GRFPU Floating-point controller (GRFPC): Clarifications on FSR register behavior

---

Technical note

2019-02-28

Doc. No GRLIB-TN-0016

Issue 1.0

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2019-02-28		First issue of document.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

This document clarifies certain corner case behaviors of GRFPU floating point unit when used with the GRFPC floating point controller in LEON3 and LEON4 based systems. The note is intended for application writers writing assembler code targeting the GRFPU.

1.2 Distribution

LEON3, LEON3FT, LEON4 and LEON4FT 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 distribution.

1.2.1 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 AFFECTED PRODUCTS

2.1 General

This document applies to all LEON3, LEON3FT, LEON4, and LEON4FT systems using the GRFPU floating-point unit and the GRFPC floating-point controller.

2.2 Cobham components

This document is applicable to any component containing the LEON3 or LEON4 together with the GRFPU, including the following components:

- UT699
- UT699E
- UT700
- GR712
- GR740

Cobham components **NOT** affected are:

- LEON3FT-RTAX NOT affected, none of these configurations use the GRFPU
- GR716 Does not use the GRFPU

2.3 How to check if a custom design uses GRFPU

In the VHDL top-level code of the design, you can check if GRFPU is used by looking at the value of the fpu generic passed into the LEON3/LEON3FT instantiation. If this has values in the ranges 1-7 or 17-23, the design uses the GRFPU and this document applies.

On a running system, this can be checked by reading out the processor's %asr17 register. The FPU field at bits 11:10 which are set to "01" if the GRFPU is used in the system.

3 IMPACT

Users writing custom floating-point code in assembly language may observe the behaviors described in this note if using the store fsr (stfsr) instruction in their code. Customers using compiled code with existing OS:es and toolchains are not impacted by the issues discussed in this document.

4 NOTES ON GRFPU/GRFPC FSR BEHAVIOR

4.1 STFSR behavior immediately after floating-point operation

If the store FSR instruction is immediately following after a floating-point operation instruction in a running program, with no other instruction in between the two, the stfsr might behave as if the order was reversed between the two instructions and the stfsr occurred before the floating-point operation. This impacts the exception (cexc/aexc) fields written out, and also means that the stfsr will not catch the fp_exception trap in case the last floating-point operation causes an exception. The exception is in that case left pending and will instead cause a trap on the next floating-point instruction.

This only affects FPop1 type instructions (floating-point operations producing results into a floating-point register such as fadds, fdivd, fmovs, fitos, etc) immediately followed by stfsr. FPop2 type instructions (floating-point compares), loads and stores, as well as any other (non-FP) instruction can be immediately followed by stfsr without triggering this reordering behavior.

To avoid this behavior, at least one unrelated instruction needs to be between the Fpop and the STFSR instruction. For compiled code, STFSR would normally be done inside a called library function and the function call and return surrounding the STFSR will prevent this sequence from occurring.

4.2 Clearing of FTT field in FSR after STFSR operation

After a fp_exception trap, when doing a STFSR operation to read out the FSR register, the GRFPC will automatically clear the %fsr.ftt field. However, this is done with a delay of up to three operations. This means if the trap handler does STFSR twice in a row both written values may have a non-zero FTT field.

As there is no reason to perform two stfsr operations that closely spaced in real applications, this is only expected to be an issue for test code.

4.3 Current exception field updating from DIV/SQRT

After each floating point operation completes in the FPU, the current exception field in the FSR register (fsr.cexc) is updated to reflect the exception status of that operation. Because the GRFPU allows operations to both start and complete in parallel to the slower operations FDIVS, FDIVD, FSQRTS, FSQRTD, the slower operations may finish and update cexc after an operation which started after it in the instruction stream.

This means that for a sequence such as FDIVS, FADDS, NOP, STFSR, the cexc field written out by store-FSR might represent the result of the FDIVS operaton rather than the FADDS operation, if the FADDS computation completed earlier in the hardware.

This is normally not a concern for compiled code as the exact order of floating-point operations is not controlled at the source code level and therefore looking at cexc is of limited use outside of trap context. The exceptions from all operations will still accumulate into the accrued exception field (aexc) as expected, and this can be checked after a series of operations.