GR712RC Incorrect Annulation of Floating-point Operation on Instruction Cache Parity Error

---

Technical note 2017-12-21

Doc. No GRLIB-TN-0012

Issue 1.3

CHANGE RECORD

Issue	Date	Section / Page	Description
1.2	2017-11-17	All	First public release
1.3	2017-12-21	Section 3.2	Overview of compiler workaround and list of toolchains with the workaround available.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

This document describes a corner case present in the GR712RC floating-point pipeline where an annulled floating-point instruction, with a tag or data parity error in instruction cache combined with an integer branch instruction resolved late in the pipeline, is executed and committed even though it should have been annulled.

1.2 Affected components

Cobham and Aeroflex components that are **affected** are:

- GR712RC

Cobham and Aeroflex components that are **unaffected** are:

- GR740 – LEON4FT is unaffected
- LEON3FT-RTAX
- UT699
- UT699E
- UT700

No GRLIB version is affected from this bug.

1.3 Distribution

GR712RC users are free to use the material in this document in their own documents and to redistribute this document. Please contact Cobham Gaisler for inquires on other use.

1.4 Contact

For questions on this document, please contact Cobham Gaisler support at support@gaisler.com. When requesting support include the part name if the question is a specific device or the full GRLIB IP library package name if the question relates to a GRLIB IP library license.

2 TECHNICAL DESCRIPTION

2.1 Overview

The processor behaviour described in this document can cause an incorrect update of a floating-point register or floating-point condition register due to execution and commit of an extra floating point instruction outside of the correct instruction flow. This can in turn cause erroneous behaviour of software.

Two different instruction sequences are vulnerable to this issue. They are described in the subsections below.

2.2 Sequence A

1. Any instruction that modifies the integer unit condition codes.
2. Branch instruction with annul bit = '1' that depends on integer condition code
3. The delay slot of the branch instruction is a floating point instruction

Notes:

- All the instructions in the above sequence must be back-to-back in the instruction flow without any intervening instructions.
- Instructions in 1. include:
  - ANDcc, ANDNcc
  - ORcc, ORNcc
  - XORcc, XNORcc
  - ADDcc, ADDXcc, TADDcc
  - SUBcc, SUBXcc, TSUBcc
  - MULScc, UMULcc, SMULcc
  - UDIVcc, SDIVcc
  - Pseudo instructions:
    - cmp
    - inccc
    - deccc
    - tst
    - btst
- Instruction in 2.include
  - Bicc,a (see SPARC architecture manual V8 section B.21)
- Instruction in 3. is the delay slot of 2 and include
  - FPop (see SPARC architecture manual V8 section B.33)

Example:

```
cmp %l0,%l1  (subcc %l0,%l1,%g0)
bne,a L1
fmuld %f10,%f10,%f16
```

In order for the erroneous behaviour to be encountered, the two following conditions must met:

1. Branch is not taken.
2. Floating point operation or branch with floating point condition at the delay slot of 1. has a tag or data parity error in instruction cache (can be caused by radiation induced single-event upset or inserted by force using the error injection diagnostic interface) by the time it is fetched from the instruction cache.

2.3 Sequence B

1. Any instruction that modifies the integer unit condition codes.
2. Branch instruction that depends on integer condition code
3. Any instruction in the delay slot of the branch.
4. At the branch target address, a floating point operate instruction or a branch with floating point condition code.

