

# ARM Processor Cortex™-A15 MPCore-NEON (MP009)

**Product Revision r4**

## **Software Developers Errata Notice**

**Non-Confidential - Released**



## Software Developers Errata Notice

Copyright © 2016 ARM. All rights reserved.

### Non-Confidential Proprietary Notice

This document is protected by copyright and the practice or implementation of the information herein may be protected by one or more patents or pending applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document.

This document is Non-Confidential but any disclosure by you is subject to you providing the recipient the conditions set out in this notice and procuring the acceptance by the recipient of the conditions set out in this notice.

Your access to the information in this document is conditional upon your acceptance that you will not use, permit or procure others to use the information for the purposes of determining whether implementations infringe your rights or the rights of any third parties.

Unless otherwise stated in the terms of the Agreement, this document is provided “as is”. ARM makes no representations or warranties, either express or implied, included but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement, that the content of this document is suitable for any particular purpose or that any practice or implementation of the contents of the document will not infringe any third party patents, copyrights, trade secrets, or other rights. Further, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of such third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT LOSS, LOST REVENUE, LOST PROFITS OR DATA, SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Words and logos marked with ® or TM are registered trademarks or trademarks, respectively, of ARM Limited. Other brands and names mentioned herein may be the trademarks of their respective owners. Unless otherwise stated in the terms of the Agreement, you will not use or permit others to use any trademark of ARM Limited.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

In this document, where the term ARM is used to refer to the company it means “ARM or any of its subsidiaries as appropriate”.

Copyright © 2016 ARM Limited 110 Fulbourn Road, Cambridge, England CB1 9NJ. All rights reserved.

### Web Address

<http://www.arm.com>

### Feedback on content

If you have any comments on content, then send an e-mail to [errata@arm.com](mailto:errata@arm.com) . Give:

- the document title
- the document number, ARM-EPM-045620
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

**Release Information**

Errata are listed in this section if they are new to the document, or marked as “updated” if there has been any change to the erratum text in Chapter 2. Fixed errata are not shown as updated unless the erratum text has changed. The summary table in section 2.2 identifies errata that have been fixed in each product revision.

**21 Oct 2016: Changes in Document v15**

Page	Status	ID	Cat	Rare	Summary of Erratum
31	New	854671	CatC		Extremely slow response for unaligned loads crossing a 64-byte cacheline boundary might cause the processor to deadlock

**09 Oct 2015: Changes in Document v14**

Page	Status	ID	Cat	Rare	Summary of Erratum
30	Updated	851024	CatC		Persistent evictions combined with interconnect backpressure might stall Write-Back No-Allocate stores

**21 Sep 2015: Changes in Document v13**

Page	Status	ID	Cat	Rare	Summary of Erratum
30	New	851024	CatC		Persistent evictions combined with interconnect backpressure might stall Write-Back No-Allocate stores

**20 Feb 2015: Changes in Document v12**

Page	Status	ID	Cat	Rare	Summary of Erratum
29	New	842119	CatC		Instruction issued through ITR by debugger executes incorrectly for PCLKDBG frequencies much slower than the processor

**04 Nov 2014: Changes in Document v11**

Page	Status	ID	Cat	Rare	Summary of Erratum
15	New	836969	CatB		Code sequence continuously hitting the L1 cache can block snoop

**09 Sept 2014: Changes in Document v10**

Page	Status	ID	Cat	Rare	Summary of Erratum
28	New	834569	CatC		PMU event BUS_CYCLES might be incorrect in some cases

**08 Aug 2014: Changes in Document v9**

Page	Status	ID	Cat	Rare	Summary of Erratum
27	New	832972	CatC		HSTR.{T7,T8,T15} bits incorrectly trap CDP instructions

**16 May 2014: Changes in Document v8**

Page	Status	ID	Cat	Rare	Summary of Erratum
14	New	827671	CatB		WriteClean and WriteBack transaction reordering might cause data corruption
19	New	827923	CatB	Rare	TLB maintenance operations might not be synchronized by DSB instruction
25	New	826375	CatC		Debug accesses in User mode do not properly generate undefined instruction exceptions for some SIMD and VFP registers
26	New	826969	CatC		Cortex-A15 might violate read-after-read memory ordering on a load forwarding from a store crossing a 16-byte boundary

**10 Mar 2014: Changes in Document v7**

Page	Status	ID	Cat	Rare	Summary of Erratum
24	New	822719	CatC		Read following a write of a Timer TVAL register might return incorrect value

**11 Dec 2013: Changes in Document v6**

Page	Status	ID	Cat	Rare	Summary of Erratum
23	New	816469	CatC		Double-bit ECC error during hardware correction of a single-bit ECC error might cause data corruption

**22 Nov 2013: Changes in Document v5**

Page	Status	ID	Cat	Rare	Summary of Erratum
13	New	816470	CatB		DSB instruction in processor power down sequence may not drain all required transactions

**01 Nov 2013: Changes in Document v4**

Page	Status	ID	Cat	Rare	Summary of Erratum
11	New	813469	CatB		An unaligned store instruction crossing a 4k page boundary at the same time as the lower page is invalidated might stall

**25Sept 2013: Changes in Document v3**

Page	Status	ID	Cat	Rare	Summary of Erratum
18	New	812169	CatB	Rare	DVM message blocked by copyback on AMBA Chi interconnect

**21 Aug 2013: Changes in Document v2**

Page	Status	ID	Cat	Rare	Summary of Erratum
22	New	810072	CatC		When a single-bit ECC error occurs in the L2, uncorrected data might be returned

**10 Jul 2013: Changes in Document v1**

Page	Status	ID	Cat	Rare	Summary of Erratum
9	New	784420	CatB		Speculative instruction fetches with MMU disabled might not comply with architectural requirements
16	New	763126	CatB	Rare	Three processor exclusive access livelock
10	New	785769	CatB		Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode
20	New	773023	CatC		Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI
21	New	777769	CatC		ICache parity error may not be corrected for NC code

