1.0 Implementation Characteristice of IEEE 1754 based products

IEEE

1754

SPARC International

FINAL RELEASE

2/26/99

© 1990-1999 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.

The manual pages for socket functions are
© 1992, 1993 The Regents of the University of California. All rights reserved

Includes material copyrighted by UNIX System Laboratories, Inc., a subsidiary of SCO, Inc. Reprinted with permission.

The SPARC Compliance Definition 2.4 is published and printed by SPARC International.

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 (Ext 228)

FAX: (408) 743-9777

URL: www.sparc.org

ATTN: Ghassan Abbas (abbas@sparc.org)

Trademarks

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

SPARCstation™ is a trademark of SPARC International, Inc.

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

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

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

UNIX® and OPEN LOOK® are registered trademarks of UNIX System Laboratories, Inc.

The X-Window System™ is a trademark of Massachusetts Institute of Technology.

OSF/Motif™ is a trademark of the TOG (X/Open + Open Software Foundation, 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.

This product contains intellectual property of Sun Microsystems, Inc., and any user of this product will be required to obtain a license from Sun Microsystems, Inc., prior to use.

Table of Contents

Introduction . . . . . **i**

Chapter 1 : 1754 Architecture Implementation Dependencies 1

1.1 Definition of an Implementation Dependency 1

1.2 Hardware Dependencies 2

1.3 Categories of Implementation Dependencies 2

1.4 List of Implementation Dependencies 3

Chapter 2 : Dependencies Questionnaire 14

2.1 Dependencies Questionnaire 15

2.2 Hardware Dependencies 15

2.3 Implementation Dependencies 15

Chapter 3 : Current Product Information 32

Chapter 4 : Current PSR Assignments 34

Introduction

1. Introduction

This document is maintained by SPARC International and describes the implementation-dependent design features of 1754-compliant implementations. The document is structured to match Annex E of the 1754 standard which was written knowing that such an document would be available. Annex E is reproduced in this document as Chapter 1. System and software designers using this document must read it as a complement to the 1754 standard which can be obtained from IEEE.....

Designers of 1754-compliant processor must contact SPARC International to register their design, must furnish the information contained in the Implementation Dependencies Questionnaire (Chapter 2), and participate in the Compliance and Certification Program run by SPARC International. SPARC International will also help them with the Assigned Value information which is part of the standard (see section 1.3).

The information provided in Chapter 3, Current Product Information, was furnished by each vendor and should reflect the implementation dependent features of each of the described products. Although SPARC International will make every effort to collect accurate information only the specific vendors can be contacted when a problem arise from the accuracy of the information..

Chapter 1: 1754 Architecture Implementation Dependencies

This chapter provides a convenient summary list of all the architectural implementation dependencies of the 1754 standard. It is a copy of the normative Annex E of the standard. The notation "***Impl. Dep. #nn***" is used to identify the definition of an architectural implementation dependency. The notation "(impl. dep. #***nn***)" is used to identify a minor reference to an architectural implementation dependency. The number ***nn*** provides a unique index to an architectural implementation dependency definition in this chapter.

1.1. Definition of an Implementation Dependency

The 1754 architecture is a model which specifies unambiguously the behavior observed by software on 1754 systems. Therefore, it does not necessarily describe the operation of the hardware in any actual implementation.

An implementation is **not** required to execute every instruction in hardware. An attempt to execute a 1754 instruction that is not implemented in hardware generates a trap. If the unimplemented instruction is nonprivileged, then it must be possible to emulate it in software. If it is a privileged instruction, whether it is emulated by software is implementation-dependent (impl. dep. #1).

Compliance with this specification shall be claimed only by a collection of components which is capable of fully implementing all 1754 opcodes, through any combination of hardware or software. Specifically, nonprivileged instructions which are not implemented in hardware shall trap to the software such that they can be implemented in software. For the implementation to be complete, by default the implementation shall trap and report all undefined and reserved instructions.

Some elements of the architecture are defined to be implementation-dependent. These elements include certain registers and operations that may vary from implementation to implementation, and are explicitly identified as such in this chapter.

Implementation elements (such as instructions or registers) that

appear in an implementation but are not defined in this document (or its updates) are not considered to be 1754 elements of that implementation.

