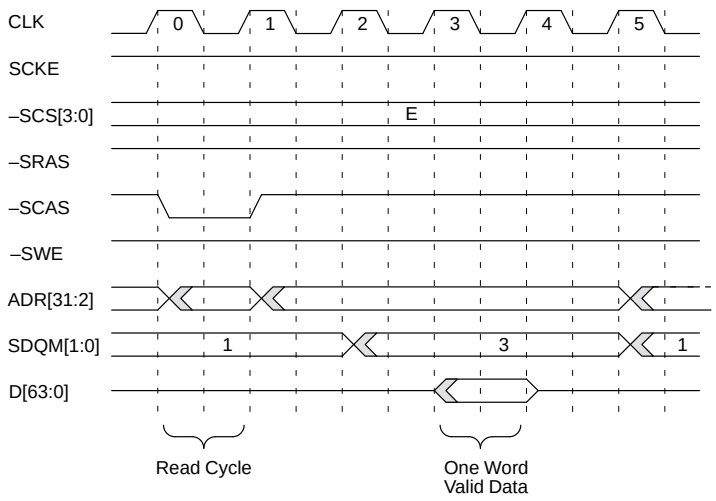


Figure D2–6. 32-Bit BIU Read Single — Page Hit (SDRAM to BIU)

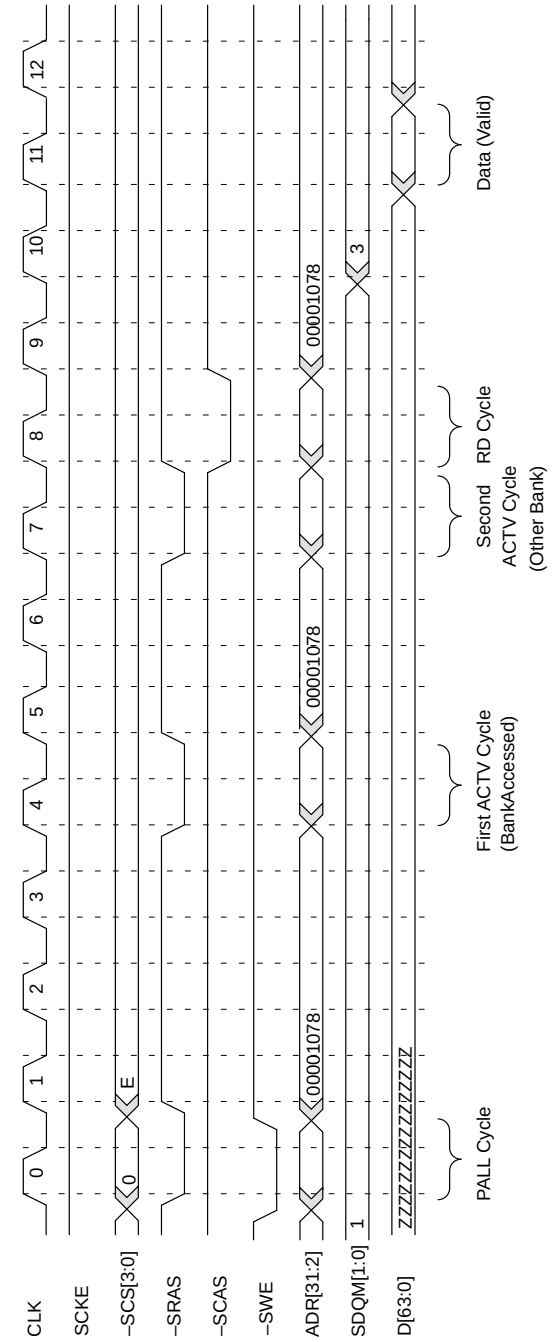


Figure D2–7. 32–Bit BIU Read Single–Page Miss (SDRAM to BIU)

32-Bit Read Single — Page Miss

The read operation begins with a PALL cycle, followed by two ACTV cycles and a read cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and the bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The RD command is asserted during cycle 8 to start the read cycle.

The rest of the operation is similar to the 32-bit read single page hit case.

Figure D2–10 shows timing for a 32-bit single-word read with page-miss.

32-Bit Read Double — Page Hit

The operation begins with a valid address on the address lines. The RD command is asserted during cycle 0 to start the read cycle. SDQM[1:0] is set to 0x01 during cycles 1 and 2 to mask the upper word (D[63:32]) and read the lower word (D[31:0]), which is asserted on the data bus during cycles 3 and 4. SDQM is set to 0x11 during cycles 3 and 4 to disable the SDRAM output drivers during cycles 5 and 6.

Figure D2-11 shows timing for a 32-bit double-word read with page hit.

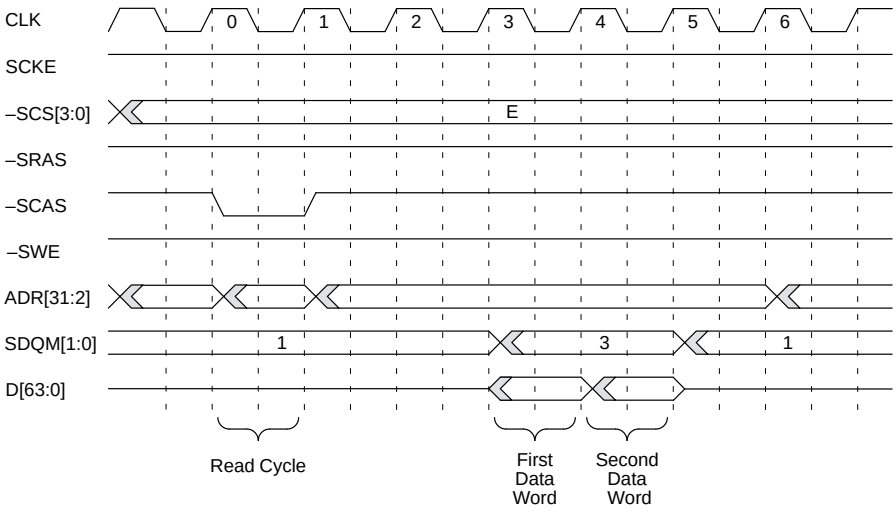


Figure D2–8. 32-Bit BIU Read Double–Page Hit (SDRAM to BIU)

32-Bit Read Double — Page Miss

The read operation begins with a PALL cycle, followed by two ACTV cycles and a read cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all

memory banks. The row address and the bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The RD command is asserted during cycle 8 to start the read cycle.

The rest of the operation is similar to the 32-bit read double page hit case.

32-Bit Burst Read — Page Hit

The operation begins with a valid address on the address lines. The RD command is asserted during cycle 0 to start the read cycle. SDQM[1:0] is set to 0x01 during cycles 1 through 4 to mask the upper word (D[63:32]) and read the lower word (D[31:0]), which is asserted on the data bus during cycles 3 through 6.

Figure D2-12 shows timing for a 32-bit burst read with page hit.

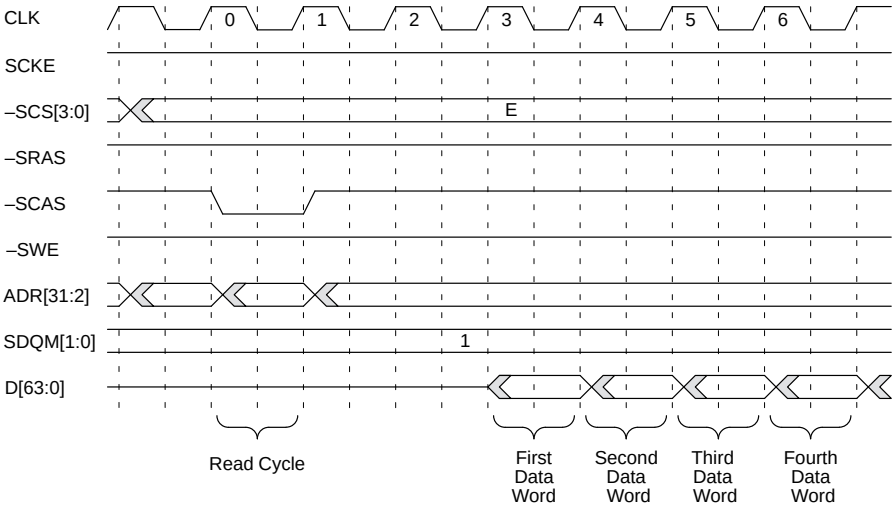


Figure D2–9. 32-Bit Burst Read–Page Hit (SDRAM to BIU)

32-Bit Burst Read — Page Miss

The read operation begins with a PALL cycle, followed by two ACTV cycles and a read cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and the bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The RD command is asserted during cycle 8 to start the read cycle.

Figure D2-13 shows timing for a 32-bit burst read with page miss.

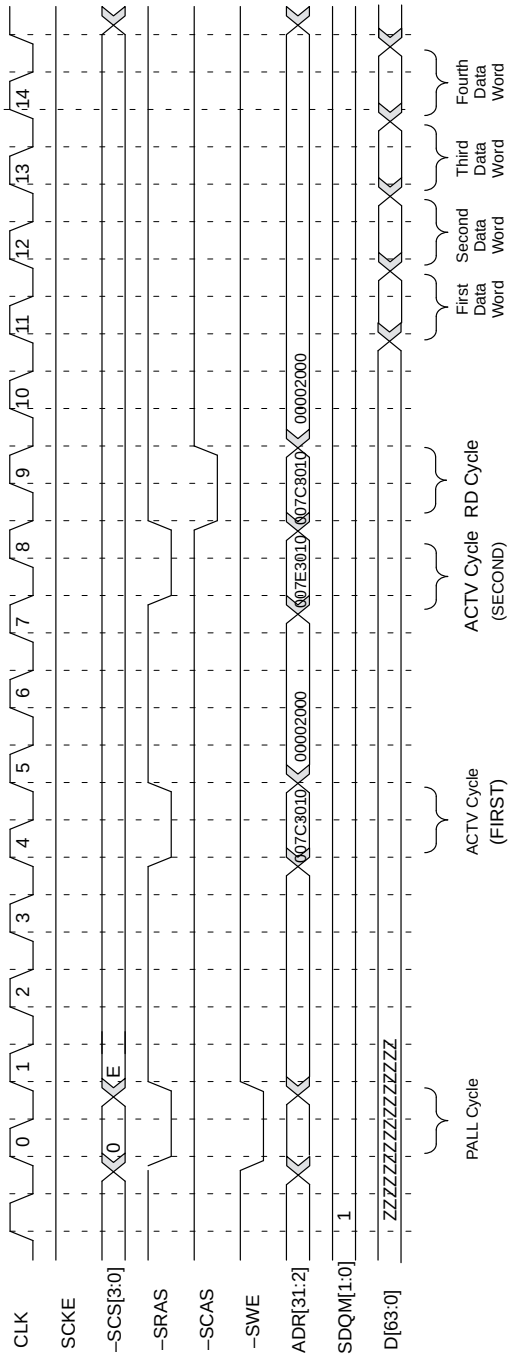


Figure D2–10. 32–Bit Burst Read–Page Miss (SDRAM to BIU)

64-bit Read Single — Page Hit

The read operation begins with a valid column address on the address lines. The RD command is asserted in cycle 0 to start the read cycle, and the data is asserted on the data bus during cycle 3. SDQM[1:0] is set to either 0x01 in cycle 1 to mask the upper word (D[63:32]) and read the lower word (D[31:0]), or to 0x10 to read the upper word and mask the lower word. SDQM is set to 0x11 during cycles 2-4 to disable The SDRAM output drivers during cycles 4-6.

64-Bit Read Single — Page Miss

The read operation begins with a PALL cycle, followed by two ACTV cycles and a read cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The RD command is asserted during cycle 8 to start the read cycle.

The rest of the operation is similar to the 64-bit read single page hit case.

64-Bit Read Double — Page Hit

The operation begins with a valid address on the address lines. The RD command is asserted during cycle 0 to start the read cycle. SDQM[1:0] is set to 0x00 during cycle 1 to read the double word, which is asserted on D[63:0] during cycle 3. SDQM is set to 0x11 during cycles 2-4 to disable the SDRAM output drivers during cycles 4-6.

64-Bit Read Double — Page Miss

The read operation begins with a PALL cycle, followed by two ACTV cycles and a read cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The RD command is asserted during cycle 8 to start the read cycle.

The rest of the operation is similar to the 64-bit read double page hit case.

64-Bit Burst Read — Page Hit

The operation begins with a valid address on the address lines. The RD command is asserted during cycle 0 to start the read cycle. SDQM[1:0] is set to 0x00 during cycles 1 and 2 to read the double word, which is asserted on D[63:0] during cycles 3 and 4. SDQM is set to 0x11 during cycles 3 and 4 to disable the SDRAM output drivers during cycles 5 and 6.

Figure D2-14 shows timing for a 64-bit burst read with page hit.

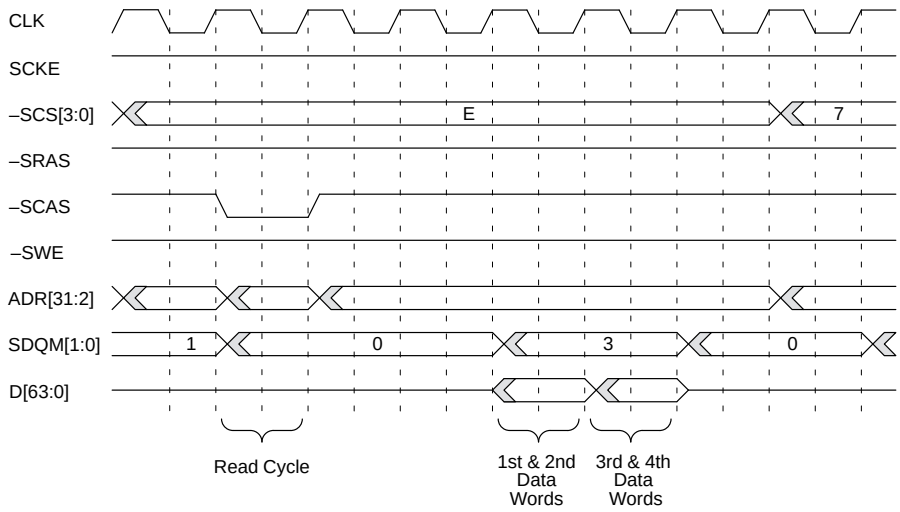


Figure D2–11. 64-Bit BIU Burst Read — Page Hit (SDRAM to BIU)

64-Bit Burst Read — Page Miss

The read operation begins with a PALL cycle, followed by two ACTV cycles and a read cycle. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on the address lines. The RD command is asserted during cycle 8 to start the read cycle.

The rest of the operation is similar to the 64-bit burst read hit case. Figure D2-15 shows timing for a 64-bit burst read with page miss.

Figure D2-16 shows timing for six words read from 64-bit mode where the first three words were read from one page and the last three words were read from a different page.

D2.4.4 Read-Modify-Write, byte and half-word — Page Hit

The operation begins with a valid column address on the address lines. The RD command is asserted during cycle 0 to start the read cycle. SDQM[1:0] is set during cycle 1 to either 0x01 to mask the upper word (D[63:32]) and read the lower word (D[31:0]), or to 0x10 to read the upper word and mask the lower word. The data is asserted on the data bus three cycles later, during cycle 3. SDQM[1:0] is set to 0x11 during cycles 2-4 to disable the SDRAM drivers during cycles 4-6.

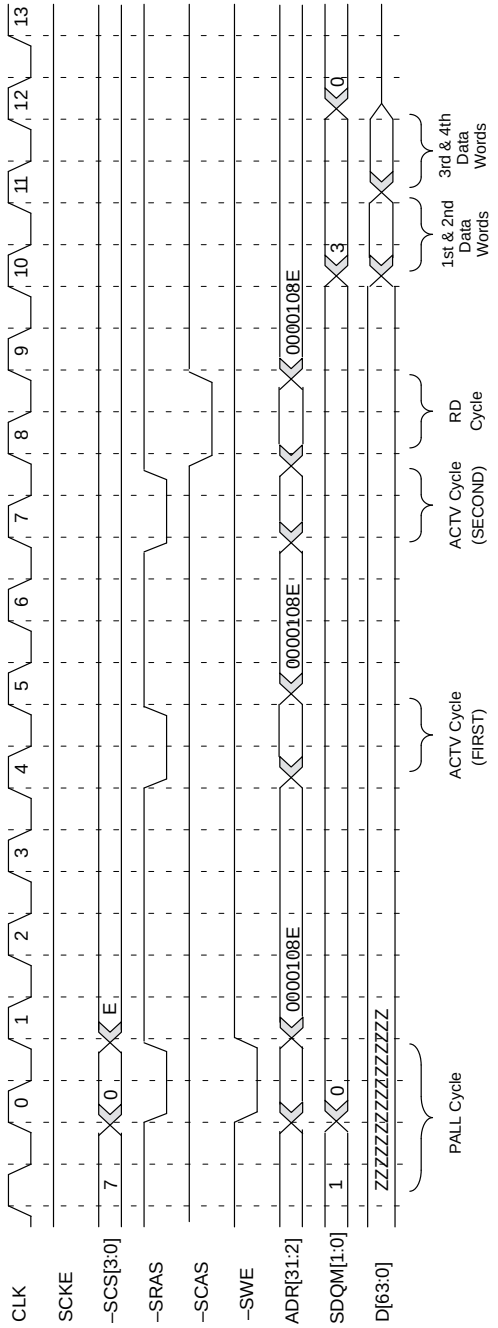


Figure D2–12. 64–Bit BIU Burst Read–Page Miss (SDRAM to BIU)

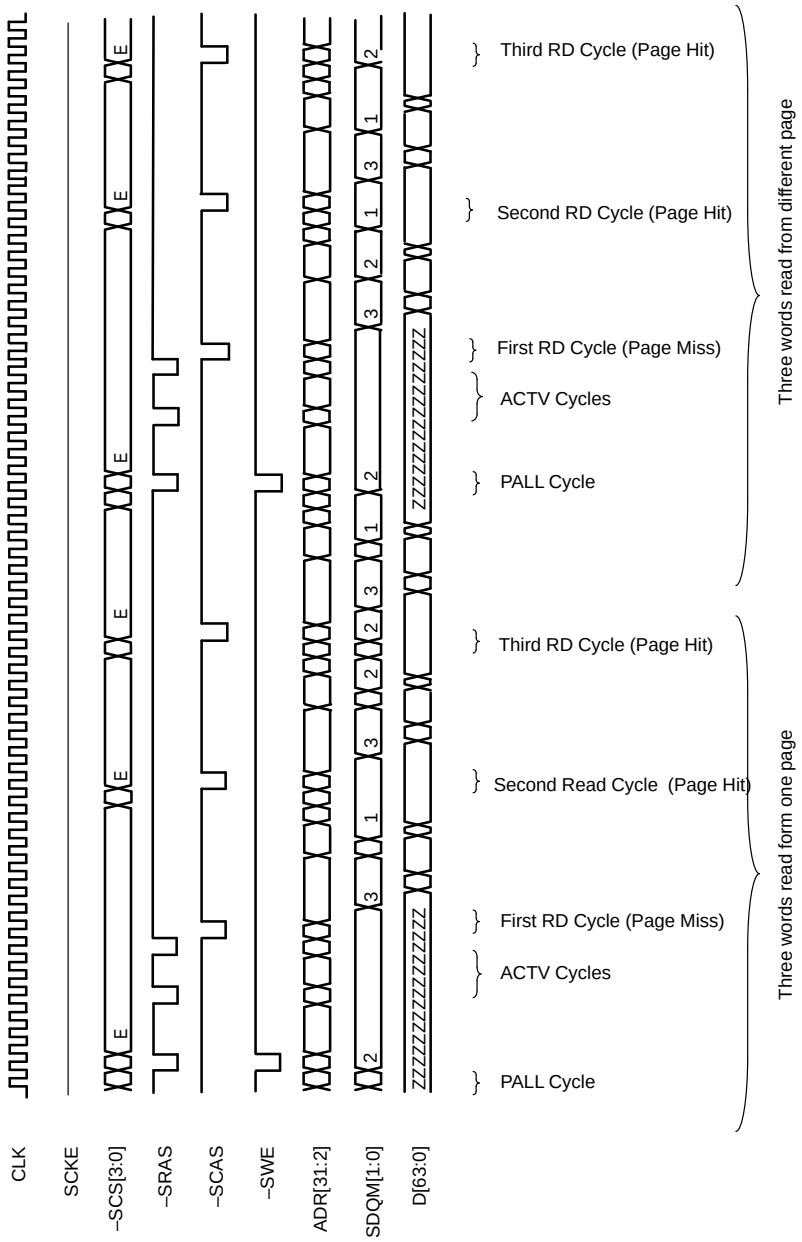


Figure D2–13. Three Words SDRAM Read in 64-Bit Mode in One Page Followed by Three Words SDRAM Read in 64-Bit Mode in Different Page

The RD command in the SDRAM is aborted when the WR command is issued during cycle 5. SDQM[1:0] is set during cycle 5 to either 0x01 to mask the upper word (D[63:32]) and write the lower word (D[31:0]), or to 0x10 to write the upper word and mask the lower word. The data to be written is asserted on the data bus during the same cycle (cycle 5). SDQM[1:0] is set to 0x11 during cycles 6-8 to disable SDRAM writes during these cycles.