## Contents

<b>CHAPTER 1.</b>	<b>6</b>
<b>INTRODUCTION</b>	<b>6</b>
1.1. Scope of this document	6
1.2. Categorization of errata	6
<b>CHAPTER 2.</b>	<b>7</b>
<b>ERRATA DESCRIPTIONS</b>	<b>7</b>
2.1. Product Revision Status	7
2.2. Revisions Affected	7
<b>r4p0 implementation fix</b>	<b>8</b>
2.3. Category A	9
2.4. Category A (Rare)	9
2.5. Category B	9
784420: Speculative instruction fetches with MMU disabled might not comply with architectural requirements .....	9
785769: Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode.....	10
813469: An unaligned store instruction crossing a 4k page boundary at the same time as the lower page is invalidated might stall.....	11
816470: DSB instruction in processor power down sequence may not drain all required transactions .....	13
827671: WriteClean and WriteBack transaction reordering might cause data corruption .....	14
836969: Code sequence continuously hitting the L1 cache can block snoop.....	15
2.6. Category B (Rare)	16
763126: Three processor exclusive access livelock .....	16
812169: DVM message blocked by copyback on AMBA Chi interconnect.....	18
827923: TLB maintenance operations might not be synchronized by DSB instruction.....	19
2.7. Category C	20
773023: Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI .....	20
777769: ICache parity error may not be corrected for NC code .....	21
810072: When a single-bit ECC error occurs in the L2, uncorrected data might be returned.....	22
816469: Double-bit ECC error during hardware correction of a single-bit ECC error might cause data corruption .....	23
822719: Read following a write of a Timer TVAL register might return incorrect value .....	24
826375: Debug accesses in User mode do not properly generate undefined instruction exceptions for some SIMD and VFP registers.....	25
826969: Cortex-A15 might violate read-after-read memory ordering on a load forwarding from a store crossing a 16-byte boundary .....	26
832972: HSTR.{T7,T8,T15} bits incorrectly trap CDP instructions .....	27
834569: PMU event BUS_CYCLES might be incorrect in some cases .....	28
842119: Instruction issued through ITR by debugger executes incorrectly for PCLKDBG frequencies much slower than the processor.....	29
851024: Persistent evictions combined with interconnect backpressure might stall Write-Back No-Allocate stores.....	30
854671: Extremely slow response for unaligned loads crossing a 64-byte cacheline boundary might cause the processor to deadlock.....	31

# Chapter 1.

## Introduction

This chapter introduces the errata notice for the ARM Cortex-A15 MPCore-NEON processor.

### 1.1. Scope of this document

This document describes errata categorized by level of severity. Each description includes:

- the current status of the defect
- where the implementation deviates from the specification and the conditions under which erroneous behavior occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a ‘work-around’ where possible

This document describes errata that may impact anyone who is developing software that will run on implementations of this ARM product.

### 1.2. Categorization of errata

Errata recorded in this document are split into the following levels of severity:

**Table 1**      **Categorization of errata**

Errata Type	Definition
Category A	A critical error. No workaround is available or workarounds are impactful. The error is likely to be common for many systems and applications.
Category A(rare)	A critical error. No workaround is available or workarounds are impactful. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category B	A significant error or a critical error with an acceptable workaround. The error is likely to be common for many systems and applications.
Category B(rare)	A significant error or a critical error with an acceptable workaround. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category C	A minor error.

## Chapter 2.

# Errata Descriptions

### 2.1. Product Revision Status

The *mpn* identifier indicates the revision status of the product described in this book, where:

- rn** Identifies the major revision of the product.
- pn** Identifies the minor revision or modification status of the product.

### 2.2. Revisions Affected

Table 2 below lists the product revisions affected by each erratum. A cell marked with **X** indicates that the erratum affects the revision shown at the top of that column.

This document includes errata that affect revision r4 only.

Refer to the reference material supplied with your product to identify the revision of the IP.

**Table 2** Revisions Affected

ID	Cat	Rare	Summary of Erratum	r4p0	r4p1
784420	CatB		Speculative instruction fetches with MMU disabled might not comply with architectural requirements	X	X
785769	CatB		Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode	X	X
813469	CatB		An unaligned store instruction crossing a 4k page boundary at the same time as the lower page is invalidated might stall	X	
816470	CatB		DSB instruction in processor power down sequence may not drain all required transactions	X	X
827671	CatB		WriteClean and WriteBack transaction reordering might cause data corruption	X	X
836969	CatB		Code sequence continuously hitting the L1 cache can block snoop	X	X
763126	CatB	Rare	Three processor exclusive access livelock	X	X
812169	CatB	Rare	DVM message blocked by copyback on AMBA Chi interconnect	X	X
827923	CatB	Rare	TLB maintenance operations might not be synchronized by DSB instruction	X	
773023	CatC		Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI	X	X
777769	CatC		ICache parity error may not be corrected for NC code	X	X
810072	CatC		When a single-bit ECC error occurs in the L2, uncorrected data might be returned	X	
816469	CatC		Double-bit ECC error during hardware correction of a single-bit ECC error might cause data corruption	X	
822719	CatC		Read following a write of a Timer TVAL register might return incorrect value	X	X

ID	Cat	Rare	Summary of Erratum	r4p0	r4p1
826375	CatC		Debug accesses in User mode do not properly generate undefined instruction exceptions for some SIMD and VFP registers	X	X
826969	CatC		Cortex-A15 might violate read-after-read memory ordering on a load forwarding from a store crossing a 16-byte boundary	X	
832972	CatC		HSTR.{T7,T8,T15} bits incorrectly trap CDP instructions	X	X
834569	CatC		PMU event BUS_CYCLES might be incorrect in some cases	X	X
842119	CatC		Instruction issued through ITR by debugger executes incorrectly for PCLKDBG frequencies much slower than the processor	X	X
851024	CatC		Persistent evictions combined with interconnect backpressure might stall Write-Back No-Allocate stores	X	X
854671	CatC		Extremely slow response for unaligned loads crossing a 64-byte cacheline boundary might cause the processor to deadlock	X	X

### r4p0 implementation fix

Note the following errata may be fixed in some implementations of r4p0. This can be determined by reading the REVIDR register where a set bit indicates that the erratum is fixed in this part.

REVIDR[9:0]	Reserved
REVIDR[10]	810072 When a single-bit ECC error occurs in the L2, uncorrected data might be returned
REVIDR[11]	Reserved
REVIDR[12]	813469 An unaligned store instruction crossing a 4k page boundary at the same time as the lower page is invalidated might stall

Note that there is no change to the MIDR which remains at r4p0, but the REVIDR might be updated from 0x00 to indicate that one or more errata are corrected as shown above. Software will identify this release through the combination of MIDR and REVIDR.



