LEON3FT Stale Cache Entry After Store with Data Tag Parity Error

---

Technical note 2017-06-15

Doc. No GRLIB-TN-0009

Issue 1.1

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2016-09-23	All	First issue
1.1	2017-06-15	2.2.2	Updated BCC1 Added BCC2

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

This document describes a corner case present in the LEON3FT integer pipeline and data cache where a data cache tag RAM parity error, caused by radiation single event upset or inserted by force using the error injection diagnostic interface, combined with a specific sequence of store instructions and a number of additional conditions, can lead to an out-of-date entry in the data cache.

Note that software documentation may refer to the behaviour described in this document as the back-to-back, or b2b, store issue.

The stale cache entry problem described by this document has shown to be difficult to reproduce due to the number of conditions that need to be met for a RAM parity error to cause a stale cache entry. The creation and occurrence of out-of-date (stale) cache entries is deemed to be of low probability. In addition to this, there is software application specific masking since the stale cache entry, like all cache entries, have limited life time before it is evicted from the cache as a result of normal operation. The only known occurrences of software malfunctions at the time of writing are during the GR712RC radiation test campaign (in 2011) and during error injection tests at Cobham Gaisler (in 2016).

1.2 Distribution

LEON3FT 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.

1.2.1 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.

1.2.1.1 Checking GRLIB version

The GRLIB build ID is present in the AMBA plug&play area. The build ID is also reported by the GRMON debug monitor when connecting to the device.

If you have licensed GRLIB for use in your own FPGA or ASIC design, the GRLIB version 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 STALE CACHE ENTRY AFTER STORE WITH DATA TAG PARITY ERROR

2.1 Affected versions

All LEON3FT versions up to GRLIB build 4174 where the cache is implemented with parity protection (*cft* VHDL generic is non-zero) are affected. The LEON4FT is not affected in any configuration. Commercial versions (non-FT) of LEON3 and LEON4 are not affected.

Affected Cobham components:

- GR712RC
- LEON3FT-RTAX – all versions
- UT699
- UT699E
- UT700

Unaffected components:

- GR740 – LEON4FT is unaffected

2.2 Description

The processor behaviour described in this document can cause a data coherency issue when the processor has executed a back-to-back store operation and the first store's address hits a cache set that has a data cache tag parity error. The second store operation successfully stores its data content into the main memory but the data cache controller fails to update the old data in the data cache with the new data from the store operation. This leads to an incoherent state, data in the data cache is not the same as in the main memory. A subsequent read from the second location will get the old data instead of the new data if the data cache line is still valid in the cache.

The stale cache entry can be created on either of two specific sequences of executed instructions:

Sequence A:

1. store of word size or less (e.g. st / stb / sth / stf)
2. any single instruction that is not a load or store
3. any store instruction (e.g. st / stb / sth / stf / std / stdf)

Sequence B:

1. store of double word size (e.g. std / stdf)
2. any store instruction (e.g. st / stb / sth / stf / std / stdf)

As both sequences involve two stores, they will be referred to as the first and second store, respectively, in the following text. Note that stores that are part of an atomic instruction (e.g. ldst / casa) are not affected. Stores to all address space identifiers (ASI) should be considered as affected.

In order for the instruction sequence to trigger the behaviour that creates a stale cache entry, a number of additional conditions must all be met:

- There must be an ongoing store already in the write buffer of the processor that has not yet completed when the first store of the sequence is executed.
- The data cache must be enabled
- The first store's address goes to a data cache set which has a tag parity error (caused by radiation SEU or inserted by force using the error injection diagnostic interface)
- The second store goes to an address which is already cached inside the data cache and does not have any parity error.

For the stale cache entry to affect software operation, the following condition must also be met:

- The location that is stored by the second store is read back some time after the sequence, while the data is still in the data cache.

