



---

---

# Implementation Characteristics of Current SPARC-V9 -based Products

---

---

Version: 2-9-99

**V9**

**SPARC INTERNATIONAL**



© 1998 SPARC International Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

Any comments relating to the material contained herein may be submitted to:

SPARC International Inc.

3333 Bowers Ave., Suite 280

Santa Clara, CA 95054-2913

TEL (408) 748-9111

FAX (408) 748-9777

ATTN: Ghassan Abbas (abbas@sparc.com)

### **Trademarks**

SPARC® is a registered trademark of SPARC International, Inc.

SPARCstation™, UltraSPARC, SPARC 64 are trademark of SPARC International, Inc.

Products bearing SPARC® trademarks are based on an architecture developed by Sun Microsystems, Inc.

ONC™, Solaris and SunOS™ are trademarks of Sun Microsystems, Inc.

NFS® is a registered trademark of Sun Microsystems, Inc.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations. SPARC International, Inc. disclaims any responsibility for specifying which trademarks are owned by which companies or organizations.



<b>Preface</b>	<b>19</b>
Audience and Purpose	19
Organization and Content	19
<b>CHAPTER 1: HAL SPARC64</b>	<b>23</b>
0. Introduction	23
1. Software emulated instructions	23
2. Number of IU registers	24
3. Incorrect IEEE Std 754-1985 results	24
4-5. Reserved	25
6. I/O registers privileged status	25
7. I/O register definitions	25
8-9. RDASR/WRASR target registers and privileged status	25
10-12 Reserved	25
13. VER.impl	26
14-15 Reserved	26
16. IU deferred-trap queue	26
17. Reserved	26
18. Nonstandard IEEE 754-1985 results	26
19. FPU version, FSR.ver	26
20-21. Reserved	26
22. FPU TEM, cexc, and aexc	26
23. Floating-point traps	27
24. FPU deferred-trap queue (FQ)	27
25. RDPR of FQ with nonexistent FQ	27
26-28. Reserved	27
29,30. Address space identifier (ASI) definitions and ASI address decoding	27
31. Catastrophic error exceptions	29
32. Deferred traps	29
33. Trap precision	29
34. Interrupt clearing	29
35,36. Implementation-dependent traps and priorities	30
37. Reset trap	31
38. Effect of reset trap on implementation-dependent registers	31
39. Entering error_state on implementation-dependent errors	31
40. Error_state processor state	31
41. Reserved	31
42. FLUSH instruction	31
43. Reserved	32
44. Data access FPU trap	32
45-46. Reserved	32
47. RDASR	32

48. WRASR	32
49-54 Reserved	33
55. Floating-point underflow detection	33
56-100. Reserved	33
101. Maximum trap level	33
102. Clean window trap	33
103. Prefetch instructions	33
104. VER.manuf	34
105. TICK register	34
106. IMPDEPn instructions	35
107. Unimplemented LDD trap	35
108. Unimplemented STD trap	35
109. LDDF_mem_address_not_aligned	35
110. STDF_mem_address_not_aligned	36
111. LDQF_mem_address_not_aligned	36
112. STQF_mem_address_not_aligned	36
113. Implemented memory models	36
114. RED_state trap vector address (RSTVaddr)	36
115. RED_state processor state	37
116. SIR_enable control flag	37
117. MMU disabled prefetch behavior	37
118. Identifying I/O locations	38
119. Unimplemented values for PSTATE.MM	38
120. Coherence and atomicity of memory operations	38
121. Implementation-dependent memory model	38
122. FLUSH latency	38
123. Input/output (I/O) semantics	39
124. Implicit ASI when TL>0	39
125. Address masking	39
126. TSTATE bits 19:18	39
127. PSTATE bits 11:10	39
128. CLEANWIN register update	40
<b>CHAPTER 2: SUN ULTRASPARC</b>	<b>43</b>
1. Software emulation of instructions	43
2. Number of IU registers	44
3. Incorrect IEEE Std 754-1985 results	44
6. I/O registers privileged status	45
7. I/O register definitions	45
8. RDASR/WRASR target registers	46
9. RDASR/WRASR privileged status	46
10 - 12. Reserved	47

---

13. VER.impl	47
14 - 15. Reserved	47
16. IU deferred-trap queue	47
17. Reserved	48
18. Nonstandard IEEE 754-1985 results	48
19. FPU version, FSR.ver	48
20 - 21. Reserved	48
22. FPU TEM. cexc. and aexc	48
23. Floating-point traps	48
24. FPU deferred-trap queue (FQ)	49
25. RDPR of FQ with nonexistent FQ	49
26 - 28. Reserved	49
29 Address space identifier (ASI) definitions	49
30. ASI address decoding	50
31. Catastrophic error exceptions	50
32. Deferred traps	50
33. Trap precision	51
34. Interrupt clearing	51
35. Implementation-dependent traps	51
36. Trap priorities	52
37. Reset trap	52
38. Effect of reset trap on implementation-dependent registers	52
39. Entering error_state on implementation-dependent errors	52
40. Error_state processor state	53
41. Reserved	53
42. FLUSH instruction	53
43. Reserved	53
44. Data access FPU trap	53
45-46. Reserved	54
47. RDASR	54
48. WRASR	54
49-54. Reserved	54
55. Floating-point underflow detection	54
56 - 100. Reserved	55
101. Maximum trap level	55
102. Clean window trap	55
103. Prefetch instructions	55
104. VER.manuf	55
105. TICK register	56
106. IMPDEP1 instructions	56
107. Unimplemented LDD trap	58
108. Unimplemented STD trap	58

109. LDDF_mem_address_not_aligned	59
110. STDF_mem_address_not_aligned	59
111. LDQF_mem_address_not_aligned	59
112. STQF_mem_address_not_aligned	59
113. Implemented memory models	60
114. RED_state trap vector address (RSTVaddr)	60
115. RED_state processor state	60
116. SIR_enable control flag	60
117. MMU disabled prefetch behavior	61
118. Identifying I/O locations	61
119. Unimplemented values for PSTATE.MM	61
120. Coherence and atomicity of memory operations	61
121. Implementation-dependent memory model	62
122. FLUSH latency	62
123. Input/output (I/O) semantics	62
124. Implicit ASI when TL>0	62
125. Address masking	63
126. TSTATE bits 19:18	63
127. PSTATE bits 11:10	63
<b>CHAPTER 3: HAL SPARC64-II</b>	<b>67</b>
0. Introduction	67
1. Software emulated instructions	67
2. Number of IU registers	68
3. Incorrect IEEE Std 754-1985 results	68
4-5. Reserved	68
6. I/O registers privileged status	68
7. I/O register definitions	69
8,9. RDASR/WRASR target registers and privileged status	69
10-12 Reserved	70
13. VER.impl	70
14-15 Reserved	70
16. IU deferred-trap queue	70
17. Reserved	70
18. Nonstandard IEEE 754-1985 results	70
19. FPU version, FSR.ver	71
20-21. Reserved	71
22. FPU TEM, cexc, and aexc	71
23. Floating-point traps	71
24. FPU deferred-trap queue (FQ)	71
25. RDPR of FQ with nonexistent FQ	72
26-28. Reserved	72



29,30. Address space identifier (ASI) definitions and ASI address decoding	72
31. Catastrophic error exceptions	73
32. Deferred traps	73
33. Trap precision	74
34. Interrupt clearing	74
35,36. Implementation-dependent traps and priorities	74
37. Reset trap	75
38. Effect of reset trap on implementation-dependent registers	75
39. Entering error_state on implementation-dependent errors	75
40. Error_state processor state	76
41. Reserved	76
42. FLUSH instruction	76
43. Reserved	76
44. Data access FPU trap	76
45-46. Reserved	77
47. RDASR	77
48. WRASR	77
49-54 Reserved	77
55. Floating-point underflow detection	77
56-100. Reserved	78
101. Maximum trap level	78
102. Clean window trap	78
103. Prefetch instructions	78
104. VER.manuf	79
105. TICK register	79
106. IMPDEPn instructions	80
107. Unimplemented LDD trap	80
108. Unimplemented STD trap	80
109. LDDF_mem_address_not_aligned	81
110. STDF_mem_address_not_aligned	81
111. LDQF_mem_address_not_aligned	81
112. STQF_mem_address_not_aligned	81
113. Implemented memory models	82
114. RED_state trap vector address (RSTVaddr)	82
115. RED_state processor state	82
116. SIR_enable control flag	83
117. MMU disabled prefetch behavior	83
118. Identifying I/O locations	83
119. Unimplemented values for PSTATE.MM	83
120. Coherence and atomicity of memory operations	83
121. Implementation-dependent memory model	84
122. FLUSH latency	84

123. Input/output (I/O) semantics	84
124. Implicit ASI when TL>0	84
125. Address masking	85
126. TSTATE bits 19:18	85
127. PSTATE bits 11:10	85
128. CLEANWIN register update	85
<b>CHAPTER 4: SUN ULTRASPARC II</b>	<b>89</b>
1. Software emulation of instructions	89
2. Number of IU registers	90
3. Incorrect IEEE Std 754-1985 results	91
4-5. Reserved	92
6. I/O registers privileged status	92
7. I/O register definitions	92
8. RDASR/WRASR target registers	92
9. RDASR/WRASR privileged status	93
10-12. Reserved	93
13. VER.impl	93
14-15. Reserved	94
16. IU deferred-trap queue	94
17. Reserved	94
18. Nonstandard IEEE 754-1985 results	94
19. FPU version, FSR.ver	94
20-21. Reserved	94
22. FPU TEM, cexc, and aexc	95
23. Floating-point traps	95
24. FPU deferred-trap queue (FQ)	95
25. RDPR of FQ with nonexistent FQ	95
26-28. Reserved	95
29. Address space identifier (ASI) definitions	96
30. ASI address decoding	96
31. Catastrophic error exceptions	97
32. Deferred traps	97
33. Trap precision	97
34. Interrupt clearing	97
35. Implementation-dependent traps	98
36. Trap priorities	98
37. Reset trap	98
38. Effect of reset trap on implementation-dependent registers	99
39. Entering error_state on implementation-dependent errors	99
40. Error_state processor state	99
41. Reserved	99

42. FLUSH instruction	99
43. Reserved	100
44. Data access FPU trap	100
45-46. Reserved	100
47. RDASR	100
48. WRASR	100
49-54. Reserved	101
55. Floating-point underflow detection	101
56-100. Reserved	101
101. Maximum trap level	101
102. Clean window trap	101
103. Prefetch instructions	102
104. VER.manuf	102
105. TICK register	102
106. IMPDEPn instructions	103
107. Unimplemented LDD trap	105
108. Unimplemented STD trap	105
109. LDDF_mem_address_not_aligned	105
110. STDF_mem_address_not_aligned	105
111. LDQF_mem_address_not_aligned	106
112. STQF_mem_address_not_aligned	106
113. Implemented memory models	106
114. RED_state trap vector address (RSTVaddr)	106
115. RED_state processor state	107
116. SIR_enable control flag	107
117. MMU disabled prefetch behavior	107
118. Identifying I/O locations	107
119. Unimplemented values for PSTATE.MM	108
120. Coherence and atomicity of memory operations	108
121. Implementation-dependent memory model	108
122. FLUSH latency	108
123. Input/output (I/O) semantics	109
124. Implicit ASI when TL > 0	109
125. Address masking	109
126. TSTATE bits 19:18	109
127. PSTATE bits 11:10	110
<b>CHAPTER 5: SUN ULTRASPARC Iii</b>	<b>113</b>
1. Software emulation of instructions	113
2. Number of IU registers	114
3. Incorrect IEEE Std 754-1985 results	115
4-5. Reserved	116

---

6. I/O registers privileged status	116
7. I/O register definitions	116
8. RDASR/WRASR target registers	116
9. RDASR/WRASR privileged status	117
10-12. Reserved	117
13. VER.impl	117
14-15. Reserved	118
16. IU deferred-trap queue	118
17. Reserved	118
18. Nonstandard IEEE 754-1985 results	118
19. FPU version, FSR.ver	118
20-21. Reserved	118
22. FPU TEM, cexc, and aexc	119
23. Floating-point traps	119
24. FPU deferred-trap queue (FQ)	119
25. RDPR of FQ with nonexistent FQ	119
26-28. Reserved	119
29. Address space identifier (ASI) definitions	120
30. ASI address decoding	120
31. Catastrophic error exceptions	120
32. Deferred traps	121
33. Trap precision	121
34. Interrupt clearing	121
35. Implementation-dependent traps	122
36. Trap priorities	122
37. Reset trap	122
38. Effect of reset trap on implementation-dependent registers	123
39. Entering error_state on implementation-dependent errors	123
40. Error_state processor state	123
41. Reserved	123
42. FLUSH instruction	123
43. Reserved	124
44. Data access FPU trap	124
45-46. Reserved	124
47. RDASR	124
48. WRASR	124
49-54. Reserved	125
55. Floating-point underflow detection	125
56-100. Reserved	125
101. Maximum trap level	125
102. Clean window trap	125
103. Prefetch instructions	125

104. VER.manuf	126
105. TICK register	127
106. IMPDEPn instructions	127
107. Unimplemented LDD trap	129
108. Unimplemented STD trap	129
109. LDDF_mem_address_not_aligned	129
110. STDF_mem_address_not_aligned	130
111. LDQF_mem_address_not_aligned	130
112. STQF_mem_address_not_aligned	130
113. Implemented memory models	131
114. RED_state trap vector address (RSTVaddr)	131
115. RED_state processor state	131
116. SIR_enable control flag	131
117. MMU disabled prefetch behavior	131
118. Identifying I/O locations	132
119. Unimplemented values for PSTATE.MM	132
120. Coherence and atomicity of memory operations	132
121. Implementation-dependent memory model	132
122. FLUSH latency	133
123. Input/output (I/O) semantics	133
124. Implicit ASI when TL > 0	133
125. Address masking	133
126. TSTATE bits 19:18	134
127. PSTATE bits 11:10	134

## CHAPTER 6: HAL SPARC64-III 137

0. Introduction	137
1. Software emulated instructions	137
2. Number of IU registers	137
3. Incorrect IEEE Std 754-1985 results	138
4-5. Reserved	138
6. I/O registers privileged status	138
7. I/O register definitions	138
8,9. RDASR/WRASR target registers and privileged status	139
10-12 Reserved	139
13. VER.impl	139
14-15 Reserved	140
16. IU deferred-trap queue	140
17. Reserved	140
18. Nonstandard IEEE 754-1985 results	140
19. FPU version, FSR.ver	140
20-21. Reserved	140

22. FPU TEM, cexc, and aexc	140
23. Floating-point traps	141
24. FPU deferred-trap queue (FQ)	141
25. RDPR of FQ with nonexistent FQ	141
26-28. Reserved	141
29,30. Address space identifier (ASI) definitions and ASI address decoding	141
31. Catastrophic error exceptions	142
32. Deferred traps	142
33. Trap precision	142
34. Interrupt clearing	143
35,36. Implementation-dependent traps and priorities	143
37. Reset trap	143
38. Effect of reset trap on implementation-dependent registers	144
39. Entering error_state on implementation-dependent errors	144
40. Error_state processor state	144
41. Reserved	144
42. FLUSH instruction	144
43. Reserved	145
44. Data access FPU trap	145
45-46. Reserved	145
47. RDASR	145
48. WRASR	145
49-54 Reserved	146
55. Floating-point underflow detection	146
56-100. Reserved	146
101. Maximum trap level	146
102. Clean window trap	146
103. Prefetch instructions	146
104. VER.manuf	147
105. TICK register	148
106. IMPDEPn instructions	148
107. Unimplemented LDD trap	148
108. Unimplemented STD trap	148
109. LDDF_mem_address_not_aligned	149
110. STDF_mem_address_not_aligned	149
111. LDQF_mem_address_not_aligned	149
112. STQF_mem_address_not_aligned	149
113. Implemented memory models	150
114. RED_state trap vector address (RSTVaddr)	150
115. RED_state processor state	150
116. SIR_enable control flag	150
117. MMU disabled prefetch behavior	151

118. Identifying I/O locations	151
119. Unimplemented values for PSTATE.MM	151
120. Coherence and atomicity of memory operations	151
121. Implementation-dependent memory model	151
122. FLUSH latency	152
123. Input/output (I/O) semantics	152
124. Implicit ASI when TL>0	152
125. Address masking	152
126. TSTATE bits 19:18	153
127. PSTATE bits 11:10	153
128. CLEANWIN register update	153
<b>APPENDIX A: VER.impl/VER.manuf</b>	<b>157</b>
<b>APPENDIX B: SPARC V9 Arch Book Changes</b>	<b>161</b>
Change to page 13	161
Change to page 21(r142)	161
Change to page 28(r142)	161
Change to page 30(r142)	161
Change to page 40(r142),	161
Change to page 51	162
Change to page 52(r142)	162
Change to page 55(r142)	162
Change to page 56(r142)	162
Change to page 57(r142)	163
Change to page 58-9(r142)	163
Change to page 76,	163
Change to page 80(r142), 6.3.6.4(r142)	163
Change to page 81(r141/r142):	163
Change to page 81(r141/r142):	163
Change to page 121(r141/r142):	163
Change to page 151(r142), A.9(r142),	164
Change to page 171	164
Change to page 181(r141/r142):	164
Change to page 191(r141/r142):	164
Change to page 195(r141/r142):	164
Change to page 212(r14[123]) A.43(r14[12])/A.44(r143),	164
Change to page 216(r142), A.46(r142),	164
Change to page 220(r142)/A.49(r142)	165
Change to page 228(r141/r142):	165
Change to page 229(r142)/A.55(r142),	165

Change to page 231(r142)/233(r143),	165
Change to page 234, A.58(r14[12])/A.59(r143)	165
Change to page 241(r142), A.62(r142),	165
Change to page 242(r142), A.62(r142)	166
Change to page 253(r142)	166
Change to page 253(4142)	166
Change to page 255(r142)	166
Change to page 258(r142)	166
Change to page 268(r142)	167
Change to page 290(r142)	167
Change to page 312(r142)	167





**Preface**

---

---

**V9**



# Preface

## Audience and Purpose

The SPARC International *Implementation Characteristics of Current SPARC-V9-based Products* is intended as companion to the SPARC Architecture Book Version 9.

## Organization and Content

This document has been divided as follows

Table of Content

Preface

Chapter 1: HAL SPARC64

Chapter 2: SUN ULTRASPARC

Chapter 3: HAL SPARC64-II

APPENDIX A: VER.impl/VER.manuf

APPENDIX B: SPARC V9 Arch Book Changes

Index



**Chapter 1: HAL Implementation of V9 Architecture**

---

---

**SPARC 64**

---

---

**V9**

**SPARC INTERNATIONAL**



# CHAPTER 1: HAL SPARC64

## 0. Introduction

This document describes the implementation details of the *SPARC64™* processor developed by *HAL Computer Systems*. The items listed below correspond to the implementation dependencies as listed in the text and by number in Appendix C of “*The SPARC Architecture Manual - Version 9*” by *SPARC International*, along with the description of the implementation dependency. The “Implementation” section for each item describes the implementation on the SPARC64 processor.

## 1. Software emulated instructions

Description: Whether an instruction is implemented directly by hardware, simulated by software, or emulated by firmware is implementation-dependent.

Implementation: SPARC64 does not implement the following instructions in hardware: All floating point instructions with quad operands or results

These operations will take an *fp\_exception\_other* trap with *FSR.ftt = unimplemented\_FPop*. The kernel will then emulate the quad operation and store the result into a quad-aligned set of floating-point registers as defined by SPARC-V9 manual.

*fsqrd, fsqrts*: Executing these instructions will cause a *fp\_exception\_other* exception with *FSR.ftt = unimplemented\_FPop*. In this case kernel emulation routines are provided to complete the instructions.

*flush*: This instruction will cause an *illegal\_instruction* trap if executed. Kernel emulation routines will be provided to flush the cache line from the data cache and invalidate any matching cache lines in the instruction cache.

*ldd, ldda, std, stda*: Executing these instructions in normal mode would generate *unimplemented\_LDD* and *unimplemented\_STD* trap. Kernel emulation routines will be provided to complete the instructions. SPARC64 also implements a special accelerated emulation trap handling for certain LDD and STD instructions, if a special mode is chosen.

*popc*: This instruction will cause an *illegal\_instruction* trap if executed.

Kernel emulation routines will be provided to complete the action.

## 2. Number of IU registers

**Description:** An implementation of the IU may contain from 64 to 258 general purpose 64 bit r registers. This corresponds to a grouping of the registers into two sets of eight global r registers, plus a circular stack of from three to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (NWINDOWS) is implementation-dependent, the total number of registers is also implementation-dependent.

**Implementation:** SPARC64 implements 4 16-register sets (windows) in hardware. Thus there are a total of 80 integer registers visible to software. They are:

- 8 global registers
- 8 alternate global registers
- 4 windows of 16 registers each (=64 registers)

## 3. Incorrect IEEE Std 754-1985 results

**Description:** An implementation may indicate that a floating-point instruction did not produce a correct ANSI/IEEE Standard 754-1985 result by generating a special floating-point unfinished or unimplemented exception. In this case, privileged mode software shall emulate any functionality not present in the hardware.

**Implementation:** SPARC64 in conjunction with the kernel emulation code produces the correct IEEE 754 results required in this section.

### 1) Traps Inhibit Results

SPARC64 in conjunction with the kernel emulation code produces results required.

### 2) Trapped Underflow Definition (UFM=1)

SPARC64 detects “tininess” before rounding as recommended.

### 3) Untrapped Underflow Definition (UFM=0)

SPARC64 meets these requirements with some help from the kernel divide fixup code.

### 4) Floating-Point Non standard Mode

SPARC64 FPU is “standard”, and therefore does not support a nonstandard mode.



## 4-5. Reserved

### 6. I/O registers privileged status

Description: Whether I/O registers can be accessed by non privileged code is implementation-dependent.

Implementation: In SPARC64 some I/O registers can be accessed by non privileged code.

### 7. I/O register definitions

Description: The contents and addresses of I/O registers are implementation-dependent.

Implementation: Please contact HAL for details of I/O registers.

### 8-9. RDASR/WRASR target registers and privileged status

Description: Software can use read/write ancillary state register instructions to read/write implementation-dependent processor registers (ASRs 16-31). Whether each of the implementation-dependent read/write ancillary state register instructions (for ASRs 16-31) is privileged is implementation dependent.

Implementation: SPARC64 implements 7 implementation-dependent ASR registers. LDD Trap Base Address (ASR24) This privileged read/write register specifies a special trap base address for some *unimplemented\_LDD* and *unimplemented\_STD* traps. Instruction Emulation Register (ASR25) This read only register is written by CPU on a trap for a LDD/STD that uses the LDD Trap Base Address described above. Data Breakpoint Register (ASR26) This privileged write-only register is used to trap any data accesses to a double word aligned breakpoint address. Software Initiated Reset (ASR27) A write to this register with a WRASR instruction will cause a software initiated reset (SIR). An SIR is a precise trap. ASR27 is privileged and write-only. Fault Address Register (ASR28) and Fault Access Type (ASR29) These registers facilitate the handling of traps that involve a data memory access. The registers are privileged and read-only. System software must take care to read these registers on entry to a fault handler before any other fault can occur that would overwrite them. State Control Register (ASR31) ASR31 is a 16bit implementation specific register that contains a set of flags for controlling the state of the CPU, MMU and Caches. The register is privileged and can be read/written.

## 10-12 Reserved

### 13. VER.impl

Description: VER.impl uniquely identifies an implementation or class of software-compatible implementations of the architecture. Values FFF0(hex)..FFFF(hex) are reserved and are not available for assignment.

Implementation: SPARC64 uses a version number of 1.

### 14-15 Reserved

### 16. IU deferred-trap queue

Description: The existence, contents, and operation of an IU deferred-trap queue are implementation-dependent; it is not visible to user application programs under normal operating conditions

Implementation: SPARC64 does not need and therefore does not implement an IU deferred-trap queue.

### 17. Reserved

### 18. Nonstandard IEEE 754-1985 results

Description: Bit 22 of the FSR, FSR\_nonstandard\_fp (NS), when set to 1, causes the FPU to produce implementation-defined results that may not correspond to IEEE Standard 754-1985.

Implementation: SPARC64 FPU is “standard”, and therefore does not support a nonstandard mode.

### 19. FPU version, FSR.ver

Description: Bits 19:17 of the FSR, FSR.ver, identify one or more implementations of the FPU architecture.

Implementation: SPARC64 uses the value of 0 for this field.

### 20-21. Reserved

### 22. FPU TEM, cexc, and aexc

Description: An implementation may choose to implement the TEM, cexc, and aexc fields in hardware in either of two ways (see section 5.1.7.11 of SPARC-V9

Architecture Manual for details).

Implementation: SPARC64 implements TEM, cexc and aexc fields of FSR conforming to IEEE Std. 754-1985.