Notes:

- Instructions 1.2.3 in the above sequence must be back-to-back in the instruction flow without any intervening instructions. Instruction 4 is the branch target address of instruction 2 which is executed after instruction 3.
- Instruction in 1. include:
  - same as Sequence A.
- Instructions in 2. include
  - Bicc (see SPARC architecture manual V8 section B.21)
  - Bicc,a (see SPARC architecture manual V8 section B.21)
- Instruction in 3. is the delay slot of 2.
- Instruction in 4. include:
  - FPop (see SPARC architecture manual V8 section B.33)
  - FBfcc (see sparc architecture manual V8 section B.22)

Example:

```
        cmp %l0,%l1 (subcc %l0,%l1,%g0)
        be L1
        nop
        …
        …

L1:
        fmuld %f10,%f10,%f16
```

Another example:

```
        cmp %l0,%l1 (subcc %l0,%l1,%g0)
        be L1
        nop
        …
        …

L1:
        fbe L2
```

In order for the erroneous behaviour to be encountered, the two following conditions must met:

1. Branch is not taken.
2. Floating point operation or branch with floating point condition at the branch target address of 1. should have a tag or data parity error in instruction cache (can be caused by radiation induced single-event upset or inserted by force using the error injection diagnostic interface) by the time it is fetched from the instruction cache.

2.3.1 Effect

When erroneous behaviour is encountered an extra floating point operation will be executed and committed which must not happen in a correct execution flow. What floating-point operation is performed and which register is affected is not predictable. Whether this creates a software error depends on if the modified register and/or condition code is subsequently used.

It should be noted that the two sequences (A,B) can be combined and two consecutively annulled floating point operations can be executed wrongly.

Example:

In this example, if the branch is not taken and both of the floating point instructions encounter a parity error in the instruction cache while they are fetched then both of them will be executed and incorrectly committed.

```
        cmp %l0,%l1
        bne,a L1
        fmuld %f10,%f10,%f16
        ..
        ..

L1:
        fadd %f12,%f14,%f8
```

3 WORKAROUND / MITIGATION

3.1 Workaround

Erroneously executed floating point operations can be avoided by avoiding sequences A and B.

Sequence A can be avoided by avoiding floating point operations in delay slots of conditional integer branches with the annul bit set.

Sequence B can be avoided by moving all floating point operations and floating point branches that resides on branch targets by inserting a nop instruction on those branch targets.

3.2 Toolchain versions with workaround

Errata workarounds are available for BCC, RCC, and VxWorks. The workarounds include compiler fixes as well as patched assembly code for the system libraries. The following release versions and later will include the workarounds in toolchain and OS/libraries:

- BCC 1.0.50
- BCC 2.0.2
- RCC 1.2.22
- RCC 1.3-RC3
- VxWorks 6.7 1.0.21 – Toolchain 1.0.15
- VxWorks 6.9 2.0.8 – Toolchain 1.0.4
- VxWorks 7 – Toolchain 7.2.0.0
- MKPROM 2.0.63

The compiler workaround will prevent any floating-point operation from being placed in the delay slot of conditional integer branches with the annul bit set. It will also place a NOP instruction at the branch target of an integer branch if it is a floating-point operation or a floating-point branch. The workaround is thus in accordance with the method described in the previous section.

Note that the compiler workaround will not make any modifications to inline assembly code in the user application. Any sensitive assembly instruction sequence will have to be modified manually.

The compiler workaround is activated by compiling with the "-mfix-gr712rc" flag.

The patches for the compiler workaround are listed below. They are included in the official GCC distribution for version 7.3.0 and later.

[SPARC] Errata workaround for GRLIB-TN-0012
https://gcc.gnu.org/viewcvs/gcc?view=revision&revision=255234
* config/sparc/sparc.c (sparc_do_work_around_errata): Use mem_ref
https://gcc.gnu.org/viewcvs/gcc?view=revision&revision=255393
SPARC: Make sure that jump is to a label in errata workaround
https://gcc.gnu.org/viewcvs/gcc?view=revision&revision=255807

3.3 Scan script

A scan script is provided to allow scans of disassembled programs for code that can trigger the issue described in this document. The script is available from:

http://www.gaisler.com/notes

Please see the header in the script file for usage instructions.

4 FAQ

4.1 Are floating-point loads and stores affected?

No, loads and stores are not affected and can be placed in delay slots and branch targets.