

---

## Checking AT697E Code for Occurrence of LDF/FPOPd Instructions Sequence with a dependency on an Odd-Numbered Register

This application note provides AT697E users with a description of the procedure to check their code for potential occurrence of a specific instructions sequence: a load singleword floating-point instruction (LDF) involving an odd-numbered floating-point register as a destination of the load and an immediately following double-precision floating-point instruction (FADDd, FSUBd, FMULd, FDIVd or FSQRTd) that satisfies all of the following conditions:

- the odd floating-point register is used as (part of) a source operand
- the destination floating-point register is also a source operand
- in an FSUBd or FDIVd, the two source operands are different registers

### References

- AT697E Errata-Sheet
- The SPARC® Architecture Manual Version 8 (<http://www.sparc.org/standards/V8.pdf>)



---

AT697E

---

## Application Note

Rev. 7787A-AERO-05/08



## Context

As described in the item 13 of the AT697E Errata-Sheet, data dependency is not properly checked between a floating-point load singleword instruction (LDF) and an immediately following double precision floating-point instruction.

Unless direct control over assembly language is possible to fix the code, there is a possibility that high-level language compilers may generate one or more of such a SPARC instructions sequence without the user ever noticing it.

Due to the variety of high-level language compilers available for the SPARC architecture, and because this procedure must apply not only to applications currently in development, but also to applications already in the field, code checking for this specific SPARC instructions sequence occurrence shall be done on a binary byte stream representation of the AT697E application, as found in the files used for programming an (E/E2)PROM in the final application.

## Checking Software

The software provided with this application note (“CHECK\_LDF\_FPopD\_FOR\_ODD\_FPReg\_DEPENDENCY.c”) will check any big-endian <sup>(1)</sup> binary byte file given as the sole program argument.

The program will first display a banner:

```

+=====+
| This software checks an AT697E binary file for occurrence of a specific instructions |
| sequence: a load singleword floating-point instruction (LDF) involving an odd-numbered |
| floating-point register as a destination of the load and an immediately following |
| double-precision floating-point instruction (FADDd, FSUBd, FMULd, FDIVd or FSQRTd) |
| that satisfies all of the following conditions: |
| - the odd floating-point register is used as (part of) a source operand |
| - the destination floating-point register is also a source operand |
| - in an FSUBd or FDIVd, the two source operands are different registers |
+=====+

```

Then it will display its check specification, which is the list of instructions sequences to be searched for. This list will show each instruction symbolic name, opcode detection radix and associated detection mask. In our case:

Check specification:  
 =====

```

opcode : name          ->      radix/mask

- inst #1: LDF(immediate) -> 0xC1002000/0xC1F82000
           LDF(indexed)   -> 0xC1000000/0xC1F80000

- inst #2: FADDd       -> 0x81A00840/0xC1F83FE0
           FMULd       -> 0x81A00940/0xC1F83FE0
           FSQRTd      -> 0x81A00540/0xC1F83FE0
           FSUBd       -> 0x81A008C0/0xC1F83FE0
           FDIVd       -> 0x81A009C0/0xC1F83FE0

```

1. Big-Endianness is a representation where bytes in a word are stored with most-significant bytes in the lower addresses and least-significant bytes in the higher addresses.

Then the program will parse the binary byte file. Whenever an instructions sequence matching the specification is found, a table line will be displayed showing:

- First instruction and second instruction symbolic name
- Destination register number (Rd) used by the first instruction
- Source(s) and Destination registers number (Rs1 / Rs2 / Rd) used by the second instruction
- Start byte location of the sequence
- Opcode composing the sequence in hexadecimal format (first and second instruction)

Now processing file: "SPARC/SCAN\_LDF\_FPopD\_FOR\_ODD\_FPReg\_DEPENDENCY-AT697E.bin"

OCCURRENCE FOUND:

```

=====
+-----+
|      ASM      | |1st Instruction|  2nd Instruction  |      Sequence      | | Instructions | | |
|              | |-----+-----+ |      Start Byte   | |              |
| SEQUENCE     | |   Rd   |   Rs1 |   Rs2 |   Rd   |      Location     | |   Opcode     |
+-----+-----+-----+-----+-----+-----+-----+
| LDF(indexed) | |   %f9  |       |       |       |      0x00001c28   | | 0xD300C000   |
| FADDd        | |       |   %f8 |   %f8 |   %f8 |                   | | 0x91A20848   |
+-----+-----+-----+-----+-----+-----+
| LDF(indexed) | |   %f9  |       |       |       |      0x000020bc   | | 0xD300C000   |
| FADDd        | |       |   %f8 |   %f10|   %f8 |                   | | 0x91A2084A   |
+-----+-----+-----+-----+-----+-----+

```

(following output not shown here)

By the end of the check, a message will show the total amount of SPARC instruction sequence occurrences found:

Found 19 potential sequences occurrence in file "SPARC/SCAN\_LDF\_FPopD\_FOR\_ODD\_FPReg\_DEPENDENCY-AT697E.bin".