### 23. Floating-point traps

Description:

Floating point traps may be precise or deferred. If deferred, a floating point deferred-trap queue (FQ) must be present.

Implementation: The only deferred traps in SPARC64 are: *fp\_exception\_other* (*ftt = unfinished\_FPop*) for FDIV with unusual arguments and the *data\_breakpoint* trap. SPARC64 does not need a floating-point deferred-trap queue because the FDIV that caused the trap is the only deferred instruction.

### 24. FPU deferred-trap queue (FQ)

Description: The presence, contents of, and operations on the floating-point deferred-trap queue (FQ) are implementation-dependent.

Implementation: SPARC64 does not have or need a floating-point deferred-trap queue.

### 25. RDPR of FQ with nonexistent FQ

Description: On implementations without a floating-point queue, an attempt to read the FQ with an RDPR instruction shall cause either an *illegal\_instruction* exception or an *fp\_exception\_other* exception with FSR.ftt set to 4 (*sequence\_error*).

Implementation: A RDPR of %FPQ instruction will cause an *illegal\_instruction* trap.

### 26-28. Reserved

### 29,30. Address space identifier (ASI) definitions and ASI address decoding

Description: The following ASI assignments are implementation-dependent: restricted ASIs (all values hex) 00..03, 05..0B, 0D..0F, 12..17, and 1A..7F; and unrestricted ASIs C0..FF.

An implementation may choose to decode only a subset of the 8-bit ASI specifier; however, it shall decode at least enough of the ASI to distinguish ASI\_PRIMARY, ASI\_PRIMARY\_LITTLE, ASI\_AS\_IF\_USER\_PRIMARY, ASI\_AS\_IF\_USER\_PRIMARY\_LITTLE, ASI\_PRIMARY\_NOFAULT,

ASI\_PRIMARY\_NOFAULT\_LITTLE, ASI\_SECONDARY, ASI\_SECONDARY\_LITTLE, ASI\_AS\_IF\_USER\_SECONDARY, ASI\_AS\_IF\_USER\_SECONDARY\_LITTLE, ASI\_SECONDARY\_NOFAULT, and ASI\_SECONDARY\_NOFAULT\_LITTLE. If ASI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE are supported (impl. dep. #124), they must be decoded also. Finally, an implementation must always decode ASI bit<7> while PSTATE.PRIV = 0, so that an attempt by nonprivileged software to access a restricted ASI will always cause a `privileged_action` exception.

Implementation: The encoding of ASIs in the SPARC64 processor is shown below:

NR	V (M3)	PO	AS_IF	LE	M2	M1	M0
7	6	5	4	3	2	1	0

NR (Non-Restricted). This bit conforms to SPARC-V9 definition. An attempt to use a restricted ASI in non-privileged mode results in a *privileged\_action* trap.

V (Vendor-specific). This bit conforms to SPARC-V9 definition for non-restricted ASIs that are implementation-dependent (0xc0 - 0xff). This bit will be set in all ASIs that are specific to SPARC64.

PO (Program Order). An instruction using an ASI with this bit set is executed by SPARC64 strictly in program order.

AS\_IF. This bit conforms to SPARC-V9 requirement that there be an implementation specific ASI encoding that allows the corresponding access to be made as if the CPU were executing in non-privileged mode, independent of PSTATE.PRIV.

LE. This bit conforms to SPARC-V9 definition of ASIs that specify little-endian byte ordering. If this bit is set to zero, the access is done using big-endian byte ordering.

M2..M0. These bits are interpreted by the SPARC64 MMU.

SPARC64 does not support a nucleus context and hence does not decode ASI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE.

### 31. Catastrophic error exceptions

**Description:** The causes and effects of catastrophic error exceptions are implementation-dependent. They may cause precise, deferred or disrupting traps.

**Implementation:** An internal CPU watchdog time-out occurs after no instruction has been committed for  $2^{*}n$  cycles ( $n$  can be scan initialized to one of {12,14,16,18,19,20,21,22,24}, with 24 being the default value). This would take the processor into error state.

### 32. Deferred traps

**Description:** Whether any deferred traps (and associated deferred-trap queues) are present is implementation-dependent.

**Implementation:** SPARC64 implements a deferred trap for the following trap types:  
fp\_exception\_other (when FSR.ftt = unfinished\_FPop).  
data\_breakpoint.

Deferred trap queues are not necessary, since the trapping instruction is the only deferred instruction.

### 33. Trap precision

**Description:** Exceptions that occur as the result of program execution may be precise or deferred, although it is recommended that such exceptions be precise. Examples include mem\_address\_not\_aligned and division\_by\_zero.

**Implementation:** SPARC64 will generate a precise trap for all traps induced by instruction execution, except for *unfinished\_FPop*, *data\_breakpoint* and *Chip\_crossing\_errors (CPU\_xing)*.

### 34. Interrupt clearing

**Description:** How quickly a processor responds to an interrupt request and the method by which an interrupt request is removed are implementation-dependent.

**Implementation:** When SPARC64 is ready to accept an interrupt signal (based on PSTATE.IE and the PIL), it stops issuing instructions and waits for the CPU to quiesce. It then issues instructions from the corresponding trap handler if the interrupt condition is still valid. The TPC points to the instruction that would have executed in the absence of the interrupt. All instructions prior to the TPC have completed and all instructions including and subsequent to TPC remain unexecuted.

### 35,36. Implementation-dependent traps and priorities

**Description:** Trap type (TT) values 060(hex)..07f(hex) are reserved for implementation-dependent exceptions. The existence of implementation\_dependent\_n traps and whether any that do exist are precise, deferred, or disrupting is implementation-dependent.

The priorities of the particular traps are relative and are implementation-dependent, because a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities.

**Implementation:** The following trap types defined by SPARC-V9 are not used in SPARC64.

<b>trap not used in SPARC64</b>
<i>instruction_access_MMU_miss</i>
<i>internal_processor_error</i>
<i>data_access_MMU_miss</i>
<i>LDQF_mem_address_not_aligned</i>
<i>STQD_mem_address_not_aligned</i>
<i>async_data_error</i>

SPARC64 defines the following implementation-dependent trap types.

<b>tt (in Hex)</b>	<b>Trap</b>	<b>priority</b>	<b>type</b>
<i>0x60</i>	prgorammed_emulation_trap	6	precise
<i>0x61</i>	data_breakpoint	14	deferred
<i>0x62</i>	IO_parity	2	precise
<i>0x63</i>	RED_alert	2	disrupting
<i>0x64</i>	CPU_xing	2	disrupting
<i>0x65</i>	Watchdog	1	disrupting
<i>0x66</i>	ECC_trap	2	precise

SPARC64 implements a special accelerated emulation trap for certain LDD and STD instructions.

### **37. Reset trap**

Description: Some of a processor's behavior during a reset trap is implementation-dependent.

Implementation: Power-on Reset (POR) and Watchdog reset (WDR) are implemented by scanning in the reset state on SPARC64.

### **38. Effect of reset trap on implementation-dependent registers**

Description: Implementation-dependent registers may or may not be affected by the various reset traps.

Implementation: None of the implementation-dependent registers are affected by reset traps in SPARC64.

### **39. Entering error\_state on implementation-dependent errors**

Description: The processor may enter error\_state when an implementation-dependent error condition occurs.

Implementation: An internal CPU watchdog time-out occurs after no instruction has been committed for  $2^{*}n$  cycles (n can be scan initialized to one of {12,14,16,18,19,20,21,22,24}, with 24 being the default value). This would take the processor into error state.

### **40. Error\_state processor state**

Description: What occurs after error\_state is entered is implementation-dependent, but it is recommended that as much processor state as possible be preserved upon entry to error\_state.

Implementation: On entry to error state, SPARC64 asserts the output signal CPU\_HALTED. The clock chip in the HAL system stops the clocks to the CPU in response to this signal. A scan out of processor state could be performed at this stage for diagnosis.

### **41. Reserved**

### **42. FLUSH instruction**

Description: If flush is not implemented in hardware, it causes an illegal\_instruction

exception and its function is performed by system software. Whether FLUSH traps is implementation-dependent.

Implementation: SPARC64 takes an `illegal_instruction` trap when a FLUSH instruction is executed.

### 43. Reserved

### 44. Data access FPU trap

Description: If a load floating-point instruction traps with any type of access error exception, the contents of the destination floating-point register(s) either remain unchanged or are undefined.

Implementation: Contents of destination floating-point register(s) remain unchanged.

### 45-46. Reserved

### 47. RDASR

Description: RDASR instructions with `rd` in the range 16..31 are available for implementation-dependent uses (impl. dep #8). For an RDASR instruction with `rs1` in the range 16..31, the following are implementation-dependent: the interpretation of bits 13:0 and 29:25 in the instruction, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

Implementation: See *items 8,9* for details. SPARC64 causes an *illegal\_instruction* trap for reads of the unused ASR values.

### 48. WRASR

Description: WRASR instructions with `rd` in the range 16..31 are available for implementation-dependent uses (impl. dep. #8). For a WRASR instruction with `rd` in the range 16..31, the following are implementation-dependent: the interpretation of bits 18:0 in the instruction, the operation(s) performed (for example, xor) to generate the value written to the ASR, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

Implementation: See *items 8,9* for details. SPARC64 causes an *illegal\_instruction* trap for writes of the unused ASR values.



## 49-54 Reserved

### 55. Floating-point underflow detection

Description: Whether “tininess” (in IEEE 754 terms) is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.

Implementation: SPARC64 detects “tininess” before rounding.

## 56-100. Reserved

### 101. Maximum trap level

Description: It is implementation-dependent how many additional levels, if any, past level 4 are supported.

Implementation: SPARC64 implements 4 levels of traps.

### 102. Clean window trap

Description: An implementation may choose either to implement automatic “cleaning” of register windows in hardware, or generate a `clean_window` trap, when needed, for window(s) to be cleaned by software.

Implementation: SPARC64 generates a `clean_window` trap, when needed, for windows to be cleaned by software.

### 103. Prefetch instructions

Description: The following aspects of the `PREFETCH` and `PREFETCHA` instructions are implementation-dependent: (1) whether they have an observable effect in privileged code; (2) whether they can cause a `data_access_MMU_miss` exception; (3) the attributes of the block of memory prefetched: its size (minimum = 64 bytes) and its alignment (minimum = 64-byte alignment); (4) whether each variant is implemented as a NOP, with its full semantics, or with common-case prefetching semantics; (5) whether and how variants 16..31 are implemented.

Implementation: (1) `PREFETCH` and `PREFETCHA` have identical affects in privileged or non-privileged code.  
(2) Can not cause a `data_access_MMU_miss` exception  
(3) Size and alignments are 128-bytes

(4),(5) See table-1

**Table 1: Prefetch Data**

fcn	V9 Prefetch Function	SPARC64 Function
0	Prefetch for several reads	Prefetch for read
1	Prefetch for one read	Prefetch for read
2	Prefetch for several writes	Prefetch for write
3	Prefetch for one write	Prefetch for write
4	Prefetch page	Prefetch for read
5-15	Reserved	<i>illegal_instruction</i> trap
16-31	Implementation dependent	NOP

#### 104. VER.manuf

**Description:** VER.manuf contains a 16-bit semiconductor manufacturer code. This field is optional, and if not present reads as zero. VER.manuf may indicate the original supplier of a second-sourced chip in cases involving mask-level second-sourcing. It is intended that the contents of VER.manuf track the JEDEC semiconductor manufacturer code as closely as possible. If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a VER.manuf value.

**Implementation:** SPARC64 uses a code of 4 for this field. This is Fujitsu's JEDEC code.

#### 105. TICK register

**Description:** The difference between the values read from the TICK register on two reads should reflect the number of processor cycles executed between the reads. If an accurate count cannot always be returned, an inaccuracy should be small, bounded, and documented. An implementation may implement fewer than 63 bits in TICK.counter; however, the counter as implemented must be able to count for at least 10 years without overflowing. Any upper bits not implemented must be read as zero.

**Implementation:** SPARC64 implements all the bits of TICK register and returns accurate count of the processor cycles, in response to reads from TICK register.

## 106. IMPDEPn instructions

**Description:** The IMPDEP1 and IMPDEP2 instructions are completely implementation-dependent. Implementation-dependent aspects include their operation, the interpretation of bits 29:25 and 18:0 in their encoding, and which (if any) exceptions they may cause.

**Implementation:** SPARC64 uses IMPDEP2 to encode the HAL specific Floating Point Multiply-Add/Subtract instructions. IMPDEP1 is not used and will cause an `illegal_instruction` trap if such an opcode is encountered. Please refer to *SPARC64 Processor User Guide* for more details.

## 107. Unimplemented LDD trap

**Description:** It is implementation-dependent whether LDD and LDDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_LDD` trap.

**Implementation:** SPARC64 does not implement LDD and LDDA is hardware. It uses the `unimplemented_LDD` trap. However in a special mode, there is partial support in hardware for these instructions. Please refer to *SPARC64 Processor User Guide* for more details.

## 108. Unimplemented STD trap

**Description:** It is implementation-dependent whether STD and STDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_STD` trap.

**Implementation:** SPARC64 does not implement STD and STDA is hardware. It uses the `unimplemented_STD` trap. However in a special mode, there is partial support in hardware for these instructions. Please refer to *SPARC64 Processor User Guide* for more details.

## 109. LDDF\_mem\_address\_not\_aligned

**Description:** LDDF and LDDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an `LDDF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the LDDF (or LDDFA) instruction and return.

**Implementation:** SPARC64 causes `LDDF_mem_address_not_aligned` trap for both word and double-word misaligned addresses.

### 110. STDF\_mem\_address\_not\_aligned

Description: STDF and STDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an `STDF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the STDF (or STDFA) instruction and return.

Implementation: SPARC64 causes *STDF\_mem\_address\_not\_aligned* trap for both word and double-word misaligned addresses.

### 111. LDQF\_mem\_address\_not\_aligned

Description: LDQF and LDQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an `LDQF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the LDQF (or LDQFA) instruction and return.

Implementation: SPARC64 generates `fp_exception_other` trap for LDQF, LDQFA instructions and kernel provides emulation routines to complete the load. It does not generate *LDQF\_mem\_address\_not\_aligned* trap.

### 112. STQF\_mem\_address\_not\_aligned

Description: STQF and STQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an `STQF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the STQF (or STQFA) instruction and return.

Implementation: SPARC64 generates `fp_exception_other` trap for STQF, STQFA instructions and kernel provides emulation routines to complete the load. It does not generate *STQF\_mem\_address\_not\_aligned* trap.

### 113. Implemented memory models

Description: Whether the Partial Store Order (PSO) or Relaxed Memory Order (RMO) models are supported is implementation-dependent.

Implementation: SPARC64 supports *Load/Store ordering (LSO)* and *Store ordering (STO)*. *Partial Store Order (PSO)* is implemented using *LSO* and *Relaxed Memory Order (RMO)* is implemented using *STO*.

### 114. RED\_state trap vector address (RSTVaddr)

Description: The `RED_state` trap vector is located at an implementation-dependent

address referred to as RSTVaddr.

Implementation: SPARC64 has a scan only register that holds RSTVaddr.

### **115. RED\_state processor state**

Description: What occurs after the processor enters RED\_state is implementation-dependent.

Implementation: SPARC64 has the following behavior in RED\_state.

- 1) The output signal RED\_MODE is asserted indicating CPU is in RED\_state.
- 2) The CPU executes in sequential mode.
- 3) On entry into and exit from RED\_state, the CPU invalidates the on-chip instruction cache and prefetch buffers.
- 4) Off chip data and instruction caches are disabled.
- 5) The MMU uses a special translation mechanism.
- 6) All I/O accesses are disabled.
- 7) Further red state errors are ignored.
- 8) XIR, and Chip Crossing Errors are not masked and could cause a trap.

### **116. SIR\_enable control flag**

Description: The location of and the means of accessing the SIR\_enable control flag are implementation-dependent. In some implementations, it may be permanently zero.

Implementation: SIR\_enable control flag is permanently zero in SPARC64.

### **117. MMU disabled prefetch behavior**

Description: Whether Prefetch and Non-faulting Load always succeed when the MMU is disabled is implementation-dependent.

Implementation: In SPARC64, Prefetch and Non-faulting Loads will have undefined behavior if the MMU is disabled.

## 118. Identifying I/O locations

Description: The manner in which I/O locations are identified is implementation-dependent.

Implementation: Please contact HAL Computer Systems for details of I/O operation.

## 119. Unimplemented values for PSTATE.MM

Description: The effect of writing an unimplemented memory-mode designation into PSTATE.MM is implementation-dependent

Implementation: SPARC64 only the most significant bit of MM is used to determine the memory model; the least significant bit is ignored. However, the system software should not use the encoding '11' since it is reserved for future SPARC-V9 extensions.

## 120. Coherence and atomicity of memory operations

Description: The coherence and atomicity of memory operations between processors and I/O DMA memory accesses are implementation-dependent.

Implementation: In SPARC64, coherence and atomicity of memory operations between processors and I/O DMA memory accesses are variable and depend on the I/O device. Please contact HAL Computer Systems for details.

## 121. Implementation-dependent memory model

Description: An implementation may choose to identify certain addresses and use an implementation dependent memory model for references to them.

Implementation: In SPARC64, certain addresses use implementation dependent memory models for references to them. Please contact HAL Computer Systems for details.

## 122. FLUSH latency

Description: Latency between the execution of FLUSH on one processor and the point at which the modified instructions have replaced out-dated instructions in a multiprocessor is implementation-dependent.

Implementation: Not applicable since, SPARC64 does not support a multi-processor configuration.

### 123. Input/output (I/O) semantics

Description: The semantic effect of accessing input/output (I/O) registers is implementation-dependent.

Implementation: In SPARC64, the semantic effect of accessing input/output (I/O) registers is undefined.

### 124. Implicit ASI when TL>0

Description: When  $TL > 0$ , the implicit ASI for instruction fetches, loads, and stores is implementation-dependent. See SPARC-V9 Architecture Manual section F.4.4, “Contexts,” for more information.

Implementation: SPARC64 uses *ASI\_PRIMARY* or *ASI\_PRIMARY\_LITTLE* for instruction fetches, loads and stores when  $TL > 0$

### 125. Address masking

Description: When  $PSTATE.AM = 1$ , the value of the high-order 32-bits of the PC transmitted to the specified destination registers(s) by CALL, JMPL, RDPC, and on a trap is implementation-dependent.

Implementation: When  $PSTATE.AM$  bit is set on SPARC64, a full 64-bit address is transmitted to the specified destination registers by CALL, JMPL, RDPC and traps transmit all 64-bits to  $TPC[n]$  and  $TNPC[n]$ .

### 126. TSTATE bits 19:18

Description: If  $PSTATE$  bit 11 (10) is implemented,  $TSTATE$  bit 19 (18) shall be implemented and contain the state of  $PSTATE$  bit 11 (10) from the previous trap level. If  $PSTATE$  bit 11 (10) is not implemented,  $TSTATE$  bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from  $PSTATE$ , or to zeroes.

Implementation: SPARC64 does not implement  $PSTATE$  bits 10 & 11 and they are read as zeroes.  $TSTATE$  bits 19 and 18 are read as zeroes.

### 127. PSTATE bits 11:10

Description: The presence and semantics of  $PSTATE.PID1$  and  $PSTATE.PID0$  are implementation-dependent. The presence of  $TSTATE$  bits 19 and 18 is implementation-dependent. If  $PSTATE$  bit 11 (10) is implemented,  $TSTATE$  bit 19 (18) shall be implemented and contain the state of  $PSTATE$  bit 11 (10) from the previous trap level. If  $PSTATE$  bit 11 (10) is not implemented,  $TSTATE$  bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously

read from PSTATE, or to zeroes.

Implementation: SPARC64 does not implement PSTATE bits 10 & 11 and they are read as zeroes. TSTATE bits 19 and 18 are read as zeroes.

## 128. CLEANWIN register update

Earlier implementations of SPARC chips implemented the SPARC-V9 specification for RESTORED using the following equation to update CLEANWIN register:

if (CLEANWIN != NWINDOWS) CLEANWIN++;

Subsequently V9 definition changed to modify the equation as:

if (CLEANWIN < NWINDOWS-1) CLEANWIN++;

SPARC64 implements the *RESTORED* using the earlier definition. The SPARC64 Kernel will ensure that *CLEANWIN* does not have a value beyond *NWINDOWS-1*.



**Chapter 2: SUN Implementation of V9 Architecture**

---

---

**UltraSPARC - I**

---

---

**V9**

**SPARC INTERNATIONAL**



## CHAPTER 2: SUN ULTRASPARC

### 0. Introduction

This document describes the implementation-dependencies of Sun's STP 1030BGA-UltraSPARC-1 processor as put forth in "The SPARC Architecture Manual - Version 9" by SPARC International. The items listed below correspond to the implementation dependencies as listed in the text and by number in Appendix C of the manual along with the description of the implementation dependency from the manual. The "Implementation" section for each item describes the implementation on the UltraSPARC-I processor.

### 1. Software emulation of instructions

**Description:** whether an instruction is implemented directly by hardware, simulated by software, or emulated by firmware is implementation-dependent.

**Implementation:** all instructions are implemented in hardware except the following, which must be simulated by software.

POPC	Population count
LDQF	Load quad-precision FP register
LDQFA	Load quad-precision FP register from alternate space
STQF	Store quad-precision FP register
STQFA	Store quad-precision FP register to alternate space
F{s,d}TOq	Convert single-/double- to quad precision FP
F{i,x}TOq	Convert 32-/64-bit integer to quad-precision FP
FqTO{s,d}	Convert quad- to single-/double-precision FP
FqTO{i,x}	Convert quad-precision FP to 32-/64-bit integer
FADDq	Quad-precision FP add
FSUBq	Quad-precision FP subtraction
FCMP{E}q	Quad-precision FP compares
FMOVqcc	Move quad-precision FP register on condition
FMOVqr	Move quad-precision FP register on integer register condition
FMOVq	Move quad-precision FP register
FABSq	Quad-precision FP absolute value
FNEGq	Quad-precision FP negate
FdMULq	Double- to quad-precision FP multiply
FNULq	Quad-precision FP multiply

FDIV	Quad-precision FP divide
FSQRTq	Quad-precision FP divide

## 2. Number of IU registers

**Description:** an implementation of the IU may contain from 64 to 258 general purpose 64 bit registers. This corresponds to a grouping of the registers into two sets of eight global r registers, plus a circular stack of from three to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (NWINDOVS) is implementation-dependent, the total number of registers is also implementation-dependent.

**Implementation:** UltraSPARC-I implements eight register windows plus four sets of eight global r registers, for a total of 160 64-bit r registers.

## 3. Incorrect IEEE Std 754-1985 results

**Description:** an implementation may indicate that a floating-point instruction did not produce a correct ANSI/IEEE Standard 754-1985 result by generating a special floating-point unfinished or unimplemented exception. In this case, privileged mode software shall emulate any functionality not present in the hardware.

**Implementation:** the quad-precision floating-point instructions listed in implementation dependency #1 above all generate floating-point unimplemented exceptions.

UltraSPARC-I generates floating-point unimplemented exceptions for the following cases of subnormal operands or results.

Subnormal Operand Unimplemented Exception Cases:

$F\{s,d\}TO\{i,x\}$	one subnormal operand
$F\{s,d\}TO\{i,x\}$	one subnormal operand
$FSQRT\{s,d\}$	one subnormal operand
$FADD\{s,d\}$	one or two subnormal operand
$FMUL\{s,d\}$	-25 <Er <255 (SP) one subnormal operand -54 <Er <2047 (DP) one subnormal operand two subnormal operands

FDIV{s,d}	-25 <Er <255 (SP) one subnormal operand -54 <Er <2047 (DP) one subnormal operand two subnormal operands
-----------	---

Subnormal Result Unimplemented Exception Cases:

FdTOs	-25 <Er < 1 (SP) -54 <Er < 1 (DP)
FADD{s,d}	-25 <Er < 1 (SP) -54 <Er < 1 (DP)
FMUL{s,d}	-25 <Er < 1 (SP) -54 <Er < 1 (DP)
FDIV{s,d}	-25 <Er < 1 (SP) -54 <Er < 1 (DP)

Prediction of overflow, underflow and inexact traps for divide and square roots is used. For divide, pessimistic prediction occurs when underflow/overflow cannot be determined from examining the source operand exponents. For divide and square root, pessimistic prediction of inexact occurs unless one of the operands is a zero, NSN or infinity. When pessimistic prediction occurs and the exception is enabled, a floating-point unfinished exception is generated.

## 4 - 5. Reserved

## 6. I/O registers privileged status

Description: whether I/O registers can be accessed by non-privileged code is implementation-dependent.

Implementation: For systems using UltraSPARC-I, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation.

