ARM Cortex™-A15 MPCore - NEON

Product Revision r0p4

Software Developers Errata Notice

Non-Confidential - Released

ARM

Software Developers Errata Notice

Non-Confidential Proprietary Notice

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 . Give:

- the document title
- the document number, ARM-EPM-025235
- 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 affect the r0p4 product revision.

20 Feb 2015: Changes in Document v14

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

04 Nov 2014: Changes in Document v13

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

09 Sept 2014: Changes in Document v12

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

08 Aug 2014: Changes in Document v11

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
36	New	830321	Prog	CatB		Cortex-A15 might falsely trigger a watchpoint exception on a CLREX instruction
67	New	832972	Prog	CatC		HSTR.{T7,T8,T15} bits incorrectly trap CDP instructions

07 May 2014: Changes in Document v10

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
65	New	826375	Prog	CatC		Debug accesses in User mode do not properly generate undefined instruction exceptions for some SIMD and VFP registers
66	New	826969	Prog	CatC		Cortex-A15 might violate read-after-read memory ordering on a load forwarding from a store crossing a 16-byte boundary

10 March 2014: Changes in Document v9

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

11 Dec 2013: Changes in Document v8

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

01 Nov 2013: Changes in Document v7

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

21 Aug 2013: Changes in Document v6

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
62	New	810072	Prog	CatC		When a single-bit ECC error occurs in the L2, uncorrected data might be returned
60	Updated	803670	Prog	CatC		Unaligned load accessing two cache lines could return uncorrected data in case of single bit ECC error in the L2 cache

05 June 2013: Changes in Document v5

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
33	New	804622	Prog	CatB		Extension register load to SO/Dev address combined with erroneous BLX can stall core
41	New	804969	Prog	CatB	Rare	Extension register loads and stores can livelock core

29	Updated	798181	Prog	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

24 Apr 2013: Changes in Document v4

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
59	New	803620	Prog	CatC		Near zero Advanced SIMD fused multiply add may be incorrectly flushed to zero
60	New	803670	Prog	CatC		Unaligned load accessing two cache lines could return uncorrected data in case of single bit ECC error in the L2 cache
61	New	803671	Prog	CatC		The ACE ARDOMAIN field may be incorrect on some Instruction or Tablewalk prefetch requests

21 Mar 2013: Changes in Document v3

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
14	New	801819	Prog	CatA	Rare	An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing
29	New	798181	Prog	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

31 Jan 2013: Changes in Document v2

Page	Status	ID	Area	Cat	Rare	Summary of Erratum
11	New	799271	Prog	CatA		A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock
31	New	799270	Prog	CatB		Writing ACTLR.SMP when the L2 cache has been idle for an extended period may not work correctly
38	Updated	763126	Prog	CatB	Rare	Three processor exclusive access livelock

23 Oct 2012: Changes in Document v1

First issue of the r0p4 Software Developers Errata Notice

Contents

Chapter 1.
Introduction

This chapter introduces the errata notice for the ARM Cortex-A15 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 r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n* Identifies the major revision of the product.

p*n* 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 r0p4 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	r0p4
794724	CatA		L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time	X
799271	CatA		A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock	X
773769	CatA	Rare	Large data RAM latencies can lead to rare data corruption	X
775619	CatA	Rare	L2 may deadlock or corrupt data if all ways in a single set become blocked from replacement	X
801819	CatA	Rare	An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing	X
766170	CatB		HCR.FB doesn't upgrade ICIALLU to ICIALLUIS	X
766171	CatB		TLBINV followed by changing VTTBR VMID value could cause incorrect TLB invalidation	X
766173	CatB		Prefetch hit by WB-No Allocate load can incorrectly set inclusion bit	X
766421	CatB		Strongly-Ordered/Device load or NC LDREX could return incorrect data	X
766422	CatB		Thumb store translation fault to Hypervisor may not have correct HSR Rt value	X
769270	CatB		L2 Tag Latency = 3 should not be used in a system configured with Tag Slice	X
773022	CatB		Incorrect instructions may be executed from loop buffer	X
774569	CatB		Watchpoint may not be taken on second half of unaligned ld/st crossing a 64-byte aligned boundary	X
774769	CatB		Data corruption may occur with store streaming in a system	X
777770	CatB		ESR incorrect for AdvSID HCPTR trapped inst in Hyp mode	X