## 2.3. Category A

## 2.4. Category A (Rare)

## 2.5. Category B

### **784420: Speculative instruction fetches with MMU disabled might not comply with architectural requirements**

#### **Category B**

**Products Affected: Cortex-A15 MP Core -NEON.**

**Present in: r4p0, r4p1.**

#### **Description**

When all applicable stages of translation are disabled, an ARMv7 processor must follow some architectural rules regarding speculative fetches and the addresses to which these can be initiated. These rules avoid potential reads to read-sensitive areas. For more information about these rules see the description of "Behavior of instruction fetches when all associated MMUs are disabled" in the ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition. Cortex-A15 normally operates with both the MMU and branch prediction enabled. If the processor operates in this condition for any significant amount of time, the BTB (branch target buffer) will contain branch predictions. If both stages of translation are then disabled, but branch prediction remains enabled, these stale BTB entries can cause A15 to violate the rules for speculative fetches.

Note: This erratum matches bug #5360 in the ARM internal JIRA database.

#### **Conditions**

The erratum can occur only if the following sequence of conditions is met:

- 1) MMU enabled for at least one stage of address translation
- 2) Branch prediction enabled
- 3) Branches executed
- 4) MMU disabled for all applicable stages of address translation

Note: When executing in a Non-secure PL1 or PL0 mode, for condition 1 at least one stage of address translation must be enabled, and for condition 4 both stages of address translation must be disabled. When executing in any other mode, there is only one stage of address translation, that must be enabled for condition 1 and disabled for condition 4.

#### **Implications**

If the above conditions occur, it is possible that after the MMU is disabled, speculative instruction fetches will occur to read-sensitive locations.

#### **Workaround**

Branch prediction should be disabled when the MMU is disabled after having been enabled. This should be done by clearing the appropriate Z bit in the System Control register at the same time as or just before the final stage of translation is disabled. Branch prediction should remain disabled until the MMU is enabled, or until the BTB has been flushed. On A15, the BPI\* branch predictor maintenance commands will not invalidate the BTB. The BTB can be flushed by setting bit 0 of the ACTLR register, doing any instruction cache invalidate instruction (e.g. ICIALLU), and then clearing bit 0 of the ACTLR register.

**785769: Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode****Category B****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1.****Description**

The Debug Communication Channel (DCC) registers are accessible using MCR/MRC instructions. LDC/STC instructions provide alternate access to DCC registers DBGDTRTXint/DBGDTRRXint.

In Non-debug state when DBGDSCR.UDCCdis is set to 1, then access to DCC register using MCR/MRC and LDC/STC instructions from User mode should generate an Undefined Instruction exception. Access using MCR/MRC instructions correctly generates an Undefined Instruction exception correctly. However access using LDC/STC instructions does not generate the Undefined Instruction exception and incorrectly accesses the register.

Note: This erratum matches bug #5372 in the ARM internal JIRA database.

**Conditions**

- 1) The processor is in Non-debug state and User mode.
- 2) DBGDSCR.UDCCdis is set to 1.
- 3) Either the Hypervisor trap to debug registers is not set (HDCR.TDA==0) or the processor is in Secure state.
- 4) LDC to DBGDTRTXint or STC to DBGDTRRXint is executed.

**Implications**

If LDC or STC instructions are executed in User mode, then DCC traffic between debug host and debug target from FIQ/IRQ/Supervisor/Monitor/Abort/Hypervisor/Undefined/System modes can be corrupted due to this errata.

**Workaround**

Tools must use MCR/MRC instructions to access Debug Communication Channel (DCC) registers from User mode, instead of LDC/STC instructions, in Non-debug state.

Alternatively, software can avoid corruption of DCC traffic by Non-secure User mode code by setting the hypervisor trap for debug register accesses (HDCR.TDA), and handling the LDC/STC instruction appropriately in hypervisor code. There is no software workaround to prevent LDC/STC instructions executing in Secure User mode from corrupting DCC traffic handled in other processor modes.

**813469: An unaligned store instruction crossing a 4k page boundary at the same time as the lower page is invalidated might stall****Category B****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0. Fixed in r4p1.****Description**

In a very unusual timing boundary condition, an unaligned store instruction crossing a 4k boundary at the same time as the lower page is being invalidated might stall until the next interrupt. If the store instruction is a VSTM of 88 bytes or larger, not 8-byte aligned, with 80 of the bytes in the lower page, the stall will not be broken by an interrupt and the core will stall until the next reset.

Note: This erratum matches bug #5469 in the ARM internal JIRA database.

**Conditions**

- 1) A store is executed that crosses a 4k boundary.
- 2) The store has the following alignment:
  - 2-byte store, not 2-byte aligned
  - 4-byte store, not 4-byte aligned
  - 8-byte or larger store, not 8-byte aligned
- 3) The lower page hits a valid translation in the L1DTLB.
- 4) The upper page misses in the L1DTLB and the table walk returns a translation fault.
- 5) While the TLB miss for the upper page is being serviced, the lower page translation is invalidated by a TLBI instruction from an ARM core.
- 6) The lower page misses in the L1DTLB and the table walk returns a translation fault.
  - All of the above conditions are required to hit the basic erratum, which will generally be broken by the next interrupt. To hit the stall that will not be broken by an interrupt, further conditions are required:
- 7) The store is a VSTM of 88 or more bytes, 4-byte aligned but not 8-byte aligned.
- 8) The first 80 bytes of the VSTM are in the lower page, with some of the bytes in the upper page.

**Implications**

If conditions 1-6 above occur with the correct timing, the CPU will stall until the next interrupt. If conditions 1-8 occur, that CPU will stall indefinitely, broken only by resetting that CPU.

The erratum will only occur in rare boundary conditions when an OS is invalidating a page that is actively being used by a process on another CPU and all the conditions are met. Because hitting the erratum is expected to be quite rare, in a system where the A15 CPUs get periodic interrupts the impact in most systems is likely to be minimal.

Compilers are not expected to generate an unaligned VSTM large enough to cause the indefinite stall. Optimized memcpy using Neon registers uses VSTR, not VSTM. Function prologues may use VSTM, but are not expected to store more than 64 bytes with normal calling conventions, even if the stack were not 8 byte aligned. Large VSTMs might occur in cases such as exception unwinding and state saving, but in those cases the VSTM will be 8-byte aligned and not affected. The VSTM indefinite stall case is therefore not expected to occur in a typical system.

**Workaround**

The suggested workaround is to avoid the use of large unaligned VSTM and to supply periodic interrupts to each CPU to clear the stall should it ever occur.

If large unaligned VSTM cannot be avoided, interrupts are not available, or a rare delay until the next interrupt is not acceptable, an alternative workaround is to do local TLB maintenance on each CPU rather than using the distributed TLB maintenance mechanism:

- Do TLB maintenance locally on each CPU using the local (not “inner shared”) TLB maintenance operations.