## 7. I/O register definitions

Descriptions: the contents and addresses of I/O registers are implementation-dependent

Implementation: For systems using UltraSPARC-I, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation.

## 8. RDASR/WRASR target registers

Description: software can use read/write ancillary state register instructions to read/write implementation-dependent processor registers (ASRs 16-31).

Implementation: UltraSPARC-I implements the following implementation-dependent ASRs.

<b>rd</b>	<b>name</b>	<b>access</b>
<i>16</i>	PERFA_CONTROL_REG	<i>RW</i>
<i>17</i>	PERF_COUNTER	<i>RW</i>
<i>18</i>	DISPATCH_CONTROL_REG	<i>RW</i>
<i>19</i>	GRAPHICS_STATUS_REG	<i>RW</i>
<i>20</i>	SET_SOFTINT	<i>W</i>
<i>21</i>	CLEAR_SOFTINT	<i>W</i>
<i>22</i>	SOFTINT_REG	<i>RW</i>
<i>23</i>	TICK_CMPR_REG	<i>RW</i>

## 9. RDASR/WRASR privileged status

Description: whether each of the implementation-dependent read/write ancillary state register instructions (for ASRs 16-31) is privileged is implementation dependent.

Implementation: The privileged status of UltraSPARC-I's implementation-dependent registers is as follows:

<b>rd</b>	<b>name</b>	<b>access</b>
16	PERFA_CONTROL_REG	<i>PRIVILEGED</i>
17	PERF_COUNTER	<i>PRIVILEGED*</i>
18	DISPATCH_CONTROL_REG	<i>PRIVILEGED</i>
19	GRAPHICS_STATUS_REG	<i>NONPRIVILEGED</i>
20	SET_SOFTINT	<i>PRIVILEGED</i>
21	CLEAR_SOFTINT	<i>PRIVILEGED</i>
22	SOFTINT_REG	<i>PRIVILEGED</i>
23	TICK_CMPR_REG	<i>PRIVILEGED</i>

\* If PERF\_CONTROL\_REG. PRIV =1)

## 10 - 12. Reserved

## 13. VER.impl

Description: VER.impl uniquely identifies an implementation or class of software-compatible implementations of the architecture. Values FFF0 (hex)..FFFF(hex) are reserved and are not available for assignment.

Implementation: UltraSPARC-I uses the implementation code 0010 (hex).

## 14 - 15. Reserved

## 16. IU deferred-trap queue

Description: the existence, contents, and operation of an IU deferred-trap queue are implementation-dependent; it is not visible to user application programs under normal operating conditions.

Implementation: UltraSPARC-I does not implement a deferred-trap queue.

## 17. Reserved

## 18. Nonstandard IEEE 754-1985 results

Description: bit 22 of the FSR, RSR\_nonstandard\_fp (NS), when set to 1, causes the FPU to produce implementation-defined results that may not correspond to IEEE Standard 754-1985.

Implementation: if FSR.NS is set to one, the subnormal operand and results cases identified for implementation dependency #3 above, are flushed to zero.

## 19. FPU version, FSR.ver

Description: bits 19:17 of the FSR, FSR.ver, identify one or more implementations of the FPU architecture.

Implementation: on UltraSPARC-I the FSR.VER field is set to zero.

## 20 - 21. Reserved

## 22. FPU TEM, cexc, and aexc

Description: an implementation may choose to implement the TEM, cexc, and aexc fields in hardware in either of two ways (see section 5.1.7.11 of SPARC-V9 Architecture Manual for details).

Implementation: UltraSPARC-I implements the TEM, cexc and aexc fields in conformance to IEEE Std 754-1985.

## 23. Floating-point traps

Description: floating point traps may be precise or deferred. If deferred, a floating point deferred-trap queue (FQ) must be present.

Implementation: UltraSPARC-I floating-point traps are precise and it does not implement



an FQ.

## 24. FPU deferred-trap queue (FQ)

**Description:** the presence, contents of, and operations on the floating-point deferred-trap queue (FQ) are implementation-dependent.

**Implementation:** UltraSPARC-I does not implement an FQ.

## 25. RDPR of FQ with nonexistent FQ

**Description:** on implementations without a floating-point queue, an attempt to read the FQ with an RDPR instruction shall cause either an `illegal_instruction` exception or an `fp_exception_other` exception with `FSR.Ftt` set to 4 (`sequence_error`).

**Implementation:** attempting to read the FQ with a RDPR instruction causes an `illegal_instruction` exception.

## 26 - 28. Reserved

## 29 Address space identifier (ASI) definitions

**Description:** the following ASI assignments are implementation-dependent: restricted ASIs (all values hex) 00..03.05..0B. 0D..0F, 12..17, and 1A..7F; and unrestricted ASIs C0..FF..

**Implementation:** UltraSPARC-I assigns the following implementation-dependent ASI values.

restricted ASI values (all values hex):

14, 15, 1C, 1D, 24, 2C, 45, 46, 47, 48, 49, 4A, 4B, 4C, 4D, 4E, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 5A, 5B, 5C, 5D, 5E, 5F, 66, 67, 6E, 6F, 70, 71, 76, 77, 78, 79, 7E, 7F

restricted ASI values (all values hex):

C0, C1, C2, C3, C4, C5, C8, C9, CA, CB, CC, CD, D0, D1, D2, D3, D8, D9, DA, DB, E0, E1, F0, F1, F8, F9

### 30. ASI address decoding

**Description:** an implementation may choose to decode only a subset of the 8-bit ASI specifier; however, it shall decode at least enough of the ASI to distinguish ASI\_PRIMARY, ASI\_PRIMARY\_LITTLE, ASI\_AS\_IF\_USER\_PRIMARY, ASI\_AS\_IF\_USER\_PRIMARY\_LITTLE, ASI\_PRIMARY\_NOFAULT, ASI\_PRIMARY\_NOFAULT\_LITTLE, ASI\_SECONDARY, ASI\_SECONDARY\_LITTLE, ASI\_AS\_IF\_USER\_SECONDARY, ASI\_AS\_IF\_USER\_SECONDARY\_LITTLE, ASI\_SECONDARY\_NOFAULT\_LITTLE. If AFI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE are supported (impl. dep. #124), they must be decoded also. Finally, an implementation must always decode ASI bit<7> while PSTATE.PRIV = 0, S0 so that an attempt by nonprivileged software to access a restricted ASI will always cause a privileged\_action exception.

**Implementation:** UltraSPARC-I decodes the entire 8-bit ASI specifier.

### 31. Catastrophic error exceptions

**Description:** the causes and effects of catastrophic error exceptions are implementation-dependent. They may cause precise, deferred or disrupting traps.

**Implementation:** UltraSPARC-I catastrophic error exceptions cause deferred traps. The PSTATE.RED bit is not automatically set in hardware for any catastrophic error exceptions other than when trapping to MAXTL-1.

### 32. Deferred traps

**Description:** whether any deferred traps (and associated deferred-trap queues) are present is implementation-dependent.

**Implementation:** UltraSPARC-I may encounter deferred traps during memory accesses. Such errors lead to termination of the currently executing process or result in a system reset if system state has been corrupted. Error logging information allows software to determine if the system state has been corrupted.

### 33. Trap precision

**Description:** exceptions that occur as the result of program execution may be precise or deferred, although it is recommended that such exceptions be precise. Examples include `mem_address_not_aligned` and `division_by_zero`.

**Implementation:** all of the exceptions listed in the SPARC-V9 Architecture Manual section 7.3.5, item (2) are precise with the exception of `instruction_access_error`, which is deferred.

### 34. Interrupt clearing

**Description:** how quickly a processor responds to an interrupt request and the method by which an interrupt request is removed are implementation-dependent.

**Implementation:** The response time to interrupt is dependent the activity the processor is executing at the time the interrupt is received (e.g., whether executing a trap handler with `PSTATE.IE=0`, etc.). The interrupt request is removed by clearing a bit in the implementation-dependent interrupt vector receive register.

### 35. Implementation-dependent traps

**Description:** trap type (TT) values `060(hex)..07f(hex)` are reserved for implementation-dependent exceptions. The existence of `implementation_dependent_n` traps and whether any that do exist are precise, deferred, or disrupting is implementation-dependent.

**Implementation:** the following implementation-dependent trap types are implemented on UltraSPARC-I.

TT (hex)	Exception	Category
<i>060</i>	<code>interrupt_vector</code>	<i>disrupting</i>
<i>061</i>	<code>PA_watchpoint</code>	<i>disrupting</i>
<i>062</i>	<code>VA_watchpoint</code>	<i>disrupting</i>

TT (hex)	Exception	Category
063	corrected_ECC_error	<i>disrupting</i>
064..067	fast_instruction_access_MMU_miss	<i>precise</i>
068..06B	fast_data_access_MMU_miss	<i>precise</i>
06C..06F	fast_data_access_protection	<i>precise</i>

### 36. Trap priorities

**Description:** the priorities of the particular traps are relative and are implementation-dependent, because a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities.

**Implementation:** UltraSPARC-I traps are prioritized relative to each other according to the relative priorities in the SPARC-V9 Manual.

### 37. Reset trap

**Description:** some of a processor's behavior during a reset trap is implementation-dependent.

**Implementation:** UltraSPARC-I conforms to the required behavior during a reset trap. Unspecified behavior is either defined during reset or specified as requiring initialization.

### 38. Effect of reset trap on implementation-dependent registers

**Description:** implementation-dependent registers may or may not be affected by the various reset traps.

**Implementation:** Implementation-dependent registers on UltraSPARC-I either have defined behavior during reset traps or are specified as requiring initialization.

### 39. Entering error\_state on implementation-dependent errors

Description: the processor may enter `error_state` when an implementation-dependent error condition occurs.

Implementation: UltraSPARC-I enters `error_state` only by trapping when `TL = MAXTL`. Any type of trap may cause this.

#### **40. Error\_state processor state**

Description: what occurs after `error_state` is entered is implementation-dependent, but it is recommended that as much processor state as possible be preserved upon entry to `error_state`.

Implementation: Entering `error_state` causes UltraSPARC-I to trigger a `watchdog_reset` trap. As much state as possible is preserved during this action.

#### **41. Reserved**

#### **42. FLUSH instruction**

Description: if `flush` is not implemented in hardware, it causes an `illegal_instruction` exception and its function is performed by system software. Whether `FLUSH` traps is implementation-dependent.

Implementation: UltraSPARC-I implements `FLUSH` in hardware and it can cause a `data_access_exception` if the page is mapped with side effects or no-fault-only bits set, virtual address out of range, privilege violation, or a `data_access_MMU_miss` trap.

#### **43. Reserved**

#### **44. Data access FPU trap**

Description: if a load floating-point instruction traps with any type of access error exception, the contents of the destination floating-point register(s) either remain unchanged or are undefined.

Implementation: access error exceptions on floating-point load instructions leave the destination floating-point register contents unchanged.

## 45-46. Reserved

## 47. RDASR

**Description:** RDASR instructions with rd in the range 16..31 are available for implementation-dependent uses (impl. dep #8 ). For an RDASR instruction with rs1 in the range 16..31, the following are implementation-dependent: the interpretation of bits 13:0 and 29:25 in the instruction, whether the instruction is privileged (impl. dep. #9), and whether the instruction is privileged (impl. dep. #9), and whether it causes and illegal\_instruction trap.

**Implementation:** the bit fields specified above are not used for UltraSPARC-I implementation-dependent RDASR instructions. Reads of unused rs1 values and reads of write-only implementation-dependent ASRs cause illegal\_instruction traps

## 48. WRASR

**Description:** WRASR instructions with rd in the range 16..31 are available for implementation-dependent uses (impl.dep.#8). For a WRASR instruction with rd in the range 16..31, the following are implementation-dependent: the interpretation of bits 18:0 in the instruction, the operation(s) performed (for example, xor) to generate the value written to the ASR, whether the instruction is privileged (impl. dep.#9), and whether it causes an illegal\_instruction trap.

**Implementation:** UltraSPARC-I does not interpret bits 18:0 of the WRASR instruction. Using WRASR to the SET\_SOFTINT and CLEAR\_SOFTINT ASRs will set and clear (respectively) bits in the SOFTINT\_REG ASR. Writes of the unused ASR values cause illegal\_instruction traps.

## 49-54. Reserved

## 55. Floating-point underflow detection

**Description:** whether “tininess” (in IEEE 754 terms) is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.

**Implementation:** UltraSPARC-I detects underflow before rounding.

## 56 - 100. Reserved

### 101. Maximum trap level

Description: it is implementation-dependent how many additional levels, if any, past level 4 are supported.

Implementation: UltraSPARC-I implements 5 trap levels.

### 102. Clean window trap

Description: an implementation may choose either to implement automatic “cleaning” of register windows in hardware, or generate a clean\_window trap, when needed, for window(s) to be cleaned by software.

Implementation: UltraSPARC-I cleans register windows by generating a clean\_window trap for windows to be cleaned by software.

### 103. Prefetch instructions

Description: the following aspects of the PREFETCH and PREFETCHA instructions are implementation-dependent: (1) whether they have an observable effect in privileged code; (2) whether they can cause a data\_access\_MMU\_miss exception; (3) the attributes of the block of memory prefetched: its size (minimum = 64 bytes) and its alignment (minimum = 64 byte alignment); (4) whether each variant is implemented as NOP, with its full semantics, or with common-case prefetching semantics; (5) whether and how variants 16..31 are implemented.

Implementation: on UltraSPARC-I, PREFETCH and PREFETCHA have the same observable effect as a NOP in both privileged and nonprivileged modes.

### 104. VER.manuf

Description: VER.manuf contains a 16-bit semiconductor manufacturer code. This field is optional, and if not present reads a zero. VER.manuf may indicate the original supplier of a second-sourced chip in cases involving mask-level

second-sourcing. It is intended that the contents of VER.manuf track the JEDEC semiconductor manufacturer code as closely as possible. If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a VER.manuf value.

Implementation: UltraSPARC-I uses the manufacturer code 0017(hex)

## 105. TICK register

Description: the difference between the values read from the TICK register on two reads should reflect the number of processor cycles executed between the reads. If an accurate count cannot always be returned, an inaccuracy should be small, bounded, and documented. An implementation may implement fewer than 63 bits in TICK.counter; however, the counter as implemented must be able to count for at least 10 years without overflowing. Any upper bits not implemented must be read as zero.

Implementation: UltraSPARC-I implements 63 bits of TICK.counter and reflects the number of processor clocks between reads.

## 106. IMPDEP1 instructions

Description: the IMPDEP1 and IMPDEP2 instructions are completely implementation-dependent. Implementation-dependent aspects include their operation, the interpretation of bits 29:25 and 18:0 in their encodings, and which (if any) exceptions they may cause.

Implementation: UltraSPARC-I implements implementation-dependent instructions using the following field values:

op	op3	opf
10	110110	010000000
10	110110	001010000
10	110110	001010001
10	110110	001010010
10	110110	001010011
10	110110	001010100
10	110110	001010101
10	110110	001010110



---

10	110110	001010111
10	110110	000111011
10	110110	000111010
10	110110	000111101
10	110110	001001101
10	110110	001001011
10	110110	000110001
10	110110	000110011
10	110110	000110101
10	110110	000110110
10	110110	000110111
10	110110	000111000
10	110110	000111001
10	110110	000011000
10	110110	000011010
10	110110	001001000
10	110110	001100000
10	110110	001100001
10	110110	001111110
10	110110	001111111
10	110110	001110100
10	110110	001110101
10	110110	001111000
10	110110	001111001
10	110110	001101010
10	110110	001101011
10	110110	001100110
10	110110	001100111
10	110110	001111100
10	110110	001111101
10	110110	001100010
10	110110	001100011
10	110110	001110000
10	110110	001110001
10	110110	001101110
10	110110	001101111
10	110110	001101100
10	110110	001101101
10	110110	001110010
10	110110	001111010
10	110110	001111011

---

10	110110	001110110
10	110110	001110110
10	110110	001110111
10	110110	001101000
10	110110	001101001
10	110110	001100100
10	110110	001100101
10	110110	000101000
10	110110	000101100
10	110110	000100000
10	110110	000100100
10	110110	000100010
10	110110	000100110
10	110110	000101010
10	110110	000101110
10	110110	000000000
10	110110	000000010
10	110110	000000100
10	110110	000000110
10	110110	000001000
10	110110	000001010
10	110110	000111110
10	110110	000010000
10	110110	000010010
10	110110	000010100

### 107. Unimplemented LDD trap

**Description:** it is implementation-dependent whether LDD and LDDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_LDD` trap.

**Implementation:** UltraSPARC-I implements LDD and LDDA in hardware.

### 108. Unimplemented STD trap

**Description:** it is implementation-dependent whether STD and STDA are implemented in hardware. If not, an attempt to execute either will cause an

unimplemented\_STD trap.

Implementation: UltraSPARC-I implements STD and STDA in hardware.

### **109. LDDF\_mem\_address\_not\_aligned**

Description: LDDF and LDDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an LDDF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the LDDF (or LDDFA) instruction and return.

Implementation: UltraSPARC-I generates an LDDF\_mem\_address\_not\_aligned exception if an LDDF or LDDFA effective address is word-aligned but not doubleword-aligned.

### **110. STDF\_mem\_address\_not\_aligned**

Description: STDF and STDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an STDF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STDF (or STDFA) instruction and return.

Implementation: UltraSPARC-I generates an STDF\_mem\_address\_not\_aligned exception if an STDF or STDFA effective address is word-aligned but not doubleword-aligned.

### **111. LDQF\_mem\_address\_not\_aligned**

Description: LDQF and LDQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an LDQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the LDQF (or LDQFA) instruction and return.

Implementation: UltraSPARC-I does not implement the LDQF and LDQFA in hardware, they must be emulated in software using other instructions.

### **112. STQF\_mem\_address\_not\_aligned**

Description: STQF and STQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an STQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STQF (or STQFA) instruction and return.

### 113. Implemented memory models

Description: whether the Partial Store Order (PSO) or Relaxed Memory Order (RMO) models are supported is implementation-dependent.

Implementation: UltraSPARC-I supports the Partial Store Order and Relaxed Memory Order models.

### 114. RED\_state trap vector address (RSTVaddr)

Description: the RED\_state trap vector is located at an implementation-dependent address referred to as RSTVaddr.

Implementation: RSTVaddr = 1fff0000000 (hex)

### 115. RED\_state processor state

Description: what occurs after the processor enters RED\_state is implementation-dependent.

Implementation: On UltraSPARC-I some register contents are forced to specified values and some hardware functions are disabled upon entering RED\_state to avoid as much as possible any additional traps which would cause the processor to enter error\_state.

### 116. SIR\_enable control flag

Description: the location of and the means of accessing the SIR\_enable control flag are implementation-dependent. In some implementations, it may be permanently zero.

Implementation: the SIR\_enable in UltraSPARC-I is permanently zero.

## 117. MMU disabled prefetch behavior

Description: whether Prefetch and Non-faulting Load always succeed when the MMU is disabled is implementation-dependent.

Implementation: prefetch instructions behave as NOP instructions. Non-faulting Load instructions may or may not succeed when the MMU is disabled depending on the state of an implementation-dependent register determining whether the cache is enabled.

## 118. Identifying I/O locations

Description: the manner in which I/O locations are identified is implementation- dependent.

Implementation: For systems using UltraSPARC-I, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation.

## 119. Unimplemented values for PSTATE.MM

Description: the effect of writing an unimplemented memory-mode designation into PSTATE.MM is implementation-dependent.

Implementation: UltraSPARC-I implements all three memory modes specified in the SPARC Architecture Manual Version 9. If the reserved PSTATE.MM value (3) were written, UltraSPARC-I would interpret it as RMO.

## 120. Coherence and atomicity of memory operations

Description: the coherence and atomicity of memory operations between processors and I/O DMA memory accesses are implementation-dependent.

Implementation: This is dependent on the system implementation rather than the processor

implementation for systems that use UltraSPARC-I.

### **121. Implementation-dependent memory model**

**Description:** an implementation may choose to identify certain addresses and use an implementation-dependent memory model for references to them.

**Implementation:** UltraSPARC-I does not use any implementation-dependent memory models.

### **122. FLUSH latency**

**Description:** latency between the execution of FLUSH on one processor and the point at which the modified instructions have replaced out-dated instructions in a multiprocessor is implementation-dependent.

**Implementation:** This is dependent on the system implementation rather than the processor implementation for systems that use UltraSPARC-I.

### **123. Input/output (I/O) semantics**

**Description:** the semantic effect of accessing input/output (I/O) registers is implementation-dependent.

**Implementation:** For systems using UltraSPARC-I, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation.

### **124. Implicit ASI when TL>0**

**Description:** when TL>0, the implicit ASI for instruction fetches, loads, and stores is implementation-dependent. See SPARC-V9 Architecture Manual section F.4.4, "Contexts," for more information.

**Implementation:** the implicit ASI for instruction fetches, loads, and stores when TL>0 is ASI\_PRIMARY.

## 125. Address masking

**Description:** when PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination register(s) by CALL, JMPL, RDPC, and on a trap is implementation-dependent.

**Implementation:** when PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination register(s) by CALL, IMPL, RDPC, and on a trap is zero.

## 126. TSTATE bits 19:18

**Description:** If PSTATE bit 11 (10) is implemented, TSTATE bit 9 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeros.

**Implementation:** UltraSPARC-I implements TSTATE bits 19:18 to hold the state of PSTATE bits 11:10 for each previous trap level.

## 127. PSTATE bits 11:10

**Description:** The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are implementation-dependent. The presence of TSTATE bits 19 and 18 is implementation-dependent. If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeros.

**Implementation:** PSTATE.PID1 and PSTATE.PID0 are implemented on UltraSPARC-I as selects for two additional sets of eight trap global registers. The corresponding bits in the TSTATE register are implemented to store these bits for the previous trap level.





**Chapter 3: HAL Implementation of V9 Architecture**

---

---

**SPARC 64-II**

---

---

**V9**

**SPARC INTERNATIONAL**



## CHAPTER 3: HAL SPARC64-II

### 0. Introduction

This document describes the implementation details of the *SPARC64-II* processor developed by *HAL Computer Systems*. The items listed below correspond to the implementation dependencies as listed in the text and by number in Appendix C of “*The SPARC Architecture Manual - Version 9*” by *SPARC International*, along with the description of the implementation dependency. The “Implementation” section for each item describes the implementation on the SPARC64 processor.

### 1. Software emulated instructions

#### Description:

Whether an instruction is implemented directly by hardware, simulated by software, or emulated by firmware is implementation-dependent.

#### Implementation:

Sparc64 does not implement the following instructions in hardware:

- *All floating point instructions with quad operands or results*  
These operations will take an *fp\_exception\_other* trap with *FSR.ftt = unimplemented\_FPop*. The kernel will then emulate the quad operation and store the result into a quad-aligned set of floating-point registers as defined by Sparc-V9 manual.
- *fsqrt, fsqrt*  
Executing these instructions will cause a *fp\_exception\_other* exception with *FSR.ftt = unimplemented\_FPop*. In this case kernel emulation routines are provided to complete the instructions.
- *flush*  
This instruction will cause an *illegal\_instruction* trap if executed. Kernel emulation routines will be provided to flush the cache line from the data cache and invalidate any matching cache lines in the instruction cache.
- *ldd, ldda, std, stda*  
Executing these instructions in normal mode would generate *unimplemented\_LDD* and *unimplemented\_STD* trap. Kernel emulation routines will be provided to complete the instructions. Sparc64 also implements a special accelerated emulation trap handling for certain LDD and STD instructions, if a special mode is chosen.
- *popc*  
This instruction will cause an *illegal\_instruction* trap if executed. Kernel emulation routines will be provided to complete the action.

---

## 2. Number of IU registers

### Description:

An implementation of the IU may contain from 64 to 528 general purpose 64 bit r registers. This corresponds to a grouping of the registers into two sets of eight global r registers, plus a circular stack of from 3 to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (NWINDOWS) is implementation-dependent, the total number of registers is also implementation-dependent.

### Implementation:

Sparc64 implements 5 16-register sets (windows) in hardware. Thus there are a total of 96 integer registers visible to software. They are:

- 8 global registers
- 8 alternate global registers
- 5 windows of 16 registers each (=80 registers)

## 3. Incorrect IEEE Std 754-1985 results

### Description:

An implementation may indicate that a floating-point instruction did not produce a correct ANSI/IEEE Standard 754-1985 result by generating a special floating-point unfinished or unimplemented exception. In this case, privileged mode software shall emulate any functionality not present in the hardware.

### Implementation:

Sparc64 in conjunction with the kernel emulation code produces the correct IEEE 754 results required in this section.

