AEROSPACE DATA EXCHANGE PROGRAM TRANSMITTAL

PROBLEM ADVISORY

CAES

1. TITLE	2. DOCUMENT NUMBER
UT699 LEON 3FT, GRFPU FLOATING POINT CONTROLLER STORE FORWARDING ERROR	SPO-2014-PA-0004
	3. DATE *(Year, Month, Date)* 2014, May, 28
4. MANUFACTURER NAME AND ADDRESS CAES 4350 CENTENNIAL BOULEVARD COLORADO SPRINGS, COLORADO 80907-3486	5. MANUFACTURER POINT OF CONTACT NAME Gwen Butler
	6. MANUFACTURER POINT OF CONTACT TELEPHONE (719) 594-8466
	7. MANUFACTURER POINT OF CONTACT EMAIL Gwen.Butler@cobhamaes.com

8. CAGE CODE	9. LDC START	10. LDC END	11. PRODUCT IDENTIFICATION CODE	12. BASE PART
65342	All	All	WG07A	UT699
13. BLANK			14. SMD NUMBER 5962-08228	15. DEVICE TYPE DESIGNATOR 01 & 02
			16. RHA LEVELS R & F	17. QML LEVEL Q & V
			18. NON QML LEVEL Proto & HiRel	19. BLANK

20. PROBLEM DESCRIPTION / DISCUSSION / EFFECT

A bug within GRLIB IP (rev. 1.0.22-b3680 and previous) residing in the UT699 (5962-08228) microprocessor causes a store forwarding error under certain specific conditions when the GRFPU high performance floating-point unit is enabled.

The error occurs when the program running on the microprocessor executes instructions that complete both of two patterns simultaneously. The appended erratum describes the two patterns in detail. Under this circumstance, the GRFPC floating-point controller can store out incorrect data leading to software error.

21. ACTION TAKEN / PLANNED

1. Created an errata to describe the workaround and mitigation methods to handle the error. (Complete - UT699 LEON 3FT GRFPU Floating-Point Controller (GRFPC) Erratum – appended to this Product Advisory)
2. Add a compiler switch to ensure patterns never appear in compiler output. (Complete - Reference Errata Workarounds in section 5.0 of the appended errata.)
3. The bug will be corrected in the next LEON3FT (Generic P.N. UT699E) that is currently under development at CAES. Prototypes are currently available and the device is only offered in the 484 ceramic land grid array package, and column grid array package.

22. DISPOSITIONARY RECOMMENDATION:	CHECK & USE AS IS ☐	CONTACT MANUFACTURER ☐	REMOVE & REPLACE ☐	CORRECT & USE AS SPECIFIED ☒

23. ADEPT REPRESENTATIVE	24. SIGNATURE	25. DATE
Timothy L. Meade		**May 28, 2014**

UT699 LEON 3FT GRFPU Floating-point controller (GRFPC) erratum: Store forwarding error after single-precision loads to adjacent registers

Table 1: Cross Reference of Applicable Products

Product Name:	Manufacturer Part Number	SMD #	Device Type	Internal PIC* Number:
LEON3FT	UT699	5962-08228	All	WG07A

*PIC: Product Identification Code

1.0 Overview

This document describes a design erratum in the GRFPC floating-point controller: the hardware that interfaces the GRFPU floating-point unit with the LEON3FT processor. For complete details regarding the impact of the erratum, which versions are affected, and possible workarounds see Appendix A.

2.0 Affected Products

This erratum applies to LEON3/LEON3FT-based devices using GRLIB revisions earlier than b3680, where the GRFPU high performance floating-point unit is enabled in the processor.

For further information on how to determine if your product is affected see section 2 in Appendix A.

3.0 Description

Under certain specific conditions preceding a single-precision floating-point store, the GRFPC can behave incorrectly, causing the FPC to store out incorrect data. For a detailed description of the triggering conditions, see section 4 in Appendix A.

4.0 Implications

On systems where the erratum is applicable and not using any of the workarounds described in section 5 of Appendix A of this document, the impact is that certain single-precision floating-point stores can store incorrect data, leading to software error.

5.0 Workarounds

See section 5 of Appendix A.

6.0 Summary/Conclusion and Possible Corrective Action

An error in the GRFPC floating-point controller, when triggered by a special case, can cause the GRFPC to store out incorrect data. Applying the workarounds in section 5 of Appendix A will avoid the problem, in most cases without performance or size penalty.