1.2. Hardware Dependencies

Hardware dependencies which do not affect the behavior observed by software on 1754 systems are not considered architectural implementation dependencies. A hardware dependency may be relevant to the user system design (e.g., the speed of execution of an instruction) or may be fairly transparent to the user (e.g, the method used for achieving cache consistency). The SPARC International document, “Implementation Characteristics of Current 1754-based Products, Revision 1.x”, provides a useful list of these hardware dependencies, along with the list of implementation-dependent design features of 1754-compliant implementations.

In general, hardware dependencies deal with:

- the speed of execution of instructions
- the fact that instructions are implemented in hardware or not
- the degree of concurrence of the various hardware units comprising 1754.

1.3. Categories of Implementation Dependencies

To understand many of the implementation dependencies, one can group them into five categories represented with their first letter abbreviation in the rest of this chapter:

- Value (v)
  The semantic of the architectural feature is well-defined, except that a value associated with it is different across implementations. A typical example is the number of implemented register windows (impl. dep. #2).
- Assigned Value (a)
  Similar to Value, except that a body is appointed to assign values for this dependency (specifically, the body is SPARC International, 535 Middlefield Rd., Suite 210, Menlo Park, CA 94025). Typical ex-

amples are the version fields associated with the IU (impl. dep. #13), Floating Point (impl. dep. #19), or MMU units (impl. dep. #59).

- Functional choice (f)
  The 1754 architecture allows the implementation to choose between several semantic treatments related to an architectural function. A typical example is the treatment of “catastrophic error” exception, which may cause either a deferred or a disrupting trap (impl. dep. #31).
- Total Unit (t)
  The existence of the architectural unit or function is recognized, but details are totally left to each implementation. The obvious example of this is the coprocessor unit (impl. dep. #4). More complex cases include the handling of I/O registers (impl. dep. #7) and alternate address spaces (impl. dep. #29).
- Compatibility and Miscellaneous issues (c)
  The architecture has evolved, and the standard recognizes the existence of past variants without encouraging them to be used in the future. One example concerns the decoding of the 8 bits of the ASI (impl. dep. #30) or the writing of the Y register by a WRASR instruction (impl. dep. #50).

1.4. List of Implementation Dependencies

This section provides a complete list of all the implementation dependencies of the architecture, the definition of each, and a reference to the section number in the standard where they are defined. Most implementation dependencies occur because of the coprocessor unit, the address spaces, I/O registers, registers (including ASR’s), the type of trapping used for an exception, the handling of errors, and miscellaneous non-1754-architectural units such as the MMU or caches (which affect the FLUSH instruction).

Impl Dep#	Cate-gory	Defining Section#	Description
1	f	1.5	**Software emulation of instructions** An implementation is not required to execute every instruction An implementation is not required to execute every instruction in hardware. An attempt to execute a 1754 instruction that is not implemented in hardware generates a trap. If the unimplemented instruction is nonprivileged, then it must be emulated by software. If it is a privileged instruction, whether it is emulated by software is implementation-dependent.
2	v	3.1.1	**Number of IU registers** An implementation of the IU may contain from 40 to 520 general-purpose 32-bit r registers. This corresponds to a grouping of the registers into 8 global r registers, plus a circular stack of from 2 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 implementation dependent.
3	f	3.1.2	**Incorrect ANSI/IEEE 754-1985 results** An implementation can 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. Software must emulate any functionality not present in the hardware.
4	t	3.1.3, 5.2.1.9	**Coprocessor** An implementation can 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. Software must emulate any functionality not present in the hardware
5	t	3.2.1.3	**Load/Store alternate target registers** The privileged load/store alternate instructions can be used by supervisor software to access special protected registers, such as MMU, cache control, and processor state registers, and other processor- or system-dependent values.

Impl Dep#	Cate-gory	Defining Section#	Description
6	f	3.2.2	**I/O registers privileged status** If normal load/store instructions, coprocessor instructions, or read/write Ancillary State Register instructions are used, whether the I/O registers can be accessed outside of supervisor code or not is implementation-dependent.
7	t	3.2.2	**I/O registers definitions** The contents and addresses of I/O registers are implementation-dependent.
8	t	3.2.5	**RDASR/WRASR target registers** There are read/write "ancillary state register" instructions that software can use to read/write unique implementation-dependent processor registers (ASR's 16-31).
9	f	3.2.5, B.9.24, B.9.25	**RDASR/WRASR privileged status** Whether each of the implementation-dependent read/write ancillary state register instructions (for ASR's 16-31) is privileged or not is implementation-dependent.
10	f	5.1.4	**Misaligned IU register numbers** An attempt to execute a doubleword load or store instruction that refers to a misaligned (odd) destination register number should cause an illegal_instruction trap.
11	f	5.1.4, A.7, A.8	**Trap on Doubleword Load** During execution of a load doubleword integer or coprocessor (LDD, LDDA, or LDDC) instruction, if an exception is generated during the memory cycle in which the second word is being loaded, the destination registers(s) may be modified before the trap is taken. A similar exception can occur during the store memory access but not the load access of a LDSTUB, LDSTUBA, SWAP, or SWAPA instruction. Thus, the destination register may be modified before the trap is taken.