- Traps Inhibit Results  
Sparc64 in conjunction with the kernel emulation code produces results required.
- Trapped Underflow Definition (UFM=1)  
Sparc64 detects “tininess” before rounding as recommended.
- Untrapped Underflow Definition (UFM=0)  
Sparc64 meets these requirements with some help from the kernel divide fixup code.
- Floating-Point Nonstandard Mode  
Sparc64 FPU is “standard”, and therefore does not support a nonstandard mode.

## 4-5. Reserved

## 6. I/O registers privileged status

### Description:

Whether I/O registers can be accessed by non privileged code is implementation-dependent.

**Implementation:**

In Sparc64 some I/O registers can be accessed by non privileged code.

**7. I/O register definitions****Description:**

The contents and addresses of I/O registers are implementation-dependent.

**Implementation:**

Please contact HaL for details of I/O registers.

**8,9. RDASR/WRASR target registers and privileged status****Description:**

Software can use read/write ancillary state register instructions to read/write implementation-dependent processor registers (ASRs 16-31).

Whether each of the implementation-dependent read/write ancillary state register instructions (for ASRs 16-31) is privileged is implementation dependent.

**Implementation:**

Sparc64 implements 9 implementation-dependent ASR registers.

- PIO Address Match Register (ASR23)  
This privileged read/write register is used to specify a range of addresses which force program ordering for all LD and ST instructions which are within this range.
- LDD Trap Base Address (ASR24)  
This privileged read/write register specifies a special trap base address for some *unimplemented\_LDD* and *unimplemented\_STD* traps.
- Instruction Emulation Register (ASR25)  
This read only register is written by CPU on a trap for a LDD/STD that uses the LDD Trap Base Address described above.
- Data Breakpoint Register (ASR26)  
This privileged write-only register is used to trap any data accesses to a double word aligned breakpoint address.
- Software Initiated Reset (ASR27)  
A write to this register with a WRASR instruction will cause a software initiated reset (SIR). An SIR is a precise trap. ASR27 is privileged and write-only.
- Fault Address Register (ASR28) and Fault Access Type (ASR29)  
These registers facilitate the handling of traps that involve a data memory access. The registers are privileged and read-only. System software must take care to read these registers on entry to a fault handler before any other fault can occur that would overwrite them.
- Performance Monitor Register (ASR30)  
This privilege read/write register is used to evaluate processor performance.
- State Control Register (ASR31)

ASR31 is a 16bit implementation specific register that contains a set of flags for controlling the state of the CPU, MMU and Caches. The register is privileged and can be read/written.

## **10-12 Reserved**

## **13. VER.impl**

### **Description:**

VER.impl uniquely identifies an implementation or class of software-compatible implementations of the architecture. Values FFF0(hex)..FFFF(hex) are reserved and are not available for assignment.

### **Implementation:**

Sparc64 uses a version number of 2.

## **14-15 Reserved**

## **16. IU deferred-trap queue**

### **Description:**

The existence, contents, and operation of an IU deferred-trap queue are implementation-dependent; it is not visible to user application programs under normal operating conditions

### **Implementation:**

Sparc64 does not need and therefore does not implement an IU deferred-trap queue.

## **17. Reserved**

## **18. Nonstandard IEEE 754-1985 results**

### **Description:**

Bit 22 of the FSR, FSR\_nonstandard\_fp (NS), when set to 1, causes the FPU to produce implementation-defined results that may not correspond to IEEE Standard 754-1985.

### **Implementation:**

Sparc64 FPU is “standard”, and therefore does not support a nonstandard mode.

## 19. FPU version, FSR.ver

### Description:

Bits 19:17 of the FSR, FSR.ver, identify one or more implementations of the FPU architecture.

### Implementation:

Sparc64 uses the value of 0 for this field.

## 20-21. Reserved

## 22. FPU TEM, cexc, and aexc

### Description:

An implementation may choose to implement the TEM, cexc, and aexc fields in hardware in either of two ways (see section 5.1.7.11 of SPARC-V9 Architecture Manual for details).

### Implementation:

Sparc64 implements TEM, cexc and aexc fields of FSR conforming to IEEE Std. 754-1985.

## 23. Floating-point traps

### Description:

Floating point traps may be precise or deferred. If deferred, a floating point deferred-trap queue (FQ) must be present.

### Implementation:

The only deferred traps in Sparc64 are: *fp\_exception\_other* (*ftt = unfinished\_FPop*) for FDIV with unusual arguments and the *data\_breakpoint* trap. Sparc64 does not need a floating-point deferred-trap queue because the FDIV that caused the trap is the only deferred instruction.

## 24. FPU deferred-trap queue (FQ)

### Description:

The presence, contents of, and operations on the floating-point deferred-trap queue (FQ) are implementation-dependent.

### Implementation:

Sparc64 does not have or need a floating-point deferred-trap queue.

## 25. RDPR of FQ with nonexistent FQ

### Description:

On implementations without a floating-point queue, an attempt to read the FQ with an RDPR instruction shall cause either an `illegal_instruction` exception or an `fp_exception_other` exception with `FSR.ftt` set to 4 (`sequence_error`).

### Implementation:

A RDPR of %FPQ instruction will cause an *illegal\_instruction* trap.

## 26-28. Reserved

## 29,30. Address space identifier (ASI) definitions and ASI address decoding

### Description:

The following ASI assignments are implementation-dependent: restricted ASIs (all values hex) 00..03, 05..0B, 0D..0F, 12..17, and 1A..7F; and unrestricted ASIs C0..FF.

An implementation may choose to decode only a subset of the 8-bit ASI specifier; however, it shall decode at least enough of the ASI to distinguish `ASI_PRIMARY`, `ASI_PRIMARY_LITTLE`, `ASI_AS_IF_USER_PRIMARY`, `ASI_AS_IF_USER_PRIMARY_LITTLE`, `ASI_PRIMARY_NOFAULT`, `ASI_PRIMARY_NOFAULT_LITTLE`, `ASI_SECONDARY`, `ASI_SECONDARY_LITTLE`, `ASI_AS_IF_USER_SECONDARY`, `ASI_AS_IF_USER_SECONDARY_LITTLE`, `ASI_SECONDARY_NOFAULT`, and `ASI_SECONDARY_NOFAULT_LITTLE`. If `ASI_NUCLEUS` and `ASI_NUCLEUS_LITTLE` are supported (impl. dep. #124), they must be decoded also. Finally, an implementation must always decode ASI bit<7> while `PSTATE.PRIV = 0`, so that an attempt by nonprivileged software to access a restricted ASI will always cause a `privileged_action` exception.

### Implementation:

The encoding of ASIs in the Sparc64 processor is shown below:

NR	V (M3)	PO	AS_IF	LE	M2	M1	M0
7	6	5	4	3	2	1	0



- NR (Non-Restricted). This bit conforms to Sparc V9 definition. An attempt to use a restricted ASI in non-privileged mode results in a *privileged\_action* trap.
- V (Vendor-specific). This bit conforms to Sparc V9 definition for non-restricted ASIs that are implementation-dependent (0xc0 - 0xff). This bit will be set in all ASIs that are specific to Sparc64.
- PO (Program Order). An instruction using an ASI with this bit set is executed by Sparc64 strictly in program order.
- AS\_IF. This bit conforms to Sparc V9 requirement that there be an implementation specific ASI encoding that allows the corresponding access to be made as if the CPU were executing in non-privileged mode, independent of PSTATE.PRIV.
- LE. This bit conforms to Sparc V9 definition of ASIs that specify little-endian byte ordering. If this bit is set to zero, the access is done using big-endian byte ordering.
- M2..M0. These bits are interpreted by the Sparc64 MMU.

Sparc64 does not support a nucleus context and hence does not decode ASI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE.

### 31. Catastrophic error exceptions

#### Description:

The causes and effects of catastrophic error exceptions are implementation-dependent. They may cause precise, deferred or disrupting traps.

#### Implementation:

An internal CPU watchdog time-out occurs after no instruction has been committed for  $2^{*}n$  cycles (n can be scan initialized to one of {12,14,16,18,19,20,21,22,24}, with 24 being the default value). This would take the processor into error state.

### 32. Deferred traps

#### Description:

Whether any deferred traps (and associated deferred-trap queues) are present is implementation-dependent.

#### Implementation:

Sparc64 implements a deferred trap for the following trap types:

- fp\_exception\_other (when FSR.ftt = unfinished\_FPop).
- data\_breakpoint.

Deferred trap queues are not necessary, since the trapping instruction is the only deferred instruction.

### 33. Trap precision

**Description:**

Exceptions that occur as the result of program execution may be precise or deferred, although it is recommended that such exceptions be precise. Examples include `mem_address_not_aligned` and `division_by_zero`.

**Implementation:**

Sparc64 will generate a precise trap for all traps induced by instruction execution, except for *unfinished\_FPop*, *data\_breakpoint* and *Chip\_crossing\_errors (CPU\_xing)*.

### 34. Interrupt clearing

**Description:**

How quickly a processor responds to an interrupt request and the method by which an interrupt request is removed are implementation-dependent.

**Implementation:**

When Sparc64 is ready to accept an interrupt signal (based on `PSTATE.IE` and the `PIL`), it stops issuing instructions and waits for the CPU to quiesce. It then issues instructions from the corresponding trap handler if the interrupt condition is still valid. The `TPC` points to the instruction that would have executed in the absence of the interrupt. All instructions prior to the `TPC` have completed and all instructions including and subsequent to `TPC` remain unexecuted.

### 35,36. Implementation-dependent traps and priorities

**Description:**

Trap type (`TT`) values `060(hex)..07f(hex)` are reserved for implementation-dependent exceptions. The existence of `implementation_dependent_n` traps and whether any that do exist are precise, deferred, or disrupting is implementation-dependent.

The priorities of the particular traps are relative and are implementation-dependent, because a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities.

**Implementation:**

The following trap types defined by Sparc-V9 are not used in Sparc64.

- `instruction_access_MMU_miss`.
- `internal_processor_error`

- data\_access\_MMU\_miss.
- LDQF\_mem\_address\_not\_aligned.
- STQD\_mem\_address\_not\_aligned.
- async\_data\_error.

Sparc64 defines the following implementation-dependent trap types.

- programmed\_emulation\_trap (tt=0x60, priority = 6, precise).
- data\_breakpoint (tt=0x61, priority = 14, deferred).
- IO\_parity (tt=0x62, priority = 2, precise).
- RED\_alert (tt=0x63, priority = 2, disrupting).
- CPU\_xing (tt=0x64, priority = 2, disrupting).
- Watchdog (tt=0x65, priority = 1, disrupting).
- ECC\_trap (tt=0x66, priority = 2, precise).

Sparc64 implements a special accelerated emulation trap for certain LDD and STD instructions.

### 37. Reset trap

#### Description:

Some of a processor's behavior during a reset trap is implementation-dependent.

#### Implementation:

Power-on Reset (POR) and Watchdog reset (WDR) are implemented by scanning in the reset state on Sparc64.

### 38. Effect of reset trap on implementation-dependent registers

#### Description:

Implementation-dependent registers may or may not be affected by the various reset traps.

#### Implementation:

None of the implementation-dependent registers are affected by reset traps in Sparc64.

### 39. Entering error\_state on implementation-dependent errors

#### Description:

The processor may enter error\_state when an implementation-dependent error condition occurs.

#### Implementation:

An internal CPU watchdog time-out occurs after no instruction has been committed for  $2^{**n}$  cycles (n can be scan initialized to one of {12,14,16,18,19,20,21,22,24},

with 24 being the default value). This would take the processor into error state.

#### **40. Error\_state processor state**

##### **Description:**

What occurs after error\_state is entered is implementation-dependent, but it is recommended that as much processor state as possible be preserved upon entry to error\_state.

##### **Implementation:**

On entry to error state, Sparc64 asserts the output signal CPU\_HALTED. The clock chip in the HaL system stops the clocks to the CPU in response to this signal. A scan out of processor state could be performed at this stage for diagnosis.

#### **41. Reserved**

#### **42. FLUSH instruction**

##### **Description:**

If flush is not implemented in hardware, it causes an illegal\_instruction exception and its function is performed by system software. Whether FLUSH traps is implementation-dependent.

##### **Implementation:**

Sparc64 takes an illegal\_instruction trap when a FLUSH instruction is executed.

#### **43. Reserved**

#### **44. Data access FPU trap**

##### **Description:**

If a load floating-point instruction traps with any type of access error exception, the contents of the destination floating-point register(s) either remain unchanged or are undefined.

##### **Implementation:**

Contents of destination floating-point register(s) remain unchanged.

## 45-46. Reserved

## 47. RDASR

### Description:

RDASR instructions with rd in the range 16..31 are available for implementation-dependent uses (impl. dep #8). For an RDASR instruction with rs1 in the range 16..31, the following are implementation-dependent: the interpretation of bits 13:0 and 29:25 in the instruction, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

### Implementation:

See *items 8,9* for details. Sparc64 causes an *illegal\_instruction* trap for reads of the unused ASR values.

## 48. WRASR

### Description:

WRASR instructions with rd in the range 16..31 are available for implementation-dependent uses (impl. dep. #8). For a WRASR instruction with rd in the range 16..31, the following are implementation-dependent: the interpretation of bits 18:0 in the instruction, the operation(s) performed (for example, xor) to generate the value written to the ASR, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

### Implementation:

See *items 8,9* for details. Sparc64 causes an *illegal\_instruction* trap for writes of the unused ASR values.

## 49-54 Reserved

## 55. Floating-point underflow detection

### Description:

Whether “tininess” (in IEEE 754 terms) is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.

### Implementation:

Sparc64 detects “tininess” before rounding.

## 56-100. Reserved

### 101. Maximum trap level

**Description:**

It is implementation-dependent how many additional levels, if any, past level 4 are supported.

**Implementation:**

Sparc64 implements 4 levels of traps.

### 102. Clean window trap

**Description:**

An implementation may choose either to implement automatic “cleaning” of register windows in hardware, or generate a `clean_window` trap, when needed, for window(s) to be cleaned by software.

**Implementation:**

Sparc64 generates a `clean_window` trap, when needed, for windows to be cleaned by software.

### 103. Prefetch instructions

**Description:**

The following aspects of the `PREFETCH` and `PREFETCHA` instructions are implementation-dependent: (1) whether they have an observable effect in privileged code; (2) whether they can cause a `data_access_MMU_miss` exception; (3) the attributes of the block of memory prefetched: its size (minimum = 64 bytes) and its alignment (minimum = 64-byte alignment); (4) whether each variant is implemented as a `NOP`, with its full semantics, or with common-case prefetching semantics; (5) whether and how variants 16..31 are implemented.

**Implementation:**

- (1) `PREFETCH` and `PREFETCHA` have identical affects in privileged or non-privileged code.
- (2) Can not cause a `data_access_MMU_miss` exception
- (3) Size and alignments are 128-bytes
- (4),(5) See table-1

**Table 2: Prefetch Data**

fcn	V9 Prefetch Function	Sparc64 Function
0	Prefetch for several reads	Prefetch for read
1	Prefetch for one read	Prefetch for read
2	Prefetch for several writes	Prefetch for write
3	Prefetch for one write	Prefetch for write
4	Prefetch page	Prefetch for read
5-15	Reserved	<i>illegal_instruction</i> trap
16-31	Implementation dependent	NOP

## 104. VER.manuf

### Description:

VER.manuf contains a 16-bit semiconductor manufacturer code. This field is optional, and if not present reads as zero. VER.manuf may indicate the original supplier of a second-sourced chip in cases involving mask-level second-sourcing. It is intended that the contents of VER.manuf track the JEDEC semiconductor manufacturer code as closely as possible. If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a VER.manuf value.

### Implementation:

Sparc64 uses a code of 4 for this field. This is Fujitsu's JEDEC code.

## 105. TICK register

### Description:

The difference between the values read from the TICK register on two reads should reflect the number of processor cycles executed between the reads. If an accurate count cannot always be returned, an inaccuracy should be small, bounded, and documented. An implementation may implement fewer than 63 bits in TICK.counter; however, the counter as implemented must be able to count

for at least 10 years without overflowing. Any upper bits not implemented must be read as zero.

**Implementation:**

Sparc64 implements all the bits of TICK register and returns accurate count of the processor cycles, in response to reads from TICK register.

**106. IMPDEPn instructions****Description:**

The IMPDEP1 and IMPDEP2 instructions are completely implementation-dependent. Implementation-dependent aspects include their operation, the interpretation of bits 29:25 and 18:0 in their encoding, and which (if any) exceptions they may cause.

**Implementation:**

Sparc64 uses IMPDEP2 to encode the HaL specific Floating Point Multiply-Add/Subtract instructions. IMPDEP1 is not used and will cause an illegal\_instruction trap if such an opcode is encountered. Please refer to *Sparc64 Processor User Guide* for more details.

**107. Unimplemented LDD trap****Description:**

It is implementation-dependent whether LDD and LDDA are implemented in hardware. If not, an attempt to execute either will cause an unimplemented\_LDD trap.

**Implementation:**

Sparc64 does not implement LDD and LDDA in hardware. It uses the unimplemented\_LDD trap. However in a special mode, there is partial support in hardware for these instructions. Please refer to *Sparc64 Processor User Guide* for more details.

**108. Unimplemented STD trap****Description:**

It is implementation-dependent whether STD and STDA are implemented in hardware. If not, an attempt to execute either will cause an unimplemented\_STD trap.

**Implementation:**

Sparc64 does not implement STD and STDA in hardware. It uses the unimplemented\_STD trap. However in a special mode, there is partial support in hardware for these instructions. Please refer to *Sparc64 Processor User Guide* for more details.



## 109. LDDF\_mem\_address\_not\_aligned

### Description:

LDDF and LDDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an `LDDF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the LDDF (or LDDFA) instruction and return.

### Implementation:

Sparc64 causes *LDDF\_mem\_address\_not\_aligned* trap for both word and double-word misaligned addresses.

## 110. STDF\_mem\_address\_not\_aligned

### Description:

STDF and STDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an `STDF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the STDF (or STDFA) instruction and return.

### Implementation:

Sparc64 causes *STDF\_mem\_address\_not\_aligned* trap for both word and double-word misaligned addresses.

## 111. LDQF\_mem\_address\_not\_aligned

### Description:

LDQF and LDQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an `LDQF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the LDQF (or LDQFA) instruction and return.

### Implementation:

Sparc64 generates `fp_exception_other` trap for LDQF, LDQFA instructions and kernel provides emulation routines to complete the load. It does not generate *LDQF\_mem\_address\_not\_aligned* trap.

## 112. STQF\_mem\_address\_not\_aligned

### Description:

STQF and STQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an `STQF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the STQF (or STQFA) instruction and return.

**Implementation:**

Sparc64 generates `fp_exception_other` trap for STQF, STQFA instructions and kernel provides emulation routines to complete the load. It does not generate *STQF\_mem\_address\_not\_aligned* trap.

**113. Implemented memory models****Description:**

Whether the Partial Store Order (PSO) or Relaxed Memory Order (RMO) models are supported is implementation-dependent.

**Implementation:**

Sparc64 supports *Load/Store ordering (LSO)* and *Store ordering (STO)*. *Partial Store Order (PSO)* is implemented using *LSO* and *Relaxed Memory Order (RMO)* is implemented using *STO*.

**114. RED\_state trap vector address (RSTVaddr)****Description:**

The RED\_state trap vector is located at an implementation-dependent address referred to as RSTVaddr.

**Implementation:**

Sparc64 has a scan only register that holds RSTVaddr.

**115. RED\_state processor state****Description:**

What occurs after the processor enters RED\_state is implementation-dependent.

**Implementation:**

Sparc64 has the following behavior in RED\_state.

- The output signal RED\_MODE is asserted indicating CPU is in RED\_state.
- The CPU executes in sequential mode.
- On entry into and exit from RED\_state, the CPU invalidates the on-chip instruction cache and prefetch buffers.
- Off chip data and instruction caches are disabled.
- The MMU uses a special translation mechanism.
- All I/O accesses are disabled.
- Further red state errors are ignored.
- XIR, and Chip Crossing Errors are not masked and could cause a trap.

## 116. SIR\_enable control flag

### Description:

The location of and the means of accessing the SIR\_enable control flag are implementation-dependent. In some implementations, it may be permanently zero.

### Implementation:

SIR\_enable control flag is permanently zero in Sparc64.

## 117. MMU disabled prefetch behavior

### Description:

Whether Prefetch and Non-faulting Load always succeed when the MMU is disabled is implementation-dependent.

### Implementation:

In Sparc64, Prefetch and Non-faulting Loads will have undefined behavior if the MMU is disabled.

## 118. Identifying I/O locations

### Description:

The manner in which I/O locations are identified is implementation-dependent.

### Implementation:

Please contact HaL Computer Systems for details of I/O operation.

## 119. Unimplemented values for PSTATE.MM

### Description:

The effect of writing an unimplemented memory-mode designation into PSTATE.MM is implementation-dependent

### Implementation:

Sparc64 only the most significant bit of MM is used to determine the memory model; the least significant bit is ignored. However, the system software should not use the encoding '11' since it is reserved for future SPARC-V9 extensions.

## 120. Coherence and atomicity of memory operations

### Description:

The coherence and atomicity of memory operations between processors and I/O DMA memory accesses are implementation-dependent.

**Implementation:**

In Sparc64, coherence and atomicity of memory operations between processors and I/O DMA memory accesses are variable and depend on the I/O device. Please contact HaL Computer Systems for details.

**121. Implementation-dependent memory model****Description:**

An implementation may choose to identify certain addresses and use an implementation dependent memory model for references to them.

**Implementation:**

In Sparc64, certain addresses use implementation dependent memory models for references to them. Please contact HaL Computer Systems for details.

**122. FLUSH latency****Description:**

Latency between the execution of FLUSH on one processor and the point at which the modified instructions have replaced out-dated instructions in a multiprocessor is implementation-dependent.

**Implementation:**

Not applicable since, Sparc64 does not support a multi-processor configuration.

**123. Input/output (I/O) semantics****Description:**

The semantic effect of accessing input/output (I/O) registers is implementation-dependent.

**Implementation:**

In Sparc64, the semantic effect of accessing input/output (I/O) registers is undefined.

**124. Implicit ASI when TL>0****Description:**

When  $TL > 0$ , the implicit ASI for instruction fetches, loads, and stores is implementation-dependent. See SPARC-V9 Architecture Manual section F.4.4, "Contexts," for more information.

**Implementation:**

Sparc64 uses *ASI\_PRIMARY* or *ASI\_PRIMARY\_LITTLE* for instruction fetches, loads and stores when  $TL > 0$

## 125. Address masking

### Description:

When PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination registers(s) by CALL, JMPL, RDPC, and on a trap is implementation-dependent.

### Implementation:

When PSTATE.AM bit is set on Sparc64, a full 64-bit address is transmitted to the specified destination registers by CALL, JMPL, RDPC and traps transmit all 64-bits to TPC[n] and TNPC[n].

## 126. TSTATE bits 19:18

### Description:

If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

### Implementation:

Sparc64 does not implement PSTATE bits 10 & 11 and they are read as zeroes. TSTATE bits 19 and 18 are read as zeroes.

## 127. PSTATE bits 11:10

### Description:

The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are implementation-dependent. The presence of TSTATE bits 19 and 18 is implementation-dependent. If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

### Implementation:

Sparc64 does not implement PSTATE bits 10 & 11 and they are read as zeroes. TSTATE bits 19 and 18 are read as zeroes.

## 128. CLEANWIN register update

Earlier implementations of Sparc chips implemented the V9 specification for RESTORED

using the following equation to update *CLEANWIN* register:

*if (CLEANWIN != NWINDOWS) CLEANWIN++;*

Subsequently V9 definition changed to modify the equation as:

*if (CLEANWIN < NWINDOWS-1) CLEANWIN++;*

Sparc64 implements the *RESTORED* using the current definition. The Sparc64 Kernel will ensure that *CLEANWIN* does not have a value beyond *NWINDOWS-1*.

**Chapter 4: SUN Implementation of V9 Architecture**

---

---

**UltraSPARC - II**

---

---

**V9**

**SPARC INTERNATIONAL**





## CHAPTER 4: SUN ULTRASPARC II

### 0. Introduction

This document describes the implementation on the UltraSPARC-II processor developed by Sun Microelectronics, a business unit of Sun Microsystems, Inc., of the implementation dependencies as put forth in “The SPARC Architecture Manual - Version 9” by SPARC International. The items listed below correspond to the implementation dependencies as listed in the text and by number in Appendix C of the manual along with the description of the implementation dependency from the manual. The “Implementation” section for each item describes the implementation on the UltraSPARC-II processor.

### 1. Software emulation of instructions

**Description:** whether an instruction is implemented directly by hardware, simulated by software, or emulated by firmware is implementation-dependent.

**Implementation:** all instructions are implemented in hardware except the following, which must be simulated by software.