ALTHOUGH OCCURRENCES WERE FOUND, PLEASE CONSIDER THAT TECHNIQUES SUCH AS CODE COMPRESSION AND/OR SELF-MODIFYING CODE MAY PREVENT PROPER DETECTION OF ANY DOUBLE FLOATING INSTRUCTION FOLLOWING A FLOATING INSTRUCTIONS SEQUENCE WITH A DEPENDENCY BETWEEN THE DESTINATION REGISTER (Rd) OF THE FIRST AND THE SOURCE ODD-NUMBERED REGISTER (Rs1, Rs2, Rd) OF THE SECOND.

PLEASE REVIEW EACH OCCURRENCE FOR APPROPRIATE CLASSIFICATION.

If no occurrence was found, then another message will be displayed:

Found 0 potential sequence occurrence in file "SPARC/SCAN\_LDF\_FPopD\_FOR\_ODD\_FPReg\_DEPENDENCY-AT697E.bin".

ALTHOUGH NO OCCURRENCES WERE FOUND, PLEASE CONSIDER THAT TECHNIQUES SUCH AS CODE COMPRESSION AND/OR SELF-MODIFYING CODE MAY PREVENT PROPER DETECTION OF ANY DOUBLE FLOATING INSTRUCTION FOLLOWING A FLOATING INSTRUCTIONS SEQUENCE WITH A DEPENDENCY BETWEEN THE DESTINATION REGISTER (Rd) OF THE FIRST AND THE SOURCE ODD-NUMBERED REGISTER (Rs1, Rs2, Rd) OF THE SECOND.

## Platforms

This checking software has been successfully tested on different platforms:

- RED HAT® ENTERPRISE Linux® 3 with BCC-1.0.30 cross-compiling environment
- CYGWIN™ under Windows® 2000/XP with BCC-1.0.30 cross-compiling environment
- MSYS under Windows 2000/XP with BCC-1.0.30 cross-compiling environment

## Limitations

On one hand, this checking software can only detect explicit instruction sequences, as specified in the AT697E Errata-Sheet. Techniques such as code compression and/or self-modifying code will prevent any detection during binary byte stream analysis.

On the other hand, this checking software may detect byte sequences matching instructions sequences specified in the AT697E Errata-Sheet, although these byte sequences are pure program data.

In any case, this checking software shall only be considered as an application support tool, the final user being responsible for appropriate classification of each detected occurrence.



## Headquarters

---

**Atmel Corporation**  
2325 Orchard Parkway  
San Jose, CA 95131  
USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 487-2600

## International

---

**Atmel Asia**  
Room 1219  
Chinachem Golden Plaza  
77 Mody Road Tsimshatsui  
East Kowloon  
Hong Kong  
Tel: (852) 2721-9778  
Fax: (852) 2722-1369

**Atmel Europe**  
Le Krebs  
8, Rue Jean-Pierre Timbaud  
BP 309  
78054 Saint-Quentin-en-  
Yvelines Cedex  
France  
Tel: (33) 1-30-60-70-00  
Fax: (33) 1-30-60-71-11

**Atmel Japan**  
9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
Tel: (81) 3-3523-3551  
Fax: (81) 3-3523-7581

## Product Contact

---

**Web Site**  
[www.atmel.com](http://www.atmel.com)

**Technical Support**  
[aerospace@nto.atmel.com](mailto:aerospace@nto.atmel.com)

**Sales Contact**  
[www.atmel.com/contacts](http://www.atmel.com/contacts)

**Literature Requests**  
[www.atmel.com/literature](http://www.atmel.com/literature)

---

**Disclaimer:** The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2008 Atmel Corporation. All rights reserved. Atmel® logo and combinations thereof, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Windows® and others are registered trademarks of Microsoft Corporation in the US and/or other countries. Other terms and product names may be trademarks of others.