Figure D2–17 shows timing for a read-modify-write with page hit.

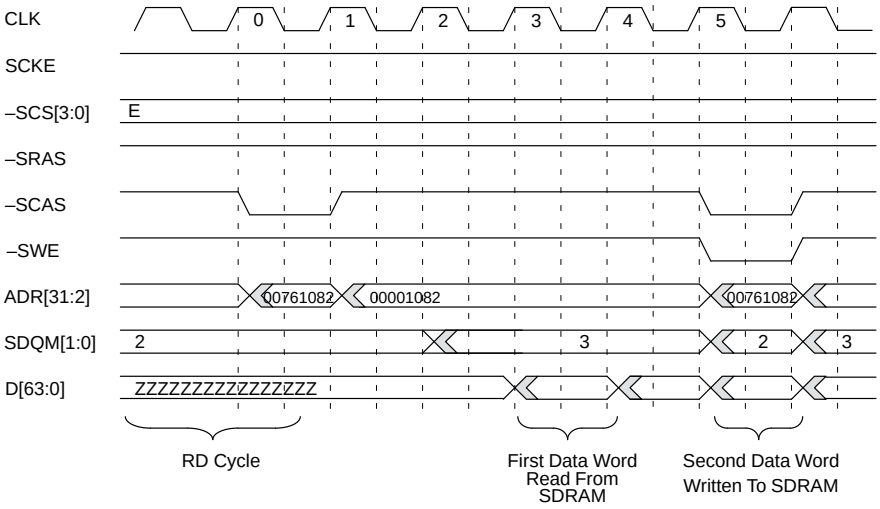


Figure D2–14. Read – Modify – Write – Page Hit

D2.4.5 Read-Modify-Write, Byte, Halfword–Page Miss

The operation begins with a PALL cycle, followed by two ACTV cycles. –SCS[3:0] is set to 0x0000 during the PALL cycle (cycle 0) to precharge all memory banks. The row address and bank select are asserted on the address lines during the first ACTV cycle (cycle 4). During cycle 7 second ACTV command is asserted with row address and other bank select on address lines. The RD command is asserted during cycle 8.

The rest of the operation is similar to the read-modify-write page hit case.

D2.4.6 FIFO-DMA-SDRAM Data Transfers

FIFO-DMA-SDRAM data transfers can be single-word transfers, and quad-word transfers. The SDRAM aligns the source and destination addresses. For single-word transfers, the addresses are aligned to word boundaries; for quad-word transfers, the addresses are aligned to quad-word boundaries.

DMA accesses SDRAM through the SDIU using the standard BIU-SDRAM read and write operations (see *BIU Write Operations* and *BIU Read Operations* in Sections D2.4.2 and D2.4.3).

D2.4.7 External Bus Master Single-Word Writes to SDRAM Through the SDIU — Page Miss

An external bus master requests the bus by asserting –BREQ low. The BIU grants the bus to the bus master by asserting –BGRNT low. The external bus master then asserts –AS low and, in the same cycle, asserts RD/–WR (low), ADR[31:2], ASI[3:0], and D[63:0]. The BIU figures out SDRAM access and drives the write signal active for SDIU. The SDIU initiates the two ACTV cycles, followed by the WR cycle. Data is written into the SDRAM during the WR cycle.

The SDIU then asserts the ready signal to the BIU which asserts –READYOUT low, indicating that the SDRAM write is complete. The external bus master then deasserts –BREQ by pulling it high, and the BIU deasserts –BGRNT during the next cycle by pulling it high.

Figure D2-18 shows timing for a single-word write by an external bus master.

D2.4.8 External Bus Master Single-Word Reads From SDRAM Through the SDIU — Page Hit

An external bus master requests the bus by asserting –BREQ low. The BIU grants the bus to the bus master by asserting –BGRNT low. The external bus master then asserts –AS low and, in the same cycle, asserts RD/–WR (high), ADR[31:2], and ASI[3:0]. The BIU figures out SDRAM access and drives the read signal active for SDIU. The SDIU initiates the RD cycle. The data is asserted by the SDRAM 3 cycles after the RD cycle.

The SDIU then asserts the ready signal to the BIU, which asserts –READYOUT low and drives data (which was read from the SDRAM) during the same cycle. The external bus master then deasserts –BREQ by pulling it high. Once –BREQ is deasserted, the BIU deasserts –BGRNT by pulling it high.

Figure D2-19 shows timing for a single-word read by an external bus master.

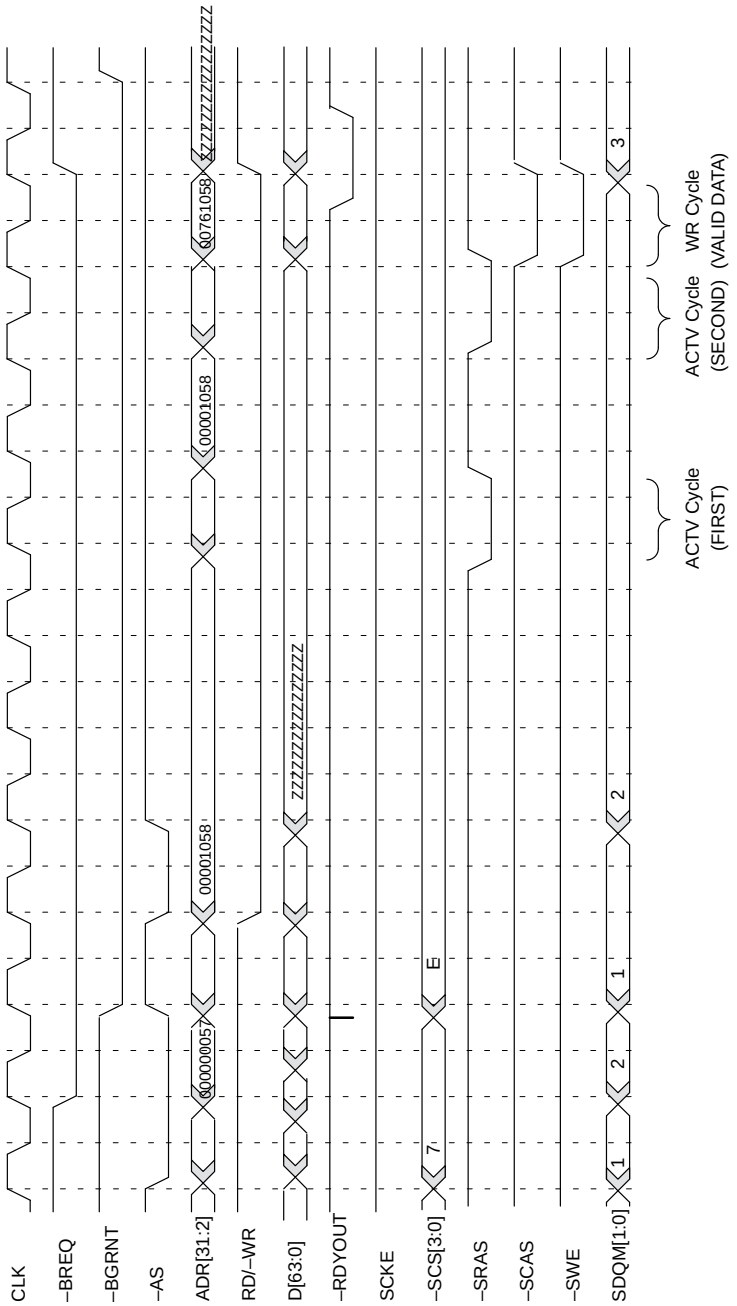


Figure D2–15. External Bus Master Single–Word Write–Page Miss

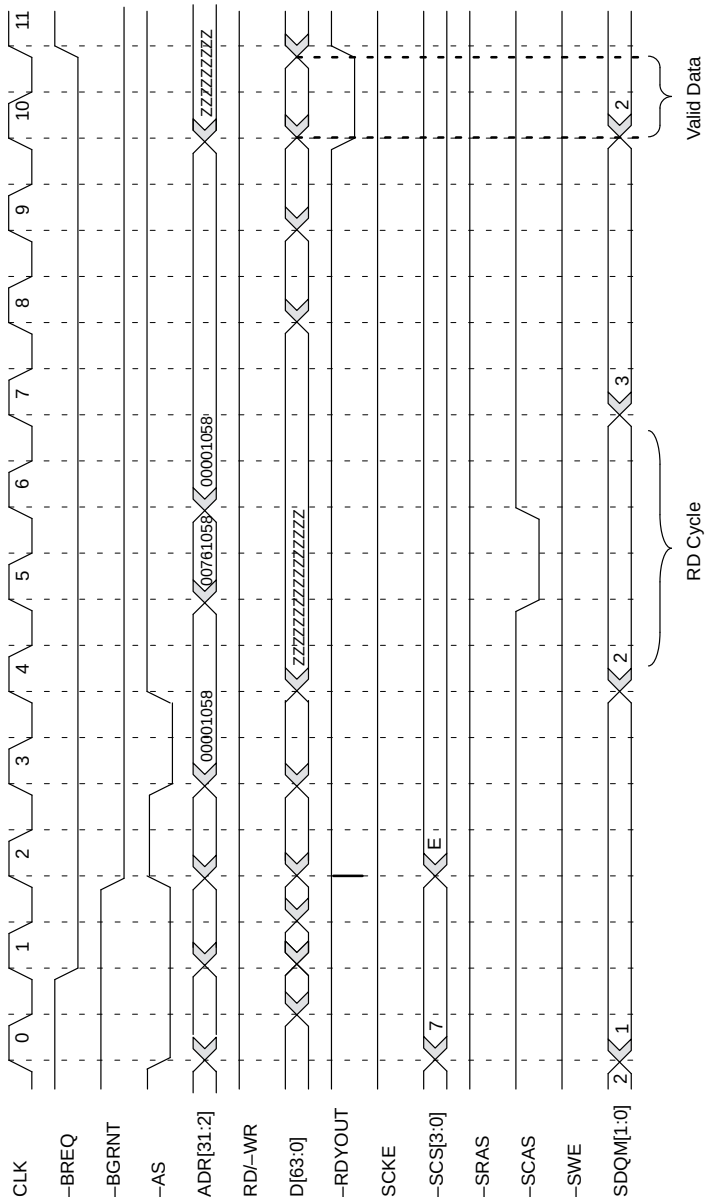


Figure D2–16. External Bus Master Single–Word Read–Page Hit

CHAPTER

D3

MB86934 Caches

D3.1 Overview of MB86934 Caches

The MB86934 offers enhanced support for cacheing: its instruction cache is 8K-bytes in size, and has 8-word lines. (The corresponding values for the MB86930 are 2K-bytes and 4-word lines.) The data cache of the MB86934 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).

D3.2 Programmer's Model

The cache control/status registers of the MB86934 form a superset of those in the MB86930. The registers common to the two chips are as follows:

Table D3–1. Control / Status Registers

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 MB86934 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.

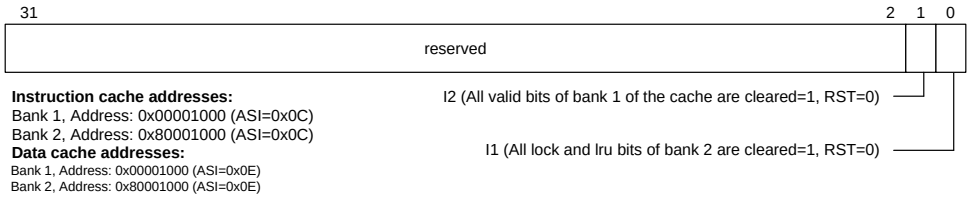


Figure D3–1. Cache Invalidate Register Format

D3.2.1 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.

D3.2.2 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.

D3.3 Internal Architecture of MB86934 Caches

Figure D3-3 shows cache operation (in the example shown, the Instruction Cache):

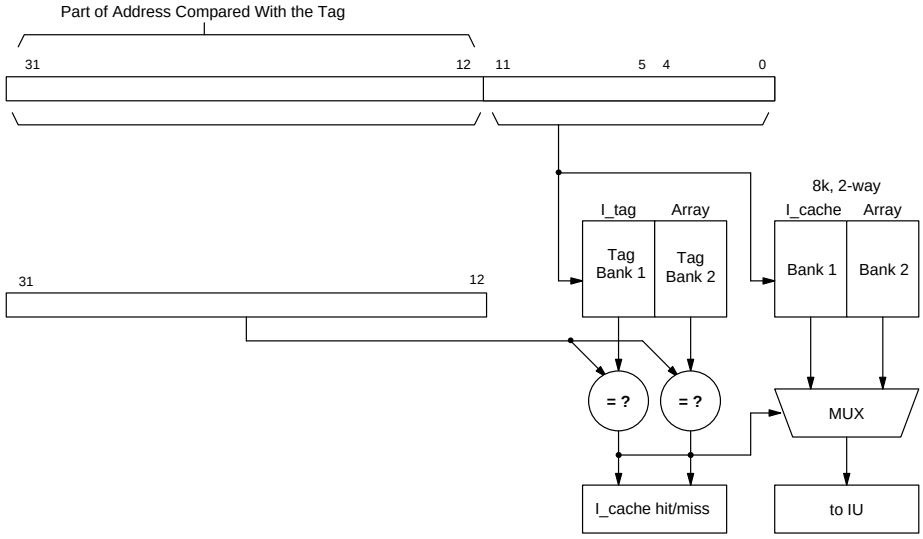


Figure D3–2. Cache Operation

D3.3.1 Instruction Cache

The instruction cache is an 8K-byte, 2-way associative, sectored cache, with 8-word lines. The basic operation of the cache is as follows: the IU sends the address to the I_cache, and I_cache tags. The lower 12 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 31-12 of the address to determine hit or miss.

Figure D3–3. Address to I_cache and Tag Array

The address coming out of the IU goes to the I_cache and tags. Bits 31-12 go to the tag array for comparison. Bits 11-5 select two tags (one for each bank) out of the 256-entry tag array, and also choose two lines (one for each bank) out of the 8K I_cache. Bits 4-2 select a word out of the 8-word line.

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

Valid bits have been added for the other four words of the 8-word line. Bit 5 is used to indicate whether the line can be accessed by supervisor only. Bit 1 is the least-recently used bit, which is used when doing a line replacement in the I_cache. Note that because of the increase in cache size and line size, the tag format of the MB86934 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

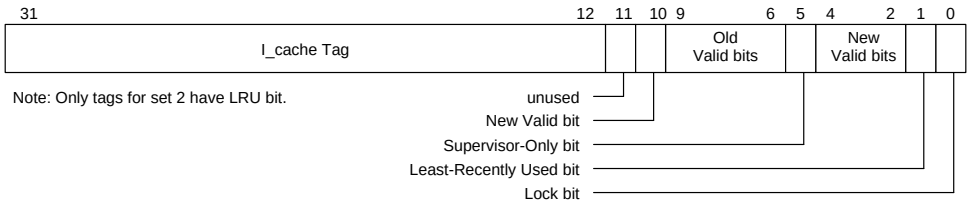


Figure D3–4. I_cache Tag Format

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.

D3.3.2 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 31-12 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."

D3.3.3 Miss Processing

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

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

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

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

D3.3.4 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.

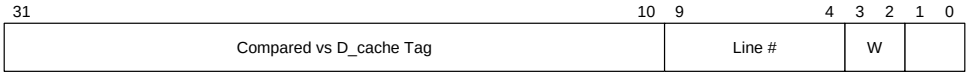


Figure D3–5. Address to D_cache and Tag Array

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.

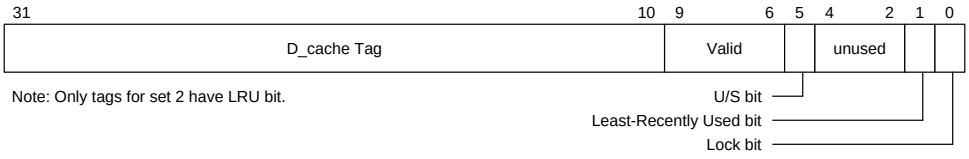


Figure D3–6. D_cache Tag Format

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.

D3.3.5 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.

D3.3.6 Write Hit

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

D3.3.7 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."

D3.3.8 Atomic Load and Store

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

CHAPTER

D4

MB86934 DMA

D4.1 Overview

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

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

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

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

The two figures that follow give, respectively, an overall picture of the relationship of the DMAC to other major functional components of the MB86934, and a detailed picture of the flow within the DMAC.