POPC	Population count
LDQF	Load quad-precision FP register
LDQFA	Load quad-precision FP register from alternate space
STQF	Store quad-precision FP register
STQFA	Store quad-precision FP register to alternate space
F{s,d}TOq	Convert single-/double- to quad-precision FP
F{i,x}TOq	Convert 32-/64-bit integer to quad-precision FP
FqTO{s,d}	Convert quad- to single-/double-precision FP
FqTO{i,x}	Convert quad-precision FP to 32-/64-bit integer
FADDq	Quad-precision FP add
FSUBq	Quad-precision FP subtraction
FCMP{E}q	Quad-precision FP compares
FMOVqcc	Move quad-precision FP register on condition
FMOVqr	Move quad-precision FP register on integer register condition
FMOVq	Move quad-precision FP register
FABSq	Quad-precision FP absolute value
FNEGq	Quad-precision FP negate
FdMULq	Double- to quad-precision FP multiply
FMULq	Quad-precision FP multiply

FDIVq	Quad-precision FP divide
FSQRTq	Quad-precision FP square root
DONE	for fcn = 2..31 executed in nonprivileged mode
RETRY	for fcn = 2..31 executed in nonprivileged mode
SAVED	for fcn = 2..31 executed in nonprivileged mode
RESTORED	for fcn = 2..31 executed in nonprivileged mode

The DONE/RETRY/SAVED/RESTORED instructions with fcn = 2..31 executed in nonprivileged mode will take a privileged\_opcode trap rather than an illegal\_instruction trap. The opcode can be recognized by software to emulate the proper illegal\_instruction behavior. This can be done with SPARC code in the privileged\_opcode trap handler that does the following

```

PRIVILEGED_OPCODE_HANDLER:
    rdpr    %tpc, %g1
    ld      [%g1], %g2
    setx    0xc1f80000, %g3, %g4
    and     %g4, %g2, %g4    ! %g4 has op/op3 of trapping instr.
    setx    0x3e000000, %g3, %g6
    and     %g6, %g2, %g6
    srl     %g6, 25, %g6    ! %g6 has fcn of trapping instr.
check_illegal_saved_restored:
    setx    0x81880000, %g3, %g5
    subcc   %g4, %g5, %g0    ! saved/restored opcode?
    bne     check_illegal_done_retry
    subcc   %g6, 2, %g0      ! illegal fcn value?
    bge     ILLEGAL_HANDLER
    nop
check_illegal_done_retry:
    setx    0x81f00000, %g3, %g5
    subcc   %g4, %g5, %g0    ! done/retry opcode?
    bne     not_illegal
    subcc   %g6, 2, %g0      ! illegal fcn value?
    bge     ILLEGAL_HANDLER
    nop
not_illegal:
    <handle privileged_opcode exception as desired here>

```

## 2. Number of IU registers

**Description:** an implementation of the IU may contain from 64 to 258 general purpose 64 bit r registers. This corresponds to a grouping of the registers into two sets of eight global r registers, plus a circular stack of from three to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (NWINDOWS) is implementation-dependent, the total number of registers is also implementation-dependent.

Implementation: UltraSPARC-II implements eight register windows plus four sets of eight global r registers, for a total of 160 64 bit r registers.

### 3. Incorrect IEEE Std 754-1985 results

Description: an implementation may indicate that a floating-point instruction did not produce a correct ANSI/IEEE Standard 754-1985 result by generating a special floating-point unfinished or unimplemented exception. In this case, privileged mode software shall emulate any functionality not present in the hardware.

Implementation: the quad-precision floating-point instructions listed in implementation dependency #1 above all generate floating-point unimplemented exceptions.

UltraSPARC-II generates floating-point unimplemented exceptions for the following cases of subnormal operands or results.

#### Subnormal Operand Unimplemented Exception Cases:

$F\{s,d\}TO\{i,x\}$  one subnormal operand

$F\{s,d\}TO\{i,x\}$  one subnormal operand

$FSQRT\{s,d\}$  one subnormal operand

$FADD\{s,d\}$  one or two subnormal operands

$FMUL\{s,d\}$ -25 < Er < 255 (SP) one subnormal operand -54 < Er < 2047 (DP) one subnormal operand two subnormal operands

$FDIV\{s,d\}$ -25 < Er < 255 (SP) one subnormal operand  
-54 < Er < 2047 (DP) one subnormal operand  
two subnormal operands

#### Subnormal Result Unimplemented Exception Cases:

$FdTOs$  -25 < Er < 1 (SP)  
-54 < Er < 1 (DP)

$FADD\{s,d\}$ -25 < Er < 1 (SP)  
-54 < Er < 1 (DP)

FMUL{s,d}-25 < Er < 1 (SP)

-54 < Er < 1 (DP)

FDIV{s,d}-25 < Er <= 1 (SP)

-54 < Er <= 1 (DP)

Prediction of overflow, underflow and inexact traps for divide and square roots is used. For divide, pessimistic prediction occurs when underflow/overflow cannot be determined from examining the source operand exponents. For divide and square root, pessimistic prediction of inexact occurs unless one of the operands is a zero, NAN or infinity. When pessimistic prediction occurs and the exception is enabled, a floating-point unfinished exception is generated.

#### **4-5. Reserved**

### **6. I/O registers privileged status**

Description: whether I/O registers can be accessed by nonprivileged code is implementation-dependent.

Implementation: For systems using UltraSPARC-II, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation.

### **7. I/O register definitions**

Description: the contents and addresses of I/O registers are implementation-dependent

Implementation: For systems using UltraSPARC-II, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation

### **8. RDASR/WRASR target registers**

Description: software can use read/write ancillary state register instructions to read/write implementation-dependent processor registers (ASRs 16-31).

Implementation: UltraSPARC-II implements the following implementation-dependent ASRs.

<u>rd</u>	<u>name</u>	<u>access</u>
16	PERF_CONTROL_REG	RW
17	PERF_COUNTER	RW
18	DISPATCH_CONTROL_REG	RW
19	GRAPHICS_STATUS_REG	RW
20	SET_SOFTINT	W
21	CLEAR_SOFTINT	W
22	SOFTINT_REG	RW
23	TICK_CMPR_REG	RW

## 9. RDASR/WRASR privileged status

Description: whether each of the implementation-dependent read/write ancillary state register instructions (for ASRs 16-31) is privileged is implementation dependent.

Implementation: The privileged status of UltraSPARC-II's implementation-dependent registers is as follows:

<u>rd</u>	<u>name</u>	<u>access</u>
16	PERF_CONTROL_REG	PRIVILEGED
17	PERF_COUNTER	PRIVILEGED (if PERF_CONTROL_REG.PRIV = 1)
18	DISPATCH_CONTROL_REG	PRIVILEGED
19	GRAPHICS_STATUS_REG	NONPRIVILEGED
20	SET_SOFTINT	PRIVILEGED
21	CLEAR_SOFTINT	PRIVILEGED
22	SOFTINT_REG	PRIVILEGED
23	TICK_CMPR_REG	PRIVILEGED

## 10-12. Reserved

## 13. VER.impl

Description: VER.impl uniquely identifies an implementation or class of software-com-

patible implementations of the architecture. Values FFF0(hex)..FFFF(hex) are reserved and are not available for assignment.

Implementation: UltraSPARC-II uses the implementation code 0011 (hex)

## **14-15. Reserved**

## **16. IU deferred-trap queue**

Description: the existence, contents, and operation of an IU deferred-trap queue are implementation-dependent; it is not visible to user application programs under normal operating conditions.

Implementation: UltraSPARC-II does not implement a deferred-trap queue.

## **17. Reserved**

## **18. Nonstandard IEEE 754-1985 results**

Description: bit 22 of the FSR, FSR\_nonstandard\_fp (NS), when set to 1, causes the FPU to produce implementation-defined results that may not correspond to IEEE Standard 754-1985.

Implementation: if FSR.NS is set to one, the subnormal operand and results cases identified for implementation dependency #3 above, are flushed to zero.

## **19. FPU version, FSR.ver**

Description: bits 19:17 of the FSR, FSR.ver, identify one or more implementations of the FPU architecture.

Implementation: on UltraSPARC-II the FSR.VER field is set to zero.

## **20-21. Reserved**

## 22. FPU TEM, cexc, and aexc

Description: an implementation may choose to implement the TEM, cexc, and aexc fields in hardware in either of two ways (see section 5.1.7.11 of SPARC-V9 Architecture Manual for details).

Implementation: UltraSPARC-II implements the TEM, cexc and aexc fields in conformance to IEEE Std 754-1985.

## 23. Floating-point traps

Description: floating point traps may be precise or deferred. If deferred, a floating point deferred-trap queue (FQ) must be present.

Implementation: UltraSPARC-II floating-point traps are precise and it does not implement an FQ.

## 24. FPU deferred-trap queue (FQ)

Description: the presence, contents of, and operations on the floating-point deferred-trap queue (FQ) are implementation-dependent.

Implementation: UltraSPARC-II does not implement an FQ.

## 25. RDPR of FQ with nonexistent FQ

Description: on implementations without a floating-point queue, an attempt to read the FQ with an RDPR instruction shall cause either an illegal\_instruction exception or an fp\_exception\_other exception with FSR.ftt set to 4 (sequence\_error).

Implementation: attempting to read the FQ with a RDPR instruction causes an illegal\_instruction exception.

## 26-28. Reserved

## 29. Address space identifier (ASI) definitions

Description: the following ASI assignments are implementation-dependent: restricted ASIs (all values hex) 00..03, 05..0B, 0D..0F, 12..17, and 1A..7F; and unrestricted ASIs C0..FF.

Implementation: UltraSPARC-II assigns the following implementation-dependent ASI values.

restricted ASI values (all values hex):

14, 15, 1C, 1D, 24, 2C, 45, 46, 47, 48, 49, 4A, 4B, 4C, 4D, 4E, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 5A, 5B, 5C, 5D, 5E, 5F, 66, 67, 6E, 6F, 70, 71, 76, 77, 78, 79, 7E, 7F

unrestricted ASI values (all values hex):

C0, C1, C2, C3, C4, C5, C8, C9, CA, CB, CC, CD, D0, D1, D2, D3, D8, D9, DA, DB, E0, E1, F0, F1, F8, F9

## 30. ASI address decoding

Description: an implementation may choose to decode only a subset of the 8-bit ASI specifier; however, it shall decode at least enough of the ASI to distinguish ASI\_PRIMARY, ASI\_PRIMARY\_LITTLE, ASI\_AS\_IF\_USER\_PRIMARY, ASI\_AS\_IF\_USER\_PRIMARY\_LITTLE, ASI\_PRIMARY\_NOFAULT, ASI\_PRIMARY\_NOFAULT\_LITTLE, ASI\_SECONDARY, ASI\_SECONDARY\_LITTLE, ASI\_AS\_IF\_USER\_SECONDARY, ASI\_AS\_IF\_USER\_SECONDARY\_LITTLE, ASI\_SECONDARY\_NOFAULT, and ASI\_SECONDARY\_NOFAULT\_LITTLE. If ASI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE are supported (impl. dep. #124), they must be decoded also. Finally, an implementation must always decode ASI bit<7> while PSTATE.PRIV = 0, so that an attempt by nonprivileged software to access a restricted ASI will always cause a privileged\_action exception.

Implementation: UltraSPARC-II decodes the entire 8-bit ASI specifier.



### 31. Catastrophic error exceptions

**Description:** the causes and effects of catastrophic error exceptions are implementation-dependent. They may cause precise, deferred or disrupting traps.

**Implementation:** UltraSPARC-II catastrophic error exceptions cause deferred traps. The PSTATE.RED bit is not automatically set in hardware for any catastrophic error exceptions other than when trapping to MAXTL-1.

### 32. Deferred traps

**Description:** whether any deferred traps (and associated deferred-trap queues) are present is implementation-dependent.

**Implementation:** UltraSPARC-II may encounter deferred traps during memory accesses. Such errors lead to termination of the currently executing process or result in a system reset if system state has been corrupted. Error logging information allows software to determine if the system state has been corrupted.

### 33. Trap precision

**Description:** exceptions that occur as the result of program execution may be precise or deferred, although it is recommended that such exceptions be precise. Examples include `mem_address_not_aligned` and `division_by_zero`.

**Implementation:** all of the exceptions listed in the SPARC-V9 Architecture Manual section 7.3.5, item (2) are precise with the exception of `instruction_access_error`, which is deferred.

### 34. Interrupt clearing

**Description:** how quickly a processor responds to an interrupt request and the method by which an interrupt request is removed are implementation-dependent.

**Implementation:** The response time to interrupt is dependent the activity the processor is executing at the time the interrupt is received (e.g., whether executing a trap handler with PSTATE.IE=0, etc.). The interrupt request is removed by

clearing a bit in the implementation-dependent interrupt vector receive register.

### 35. Implementation-dependent traps

**Description:** trap type (TT) values 060(hex)..07f(hex) are reserved for implementation-dependent exceptions. The existence of implementation\_dependent\_n traps and whether any that do exist are precise, deferred, or disrupting is implementation-dependent.

**Implementation:** the following implementation-dependent trap types are implemented on UltraSPARC-II.

<u>TT (hex)</u>	<u>Exception</u>	<u>Category</u>
060	interrupt_vector	disrupting
061	PA_watchpoint	disrupting
062	VA_watchpoint	disrupting
063	corrected_ECC_error	disrupting
064..067	fast_instruction_access_MMU_miss	precise
068..06B	fast_data_access_MMU_miss	precise
06C..06F	fast_data_access_protection	precise

### 36. Trap priorities

**Description:** the priorities of the particular traps are relative and are implementation-dependent, because a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities.

**Implementation:** UltraSPARC-II traps are prioritized relative to each other according to the relative priorities in the SPARC-V9 Manual.

### 37. Reset trap

**Description:** some of a processor's behavior during a reset trap is implementation-dependent.

**Implementation:** UltraSPARC-II conforms to the required behavior during a reset trap. Unspecified behavior is either defined during reset or specified as requiring

initialization.

### **38. Effect of reset trap on implementation-dependent registers**

Description: implementation-dependent registers may or may not be affected by the various reset traps.

Implementation: Implementation-dependent registers on UltraSPARC-II either have defined behavior during reset traps or are specified as requiring initialization.

### **39. Entering error\_state on implementation-dependent errors**

Description: the processor may enter error\_state when an implementation-dependent error condition occurs.

Implementation: UltraSPARC-II enters error\_state only by trapping when TL = MAXTL. Any type of trap may cause this.

### **40. Error\_state processor state**

Description: what occurs after error\_state is entered is implementation-dependent, but it is recommended that as much processor state as possible be preserved upon entry to error\_state.

Implementation: Entering error\_state causes UltraSPARC-II to trigger a watchdog\_reset trap. As much state as possible is preserved during this action.

### **41. Reserved**

### **42. FLUSH instruction**

Description: if flush is not implemented in hardware, it causes an illegal\_instruction exception and its function is performed by system software. Whether FLUSH traps is implementation-dependent.

Implementation: UltraSPARC-II implements FLUSH in hardware and it can cause a data\_access\_exception if the page is mapped with side effects or no-fault-

only bits set, virtual address out of range, privilege violation, or a `data_access_MMU_miss` trap.

### 43. Reserved

### 44. Data access FPU trap

**Description:** if a load floating-point instruction traps with any type of access error exception, the contents of the destination floating-point register(s) either remain unchanged or are undefined.

**Implementation:** access error exceptions on floating-point load instructions leave the destination floating-point register contents unchanged.

### 45-46. Reserved

### 47. RDASR

**Description:** RDASR instructions with `rd` in the range 16..31 are available for implementation-dependent uses (impl. dep. #8). For an RDASR instruction with `rs1` in the range 16..31, the following are implementation-dependent: the interpretation of bits 13:0 and 29:25 in the instruction, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

**Implementation:** the bit fields specified above are not used for UltraSPARC-II implementation-dependent RDASR instructions. Reads of unused `rs1` values and reads of write-only implementation-dependent ASRs cause `illegal_instruction` traps.

### 48. WRASR

**Description:** WRASR instructions with `rd` in the range 16..31 are available for implementation-dependent uses (impl. dep. #8). For a WRASR instruction with `rd` in the range 16..31, the following are implementation-dependent: the interpretation of bits 18:0 in the instruction, the operation(s) performed (for example, `xor`) to generate the value written to the ASR, whether the

instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

**Implementation:** UltraSPARC-II does not interpret bits 18:0 of the WRASR instruction. Using WRASR to the SET\_SOFTINT and CLEAR\_SOFTINT ASRs will set and clear (respectively) bits in the SOFTINT\_REG ASR. Writes of the unused ASR values cause `illegal_instruction` traps.

## **49-54. Reserved**

## **55. Floating-point underflow detection**

**Description:** whether “tininess” (in IEEE 754 terms) is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.

**Implementation:** UltraSPARC-II detects underflow before rounding.

## **56-100. Reserved**

## **101. Maximum trap level**

**Description:** it is implementation-dependent how many additional levels, if any, past level 4 are supported.

**Implementation:** UltraSPARC-II implements 5 trap levels.

## **102. Clean window trap**

**Description:** an implementation may choose either to implement automatic “cleaning” of register windows in hardware, or generate a `clean_window` trap, when needed, for window(s) to be cleaned by software.

**Implementation:** UltraSPARC-II cleans register windows by generating a `clean_window` trap for windows to be cleaned by software.

### 103. Prefetch instructions

**Description:** the following aspects of the PREFETCH and PREFETCHA instructions are implementation-dependent: (1) whether they have an observable effect in privileged code; (2) whether they can cause a data\_access\_MMU\_miss exception; (3) the attributes of the block of memory prefetched: its size (minimum = 64 bytes) and its alignment (minimum = 64-byte alignment); (4) whether each variant is implemented as a NOP, with its full semantics, or with common-case prefetching semantics; (5) whether and how variants 16..31 are implemented.

**Implementation:** on UltraSPARC-II, PREFETCH and PREFETCHA have the same observable effect as a NOP in both privileged and nonprivileged modes.

### 104. VER.manuf

**Description:** VER.manuf contains a 16-bit semiconductor manufacturer code. This field is optional, and if not present reads as zero. VER.manuf may indicate the original supplier of a second-sourced chip in cases involving mask-level second-sourcing. It is intended that the contents of VER.manuf track the JEDEC semiconductor manufacturer code as closely as possible. If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a VER.manuf value.

**Implementation:** UltraSPARC-II uses the manufacturer code 0017(hex)

### 105. TICK register

**Description:** the difference between the values read from the TICK register on two reads should reflect the number of processor cycles executed between the reads. If an accurate count cannot always be returned, an inaccuracy should be small, bounded, and documented. An implementation may implement fewer than 63 bits in TICK.counter; however, the counter as implemented must be able to count for at least 10 years without overflowing. Any upper bits not implemented must be read as zero.

**Implementation:** UltraSPARC-II implements 63 bits of TICK.counter and reflects the number of processor clocks between reads.

## 106. IMPDEPn instructions

**Description:** the IMPDEP1 and IMPDEP2 instructions are completely implementation-dependent. Implementation-dependent aspects include their operation, the interpretation of bits 29:25 and 18:0 in their encodings, and which (if any) exceptions they may cause.

**Implementation:** UltraSPARC-II implements implementation-dependent instructions using the following field values:

op	op3	opf
10	110110	010000000
10	110110	001010000
10	110110	001010001
10	110110	001010010
10	110110	001010011
10	110110	001010100
10	110110	001010101
10	110110	001010110
10	110110	001010111
10	110110	000111011
10	110110	000111010
10	110110	000111101
10	110110	001001101
10	110110	001001011
10	110110	000110001
10	110110	000110011
10	110110	000110101
10	110110	000110110
10	110110	000110111
10	110110	000111000
10	110110	000111001
10	110110	000011000
10	110110	000011010
10	110110	001001000
10	110110	001100000
10	110110	001100001
10	110110	001111110

---

10	110110	001111111
10	110110	001110100
10	110110	001110101
10	110110	001111000
10	110110	001111001
10	110110	001101010
10	110110	001101011
10	110110	001100110
10	110110	001100111
10	110110	001111100
10	110110	001111101
10	110110	001100010
10	110110	001100011
10	110110	001110000
10	110110	001110001
10	110110	001101110
10	110110	001101111
10	110110	001101100
10	110110	001101101
10	110110	001110010
10	110110	001110011
10	110110	001111010
10	110110	001111011
10	110110	001110110
10	110110	001110111
10	110110	001101000
10	110110	001101001
10	110110	001100100
10	110110	001100101
10	110110	000101000
10	110110	000101100
10	110110	000100000
10	110110	000100100
10	110110	000100010
10	110110	000100110
10	110110	000101010
10	110110	000101110
10	110110	000000000
10	110110	000000010
10	110110	000000100
10	110110	000000110
10	110110	000001000
10	110110	000001010



10	110110	000111110
10	110110	000010000
10	110110	000010010
10	110110	000010100

### 107. Unimplemented LDD trap

Description: it is implementation-dependent whether LDD and LDDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_LDD` trap.

Implementation: UltraSPARC-II implements LDD and LDDA in hardware.

### 108. Unimplemented STD trap

Description: it is implementation-dependent whether STD and STDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_STD` trap.

Implementation: UltraSPARC-II implements STD and STDA in hardware.

### 109. LDDF\_mem\_address\_not\_aligned

Description: LDDF and LDDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an `LDDF_mem_address_not_aligned` trap, in which case the trap handler software shall emulate the LDDF (or LDDFA) instruction and return.

Implementation: UltraSPARC-II generates an `LDDF_mem_address_not_aligned` exception if an LDDF or LDDFA effective address is word-aligned but not doubleword-aligned.

### 110. STDF\_mem\_address\_not\_aligned

Description: STDF and STDFA require only word alignment. However, if the effective

address is word-aligned but not doubleword-aligned, either may cause an STDF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STDF (or STDFA) instruction and return.

Implementation: UltraSPARC-II generates an STDF\_mem\_address\_not\_aligned exception if an STDF or STDFA effective address is word-aligned but not doubleword-aligned.

### **111. LDQF\_mem\_address\_not\_aligned**

Description: LDQF and LDQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an LDQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the LDQF (or LDQFA) instruction and return.

Implementation: UltraSPARC-II does not implement the LDQF and LDQFA in hardware, they must be emulated in software using other instructions.

### **112. STQF\_mem\_address\_not\_aligned**

Description: STQF and STQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an STQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STQF (or STQFA) instruction and return.

Implementation: UltraSPARC-II does not implement the STQF and STQFA in hardware, they must be emulated in software using other instructions.

### **113. Implemented memory models**

Description: whether the Partial Store Order (PSO) or Relaxed Memory Order (RMO) models are supported is implementation-dependent.

Implementation: UltraSPARC-II supports the Partial Store Order and Relaxed Memory Order models.

### **114. RED\_state trap vector address (RSTVaddr)**

Description: the RED\_state trap vector is located at an implementation-dependent address referred to as RSTVaddr.

Implementation: RSTVaddr = 1fff0000000 (hex)

### **115. RED\_state processor state**

Description: what occurs after the processor enters RED\_state is implementation-dependent.

Implementation: On UltraSPARC-II some register contents are forced to specified values and some hardware functions are disabled upon entering RED\_state to avoid as much as possible any additional traps which would cause the processor to enter error\_state.

### **116. SIR\_enable control flag**

Description: the location of and the means of accessing the SIR\_enable control flag are implementation-dependent. In some implementations, it may be permanently zero.

Implementation: the SIR\_enable in UltraSPARC-II is permanently zero.

### **117. MMU disabled prefetch behavior**

Description: whether Prefetch and Non-faulting Load always succeed when the MMU is disabled is implementation-dependent.

Implementation: prefetch instructions behave as NOP instructions. Non-faulting Load instructions may or may not succeed when the MMU is disabled depending on the state of an implementation-dependent register determining whether the cache is enabled.

### **118. Identifying I/O locations**

Description: the manner in which I/O locations are identified is implementation-dependent.

**Implementation:** For systems using UltraSPARC-II, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation

## 119. Unimplemented values for PSTATE.MM

**Description:** the effect of writing an unimplemented memory-mode designation into PSTATE.MM is implementation-dependent

**Implementation:** UltraSPARC-II implements all three memory modes specified in the SPARC-V9 manual. If the reserved PSTATE.MM value (3) were written, UltraSPARC-II would interpret it as RMO.

## 120. Coherence and atomicity of memory operations

**Description:** the coherence and atomicity of memory operations between processors and I/O DMA memory accesses are implementation-dependent.

**Implementation:** This is dependent on the system implementation rather than the processor implementation for systems that use UltraSPARC-II

## 121. Implementation-dependent memory model

**Description:** an implementation may choose to identify certain addresses and use an implementation-dependent memory model for references to them.

**Implementation:** UltraSPARC-II does not use any implementation-dependent memory models.

## 122. FLUSH latency

**Description:** latency between the execution of FLUSH on one processor and the point at which the modified instructions have replaced out-dated instructions in a multiprocessor is implementation-dependent.