ID	Cat	Rare	Summary of Erratum	r0p4
784420	CatB		Speculative instruction fetches with MMU disabled might not comply with architectural requirements	X
784477	CatB		CTIINTACK register needs clearing each time it is set	X
785769	CatB		Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode	X
798181	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior	X
799270	CatB		Writing ACTLR.SMP when the L2 cache has been idle for an extended period may not work correctly	X
804622	CatB		Extension register load to SO/Dev address combined with erroneous BLX can stall core	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
830321	CatB		Cortex-A15 might falsely trigger a watchpoint exception on a CLREX instruction	X
836969	CatB		Code sequence continuously hitting the L1 cache can block snoop	X
763126	CatB	Rare	Three processor exclusive access livelock	X
773319	CatB	Rare	Unaligned page boundary crossing load hit by TLB invalidate can stall until next interrupt	X
804969	CatB	Rare	Extension register loads and stores can livelock core	X
766174	CatC		ID_PFR0 Jazelle extension support incorrectly encoded	X
766419	CatC		Benign op instead of UNDEFINED in certain unused opcodes	X
768529	CatC		Debug version of ID_PFR0 Jazelle extension support incorrectly encoded	X
768532	CatC		Mapping the GIC memory mapped region as cacheable can result in unpredictable behavior or a deadlock	X
770320	CatC		Single bit ECC error can cause cache maintenance operation to violate memory ordering	X
773023	CatC		Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI	X
774570	CatC		Fault Status bit in register DBGDSCR is implemented as sticky	X
774571	CatC		Sampling registers DBGCIDSR and DBGVIDSR can return incorrect value	X
775622	CatC		Debug version of Cache Type Register (CTR) IminLine field are incorrectly encoded	X
777769	CatC		ICache parity error may not be corrected for NC code	X
777771	CatC		Ordering of read accesses to the same memory location may not be ensured in the case of an unaligned load	X
777772	CatC		Device LDM may stall if memory type changed asynchronously from Device to Normal	X
777774	CatC		DCCMVAC/DCCSW colliding with ReadOnce from ACE could cause data corruption	X
780121	CatC		PTM might not acknowledge a trace flush request when cpu is in WFI.	X
784419	CatC		An unaligned load crossing a 4K boundary between SO/Dev memory and WB memory can stall the core until the next interrupt	X
784469	CatC		CTI Authentication Status register is incorrect	X
803620	CatC		Near zero Advanced SIMD fused multiply add may be incorrectly flushed to zero	X
803670	CatC		Unaligned load accessing two cache lines could return uncorrected data in case of single bit ECC error in the L2 cache	X
803671	CatC		The ACE ARDOMAIN field may be incorrect on some Instruction or Tablewalk prefetch requests	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
826375	CatC		Debug accesses in User mode do not properly generate undefined instruction exceptions for some SIMD and VFP registers	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
834569	CatC		PMU event BUS_CYCLES might be incorrect in some cases	X
842119	CatC		Instruction issued through ITR by debugger executes incorrectly for PCLKDBG frequencies much slower than the processor	X

2.3. Category A

794724: L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If a Cortex-A15 MPCore implementation uses an L2 tag RAM that requires a two cycle setup time for address and data, the L2 cache might not be correctly invalidated by the hardware initialization sequence that occurs after the deassertion of nL2RESET.

By default, the Cortex-A15 MPCore L2 tag RAM input paths are single cycle paths for both setup and hold. However, if the L2 tag RAM used in an implementation requires it, software can program L2CTLR[9] to 1 to configure the setup paths to be two cycle multicycle paths. These must be set correctly before the data cache is enabled.

Cortex-A15 MPCore initializes the L2 cache in hardware when the L2 is reset. Because this hardware initialization occurs before the L2CTLR register can be programmed by software, the hardware sequence must assume tag RAM array timings that will work with all legal RAM instances.

During hardware RAM initialization of the L2 on affected versions of Cortex-A15 MPCore the setup used for the L2 tag RAM is a single cycle. It should be two cycles to support RAMs that require more setup.

Note: This erratum matches bug #5403 in the ARM internal JIRA database.

Configurations Affected

implementations that require L2CTLR[9] to be set to 1 because the L2 tag RAM requires two cycle setup.

Conditions

This erratum requires the following condition:

- The implementation uses an L2 tag RAM that requires two cycle setup on address and data. Affected implementations will program L2CTLR[9]=1 before enabling the data cache.

Implications

The L2 Cache may not be correctly initialized after deassertion of nL2RESET.

1) Valid lines with unknown addresses and data might be present in the cache after reset. If the valid lines match any address in physical memory that is accessed before the lines are evicted, they could deliver incorrect data on reads.

2) If the lines appear valid and dirty, they could generate spurious memory transactions that would corrupt memory or generate errors in the system.