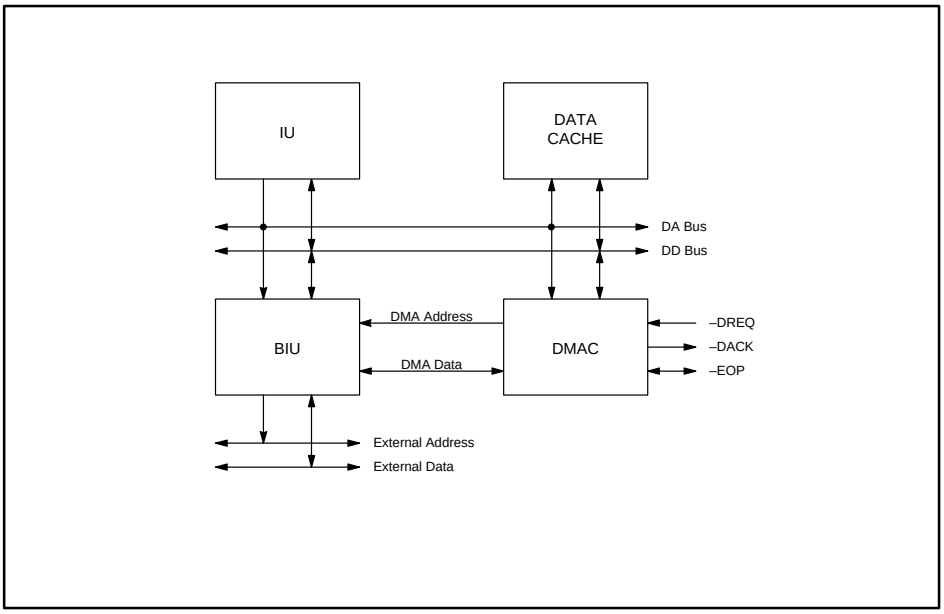


Figure D4-1. Relation of DMAC to Other Major Components

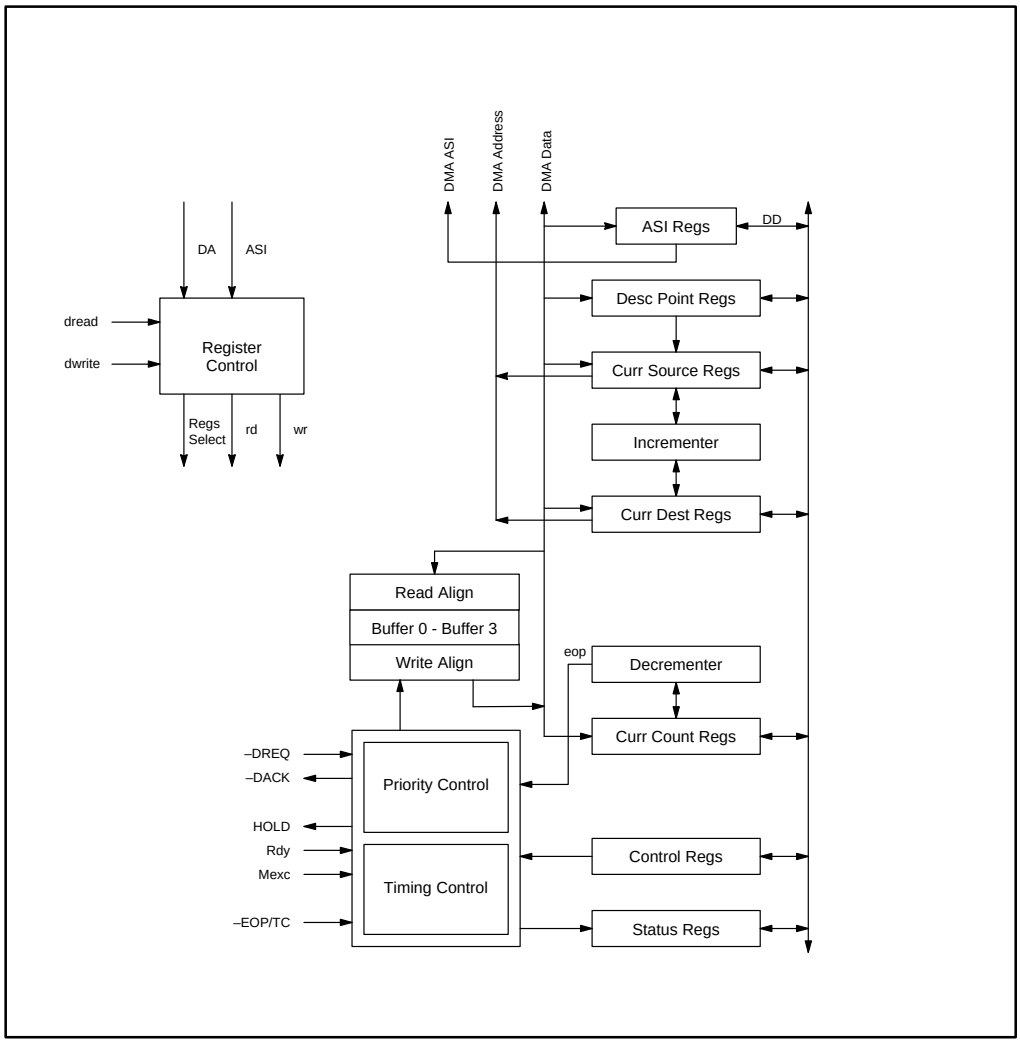


Figure D4-2. DMA Block Diagram

D4.2 Programmer's Model

Table D4-1. DMA Signal Descriptions

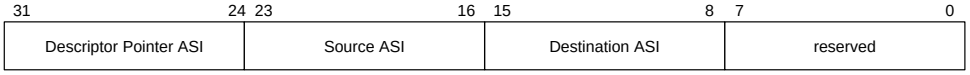
Signal	Function
–DREQ1 / –DREQ0	**DMA REQUEST (I):** This input signal indicates that an external device is requesting DMA transfer. It is an edge-sensitive signal for single transfer, and a level-sensitive signal for demand transfer.
–DACK1 / –DACK0	**DMA ACKNOWLEDGE (O):** This output signal is sent to the external device to acknowledge the DMA request, and is active when the requesting device is accessed.
–EOP1 / –EOP0	**END OF PROCESS (I/O):** This pin is used as input when an external device wants to cause the DMA process to terminate. It functions as output when the byte count reaches zero. When not active, –EOP output will be tristated. For signalling the Terminal Count (TC) , –EOP will be pulled down, and then be pulled up for one cycle. A high impedance internal pull up is used to hold the signal high when –EOP is tristated. The –EOP issued by the DMAC can be used as input to the interrupt controller. If –EOPx is asserted by the external device, channel x will be disabled. Reprogramming is needed to enable a channel.
D<63:0>	**DATA BUS (I/O):** Bits 31–0 are used for word sized transfers, Bits 31–16 are used for halfword transfers, and only Bits 31–24 are used for Byte transfers.

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

D4.2.1 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.

D4.2.2 DP/Source/Destination ASI Register



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

Figure D4-4. DP/Source/Destination ASI Register

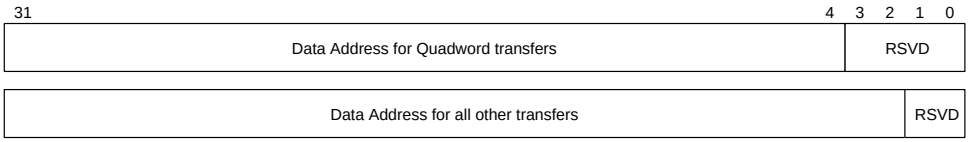
Bits 31-24: Descriptor Pointer ASI (DP ASI)—ASI of the Descriptor Pointer, a register used in buffer-chaining mode. It points to the next element of the linked list whose elements describe the source and destination of the DMA transfer.

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

D4.2.3 Current Source Address Register

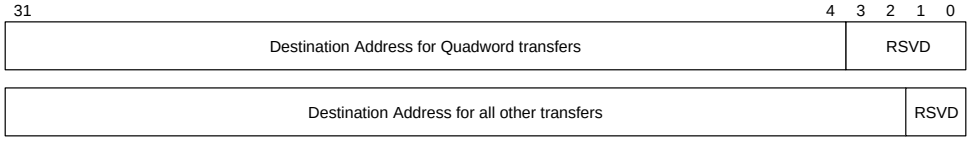


Address:0x00000184 (DMA0) (ASI=0x01)
0x000001A4(DMA1)

Figure D4-5. 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.

D4.2.4 Current Destination Address Register

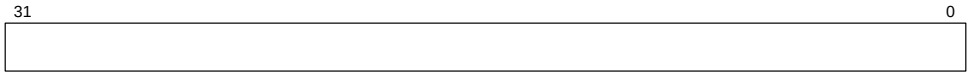


Address:0x00000188 (DMA0) (ASI=0x01)
0x000001A8 (DMA1)

Figure D4-6. 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.

D4.2.5 Current Byte Count Register

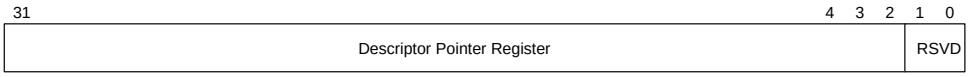


Address:0x0000018C (DMA0) (ASI=0x01)
0x000001AC (DMA1)

Figure D4-7. 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.

D4.2.6 Descriptor Pointer Register



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

Figure D4-8. 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.

D4.2.7 Channel Control Register

Bits 31:21 and bit 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.

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

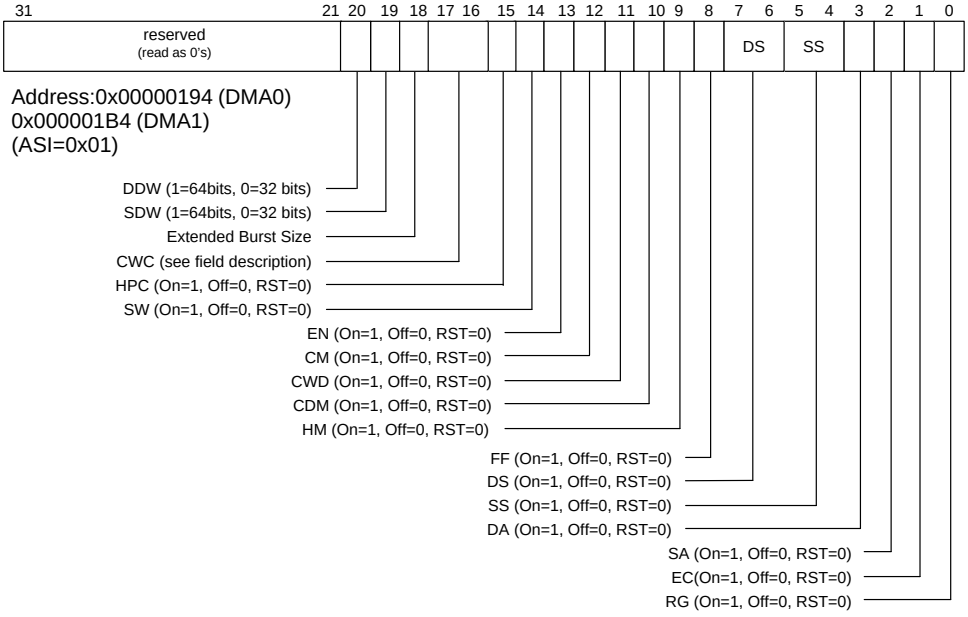


Figure D4-9. Channel Control Register

Bits 31-21 Reserved

Bit 20: Destination Data Width (DDW)—Sets the data width of the DMA transfer destination. This bit should be set to 1 if the DMA destination transfers 64-bit data, and set to 0 if the DMA destination transfers 32-bit data.

Note:
For flyby data transfers, the source width must be the same as the destination width.

Bit 19: Source Data Width (SDW)—Sets the data width of the DMA transfer source. This bit should be set to 1 if the DMA source transfers 64-bit data, and set to 0 if the DMA source transfers 32-bit data.

Note:
For flyby data transfers, the source width must be the same as the destination width.

Bit 18: Extended Burst Size. This bit is used in conjunction with the size bits. If this bit is 0, then the size bits act normally. If this bit is 1 and a flyby DMA transfer is set up, then the size bits take on the following meanings:

Size Bits	**Burst Size**	
00	8-word	(32-bit words)
01	16-word	
10	32-word	
11	64-word	

Note:

In flyby mode, the source-size and destination-size bits should have the same value. Also note that in the 86934 the only case where extended-burst size can be used is for DMA between SDRAM and the floating-point FIFOs (the BIU can only support butst up to 4-words). If the bus width is 64 bits, then size 00 corresponds the 4 64–bit words.

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 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.

D4.2.8 Channel Status Register

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.

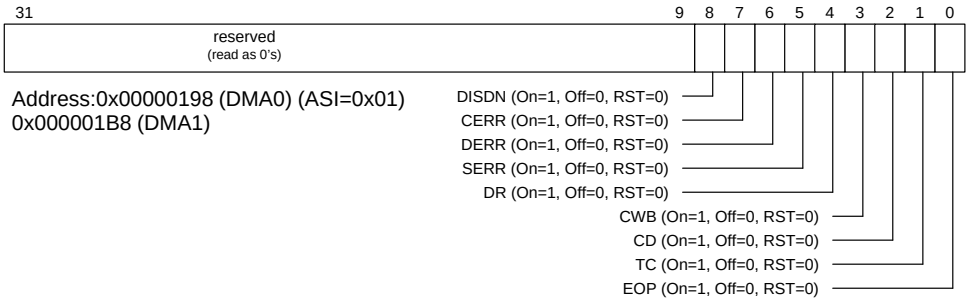


Figure D4-10. Channel Status Register

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.

D4.2.9 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.

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

D4.2.10 Buffer Chaining Data Structure

Each chaining–descriptor consists of 5 32–bit words. These words are loaded directly into the DMA registers from memory and are formatted exactly like the corresponding DMA registers.

- 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.

D4.2.11 DMA Initiation

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.

D4.2.12 DMA To/From the Floating-Point FIFO

The DMA controller can directly transfer data between memory and the floating-point FIFOs. To initialize a FIFO-DMA transfer, set the source address to the address of the data in memory, and set the destination address to the FIFO address. For DMA from memory to FIFO, set the DMAC to destination-request mode; and for DMA from FIFO to memory, set the DMAC to source-request mode.

The FIFO can be used as either a 32-bit wide or a 64-bit wide device, and DMA to and from the FIFO can occur only in flyby mode. Byte and half-word transfers to and from the FIFO are not supported.

D4.2.13 DMA To/From the SDRAM

The DMA controller can transfer data to and from the SDRAM in either flyby mode or flowthru mode. All transfers to or from the SDRAM must be specified as 64-bit wide data. (In flowthru mode, the SDRAM side of the transfer must be 64 bits wide, but the other side of the DMA transfer can be any width.)

To initialize the DMA controller for an SDRAM transfer, just initialize the registers as if you were setting up a memory-to-memory transfer, with either the source or destination address register pointing into SDRAM space.

For example, to perform a DMA transfer between the FP FIFO and SDRAM, set both the source and destination width to 64 (in DMA control register), and use flyby transfer mode. To perform a DMA transfer from DRAM to SDRAM, set the source data width to 32 bits (DRAM), and the destination data width to 64 bits (SDRAM), and use flowthru mode.

D4.2.14 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.

D4.2.15 Error Conditions

Memory Access Exceptions:

- Source Transfer Exception
- Destination Transfer Exception
- Chaining Exception

When an Error condition occurs, the relevant bits in the Status Register will be set up, and –EOP will be asserted. For example, if a MEXC or external EOP occurs during a read operation, the source transfer exception bit will be set.

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.

D4.3 External Interface

D4.3.1 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 D4-11. 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 D4-13 below, showing a quadword transfer taking four data cycles. The last DMA transfer is accompanied by EOP.

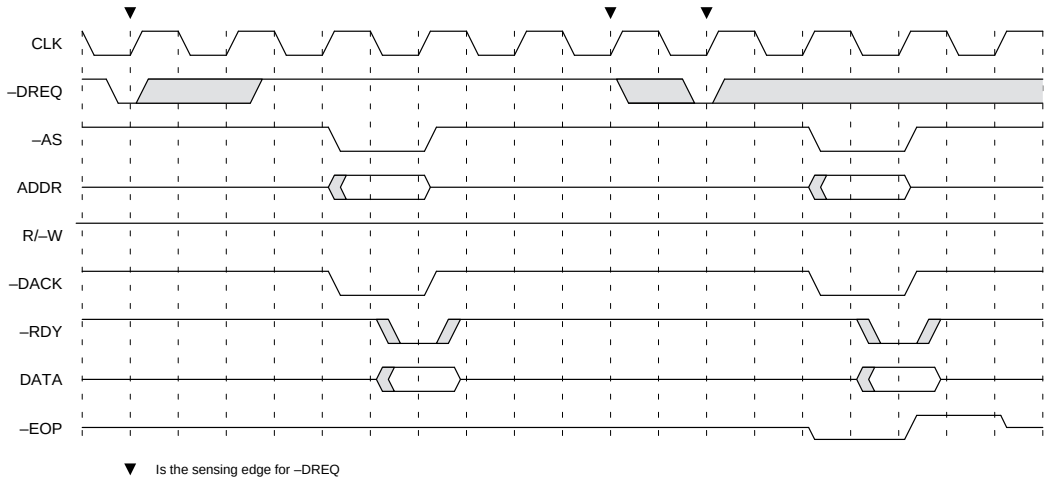


Figure D4-11. Single Transfer, Edge-Sensitive, Flyby (R/–W high)

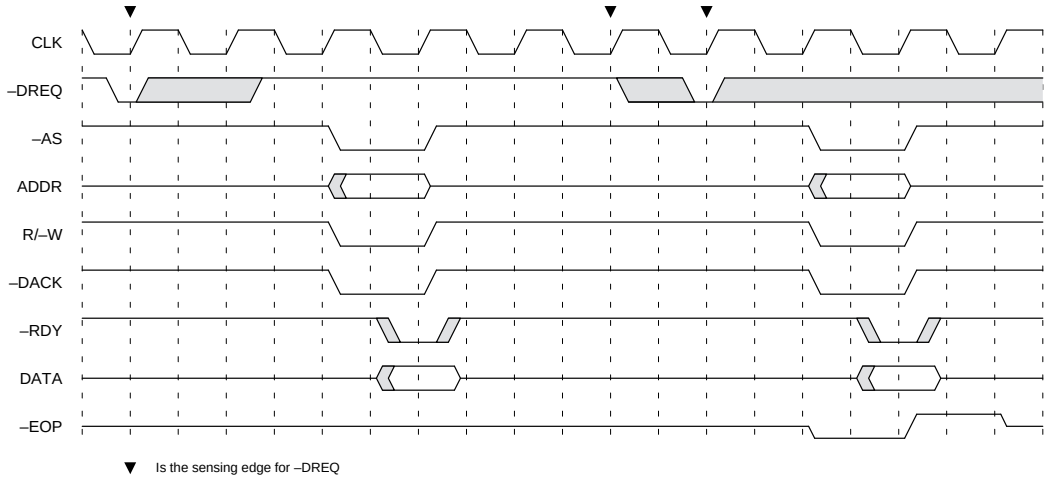


Figure D4-12. Single Transfer, Edge-Sensitive, Flyby (R/–W low)

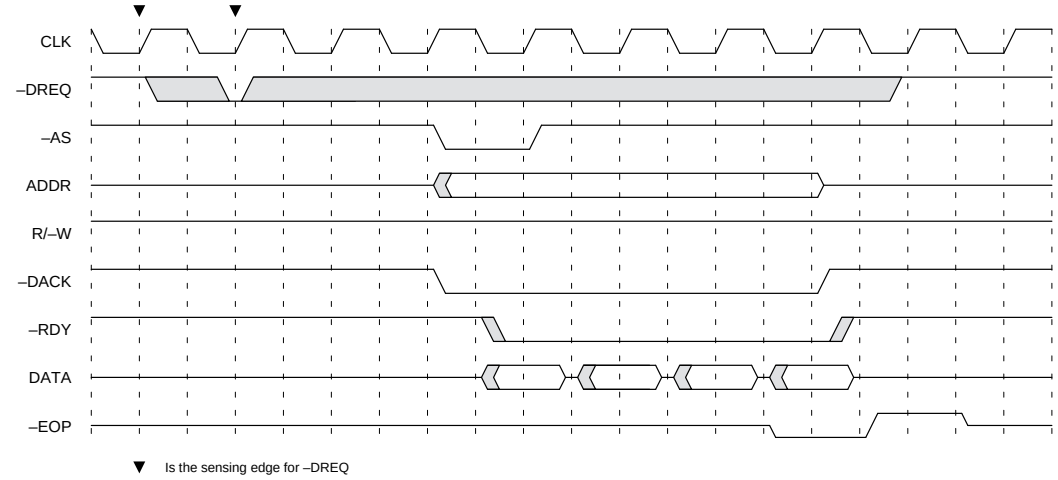


Figure D4-13. Single Transfer, Edge-Sensitive, Flyby, Quadword (R/–W high)

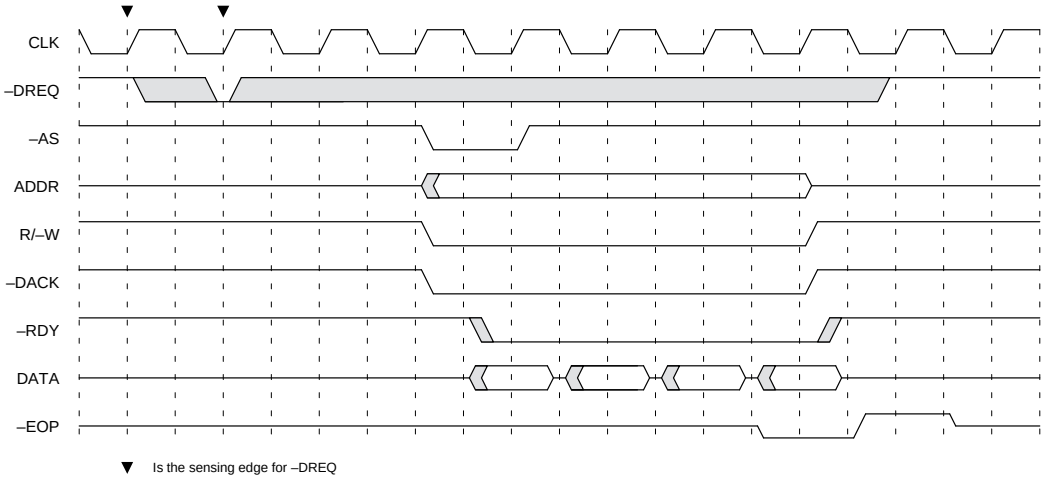


Figure D4-14. Single Transfer, Edge-Sensitive, Flyby, Quadword (R/–W low)

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 D4-15 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.