**Implementation:** This is dependent on the system implementation rather than the processor

implementation for systems that use UltraSPARC-II

### 123. Input/output (I/O) semantics

**Description:** the semantic effect of accessing input/output (I/O) registers is implementation-dependent.

**Implementation:** For systems using UltraSPARC-II, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation

### 124. Implicit ASI when TL > 0

**Description:** when TL > 0, the implicit ASI for instruction fetches, loads, and stores is implementation-dependent. See SPARC-V9 Architecture Manual section F.4.4, “Contexts,” for more information.

**Implementation:** the implicit ASI for instruction fetches, loads, and stores when TL > 0 is ASI\_PRIMARY

### 125. Address masking

**Description:** when PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination registers(s) by CALL, JMPL, RDPC, and on a trap is implementation-dependent.

**Implementation:** when PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination register(s) by CALL, JMPL, RDPC, and on a trap is zero.

### 126. TSTATE bits 19:18

**Description:** If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

Implementation: UltraSPARC-II implements TSTATE bits 19:18 to hold the state of PSTATE bits 11:10 for each previous trap level.

## 127. PSTATE bits 11:10

Description: The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are implementation-dependent. The presence of TSTATE bits 19 and 18 is implementation-dependent. If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

Implementation: PSTATE.PID1 and PSTATE.PID0 are implemented on UltraSPARC-II as selects for two additional sets of eight trap global registers. The corresponding bits in the TSTATE register are implemented to store these bits for the previous trap level.

**Chapter 5: SUN Implementation of V9 Architecture**

**UltraSPARC - Ili**

**V9**

**SPARC INTERNATIONAL**





## CHAPTER 5: SUN ULTRASPARC III

### 0. Introduction

This document describes the implementation on the UltraSPARC-III processor developed by Sun Microelectronics, a business unit of Sun Microsystems, Inc., of the implementation dependencies as put forth in “The SPARC Architecture Manual - Version 9” by SPARC International. The items listed below correspond to the implementation dependencies as listed in the text and by number in Appendix C of the manual along with the description of the implementation dependency from the manual. The “Implementation” section for each item describes the implementation on the UltraSPARC-III processor.

### 1. Software emulation of instructions

**Description:** whether an instruction is implemented directly by hardware, simulated by software, or emulated by firmware is implementation-dependent.

**Implementation:** all instructions are implemented in hardware except the following, which must be simulated by software.

POPC	Population count
LDQF	Load quad-precision FP register
LDQFA	Load quad-precision FP register from alternate space
STQF	Store quad-precision FP register
STQFA	Store quad-precision FP register to alternate space
F{s,d}TOq	Convert single-/double- to quad-precision FP
F{i,x}TOq	Convert 32-/64-bit integer to quad-precision FP
FqTO{s,d}	Convert quad- to single-/double-precision FP
FqTO{i,x}	Convert quad-precision FP to 32-/64-bit integer
FADDq	Quad-precision FP add
FSUBq	Quad-precision FP subtraction
FCMP{E}q	Quad-precision FP compares
FMOVqcc	Move quad-precision FP register on condition
FMOVqr	Move quad-precision FP register on integer register condition
FMOVq	Move quad-precision FP register
FABSq	Quad-precision FP absolute value
FNEGq	Quad-precision FP negate
FdMULq	Double- to quad-precision FP multiply
FMULq	Quad-precision FP multiply
FDIVq	Quad-precision FP divide
FSQRTq	Quad-precision FP square root
DONE	for fcn = 2..31 executed in nonprivileged mode

RETRY        for fcn = 2..31 executed in nonprivileged mode  
 SAVED        for fcn = 2..31 executed in nonprivileged mode  
 RESTORED    for fcn = 2..31 executed in nonprivileged mode

The DONE/RETRY/SAVED/RESTORED instructions with fcn = 2..31 executed in nonprivileged mode will take a privileged\_opcode trap rather than an illegal\_instruction trap. The opcode can be recognized by software to emulate the proper illegal\_instruction behavior. This can be done with SPARC code in the privileged\_opcode trap handler that does the following

```

PRIVILEGED_OPCODE_HANDLER:
    rdpr    %tpc, %g1
    ld      [%g1], %g2
    setx    0xc1f80000, %g3, %g4
    and     %g4, %g2, %g4    ! %g4 has op/op3 of trapping instr.
    setx    0x3e000000, %g3, %g6
    and     %g6, %g2, %g6
    srl     %g6, 25, %g6    ! %g6 has fcn of trapping instr.
check_illegal_saved_restored:
    setx    0x81880000, %g3, %g5
    subcc   %g4, %g5, %g0    ! saved/restored opcode?
    bne     check_illegal_done_retry
    subcc   %g6, 2, %g0      ! illegal fcn value?
    bge     ILLEGAL_HANDLER
    nop
check_illegal_done_retry:
    setx    0x81f00000, %g3, %g5
    subcc   %g4, %g5, %g0    ! done/retry opcode?
    bne     not_illegal
    subcc   %g6, 2, %g0      ! illegal fcn value?
    bge     ILLEGAL_HANDLER
    nop
not_illegal:
    <handle privileged_opcode exception as desired here>

```

## 2. Number of IU registers

**Description:** an implementation of the IU may contain from 64 to 258 general purpose 64 bit r registers. This corresponds to a grouping of the registers into two sets of eight global r registers, plus a circular stack of from three to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (NWINDOWS) is implementation-dependent, the total number of registers is also implementation-dependent.

**Implementation:** UltraSPARC-IIi implements eight register windows plus four sets of eight global r registers, for a total of 160 64 bit r registers.

### 3. Incorrect IEEE Std 754-1985 results

**Description:** an implementation may indicate that a floating-point instruction did not produce a correct ANSI/IEEE Standard 754-1985 result by generating a special floating-point unfinished or unimplemented exception. In this case, privileged mode software shall emulate any functionality not present in the hardware.

**Implementation:** the quad-precision floating-point instructions listed in implementation dependency #1 above all generate floating-point unimplemented exceptions.

UltraSPARC-III generates floating-point unimplemented exceptions for the following cases of subnormal operands or results.

#### Subnormal Operand Unimplemented Exception Cases:

$F\{s,d\}TO\{i,x\}$	one subnormal operand
$F\{s,d\}TO\{i,x\}$	one subnormal operand
$FSQRT\{s,d\}$	one subnormal operand
$FADD\{s,d\}$	one or two subnormal operands
$FMUL\{s,d\}$	-25 < Er < 255 (SP) one subnormal operand -54 < Er < 2047 (DP) one subnormal operand two subnormal operands
$FDIV\{s,d\}$	-25 < Er < 255 (SP) one subnormal operand -54 < Er < 2047 (DP) one subnormal operand two subnormal operands

#### Subnormal Result Unimplemented Exception Cases:

$FdTOs$	-25 < Er < 1 (SP) -54 < Er < 1 (DP)
$FADD\{s,d\}$	-25 < Er < 1 (SP) -54 < Er < 1 (DP)
$FMUL\{s,d\}$	-25 < Er < 1 (SP) -54 < Er < 1 (DP)

FDIV{s,d}	-25 < Er <= 1 (SP)
	-54 < Er <= 1 (DP)

Prediction of overflow, underflow and inexact traps for divide and square roots is used. For divide, pessimistic prediction occurs when underflow/overflow cannot be determined from examining the source operand exponents. For divide and square root, pessimistic prediction of inexact occurs unless one of the operands is a zero, NAN or infinity. When pessimistic prediction occurs and the exception is enabled, a floating-point unfinished exception is generated.

#### 4-5. Reserved

### 6. I/O registers privileged status

Description: whether I/O registers can be accessed by nonprivileged code is implementation-dependent.

Implementation: For systems using UltraSPARC-III, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation.

### 7. I/O register definitions

Description: the contents and addresses of I/O registers are implementation-dependent

Implementation: For systems using UltraSPARC-III, I/O register locations are memory mapped to non-cacheable address space. The location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation

### 8. RDASR/WRASR target registers

Description: software can use read/write ancillary state register instructions to read/write implementation-dependent processor registers (ASRs 16-31).

Implementation: UltraSPARC-III implements the following implementation-dependent

ASRs.

rd	name	access
16	PERF_CONTROL_REG	RW
17	PERF_COUNTER	RW
18	DISPATCH_CONTROL_REG	RW
19	GRAPHICS_STATUS_REG	RW
20	SET_SOFTINT	W
21	CLEAR_SOFTINT	W
22	SOFTINT_REG	RW
23	TICK_CMPR_REG	RW

## 9. RDASR/WRASR privileged status

**Description:** whether each of the implementation-dependent read/write ancillary state register instructions (for ASRs 16-31) is privileged is implementation dependent.

**Implementation:** The privileged status of UltraSPARC-IIi's implementation-dependent registers is as follows:

rd	name	access
16	PERF_CONTROL_REG	PRIVILEGED
17	PERF_COUNTER	PRIVILEGED (if PERF_CONTROL_REG. PRIV = 1)
18	DISPATCH_CONTROL_REG	PRIVILEGED
19	GRAPHICS_STATUS_REG	NONPRIVILEGED
20	SET_SOFTINT	PRIVILEGED
21	CLEAR_SOFTINT	PRIVILEGED
22	SOFTINT_REG	PRIVILEGED
23	TICK_CMPR_REG	PRIVILEGED

## 10-12. Reserved

## 13. VER.impl

**Description:** VER.impl uniquely identifies an implementation or class of software-compatible implementations of the architecture. Values FFF0(hex)..FFFF(hex)

are reserved and are not available for assignment.

Implementation: UltraSPARC-IIi uses the implementation code 0012 (hex) [0x12]

## **14-15. Reserved**

## **16. IU deferred-trap queue**

Description: the existence, contents, and operation of an IU deferred-trap queue are implementation-dependent; it is not visible to user application programs under normal operating conditions.

Implementation: UltraSPARC-IIi does not implement a deferred-trap queue.

## **17. Reserved**

## **18. Nonstandard IEEE 754-1985 results**

Description: bit 22 of the FSR, FSR\_nonstandard\_fp (NS), when set to 1, causes the FPU to produce implementation-defined results that may not correspond to IEEE Standard 754-1985.

Implementation: if FSR.NS is set to one, the subnormal operand and results cases identified for implementation dependency #3 above, are flushed to zero.

## **19. FPU version, FSR.ver**

Description: bits 19:17 of the FSR, FSR.ver, identify one or more implementations of the FPU architecture.

Implementation: on UltraSPARC-IIi the FSR.VER field is set to zero.

## **20-21. Reserved**

## 22. FPU TEM, cexc, and aexc

Description: an implementation may choose to implement the TEM, cexc, and aexc fields in hardware in either of two ways (see section 5.1.7.11 of SPARC-V9 Architecture Manual for details).

Implementation: UltraSPARC-IIi implements the TEM, cexc and aexc fields in conformance to IEEE Std 754-1985.

## 23. Floating-point traps

Description: floating point traps may be precise or deferred. If deferred, a floating point deferred-trap queue (FQ) must be present.

Implementation: UltraSPARC-IIi floating-point traps are precise and it does not implement an FQ.

## 24. FPU deferred-trap queue (FQ)

Description: the presence, contents of, and operations on the floating-point deferred-trap queue (FQ) are implementation-dependent.

Implementation: UltraSPARC-IIi does not implement an FQ.

## 25. RDPR of FQ with nonexistent FQ

Description: on implementations without a floating-point queue, an attempt to read the FQ with an RDPR instruction shall cause either an illegal\_instruction exception or an fp\_exception\_other exception with FSR.ftt set to 4 (sequence\_error).

Implementation: attempting to read the FQ with a RDPR instruction causes an illegal\_instruction exception.

## 26-28. Reserved

## 29. Address space identifier (ASI) definitions

**Description:** the following ASI assignments are implementation-dependent: restricted ASIs (all values hex) 00..03, 05..0B, 0D..0F, 12..17, and 1A..7F; and unrestricted ASIs C0..FF.

**Implementation:** UltraSPARC-III assigns the following implementation-dependent ASI values.

restricted ASI values (all values hex):

14, 15, 1C, 1D, 24, 2C, 45, 46, 47, 48, 49, 4A, 4B, 4C, 4D, 4E, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 5A, 5B, 5C, 5D, 5E, 5F, 66, 67, 6E, 6F, 70, 71, 76, 77, 78, 79, 7E, 7F

unrestricted ASI values (all values hex):

C0, C1, C2, C3, C4, C5, C8, C9, CA, CB, CC, CD, D0, D1, D2, D3, D8, D9, DA, DB, E0, E1, F0, F1, F8, F9

## 30. ASI address decoding

**Description:** an implementation may choose to decode only a subset of the 8-bit ASI specifier; however, it shall decode at least enough of the ASI to distinguish ASI\_PRIMARY, ASI\_PRIMARY\_LITTLE, ASI\_AS\_IF\_USER\_PRIMARY, ASI\_AS\_IF\_USER\_PRIMARY\_LITTLE, ASI\_PRIMARY\_NOFAULT, ASI\_PRIMARY\_NOFAULT\_LITTLE, ASI\_SECONDARY, ASI\_SECONDARY\_LITTLE, ASI\_AS\_IF\_USER\_SECONDARY, ASI\_AS\_IF\_USER\_SECONDARY\_LITTLE, ASI\_SECONDARY\_NOFAULT, and ASI\_SECONDARY\_NOFAULT\_LITTLE. If ASI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE are supported (impl. dep. #124), they must be decoded also. Finally, an implementation must always decode ASI bit<7> while PSTATE.PRIV = 0, so that an attempt by nonprivileged software to access a restricted ASI will always cause a privileged\_action exception.

**Implementation:** UltraSPARC-III decodes the entire 8-bit ASI specifier.

## 31. Catastrophic error exceptions



Description: the causes and effects of catastrophic error exceptions are implementation-dependent. They may cause precise, deferred or disrupting traps.

Implementation: UltraSPARC-IIi catastrophic error exceptions cause deferred traps. The PSTATE.RED bit is not automatically set in hardware for any catastrophic error exceptions other than when trapping to MAXTL-1.

### 32. Deferred traps

Description: whether any deferred traps (and associated deferred-trap queues) are present is implementation-dependent.

Implementation: UltraSPARC-IIi may encounter deferred traps during memory accesses. Such errors lead to termination of the currently executing process or result in a system reset if system state has been corrupted. Error logging information allows software to determine if the system state has been corrupted.

### 33. Trap precision

Description: exceptions that occur as the result of program execution may be precise or deferred, although it is recommended that such exceptions be precise. Examples include `mem_address_not_aligned` and `division_by_zero`.

Implementation: all of the exceptions listed in the SPARC-V9 Architecture Manual section 7.3.5, item (2) are precise with the exception of `instruction_access_error`, which is deferred.

### 34. Interrupt clearing

Description: how quickly a processor responds to an interrupt request and the method by which an interrupt request is removed are implementation-dependent.

Implementation: The response time to interrupt is dependent the activity the processor is executing at the time the interrupt is received (e.g., whether executing a trap handler with PSTATE.IE=0, etc.). The interrupt request is removed by clearing a bit in the implementation-dependent interrupt vector receive register.

### 35. Implementation-dependent traps

**Description:** trap type (TT) values 060(hex)..07f(hex) are reserved for implementation-dependent exceptions. The existence of implementation\_dependent\_n traps and whether any that do exist are precise, deferred, or disrupting is implementation-dependent.

**Implementation:** the following implementation-dependent trap types are implemented on UltraSPARC-IIi.

TT (hex)	Exception	Category
060	interrupt_vector	disrupting
061	PA_watchpoint	disrupting
062	VA_watchpoint	disrupting
063	corrected_ECC_error	disrupting
064..067	fast_instruction_access_MMU_miss	precise
068..06B	fast_data_access_MMU_miss	precise
06C..06F	fast_data_access_protection	precise

### 36. Trap priorities

**Description:** the priorities of the particular traps are relative and are implementation-dependent, because a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities.

**Implementation:** UltraSPARC-IIi traps are prioritized relative to each other according to the relative priorities in the SPARC-V9 Manual.

### 37. Reset trap

**Description:** some of a processor's behavior during a reset trap is implementation-dependent.

**Implementation:** UltraSPARC-IIi conforms to the required behavior during a reset trap. Unspecified behavior is either defined during reset or specified as requiring initialization.

### 38. Effect of reset trap on implementation-dependent registers

Description: implementation-dependent registers may or may not be affected by the various reset traps.

Implementation: Implementation-dependent registers on UltraSPARC-IIi either have defined behavior during reset traps or are specified as requiring initialization.

### 39. Entering error\_state on implementation-dependent errors

Description: the processor may enter error\_state when an implementation-dependent error condition occurs.

Implementation: UltraSPARC-IIi enters error\_state only by trapping when TL = MAXTL. Any type of trap may cause this.

### 40. Error\_state processor state

Description: what occurs after error\_state is entered is implementation-dependent, but it is recommended that as much processor state as possible be preserved upon entry to error\_state.

Implementation: Entering error\_state causes UltraSPARC-IIi to trigger a watchdog\_reset trap. As much state as possible is preserved during this action.

### 41. Reserved

### 42. FLUSH instruction

Description: if flush is not implemented in hardware, it causes an illegal\_instruction exception and its function is performed by system software. Whether FLUSH traps is implementation-dependent.

Implementation: UltraSPARC-IIi implements FLUSH in hardware and it can cause a data\_access\_exception if the page is mapped with side effects or no-fault-only bits set, virtual address out of range, privilege violation, or a data\_access\_MMU\_miss trap.

### 43. Reserved

### 44. Data access FPU trap

Description: if a load floating-point instruction traps with any type of access error exception, the contents of the destination floating-point register(s) either remain unchanged or are undefined.

Implementation: access error exceptions on floating-point load instructions leave the destination floating-point register contents unchanged.

### 45-46. Reserved

### 47. RDASR

Description: RDASR instructions with rd in the range 16..31 are available for implementation-dependent uses (impl. dep. #8). For an RDASR instruction with rs1 in the range 16..31, the following are implementation-dependent: the interpretation of bits 13:0 and 29:25 in the instruction, whether the instruction is privileged (impl. dep. #9), and whether it causes an illegal\_instruction trap.

Implementation: the bit fields specified above are not used for UltraSPARC-IIi implementation-dependent RDASR instructions. Reads of unused rs1 values and reads of write-only implementation-dependent ASRs cause illegal\_instruction traps.

### 48. WRASR

Description: WRASR instructions with rd in the range 16..31 are available for implementation-dependent uses (impl. dep. #8). For a WRASR instruction with rd in the range 16..31, the following are implementation-dependent: the interpretation of bits 18:0 in the instruction, the operation(s) performed (for example, xor) to generate the value written to the ASR, whether the instruction is privileged (impl. dep. #9), and whether it causes an illegal\_instruction trap.

Implementation: UltraSPARC-IIi does not interpret bits 18:0 of the WRASR instruction.

Using WRASR to the SET\_SOFTINT and CLEAR\_SOFTINT ASRs will set and clear (respectively) bits in the SOFTINT\_REG ASR. Writes of the unused ASR values cause illegal\_instruction traps.

## **49-54. Reserved**

## **55. Floating-point underflow detection**

Description: whether “tininess” (in IEEE 754 terms) is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.

Implementation: UltraSPARC-IIi detects underflow before rounding.

## **56-100. Reserved**

## **101. Maximum trap level**

Description: it is implementation-dependent how many additional levels, if any, past level 4 are supported.

Implementation: UltraSPARC-IIi implements 5 trap levels.

## **102. Clean window trap**

Description: an implementation may choose either to implement automatic “cleaning” of register windows in hardware, or generate a clean\_window trap, when needed, for window(s) to be cleaned by software.

Implementation: UltraSPARC-IIi cleans register windows by generating a clean\_window trap for windows to be cleaned by software.

## **103. Prefetch instructions**

Description: the following aspects of the PREFETCH and PREFETCHA instructions

are implementation-dependent: (1) whether they have an observable effect in privileged code; (2) whether they can cause a `data_access_MMU_miss` exception; (3) the attributes of the block of memory prefetched: its size (minimum = 64 bytes) and its alignment (minimum = 64-byte alignment); (4) whether each variant is implemented as a NOP, with its full semantics, or with common-case prefetching semantics; (5) whether and how variants 16..31 are implemented.

Implementation: on UltraSPARC-IIi, `PREFETCH` and `PREFETCHA` instructions with the `fcn=0..4` have the following meanings:

FCN	Function	Action
0	Prefetch for several reads	generate <code>read_to_share</code> request if desired line is not present in E-cache
1	Prefetch for one read	generate <code>read_to_share</code> request if desired line is not present in E-cache
2	Prefetch page	generate <code>read_to_share</code> request if desired line is not present in E-cache
FCN	Function	Action
3	Prefetch for several writes	generate <code>read_to_own</code> request if desired line is not present in E-cache in either E or M state
4	Prefetch for one write	generate <code>read_to_own</code> request if desired line is not present in E-cache in either E or M state

## 104. VER.manuf

Description: `VER.manuf` contains a 16-bit semiconductor manufacturer code. This field is optional, and if not present reads as zero. `VER.manuf` may indicate the original supplier of a second-sourced chip in cases involving mask-level second-sourcing. It is intended that the contents of `VER.manuf` track the JEDEC semiconductor manufacturer code as closely as possible. If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a `VER.manuf` value.

Implementation: UltraSPARC-IIi uses the manufacturer code 0017(hex)

## 105. TICK register

**Description:** the difference between the values read from the TICK register on two reads should reflect the number of processor cycles executed between the reads. If an accurate count cannot always be returned, an inaccuracy should be small, bounded, and documented. An implementation may implement fewer than 63 bits in TICK.counter; however, the counter as implemented must be able to count for at least 10 years without overflowing. Any upper bits not implemented must be read as zero.

**Implementation:** UltraSPARC-IIi implements 63 bits of TICK.counter and reflects the number of processor clocks between reads.

## 106. IMPDEPn instructions

**Description:** the IMPDEP1 and IMPDEP2 instructions are completely implementation-dependent. Implementation-dependent aspects include their operation, the interpretation of bits 29:25 and 18:0 in their encodings, and which (if any) exceptions they may cause.

**Implementation:** UltraSPARC-IIi implements implementation-dependent instructions using the following field values:

op	op3	opf
10	110110	010000000
10	110110	001010000
10	110110	001010001
10	110110	001010010
10	110110	001010011
10	110110	001010100
10	110110	001010101
10	110110	001010110
10	110110	001010111
10	110110	000111011
10	110110	000111010
10	110110	000111101
10	110110	001001101
10	110110	001001011
10	110110	000110001

---

10	110110	000110011
10	110110	000110101
10	110110	000110110
10	110110	000110111
10	110110	000111000
10	110110	000111001
10	110110	000011000
10	110110	000011010
10	110110	001001000
10	110110	001100000
10	110110	001100001
10	110110	001111110
10	110110	001111111
10	110110	001110100
10	110110	001110101
10	110110	001111000
10	110110	001111001
10	110110	001101010
10	110110	001101011
10	110110	001100110
10	110110	001100111
10	110110	001111100
10	110110	001111101
10	110110	001100010
10	110110	001100011
10	110110	001110000
10	110110	001110001
10	110110	001101110
10	110110	001101111
10	110110	001101100
10	110110	001101101
10	110110	001110010
10	110110	001110011
10	110110	001111010
10	110110	001111011
10	110110	001110110
10	110110	001110111
10	110110	001101000
10	110110	001101001
10	110110	001100100
10	110110	001100101
10	110110	000101000



---

10	110110	000101100
10	110110	000100000
10	110110	000100100
10	110110	000100010
10	110110	000100110
10	110110	000101010
10	110110	000101110
10	110110	000000000
10	110110	000000010
10	110110	000000100
10	110110	000000110
10	110110	000001000
10	110110	000001010
10	110110	000111110
10	110110	000010000
10	110110	000010010
10	110110	000010100

### 107. Unimplemented LDD trap

Description: it is implementation-dependent whether LDD and LDDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_LDD` trap.

Implementation: UltraSPARC-IIi implements LDD and LDDA in hardware.

### 108. Unimplemented STD trap

Description: it is implementation-dependent whether STD and STDA are implemented in hardware. If not, an attempt to execute either will cause an `unimplemented_STD` trap.

Implementation: UltraSPARC-IIi implements STD and STDA in hardware.

### 109. LDDF\_mem\_address\_not\_aligned

**Description:** LDDF and LDDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an LDDF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the LDDF (or LDDFA) instruction and return.

**Implementation:** UltraSPARC-IIi generates an LDDF\_mem\_address\_not\_aligned exception if an LDDF or LDDFA effective address is word-aligned but not doubleword-aligned.

### **110. STDF\_mem\_address\_not\_aligned**

**Description:** STDF and STDFA require only word alignment. However, if the effective address is word-aligned but not doubleword-aligned, either may cause an STDF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STDF (or STDFA) instruction and return.