Impl Dep#	Cate-gory	Defining Section#	Description
12	f	5.1.4	**Trap on Doubleword Store** An implementation might cause a data_access_exception trap due to a "catastrophic" error during the "effective address + 4" memory access of a store-doubleword (STD, STDA, STDF, STDFQ, STDC, or STDCQ) instruction, even though the corresponding "effective address" access did not cause an error. Thus, memory data at the effective memory address may be modified before the trap is taken.
13	a	5.2.1.1	**PSR.*impl*** Bits 31 through 24 of the PSR are hardwired to uniquely identify an implementation or class of software-compatible implementations of the architecture.
14	t	5.2.1.8, 5.2.7, 7.5.2, A.36	**Enable Extensions Bit** Whether PSR.EE is implemented or not is implementation-dependent. If it is not implemented, PSR.EE reads as 0 and writes to it are ignored. The existence of the Implementation-dependent Extensions register (IER) and the meaning of specific values contained in it are implementation-dependent. The function of the WRIER instruction is implementation-dependent: if the IER register exists, WRIER writes it; if the IER register does not exist, WRIER produces undefined results, but an illegal_instruction trap should be taken. Whether and to what value the enable extensions bit (PSR.EE) is written by a non-reset trap is implementation-dependent.
15	f	5.2.1.10	**PSR.EF when no FPU is implemented** If an implementation does not support a hardware FPU, PSR.EF should always read as 0 and writes to it should be ignored.
16	t	5.2.8	**IU deferred-trap queue** The contents and operation of an IU deferred-trap queue are implementation-dependent and are not visible to user application programs.

Impl Dep#	Cate-gory	Defining Section#	Description
17	f	5.3.1	**Misaligned FPU register numbers** An attempt to execute an instruction that refers to a misaligned floating-point register operand (double-precision operand in a register whose number is not 0 mod 2, or quadruple-precision operand in a register whose number is not 0 mod 4) should cause an fp_exception trap, with FSR.*ftt* = 6 (invalid_fp_register).
18	f	5.4.1.4	**Non standard ANSI/IEEE 754-1985 results** 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 ANSI/IEEE Standard 754-1985. 1754 implementations are permitted but not encouraged to deviate from 754 requirements when the nonstandard mode bit of the FSR is 1. In an implementation where the nonstandard floating-point mode operates identically to standard 1754 mode, the NS bit of the FSR always reads as 0, even after a 1 is written to it.
19	a	5.4.1.6	**FPU version, FSR.*ver*** Bits 19 through 17 of the FSR, FSR_version, identify one or more particular implementations of the FPU architecture.
20	v	5.4.1.7	**Zeroing of FSR.*ftt*** Supervisor-mode software which handles floating-point traps must execute an STFSR to determine the floating-point trap type. Whether STFSR explicitly zeroes FSR.*ftt* is implementation-dependent.
21	f	5.4.1.11	**FSR.*cexc* contents upon trap** An IEEE_754_exception which traps should cause exactly one bit in FSR.*cexc* to be set, corresponding to the detected IEEE 754 exception.
22	f	5.4.3	**FPU TEM, *cexc*, and *aexc*** An implementation may choose to implement the TEM, *cexc*, and *aexc* fields in hardware in either of two ways (see section 5.4.3 for details).
23	f	5.4.4	**Floating Point Traps** Floating-point traps may be precise or deferred. If deferred, a floating-point deferred trap queue (FQ) must be present.