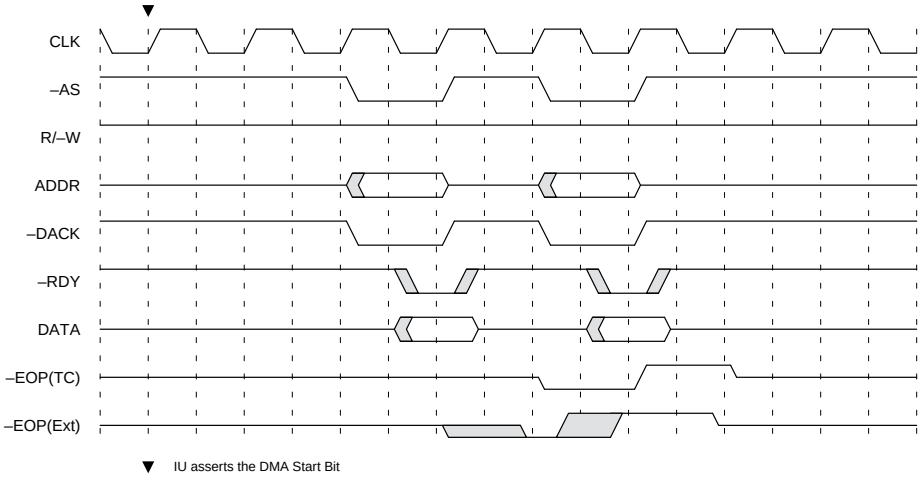


Figure D4-15. Block Transfer, Flyby (R/–W high)

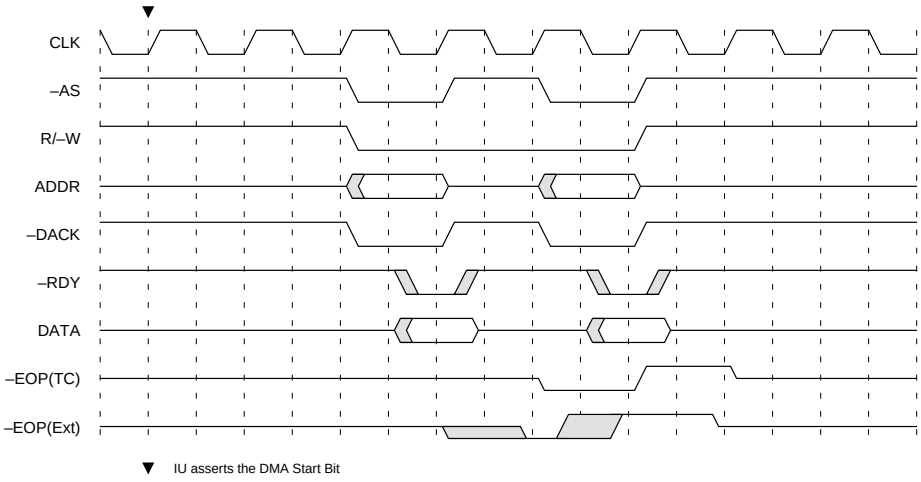


Figure D4-16. Block Transfer, Flyby (R/–W low)

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 D4-17. When a request for a demand transfer is made, the DMAC will look at the –DREQ to see if any request is pending.

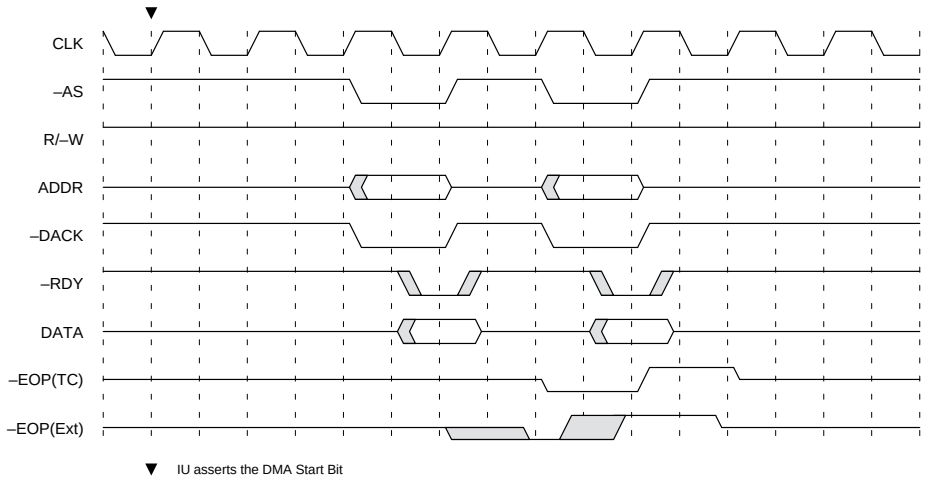


Figure D4-17. Demand Transfer, Flyby (R/–W high)

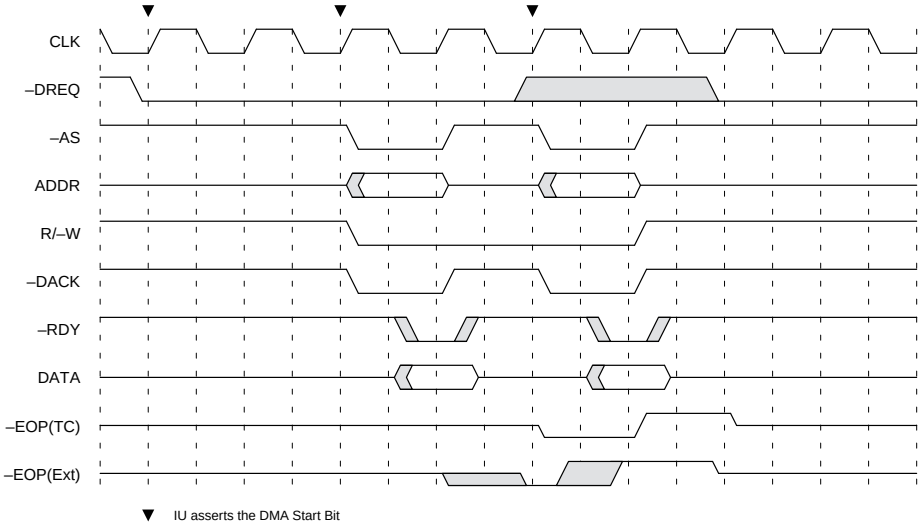


Figure D4-18. Demand Transfer, Flyby (R/–W low)

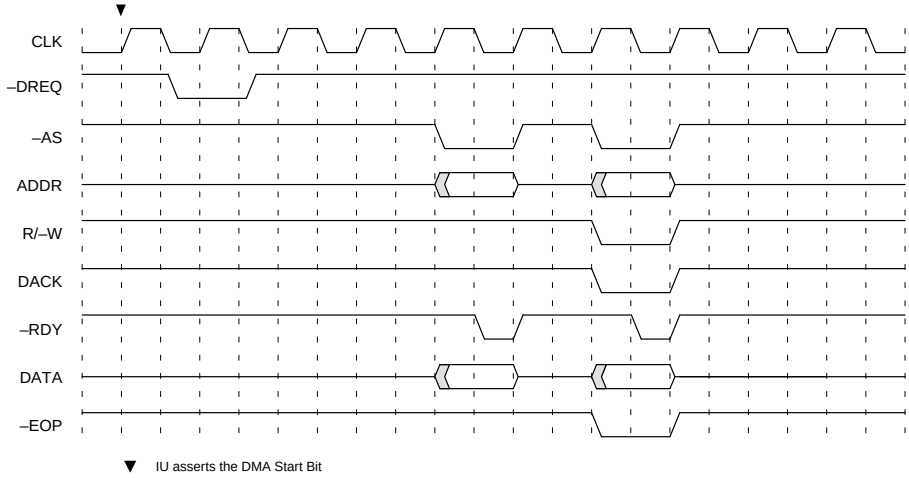


Figure D4-19. Single Transfer, Edge Sensitive Flow Through, Destination Request

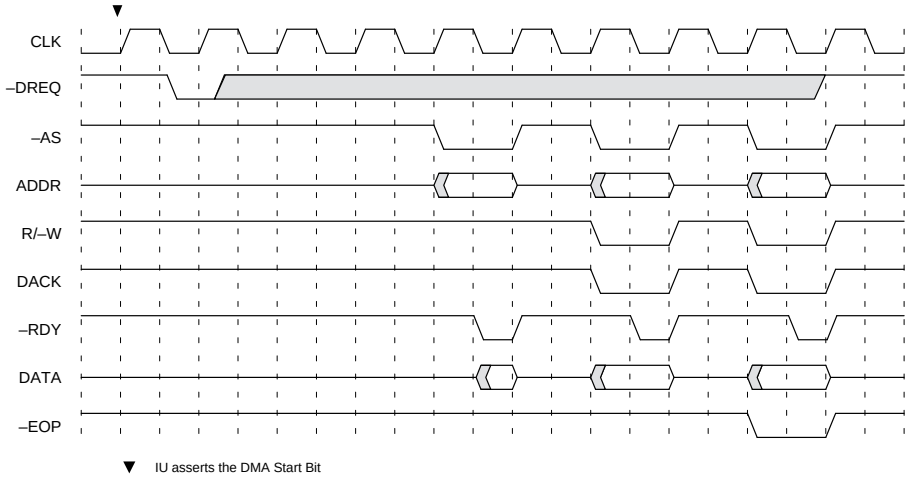


Figure D4-20. Demand Transfer, Flow Through, Word to Half–Word, Destination Request

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 byte, 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.

In single and demand transfer modes, if the source and destination size are not equal and flowthrough DMA is being used, one active pulse on DREQ will cause the DMA controller to perform one full read–write cycle. This means that if, for example, the source size is byte and the destination size is word, when DREQ is asserted, the DMA controller will immediately perform four reads (reading bytes), and then write one word. Note also that all four bytes will be expected on the Data bus pins 31:24, and that the Address pins 31:2 will be incremented after each byte is read (not Address pins 31:0).

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 trans-
                                    fer is completed

                           Chaining wait function set in Control
                                    Register and the TC (byte
                                    count expires)

                           Error on abnormal read/write transfer

                           If chaining debug mode is set in the
                                    control register, –EOP will
                                    be asserted at the end of
                                    each transferred block.
```

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

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

Memory Exception

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

CHAPTER

D5

Floating-Point Unit

D5.1 Overview of the MB86934 Floating-Point Unit

The MB86934 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 MB86934 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.

The FPU also executes Enhanced Floating-Point operations that are newly defined for the SPARClite FPU to access the FIFOs. These operations can read operands from the FIFOs and write their results to the FIFOs instead of the floating-point registers.

D5.2 FPU Data Formats

The MB86934 architecture recognizes three floating-point data formats:

- Floating-Point Single
- Floating-Point Double
- Floating-Point Quad

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

Figure D5-1 shows the floating-point data formats and the subwords within each format. Table D5-1 shows the subformat arrangements in memory, and in the processor registers. Tables D5-2 through D5-4 define the formats.

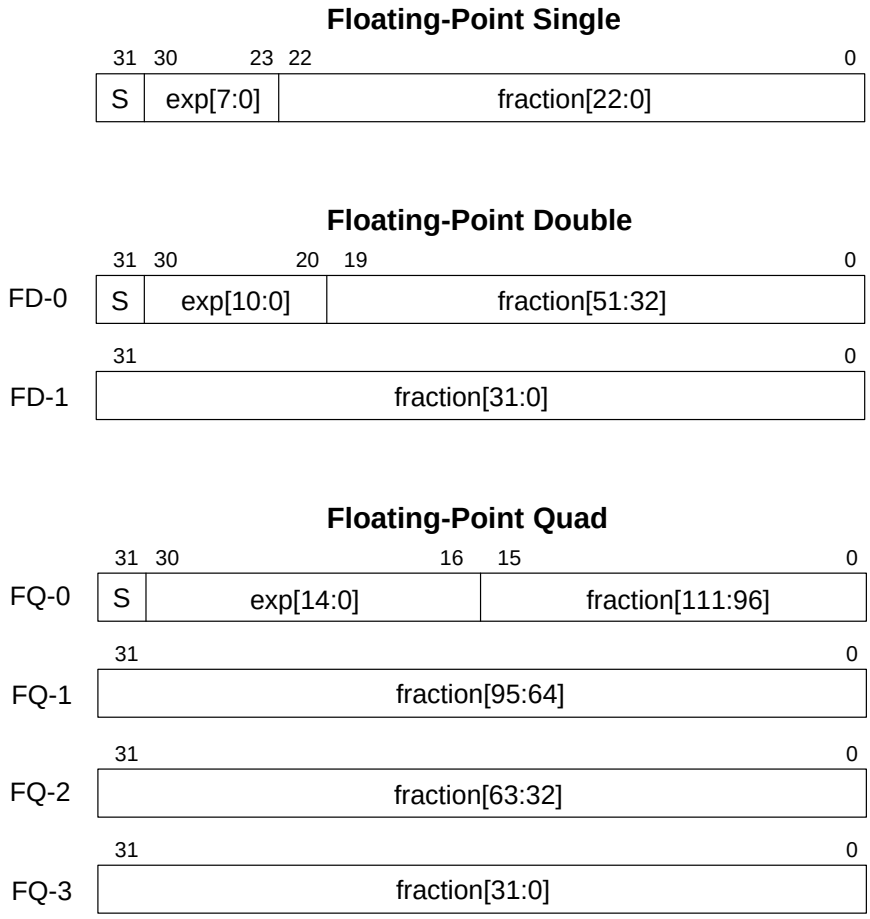


Figure D5–1. Data Formats

Table D5–1. 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 D5–2. 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 D5–3. 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 D5–4. 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

D5.3 FPU Registers

The FPU contains one register and two sets of registers: the Floating-Point State Register (FSR), the Enhanced f Register Set (f registers and FIFOs), and the Floating-Point Deferred-Trap Queue (FQ). In addition, three flags in the Processor State Register (PSR) and the Ancillary State Register 17 (ASR17) enable and disable the FPU and its FIFOs.

D5.3.1 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.

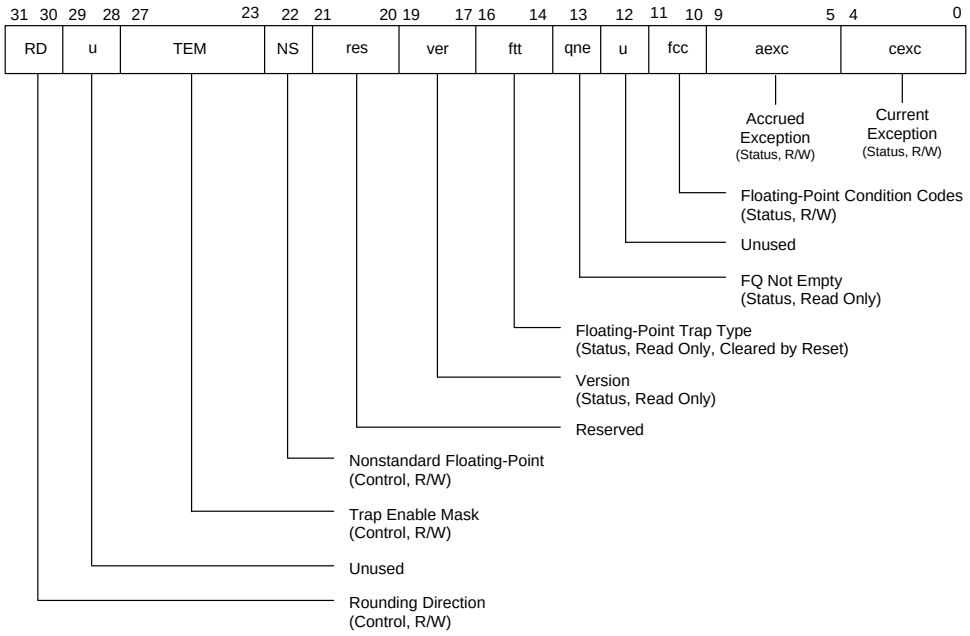


Figure D5–2. Floating-Point State Register

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:

Table D5–5. Rounding Direction For Floating–Point Results

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 D5.4.3, *IEEE 754 Exception*, for details).

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

Table D5–6. TEM Field Floating–Point Exceptions

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 D5.5.2, *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 D5.5.4, *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 MB86934 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 MB86934 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 MB86934), 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 D5.4.2, *Floating-Point Exception Trap Types*, for details):

Table D5–7. Floating–Point Exception Trap Types

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 MB86934)
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 MB86934 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, CMPE, EFCMP, and EFCMPE as follows:

Table D5–8. Floating–Point Compare Instruction

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 MB86934 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/EFCMP/EFCMPE 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:

Table D5–9. Floating–Point Exceptions During Trap Disable

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 D5.4.3, *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:

Table D5–10. Floating–Point Exceptions Generated By FPop Instruction

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 D5.4.3, *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.

D5.3.2 Enhanced Register Set (f Registers and FIFOs)

The MB86934 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).

The FPU also features six vector-type f registers called FIFOs or vector registers. The FIFOs are mapped to f[20], f[22], f[24], f[26], f[28], and f[30]. They can be read and written with the single/double-precision Enhanced-FPop instructions, which are newly defined to access the FIFOs in SPARClite processors.

Table D5-11 shows FPU f register and FIFO access. Note that when accessing f[20], f[22], f[24], f[26], f[28], and f[30], the FPop instructions access f registers, and the Enhanced FPop instructions access the FIFO.

Table D5–11. FPU Register Access

Operand Location	FPop Access	EFPop Access
%f0	f Register	f Register
%f1	f Register	f Register
%f2	f Register	f Register
:	:	:
:	:	:
%f17	f Register	f Register
%f18	f Register	f Register
%f19	f Register	f Register
%f20	f Register	FIFO (vector type f register)
%f21	f Register	f Register
%f22	f Register	FIFO (vector type f register)
%f23	f Register	f Register
%f24	f Register	FIFO (vector type f register)
%f25	f Register	f Register
%f26	f Register	FIFO (vector type f register)
%f27	f Register	f Register
%f28	f Register	FIFO (vector type f register)
%f29	f Register	f Register
%f30	f Register	FIFO (vector type f register)
%f31	f Register	f Register

f registers

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]...).

FIFOs

Each of the six FIFOs is 32 bits wide and 64 words deep. Each FIFO has a pointer, and a depth register. The FIFO pointer contains the byte address of the next word in the

FIFO that is to be accessed. The depth register contains the byte address of the last word in the FIFO that can be accessed.

Figure D5–3 shows the format of the FIFO pointer register and the FIFO depth register. Bits 7 through 2 of the registers are implemented, but Bits 31 through 8, and Bits 1 through 0 are not. Those are reserved. If the registers are read, the valus of those bits are unknown.

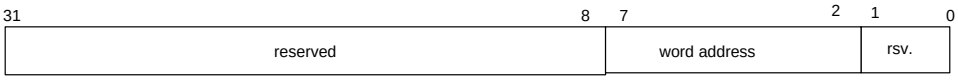


Figure D5–3. FIFO Pointer and Depth Register Format

When single-precision size data (single-word) is accessed with an Enhanced FPop instruction, the data is transferred to/from the word that is pointed to by the FIFO pointer. The FIFO pointer then increments by 4 to point to the next word in the FIFO (see Figure D5-4).

When double-precision size data (double word) is accessed with an Enhanced-FPop instruction, data bits 63:32 are transferred to/from the word that is pointed to by the FIFO pointer, then data bits 31:0 are transferred to/from the word that is pointed to by the FIFO pointer +4. The FIFO pointer then increments by 8 to point to the next double word in the FIFO.

When the source f registers (frs1 and frs2) of an Enhanced-FPop designate the same FIFO (e.g., efadds %f30,%f30,%f0), the FIFO is accessed only once, not twice (see Figure D5-5). As a result, the operands have the same value, and the FIFO's pointer is incremented only once (+4 for a single word access, or +8 for a double word access).

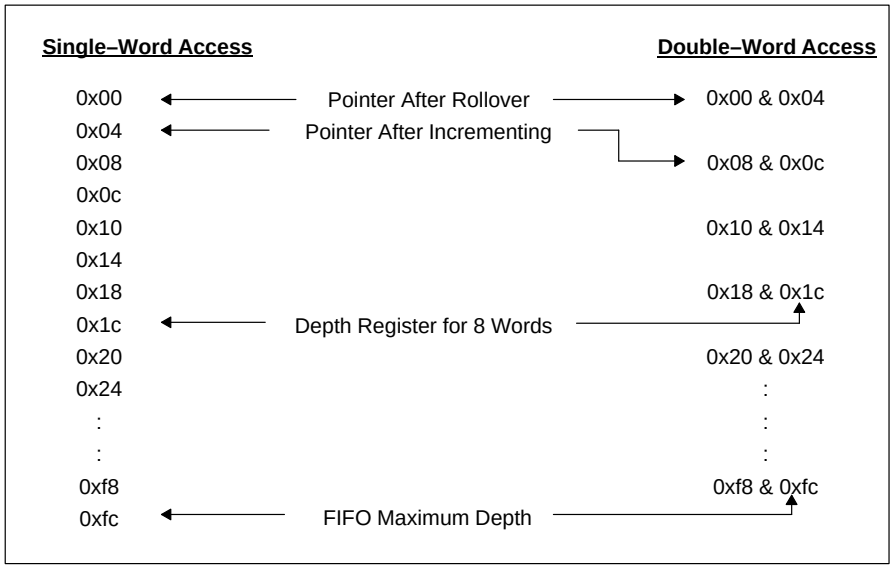


Figure D5–4. FIFO Pointer Operation

The logical depth of each FIFO (i.e., the logical length of each vector register) is programmable in the range 1 to 64 by writing the required depth into the FIFO Depth Register. The FIFO Depth Register must be set to the byte address of the last (bottom) word of the FIFO (i.e., depth_register = (depth - 1) ∗ 4), regardless of the size of access. For example, if the logical depth of a FIFO is 8 words, the value of the FIFO depth register must be (8-1) ∗ 4 = 28 = 0x1c; if the logical depth is 64 words, the value of the FIFO depth register must be (64-1) ∗ 4 = 252 = 0xfc.

Once a FIFO pointer reaches the bottom, the FIFO pointer rolls over to zero for the next access.

Table D5-12 shows FIFO pointer and depth register addressing. The pointers and the depth registers of all FIFOs are allocated in the address range 0x00000500 to 0x0000052c in ASI=0x01. They can be read and written with the STA and LDA instructions, respectively. (FIFO–DMA address cannot be read and written in this manner. Please see the FIFO–DMA Transfer Section.)

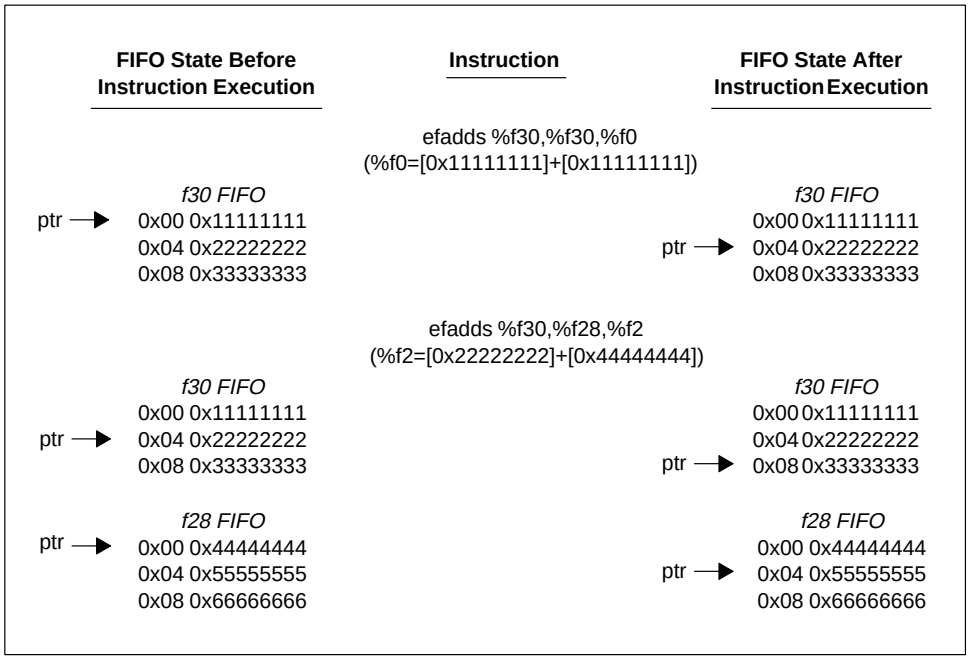


Figure D5–5. FIFO Pointer Operation - Same Register Access

Table D5–12. FIFO Pointer and Depth Register Addressing; also FIFO–DMA Addressing

	Pointer Register Address (ASI = 0x01)	Depth Register Address (ASI = 0x01)	FIFO–DMA Address (ASI = 0x01)
FIFO for f20	0x00000500	0x00000518	0x00000530
FIFO for f22	0x00000504	0x0000051C	0x00000534
FIFO for f24	0x00000508	0x00000520	0x00000538
FIFO for f26	0x0000050C	0x00000524	0x0000053c
FIFO for f28	0x00000510	0x00000528	0x00000540
FIFO for f30	0x00000514	0x0000052C	0x00000544

The following code fragment contains examples of FIFO pointer and depth register initialization:

```
set (5-1)*4,    %g1     ! FIFO depth is 5 words for single-word access.
set 0x500,      %g2     ! FIFO f20 pointer register address is 0x500.
set 0x518,      %g3     ! FIFO f20 depth   register address is 0x518.
sta %g0,[%g2]   0x1     ! initialize pointer register to 0.
sta %g1,[%g3]   0x1     ! initialize depth register to (5-1)*4.

set (6-1)*4,    %g1     ! FIFO depth is 6 words for double-word access.
set 0x504,      %g2     ! FIFO f22 pointer register address is 0x504.
set 0x51c,      %g3     ! FIFO f22 depth   register address is 0x51c.
sta %g0,[%g2]   0x1     ! initialize pointer register to 0.
sta %g1,[%g3]   0x1     ! initialize depth register to (6-1)*4.
```

Programming Notes:

(1) Both single words and double words can be accessed in the same FIFO. However, the programmer must maintain consistency in the single-word and double-word boundaries. The FIFO pointer must be on double-word boundaries (i.e., pointer[2:0] = 000) for double-word accesses to the FIFO, and must be on the single-word boundaries (i.e., pointer[1:0] = 00) for the single-word accesses. Otherwise, the contents of the FIFO may become unknown without warning, since accesses with inconsistent pointer boundaries do not generate traps.

Similarly, a FIFO's pointer may become unknown without warning if the FIFO's depth register is not on the proper boundary for single or double-word accesses (e.g., a double-word access to a FIFO with a length of one word (depth_register==0x00)).

(2) Both read accesses and write accesses to each FIFO are allowed. Since each FIFO has only one pointer and one depth register, the read accesses and the write accesses share the same pointer. For this reason, the FIFOs are not true "first in-first out" queues. However, if read accesses and the write accesses are not "mixed" in individual vector operations, the FIFOs can be considered "first in-first out" queues.

FIFO–DMA Transfer

The DMA controller can directly transfer data between the memory and the FIFOs.

Unlike other DMA transfers, in the FIFO–DMA transfer, its direction is controlled by the External Control Option (EC) bit in the DMA Channel Control Register. If EC=0 (i.e. source request mode), the FIFO becomes the source of the FIFO–DMA transfer. If EC=1 (i.e. destination request mode), the FIFO becomes the destination of the FIFO–DMA transfer.

For the FIFO–DMA transfer, the DMA Current Source Address Register is set to the address of the memory, whether the memory is source or destination. Also, the DMA

Current Destination Register is set to the FIFO–DMA address (shown in the Table D5–11, whether the FIFO is source or destination. The Source ASI and Destination ASI fields in the DMA DP/Source/Destination ASI Register, and some bits in the DMA Channel Control Register are also set in the same way (i.e. Source==Memory; Destination==FIFO).

Byte and half–word size transfers to/from the FIFO are not supported. Also, the DMA transfer to/from the FIFO can occur only in flyby mode. The FIFO can be used as either a 32-bit or a 64-bit wide data device. If SDRAM is the destination or source of the FIFO-DMA transfer, the Destination Data Width (DDW) bit and the Source Data Width (SDW) bit in the DMA Channel Control Register should be set to 1 (i.e. 64-bit wide data mode / SDRAM mode). If not, both bits should be set to 0 (i.e. 32-bit wide data mode / non–SDRAM mode).

For details about the DMA function, please refer to the Chapter D4 : DMA of this manual.

Programming Note:

> If the DMA controller and FPU both access the same FIFO simultaneously, the result is unpredictable. Programming must ensure that each FIFO is exclusively accessed from the DMA controller and FPU.

D5.3.3 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.

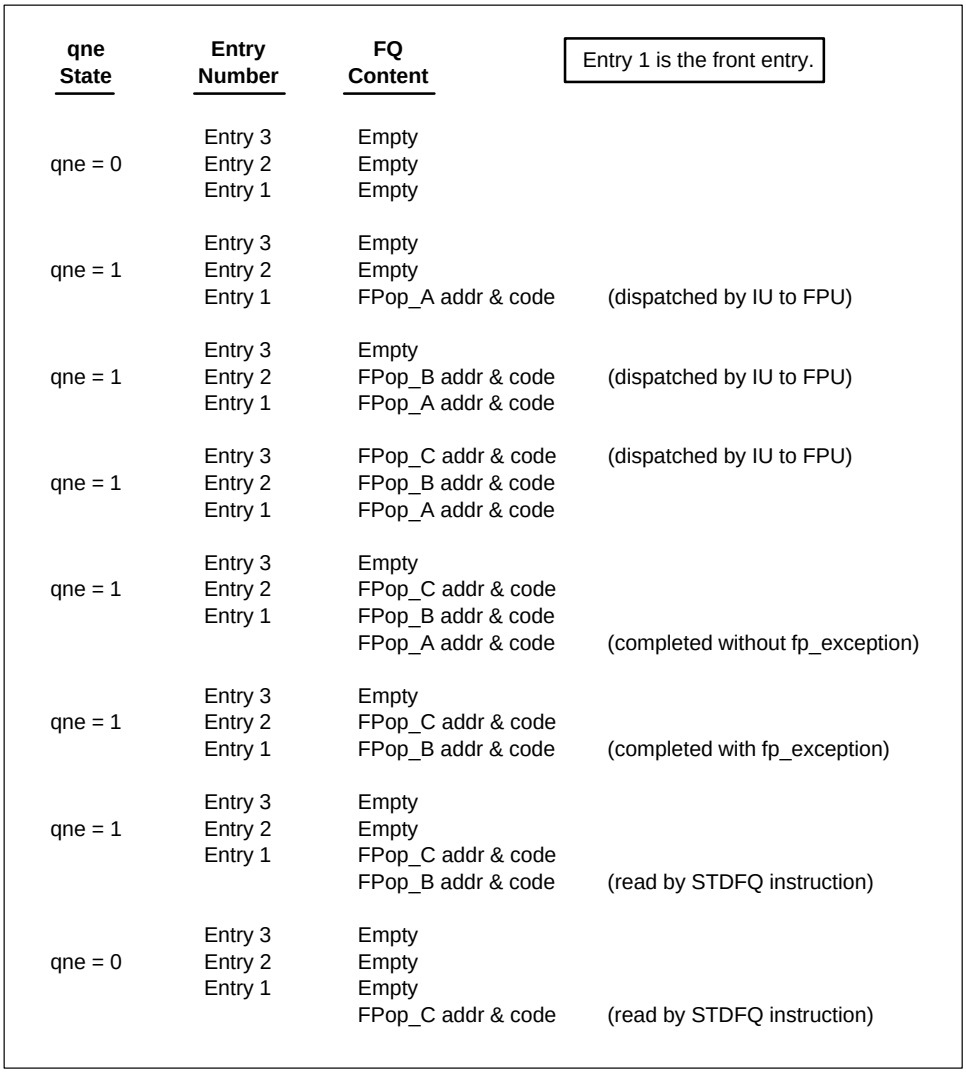
The FQ is a first-in-first-out queue. The FPU therefore cannot change the order of completion of the instructions in the FQ. The number of entries of the FQ is implementation-dependent, so FQs in other SPARC devices may hold a different number of entries.

Figure D5-6 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 D5.3.1, *Floating-Point State Register (FSR)*, for a description of the qne bit.)



qne State	Entry Number	FQ Content	Entry 1 is the front entry.
	Entry 3	Empty	
qne = 0	Entry 2	Empty	
	Entry 1	Empty	
	Entry 3	Empty	
qne = 1	Entry 2	Empty	
	Entry 1	FPop_A addr & code	(dispatched by IU to FPU)
	Entry 3	Empty	
qne = 1	Entry 2	FPop_B addr & code	(dispatched by IU to FPU)
	Entry 1	FPop_A addr & code	
	Entry 3	FPop_C addr & code	(dispatched by IU to FPU)
qne = 1	Entry 2	FPop_B addr & code	
	Entry 1	FPop_A addr & code	
	Entry 3	Empty	
qne = 1	Entry 2	FPop_C addr & code	
	Entry 1	FPop_B addr & code	
		FPop_A addr & code	(completed without fp_exception)
	Entry 3	Empty	
qne = 1	Entry 2	FPop_C addr & code	
	Entry 1	FPop_B addr & code	(completed with fp_exception)
	Entry 3	Empty	
qne = 1	Entry 2	Empty	
	Entry 1	FPop_C addr & code	
		FPop_B addr & code	(read by STDFQ instruction)
	Entry 3	Empty	
qne = 0	Entry 2	Empty	
	Entry 1	Empty	
		FPop_C addr & code	(read by STDFQ instruction)

Figure D5–6. 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.

If an FPop in the FQ is an Enhanced-FPop that accesses a FIFO as a source register, the floating-point trap handler must restore the FIFO's pointer before re-executing or emulating the FPop because the pointer was increased when that FPop was dispatched to the FPU by the IU (i.e., when the FPop moved into the FQ). The floating-point trap handler must therefore check all Enhanced-FPops in the FQ to see if any FIFOs were read. If a FIFO was read once, the handler must decrease the FIFO pointer value by 4 for single-word access, or by 8 for double-word access. The trap handler can then use the FMOV instruction once for single-word access or twice for double-word access to get the source register value from the FIFO before it emulates the Enhanced FPop in software.

Unlike the source registers, an FPop destination register is accessed only when the FPop has completed execution without a floating-point exception. Therefore, a FIFO's pointer does not require restoration by the trap handler if the FIFO is accessed as a destination register.

D5.3.4 EF and EC bit in PSR; EFIFO bit in ASR17

The Enable Floating-Point (EF) bit is Bit 12 of the Processor State Register (PSR). The Enable_Coprocessor (EC) bit is Bit 13 of the PSR. The Enable_FIFO (EFIFO) bit is Bit 3 of the Ancillary State Register 17 (ASR17).

Table D5-13 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 D5–13. 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

Table D5-14 shows the effects of the EC, EF, and EFIFO bit states on the EFPop1 and EFPop2 instructions. When EC = 0, EF = 1, and EFIFO = 1, these instructions can be executed. If the bits are in other states, these instructions cause the cp_disabled trap, and the EFPop1/EFPop2 instructions are not dispatched to the FPU.

Table D5–14. EC/EF/EFIFO Bit Effects on Instruction Execution

EC Bit State	EF State	EFIFO State	Effect on EFPop1/EFPop2
0	0	*	Causes cp_disabled trap
0	*	0	Causes cp_disabled trap
0	1	1	Executed by FPU
1	*	*	Causes cp_disabled trap

Although the EF, EC, and EFIFO bits control whether a floating-point instruction is trapped and whether an FPop instruction is dispatched to the FPU, they do not control the FPU and FIFO directly. The FPU continues to execute FPop and Enhanced FPop instructions in the FQ even when the FPU is disabled, and DMA can access the FIFO.

The EFIFO bit is cleared by the reset, so software begins execution with the FIFOs disabled (i.e., without the FIFOs). From the configuration point of view, this initial mode can be considered the IU+FPU mode. When the EFIFO bit is set to 1, the new mode can be considered the IU+FPU+FIFO mode.

Programming Notes:

(1) 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/FIFO/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.

If the process uses FIFOs, the OS must also save information such as pointer, depth register, data, and/or DMA information for all 6 FIFOs for restoration later. If the process uses only the FPU and not the FIFOs, the OS does not have to save the FIFO information.

(2) In the MB86934, the Enhanced-FPop (EFPop) instruction set is mapped over the object code field of the CPop instruction set (i.e., op=2, op3=0x36 or 0x37).

Therefore, a program written for the MB86934 should not have CPop instructions.

D5.4 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.

D5.4.1 Traps Associated with Floating-Point Instructions

Floating-point instructions consist of FPop (FPop1, FPop2, EFPop1, EFPop2), LDfp (LDF, LDDF, LDFSR), STfp (STF, STDF, STFSR, STDFQ), and FBfcc instructions. There are 5 traps associated with the floating-point instructions: fp_disabled trap, cp_disabled trap, fp_exception trap, mem_address_not_aligned trap, and data_access_exception trap.

fp_disabled trap and cp_disabled trap

With certain combinations of the EF, EC, and EFIFO bits, the floating-point instructions may cause an fp_disabled trap or a cp_disabled trap. (See Section D5.3.4, *EF and EC bit in PSR; EFIFO bit in ASR17*, for details.)

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 the precise trap. The rest are the 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 MB86934, 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 MB86934, and the LSB of the frd field is ignored (i.e., forced to 0 internally).

D5.4.2 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 MB86934 FPU. The rest of the Floating-Point Trap Types are implemented in the MB86934, including a new trap type defined in the SPARC Version 8 specification, ftt 6 (invalid_fp_register).

The MB86934 FPU uses the ftt 2 (unfinished_FPop) trap type to handle subnormal numbers. (See Section D5.5.4, *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 D5.5.5, *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 D5-15. 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 D5–15. Floating-Point Trap Types

ftt	Priority	Trap Type	Implementation in MB86934 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 MB86934 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 MB86934, 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 MB86934 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 MB86934 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 MB86934 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.

D5.4.3 IEEE 754 Exception

The IEEE 754 exception has five exception types: invalid, overflow, underflow, division-by-zero, and inexact. Figure D5-7 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 D5.5.4, *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.

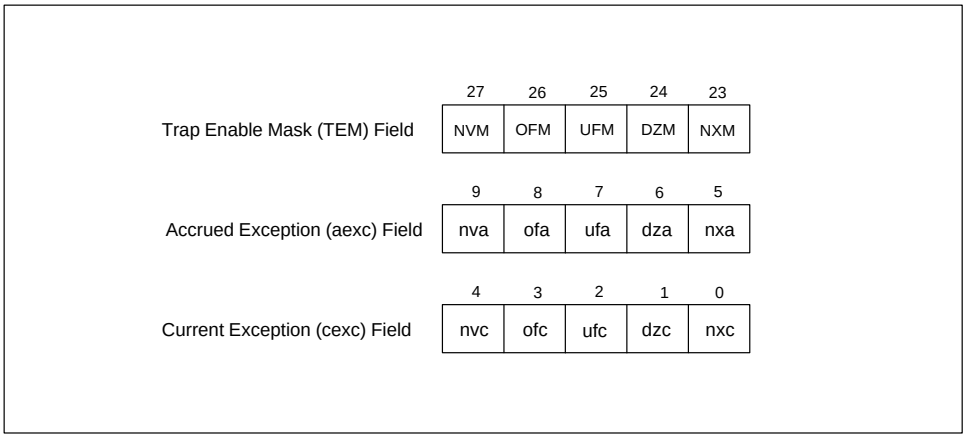


Figure D5–7. FSR TEM, aexc, and cexc Fields

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 D5.5.2, *Overflow, Underflow, and Inexact,* for details.)

D5.4.4 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_exeption 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. If the frd is a FIFO, its pointer is also unchanged.

(6) If the source f register 1 (frs1) or 2 (frs2) is a FIFO, its pointer was already increased before the trap occurred, so the pointer is not restored by the FPU after the trap. However, a floating-point trap handler can restore the value of the pointer by analyzing the FPop(s) in the FQ.

Programming Note:

> If the frs1 (or frs2) of an FPop is a FIFO, its pointer is increased when the IU dispatches the FPop to the FPU (i.e., when the FPop moves into the FQ). If the frd of an FPop is a FIFO, its pointer is increased when the FPU completes the FPop without a trap (i.e., when the FPop moves out of the FQ).
>
> Unlike the frd (not yet accessed), the frs1/frs2 (already accessed) requires additional work by a floating-point trap handler. First, the trap handler must read each FPop (particularly Enhanced-FPops) remaining in the FQ after the trap so that it can determine which FIFO was accessed, when it was accessed, and how it was accessed (double word or single word access). Next, the trap handler must determine the current value of each FIFO's pointer by reading it directly. With this information the trap handler can identify each data which the FPop(s) in the FQ accessed as the frs1 (or frs2).

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 D5-8 summarizes Floating-Point trap handling.

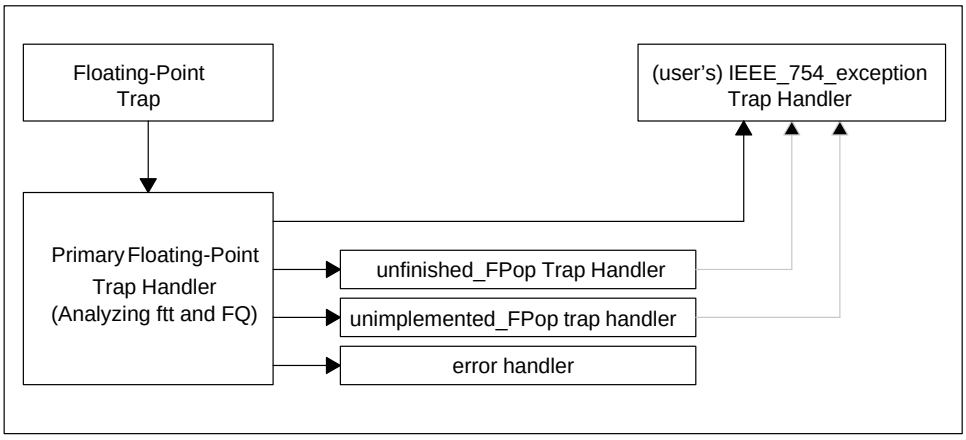


Figure D5–8. Floating-Point Trap Handling

D5.4.5 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 D5-16 summarizes the FPU states.

Table D5–16. 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 MB86934 FPU. Other SPARC FPUs may have different definitions.

D5.4.6 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 MB86934 FPU. Other SPARC FPUs may have different definitions.

D5.5 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 MB86934 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 MB86934 FPU incorporates the recommendations, except for the Nonstandard (NS) mode recommendation.

D5.5.1 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.

(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:

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

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 table shows FPop results from NaN operand(s)

Table D5–17. FPop Results From NaN Operand(s)

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

	-- *frs2* --		
-- *frs1* --	**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}

	-- *frs2* --		
-- *frs1* --	**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.

Table D5–18. Invalid Exception ConditionsTable D5–2.

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

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

[exc] {FCMPE}

-- frs1 --	*-- frs2 --* 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.

Table D5–19. Instruction Operations

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

D5.5.2 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

MB86934 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:

Table D5–20. Rounded Result

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:

Table D5–21. Rounding Mode And Sign Of Result

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:

Table D5–22. Rounded Result

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:

Table D5–23. Default Value Of Rounding Mode

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.

D5.5.3 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 SPARC'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 the polarity, the next 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.

Table D5–24. Destination f Register Values

[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.

D5.5.4 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.

D5.5.5 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 MB86934 implementation. Table D5–26 shows the MB86934 trap priorities.

Table D5–25. 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.

D5.5.6 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.

Table D5–3. 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]

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

[exc]

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

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

FDIV Without NaN/DNRM Operands

[frd]

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

[exc]

	-- frs2 --		
-- frs1 --	±**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.

D5.6 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 MB86934 FPU, but the highest performance is realized with code written specifically for the MB86934 FPU.

This section describes the MB86934 pipeline structure and interlock conditions. It is intended for the software engineer who's goal is to write software (such as an MB86934-specific compiler or key routines or libraries in assembly language) that has the highest-performance possible.

The MB86934 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.

D5.6.1 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.

D5.6.2 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 D5-28 shows floating-point instruction throughput and latency.

Table D5–26. Floating-Point Instruction Throughput and Latency

Instruction	Throughput	Latency	Latency (if FIFO)
FDIVs/FSQRTs	13	14	16
FMULs/FsMULd	1	3	5
FADDs/FSUBs	1	3	5
All Other FPops_s	1	3	5
FDIVd/FSQRTd	28	29	31
FMULd	4	6	8
FADDd/FSUBd	1	3	5
All Other FPop_d	1	3	5

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 technic works for the f register.

However, unlike the f register, when a FIFO is written with a value that is then read, the written value and the read value may not be the same. Therefore, the bypass technic cannot apply to the FIFO.

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.

Unlike the f register, the "input port to output port" technic cannot be applied to the FIFO because the FIFO has only one port (and one pointer) that is shared between the input and the output.

The FPU requires 1 cycle to write the result to a FIFO in the D_stage. The IU requires another 1 cycle to read the operand(s) from the FIFO in the Execution stage. As a result, 2 cycles are added to the latency of each FPop if a dependency occurs in a FIFO.

D5.6.3 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.

D5.6.4 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 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 D5-9 shows an example of FPU_full interlock.

D5.6.5 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:		IU_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 D5–9. 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) When an FPop follows another FPop and there is a RAW dependency in a FIFO, the following FPop is stalled while the preceding FPop is in the FPU_A, FPU_B, FPU_C, or FPU_D stage because the bypass technique and "input port to output port" technique cannot be applied to the FIFO.

For example, if "EFSUBs %f20,%f3,%f4" follows "EFADDs %f0,%f1,%f20" without any instruction between them, the EFSUBs instruction is stalled for 4 cycles by the RAW Data Hazard interlock in the IU_E stage because there is a RAW dependency in the %f20 FIFO. The EFSUBs instruction "reads" the %f20 FIFO "after" the EFADDs instruction "writes" the %f20 FIFO.

If "EFSUBs %f20,%f3,%f4" follows "FADDs %f0,%f1,%f20", there is no RAW Data Hazard interlock because the EFSUBs instruction reads the %f20 FIFO, and the FADDs instruction writes the %f20 register.

(4) 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 date moves from the IU_W stage to the IU_E stage via the designated f register. (Note: If the IU_M to IU_E bypass were taken, this interlock would not be required, but the critical timing path would be slower. This is one example of the trade-off between a penalty and maximum frequency.)

For example, if "FSUBs %f2,%f3,%f4" follows "LD [4],%f2" without any instruction between them, the LDF 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.

(5) 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 date 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 (it depends on the hold conditions.). The STF instruction "reads" the %f2 register "after" the LDF instruction "writes" the %f2 register.

(6) 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.

(7) 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 it 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.

D5.6.6 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.

D5.6.7 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 FBfcc makes decision in the Decode stage of the IU, so waiting 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.

D5.6.8 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 at 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

D6

Floating-Point Instructions

This chapter describes all floating-point instructions that the MB86934 supports in hardware. The Enhanced Floating-point Operate 1/2 (EFPop1/2) instructions are new instructions not defined in the original SPARC specification that are used to access the FIFOs in the MB86934.

D6.1 Floating-point Operate (FPop) Instructions

opcode	op3	operation
FPop1	110100	Floating-point operate
FPop2	110101	Floating-point operate
EFPop1	110110	Enhanced Floating-point operate
EFPop2	110111	Enhanced 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
EFPop1	10	rd	110110	rs1	opf	rs2
EFPop2	10	rd	110111	rs1	opf	rs2

Description:

The Floating-point operate (FPop) instructions are encoded using four "type 3" formats: FPop1, FPop2, EFPop1, and EFPop2. 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 EFPop1 instructions do not affect the floating-point condition codes, but the FPop2 and EFPop2 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, and an EFPop1 or EFPop2 instruction causes a cp_disabled trap.

Floating-point exceptions may cause either precise or deferred traps.

EFPop1 and EFPop2 instructions operate in the same way as the respective FPop1 and FPop2 instructions with the exception that EFPop1 and EFPop2 instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Programming Note:

The following restriction does not apply to the MB86934, but may apply to other SPARC processors.

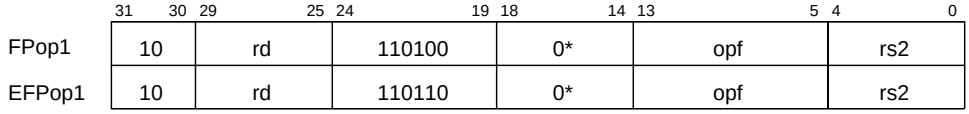
If an FPop2 instruction such as FCMP or FCMPE sets the floating-point condition codes, then at least one non-FPop2 (non-floating-point operate 2) instruction must be executed between the FPop2 instruction and a following FBfcc instruction. Otherwise, the result of the FBfcc instruction is undefined.

The MB86934 will generate a one cycle interlock delay automatically if floating branch immediately follows floating compare.

D6.1.1 Convert Integer to Floating-Point Instructions

opcode	opf	operation
FiTOs/EFiTOs	011000100	Convert Integer to Single
FiTOd/EF iTOd	011001000	Convert Integer to Double
FiTOq	011001100	Convert Integer to Quad *

Format:



	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	0*	opf	rs2
EFPop1	10	rd	110110	0*	opf	rs2

* Not used, must be 0.

Syntax:

fitos	*fregrs2 , fregrd*
fitod	*fregrs2 , fregrd*
efitos	*fregrs2 , fregrd*
efitod	*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.

The EFiTOs and EFiTOd instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Traps:

fp_disabled/cp_disabled
fp_exception (NX (FiTOs/EFiTOs only), Invalid_fp_register (FiTOd/EFiTOd, FiTOq))

*Generates unimplemented_ FPop Trap

D6.1.2 Convert Floating-Point to Integer Instructions

opcode	opf	operation
FsTOi/EFsTOi	011010001	Convert Single to Integer
FdTOi/EFdTOi	011010010	Convert Double to Integer
FqTOi	011010011	Convert Quad to Integer *

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	0*	opf	rs2
EFPop1	10	rd	110110	0*	opf	rs2

* Not used, must be 0.

Syntax:

fstoi	*fregrs2 , fregrd*
fdtoi	*fregrs2 , fregrd*
efstoi	*fregrs2 , fregrd*
efdtoi	*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).

The EFsTOi and EFdTOi instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Traps:

fp_disabled/cp_disabled
fp_exception (NV, NX, invalid_fp_register (FdTOi/EFdTOi, FqTOi))

*Generates unimplemented_ FPop Trap

D6.1.3 Convert Between Floating-Point Formats Instructions

opcode	opf	operation
FsTOd/EFsTOd	011001001	Convert Single to Double
FsTOq	011 001101	Convert Single to Quad *
FdTOs/EFdTOs	011000110	Convert Double to Single
FdTOq	011001110	Convert Double to Quad *
FqTOs	011000111	Convert Quad to Single *
FqTOd	011001011	Convert Quad to Double *

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	0*	opf	rs2
EFPop1	10	rd	110110	0*	opf	rs2

* Not used, must be 0.

Syntax:

fstod *$freg_{rs2}$, $freg_{rd}$*
efstod *$freg_{rs2}$, $freg_{rd}$*
fstoq *$freg_{rs2}$, $freg_{rd}$*
fdtos *$freg_{rs2}$, $freg_{rd}$*
efdtos *$freg_{rs2}$, $freg_{rd}$*
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/EFdTOs (the "narrowing" conversion instructions) can result in OF, UF, and NX exceptions. FdTOq, FsTOq, and FsTOd/EFsTOd (the "widening" conversion instructions) cannot. Any of these eight instructions can trigger an NV exception if the source operand is a signaling NaN.

The EFsTOd and EFdTOs instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Traps:

fp_disabled/cp_disabled
fp_exception (OF, UF, NV, NX, invalid_fp_register)

*Generates unimplemented_ FPop Trap

D6.1.4 Floating-Point Move Instructions

opcode	opf	operation
FMOVs/EFMOVs	000000001	Move
FNEGs/EFNEGs	000000101	Negate
FABSs/EFABSs	000001001	Absolute Value

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	0*	opf	rs2
EFPop1	10	rd	110110	0*	opf	rs2

* Not used, must be 0.

Syntax:

fmovs	$freg_{rs2}$, $freg_{rd}$
efmovs	$freg_{rs2}$, $freg_{rd}$
fnegs	$freg_{rs2}$, $freg_{rd}$
efnegs	$freg_{rs2}$, $freg_{rd}$
fabss	$freg_{rs2}$, $freg_{rd}$
efabss	$freg_{rs2}$, $freg_{rd}$

Description:

FMOVs/EFMOVs copies the contents of f[rs2] to f[rd]. FNEGs/EFNEGs copies the contents of f[rs2] to f[rd] with the sign bit complemented. FABSs/EFABSs copies the contents of f[rs2] to f[rd] with the sign bit cleared. These instructions do not round.

The EFMOVs, EFNEGs, and EFABSs instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Programming Notes:

(1) One FMOVs/EFMOVs 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/EFNEGs (FABSs/EFABSs) and a following FMOVs/EFMOVs instruction perform a double-precision negation (absolute value); an FNEGs/EFNEGs (FABSs/EFABSs) and three following FMOVs/EFMOVs instructions perform a quad-precision negation (absolute value).

Traps:

fp_disabled/cp_disabled

D6.1.5 Floating-Point Square Root Instructions

opcode	opf	operation
FSQRTs/EFSQRTs	000101001	Square Root Single
FSQRTd/EFSQRT d	000101010	Square Root Double
FSQRTq	000101011	Square Root Quad *

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	0*	opf	rs2
EFPop1	10	rd	110110	0*	opf	rs2

* Not used, must be 0.

Syntax:

fsqrts	$freg_{rs2}$, $freg_{rd}$
efsqrts	$freg_{rs2}$, $freg_{rd}$
fsqrtd	$freg_{rs2}$, $freg_{rd}$
efsqrtd	$freg_{rs2}$, $freg_{rd}$
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.

The EFSQRTs and EFSQRTd instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Traps:

fp_disabled/cp_disabled
fp_exception (NV, NX, invalid_fp_register (FSQRTd/EFSQRTd, FSQRTq))

*Generates unimplemented_ FPop Trap

D6.1.6 Floating-Point Add and Subtract Instructions

opcode	opf	operation
FADDs/EFADDs	001000001	Add Single
FADDd/EFADDd	001000010	Add Double
FADDq	001000011	Add Quad *
FSUBs/EFSUBs	001000101	Subtract Single
FSUB d/EFSUBd	001000110	Subtract Double
FSUBq	001000111	Subtract Quad *

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	rs1	opf	rs2
EFPop1	10	rd	110110	rs1	opf	rs2

Syntax:

fadds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efadds	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
faddd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efaddd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
faddq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsubs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efsubs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsubd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efsubd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
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 difference 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.

The EFADDs, EFADDd, EFSUBs, and EFSUBd instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Traps:

fp_disabled/cp_disabled
fp_exception (OF, UF, NX, NV (∞–∞), invalid_fp_register (all except FADDs/EFADDs and FSUBs/EFSUBs))

*Generates unimplemented_ FPop Trap

D6.1.7 Floating-Point Multiply and Divide Instructions

opcode	opf	operation
FMULs/EFMULs	001001001	Multiply Single
FMULd/EFMULd	00100 1010	Multiply Double
FMULq	001001011	Multiply Quad *
FsMULd/EFsMULd	001101001	Multiply Single to Double
FdMULq	001101110	Multiply Double to Quad *
FDIVs/EFDIV s	001001101	Divide Single
FDIVd/EFDIVd	001001110	Divide Double
FDIVq	001001111	Divide Quad *

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop1	10	rd	110100	rs1	opf	rs2
EFPop1	10	rd	110110	rs1	opf	rs2

Syntax:

fmuls	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efmuls	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fsmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efsmuld	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdmulq	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdivs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efdivs	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
fdivd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
efdivd	$freg_{rs1}$, $freg_{rs2}$, $freg_{rd}$
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/EFsMULd 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.

The EFMULs, EFMULd, EFsMULd, EFDIVs, and EFDIVd instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Traps:

fp_disabled/cp_disabled
fp_exception (OF, UF, DZ (FDIV/EFDIV only), NV, NX, invalid_fp_register
(all except FMULs/EFMULs and FDIVs/EFDIVs))

*Generates unimplemented_ FPop Trap

D6.1.8 Floating-Point Compare Instructions

opcode	opf	operation
FCMPs/EFCMPs	001010001	Compare Single
FCMPd/EFCMPd	001010 010	Compare Double
FCMPq	001010011	Compare Quad *
FCMPEs/EFCMPEs	001010101	Compare Single and Exception if Unordered
FCMPEd/EFCMPEd	001010110	Compare Double and Exception if Unordered
FCMPEq	001010111	Compare Quad and Exception if Unordered *

Format:

	31 30	29 25	24 19	18 14	13 5	4 0
FPop2	10	0*	110101	rs1	opf	rs2
EFPop2	10	0*	110111	rs1	opf	rs2

* Not used, must be 0.

Syntax:

fcmps	$freg_{rs1}$, $freg_{rs2}$
efcmps	$freg_{rs1}$, $freg_{rs2}$
fcmpd	$freg_{rs1}$, $freg_{rs2}$
efcmpd	$freg_{rs1}$, $freg_{rs2}$
fcmpq	$freg_{rs1}$, $freg_{rs2}$
fcmpes	$freg_{rs1}$, $freg_{rs2}$
efcmpes	$freg_{rs1}$, $freg_{rs2}$
fcmped	$freg_{rs1}$, $freg_{rs2}$
efcmped	$freg_{rs1}$, $freg_{rs2}$
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/EFCMPEs, FCMPEd/EFCMPEd, and FCMPEq) cause an invalid (NV) exception if either operand is a signaling NaN or a quiet NaN. FCMP/EFCMP causes an invalid (NV) exception if either operand is a signaling NaN.

The EFCMPs, EFCMPd, EFCMPEs, and EFCMPEd instructions access FIFO(s) when their f register address(es) are f20, f22, f24, f26, f28, or f30.

Programming Note:

The following restriction does not apply to the MB86934, 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/cp_disabled
fp_exception (NV, invalid_fp_register (all except FCMPs/EFCMPs and FCMPEs/EFCMPEs)

*Generates unimplemented_ FPop Trap

D6.2 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:

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:

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 MB86934 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 MB86934, 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

D6.3 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 MB86934 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 MB86934 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.)

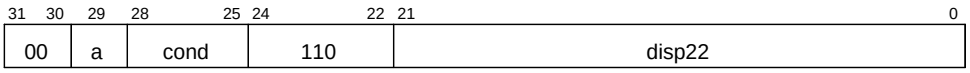
Traps:

fp_disabled
fp_exception (sequence_error (STDFQ))
privileged_instruction (STDFQ only)
mem_address_not_aligned
data_access_exception

D6.4 Branch on Floating-Point Codes (FBfcc) Instructions

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 MB86934, 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

D7

MB86934 Bus Interface Unit

D7.1 Overview of Bus Interface Unit

The BIU on the MB86934 includes all the features of the MB86930, and in addition offers the following:

- Double system clock frequency option,
- Four-word burst mode for instruction fetches and data loads,
- Byte-based parity generation/checking for the external data bus,
- A modified Wait State Specifier Register that supports burst mode and parity generation/checking on specified address ranges,
- A ROM/PROM interface that allows the MB86934 to boot from either 8-bit wide or 16-bit wide ROM/PROM,
- A processor bus request feature that enables the MB86934 to request access to external address and data buses,
- A peripheral-to-SDRAM interface,
- Control signals for non-volatile memory,
- Handshaking signals for external bus masters.

D7.2 Burst Mode

D7.2.1 Overview

The Bus Interface Unit (BIU) supports the fetching of instructions and data from external memory to the appropriate cache in 'bursts' of four words at a time. A burst mode transfer is initiated either by a cache miss or by a DMA request. For a cache miss, burst mode is supported only for instruction fetches and data loads, not for stores. The IU is held until all four words are fetched. For DMA burst access, both data burst reads and data burst writes are supported. (Note, however, that the DMA does not support movement of data to/from cache.)

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

D7.2.2 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.

D7.2.3 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 sequence is optimized for a 2-way interleaved memory.) The table below shows the four possible sequences of words, in terms of their address LSBs, depending on the LSBs of the word causing the miss. Note that the first word accessed in a burst is always the one requested by the IU and that during a burst access, bits ADR[3:2] do not change.

Table D7–1. 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

D7.2.4 Bus Mode Control Bits

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

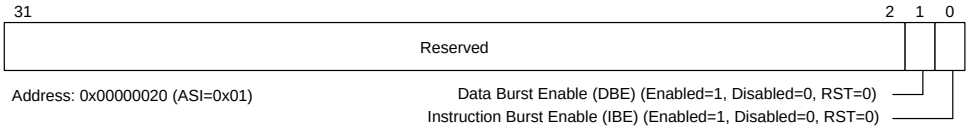


Figure D7–1. Bus Control Register

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.

D7.2.5 PROM Address Space

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

D7.2.6 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.

D7.2.7 Cache Off

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

D7.2.8 Bus Request

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

D7.2.9 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.

D7.2.10 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.

D7.2.11 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.

D7.2.12 Interface Timing

Figure D7-2 shows the timing of a burst mode transaction for an instruction fetch, data load, or DMA read. To start the transaction, the MB86934 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 D7-3.

Figure D7-4 shows the timing for the write portion of a DMA burst operation. The timing is identical to that in Figure D7-2, except that the RD/–WR line is low, indicating a write operation is in progress.

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

D7.3 Parity

The MB86934 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.

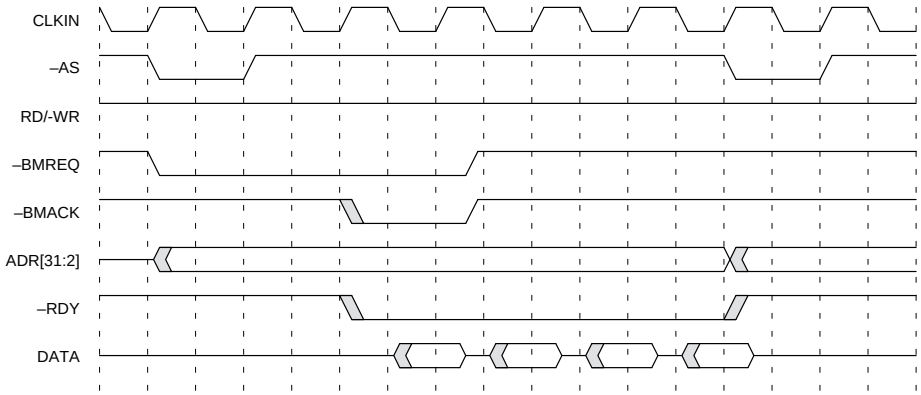


Figure D7–2. Burst Mode (0 wait state)

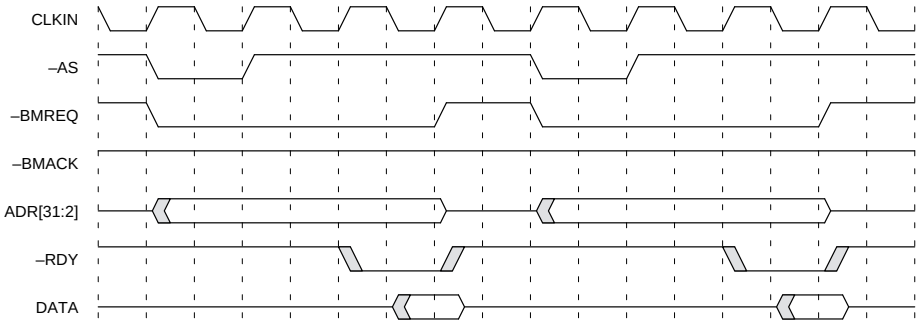


Figure D7–3. Terminated Burst Mode Due to –BMACK=1

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.

Note: PARITY<3> corresponds to D<31:24>
PARITY<2> corresponds to D<23:16>
PARITY<1> corresponds to D<15:8>
PARITY<0> corresponds to D<7:0>

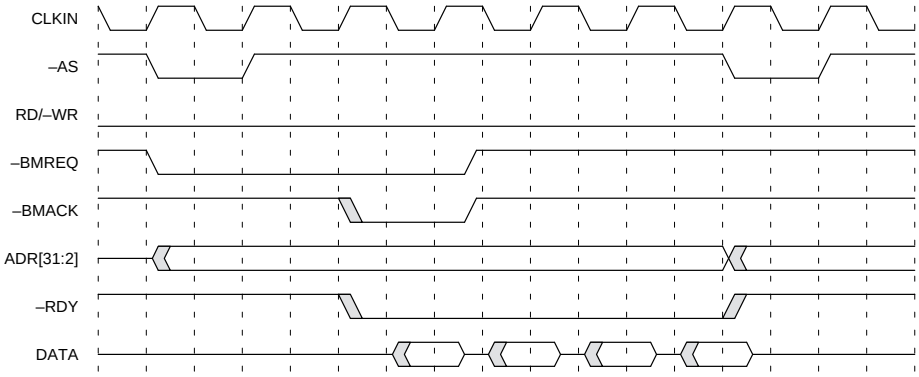


Figure D7–4. DMA Burst Mode, Write Portion

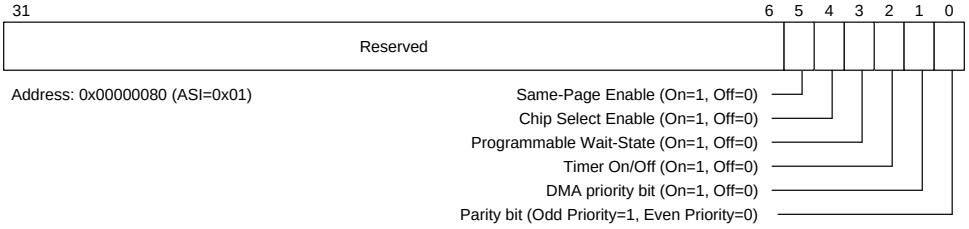


Figure D7–5. System Support Control Register

D7.4 Non Volatile Memory Support Signals

The MB86934 has two new signals, –NVWE (non-volatile RAM write enable) and –OE (output enable), that control non-volatile 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 when –READY is asserted, as shown in Figure D7-6. 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 D7-7.

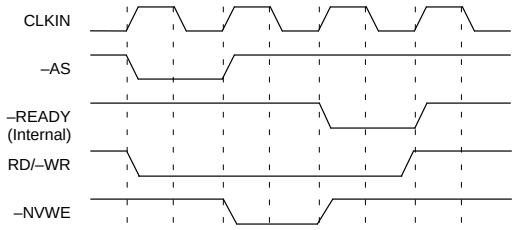


Figure D7–6. Non-Volatile Memory Write Timing

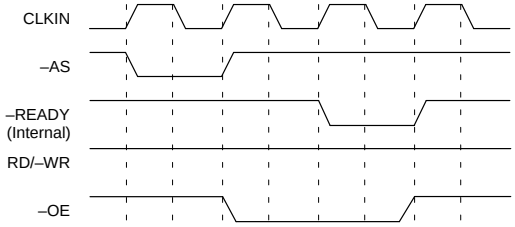


Figure D7–7. Non-Volatile Memory Read Timing

–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.

D7.5 SDRAM Address Space

Sparcpix 934 reserves chip select 5 exclusively for SDRAM access. CS5 is programable by using Address Range Specifier Register [5] and Address Mask Register [5].

D7.6 External Bus Master Support

The processor responds to a bus request by an external bus master by asserting the –BGRNT (Bus Grant) signal. The BIU asserts –CS (Chip Select) for the external bus master one cycle after asserting –AS, and the wait state control logic asserts –READYOUT to terminate the operation, as shown in Figure D7-8. The external bus master can drive the address and data bus only during the –AS cycle.

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

Note: –READYOUT is valid after the –BGRNT one cycle.

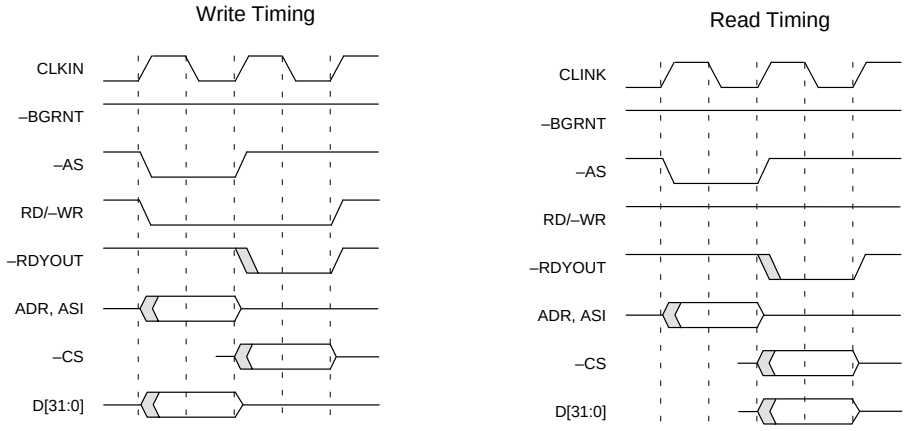


Figure D7–8. External Bus Master Signal Timing

D7.7 Same Page Support

The MB86934 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.

D7.8 Wait State Specifier Register

D7.8.1 Purpose

The Wait-State Specifier Register (WSSR) format on the MB86934 has been changed from that on the MB86930 to accommodate the burst mode bus transaction using internal –READY and Parity generation/checking.

D7.8.2 Format

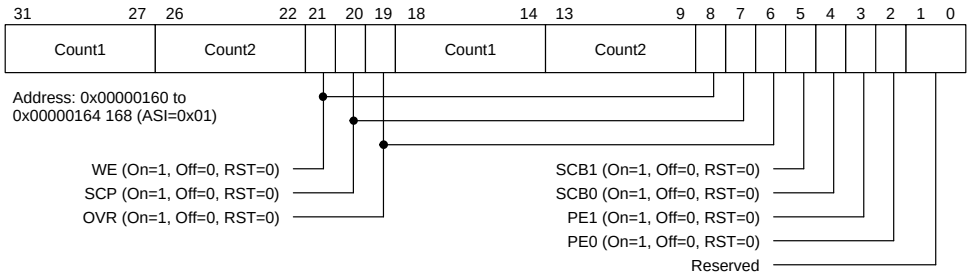


Figure D7–9. Wait State Specifier Register

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

D7.8.3 Wait State in CS [3:0]

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.

D7.8.4 Wait State in CS [4]

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

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

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

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

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

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 D7–2. 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

D7.9 Wait State Generation

The MB86934 Wait-State Specifier Register (WSSR) format is the same as the MB86932 Wait-State Specifier Register format. MB86934 wait state generation, however, differs as follows:

(1) For –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 –CS4, wait state generation is the same as in the MB86932.

(3) For –CS5, there is no wait state control, because this is the SDRAM range.

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

D7.10 ROM Interface

D7.10.1 Purpose

The data bus of the MB86934 can be configured upon reset to 8- and 16-bit bus modes as well as the standard 32-bit mode. This flexibility accommodates those cases in which boot code resides in PROMs organized as blocks of bytes or halfwords.

D7.10.2 Features

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

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

D7.10.3 Bus Configuration on Reset

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

(reserved): –BMODE16=0, –BMODE8=0

8-bit mode: –BMODE16=1, –BMODE8=0

16-bit mode: –BMODE16=0, –BMODE8=1

32-bit mode: –BMODE16=1, –BMODE8=1

D7.10.4 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[31:2] and selected –BE bits can be concatenated to form a complete address for a non-32 bit bus mode.

Table D7–3. System Interface –BE Bits

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

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

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

D7.10.5 PROM Address Space

The PROM address space is defined by the –CS[0] address-range specifier. On reset, the –CS[0] address range defaults to 32K bytes (starting address=0x0), and the ASI is initialized to 0x9. The PROM address range can be changed later using the mask bit register associated with –CS[0]. An example of the supervisor address space (ASI=0x9) memory map is shown below:

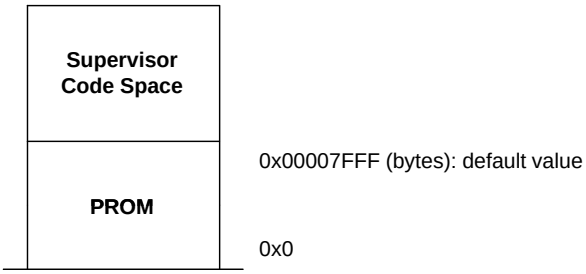


Figure D7–10. Supervisor Address Space (ASI=0x9) Memory Map

Any memory access from the PROM address space, in a non-32 bit mode, will make the –BE bit encodings reflect the LSBs of a byte/halfword address. Furthermore, the

fetched bytes/halfwords will be assembled into a 32-bit word. On the other hand, any access from the non-PROM address range will result in a normal, 32-bit memory access.

D7.10.6 Load/Stores

One of the functions of the boot code is to set the processor and system configuration. This might involve loading system parameters from PROM, loading data from memory mapped I/O, and storing data to non-PROM address space. All loads from the PROM address space behave the same way as instruction fetches, in that, for a non-32 bit bus mode –BE bit encoding and word assembly are done. Loads from a non-PROM address space behave in the normal (32-bit) manner. In order to meet the –BE AC timing, the –BE bits on the MB86934 need to be all 0's for all types of loads—word, halfword, and byte—from the non-PROM address space. This requires a functional change from the current specification of the MB86930's –BE bits, which reflect the byte information for loads. This change does not cause a problem, since the processor fetches a full 32-bit word on a load, and the IU selects the byte appropriately. As on the MB86930, –BE bits should be ignored for 32-bit loads.

Furthermore, store word operations to the PROM address space will not result in a dis-assembly process.

A summary of the –BE[0:3] bit behavior for loads from the PROM address space is shown below. For all load instructions (byte, halfword, word), a full 32-bit fetch occurs. For example, in the 8-bit bus mode, four bytes will be fetched for all loads, and the –BE bits will sequence with the proper 2 LSBs of the byte address.

Table D7–4. Load –BE[0:3] Bit Behavior

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

D7.10.7 8/16 Bit Bus Mode Write

The MB86934 also supports 8/16-bit Bus Mode write operations as follows:

(1) In 8-bit Bus Mode, only store byte is permitted. {ADR<31:2>, –BE2, –BE3} is the store address.

(2) In 16-bit Bus Mode, only store byte and halfword are permitted. {ADR<31:2>, –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]).

D7.10.8 Burst Mode

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

D7.10.9 Memory Exception

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

D7.10.10 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.

D7.10.11 Timing

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

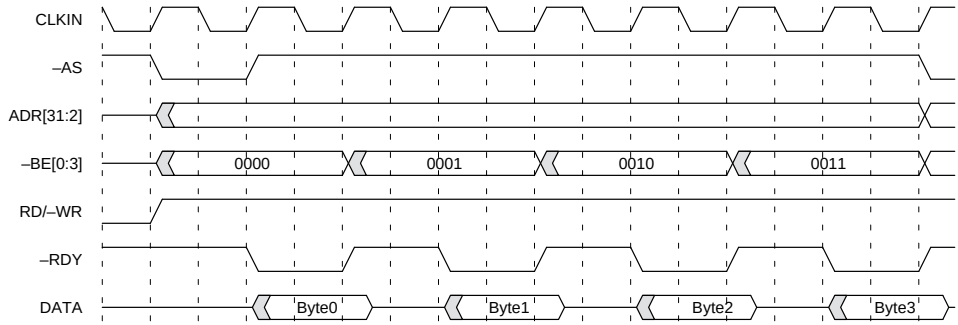


Figure D7–11. 8-bit Bus Mode (1 Wait State)

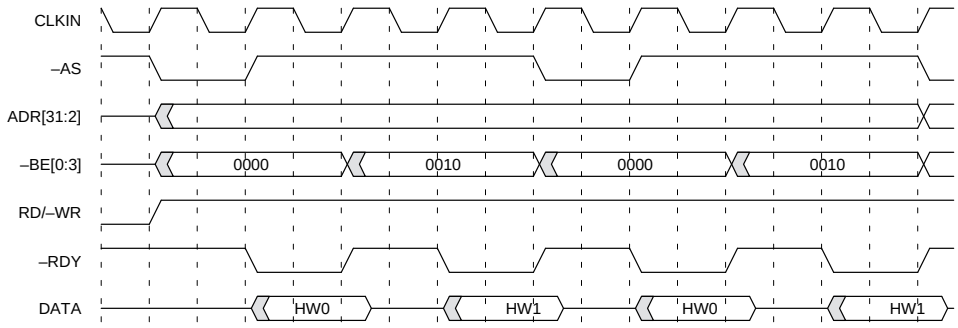


Figure D7–12. 16-bit Bus Mode (1 Wait State)

D7.11 Processor Bus Request

D7.11.1 Purpose

When the bus is released in response to an external device's request for the bus (by asserting –BREQ), the MB86934 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.

D7.11.2 Features

–PBREQ pin: An external pin is used to output the processor bus request signal, –PBREQ. The –PBREQ will be asserted whenever the MB86934 requires the bus while the bus is granted to an external device. The external device using the bus can monitor the –PBREQ signal, and remove the –BREQ signal at an appropriate time. An example of the –PBREQ timing is shown in Figure D7-13.

In the figure, 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.

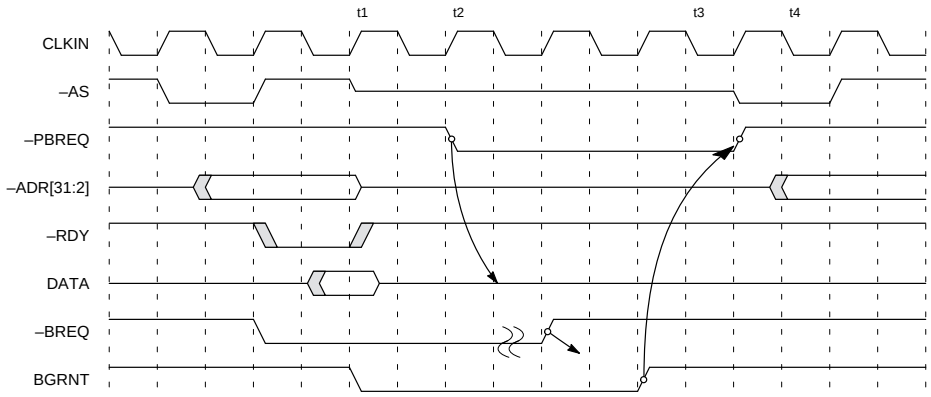


Figure D7–13. Example of –PBREQ timing

D7.12 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.
- A pending SDRAM interface request is recognized, and the SDRAM operation is completed.
- If there is a pending external bus request, the bus will be granted to the external requestor.
- If there is a pending DMA request, the bus will be granted to the DMA controller.
- If the write buffer is full, the buffer will be emptied.
- If there is a pending load or store operation it will be serviced.
- If there is a pending request for an instruction it will be fetched.
- If the prefetch buffer is empty, a prefetch cycle will be initiated.

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

CHAPTER

D8

MB86934 Debug Support Unit (DSU)

The MB86934 DSU functions identically to the MB86930 DSU for all instructions except the MB86934 FPop, EFPop, LDDF, STDF, and STDFQ instructions.

D8.1 Data Breakpoints Immediately Before FPop/EFPop

In the MB86934, the FPop1, FPop2, EFPop1, and EFPop2 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.
```