**Implementation:** UltraSPARC-IIi generates an STDF\_mem\_address\_not\_aligned exception if an STDF or STDFA effective address is word-aligned but not doubleword-aligned.

### **111. LDQF\_mem\_address\_not\_aligned**

**Description:** LDQF and LDQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an LDQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the LDQF (or LDQFA) instruction and return.

**Implementation:** UltraSPARC-IIi does not implement the LDQF and LDQFA in hardware, they must be emulated in software using other instructions.

### **112. STQF\_mem\_address\_not\_aligned**

**Description:** STQF and STQFA require only word alignment. However, if the effective address is word-aligned but not quadword-aligned, either may cause an STQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STQF (or STQFA) instruction and return.

**Implementation:** UltraSPARC-IIi does not implement the STQF and STQFA in hardware, they must be emulated in software using other instructions.

### 113. Implemented memory models

Description: whether the Partial Store Order (PSO) or Relaxed Memory Order (RMO) models are supported is implementation-dependent.

Implementation: UltraSPARC-III supports the Partial Store Order and Relaxed Memory Order models.

### 114. RED\_state trap vector address (RSTVaddr)

Description: the RED\_state trap vector is located at an implementation-dependent address referred to as RSTVaddr.

Implementation: RSTVaddr = 1fff0000000 (hex)

### 115. RED\_state processor state

Description: what occurs after the processor enters RED\_state is implementation-dependent.

Implementation: On UltraSPARC-III some register contents are forced to specified values and some hardware functions are disabled upon entering RED\_state to avoid as much as possible any additional traps which would cause the processor to enter error\_state.

### 116. SIR\_enable control flag

Description: the location of and the means of accessing the SIR\_enable control flag are implementation-dependent. In some implementations, it may be permanently zero.

Implementation: the SIR\_enable in UltraSPARC-III is permanently zero.

### 117. MMU disabled prefetch behavior

Description: whether Prefetch and Non-faulting Load always succeed when the MMU is disabled is implementation-dependent.

Implementation: When the data MMU is disabled, accesses are assumed to be non-cacheable and with side-effect. Non-faulting loads encountered with the MMU is disabled cause a `data_access_exception` trap with `SFSR.FT-2` (speculative load to page with side-effect attribute). Prefetch behaves as a NOP when the MMU is disabled.

## 118. Identifying I/O locations

Description: the manner in which I/O locations are identified is implementation-dependent.

Implementation: For systems using UltraSPARC-IIi, I/O register locations are memory mapped to non-cacheable address space. In general, the location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation. PCI bus I/O Space is hard-wired to locations `PA[40:0] = 1FE0200000(hex)` through `PA[40:0] = 1FE0201FFFF(hex)`.

## 119. Unimplemented values for PSTATE.MM

Description: the effect of writing an unimplemented memory-mode designation into PSTATE.MM is implementation-dependent

Implementation: UltraSPARC-IIi implements all three memory modes specified in the SPARC-V9 manual. If the reserved PSTATE.MM value (3) were written, UltraSPARC-IIi would interpret it as RMO.

## 120. Coherence and atomicity of memory operations

Description: the coherence and atomicity of memory operations between processors and I/O DMA memory accesses are implementation-dependent.

Implementation: This is dependent on the system implementation rather than the processor implementation for systems that use UltraSPARC-IIi

## 121. Implementation-dependent memory model

Description: an implementation may choose to identify certain addresses and use an

implementation-dependent memory model for references to them.

Implementation: UltraSPARC-III does not use any implementation-dependent memory models.

## 122. FLUSH latency

Description: latency between the execution of FLUSH on one processor and the point at which the modified instructions have replaced out-dated instructions in a multiprocessor is implementation-dependent.

Implementation: This is dependent on the system implementation rather than the processor implementation for systems that use UltraSPARC-III

## 123. Input/output (I/O) semantics

Description: the semantic effect of accessing input/output (I/O) registers is implementation-dependent.

Implementation: For systems using UltraSPARC-III, the location, access, contents, and side effects of the I/O registers are dependent on the system implementation, not the processor implementation

## 124. Implicit ASI when TL > 0

Description: when TL > 0, the implicit ASI for instruction fetches, loads, and stores is implementation-dependent. See SPARC-V9 Architecture Manual section F.4.4, "Contexts," for more information.

Implementation: the implicit ASI for instruction fetches, loads, and stores when TL > 0 is ASI\_PRIMARY

## 125. Address masking

Description: when PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination registers(s) by CALL, JMPL, RDPC, and on a trap is implementation-dependent.

Implementation: when PSTATE.AM = 1, the value of the high-order 32-bits of the PC transmitted to the specified destination register(s) by CALL, JMPL, RDPC, and on a trap is zero.

## 126. TSTATE bits 19:18

Description: If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

Implementation: UltraSPARC-III implements TSTATE bits 19:18 to hold the state of PSTATE bits 11:10 for each previous trap level.

## 127. PSTATE bits 11:10

Description: The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are implementation-dependent. The presence of TSTATE bits 19 and 18 is implementation-dependent. If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

Implementation: PSTATE.PID1 and PSTATE.PID0 are implemented on UltraSPARC-III as selects for two additional sets of eight trap global registers. The corresponding bits in the TSTATE register are implemented to store these bits for the previous trap level.

**Chapter 6: HAL Implementation of V9 Architecture**

---

---

**SPARC 64-III**

---

---

**V9**

**SPARC INTERNATIONAL**





---

## CHAPTER 6: HAL SPARC64-III

### 0. Introduction

This document describes the implementation details of the *SPARC64-III* processor developed by *HAL Computer Systems*. The items listed below correspond to the implementation dependencies as listed in the text and by number in Appendix C of “*The SPARC Architecture Manual - Version 9*” by *SPARC International*, along with the description of the implementation dependency. The “Implementation” section for each item describes the SPARC64-III processor.

### 1. Software emulated instructions

#### Description:

Whether an instruction is implemented directly by hardware, simulated by software, or implemented by firmware is implementation-dependent.

#### Implementation:

SPARC64-III does not implement the following instructions in hardware:

- *All floating point instructions with quad operands or results*  
These operations will take an *fp\_exception\_other* trap with *FSR.ftt = unimplemented\_FPop*. The kernel will then emulate the quad operation and store the result into floating-point registers as defined by Sparc-V9 manual.
- *popc*  
This instruction will cause an *illegal\_instruction* trap if executed. Kernel emulation routines will be provided to complete the action.

### 2. Number of IU registers

#### Description:

An implementation of the IU may contain from 64 to 528 general purpose 64 bit registers. This corresponds to a grouping of the registers into two sets of eight global registers, plus a circular stack of from 3 to 32 sets of 16 registers each, known as register windows. Since the number of register windows present (NWINDOWS) is implementation-dependent, the total number of registers is also implementation-dependent.

#### Implementation:

SPARC64-III implements 5 16-register sets (windows) in hardware. Thus there are 96 integer registers visible to software. They are:

- 8 global registers
- 8 alternate global registers
- 5 windows of 16 registers each (=80 registers)

### 3. Incorrect IEEE Std 754-1985 results

#### Description:

An implementation may indicate that a floating-point instruction did not produce aANSI/IEEE Standard 754-1985 result by generating a special floating-point unfinished or unimplemented exception. In this case, privileged mode software shall emulate any functionality not present in the hardware.

#### Implementation:

SPARC64-III in conjunction with the kernel emulation code produces the correct IEEE results required in this section.

- Traps Inhibit Results  
SPARC64-III in conjunction with the kernel emulation code produces results
- Trapped Underflow Definition (UFM=1)  
SPARC64-III detects “tininess” before rounding as recommended.
- Untrapped Underflow Definition (UFM=0)  
SPARC64-III meets these requirements with some help from the kernel divide/squarefixup code.
- Floating-Point Nonstandard Mode  
SPARC64-III FPU is “standard”, and therefore does not support a nonstandard

### 4-5. Reserved

### 6. I/O registers privileged status

#### Description:

Whether I/O registers can be accessed by non privileged code is

#### Implementation:

In SPARC64-III some I/O registers can be accessed by non privileged code.

### 7. I/O register definitions

#### Description:

The contents and addresses of I/O registers are implementation-dependent.

#### Implementation:

Please contact HaL for details of I/O registers.

## 8.9. RDASR/WRASR target registers and privileged status

### Description:

Software can use read/write ancillary state register instructions to read/writedependent processor registers (ASRs 16-31).

Whether each of the implementation-dependent read/write ancillary state register(for ASRs 16-31) is privileged is implementation dependent.

### Implementation:

SPARC64-III implements 9 implementation-dependent ASR registers.

- Hardware mode Register(ASR18): These register controls, Branch prediction mode andHardware memory models.
- Graphic Status Register(ASR19): Access to this register will cause fp\_disabledeither PSTATE.PER or FPRS.FER is 0.
- Schedule Interrupt(SCHED\_INT) Register (ASR22): The OS kernel uses this privileged,write register to schedule interrupts.
- TICK match Register(ASR23): Privileged read/write register.
- Instruction Access Fault Type Register(ASR24): Privileged , read only register isthe hardware on instruction\_access\_error traps.
- Software Scratch Registers 0 through 3(ASR25): These registers are privileged,
- Data Breakpoint Registers(ASR26A): These privileged read/write registers are usedany data accesses to a double word aligned breakpoint address. ASR26B: privilegedwrite register specifies the double-word aligned virtual address of the data
- Fault Address Register (ASR28) and Fault Access Type (ASR29)  
These registers facilitate the handling of traps that involve a data memory access.are privileged and read-only. System software must take care to read thesea fault handler before any other fault can occur that would overwrite them.
- Performance Monitor Register (ASR30)  
This privilege read/write register is used to evaluate processor performance.
- State Control Register (ASR31)  
ASR31 is a 16bit implementation specific register that contains a set of flags forthe state of the CPU, MMU and Caches. The register is privileged and can be

## 10-12 Reserved

## 13. VER.impl

### Description:

VER.impl uniquely identifies an implementation or class of software-compatibilities of the architecture. Values FFF0(hex)..FFFF(hex) are reserved and are not available for assignment.

**Implementation:**

SPARC64-III uses a version number of 3.

**14-15 Reserved****16. IU deferred-trap queue****Description:**

The existence, contents, and operation of an IU deferred-trap queue are dependent; it is not visible to user application programs under normal operating conditions

**Implementation:**

SPARC64-III does not need and therefore does not implement an IU deferred-trap

**17. Reserved****18. Nonstandard IEEE 754-1985 results****Description:**

Bit 22 of the FSR, FSR\_nonstandard\_fp (NS), when set to 1, causes the FPU to implementa-tion-defined results that may not correspond to IEEE Standard 754-1985.

**Implementation:**

SPARC64-III FPU is “standard”, and therefore does not support a nonstandard

**19. FPU version, FSR.ver****Description:**

Bits 19:17 of the FSR, FSR.ver, identify one or more implementations of the FPU ure.

**Implementation:**

SPARC64-III uses the value of 0 for this field.

**20-21. Reserved****22. FPU TEM, cexc, and aexc****Description:**

An implementation may choose to implement the TEM, cexc, and aexc fields in either of two

ways (see section 5.1.7.11 of SPARC-V9 Architecture Manual for details).

**Implementation:**

SPARC64-III implements TEM, cexc and aexc fields of FSR conforming to IEEE Std.1985.

**23. Floating-point traps****Description:**

Floating point traps may be precise or deferred. If deferred, a floating pointqueue (FQ) must be present.

**Implementation:**

Floating point traps are always precise.

**24. FPU deferred-trap queue (FQ)****Description:**

The presence, contents of, and operations on the floating-point deferred-trap queueimplementation-dependent.

**Implementation:**

SPARC64-III does not have or need a floating-point deferred-trap queue.

**25. RDPR of FQ with nonexistent FQ****Description:**

On implementations without a floating-point queue, an attempt to read the FQ withinstruction shall cause either an illegal\_instruction exception or an fp\_exception\_other exception with FSR.ftt set to 4 (sequence\_error).

**Implementation:**

A RDPR of %FPQ instruction will cause an *illegal\_instruction* trap.

**26-28. Reserved****29,30. Address space identifier (ASI) definitions and ASI address decoding****Description:**

The following ASI assignments are implementation-dependent: restricted ASIs (allhex) 00..03, 05..0B, 0D..0F, 12..17, and 1A..7F; and unrestricted ASIs C0..FF.

An implementation may choose to decode only a subset of the 8-bit ASI specifier;shall decode

at least enough of the ASI to distinguish ASI\_PRIMARY, ASI\_PRIMARY\_LITTLE, ASI\_AS\_IF\_USER\_PRIMARY, ASI\_AS\_IF\_USER\_PRIMARY\_LITTLE, ASI\_PRIMARY\_NOFAULT, ASI\_PRIMARY\_NOFAULT\_LITTLE, ASI\_SECONDARY, ASI\_SECONDARY\_LITTLE, ASI\_AS\_IF\_USER\_SECONDARY, ASI\_AS\_IF\_USER\_SECONDARY\_LITTLE, ASI\_SECONDARY\_NOFAULT, and ASI\_SECONDARY\_NOFAULT\_LITTLE. If ASI\_NUCLEUS and ASI\_NUCLEUS\_LITTLE are supported (impl. dep. #124), they must be decoded also. Finally, an implementation must always decode ASI bit<7> while PSTATE.PRIV = 0, so that an attempt by nonprivileged software to access a restricted ASI will always cause a privileged\_action exception.

**Implementation:**

Please See pg 409 of *SPARC64-III user Guide*.(L ASR Assignments).

**31. Catastrophic error exceptions****Description:**

The causes and effects of catastrophic error exceptions are may cause precise, deferred or disrupting traps.

**Implementation:**

An internal CPU watchdog time-out occurs after no instruction has been committed cycles (n can be scan initialized to one of { 12,16,18,20,22,24,28,30}). This would take the processor into error state.

**32. Deferred traps****Description:**

Whether any deferred traps (and associated deferred-trap queues) are present is condition-dependent.

**Implementation:**

SPARC64-III implements a deferred trap for the following trap types:

- data\_breakpoint.

Deferred trap queues are not necessary, since the trapping instruction is the only instruction.

**33. Trap precision****Description:**

Exceptions that occur as the result of program execution may be precise or it is recommended that such exceptions be precise. Examples include

mem\_address\_not\_aligned and division\_by\_zero.

**Implementation:**

SPARC64-III will generate a precise trap for all traps induced by instructiondata\_breakpoint.

### 34. Interrupt clearing

**Description:**

How quickly a processor responds to an interrupt request and the method by which an interrupt request is removed are implementation-dependent.

**Implementation:**

Please See *SPARC64-III user guide* pg. 427 (N Interrupt Handling)

### 35,36. Implementation-dependent traps and priorities

**Description:**

Trap type (TT) values 060(hex)..07f(hex) are reserved for implementation-dependent traps. The existence of implementation\_dependent\_n traps and whether any that do exist are precise, deferred, or disrupting is implementation-dependent.

The priorities of the particular traps are relative and a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities.

**Implementation:**

The following trap types defined by Sparc-V9 are not used in SPARC64-III. Please See *SPARC64-III user guide* pg. 152 (7.5.3.3 Unimplemented Traps in SPARC64-III)

SPARC64-III defines the following implementation-dependent trap types. Please See *SPARC64-III user guide* pg. 341 (B IEEE Std 754-1985 Requirements for SPARC-V9)

### 37. Reset trap

**Description:**

Some of a processor's behavior during a reset trap is

**Implementation:**

Power-on Reset (POR) are implemented by scanning in the reset state on

### 38. Effect of reset trap on implementation-dependent registers

**Description:**

Implementation-dependent registers may or may not be affected by the various reset

**Implementation:**

All register gets affected on POR  
XIR all register except ASR31 gets affected..

### 39. Entering error\_state on implementation-dependent errors

**Description:**

The processor may enter error\_state when an implementation-dependent error occurs.

**Implementation:**

An internal CPU watchdog time-out occurs after no instruction has been committed cycles (n can be scan initialized to one of { 12,16,18,20,22,24,28,30}). This would take the processor into error state.

### 40. Error\_state processor state

**Description:**

What occurs after error\_state is entered is implementation-dependent, but it is that as much processor state as possible be preserved upon entry to error\_state.

**Implementation:**

On entry to error state, SPARC64-III asserts the output signal P\_FERR. . Most error register state will be preserved and can be read after a power on reset.

### 41. Reserved

### 42. FLUSH instruction

**Description:**

If flush is not implemented in hardware, it causes an illegal\_instruction exception to be performed by system software. Whether FLUSH traps is implementation-dependent.

**Implementation:**

SPARC64-III implements a FLUSH instruction.



### 43. Reserved

### 44. Data access FPU trap

**Description:**

If a load floating-point instruction traps with any type of access error exception, of the destination floating-point register(s) either remain unchanged or are undefined.

**Implementation:**

Contents of destination floating-point register(s) remain unchanged.

### 45-46. Reserved

### 47. RDASR

**Description:**

RDASR instructions with rd in the range 16..31 are available for uses (impl. dep. #8). For an RDASR instruction with rs1 in the range 16..31, the following are implementation-dependent: the interpretation of bits 13:0 and 29:25 in the instruction, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

**Implementation:**

See *items 8,9* for details. SPARC64-III causes an `illegal_instruction` trap for reads of the unused ASR values.

### 48. WRASR

**Description:**

WRASR instructions with rd in the range 16..31 are available for uses (impl. dep. #8). For a WRASR instruction with rd in the range 16..31, the following are implementation-dependent: the interpretation of bits 18:0 in the instruction, the operation(s) performed (for example, xor) to generate the value written to the ASR, whether the instruction is privileged (impl. dep. #9), and whether it causes an `illegal_instruction` trap.

**Implementation:**

See *items 8,9* for details. SPARC64-III causes an `illegal_instruction` trap for writes of the unused ASR values.

## 49-54 Reserved

## 55. Floating-point underflow detection

### Description:

Whether “tininess” (in IEEE 754 terms) is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.

### Implementation:

SPARC64-III detects “tininess” before rounding.

## 56-100. Reserved

## 101. Maximum trap level

### Description:

It is implementation-dependent how many additional levels, if any, past level 4 are

### Implementation:

SPARC64-III implements 4 levels of traps.

## 102. Clean window trap

### Description:

An implementation may choose either to implement automatic “cleaning” of hardware, or generate a `clean_window` trap, when needed, for window(s) to be cleaned by software.

### Implementation:

SPARC64-III generates a `clean_window` trap, when needed, for windows to be cleaned by software.

## 103. Prefetch instructions

### Description:

The following aspects of the `PREFETCH` and `PREFETCHA` instructions are dependent: (1) whether they have an observable effect in privileged code; (2) whether they can cause a `data_access_MMU_miss` exception; (3) the attributes of the block of memory prefetched: its size (minimum = 64 bytes) and its alignment (minimum = 64-byte alignment); (4) whether each variant is imple-

mented as a NOP, with its full semantics, or with common-case prefetching semantics; (5) whether and how variants 16..31 are implemented.

**Implementation:**

- (1) PREFETCH and PREFETCHA have identical affects in privileged or non-privileged
- (2) Can not cause a *data\_access\_MMU\_miss* exception
- (3) Size and alignments are 64-bytes
- (4),(5) See table-1

**Table 3: Prefetch Data**

fcn	V9 Prefetch Function	SPARC64-III Function
0	Prefetch for several reads	Prefetch for several reads
1	Prefetch for one read	Prefetch for several reads
2	Prefetch for several writes	Prefetch for several writes
3	Prefetch for one write	Prefetch for several writes
4	Prefetch page	Prefetch for several reads
5-15	Reserved	<i>illegal_instruction</i> trap
16-31	Implementation dependent	NOP

## 104. VER.manuf

**Description:**

VER.manuf contains a 16-bit semiconductor manufacturer code. This field is not present reads as zero. VER.manuf may indicate the original supplier of a second-sourced chip in cases involving mask-level second-sourcing. It is intended that the contents of VER.manuf track the JEDEC semiconductor manufacturer code as closely as possible. If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a VER.manuf value.

**Implementation:**

SPARC64-III uses a code of 4 for this field. This is Fujitsu's JEDEC code.

## 105. TICK register

### Description:

The difference between the values read from the TICK register on two reads should be the number of processor cycles executed between the reads. If an accurate count cannot always be returned, an inaccuracy should be small, bounded, and documented. An implementation may implement fewer than 63 bits in the TICK counter; however, the counter as implemented must be able to count for at least 10 years without overflowing. Any upper bits not implemented must be read as zero.

### Implementation:

SPARC64-III implements all the bits of TICK register and returns accurate count of processor cycles, in response to reads from TICK register.

## 106. IMPDEPn instructions

### Description:

The IMPDEP1 and IMPDEP2 instructions are completely implementation-dependent. Implementation-dependent aspects include their operation, the interpretation of bits 29:25 and 18:0 in their encoding, and which (if any) exceptions they may cause.

### Implementation:

SPARC64-III uses IMPDEP2 to encode the HaL specific Floating Point instructions. IMPDEP1 is not used and will cause an illegal\_instruction trap if such an opcode is encountered. Please refer to *SPARC64-III Processor User Guide* for more details.

## 107. Unimplemented LDD trap

### Description:

It is implementation-dependent whether LDD and LDDA are implemented in hardware. If not, an attempt to execute either will cause an unimplemented\_LDD trap.

### Implementation:

SPARC64-III implements LDD and LDDA in hardware.

## 108. Unimplemented STD trap

### Description:

It is implementation-dependent whether STD and STDA are implemented in hardware. If an attempt to execute either will cause an unimplemented\_STD trap.

**Implementation:**

SPARC64-III implements STD and STDA in hardware.

**109. LDDF\_mem\_address\_not\_aligned****Description:**

LDDF and LDDFA require only word alignment. However, if the effective address is aligned but not doubleword-aligned, either may cause an *LDDF\_mem\_address\_not\_aligned* trap, in which case the trap handler software shall emulate the LDDF (or LDDFA) instruction and return.

**Implementation:**

SPARC64-III causes *LDDF\_mem\_address\_not\_aligned* trap for both word and double-word misaligned addresses.

**110. STDF\_mem\_address\_not\_aligned****Description:**

STDF and STDFA require only word alignment. However, if the effective address is aligned but not doubleword-aligned, either may cause an *STDF\_mem\_address\_not\_aligned* trap, in which case the trap handler software shall emulate the STDF (or STDFA) instruction and return.

**Implementation:**

SPARC64-III causes *STDF\_mem\_address\_not\_aligned* trap for both word and double-word misaligned addresses.

**111. LDQF\_mem\_address\_not\_aligned****Description:**

LDQF and LDQFA require only word alignment. However, if the effective address is aligned but not quadword-aligned, either may cause an *LDQF\_mem\_address\_not\_aligned* trap, in which case the trap handler software shall emulate the LDQF (or LDQFA) instruction and return.

**Implementation:**

SPARC64-III generates an illegal instruction exception for LDQF, LDQFA instructions. kernel provides emulation routines to complete the load.

**112. STQF\_mem\_address\_not\_aligned****Description:**

STQF and STQFA require only word alignment. However, if the effective address is aligned

but not quadword-aligned, either may cause an STQF\_mem\_address\_not\_aligned trap, in which case the trap handler software shall emulate the STQF (or STQFA) instruction and return.

**Implementation:**

SPARC64-III generates an illegal instruction exception for STQF, STQFA instructions. The kernel provides emulation routines to complete the load.

**113. Implemented memory models****Description:**

Whether the Partial Store Order (PSO) or Relaxed Memory Order (RMO) models are reported is implementation-dependent.

**Implementation:**

SPARC64-III supports *Load/Store ordering (LSO)*, *Total store Ordering (TSO)* and *Store ordering (STO)*. *Partial Store Order (PSO)* is implemented using *TSO* and *Relaxed Memory Order (RMO)* is implemented using *STO*.

**114. RED\_state trap vector address (RSTVaddr)****Description:**

The RED\_state trap vector is located at an implementation-dependent address RSTVaddr.

**Implementation:**

RSTVaddr is a Constant when VA = FFFF FFFF F000 0000 and PA = 1FF F000 0000.

**115. RED\_state processor state****Description:**

What occurs after the processor enters RED\_state is implementation-dependent.

**Implementation:**

Please See *SPARC64-III user guide* pg.139 (7.2.1.2 RED\_state Execution Environment).

**116. SIR\_enable control flag****Description:**

The location of and the means of accessing the SIR\_enable control flag are dependent. In some implementations, it may be permanently zero.

**Implementation:**

SIR\_enable control flag is permanently zero in SPARC64-III.

## 117. MMU disabled prefetch behavior

### Description:

Whether Prefetch and Non-faulting Load always succeed when the MMU is disabled is implementation-dependent.