The intended behaviour under these conditions, is that the first store should detect the parity error and invalidate the corresponding cache set, and the store is then sent to the memory controller on

the on-chip bus. The second store should both update the already cached copy of the data and send it to the memory controller. Following loads to the second store's address can get the updated data either from cache (as long as the line is still in cache) or from the bus if the cache line has become evicted.

When all the conditions in the itemized lists above are met, the second store in the sequence does not update the cache correctly. Therefore following loads that are done to the stored address while the data cache entry is still inside the cache will get stale data from before the store was issued. Table 1 shows the state of the data cache and memory for both the expected case and for the case where the stale cache entry is created.

	Expected state				**State if stale entry is created**			
Time in sequence	**DCache 1**	**Memory 1**	**DCache 2**	**Memory 2**	**DCache 1**	**Memory 1**	**Dcache 2**	**Memory 2**
Before 1st store	Tag parity error	Old value 1	Old value 2	Old value 2	Tag parity error	Old value 1	Old value 2	Old value 2
After 1st store	Empty	New value 1			Empty	New value 1		
After 2nd store			New value 2	New value 2			Old value 2	New value 2

Table 1: Description of cache state

2.2.1 Workaround / Mitigation

The stale cache entry can be avoided by avoiding the instruction sequences A and B. This can be done by inserting an additional NOP between the two stores whenever otherwise one of the sequences would be generated. An alternative can be to reorder the second store and the following instruction, provided that this does not affect correctness of the program.

The workaround has low performance impact since the processor's write buffer only holds one store and the second store will cause the pipeline to block until the first store operation has left the write buffer. The extra cycle caused by an added NOP will therefore be masked by the first store's delay. Some secondary performance impact can be caused by the increase in code size and resulting increased instruction cache utilization. Trials with patched toolchains used on a range of test suite programs show an average code size increase of less than 1%.

Note that software documentation may refer to the behaviour described in this document as the back-to-back, or b2b, store issue.

2.2.2 Toolchain versions with workaround

Toolchains with compilers that insert one NOP instruction when otherwise vulnerable sequences would be generated will be available in October 2016 for download from Cobham Gaisler's website. The “toolchain” term refers to the complete chain required to create an executable binary (compiler, linker, libraries). For example, with the BCC toolchain the GCC compiler will be extended with a

patch that inserts NOP instructions to avoid vulnerable sequences of code. The libraries that are distributed with the compiler will be recompiled to avoid linking in library functions that have code that is vulnerable in combination with the condition listed in this document. In addition to this, code for trap handlers such as window overflow trap will be manually patched since these handlers are written in assembly language.

The more complex operating systems such as RTEMS and VxWorks have additional code written in assembly language that will be manually corrected before rebuilding the toolchains and recreating the operating system distributions. The scan script developed by Gaisler is used to verify that all operating systems are free from vulnerable code sequences. With the software workarounds in place, end user code will not need to take further steps as long as hand crafted assembly sequences (that contain vulnerable code sequences) are not included by the users. The scan script is available from http://gaisler.com/notes.

The compiler workaround is activated using -mfix-b2bst. The workaround is by default activated when compiling for UT699. This means that when building using -mtune=ut699 or -mfix-u699 (depending on toolchain being used) no changes are required to the compiler switches. During linking the same set of compiler flags must used to select the matching prebuilt libraries.

The following release versions and later will include the workaround in toolchain and OS/libraries:

- BCC2 2.0.1
- BCC1 1.0.47
- RCC-1.2.20
- VxWorks 6.9 2.0.4, toolchain 4.9-1.0.2
- VxWorks 6.7 1.0.19, toolchain 4.1-1.0.13
- MKPROM 2.0.61

2.3 Risk of software errors

As described in section 2.2, several conditions have to be met for the stale cache entry to be created and for program state to be affected by the stale entry. In addition to execution of sequence A or sequence B:

- There must be an ongoing store already in the write buffer of the processor that has not yet completed when the first store of the sequence is executed.
- The data cache must be enabled
- The first store's address goes to a data cache set which has a tag parity error (caused by radiation SEU or inserted by force using the error injection diagnostic interface)
- The second store goes to an address which is already cached inside the data cache and does not have any parity error.
- The location that is stored by the second store is read back some time after the sequence, while the data is still in the data cache.