Workaround

Using a lower frequency such that the tag RAM instance can meet timing without the two cycle setup is a valid system workaround. The lower frequency need only be used between deassertion of nL2RESET and the completion of the cache initialization sequence. Software can ensure that the hardware initialization sequence has completed by executing any data cache maintenance operation (e.g. DCCISW) followed by a DSB. The maintenance operation will not complete until the L2 initialization is complete.

Software initialization of the L2 cache is not a valid workaround. The DCISW invalidate by set/way instruction is treated by A15 as a DCCISW clean/invalidate by set/way instruction. At the end of a software initialization of the L2 cache, the cache would be correctly invalid, but the software initialization could cause spurious evictions of lines incorrectly marked as valid and dirty.

799271: A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If a certain combination of snoops, DSB instruction execution, and stores occurs with a particular timing, a Cortex-A15 MPCore CPU can deadlock. The deadlocked CPU will be unable to complete an eviction until earlier writes complete, and those writes will not be able to complete until that eviction completes.

Note: This erratum matches bug #5416 in the ARM internal JIRA database.

Conditions

1) One core (coreX) is executing store instructions that are not allocating to the L1 cache. These will be a combination of:
   - Writes to Non-cacheable, Strongly-ordered, or Device memory locations.
   - Write-streaming full line cacheable writes.
2) These stores back up in the memory system and allocate 12 entries of the L2 Write Request Queue
3) CoreX begins the eviction of a cache line (A) that is being replaced by a returning cache line fill
4) Another core (coreY) executes a DSB instruction that generates a TLB synchronization request to coreX
5) A memory request occurs (from coreZ, an ACP request, an external snoop, or a translation table walk request) that
   - hits the cache line (A) being evicted from coreX, OR
   - hits another line in the L1 data cache of coreX that is in uniqueClean or uniqueDirty state.

Implications

If the above scenario occurs with a particular timing, it is possible that the eviction from coreX cannot complete until another L2 write buffer credit is returned, and no such credit can be returned until the eviction completes. In this situation, the processor deadlocks.

In a single core configuration without ACE, the erratum cannot not occur.

In a single core configuration with ACE, but with no other ARM cores capable of issuing a “DVM Sync” request, the erratum cannot occur.

In a two core configuration without ACE, the erratum is believed to be very rare.

In a two core configuration with ACE, or a three or more core configuration, the erratum can occur.

Workaround

There is no known workaround.

2.4. Category A (Rare)

773769: Large data RAM latencies can lead to rare data corruption

Category A Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In a system with ACP in use, with long L2 data RAM latencies (4-8 cycles), or with an L2 data RAM slice configured, it is possible that a rare collision between non-cacheable stores and L1 data cache evictions can lead to data corruption.

Note: This erratum matches bug #5269 in the ARM internal JIRA database.

Configurations Affected

System has one or more devices connected to the A15 ACP port

Conditions

1) The L2 Data RAM latency is programmed to 4 or more cycles, OR ACP is in use, OR A15 is configured with one or more Data RAM slices
2) Multiple stores are sent from the L1 data memory to the L2 cache
3) Memory type of the stores: strongly ordered, device; normal inner-NC, inner-WT, or inner-WBNoAllocate
4) Multiple evictions from the L1 occur intermixed with the non-cacheable stores
5) The evictions and stores are stalled and hazarded in an extremely rare manner

Implications

Incorrect data will be written to the L2 cache or to memory.

A typical 2-CPU A15 implementation will not use a data slice, and will have a data RAM latency of 3 or less. If ACP is not used, that implementation will not be affected.

If an A15 implementation is using ACP, is configured with a data RAM slice, or is using a slower data RAM, in very rare circumstances data corruption could occur. This issue has been reproduced a single time in a stressful environment with a data RAM latency of 8 cycles. The required boundary conditions become less likely to align with lower latency data RAMs. The issues has not been reproduced in other configurations, but can't be ruled out except in the configurations described above.

The default reset value of the data RAM latency is 8 cycles, and would then be programmed by boot code to the faster value (typically 3-5 cycles) before turning on the MMU or enabling the cache.

Workaround

If a Data RAM slice is configured in the A15, there is no workaround.

If the system uses ACP, ACP masters must not issue memory requests to A15.

Data RAM latency in the L2 should be programmed to 2 or 3 cycles. 4, 5, 6, 7, and 8 cycle data RAMs should never be used. Because the reset value of the L2CTLR[2:0] is 3'b000 (corresponding to 8 cycle data RAM latency), after reset and before the MMU is enabled or the SCTLR.c bit is set, the L2CTLR register data ram latency bits (L2CTLR[2:0]) should be programmed to 3'b001 or 3'b010, corresponding to data RAM latencies of 2 or 3 cycles respectively.