### Implementation:

In SPARC64-III, Prefetch and Non-faulting Loads always succeed if the MMU is

## 118. Identifying I/O locations

### Description:

The manner in which I/O locations are identified is implementation-dependent.

### Implementation:

Please contact HaL Computer Systems for details of I/O operation.

## 119. Unimplemented values for PSTATE.MM

### Description:

The effect of writing an unimplemented memory-mode designation into PSTATE.MM is implementation-dependent

### Implementation:

Writing '11' into PSTATE.MM causes the machine to use the STO Memoryever, the system software should not use the encoding '11' since it is reserved for future SPARC-V9 extensions.

## 120. Coherence and atomicity of memory operations

### Description:

The coherence and atomicity of memory operations between processors and I/O DMAory accesses are implementation-dependent.

### Implementation:

Plese See *SPARC64-III user guide* pg.355 (Nbr 121)

## 121. Implementation-dependent memory model

### Description:

An implementation may choose to identify certain addresses and use an independent memory model for references to them.

**Implementation:**

In SPARC64-III, certain addresses use implementation dependent memory models forences to them. Please contact HaL Computer Systems for details.

**122. FLUSH latency****Description:**

Latency between the execution of FLUSH on one processor and the point at which thefied instructions have replaced out-dated instructions in a multiprocessor is implementation-dependent.

**Implementation:**

Please contact HaL for FLUSH latency

**123. Input/output (I/O) semantics****Description:**

The semantic effect of accessing input/output (I/O) registers is

**Implementation:**

Please contact HaL for I/O semantics..

**124. Implicit ASI when TL>0****Description:**

When  $TL > 0$ , the implicit ASI for instruction fetches, loads, and stores isdependent. See SPARC-V9 Architecture Manual section F.4.4, "Contexts," for more information.

**Implementation:**

SPARC64-III uses *ASI\_NUCLEUS* for instruction fetches and *ASI\_NUCLEUS{ \_LITTLE }*, loads and stores when  $TL > 0$

**125. Address masking****Description:**

When *PSTATE.AM* = 1, the value of the high-order 32-bits of the PC transmitted tofied destination registers(s) by *CALL*, *JMPL*, *RDPC*, and on a trap is implementation-dependent.

**Implementation:**

When *PSTATE.AM* bit is set on SPARC64-III, a full 64-bit address is transmitted tofied destination registers by *CALL*, *JMPL*, *RDPC* and traps transmit all 64-bits to



TPC[n] and TNPC[n].

## 126. TSTATE bits 19:18

### Description:

If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

### Implementation:

SPARC64-III does not implement PSTATE bits 10 & 11 and they are read as zeroes. bits 19 and 18 are read as zeroes.

## 127. PSTATE bits 11:10

### Description:

The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are not defined. The presence of TSTATE bits 19 and 18 is implementation-dependent. If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.

### Implementation:

SPARC64-III does not implement PSTATE bits 10 & 11 and they are read as zeroes. bits 19 and 18 are read as zeroes.

## 128. CLEANWIN register update

Earlier implementations of Sparc chips implemented the V9 specification for using the following equation to update CLEANWIN register:

```
if (CLEANWIN != NWINDOWS) CLEANWIN++;
```

Subsequently V9 definition changed to modify the equation as:

```
if (CLEANWIN < NWINDOWS-1) CLEANWIN++;
```

SPARC64-III implements the *RESTORED* using the current definition. The SPARC64-III Kernel will ensure that *CLEANWIN* does not have a value beyond *NWINDOWS-1*.



**Appendix A: Assgined VER.manuf and VER.impl**

---

---

**V9**

**SPARC INTERNATIONAL**



## APPENDIX A: VER.impl/VER.manuf

The table 1 below includes all the V9 VER.impl and VER.manuf assigned by SPARC International as stated by the V9 Architecture Book (page 57). From Section 5.2.9: “*If the manufacturer does not have a JEDEC semiconductor manufacturer code, SPARC International will assign a value of VER.manuf*”.

To assign new number please contact:

Ghassan Abbas  
 abbas@sparc.com  
 Tel: 415-321-8692 x228.

**Table 4: assigned VER.impl and VER.manuf by SI**

COMPANY	CPU	VER.impl	VER.manuf
HAL	SPARC64	0x0001	0x0004
Sun Microsystems	UltraSPARC (TI)	0x0010	0x0017
Sun Microsystems	UltraSPARC (NEC)	0x0010	0x0022
Sun Microsystems	UltraSPARC II	0x0011	0x0017
Sun Microsystems	UltraSPARC Iii	0x0012	0x0017
Sun Microsystems	UltraSPARC-e	0x0013	0x0017



**Appendix B: V9 Architecture Errata**

---

---

**as of 17 Jul 1995**

---

---

**V9**

**SPARC INTERNATIONAL**





---

## APPENDIX B: SPARC V9 Arch Book Changes

*r141 = R1.4.1 = distrib draft*

*r142 = R1.4.2 = book first printing; doc dated 15 Sep 93*

*r143 = R1.4.3 = revision (not used);*

*r144 = R1.4.4 = current revision; doc dated 17 Jul 95*

*All changes below are those since R1.4.2, incorporated in R1.4.4.*

### Change to page 13

subsection 2.57:

definition of “reserved”: “...intended to run on future version of” was corrected to read: “...intended to run on future versions of”.

The sentence beginning “Reserved register fields” was amend to read: “Reserved register fields should always be written by software with values of those fields previously read from that register, or with zeroes; they should read as zero in hardware.”

### Change to page 21(r142)

Editor's Notes: Added Les Kohn's name to the Acknowledgments.

### Change to page 28(r142)

Tables 3,4,5: Made use of hyphens & dashes made consistent, and easier to read.

### Change to page 30(r142)

paragraph just above subsection 5.1: Changed end of sentence to read:

“...should be written with the values of those bits previously read from that register, or with zeroes.”

### Change to page 40(r142),

Table 7: Added lines for 32-bit and 64-bit signed integers in f.p. registers, for clarity.

**Change to page 51**

In figure 17, added bits 11 and 10 to the figure, so it looks like:

PID1	PID0	CLE	TLE	MM	RED	PEF	AM	PRIV	IE	AG
11	10	9	8	7 6	5	4	3	2	1	0

**Change to page 52(r142)**

inserted new subsection 5.2.1.1 before old one: “IMPL. DEP. #127: The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are implementation- dependent. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes. See also TSTATE bits 19 and 18.”

**Change to page 55(r142)**

In Figure 22, (TSTATE register): Extended the “saved PSTATE” field up through bit 19 of TSTATE; changed the diagram to look like:

TSTATE 1	CCR from TL=0	ASI from TL = 0	-	PSTATE from TL=0	-	CWP from TL = 0
TSTATE 2	CCR from TL=1	ASI from TL = 1	-	PSTATE from TL=1	-	CWP from TL = 1
TSTATE 3	CCR from TL=2	ASI from TL = 2	-	PSTATE from TL=2	-	CWP from TL = 2
TSTATE 4	CCR from TL=3	ASI from TL = 3	-	PSTATE from TL=3	-	CWP from TL = 3
	39 32	31 24	23 20	19 8	7 5	4 0

**Change to page 56(r142)**

Added a new paragraph to the end of subsection 5.2.6: “TSTATE bits 19 and 18 are implementation-dependent. ImplDep#126: If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level. If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as

zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.”

**Change to page 57(r142)**

subsection 5.2.10 (Register-Window State Registers): Added implementation dependency #126.

**Change to page 58-9(r142)**

In subsection 5.2.10 (Register-Window State Registers): Added note to descriptions of CWP, CANSAVE, CANRESTORE, OTHERWIN, and CLEANWIN registers that the effect of writing a value to them greater than NWINDOWS-1 is undefined.

**Change to page 76,**

In Section 6, last sentence in 6.3.4.1, “Conditional Branches” changed to: Note that the annul behavior of a taken conditional branch is different from that of an unconditional branch. And the last sentence in 6.3.4.2, “Unconditional Branches” changed to: Note that the annul behavior of a unconditional branch is different from that of a taken conditional branch.

**Change to page 80(r142), 6.3.6.4(r142)**

RESTORED: correct the equation with CLEANWIN to read “(CLEANWIN < (NWINDOWS-1))”. and correct the text above it.

**Change to page 81(r141/r142):**

In section 6.3.9, “FMOVc” was corrected to read “FMOVr”.

**Change to page 81(r141/r142):**

In section 6.3.9, a sentence was added stating that FSR.cexc and FSR.ftt are cleared by FMOVcc and FMOVr whether or not the move occurs.

**Change to page 121(r141/r142):**

An index entry for “non-faulting loads” was fixed in section 8.3.

**Change to page 151(r142), A.9(r142),**

In Compare and Swap page: Added mention of CASL and CASXL to the Programming Note.

**Change to page 171**

In Annex A, sentence added specifying that LDFSR does not affect the upper 32 bits of FSR.

**Change to page 181(r141/r142):**

“A.31” number was fixed so it now increments to A.32. All following section numbers and odd page headers in Annex A have changed.

**Change to page 191(r141/r142):**

Page heading: “Condition” --> “Condition”

**Change to page 195(r141/r142):**

Order of instructions in Suggested Assembly Language Syntax was rearranged to correspond to order of the instructions in the Opcode/op3/Operation table above it.

“more” and “movrz”, as the assembly-language mnemonic and its synonym, were exchanged to correspond with the instruction name of MOVZR.

“movrne” and “movrnz”, as the assembly-language mnemonic and its synonym, were exchanged to correspond with the instruction name of MOVRNZ.

**Change to page 212(r14[123]) A.43(r14[12])/A.44(r143),**

In second page of the Read State Register instruction description, 4th paragraph SHOULD read: “RDFPRS waits for all pending FPods \*\* and loads of floating-point registers\*\* to complete before reading the FPRS register.”

**Change to page 216(r142), A.46(r142),**

RESTORED page: correct the equation with CLEANWIN to read “(CLEANWIN < (NWINDOWS-1))”

**Change to page 220(r142)/A.49(r142)**

In the third Paragraph, the words “the” and “and” were transposed in the implementation dependency description. It now reads: “The location of the SIR\_enable control flag and the means of accessing the SIR\_enable control flag...”

**Change to page 228(r141/r142):**

Order of instructions in Suggested Assembly Language Syntax was rearranged to correspond to order of the instructions in the Opcode/op3/Operation table above it.

**Change to page 229(r142)/A.55(r142),**

paragraph beginning “Store integer...: load” changed to “store”

**Change to page 231(r142)/233(r143),**

In Annex A, corrected SWAP deprecation note to recommend use of “CASA” or “CASXA” (not “CASX”) in place of SWAP.

**Change to page 234, A.58(r14[12])/A.59(r143)**

Tagged Add: op3 opcodes are wrong. Both should have “0” for low-order bit (as correctly given in Appendix E).

**Change to page 241(r142), A.62(r142),**

In “Write State Register” page, added footnote to Suggested Assembly Language Syntax table, noting that the suggested syntax for WRASR with rd=16..31 may vary, citing reference to implementation dependency #48. (Suggested Assembly Language Syntax is just that -- \*suggested\* -- so isn't part of the architecture specification anyway, but this makes it clearer that if bits are interpreted differently in the instruction, one should expect its assembly-language syntax to change, as well)

**Change to page 242(r142), A.62(r142)**

In “Write State Register” page: In the Exceptions section, “WRASR with rs1=16..31” now reads “WRASR with rd=16..31”.

**Change to page 253(r142)**

In Annex C, fixed 6 incorrect index entries.

**Change to page 253(4142)**

In annex C, added a new Implementation Dependency:

Number	Category	Def/Ref page #	Description
127	f	52,56	<p>The presence and semantics of PSTATE.PID1 and PSTATE.PID0 are implementation-dependent. The presence of TSTATE bits 19 and 18 are implementation-dependent. If PSTATE bit 11 (10) is implemented, TSTATE bit 19 (18) shall be implemented and contain the state of PSTATE bit 11 (10) from the previous trap level.</p> <p>If PSTATE bit 11 (10) is not implemented, TSTATE bit 19 (18) shall read as zero. Software intended to run on multiple implementations should only write these bits to values previously read from PSTATE, or to zeroes.</p>

**Change to page 255(r142)**

In Annex C, added implementation dependency #126.

**Change to page 258(r142)**

In D.3.3., rule (1), the text was clarified, to read: “(1) The execution of Y is conditional on X, and S(Y) is true.”

**Change to page 268(r142)**

In table 32, as a privileged instruction, “RDPR” should be listed with a superscript “P”.

**Change to page 290(r142)**

In section G, Table 43: insert “#” before the “ASI” in the compare-and-swap synthetic instruction entries

**Change to page 312(r142)**

In Annex I, Missing word “not” added to Compatibility Note.







**Index**

---

---

**V9**

**SPARC INTERNATIONAL**



*Index*

- Symbols
- , LDQ 36
- Numerics
- 16bit implementation 25
- A
- Address space identifier 27
- aex 26
- aexc 95
- AG 162
- AM 162
- and 90
- ANSI/IEEE 24
- ANSI/IEEE Standard 754-1985 44, 91
- AS\_IF 28
- ASI 27, 28, 39, 96, 109, 162, 167
- ASI\_AS\_IF\_USER\_PRIMARY 27, 96
- ASI\_AS\_IF\_USER\_PRIMARY\_LITTLE 27, 96
- ASI\_AS\_IF\_USER\_SECONDARY 28, 96
- ASI\_AS\_IF\_USER\_SECONDARY\_LITTLE 28, 96
- ASI\_NUCLEUS 28, 96
- ASI\_NUCLEUS\_LITTLE 28, 96
- ASI\_PRIMARY 27, 39, 96, 109
- ASI\_PRIMARY\_LITTLE 27, 39, 96
- ASI\_PRIMARY\_NOFAULT 27, 96
- ASI\_PRIMARY\_NOFAULT\_LITTLE 28, 96
- ASI\_SECONDARY 28, 96
- ASI\_SECONDARY\_LITTLE 28, 96
- ASI\_SECONDARY\_NOFAULT 28, 96
- ASI\_SECONDARY\_NOFAULT\_LITTLE 28, 96
- ASR 25, 32, 93, 100, 101
- ASR24 25
- ASR25 25
- ASR26 25
- ASR27 25
- ASR28 25
- ASR29 25
- ASR31 25
- ASRs 92, 93, 100
- Assembly Language Syntax 165
- associated deferred-trap queues 29
- async\_data\_error 30
- audience 19
- B
- bge 90
- bne 90
- C
- CALL 39, 109
- CANRESTORE 163
- CANSAVE 163
- CASA 165
- CASX 165
- CASXA 165
- ccelerated emulation trap 31
- CCR 162
- cex 95
- cexc 26, 95
- check\_illegal\_done\_retry 90
- check\_illegal\_saved\_restored 90
- Chip\_crossing\_errors 29
- circular stack 24
- CLE 162
- clean\_window 101
- cleaning 33
- CLEANWIN 40
- CLEANWIN register 40
- CLEAR\_SOFTINT 93, 101
- contents of SCD 2.2 19
- corrected\_ECC\_error 98
- CPU 25, 28, 29, 31, 37, 157
- CPU\_HALTED 31
- CPU\_xing 29, 30
- D
- d LDD 35
- Data access 100
- data\_access\_exception 99
- data\_access\_MMU\_miss 30, 33, 100, 102
- data\_breakpoint 30
- data\_breakpoint 27, 29
- deferred 30
- Deferred trap queues 29
- deferred-trap queue 27
- definition of audience 19
- definition of purpose 19
- DISPATCH\_CONTROL\_REG 93

- 
- disrupting 30
  - division\_by\_zero 29, 97
  - DMA 38, 108
  - DONE 90
  - doubleword-aligned 36
  - E
  - ECC\_trap 30
  - error 97
  - Error logging 97
  - error state 31
  - Error\_state 31, 99
  - error\_state 31, 99, 107
  - F
  - F{i,x}TOq 43, 89
  - F{s,d} 91
  - F{s,d}TOq 43, 89
  - FABSq 43, 89
  - FADD 91
  - FADDq 43, 89
  - fast\_data\_access\_MMU\_miss 98
  - fast\_data\_access\_protection 98
  - fast\_instruction\_access\_MMU\_miss 98
  - Fault Address Register 25
  - FCMP{E}q 43, 89
  - FDIV 27, 44, 91, 92
  - FDIVq 90
  - FdMULq 43, 89
  - FdTOs 91
  - fetches 109
  - FFF0 94
  - FFFF 94
  - Floating-Point 24
  - floating-point 32, 91
  - Floating-point underflow 33
  - FLUSH 31, 32, 38, 99, 108
  - flush 31
  - FMOVc 163
  - FMOVcc 163
  - FMOVq 43, 89
  - FMOVqcc 43, 89
  - FMOVqr 43, 89
  - FMOVr 163
  - FMUL 91, 92
  - FMULq 89
  - FNEGq 43, 89
  - FNULq 43
  - FP 44, 89
  - fp\_exception\_othe 36
  - fp\_exception\_other 23, 27, 29, 36, 95
  - FPQ 27
  - FPU 24, 26, 27, 94, 95, 100
  - FPU TEM 95
  - FPU trap 32
  - FQ 27, 95
  - FqTO 43
  - FqTO{i,x} 89
  - FqTO{s,d} 89
  - FSQRT 91
  - fsqrtd 23
  - FSQRTq 44, 90
  - fsqrts 23
  - FSR 26, 27, 94
  - FSR.cexc 163
  - FSR.ft 23
  - FSR.ftt 23, 27, 29, 95, 163
  - FSR.NS 94
  - FSR.VER 94
  - FSR.ver 26, 94
  - FSR\_nonstandard\_fp 26, 94
  - FSUBq 43, 89
  - Fujitsu 34
  - G
  - GRAPHICS\_STATUS\_REG 93
  - H
  - HAL 25, 31, 35, 38, 157
  - HAL Computer Systems 23, 38, 67, 137
  - hex 102
  - I
  - I/O register 92
  - I/O registers 25, 92
  - IE 162
  - IEEE 94
  - IEEE 754 24, 33, 101
  - IEEE 754-1985 26
  - IEEE Standard 754-1985 26, 94
  - IEEE Std 754-1985 24, 91, 95
  - IEEE Std. 754-1985 27
  - illegal\_instructio 32
  - illegal\_instruction 23, 27, 31, 32, 34, 95, 99
  - IMPDEP1 35, 103
  - IMPDEP2 35, 103
  - IMPL. DEP 162
-

- impl. dep 96
- implementation-dependent 108
- Index 19
- instruction\_access\_error 97
- instruction\_access\_MMU\_miss 30
- internal\_processor\_error 30
- Interrupt 97
- interrupt 98
- interrupt\_vector 98
- Introduction 19
- IO\_parity 30
- IU 24, 26, 90, 94
- IU registers 24
- J
- JEDEC 34, 102, 157
- JMPL 39, 109
- K
- Kernel 23
- kernel 24
- L
- Latency 38
- ld 90
- LDD 23, 25, 31, 105
- LDDA 35, 105
- ldda 23
- LDDF 35, 105
- LDDF\_mem\_address\_not\_aligned 35, 105
- LDDFA 35, 105
- LDQF 36, 43, 89, 106
- LDQF\_mem\_address\_not\_aligned 30, 36, 106
- LDQFA 36, 43, 89, 106
- LE 28
- Load/Store ordering 36
- loads 109
- LSO 36
- M
- M 38
- mask-level 34
- MAXTL 97, 99
- mem\_address\_not\_aligned 29, 97
- MM 37, 162
- MMU 25, 28, 37, 107
- multiprocessor 38
- N
- no-fault-only 99
- non-cacheable 92, 109
- Non-faulting 37
- Non-faulting Load 107
- non-faulting loads 163
- non-privileged mode 28
- Non-Restricted 28
- nonstandard mode 24
- NOP 33, 34, 102, 107
- nop 90
- not\_illegal 90
- NR 28
- NS 26, 94
- NWINDOW 40
- NWINDOWS 24, 90, 163
- NWINDOWS-1 40, 163
- O
- Opcode 165
- organization 19
- OTHERWIN 163
- overflowing 34
- P
- PA\_watchpoint 98
- Partial Store Order 36, 106
- PC 109
- PEF 162
- PERF\_CONTROL\_REG 93
- PERF\_COUNTER 93
- PID0 162
- PID1 162
- PIL 29
- PO 28
- POPC 43, 89
- popc 23
- POR 31
- Power-on Reset 31
- precise 30
- Preface 19
- PREFETCH 33, 102
- Prefetch 34, 37, 107
- PREFETCHA 33, 102
- prgrammed\_emulation\_trap 30
- PRIV 162
- privileged status 25
- privileged\_action 28
- privileged\_opcode 90
- processor cycles 34
- Program Order 28

- 
- PSO 36, 106
  - PSTATE 39, 40, 109, 110, 162, 163, 166
  - PSTATE.AM 39, 109
  - PSTATE.IE 29, 97
  - PSTATE.MM 38, 108
  - PSTATE.PID 39
  - PSTATE.PID0 110, 162
  - PSTATE.PID1 39, 110, 162
  - PSTATE.PRIV 28, 96
  - PSTATE.RED 97
  - purpose 19
  - Q
  - quad operands 23
  - quadword-aligned 36
  - R
  - r LDD 35
  - r Relaxed Memory Order 36
  - RDASR 32, 100
  - RDPC 39, 109
  - RDPR 27, 95, 167
  - rdpr 90
  - RED 162
  - RED\_alert 30
  - RED\_MODE 37
  - RED\_state 36, 37, 106, 107
  - Relaxed Memory Order 36, 106
  - Reset trap 98
  - reset trap 98
  - RESTORED 40, 90, 163
  - RETRY 90
  - RIVILEGED\_OPCODE\_HANDLER 90
  - RMO 36, 106, 108
  - rounding 33
  - RSTVaddr 37, 106, 107
  - S
  - SAVED 90
  - second-sourced chip 34
  - sequence\_error 27
  - SET\_SOFTIN 101
  - SET\_SOFTINT 93
  - setx 90
  - SIR 25
  - SIR\_enable 37, 107, 165
  - SOFTINT\_REG 93, 101
  - Software 25
  - Software Installation 19
  - SP 91, 92
  - SPARC 89, 90
  - SPARC International 23, 34, 67, 89, 102, 137, 157
  - Sparc6 28
  - SPARC64 25, 27, 28, 30, 31, 35, 38, 157
  - Sparc64 23, 24, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 67, 137
  - Sparc64 Processor User Guide 35
  - SPARC-V9 26, 28, 30, 38, 39, 40, 97, 98, 109
  - Sparc-V9 23, 28
  - srl 90
  - State Control Register 25
  - STD 23, 25, 31, 35, 105
  - std 23
  - STDA 35, 105
  - stda 23
  - STDF 36, 105, 106
  - STDF\_mem\_address\_not\_aligned 36, 105, 106
  - STDFA 36, 105, 106
  - STO 36
  - store 109
  - Store ordering 36
  - STP 1030BGA 43
  - STQD\_mem\_address\_not\_aligned 30
  - STQF 36, 43, 89, 106
  - STQF\_mem\_address\_not\_aligned 36, 106
  - STQFA 36, 43, 89, 106
  - subcc 90
  - SUN 89, 113
  - Sun Microsystems 157
  - Sun Microsystems, Inc 89
  - SWAP 165
  - T
  - TEM 26, 27, 95
  - TICK 34, 102
  - TICK.counter 102
  - TICK\_CMPR\_REG 93
  - tininess 24, 33
  - TL 39, 99, 109, 162
  - TLE 162
  - TNPC 39
  - TPC 29, 39
  - Trap precision 97
  - Trapped Underflow 24
  - TSTAT 162
-

TSTATE 39, 40, 109, 110, 162  
TT 30, 98  
U  
UFM=0 24  
UFM=1 24  
UltraSPARC 157  
ULTRASPARC II 89, 113  
UltraSPARC II 157  
Ultra-SPARC-I 43  
UltraSPARC-I 43  
UltraSPARC-II 89, 91, 92, 93, 94, 95, 96, 97, 98,  
99, 100, 101, 102, 103, 105, 106, 107,  
108, 109, 110  
unfinished\_FPop 27, 29  
unimplemented exception 24  
unimplemented\_FPop 23  
unimplemented\_LDD 23, 25, 35, 105  
unimplemented\_STD 23, 25, 35, 105  
Untrapped Underflow 24  
V  
V (Vendor-specific) 28  
VA\_watchpoint 98  
Vendor-specific 28  
VER.impl 26, 93, 157  
VER.manuf 34, 102, 157  
W  
watchdog 31  
Watchdog reset 31  
watchdog\_reset 99  
WDR 31  
Window State Registers 163  
WRASR 25, 32, 100, 101, 165, 166  
Write State Register 165, 166  
write-only register 25  
X  
XIR 37  
xor 32, 100