- Between TLB maintenance operations and the subsequent required DSB instruction, do not execute unaligned page crossing stores to the pages being invalidated.

**816470: DSB instruction in processor power down sequence may not drain all required transactions****Category B****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1.****Description**

When powering down an individual core in a Cortex-A15 MPCore processor it is possible that a DSB or DVM operation from another processor might not be completely flushed before power is removed. If the processor is powered down before the DSB or DVM operation completes the system might deadlock.

When executing a processor power-down sequence as specified in section 2.4.3 of the A15 TRM, the ACTLR.SMP bit is cleared, followed by an ISB and then a DSB to drain all required transactions from other processors before allowing the current processor to power down. On A15 r2 and earlier this sequence works correctly. However, on the r3 and later versions of A15, a performance optimization was added that allowed DSB instructions to execute more quickly if no DVM operations (TLB, branch predictor, or instruction cache maintenance operations) have been executed since the previous DSB. This faster version of the DSB does not perform all the actions required to correctly drain requests from other cores as required by the power-down sequence.

Note: This erratum matches bug #5472 in the ARM internal JIRA database.

**Conditions**

- 1) A DSB or DVM operation is executed on a processor (cpuA).
- 2) No DVM operation (TLB, branch predictor, or instruction cache maintenance operations) has been executed on cpuB since the last DSB instruction on a different processor (cpuA).
- 3) ACTLR.SMP bit is set to 0 followed by ISB/DSB on cpuB, which is being powered down.
- 4) cpuB is powered down.
- 5) The DSB or DVM operation is sent to cpuB and is dropped.

**Implications**

If the above conditions occur, the DSB or DVM operation will never complete and the system will stall until the next reset.

Whether or not the deadlock is seen will depend heavily on the timing of the power-down sequence. If there is a long period of time between clearing the ACTLR.SMP and the actual removal of power from the processor, it is very likely any in-flight operations from other cores will have time to complete.

**Workaround**

After clearing the ACTLR.SMP bit, and after the required ISB, execute a translation look-aside buffer entry invalidate instruction (TLBIMVA) before the required DSB. Any TLB address can be referenced by the instruction. This TLBIMVA will cause the DSB execution to drain all appropriate transactions from other processors and will prevent the deadlock.

**827671: WriteClean and WriteBack transaction reordering might cause data corruption****Category B****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1.****Description**

A cache maintenance operation (DCCMVAC or DCCSW instruction) that targets a dirty line in the L2 cache will generate a WriteClean transaction on the ACE interface. If a subsequent transaction causes a replacement of the line from the L2 cache then a WriteBack could be generated before the WriteClean operation completed. If the interconnect allows re-ordering between the WriteClean and WriteBack operations, data corruption might occur.

Note: This erratum matches bug #5492 in the ARM internal JIRA database.

**Configurations affected**

Systems supporting:

- Interconnect supporting re-ordering of WriteClean and WriteBack transactions
- BROADCASTCACHEMAINT=0
- BROADCASTCACHEMAINT=1 and DCCSW instruction used to clean lines from Cortex-A15

**Conditions**

- 1) Processor executes DCCMVAC or DCCSW to dirty line in Cortex-A15 cluster triggering WriteClean transaction on ACE write channel.
- 2) Processor write marks line dirty.
- 3) Subsequent linefill triggers WriteBack of line prior to WriteClean transaction completing.
- 4) WriteBack and WriteClean transactions are reordered causing memory data corruption.

Note: the CCI-400 and CCN-504 interconnects, and DMC-400 memory controller will not reorder the WriteClean and WriteBack transactions, hence will not exhibit the reordering behavior necessary for this erratum to occur.

**Implications**

If the above conditions exist, then the contents of the memory will be corrupted because the WriteClean and WriteBack requests are presented to the slave in a different order than issued on the ACE write channel.

**Workaround**

This erratum can be avoided by setting ACTLR2[0] = 1. This bit turns all DCCMVAC and DCCSW operations into the corresponding clean and invalidate operations (DCCIMVAC and DCCISW) which are not vulnerable to this erratum.

**836969: Code sequence continuously hitting the L1 cache can block snoop****Category B****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1.****Description**

If a sequence of memory operations is executed such that a single tag bank or a single data bank of the L1 data cache is accessed every cycle, and a store instruction is executed periodically (at least once every 32 cycles), a snoop to a physical address that is unrelated to the load and store instructions might be blocked until there is a single cycle where a load is not accessing that tag or data bank. This might block the snoop until the current polling loop finishes or until the next interrupt or other exception.

Note: This erratum matches bug #5500 in the ARM internal JIRA database.

**Configurations affected**

All configurations are affected.

**Conditions**

- 1) A continuous stream of load instructions hitting in the L1 data cache every cycle.
- 2) The load physical addresses (PA) must meet the following criteria:
  - PA[7:6] for all loads are the same (same tag bank) OR
  - PA[5:4] for all loads are the same (same data bank)
- 3) A store that hits the L1 data cache must occur at least once every 32 cycles.
- 4) A snoop (from the interconnect or another Cortex-A15 core) hits the L2 cache while L2 cache is waiting for the eviction of this cacheline from L1 data cache.
- 5) A software dependency that means the continuous stream of loads continues until the snoop completes.

**Implications**

If all conditions are met, the snoop could be held off until the next interrupt. This would only happen if the processor is executing an unusual polling loop containing a store waiting for a cacheable memory location to be updated. To hit the condition, the polling loop would need to contain at least two load instructions and a store instruction, all hitting the L1 data cache. The conditions cannot be met if the loop contains any load exclusive, WFE, DMB, or DSB instructions.

**Workaround**

Untrusted or user code must be run with periodic timer interrupts, that will prevent the erratum from causing a deadlock. Alternatively, if a software polling loop is found to be hitting this erratum, simplifying the polling loop to contain only the loads that are checking the required conditions and delaying any store instructions until after the conditions have been met will avoid the erratum.

## 2.6. Category B (Rare)

### 763126: Three processor exclusive access livelock

#### Category B Rare

**Products Affected:** Cortex-A15 MP Core -NEON.

**Present in:** r4p0, r4p1.

#### Description

In a system with three or more coherent masters that all use the ldrex/strex synchronization primitives to access a semaphore in coherent cacheable memory, there is a possibility of a livelock condition where two masters continuously attempt and fail to get the lock while the third master continuously reads the lock.