Impl Dep#	Cate-gory	Defining Section#	Description
24	t	5.4.4	**FPU deferred-trap queue** The contents of and operations upon the floating-point deferred-trap queue FQ are implementation-dependent.
25	f	5.4.4	**STDFQ with Empty FQ** On an implementation with a floating-point queue, an attempt to execute STDFQ when the FQ is empty (FSR.*qne* = 0) should cause an fp_exception trap with FSR.*ftt* set to 4 (sequence_error).
26	t	5.5	**Coprocessor Registers** All of the coprocessor data and control/status registers are optional and implementation-dependent.
27	t	5.5	**Coprocessor State Register** The architecture provides instruction support for reading and writing a Coprocessor State Register (CSR).
28	t	5.5	**Coprocessor Traps** Coprocessor traps may be precise or deferred. If deferred, a coprocessor deferred-trap queue (CQ) must be present.
29	t	6.3.1.3	**Alternate Space Identifier (ASI) definitions** The definitions of most alternate spaces are implementation-dependent.
30	c	6.3.1.3	**ASI address decoding** Whether an implementation decodes all eight ASI bits is implementation-dependent.
31	f	7, 7.2	**Catastrophic error exceptions** A "**catastrophic error**" exception is due to the detection of a hardware malfunction from which, due to its nature, the state of the machine at the time of the exception cannot be restored. Since the machine state cannot be restored, continued execution after such an exception may not be resumable. Catastrophic error exceptions are implementation-dependent. They may cause precise, deferred or disrupting traps.
32	t	7.1.3	**Deferred traps** Whether any deferred traps (and associated deferred-trap queues) are present is implementation-dependent.

Impl Dep#	Cate-gory	Defining Section#	Description
33	f	7.2	**Trap precision** Exceptions that occur as the result of program execution may be precise or deferred, although it is recommended that such exceptions be precise. Examples: mem_address_not_aligned, division_by_zero.
34	f	7.3.1	**Interrupt clearing** The method by which an interrupt request is removed is implementation-dependent.
35	t	7.4.1	**Special traps** Trap Type (*tt*) values 0x60 to 0x7F are reserved for implementation-dependent exceptions.
36	f	7.4.2	**Trap priorities** Note that particular traps have relative priorities and are implementation-dependent, because a future version of the architecture may define new traps, and implementations can define implementation-dependent traps which establish new relative priorities.
37	f	7.5.1	**Reset trap** A reset trap sets PSR.S ← 1, PSR.ET ←0, and PSR.EE ← 0. A reset trap should set PC ←0, nPC ← 4, and FSR.*qne* ← 0. It should not alter other processor state.
38	f	7.5.1	**Effect of reset trap on implementation-dependent registers** Implementation-dependent registers may or may not be affected by reset.
39	f	7.5.3	**Entering error_mode processor state ‘** The processor enters error_mode state when a precise trap occurs while ET = 0, or when an implementation-dependent error condition occurs. When error_mode is entered, PSR.ET (←0 and PSR.S (←1; any other processor state changes are implementation-dependent.
40	f	7.5.3	**Error_mode processor state** What occurs after error_mode is entered is implementation-dependent. Typically, the processor triggers an external reset, causing a reset trap.

Impl Dep#	Cate-gory	Defining Section#	Description
41	f	7.6,	**MMU miss** Whether MMU-miss exceptions (data_access_MMU_miss, instruction_access_MMU_miss) cause precise, deferred, or disrupting traps is implementation-dependent.
42	t	7.6	**Implementation-Dependent Traps** The existence of implementation_dependent_*n* traps, and whether any implemented ones are precise, deferred, or disrupting is entirely implementation-dependent.
43	t,f,v	7.6, A.39, B.9.28	**FLUSH instruction** Four aspects of the FLUSH instruction are implementation-dependent: (1) whether FLUSH generates no trap, an uniplemented_FLUSH trap, or an illegal_instruction trap; (2) the definition of the `flush_cache_line()` ISP macro; (3) the definition of the `flush_Ibuf_and_pipeline()` ISP macro; and (4) the number of instructions (0 to 5) which must execute after a FLUSH before its effect is complete.
44	t	7.6	**Watchpoint** Whether a 1754 processor generates watchpoint_detected exceptions is implementation-dependent.
45	f	A.2	**Data access FPU trap** If a load floating-point instruction traps with a data access exception, the destination *f* register(s) either remain unchanged or are set to an implementation-dependent predetermined constant value.
46	t,f	A.5	**STDFQ** The store double floating-point deferred-trap queue instruction (STDFQ) stores the front entry of the Floating-point Queue (FQ) into memory. An attempt to execute STDFQ on an implementation without a floating-point queue causes an fp_exception trap with FSR.*ftt* set to 4 (sequence_error). On an implementation with a floating-point queue, an attempt to execute STDFQ when the FQ is empty (FSR.*qne* = 0) should cause an fp_exception trap with FSR.*ftt* set to 4 (sequence_error). Any additional semantics of this instruction are implementation-dependent.