D8.2 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 MB86934, 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

CHAPTER

D9

Power Down Mode

The MB86934 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:

Table D9–1. Power–Down Register

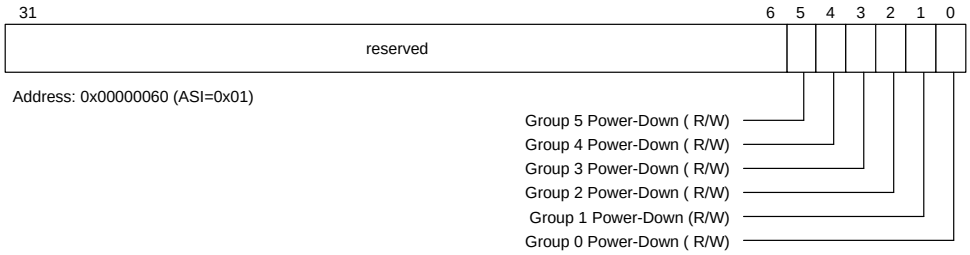
Group	Category	Processor Function
0	Independent	FPU
1	Independent	DMA
2	Dependent	Core (IU, BIU, I_cache, D_cache)
3	Independent	FIFO
4	Dependent	ICE
5	Independent	SDRAM Interface (SDIU)

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.