This erratum is heavily dependent on a unique set of initial conditions, and upon specific interconnect timing once the livelock has started. It is expected to be rare in a real system that the timing conditions will be hit.

An example: two cores C1 and C2 are contending for a lock using ldrex/strex, and core C3 is looping reading the same semaphore location. Once the livelock condition has started, from the perspective of C1, the sequence will look like this:

- 1) Execute ldrex, hits the cache in unique state.
- 2) External snoop takes line to shared state (triggered by C3 read).
- 3) Execute instructions to process the ldrex result and prepare the strex data.
- 4) Execute strex, hits cache shared, issues readUnique to bring in line unique.
- 5) External snoop invalidates line, clearing monitor (triggered by C2 strex that will eventually fail the monitor).
- 6) Line returns in unique state, but strex fails due to cleared monitor.
- 7) Loop back to step 1.

C1 and C2 constantly issue ReadUniques due to failing store exclusives that invalidate the line in the other core, each core causing the others strex to fail without making forward progress. No forward progress is made until/unless one of the cores stops (possibly due to an interrupt) or interconnect timing happens to allow enough time for one of them to complete.

NOTE: this erratum is describing additional limitations on exclusives loads and stores in a multi-core system including A15. There is no plan to fix this erratum on future A15 cores, as reasonable code following the ARM architecture guidelines should not be affected.

NOTE: This erratum matches bug #4637 in the ARM internal JIRA database.

#### Conditions

- 1) One master continuously reading the location of the semaphore.
- 2) Two masters doing a ldrex/strex loop to the semaphore.
- 3) Semaphore in write-back shared memory.
- 4) Three master system (3+ core A15, or 3+ total processors in the system over ACE).

#### Implications

Neither C1 nor C2 will ever succeed in gaining the lock. Software could stop making progress. An interrupt to one of the cores C1/C2/C3 would likely break the livelock.

#### Workaround

If there are no more than two coherent masters in the system, no workaround is needed, the issue will not be seen.

The latest version of the ACE specification adds additional command types and system logic to allow processors to avoid this issue. This specification update was not available in time for A15 to take advantage of it and A15 does not implement this ACE feature. As an alternative, A15 installed hardware in each processor to detect that the load/store exclusive livelock scenario may be occurring and delay snoops for a period of time to allow the load exclusive/store exclusive loop to complete and make forward progress. With this fix, no existing code that uses ldrex/strex should need to be rewritten if it follows the ARM Architecture Reference Manual guidelines in the “A3.4 Synchronization and Semaphores” section and is not unreasonably long.



To enable this hardware on Cortex-A15 you must set the "Snoop-delayed exclusive handling" bit in the Auxiliary Control Register, ACTLR[31] to 1. The reset value of ACTLR[31] is 0 for all product revisions r0pX, r1pX, r2pX, r3p0, r3p1 and r3p2. This reset value is 1 for product revision r3p3 and beyond.

Note: all references to "ldrex" encompass all Load-Exclusive instructions and "strex" encompass all Store-Exclusive instructions.

**812169: DVM message blocked by copyback on AMBA Chi interconnect****Category B Rare****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1.****Description**

This erratum does not apply to any A15 that is using the standard ACE/AXI interconnect. It only applies to an A15 connected to the CCN-504 Chi interconnect through the SBAS bridge.

If a DVM transaction from the CCN-504 interconnect is sent down the L2 pipeline and the L2 Fill Evict Queue (FEQ) is full, the DVM transaction will be cancelled and reissued. If that DVM transaction also happens to have an address field that matches a copyback (WriteBack, WriteClean, or Evict) in the FEQ, the snoop logic might also detect a hazard that forces a dependency between the DVM and the copyback. Instead of immediately reissuing, the DVM transaction will be held off until the copyback completes.

On a general ACE interconnect this is not a problem, because copyback writes must make forward progress. However, in the Chi interconnect it is possible for that copyback to have a dependency on the DVM transaction completing. The copyback might not complete until the A15 completes the DVM request. This can lead to a system deadlock.

Note: This erratum matches bug #5458 in the ARM internal JIRA database.

**Configurations affected**

Only affects A15 when used with CCN-504

**Conditions**

- 1) A DVM request (DVM1) is received on the AR channel from the CCN-504 interconnect.
- 2) DVM1 is processed when the FEQ is full.
- 3) DVM1 address matches the address of a WriteBack, WriteClean, or Evict (WB1) in the FEQ WB1 has started to issue, but has not completed sending out its data on the W channel.
- 4) WB1 is unable to issue its data due to back pressure from the bridge on the write channel. The back pressure will not release until DVM1 completes.

**Implications**

This erratum might very rarely lead to a system deadlock.

**Workaround**

To avoid this erratum, A15 configurations used with CCN-504 interconnect must set L2ACTLR[7]. This will enable a timeout mechanism which will allow the DVM request to be reissued even if the copyback has not completed.

When an A15 is used with the CCN-504 interconnect the L2ACTLR must be already configured as specified in the CCN-504 Technical Reference Manual. This workaround sets one additional bit in that register.

**827923: TLB maintenance operations might not be synchronized by DSB instruction****Category B Rare****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0. Fixed in r4p1.****Description**

A TLB invalidate instruction followed by a DSB instruction to ensure its completion, should remove all uses of the old translation from the system. On Cortex-A15 the DSB instruction might complete before some store memory transactions with the invalidated translation are globally observed. This affects store memory transactions with write-back-no-allocate (due to page attributes or streaming stores), write-through, non-cacheable, device, or strongly-ordered memory attributes. This does not affect write-back-read-write-allocate stores.

This might lead to data corruption if the software attempts to access or allows access to the physical memory of the invalidated or moved region before the stores have completed.

Note: This erratum matches bugs #5493 in the ARM internal JIRA database.

**Configurations affected**

Multi-cluster systems that utilize distributed virtual memory (DVM) transactions.

**Conditions**

- 1) Software modifies the translation tables to invalidate, restrict the permissions of, or change the physical address of a page or block.
- 2) Software executes a TLB invalidate instruction to invalidate any TLB entries holding the previous translation.
- 3) Software executes a DSB instruction to ensure completion of the TLB invalidate instruction and global observation of any memory transactions that depended on the previous translation.

**Implications**

When software changes the translation tables and invalidates a modified TLB entry, outstanding stores to the old translation might not be globally observed. This might lead to data corruption if the physical memory of the invalidated memory region is accessed before the memory requests that used the old translation are complete.