Appendix A

GRFPU Floating-point controller (GRFPC) errata: Store forwarding error after single-precision loads to adjacent registers

1 OVERVIEW

This document describes a design errata in certain versions of the GRFPC floating point controller, which is the hardware used to interface the GRFPU floating point unit with the LEON3 and LEON3FT processors. The impact of the error, which versions are affected, and possible workarounds are described.

For further information, contact Aeroflex Gaisler support: support@gaisler.com

2 AFFECTED PRODUCTS

2.1 General

This errata applies to LEON3/LEON3FT-based devices made using GRLIB revisions earlier than revision b3680, and where the GRFPU high performance floating-point unit is enabled in the processor. There are also additional limitations on which IP configurations are affected.

If the below description is not clear enough to determine if you are affected by the errata, contact Aeroflex Gaisler support.

2.2 Aeroflex components

Aeroflex components affected are:

- UT699

Aeroflex components **NOT** affected are:

- UT699E - NOT affected, made from a newer revision of GRLIB
- UT700 - NOT affected, made from a newer revision of GRLIB
- GR712 - NOT affected, made from a newer revision of GRLIB
- LEON3FT-RTAX - NOT affected, none of these configurations use the GRFPU
- LEON4-N2X - NOT affected, made from a newer revision of GRLIB

2.3 How to check if LEON3/LEON3FT design is affected

If you are licensing GRLIB for use in your own FPGA or ASIC design, you can check the following conditions in the design's VHDL source to see if the erratum applies to your system:

1. Check the GRLIB revision. This can be seen in the file name of the downloaded release package, in the directory name after unpacking the release, and in the file lib/grlib/stdlib/version.vhd in the release file tree (constant grlib_build). If this is higher than 3680, you are NOT affected by this error. Otherwise, continue with the following step.
2. Check if you are using the GRFPU by looking at the value of the fpu generic passed into the LEON3/LEON3FT instantiation. If this has values 1-7 or 17-23, you may have the error. If the fpu generic has values 0, 8-16, 24-31, you are NOT affected by this error.
3. If your design matches step 1 and 2 above, you may be affected by the bug depending on the value of the tech generic and any custom changes done to the technology mapping layer. Please contact Aeroflex Gaisler for more information

If source for the design is not available, you can instead find out this information from inside the design:

1. The build ID can be found at memory address 0xfffffff0, in the top half word (bits 31:16). This is also read out and reported by GRMON when connecting to the system (except when the device ID matches a known device, in that case the device name is printed instead).
2. The LEON3:s %asr17 register has an FPU field at bits 11:10 which are set to "01" if the GRFPU is used in the system.

Instead of using the methods above, a small program can be used to checks if the system it is running on is affected by this errata. See section 6 for the full listing or contact Aeroflex Gaisler Support for an electronic copy.

3 IMPACT

On systems where the errata is applicable and not using any of the workarounds described in this document, the impact is that certain single-precision floating-point stores can store incorrect data, leading to software error.

With compiler workarounds enabled, existing source code can be recompiled unchanged to avoid the errata.

Assembler code will need to be reviewed for triggering code sequences and may then need minor modifications to avoid the error. In the majority of cases this can be done without performance or size penalty.

4 ERRATA DESCRIPTION

4.1 Cause

The error is caused by logic inside the GRFPC floating point controller designed to avoid simultaneous read and write to the floating point register file. This logic can in a special case, described below, behave incorrectly causing incorrect data to be stored out from the FPC.

4.2 Trigger sequence

The errata triggers only on single-precision floating point stores satisfying certain criteria. Stores not satisfying these critera will be unaffected.

The errata depends on both the current instructions before the store that are in the execution pipeline, and which last two floating-point operations (real operations that enter the FPU, not loads and stores) that went into the pipeline before the store. This can be expressed as two different execution "patterns" that must be satisfied at the same time in order for the bug to trigger. In order to avoid triggering the bug, the code should be arranged so at least one of the two patterns is avoided.