D9.1 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: Group 5 Power-Down (G5PD) — Controls power to the SDRAM interface. When set to 1, the SDRAM interface is powered-down; when cleared to 0, the SDRAM interface is powered-up.

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: Group 3 Power-Down (G3PD) — Controls power to the FIFO. When set to 1, the FIFO is powered-down; when cleared to 0, the FIFO is powered-up.

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.

Figure D9–1. Power–Down Register

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.

D9.2 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

D10

MB86934 External Interface

Table D10–1. 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 MB86934 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 OUTPUT1:** This is an output signal against which MB86934 bus transaction 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 OUTPUT2:** This is an output signal against which MB86934 bus transaction can be referenced. The CLKOUT2 frequency is the same as the frequency applied to XTAL1. CLKOUT2 is in phase with CLK_IN.
-LOCK	O S(L) G(Z) I(1)	**BUS LOCK:** This is a control signal asserted by the processor to indicate to the system that the current bus transaction requires more than one transfer on the bus. The Atomic Load Store instruction for example requires contiguous bus transactions which cause the assertion of the bus lock signal. The bus may not be granted to another bus owner as long as –LOCK is active. –LOCK is asserted with the assertion of AS as remains active until –READY is asserted at the end of the locked transaction.
-BREQ	O 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 MB86934 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.

Table D10–1. SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–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 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 wither a data or instruction access exception trap in the IU. The current bus access is invalidated by asserting the –MEXC in the same cycle as the –READY signal. The IU ignores the contents of the data bus in cycles where –MEXC is asserted.
IRL <3:0>	I A(L)	**INTERRUPT REQUEST BUS:** The value on these pins defines the external interrupt level. IRL <3:0>=1111 forces a non-maskable interrupt. IRL value of 0000 indicates no pending interrupts. Al l other values indicate maskable interrupts as enabled in the PIL field of the processor status register (PSR). Interrupts should be latched and prioritized by external logic and should be held pending until acknowledged by the processor.
–TIMER_OVF	O S(L) G(Q) I(Q)	**TIMER UNDERFLOW:** Asserted by the processor to indicate that the internal 16-bit timer has underflowed. This signal can be used to initiate a DRAM refresh cycle of a one cycle periodic waveform. On reset, the timer is turned off and –TIMER_OVF is high.
–SAME_PAGE	O S(L) G(1) I(1)	**SAME-PAGE DETECT:** The –SAME_PAGE signal is used to take advantage of fast consecutive accesses within the same page for Fast Page Mode DRAMs. This signal is an output which is asserted when the current access in the region defined by chip select 4 is in the same page as the previous access to chip select 4. The page size is specified by writing it the SAME_PAGE MASK register.
–CS0, –CS1, –CS2, –CS3, –CS4	O S(L) G(1) I(1)	**CHIP SELECTS:** These outputs are asserted when the value on the 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 internal ready after a user defined number of bus clock cycles. This allows a variety of memory and I/O devices with different access times to be connected to the MB86934 without the need for additional logic.
ADR <31:2>	I/O S(L) G(Z) I(1)	**ADDRESS BUS:** The 30-bit ADDRESS BUS (A31-A2) is an output which identifies the data or instruction address of a 32-bit word. Reads are always one word in size while byte, half-word, or word transaction sizes for writes is identified by separate byte-enable signals (–BE0-3). The address bus is valid for the duration of the bus transaction. ADR<14:2>, ADR< 24:19>are shared by the SDRAM interface.

Table D10–1. SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
ASI <3:0>	I/O S(L) G(Z) I(1)	**ADDRESS SPACE IDENTIFIERS:** The ADDRESS SPACE IDENTIFIERS are outputs which indicate to which of 256 available spaces the current ADDRESS BUS value corresponds. ASI values are defined as follows: ASI / ADDRESS SPACE 0x1 Control Register 0x2 Instruction Cache Lock 0x3 Data Cache Lock 0x4 - 0x7 Application Definable 0x8 User Instruction Space 0x9 Supervisor Instruction Space 0xA User Data Space 0xB Supervisor Data Space 0xC Instruction Cache Tag RAM 0xD Instruction Cache Data RAM 0xE Data Cache Tag RAM 0xF Data Cache Data RAM 0x10 - 0xFC Reserved 0xFD - 0xFF Reserved for Debug Hardware 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 asserted on the address bus. The ASI pins are valid for the duration of the bus transaction. ASI values 0x8, 0x9, 0xA, and 0xB are cacheable.
–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 memory. The MB86934 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) 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 memory. The MB86934 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]. 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.

Table D10–1. SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–BE3-0	O S(L) G(Z) I(O)	**BYTES 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 a follows: Byte0 Byte1 Byte2 Byte3 31 24 23 16 15 8 7 0 Byte Writes: 1 1 1 0 | 1 1 0 1 | 1 0 1 1 | 0 1 1 1 Half-Word Writes: 1 1 0 0 | 0 0 1 1 Word Writes: 0 0 0 0 BE<2:3> are also used in 8 and 16-bit ROM 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 <63:0>	I/O S(L) G(Z) I(1)	**DATA BUS:** The bus interface has 32 bi-directional 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-byte word. In write bus cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If the preceding cycle was a write, data is driven in the cycle immediately following the cycle in which –READY was asserted. If the preceding cycle was a read, data is driven one cycle after the cycle in which –READY was asserted to minimize bus contention between the processor and the system. Pins D<7:0> are used when the 8-bit boot mode is enabled and D<15:0> are used when16-bit mode is enabled. The SDRAM interface has 64 bidirectional pins D<63:0>. D<63:32> are used exclusively by the SDRAM interface. D<31:0> are shared by the SDRAM interface with the SPARClite bus interface. D<63:32> should have pull pull/down resistors if they are not used.
–AS	I/O S(L) G(Z) I(1)	**ADDRESS STROBE:** A control signal asserted by the MB86934 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	I/O S(L) G(Z) I(1)	**READ/WRITE 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.

Table D10–1. SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
–READY	I S(L)	**READY:** This is a control signal asserted by the external memory system to indicate that the current bus transaction is being completed and that it is ready to start with the next bus transaction in the following cycle. In case of a fetch from memory, the processor will strobe the value on the data bus at the rising edge 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 internally based on the address of the current transaction. The external system can override the internal ready generator to terminate the current bus cycle early. Up to 6 address ranges each with different transaction times can be programmed.
–DREQ0-1	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 transfer. –DREQ0 corresponds to DMA channel 0, while –DREQ1 corresponds to DMA channel 1.
–DACK0-1	O S(L)	**DMA ACKNOWLEDGE:** This is asserted when an external device asserts –DREQ and the processor accesses the external device. –DACK1 corresponds to DMA channel 0, while –DACK1 corresponds to DMA channel 1.
–EOP0-1	I/O S(L)	**END OF PROCESS:** The signal is asserted by the external device when it wants to terminate a DMA transfer. Alternately, the processor drives this signal when the byte count reaches zero. –EOP0 corresponds to DMA channel 0, while –EOP1 corresponds to DMA channel 1. A pull-up holds -EOP0-1 high when it is not being driven.
–PBREQ	O S(L)	**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 hand shake between the arbiter and the processor to allow the bus to be allocated based on demand.
–BMREQ	O S(L)	**BURST MODE REQUEST:** This signal is asserted by the processor to indicate to an 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 S(L)	**BURST MODE ACKNOWLEDGE:** This signal is asserted by the system to indicate that it can support burst mode for the address currently on the bus. The system asserts –BMACK in response to the processor asserting –BMREQ.
CLK_ECB	I	**EXTERNAL CLOCK BYPASS:** Tying these signal high causes the CLK_IN signal to bypass the Phases 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.

Table D10–1. SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
PARITY <3:0>	I/O S(L)	**PARITY:** When enabled this signal provides even or odd parity checking for the data bus. **Non SDRAM Operation:** Parity3 corresponds to D<31:24> Parity2 corresponds to D<23:16> Parity1 corresponds to D<15:8> Parity0 corresponds to D<7:0> **SDRAM Operation:** Parity3 corresponds to D<63:48> Parity2 corresponds to D<47:32> Parity1 corresponds to D<31:16> Parity0 corresponds to D<15:0>
–SWE	O S(L)	**SDRAM Write Enable:** This signal should be tied to the –WE input of SDRAM.
–SRAS	O S(L)	**SDRAM Row Address Strobe:** This signal should be tied to the –RAS input of SDRAM.
–SCAS	O S(L)	**SDRAM Column Address Strobe:** This signal should be tied to the –CAS input of SDRAM.
–SCS <3:0>	O S(L)	**SDRAM Chip Select:** Enables all command inputs, –RAS, –CAS, and –WE to SDRAM.
SCKE	O S(L)	**SDRAM Clock Enable:** This is an active high clock enable signal for SDRAM.
SDQM <1:0>	O S(L)	**SDRAM INPUT MASK/OUTPUT ENABLE:** SDQM<0> and SDQM<1> correspond to DATA<63:32>and DATA<31:0> respectively.
–NVWE	O S(L)	**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 is enabled for the current access.
–OE	O S(L)	**OUTPUT ENABLE:** This 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 S(L)	Ready Out for external Bus Masters using Internal Ready Generation.
–PDRESET	I S(L)	Power Down Reset is asserted by the external system to get the part out of powerdown mode. This signal should be asserted low during Reset Cycle.
BUICLOCK	I	This signal is reserved for future use and should be tied high.
EMU_SD <3:0>	I/O	**EMULATOR STATUS/DATA BITS:** Bi-directional pins used by a hardware emulator to control and monitor MB86934 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 MB86934 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 unconnected.
–EMU_ENB	I/O	**EMULATOR ENABLE:** Tied low while the MB86934 is being reset to enable hardware emulator mode on the chip. This pin should be left unconnected.

Table D10–1. SIGNAL DESCRIPTIONS (Continued)[1]

SYMBOL	TYPE	DESCRIPTION
TCK	I	**TEST CLOCK:** JTAG compatible test clock input.
TMS	I	**TEST MODE:** JTAG compatible test mode select pin. Test is enabled when –TMS is low.
TDI	I	**TEST DATA IN:** JTAG compatible test data input.
TDO	O	**TEST DATA OUT:** JTAG compatible test data output.
–TRST	I	**TEST RESET:** Asynchronous rest for JTAG logic. If not using JTAG, this signal must be pulled low.

1. In the following description, 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

D11

MB86934 JTAG

D11.1 MB86934 JTAG Pin List

The MB86934 JTAG cells are arranged in a shift register configuration (see Figure D11–1. 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 D11–1 lists the order of all of the JTAG cells.

Table D11–1. JTAG Pin Order

Order	JTAG Cell	JTAG Cell Type	Function
1	EMU_SD_i<3>	input	Input bit 3 of EMU_SD<3:0> bus
2	EMU_SD_o<3>	output	Output bit 3 of EMU_SD<3:0> bus
:	:	:	
7	EMU_SD_i<0>	input	Input bit 0 of EMU_SD<3:0> bus
8	EMU_SD_o<0>	output	Output bit 0 of EMU_SD<3:0> bus
9	EMU_D_i<3>	input	Input bit 3 of EMU_D<3:0> bus
10	EMU_D_o<3>	output	Output bit 3 of EMU_D<3:0> bus

Table D11–1. JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
:	:	:	
15	EMU_D_i<0>	input	Input bit 0 of EMU_D<3:0> bus
16	EMU_D_o<0>	output	Output bit 0 of EMU_D<3:0> bus
17	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
18	–EMU_EN_i	input	Input bit of –EMU_ENB pin
19	–EMU_EN_o	output	Output bit of –EMU_ENB pin
20	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
21	EMU_BRK	input	Emulator break input
22	–DACK0	output	
23	–EOP0_i	input	
24	–EOP0_o	output	
25	eopio0	output	Bidirectional control for –EOP0 pin eopio0 = 1: –EOP0 is input eopio0 = 0: –EOP0 is output
26	–DREQ0	input	
27	–DACK1	output	
28	–EOP1_i	input	
29	–EOP1_o	output	
30	eopio1	output	Bidirectional control for –EOP1 pin eopio1 = 1: –EOP0 is input eopio1 = 0: –EOP0 is output
31	–DREQ1	input	
32	ADR_i<2>	input	
33	ADR_o<2>	output	
:	:	:	
90	ADR_i<31>	input	
91	ADR_o<31>	output	
92	addenbjo	output	Bidirectional control for ADR<31:2> addenbjo = 1: ADR<31:2> are inputs addenbjo = 0: ADR<31:2> are outputs
93	ASI_i<0>	input	
94	ASI_o<0>	output	

Table D11–1. JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
:	:	:	
99	ASI_i<3>	input	
100	ASI_o<3>	output	
101	BE<0>	output	
102	BE<1>	output	
103	BE<2>	output	
104	BE<3>	output	
105	–SAMEPAGE	output	
106	–CS<0>	output	
107	–CS<1>	output	
108	–CS<2>	output	
109	–CS<3>	output	
110	–CS<4>	output	
111	XTAL1	input	Crystal input
112	SCKE	output	
113	-SWE	output	
114	SDQM<0>	output	
115	SDQM<1>	output	
116	-SCS<0>	output	
:	:	:	
119	-SCS<3>	output	
120	-SCAS	output	
121	-SRAS	output	
122	-OE	output	
123	-NVWE	output	
124	–ERROR	output	Error output signal
125	–LOCK	output	Bus lock output signal
126	tstatejo†	output	Three-state control signal for ASI, –BE, –AS, RD/WR and –LOCK If tstatejo = 1: signals are three-stated. If tstatejo = 0: signals are outputs.
127	–BGRNT	output	Bus grant output signal
128	–PBREQ	output	

Table D11–1. JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
129	–BMREQ	output	
130	–RD/WR_i	input	Memory Read/Write output signal
131	-RD/WR_o	output	
132	–AS_i	input	Start of memory transaction output signal
133	-AS_o	output	
134	-READYOUT	output	
135	–READY	input	External memory transaction complete signal
136	–MEXC	input	Memory exception input
137	–BMACK	input	
138	–BREQ	input	Bus request input
139	–RESET	input	Chip reset pin
140	-PDRESET	input	
141	dbusiojo†	output	Bidirectional control signal for D<63:0>, Parity <3:0> dbusiojo = 1: D<63:0>, Parity <3:0> are inputs dbusiojo = 0: D<63:0>, Parity <3:0> are inputs
142	D_i<0>	input	
143	D_o<0>	output	
:	:	:	
268	D_i<63>	input	
269	D_o<63>	output	
270	PARITY_i<0>	input	
271	PARITY_o<0>	output	
:	:	:	
276	PARITY_i<3>	input	
277	PARITY_o<3>	output	
278	-TIMER__OVF	output	Timer Overflow pin
279	BIUCLOCK	input	
280	–CLKDBL	input	
281	CLK_ECB	input	
282	–BMODE8	input	
283	–BMODE16	input	

Table D11–1. JTAG Pin Order (Continued)

Order	JTAG Cell	JTAG Cell Type	Function
284	IRL<0>	input	
285	IRL<1>	input	
286	IRL<2>	input	
287	IRL<3>	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.

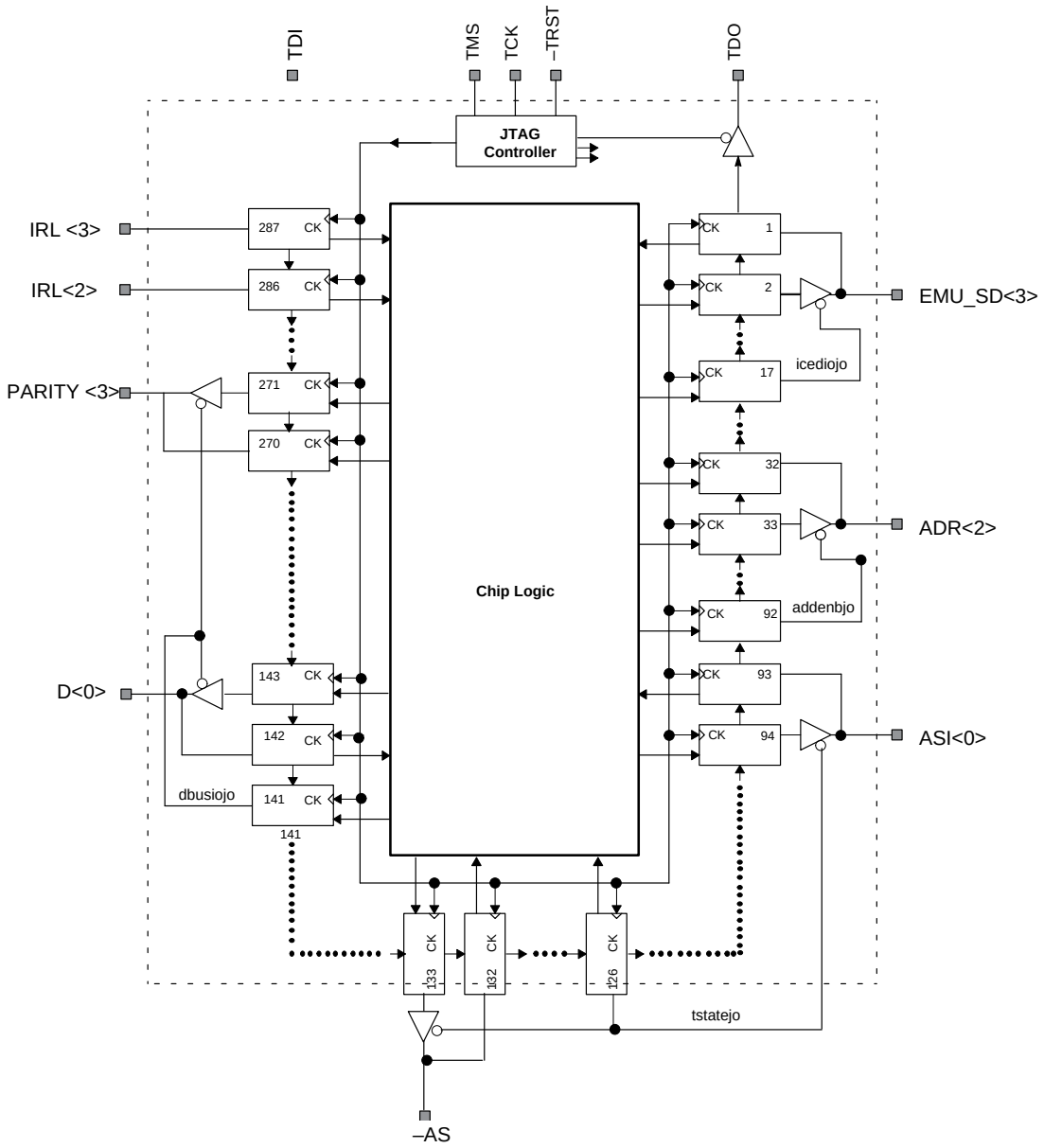


Figure D11–1. JTAG Cell Organization

FUJITSU

MB866934 Addendum, January 1996, Edition 1.0

Visit our web site for the latest information:

http://www.fujitsumicro.com

Customer Response Center:

For semiconductor products, flat panel displays, and PC cards in the U.S., Canada and South America, please contact the Fujitsu Microelectronics Customer Response Center (CRC). The CRC provides a single point of contact for resolving customer issues and answering technical questions.

Web: Click on Tech Support in the FMI home page, then submit our form

Tel: Telephone: 1–800–866–8608 Monday through Friday, 7 to 5 PST

Fax: (408) 922–9179

E–Mail: fmicrc@fmi.fujitsu.com

EC–UM–20229–1/96