**Workaround**

By following the workaround below, this erratum will not affect cacheable operations; therefore, it becomes a Category C erratum that only affects store transactions with write-through, non-cacheable, device, or strongly-ordered memory attributes. Note that disabling write streaming will have an adverse effect on the performance of memset and memcpy operations.

- 1) Do not use write-back-no-allocate page table mappings.
- 2) Set ACTLR[26:25] = 2'b11 // disable write streaming no L1-allocate threshold.
- 3) Set ACTLR[28:27] = 2'b11 // disable write streaming no-allocate threshold.

## 2.7. Category C

### **773023:** Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI

#### **Category C**

**Products Affected:** Cortex-A15 MP Core -NEON.

**Present in:** r4p0, r4p1.

#### **Description**

A15 maintains order between Strongly Ordered (SO) memory requests as required by the ARM architecture memory ordering model. This is done inside the A15 by use of internal ordering logic. On the interconnect, A15 enforces ordering by using the same ARID/AWID value for all SO memory requests from a given processor (guaranteeing read/read and write/write ordering) and by waiting for the completion of all SO and Device memory requests on one channel before issuing SO or Device requests from the same core on the other channel (guaranteeing read/write and write/read ordering).

A15 does the same for Device memory.

However, A15 uses different ARID and AWID for Device memory requests from a given CPU and Strongly Ordered memory requests from the same CPU. Due to this fact, it is possible that the an SO read from a given CPU could pass a Device read from the same CPU and arrive at a single peripheral out of order.

#### **Conditions**

- 1) A system has memory mapped peripherals larger than 4KB
- 2) Some of the pages mapped to that peripheral are mapped Strongly Ordered and some are mapped Device
- 3) Software depends upon ordering of these Strongly Ordered and Device memory requests

#### **Implications**

This is not an issue for any device that fits in one 4KB memory page, as it is only possible to have a single memory type for that page (SO/Dev aliasing is not allowed).

For larger peripherals, it is possible that Strongly Ordered or Device transactions could arrive at the peripheral out of order.

#### **Workaround**

A given peripheral device should be mapped to all Strongly Ordered or all Device memory.

**777769: ICache parity error may not be corrected for NC code****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1****Description**

If an instruction fetch to a non-cacheable page in memory gets a false hit on a line in the instruction cache due to a parity error in the instruction cache tag array, incorrect instructions may be executed.

Note: This erratum matches bug #5312 in the ARM internal JIRA database.

**Conditions**

- 1) MMU enabled
- 2) Instruction fetch to page marked SO/Dev/Normal-Non-Cacheable
- 3) Parity error in instruction cache tag
- 4) Corrupted tag matches the physical address of the cache line being fetched

**Implications**

In an A15 configured with L1 parity/ECC and with parity/ECC checking enabled, in very rare circumstances a parity error can cause delivery of bad instructions while executing non-cacheable code. This is not expected to be an issue in normal systems as no normal programs will have instructions in non-cacheable memory with the MMU enabled. At boot, or any other time that the MMU is disabled, the erratum will not occur.

**Workaround**

Place instructions in cacheable memory whenever possible. If you must run non-cacheable code with the MMU enabled, first invalidate the instruction cache.

**810072: When a single-bit ECC error occurs in the L2, uncorrected data might be returned****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0. Fixed in r4p1.****Description**

In the case of an unusual timing boundary condition, a single-bit ECC error in the L2 data array might not be corrected properly.

This erratum cannot occur if ECC is not configured or is not enabled.

Note: This erratum matches bug #5464 in the ARM internal JIRA database.

**Conditions**

- 1) A cache line fill to address A (CLF\_A) hits the L2 and returns data to the L1 data cache.
- 2) The first 16-byte beat of data returning for CLF\_A has a single-bit ECC error in the data. Write streaming is enabled (ACTLR[28:25] != b1111) in the L1.
- 3) A full cache line-streaming write to address B (WR\_B) is ready to issue to the L2.
- 4) WR\_B begins to issue during the same cycle the CLF\_A incorrect data returns.
- 5) A store is in flight (ST\_C) to the cache line of address A, or to the cache line being replaced by CLF\_A.

**Implications**

Data corruption might occur if the erratum conditions exist. A single-bit error in the L2 data array might lead to incorrect data in the L1 data cache. A very small percentage of L2 data array single-bit ECC errors will be affected, because the error must be in the critical 16 bytes, must hit a narrow timing window, and must occur when store streaming is pushing full cache line writes to the L2 at the same time as a non-streaming store is hitting the cache line fill or the same set/way.

The result is a very small increase in silent data corruption (SDC) failure in time (FIT) rate in the presence of L2 data array errors.

**Workaround**

There is no workaround.

**816469: Double-bit ECC error during hardware correction of a single-bit ECC error might cause data corruption****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0. Fixed in r4p1.****Description**

If hardware detects a single-bit or double-bit ECC error on any RAM protected by ECC, the hardware initiates a sequence to correct the error. In the case of a single-bit ECC error, a read-modified-write sequence is generated to correct the single-bit error and write the corrected data back to the RAM. In the case of a double-bit error, the nINTERRIRQ signal is de-asserted signaling the presence of the double-bit error and the hardware writes the entry to resolve the error. If the double-bit error was associated with the L2 tag RAM or the L2 snoop tag RAM, the entry is invalidated. If the error was associated with the L2 dirty RAM then the entry is marked clean.

This erratum describes a condition when, after a single-bit ECC error detection, a double-bit ECC error is detected while the hardware performs the read-modified-write correction sequence. If this condition occurs, then nINTERRIRQ is not de-asserted to report the double-bit ECC error. The hardware correction sequence then proceeds to write the RAM to correct the error and might cause data corruption. Depending on the RAM associated with the error, corruption could occur to either the data, tag address, inclusion indicator or dirty status. This corrupted data or state could then be used by a subsequent transaction which might cause unpredictable results.

Note: This erratum matches bug #5471 in the ARM internal JIRA database.

**Conditions**

- 1) A single-bit ECC error detected on L2 RAM.
- 2) A double-bit ECC error detected on the same entry during hardware read-modified-write correction sequence.

**Implications**

If the above conditions occur, the RAM contents will be corrupted as the syndrome bits will be recalculated using the corrupted data and RAM updated with this corrupted data.

It is unpredictable as to the behavior of the system following this data corruption as it affects all of the RAMs in the L2 memory system which are protected by ECC.

