LEON3 and LEON4 Multiply Accumulate (MAC) Errata

Technical note 2019-08-21

Doc. No GRLIB-TN-0017

Issue 1.0

Template: GQMS-TPLT-1-1-0

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2019-08-21		First issue of document.

TABLE OF CONTENTS

© Cobham Gaisler AB

1 INTRODUCTION

1.1 Scope of the Document

This document describes errata for the LEON3 and LEON4 processors when built with support for the SPARC V8e Multiply Accumulate feature.

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.

© Cobham Gaisler AB

2 AFFECTED PRODUCTS

2.1 General

This document applies to designs based on certain versions of GRLIB, where the design contains the LEON3, LEON3FT, LEON4, and LEON4FT processors, and support for the multiply-accumulate function has been enabled.

A GRLIB release is identified by the name grib-type-yyyy.q-b*buildid*. *buildid* is the main identifier for the version of the included IP cores. The build ID is incremented when-ever a new GRLIB release is made that has changes to the IP cores. The build ID is also included in the system's plug&play information.

The affected GRLIB versions varies for different errata and processors according to the table below.

GRLIB build ID	Processors	Issue #1	Issue #2	Issue #3
b4059 and earlier	LEON3, LEON3FT	Yes	No	Yes
b4060-b4079	LEON3	Yes	Yes	Yes
	LEON3FT	Yes	No	Yes
b4080-b4166	LEON3, LEON3FT LEON4, LEON4FT	Yes	Yes	Yes
b4167-b4241	LEON3, LEON3FT	No	Yes	Yes
	LEON4, LEON4FT	Yes	Yes	Yes
b4243 and newer	LEON3, LEON3FT LEON4, LEON4FT	No	No	No

2.2 Cobham components

No Cobham components are affected by these errata, as none of the LEON processors inside these components include the MAC feature.

© Cobham Gaisler AB

2.3 How to check if a custom 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).
2. Determine if your design is using LEON3 or LEON4, by examining the top level and seeing if the instantiated processor core is LEON3 (leon3x, leon3s) or LEON4 (leon4x, leon4s). Compare the processor and GRLIB revision with the affected versions table above.
3. Check if the MAC function is enabled by examining the mac generic into the processor. This must be set to a non-zero value for the errata to apply.

In case the VHDL sources are not available, the status of the design can be checked from software as follows:

1. Check the GRLIB revision. This can be read out from the Plug’n’play information on the devices, by default at the lower 16 bits of address 0xFFFFFFF0. The revision is also reported automatically when connecting with GRMON
2. Which processor is used can be found by scanning the AMBA plug’n’play information. It is also reported by GRMON when connecting.
3. Whether the processor contains the MAC support can be seen by examining bit 9 of the %asr17 special register. If bit 9 is ‘1’ then MAC support is implemented.

© Cobham Gaisler AB

3 IMPACT

If the suggested workarounds are not implemented on an affected system, the errata result in incorrect results of calculations involving the MAC operations. Additionally, in the case of issue 1, unrelated reads of asr17 may return the wrong value leading to issues in particular with SMP operating systems.

If the workarounds are implemented, no functional issues remain. The workarounds have a small impact on performance and code size.

© Cobham Gaisler AB

4 MAC ISSUE #1: INCORRECT DATA FORWARDING FOR ASR17 AND ASR24-31

4.1 Affected versions

See section 2.

4.2 Description

The issue triggers when you have three conditions:

1. A JMPL[1] instruction 1-2 instructions before a cacheline boundary (offset +0x14 or +0x18 into the cacheline for 8 word line size)
2. Target of the JMPL is a RDASR operation for asr17 or asr24-31
   (if you are using the non-pinelined 16x16 multiplier the problem can happen also if you jump to the instruction before the RDASR)
3. MAC instructions stored in the Icache data RAM, in the cache set following the JMPL instruction. The issue can trigger even if the cache line is invalid or valid but corresponding to a different address.

[1] Note that the JMPL opcode is used for function return and trap return (ret and retl is alias for "JMPL %i0/%o0, %g0") and also for indirect function calls ("call %reg" is alias for "JMPL %reg, %o7") but not normal function calls (CALL <constant>) or branches.

What will happen in this case is that the read ASR operation will return incorrect data. Note that this happens for asr17 register which is unrelated to the MAC operation and is supposed to be constant.

The root cause of the issue is an instruction decoding and forwarding logical path not being correctly disabled for the two pipeline bubbles following an JMPL instruction.

4.3 Workarounds

The most straightforward workaround is to copy ASR17 into a global register that is not otherwise used by software and to use that register instead of ASR17. Normally the purpose of reading asr17 is for SMP systems for the code to read out which processor it is running on, so an optimization could be to have already extracted that field in the selected global register..

If this workaround is not acceptable (for example, because no global register can be made available for this purpose) then one would have to make sure through some other means that the triggering sequence does not occur in the code. It is important to keep in mind that the triggering sequence can occur on a return from a trap, therefore an interrupt happening and then returning to an rdasr operation could trigger it. One would therefore have to design the software so that interrupts are disabled while performing the rd asr17 readout and avoid any other trap-inducing instructions just

© Cobham Gaisler AB

before. Another option is to implement the read-out of ASR17 in a software trap handler that is guaranteed to run with traps disabled.

If the MAC instructions are not used in the application, the issue is avoided as MAC instructions are never present in the instruction cache. However, care has to be taken to ensure the instruction cache is cleared or initialized with known data when powering up to avoid random instructions being present in the cache data memory. Also, care has to be taken to ensure there are no constants or data on the same cache lines as code that might enter into the instruction cache and become interpreted as MAC instructions.

© Cobham Gaisler AB

5 MAC ISSUE #2: INCORRECT BRANCH MISPREDICTION ANULLATION FOR MAC OPERATIONS

5.1 Affected versions

See section 2. In addition, for LEON3, the issue only occurs when static branch prediction is enabled in the processor.

5.2 Description

The issue triggers when the following three conditions are met:

1. An SMAC/UMAC is one of the two first instructions in the target of a conditional branch.
2. The conditional branch is not taken.
3. Within the two first instructions after the not taken branch, a RDASR operation for %asr18 or a RDY operation is performed.

The effect of the issue is that the wrong data will be returned for the RDASR/RDY operation.

5.3 Workarounds

The issue can be avoided, either by avoiding SMAC/UMAC instructions near a branch target, or by avoiding to read out the asr18 or y registers after a branch to such instruction has been not taken.

A typical scenario is that the SMAC/UMAC instruction is performed in a loop, and the asr18/y registers are read out after the last iteration of the loop. It is therefore better for performance to keep the loop as small as possible and add some instructions between the looping branch not taken after the last iteration and reading out the final asr18 / y value.

© Cobham Gaisler AB

6 MAC ISSUE #3: INCORRECT REGISTER FORWARDING FROM MAC TO STORE

6.1 Affected versions

See section 2. In addition, for LEON3, this only applies when the load delay (lddel generic) is set to 1.

6.2 Description

The sequence of a MAC instruction (smac/umac) immediately followed store of the result, will write out the wrong data. This is due to a missing interlock condition in the processor pipeline for this case.

6.3 Workaround

The workaround for this issue is to ensure there is always another operation between the MAC operation and the store, possibly a nop if no useful operation can be selected.

© Cobham Gaisler AB

Copyright © 2019 Cobham Gaisler.

Information furnished by Cobham Gaisler is believed to be accurate and reliable. However, no responsibility is assumed by Cobham Gaisler for its use, or for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Cobham Gaisler.

All information is provided as is. There is no warranty that it is correct or suitable for any purpose, neither implicit nor explicit.