Pattern 1:

1. single-precision load or single-precision FPOP to register %fX, where X is the same register as the store being checked
2. single-precision load or single-precision FPOP to register %fY , where Y is the opposite register in the same double-precision pair.
3. 0-3 instructions of any kind, except stores from %fX or %fY or operations with %fX as destination.
4. The store (from register %fX) being considered

Pattern 2:

1. double-precision FPOP
2. Any number of operations on any kind, except no double-precision FPOP and at most one (less than two) single-precision or single-to-double FPOPs
3. The store (from register %fX) being considered

For the purposes of this specific errata and the patterns above, the instructions are classed as follows:

single-precision load:	ld to FP register
single-precision store:	st from FP register
double-precision load:	ldd to (even) FP register
double-precision store:	std from (even) FP register
single-precision FPOP:	fadds, fsubs, fmuls, fdivs, fsqrts, fcmps, fcmpes, fmovs, fnegs, fabss, fitos, fstoi, fitod, fdtoi, fdtos
single-to-double FPOP:	fsmuld, fstod, fitod
double-precision FPOP:	faddd, fsubd, fmuld, fdivd, fsqrtd, fcmpd, fcmped

4.3 Examples

The following instruction sequence example will trigger the bug, note that the operations performed are double precision;

```
ld [%i0], %f0
ld [%i1], %f1
faddd %f4, %f6, %f8
faddd %f8, %f9, %f10
st %f0, [%i0]
```

The following instruction sequence will also trigger the bug:

```
faddd %f4, %f6, %f8
faddd %f8, %f9, %f10
ld [%i0], %f0
ld [%i1], %f1
st %f0, [%i0]
```

This sequence will NOT trigger the bug, because both the two previous Fpops are single-precision:

```
fadds %f4, %f6, %f8
fadds %f8, %f9, %f10
ld [%i0], %f0
ld [%i1], %f1
st %f0, [%i0]
```

This sequence will also NOT trigger the bug, because double precision store is used

```
ld [%i0], %f0
ld [%i1], %f1
std %f0, [%i2]
```

5 SOLUTIONS

5.1 Assembler-level workarounds

Either of these workarounds can be used to avoid the problem, only one of them needs to be used:

- For potentially dangerous sequences, swap registers so the two adjacent loads are done to non-neighbouring registers.
- Before an instruction sequence that might trigger this issue, or at the beginning of a function doing only single-precision computation, insert two dummy single-precision FPOPs.
- Keep the whole application in either double-precision or single-precision. Note that for C programs, using only single-precision is complicated by the fact that standard library routines such as printf convert to double-precision internally.
- When a dangerous sequence is found, insert NOPs between the loads and the store so the number of instructions between are 4 or more.

5.2 Compiler workarounds

The Aeroflex Gaisler provided GCC tool-chains, from versions shown below, have integrated a workaround for this bug into the existing -mtune=ut699 switch.

BCC (bare-C): BCC release 1.0.45 and newer (both GCC 3.4.4 and 4.4.2 based toolchains)

RCC (RTEMS): RCC release 1.2.15 and newer

VxWorks: GCC-4.1.2 based toolchain release 1.0.12 and newer

5.3 Scan scripts

A scan script can be provided to search binary files for dangerous sequences. The script is written in TCL and takes as input disassembly output from running objdump -d on the binary or on individual object files. Contact Aeroflex Gaisler support to obtain a copy.

6 TEST PROGRAM

The following test program will check for the FPU errata and report if it is found. It should be compiled with the BCC compiler and run with GRMON.

```
#include <stdio.h>

static inline int chkfpu(void)
{
        unsigned long tmp;
        asm volatile ("  mov %%asr17, %0\n" : "=r"(tmp));
        return ((tmp >> 10) & 3);
}

static inline int testfunc(void)
{
        unsigned long long buf[2];
        unsigned long tmp;
        buf[0] = 0x3ff0000000000000LL;
        buf[1] = 0x1234567800000000LL;
        asm volatile ("  ldd [%1],%%f6\n"
                      "  faddd %%f6,%%f6,%%f8\n"
                      "  faddd %%f6,%%f6,%%f8\n"
                      "  ld [%1+8],%%f2\n"
                      "  ld [%1+8],%%f3\n"
                      "  st %%f2,[%1+8]\n"
                      "  ld [%1+8], %0\n"
                : "=r"(tmp) : "r"(buf) : "f6","f7","f2","f3","memory");
        return (tmp != 0x12345678);
}

int main(int argc, char **argv)
{
        int i;
        if (chkfpu() == 0) {
                puts("No FPU in system - errata not applicable");
                return 0;
        }
        for (i=0; i<2; i++) {
                if (testfunc()) {
                        puts("Errata detected!");
                        return 1;
                }
        }
        puts("Errata not detected!");
        return 0;
}
```