UT699 LEON 3FT Microprocessor GRFPU/GPC Floating Point Errata

Overview

This errata document covers all known bugs in the GRFPU/FPC Floating Point Unit present in GRLIB revision 1996, affecting the UT699 LEON 3FT Microprocessor.

Workaround summary

For compiled (C/C++) code:

- The errata can be worked around by using a modified compiler toolchain without having to change application source code
- The remaining errors can be handled by following the restrictions below

For assembler code:

- Force all DIV/SQRT operations to complete by storing their result to memory before starting another floating-point operation
- Avoid the FSMULD operation
- Do not use FDIVS/FSQRTS, use double precision operations instead

Additional restrictions:

- Applications should not catch floating-point traps and continue execution afterward
- Non-standard mode should not be used
- Rounding modes, other than round-to-nearest (the default), should not be used
- Single precision division/square root should not be used, use double precision instead

Impact summary

If not using the proposed workarounds in this document, the errors could, in worst case, lead to an unexpected trap, causing the application to abort.

With the above workarounds and restrictions in place, the FPU will work as intended.

The workarounds cause a performance reduction in the order of a few percent and a slight increase in code size.

FSMULD signaling overflow condition

Description:

If the result of an FSMULD operation has an exponent in the range 128.256, the FPU generates an overflow (and inexact) condition instead of returning the correct value. If the result exponent is in the range -256.-128, the FPU signals underflow (and inexact) instead of returning the correct value. In other words, FSMULD generates over/underflow as if it was a single precision multiplication followed by a single-to-double conversion, rather than a single-to-double conversion followed by a

double precision multiplication

Affects:

- All software where the FSMULD instruction is used.
- Compiled code using single precision values may be affected, depending on whether the compiler emits FSMULD

Impact:

- Unexpected floating-point trap if trap, on FPU over/underflow is enabled
- Incorrect output value affecting further execution

Application impact:

Applications where conversion to double is needed in order to avoid overflow (and not only to improve precision) are affected by this bug. Also, this can cause problematic behavior with applications that trap on the unexpected condition.

For programs where conversion from single to double precision is done to improve precision and a single precision overflow is not expected, impact by this bug is limited.

Workarounds:

- Use FSTOD followed by FMULD instead of FSMULD.
- For compiled code, the GCC compiler can be modified to not generate the FSMULD instruction.

FMULS precision loss after FDIVD/FSQRTD

Description:

If an FMULS instruction is issued while an FDIVD or FSQRTD instruction is in the execution pipeline, the result of the FMULS instruction can get incorrectly rounded, causing an error of one mantissa lsb, and the inexact bit might be incorrect.

Affects:

- All software where FMULS is mixed with FDIVD and FSQRTD instructions

(Note that the converse does not apply, i.e. mixing FDIVS and FMULD does not trigger this error.)

Impact:

- One bit loss of accuracy of FMULS operation

Application impact:

This is expected to have minor impact on applications, since programs requiring this level of accuracy in general would use double precision instead.

Workaround alternatives:

- Replace single precision multiplication with double precision multiplication.
- Force each FDIVD/FSQRTD to complete by putting a floating-point store of the result to memory immediately after the instruction.
- Add NOP:s around all iterative instructions (5 NOP:s before and 20/28 NOP:s after each FDIV/FSQRT) to ensure that these instructions always execute alone in the pipeline.

Infinity arithmetic errors in certain rounding modes

Description:

In round-to-zero, round-to-infinity and round-to-minus-infinity rounding modes, infinity arithmetic operations that should

return +/- infinity as result, instead returns +/- the largest positive value.

Affects:

- Software running in rounding modes other than round-to-nearest and using infinity arithmetic.

Impact:

- Incorrect results (largest value instead of infinity)

Application impact:

For applications using the default round-to-nearest mode, this bug has no impact.

For other rounding modes, this is expected to have minor application impact.

Workaround:

- Where application permits, use round-to-nearest rounding mode

FSQRTS special case rounded incorrectly

Description:

An FSQRTS operation where the resulting root can be exactly represented is expected in all rounding modes to output this result and not set the inexact flag.

In round-to-zero and round-to-minus-infinity rounding modes, the output result of the FPU is in some cases, in the exact result minus one mantissa-lsb and with the inexact flag set.