If a double-bit error was detected initially there is no issue, because the hardware properly handles this case. If a single-bit error was detected initially and no double-bit error detected during the correction sequence there is no issue. The correction sequence duration is configuration controlled, but completes in less than 100 cycles, so this condition of a double-bit error being generated following a single-bit error detection is considered rare. For the majority of applications this is a minor error. In some applications the error might be more significant.

**Workaround**

There is no workaround.

**822719: Read following a write of a Timer TVAL register might return incorrect value****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1.****Description**

In Cortex-A15, the computation and update to the Timer CompareValue is done in one cycle, and the update to the read TimerValue is done in the next cycle. A write to a TVAL register updates the CompareValue. If a TVAL register is written and immediately followed by a read of the same TVAL register, there is insufficient time for the TimerValue to be updated and the wrong TimerValue is returned.

Note: This erratum matches bug #5480 in the ARM internal JIRA database.

**Conditions**

- 1) A write is performed to a Timer TVAL register.
- 2) A read is performed to the same Timer TVAL register in the very next cycle.

**Implications**

If the above conditions are met, the read will return the incorrect (old) TimerValue.

**Workaround**

Insert an ISB between the write and read. Only one cycle is needed between the two accesses to allow time for the TimerValue to be updated.



**826375: Debug accesses in User mode do not properly generate undefined instruction exceptions for some SIMD and VFP registers****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1****Description**

Accessing any of the Advanced SIMD and Floating-point Extension system registers (VMRS FPSID, VMRS MVFR0, VMRS MVFR1, VMRS/VMSR FPSCR, VMRS/VMSR FPEXC) while in User mode and in Debug state should generate an Undefined Instruction exception. Instead, these instructions are executed.

Note: This erratum matches bug #5488 in the ARM internal JIRA database.

**Conditions**

- 1) The processor is in User mode and Debug state.
- 2) Execution of any of these instructions issued through the Instruction Transfer Register (DBGITR):
  - VMRS FPSID
  - VMRS MVFR0
  - VMRS MVFR1
  - VMRS FPEXC
  - VMSR FPEXC
  - VMRS FPSCR (with FPEXC.EN==0)
  - VMSR FPSCR (with FPEXC.EN==0)

**Implications**

If the above conditions occur, the processor will perform the specified system register accesses instead of generating an Undefined Instruction exception.

(When FPSCR is accessed from User mode with FPEXC.EN==1 the expected behavior is to perform the access, and Cortex-A15 MP Core does behave that way).

**Workaround**

There is no workaround.

**826969: Cortex-A15 might violate read-after-read memory ordering on a load forwarding from a store crossing a 16-byte boundary****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0. Fixed in r4p1.****Description**

Cortex-A15 might violate the read-after-read memory ordering requirement when a load forwards data from an older store which crosses a 16-byte boundary but not a 64-byte boundary. When a store instruction crosses a 16-byte boundary but not a 64-byte boundary, it is possible for the lower bytes to update the cache before the upper bytes update the cache. If the lower bytes update the L1 data cache, the line is then evicted from the L1 and modified by another master, and the line is brought back into the L1 data cache before the upper bytes are processed, there is a window where two loads that access the lower bytes of the store could violate the read-after-read memory ordering requirement. Specifically, an earlier load might see the data written by the other master, and the later load might see the data written by the store on the local processor.

Note: This erratum matches bug #5491 in the ARM internal Jira database.

**Configurations affected**

All configurations are affected.

**Conditions**

- 1) A store (ST1) is executed which crosses a 16-byte boundary but not a 64-byte boundary, modifying one or more bytes (A) below the 16-byte boundary and one or more bytes (B) above the 16-byte boundary.
- 2) After the lower bytes (A) of the store are written into cache, the cacheline is evicted or snooped out of L1D before the upper bytes of the store (B) update the cacheline.
- 3) The lower bytes (A) are modified by another memory master or processor (ST2).
- 4) The cache line is brought back into the L1 data cache (possibly by a load or preload instruction).
- 5) Two younger loads (LD1 and LD2) are executed that access the lower bytes modified by the store (A) before the upper bytes are written to the cache.

**Implications**

If the above conditions occur, it is possible that LD1 will return the modified data from the other master (ST2), but LD2 will see the data from the local store (ST1). This violates the following requirement in the ARM architecture specification (known as read-after-read ordering):

"It is impossible for an observer in the shareability domain of a memory location to observe two reads to the same memory location performed by the same observer in an order that would not occur in a sequential execution of a program."

The read after read ordering is significant in situations where the reads by one processor are racing in an unsynchronized manner with a write to the same location by a different processor. In these cases, it is expected that the algorithms involved will be relying on the write being seen as occurring in a single-copy atomic manner, such that the reads will see either the old or the new value, and not some amalgamation of the two. For this erratum to be observed, the write must span a 16-byte boundary, and so is not required architecturally to be single-copy atomic. Correspondingly it is very hard to envisage any situation where this erratum will be significant, and so it is characterized as being Category C. Additionally, this erratum has been present in volume shipping devices (Cortex-A15) without being observed in the field.

**Workaround**

No workaround is necessary.

**832972: HSTR.{T7,T8,T15} bits incorrectly trap CDP instructions****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1. Open.****Description**

HSTR.Tx bits are intended to trap MCR or MRC instructions with CRn set to cx and MCRR or MRRC instructions with CRm set to cx. For  $x = \{7,8,15\}$ , the trap bits do function in this capacity but also trap CDP/CDP2 instructions with CRn set to cx.

Note: This erratum matches bug #5496 in the ARM internal JIRA database.

**Configurations affected**

All configurations are affected.

**Conditions**

- 1) HSTR.Tx is set to 1.
- 2) A Non-secure CDP or CDP2 instruction with CRn set to x is executed in PL1 or PL0 where  $x = \{7,8,15\}$ .

**Implications**

If the above conditions are met, the CDP/CDP2 instruction will be erroneously trapped to Hyp mode. The correct behavior is to take an UNDEF exception.

**Workaround**

There is no workaround.

**834569: PMU event BUS\_CYCLES might be incorrect in some cases****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1. Open.****Description**

PMU event 0x1D (BUS\_CYCLES) might be incorrect in some scenarios.

Note: This erratum matches bug #5497 in the ARM internal JIRA database.

**Configurations affected**

All configurations affected.

**Conditions**

- 1) L2 clock is gated after 256 cycles of inactivity.
- 2) BUS\_CYCLES event is read.

**Implications**

If the above conditions are met, the BUS\_CYCLES count PMU event will be incorrect.