Impl Dep#	Cate-gory	Defining Section#	Description
47	f	A.20	**Divide overflow** Which of the two Divide overflow-detection conditions is used is implementation-dependent.
48	t	A.35	**RDASR** 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 or not (impl. dep. #9), and whether the instruction causes an illegal_instruction trap or not.
49	t	A.36	**WRASR** WRASR instructions with *rd* in the range 16...31 are available for implementation-dependent uses (impl. dep. #8). For a WRASR instruction *rd* in the range 16...31, the following are all 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 or not (impl. dep. #9), and whether the instruction causes an illegal_instruction trap or not.
50	c	A.36	**Y register** In some existing implementations, WRASR and WRIER instructions may write the Y register. WRIER and WRASR in new implementations shall not write the Y register.
51	v	A.36	**Write state register delay** The write state register instructions are **delayed-write** instructions. That is, they may take until completion of the third instruction following the write instruction to consummate their write operation. The number of delay instructions (0 to 3) is implementation-dependent.
52	f	A.36	**WRPSR** In some implementations, if a WRPSR instruction updates the PSR's PIL field to a new value and simultaneously sets ET to 1, an interrupt trap at a level equal to the old value of the PIL may result.

Impl Dep#	Cate-gory	Defining Section#	Description
53	f	A.41	**Coprocessor Instruction Fields** The interpretation of the *rd, rs1, opc,* and *rs2* fields in CPop instructions is coprocessor-dependent.
54	t	A.41	**Coprocessor Data** The data types supported by a coprocessor are coprocessor-dependent. Operand alignment within the coprocessor is coprocessor-dependent.
55	t	A.41	**Coprocessor Exception trap** The conditions under which execution of a CPop instruction causes a cp_exception trap are coprocessor-dependent.
56	f	D.3	**Floating-point underflow detection** During floating-point underflow detection, whether (in IEEE 754 terms) “tininess” is detected before or after rounding is implementation-dependent. It is recommended that tininess be detected before rounding.
57	v	F.3.1	**MMU context table** The size of the Reference MMU context table is implementation-dependent.
58	t	F.3.4.2	**MMU probe operations** The presence of page, segment, region, and context probe operations is implementation-dependent; that is, an implementation may or may not provide these probe operations. If a probe operation is not implemented, the value it returns is undefined.
59	a	F.4.2	**MMU CR.IMPL and CR.VER** The Reference MMU Control Register IMPL field identifies the specific implementation of the MMU. The Reference MMU Control Register VER field identifies a particular version of this MMU implementation (and is typically a mask number). Both are hardwired into the implementation and is read-only.

Impl Dep#	Cate-gory	Defining Section#	Description
60	f	F.4.2	**MMU CR.SC** The Reference MMU Control Register System Control (SC) field bits are implementation-dependent. They may be reflected in a variable number of signals external to the MMU and need not all be implemented. If a bit is not implemented, it reads as zero and writes to it are ignored.
61	t	F.4.5	**MMU diagnostics** A 1754 Reference MMU may provide access to diagnostic registers through an alternate address space See Annex G, “Suggested ASI Assignments for 1754 Systems.” If present, their operation is implementation-dependent.
62	v	F.5	**MMU Fault Status Register EBE** The Reference MMU Fault Status Register External Bus Error (EBE) field bits are set when a system error occurs during a memory access. The meanings of the individual bits are implementation-dependent. Examples of system errors are: time-out, uncorrectable error, and parity error. The MMU need not implement all the bits in **EBE**. Unimplemented bits read as zeros.

Chapter 2: Dependencies Questionnaire

2.1. Dependencies Questionnaire

Designers of 1754-based products must furnish to SPARC International the information contained in this Chapter to help in maintaining an accurate list of all the implementation-dependent design features of 1754-compliant implementations.

2.2. Hardware Dependencies

This section provides information about:

- the speed of execution of instructions,
- the fact that instructions are implemented in hardware or not,
- the degree of concurrence of the various hardware units comprising 1754.