775619: L2 may deadlock or corrupt data if all ways in a single set become blocked from replacement

Category A Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

A15 has bits in each cache line in the Level 2 Cache (L2) that can block the line from being replaced. These bits are set to indicate a line is currently held in an L1 cache and must be retained, or may be temporarily set if there is a multi-pass request in flight to that line.

It is possible for all 16 lines in a given L2 set to be 'locked'. If this occurs, requests that need to replace a line in the L2 should detect a hazard and retry, waiting for a line to be released. However, there are two scenarios in which all 16 ways becoming locked can cause an issue.

If all 16 ways in a given set are locked and an ACP write-allocate request comes, the ACP request will not hazard correctly. It will replace a locked line, putting the L2 cache in an inconsistent state. This could lead to data corruption to the locked line that was replaced.

If all 16 ways in a given set are locked a cache line fill that needs to replace a line in that set will continuously hazard and restart. In a rare set of arbitration conditions, it is possible that that request will block the other requests that will eventually release the locked/inclusive lines. This will lead to a deadlock.

Note: This erratum matches bug #5298 and 5302 in the ARM internal JIRA database.

Configurations Affected

- Cat A Rare for 3 or 4 CPU A15 configurations
- Cat B Rare for 1 or 2 CPU A15 configurations

Conditions

1) All ways in a given L2 set have their lock or inclusion bit set
2) An ACP write allocate request is made to that set OR another cache line fill needs to replace a way in that set

Implications

In an A15 with one or two CPUs, there is an easy zero impact workaround that will prevent all the lock/inclusion bits in a set from ever being set at the same time (see workaround below), completely avoiding the data corruption or deadlock.

In an A15 with three or four CPUs, there is no workaround. A system with no ACP write allocate traffic will avoid the data corruption scenario, but could see a deadlock.

Workaround

This erratum can be completely avoided for an A15 containing only one or two CPUs. The "Force in-order requests to same set/way" bit of the Auxiliary Control Register should be set (ACTLR[23] set to 1'b1) in each CPU. This bit will prevent that CPU from having more than two cache line fills outstanding at a given time to the same set. This will completely prevent the L2 from ever seeing all 16 ways in a set with their inclusion or lock bits set and thus avoid the erratum. Because the current L2 implementation will not process more than two requests at a time to a given L1 data cache set, there is no performance impact to setting this bit.

For an A15 containing three or four CPUs, there is no workaround. Preventing ACP write traffic will prevent the silent data corruption, but there is the possibility of a deadlock.

801819:An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing

Category A Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

A livelock can occur in the L2 cache arbitration that might prevent a snoop from completing. Under certain conditions this can cause the system to deadlock.

When a cacheable store is executed in the L2 it requires an entry to be allocated in the Fill Evict Queue (FEQ). If an entry is not available the store will restart, as will any younger stores from the same CPU, because stores from the same CPU must execute in order.

If two cacheable stores from the same CPU issue into the L2 just as an FEQ entry becomes available, it is possible for the first store (ST1) to detect the FEQ as full, but the second store (ST2) to see an available FEQ entry and allocate that entry. Both stores are then restarted, the ST1 due to the lack of an FEQ entry, ST2 because the ST1 restarted. This means the FEQ entry temporarily allocated by ST2 is marked for deallocation.

The two stores will then reissue to the L2. If the timing of the reissue aligns with the deallocation of the FEQ entry temporarily allocated previously by ST2, it is possible that ST1 will again detect an FEQ full condition and ST2 will see an FEQ entry available and allocate it. If there is no other activity in the L2 to change the timings, this cycle can repeat until a second FEQ entry becomes available and both stores can proceed, or another FEQ entry is allocated for an unrelated request and both stores stop reissuing waiting for an FEQ entry to become available.

Depending on the L2 cache configuration options there might need to be one or more unrelated stores between ST1 and ST2 in order to hit the required timing for the livelock.

If there is an eviction in the write buffer that is behind the stores then that eviction might be stalled as long as the livelock continues. This stalled eviction might stall a snoop hit on the L1 data cache of that CPU. The stalled snoop might block further FEQ entries from deallocating, either through a dependency in the ACE memory system, or because the FEQ is full of snoop hits to the L1 data cache of that CPU. If all of these conditions occur the system can deadlock.

Note: This erratum matches bug #5433 in the ARM internal JIRA database.