**Workaround**

If accurate BUS\_CYCLES counts are required during periods of L2 inactivity, dynamic clock gating of the L2 logic can be disabled by setting L2ACTLR\_EL1[27] = 1'b1. However, this is not recommended for normal operation because it will result in increased power consumption.

**842119: Instruction issued through ITR by debugger executes incorrectly for PCLKDBG frequencies much slower than the processor****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1. Open.****Description**

When the processor is in Debug state, the debugger can execute instructions on the processor using Instruction Transfer Register, DBGITR. Under certain conditions when the PCLKDBG clock frequency is considerably slower than the processor clock frequency, the processor might incorrectly execute the same instruction more than once.

Note: This erratum matches bug #5501 in the ARM internal Jira database.

**Configurations affected**

All configurations where PCLKDBG clock frequency is slower than one fourth of processor clock frequency are affected.

**Conditions**

- 1) The processor is currently in Debug state.
- 2) Debug Status and Control Register, DBGDSCR.ExtDCCmode[1:0] is set to 0b01, that is, Stall mode.
- 3) The external debugger transfers an ARM instruction to the processor for execution by writing to Instruction Transfer Register, DBGITR.

**Implications**

If the above conditions are met, then the processor might incorrectly execute the contents of the Instruction Transfer Register, DBGITR more than once. This can have unintended side effects, for example additional memory transactions from the processor or incorrect execution of store exclusive instructions.

**Workaround**

Software Workaround: The debugger needs to avoid using Stall mode (DBGDSCR.ExtDCCmode[1:0]=0b01) and alternately use Non-blocking mode (DBGDSCR.ExtDCCmode[1:0]=0b00) or Fast mode (DBGDSCR.ExtDCCmode[1:0]=0b10) when an ARM instruction is transferred to DBGITR for execution on the processor.

Hardware Workaround: Run the PCLKDBG clock frequency faster than or equal to one fourth of the processor clock frequency.

**851024: Persistent evictions combined with interconnect backpressure might stall Write-Back No-Allocate stores****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1. Open.****Description**

In a coherent ACE system, a Write-Back No-Allocate (WBNA) store might be stalled if WriteUnique/WriteLineUnique (WU/WLU) transactions are enabled and the store is attempted when one or more cache evictions are pending. ACE requires that WU/WLU transactions do not bypass any outstanding evict type transactions (WriteBack/WriteEvict/WriteClean). To satisfy this requirement, a microarchitectural hazard is used to force a replay if a WU/WLU transaction is attempted when an eviction is pending. In rare scenarios with a persistent stream of L2 cache linefills and associated evictions, combined with significant backpressure in the interconnect, and with specific timing, it is possible for a WBNA store to be stalled indefinitely.

Note: This erratum corresponds to issue #5504 in the ARM internal tracking system

**Configurations affected**

Coherent ACE systems that enable WriteUnique/WriteLineUnique transactions.

**Conditions**

- 1) WriteUnique/WriteLineUnique transactions are enabled by setting L2ACTLR[4] to 1'b0. This is the reset value.
- 2) A Cortex-A15 processor issues a Write-Back No-Allocate store (OP1). This can be a streaming store that was downgraded to Write-Back No-Allocate by the processor.
- 3) There is a pending eviction, forcing (OP1) to stall because of ACE requirements that require outstanding evictions to complete before WriteUnique/WriteLineUnique stores are performed.
- 4) A continuous stream of L2 cache linefills occurs, from other cores and/or prefetch, which triggers new evictions.
- 5) There is significant sustained backpressure in the interconnect, which keeps the system backed up and the ACE write channel queue near full.
- 6) Specific arbitration and timing conditions exist which, when combined with condition 5), trigger a microarchitectural hazard that causes condition 3) to repeat.

**Implications**

If the above conditions are met, (OP1) will stall until the specific timing conditions and backpressure in the L2 subsystem are relieved. Interrupts and barriers after the Write-Back No-Allocate store are also delayed until the store completes. The conditions for this erratum are rare and not expected to significantly impact real system performance.

**Workaround**

If WriteUnique/WriteLineUnique transactions are not required, disable them by setting L2ACTLR[4] = 1'b1. Otherwise, set L2ACTLR[7] = 1'b1 to enable L2 hazard detection timeout. This will force the L2 cache to periodically re-evaluate hazards, at which point the stall will be released.

**854671: Extremely slow response for unaligned loads crossing a 64-byte cacheline boundary might cause the processor to deadlock****Category C****Products Affected: Cortex-A15 MP Core -NEON.****Present in: r4p0, r4p1. Open.****Description**

An unaligned load cache miss, which crosses a 64-byte cacheline boundary, issues two transactions to the interconnect. If the memory attributes are normal Non-Cacheable (NC), Write-Through (WT) or Write-Back No-Allocate (WBNA), such that the data will not be cached in the Cortex-A15, and the read responses are extremely slow then it might be possible for the Cortex-A15 to continually reissue the loads which causes the processor executing the unaligned load to deadlock.

Accesses to normal Write-Back Read-Allocate or Write-Back Write-Allocate (WBRWA) are not affected by this erratum. However, if the cache is disabled by setting SCTL.R.C=0, the Cortex-A15 treats WBRWA as WBNA. Therefore any normal memory access would be subject to this erratum if the MMU was enabled and the cache disabled.

If the MMU is disabled by setting SCTL.R.M=0, then all data accesses are treated as Strongly-Ordered (SO), which are not affected by this erratum.

Note: This erratum corresponds to issue #5505 in the ARM internal tracking system.

**Configurations affected**

All configurations are affected.

**Conditions**

1. MMU is enabled by setting SCTL.R.M=1.
2. The Cortex-A15 Processor (PROC0) executes unaligned load to normal NC, WT, WBNA memory or to normal WBRWA memory with the cache disabled by setting SCTL.R.C=0 which crosses a 64-byte cacheline boundary.
3. Two read transactions are issued to the interconnect.
4. Read responses are extremely slow (for example, > 10K cycles).
5. PROC0 reissues the read transactions following the slow read responses.

**Implications**

If the above conditions are met, in unusual circumstances where systems generate extremely slow memory response latencies for read transactions, the PROC0 reissues the unaligned load in condition (2) resulting in processor deadlock.

**Workaround**

For systems which have slaves that support extremely slow response times, these should be mapped to Strongly-Ordered or Device memory. If the memory cannot be mapped to Strongly-Ordered or Device memory, then it must be mapped to normal WBRWA memory and any unaligned access should only be generated after enabling the cache (SCTL.R.C=1).