All items above are software application specific except for the presence of tag data RAM errors. To get a first estimation of the probability of software being affected, provided that the application contains vulnerable code sequences, the probability of having a tag RAM parity error can be calculated. The number of SRAM bits used for data tag RAMs are:

- UT699: 29184 bits
- UT699E/UT700: 55296 bits
- GR712RC: 58368 bits (per processor core)

Note that the probability of a program failure to occur is then given by multiplying the tag RAM parity error probability with the probability of having all the conditions listed above met for a specific cache set.

As part of the analysis work performed in writing this technical note, extensive tests with error injection have been performed to evaluate the effect of stale cache entries. Trials have shown that there is no a direct correlation between the number of vulnerable code sequences in an application, or even how common the sequences are, and probability of triggering a software failure. A thorough analysis needs to take all conditions in the itemized list above into account.

It should be noted that the probability of getting an upset in the data tag cache RAM is significantly higher than the probability of actually triggering a stale cache entry. The authors of this note cannot make any assumptions about the software application used on the processor and the number of affected bits in the tag RAM is the only data input that can be provided to calculate the probability of software to be affected. The scan script provided by Cobham Gaisler can be used to assess how many vulnerable code sequences that exist in an application. Further analysis is then required to determine which of the vulnerable sequences that will also meet all conditions listed in the itemized list above.

3 FAQ

3.1 For which types of parity error does the problem occur?

Only data cache TAG errors can trigger the events needed for creation of the stale cache entry. Neither of data cache data errors, instruction cache tag errors, or instruction cache data errors will lead to creation of a stale cache entry.

3.2 What happens if the tag of the data targeted by the first store is correct, but if another tag in the same set has a parity error?

If a tag in the same set (at the same location in another cache way) has a parity error then the creation of a stale cache entry can be triggered in the same way.

3.3 Are the floating-point store instructions affected?

Yes, for the purposes of this technical note, floating point store behaves identically to integer store of the same size.

3.4 Is there a relationship with the creation of stale cache entries and the GR712RC FPU data errors reported in the 2011 RADECS report?

Yes. We have reviewed log files for all FPU data errors observed during the RADECS 2011 *Radiation characterization of a dual core LEON3-FT processor*[1] GR712RC SEE test (Table IV) together with results from a 2016 error injection effort. The error injection effort was run on both the GR712RC device and on the GR712RC IP implemented on a FPGA prototype. It has been concluded that upsets in the tag of the data cache is the only cause for all 11 FPU data errors observed at low LET (3.3 to 10.2 MeV/mg/cm^2, where only SRAM SEUs were expected, and therefore zero functional errors were expected). In other words at low LET (3.3 to 10.2 MeV/mg/cm^2) the registers in the design are not sensitive and only SRAM upsets are expected.

Furthermore it has been demonstrated that the software workarounds described in this document removes all program errors that were previously seen during the error injection runs. All errors are also removed when performing error injection, using original test software, on the GR712RC IP implemented in FPGA together with a RTL (VHDL) patch that corrects the identified issue.

The “IU data error” reported in the 2011 RADECS report for 15.9 MeV/mg/cm^2 can also be explained by a stale cache entry following an upset in the data tag RAM. Review of the test log files shows that a data tag error was observed by the test monitoring software in connection with the “IU

1 The paper was published in the RADECS 2011 conference proceedings.
IEEE Xplore: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6131334
Article Title: Radiation characterization of a dual core LEON3-FT processor, ISBN: 978-1-4577-0585-4, Posted Online Date: Fri Mar 09 00:00:00 EST 2012, Authors: Sturesson, F.; Gaisler, J.; Ginosar, R.; Liran, T.

data error”. The IU test application has been reviewed and the program does contain code sequences that are described as vulnerable in this document. The IU-test application is less sensitive to the error due to the structure of the program.