Conditions

1) The L2 Fill Evict Queue (FEQ) is full.
2) Cacheable write ST1 issues followed by cacheable write ST2 from the same CPU.
3) An eviction is waiting in the write buffer behind ST1, ST2, and one or more additional stores.
4) A snoop (from another A15 core or from the ACE AC channel) is waiting for that eviction to complete.
5) That snoop is preventing further FEQ entries from deallocating.
6) Very specific timing conditions.

Implications

If the erratum conditions are met the part will deadlock.

Cortex-A15 will only issue two types of cacheable stores from a CPU into the L2: Write-Back No-Allocate stores, or Write-Back stores that have been gathered into a full line write using the L1 write streaming logic.

For implementations of Cortex-A15 configured with the “L2 arbitration register slice” (arb-slice) option (typically four core systems) the only cacheable stores that can cause the issue are Write-Back No-Allocate stores. Because Write-Back No-Allocate stores are not commonly used and are easy to avoid, there is a straightforward workaround for these implementations. For configurations with the arb-slice, this erratum is Category B.

For implementations of Cortex-A15 configured without the arb-slice option (typically 1 or 2 core systems) Write-Back No-Allocate stores and streaming cache line writes can both lead to the deadlock, although hitting the conditions with streaming cache line writes is much more difficult. There is still a full workaround, but because it involves disabling write-streaming there is a performance hit on some streaming memory workloads. For configurations without the arb-slice, this erratum is Category A Rare.

Workaround

Do both of the following:

1) Do not use the write-back no-allocate memory type.

2) Do not issue write-back cacheable stores at any time when the cache is disabled (SCTLR.C=0) and the MMU is enabled (SCTLR.M=1). Because it is implementation defined whether cacheable stores update the cache when the cache is disabled it is not expected that any portable code will execute cacheable stores when the cache is disabled.

For implementations of Cortex-A15 configured without the "L2 arbitration register slice" option (typically one or two core systems), you must also do the following:

3) Disable write-streaming in each CPU by setting ACTLR[28:25] = 0b1111

2.5. Category B

766170: HCR.FB doesn't upgrade ICIALLU to ICIALLUIS

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

When the HCR.FB bit is set, instruction cache invalidate all commands (ICIALLU) must be upgraded to instruction cache invalidate all Inner Shared (ICIALLUIS). Instead, A15 executes the ICIALLU command. This means that the instruction cache local to that processor will be invalidated, but none of the instruction caches in the other processors in the A15 cluster will be invalidated.

Note: This erratum matches bugs #5113 in the ARM internal JIRA database.

Conditions

1) HCR.FB = 1

2) Execute ICIALLU

Implications

This feature would be used by a hypervisor to allow a Guest-OS that is not MP aware to be able to migrate between multiple processors in an A15 cluster. If the non-MP aware Guest-OS modifies instruction memory (loading instruction pages or doing self-modifying code in a JIT) and follows it with an ICIALLU to clean the instruction cache, the instruction caches of the other processors would not be invalidated. Were that Guest-OS to migrate to another processor it could find stale copies of instruction memory in the instruction cache. By upgrading the ICIALLU to ICIALLUIS, the hypervisor should have avoided this problem, but the erratum means that won't work

Workaround

When the hypervisor is migrating a non-MP aware OS from processor A to processor B within the A15 cluster, the hypervisor must execute an ICIALLU on processor B, or an ICIALLUIS on any processor. This will prevent the non-MP aware OS from seeing stale data in the icache.

766171: TLBINV followed by changing VTTBR VMID value could cause incorrect TLB invalidation

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

When a TLB invalidate command is executed in non-secure mode, the VMID is often part of the matching criteria used. This VMID should be the current VMID of the executing processor at the time the TLB invalidate command is executed.

However, if a processor executes a TLB Invalidate command followed quickly by a change to the VMID and an ISB to force completion of that VMID change, the TLB Invalidate command may incorrectly use the updated VMID value to qualify its invalidations, rather than the original VMID. TLB entries that should have been invalidated by the command may be left valid in the TLBs.

Note: This erratum matches bugs #5119 in the ARM internal JIRA database.

Conditions

1) A TLB Invalidate command is executed in non-secure state (Kernel or Hypervisor)

2) The Virtualization Translation Table Base Register (VTTBR) is written changing the VMID (VTTBR[55:48])

3) An ISB instruction is executed before the TLB invalidate has completed execution

Implications

If this erratum is hit, the TLB will not be correctly invalidated. Incorrect translations could occur.

In general, it would be difficult (but not impossible) for this erratum to affect TLB invalidate operations executed in non-secure Kernel mode. This is because NS-Kernel can’t change the VMID and it is very unlikely that TLB operations would not have had time to sample the correct VMID value before a mode change to hypervisor and then a VMID change was completed.