2.3. Implementation Dependencies

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
1	f	1.5	**Software emulation of instructions**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
2	v	3.1.1	**Number of IU registers**
3	f	3.1.2	**Incorrect ANSI/IEEE 754-1985 results**
4	t	3.1.3, 5.2.1.9	**Coprocessor**
5	t	3.2.1.3	**Load/Store alternate target registers**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
6	f	3.2.2	**I/O registers privileged status**
7	t	3.2.2	**I/O registers definitions**
8	t	3.2.5	**RDASR/WRASR target registers**
9	f	3.2.5, B.9.24, B.9.25	**RDASR/WRASR privileged status**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
10	f	5.1.4	**Misaligned IU register numbers**
11	f	5.1.4, A.7, A.8	**Trap on Doubleword Load**
12	f	5.1.4	**Trap on Doubleword Store**
13	a	5.2.1.1	**PSR.*impl***

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
14	t	5.2.1.8, 5.2.7, 7.5.2, A.36	**Enable Extensions Bit**
15	f	5.2.1.10	**PSR.EF when no FPU is implemented**
16	t	5.2.8	**IU deferred-trap queue**
17	f	5.3.1	**Misaligned FPU register numbers**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
18	f	5.4.1.4	**Non standard ANSI/IEEE 754-1985 results**
19	a	5.4.1.6	**FPU version, FSR.*ver***
20	v	5.4.1.7	**Zeroing of FSR.*ftt***
21	f	5.4.1.11	**FSR.*cexc* contents upon trap**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
22	f	5.4.3	**FPU TEM, *cexc*, and *aexc***
23	f	5.4.4	**Floating Point Traps**
24	t	5.4.4	**FPU deferred-trap queue**
25	f	5.4.4	**STDFQ with Empty FQ**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
26	t	5.5	**Coprocessor Registers**
27	t	5.5	**Coprocessor State Register**
28	t	5.5	**Coprocessor Traps**
29	t	6.3.1.3	**Alternate Space Identifier (ASI) definitions**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
30	c	6.3.1.3	**ASI address decoding**
31	f	7, 7.2	**Catastrophic error exceptions**
32	t	7.1.3	**Deferred traps**
33	f	7.2	**Trap precision**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
34	f	7.3.1	**Interrupt clearing**
35	t	7.4.1	**Special traps**
36	f	7.4.2	**Trap priorities**
37	f	7.5.1	**Reset trap**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
38	f	7.5.1	**Effect of reset trap on implementation-dependent regis-ters**
39	f	7.5.3	**Entering error_mode processor state**
40	f	7.5.3	**Error_mode processor state**
41	f	7.6,	**MMU miss**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
42	t	7.6	**Implementation-Dependent Traps**
43	t,f,v	7.6, A.39, B.9.28	**FLUSH instruction**
44	t	7.6	**Watchpoint**
45	f	A.2	**Data access FPU trap**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
46	t,f	A.5	**STDFQ**
47	f	A.20	**Divide overflow**
48	t	A.35	**RDASR**
49	t	A.36	**WRASR**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
50	c	A.36	**Y register**
51	v	A.36	**Write state register delay**
52	f	A.36	**WRPSR**
53	f	A.41	**Coprocessor Instruction Fields**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
54	t	A.41	**Coprocessor Data**
55	t	A.41	**Coprocessor Exception trap**
56	f	D.3	**Floating-point underflow detection**
57	v	F.3.1	**MMU context table**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
58	t	F.3.4.2	**MMU probe operations**
59	a	F.4.2	**MMU CR.IMPL and CR.VER**
60	f	F.4.2	**MMU CR.SC**
61	t	F.4.5	**MMU diagnostics**

Impl Dep#	Cate-gory	Defining Section#	Description of your implementation
62	v	F.5	**MMU Fault Status Register EBE**

Chapter 3: Current Product Information

Appendix A: Current PSR Assignments

A. The following are the existing PSR implementation number assignments, as of February 26, 1999:

PSR Impl	Device	Implementor
0x00	MB86900 MB86901 MB86902 L64801	Fujitsu Microelectronics, Inc.
0x02	MB86930	Fujitsu Microelectronics, Inc.
0x10, 0x11	CY7C601 L64811	Cypress Semiconductor, Corp & Ross Technology, Inc.
0x13	CY7C611	Cypress Semiconductor, Corp & Ross Technology, Inc.
0x1F		Cypress Semiconductor, Corp & Ross Technology, Inc.
0x20	B5010	Bipolar Integrated Technology
0x21		Matsushita Semiconductor, Inc.
0x30		LSI Logic, Corp
0x40, 0x41		Texas Instruments
0x50	MN10501	Matsushita Semiconductor, Inc. & Solbourne Computer, Inc.
0x60		Philips, Corp
0x70		Harvest VLSI Design Center, Inc.
0x80		System and Processes Engineering, Corp

NOTE: This list may be incomplete, since some development projects are not disclosed yet. Please contact SPARC International for more Information.