Affects:

To trigger this bug, all below conditions must be met:

- The FPU:s rounding mode is round-to-zero or round-to-minus-infinity
- Single precision square root is used
- The result can be represented as an exact single precision floating-point number

Impact:

- Inexact result instead of exact result

Application impact:

For applications using the default round-to-nearest mode, this bug has no impact.

Note that the maximum rounding error in the affected rounding modes is not increased by this bug, the bug changes the error range from 0 err < 1 to 0 err 1. It is therefore expected to have minor impact.

This may cause a problem for special algorithms that rely on exact square root results from certain input values, or software that needs exactly IEEE-conforming results to function correctly.

Workaround alternatives:

- Use double precision square root instead, and convert back to single precision if necessary
- If application permits, use round-to-nearest or round-to-infinity rounding modes instead

FDIV/FSQRT results incorrectly forwarded or stored

Description:

A rare corner case where a DIV/SQRT operation is immediately followed by another unrelated floating-point operation, can

cause the wrong data to be output from the DIV/SQRT operation.

Affects:

- All software using FDIVS,FDIVD,FSQRTS,FSQRTD instructions

Impact:

- Bad (undefined) data stored to memory or forwarded to following operations

Application impact:

This error has major impact on any software using FDIVS,FDIVD,FSQRTS or FSQRTD instructions.

Apart from being incorrect results, if the bad data happens to represent a denoral number, it could lead to an unfinished FP operation trap as the program continues. This trap causes the program to abort in many environments.

Workaround alternatives:

- Add NOP:s around all iterative instructions (5 NOP:s before and 20/28 NOP:s after each FDIV/FSQRT) to ensure that these instructions always execute alone in the pipeline
- Force each FDIVS/FDIVD/FSQRTS/FSQRTD to complete by putting a floating-point store of the result to memory immediately after the instruction

FITOS/FITOD incorrect in non-standard mode

Description:

When running in non-standard mode, the FITOS/FITOD instructions may return incorrect results and may also set condition codes incorrectly.

Affects:

- Software running in non-standard mode

Impact:

- Incorrect data output

Application impact:

For applications operating in the default mode, this bug has no impact.

Since the non-standard mode is not enabled by default and varies between FPU implementations, this only affects applications written specifically for this FPU that enable the non-standard mode.

Workaround alternatives:

- Avoid using non-standard mode if application permits
- Switch out of non-standard mode when performing FITOS/FITOD

DIV/SQRT rounding errors in certain rounding modes

Description:

The FDIVS, FDIVD, FSQRTS and FSQRTD operations round results incorrectly in round-to-zero, round-to-infinity and

round-to-minus-infinity modes, leading to 1 mantissa-lsb error. Also, the inexact flag is set incorrectly.

Affects:

- Software running in rounding-modes other than round-to-nearest.

Impact:

- One lsb loss of accuracy
- Inexact flag set incorrectly

Application impact:

This has no impact on software running in the round-to-nearest rounding mode, which is the default.

Workaround alternatives:

- Where application permits, use round-to-nearest rounding mode
- For single precision, perform DIV/SQRT operations in double precision and, if necessary, round in software
- After the operation, multiply up, compare with the input value and adjust the result +/-1 lsb to get the desired rounding.

FDIVS returning denormal results

Description:

Single precision division, FDIVS, in certain cases returns denormal results instead of returning zero and setting the underflow condition code.

Affects:

- Software using FDIVS

Impact:

- Denormal results

Application impact:

This error could lead to an unfinished FP operation trap as the program tries to use the output value. This trap leads to the program being aborted in many environments.

Workaround:

- Use double precision division instead of single precision division

Last operation executed twice on FP trap recovery

Description:

When reading out the deferred floating-point queue after a floating-point trap, the FPU may, in certain conditions,execute the last deferred operation immediately following the _gstd %fq_h instruction.

Affects:

- Applications that catch FPU traps, and continue execution afterwards

Impact:

- After handling the FPU trap and returning to the application code, the contents of the FPU registers may not be correct

Application impact:

In the majority of environments, a floating-point trap causes the program to terminate. In that case, the incorrect result is never

used so this bug has no impact.

If the application does continue after a floating-point trap, the floating-point register state after trap recovery may not be correct.

Workaround alternatives:

- Avoid catching FPU traps in the application