The code most vulnerable to this erratum would be the hypervisor TLB maintenance code. It is possible that the hypervisor would choose to execute one or more TLB entries associated with one VMID, quickly change the VMID, and execute one or more TLB entries associated with another VMID.

Workaround

After any TLB invalidate command and before the next VMID change, execute either a DMB or a DSB instruction. Either of these barriers will force all prior TLB invalidate operations to complete to the point where they will no longer be affected by VMID changes.

The simplest approach would be to find all code that adjusts the VTTBR and place a DMB before the write to the VTTBR.

766173: Prefetch hit by WB-No Allocate load can incorrectly set inclusion bit

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

The hardware prefetcher initiates a read of a write-back read-write-allocate cache line. One of the processors then issues a write-back no-allocate read request to the same line. This hits the outstanding prefetch and merges with it. This request must now recognize that, upon returning data to the requesting processor, the line will not be allocated into the L1 data cache. In the erratum case, however, the L2 inclusion bit (indicating the line is being held in an L1 data cache) is set incorrectly. This prevents it being replaced until the L1 data cache evicts that line. Because the L1 data cache does not have that line, that bit will never be cleared during normal operation and the line can never be replaced.

This will not lead to an immediate problem (other than a negligible loss of performance due to that cache set being 15 way instead of 16 way). However, over a very long period of time it would be possible for this rare boundary condition to occur multiple times for the same set. If it happened enough times for the same set (minimum eight), the A15 could deadlock.

Write-back no-allocate requests can be generated in two ways:

1) page tables that specify write-back no-allocate memory

2) accesses to any write-back cacheable memory when the data-cache is disabled

Note: This erratum matches bug #5124 in the ARM internal JIRA database.

Conditions

1) A stream of read requests to write-back cacheable memory is detected by the hardware prefetch logic

2) The final request of this stream is to write-back read-write-allocate memory, or a load exclusive to write-back no-allocate memory (which will generate a write-back read-write

3) allocate request to the L2)

4) Hardware prefetch issues a cache line read with write-back read-write-allocate attributes

5) A CPU makes a request to the same line with write-back no-allocate memory attributes

Implications

If the above conditions happen enough times (minimum >8 times for a given cache set) without any of the affected lines being loaded into the data cache or clean/invalidated through software maintenance, the part could deadlock. In our hundreds of billions of simulation cycles the event occurring even a single time is very rare, but in silicon running over a long period of time without any L2 cache clean/invalidation it would be possible to generate a deadlock scenario.

Workaround

The simplest full workaround for this erratum is to disable hardware prefetch in the L2. Write 0x0000_0400 to the L2 Prefetch Control Register to accomplish this.

An alternative workaround is to prevent the LS from issuing memory requests to write-back no-allocate memory. Write-back no-allocate requests will only occur if:

1) the long page table format is in use and write-back-NoAllocate is specified as a memory type.

2) the cache is disabled, the MMU is enabled, and a data access is made to write-back memory.

If these requests can be avoided, the erratum will not occur. The operating system should not make use of WB-NoAllocate memory. When disabling the cache (during a powerdown sequence or some other cache maintenance activity) either don’t access write-back cacheable memory, or disable the MMU.

766421: Strongly-Ordered/Device load or NC LDREX could return incorrect data

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In certain situations, a strongly ordered or device load instruction, or a non-cacheable normal memory load-exclusive instruction could match multiple fill buffers and return incorrect data. This occurs because the non-speculative request no longer needs to be translated by the TLB (it has already made its request to memory). However, if its original TLB entry is lost or if it matches a different TLB entry, it will read out a random physical address (in the case of a false TLB hit) or a physical address of zero (in the case of a TLB miss). This new physical address information can cause a false hit if there happens to be a request to that physical address in flight at the same time. This will not be common.

Note: This erratum matches bugs #5139 in the ARM internal JIRA database.

Conditions

1) MMU enabled
2) Non-speculative load (SO/Dev load or NC LDREX) executed, committed, causes memory read (LD1)
3) The TLB translation for LD1 is lost (invalidated or replaced)
4) Load or store to normal memory causes memory read to physical address 0x0-0x3f (OP2)
5) Data for both requests returns
6) LD1 issued down the pipe, misses TLB, reads 0x0 for physical address
7) LD1 Matches fill buffer entries for LD1 and OP2, merges the data

--OR--

1) Non-speculative load (SO/Dev load or NC LDREX) executed, committed, causes memory read (LD1)
2) LD1 virtual address =X, LD1 physical address=Y
3) The processor translation regime changes: any change to security state, ASID, VMID, MMU-enable
4) Load or store to normal memory causes memory read to physical address=Y (OP2)
5) Data for both requests returns
6) LD1 issued down the pipe, falsely hits TLB, reads Y for physical address
7) LD1 matches fill buffer entries for LD1 and OP2, merges the data

Implications

If the above combination of events occurs and satisfies the timing conditions required to trigger the erratum, the data returned to the register file for LD1 will be incorrect.

Workaround

Do all of the following:

1) Do not map any Normal Memory page table entry to the lowest 4k of physical memory.
2) Add a DMB instruction to flush all previous memory operations when making any change to the translation regime and before doing any new loads/stores/preloads in the new translation regime. Changes to the translation regime are:
   - ASID change
   - VMID change
   - MMU enable/disable
   - Security state change
   - Entering/leaving Hypervisor mode

For the security state changes, place a DMB in any secure exception handlers before any memory operations, and put a DMB after any memory operations before branching to non-secure state.

For the hypervisor entry/exit, place a DMB in any hypervisor trap or exception handlers before any memory operations, and put a DMB after any memory operations before branching back to the Guest-OS (non-secure non-hyp mode).

766422: Thumb store translation fault to Hypervisor may not have correct HSR Rt value

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

When a non-secure non-hypervisor memory operation instruction generates a stage2 page table translation fault, a trap to the hypervisor will be triggered. For an architecturally defined subset of instructions, the Hypervisor Syndrome Register (HSR) will have the Instruction Syndrome Valid (ISV) bit set to 1'b1, and the Rt field should reflect the source register (for stores) or destination register for loads.

On Cortex-A15, for Thumb and ThumbEE stores, the Rt value may be incorrect and should not be used, even if the ISV bit is set. All loads, and all ARM instruction set loads and stores, will have the correct Rt value if the ISV bit is set.

Note: This erratum matches bugs #5151 in the ARM internal JIRA database.

Conditions

1) Executing in Thumb or ThumbEE mode in non-secure-non-hyp

2) Execute a store instruction that triggers a stage2 translation fault

Implications

The hypervisor cannot trust the Rt value for Thumb/ThumbEE stores and must load and decode the instruction manually in order to emulate the instruction. This will lead to slower hypervisor emulation support in the hypervisor when the GuestOS is executing in Thumb/ThumbEE mode.

Workaround

In the hypervisor handler, in addition to the ISV bit, the hypervisor should check the HSR.WnR bit and the mode bits in the SPSR_hyp. If the process that generated the exception was executing in Thumb/ThumbEE mode and the HSR.WnR bit is 1'b1, the hypervisor handler should consider the Rt value to be untrustworthy and decode the instruction in software.

769270: L2 Tag Latency = 3 should not be used in a system configured with Tag Slice

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In an A15 configured with the L2 Tag Slice and programmed to an L2 Tag RAM Latency of 3 cycles, in certain situations write requests may be sent to the L2 Tag RAM as if it had a latency of 2 cycles. This could lead to corrupted Tag RAM contents and unpredictable results.

Note: This erratum matches bug #5215 in the ARM internal JIRA database.

Conditions

1) A15 configured with the L2 Tag Slice

2) A15 L2 Tag RAM latency programmed to 3 cycles

Implications

On versions of A15 affected by this erratum, the above configuration should not be used to avoid data corruption and possible processor deadlock.

Workaround

The workaround for existing silicon with the L2 Tag Slice that is affected by this erratum is to program the Tag RAM latency to 2 cycles. This could affect the maximum frequency at which the part will operate.

773022: Incorrect instructions may be executed from loop buffer

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In certain rare sequences of code, the loop buffer may deliver incorrect instructions.

NOTE: This erratum matches bug #5121 in the ARM internal JIRA database.

Conditions

1) A15 loop buffer enabled

2) Rare sequence of branches executed

Implications

If the loop buffer is enabled, incorrect code may be executed.

Workaround

On affected product versions, the loop buffer should be disabled. This can be achieved by setting bit 1 of the ACTLR register to 1'b1.

774569: Watchpoint may not be taken on second half of unaligned ld/st crossing a 64-byte aligned boundary

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

An unaligned memory transaction that crosses a 64-byte aligned memory boundary is split into two requests in the A15 pipeline. Each of these requests is compared to the active watchpoints to see if a watchpoint event needs to occur.

If the address mask field in the DBGWCR is set to ‘No mask’ or ‘Three bits masked’ then the second half of such an unaligned transaction will never trigger a watchpoint.

Note: This erratum matches bug #5275 in the ARM internal JIRA database.

Conditions

1) DBGWCR[28:24] (Mask) = 5’b00000 or 5’b00011
2) Unaligned memory request crossing a 64-byte boundary
3) DBGWVR has a watchpoint address in the upper 64-byte memory region of the request

Implications

No watchpoint will be taken.

In most cases, this will not be an issue. If a specific variable in memory is being watched, the lowest byte address of the variable should be set in the WVR and the bug will not occur. Memory requests that are moving a region of memory without regards to the underlying variable locations will tend to be aligned. The issue will only arise when the WVR location specified is not the target address of the memory transaction being issued.

Workaround

Set the DBGWCR[28:24] to 5’b00100 (4-bits masked) or higher. The watchpoint will now be taken correctly. However, Software may have to detect and ignore false triggers of the watchpoint because a watchpoint may also fire for nearby bytes in the same 64-byte aligned memory region as the desired bytes.

774769: Data corruption may occur with store streaming in a system

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

An L2 write request (A) uses a single data bank for two passes to do an ECC read/modify/write. This allows the last quad word of a cache line transaction (B) to get significantly delayed from the first quad word going through the L2 pipe. During this window the L2 buffer associated with that transaction is deallocated and reallocated to a full cache line streaming store (C) from a CPU that will allocate to the L2 cache without an external memory access. The last quad word of the original transaction then collides with the streaming store and incorrect data is written to the cache for that streaming store.

In order to get the streaming store (C) the CPU must have detected a store stream. A series of cacheable stores in a CPU triggers the store streaming mode in that CPU. The CPU then buffers up 64-byte lines of cacheable store data before sending them to the L2 cache. One of those full line writes misses the L2, but will immediately allocate to the L2 without an AXI/ACE request. This is only possible in a non-ACE system, or with streaming stores to non-shared data. Any shared data in an ACE coherent system would first need to obtain ownership of the line with an AR channel memory request.

Note: This erratum matches bug #5293 in the ARM internal JIRA database.

Conditions

1) store streaming enabled and allocating data to the L2 cache
2) a series of incrementing stores is executed to an aligned 64-byte region of cacheable memory
3) either:
   - the memory accessed by the stores is non-shared, OR
   - the system is not coherent over ACE (BroadcastInner and BroadcastOuter pins both deasserted)

Implications

If this condition is hit, external memory may be corrupted.

Workaround

The workaround is to configure write streaming on versions of A15 affected by this erratum such that no streaming-write ever allocates into the L2 cache. This can be done by setting the no-allocate threshold to be lower than the L2-allocate threshold. Once streaming starts, all store streams will go immediately to external memory. ACTLR[28:27] (Write streaming "no-allocate" threshold) should be set to 2’b00 (12 cache lines) and ACTLR[26:25] (Write streaming "no-L1-allocate" threshold) should be set to 2’b01 (64 cache lines) or 2’b10 (128 cache lines).

777770: ESR incorrect for AdvSID HCPTR trapped inst in Hyp mode

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

The EC field of the HSR should be written to 0x7 for an HCPTR trapped coprocessor instruction while executing in HYP mode. A15 treats this as an UNDEF exception and writes 0x0 to the HSR.EC field.

Note: This erratum matches bug #5317 in the ARM internal JIRA database.

Conditions

1) HCPTR[10] and/or HCPTR[11] set to 1'b1

2) A VFP/Advanced SIMD instruction is executed

Implications

A hypervisor that is trapping VFP or Advanced SIMD instructions executing in the hypervisor will see 0x0 in the EC field of the HSR. This EC value will indicate an Undefined exception. The hypervisor handler will have to load the instruction that triggered the exception and decode it to determine that the instruction was VFP or Advanced SIMD. This will take somewhat longer than determining the issue directly from the HSR register.

Workaround

If the hypervisor is running with HCPTR[10] or HCPTR[11] set and expecting to trap VFP/Advanced SIMD instructions, ensure that your Undefined exception handler loads the instruction to determine that an Advanced SIMD or VFP instruction was being executed and handle it as needed.

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

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

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.

784477: CTIINTACK register needs clearing each time it is set

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

The CTI contains a CTIINTACK register, which enables a trigger to be acknowledged through software, instead of using a hardware knowledge using the CTITRIGOUTACK input. The correct operation of this register is that writing a one to the bit corresponding to a trigger output will cause that trigger to be cleared, and this will not affect future triggers.

Because of this erratum, when a bit in the CTIINTACK register is set, it remains set until cleared by writing zero to the register. This causes the corresponding trigger outputs to be acknowledged immediately if they occur again, which can lead to them being missed.

The CTIINTACK register is normally used in two cases:

- To clear a debug-originated interrupt, if required by the interrupt controller.
- To clear a debug entry request generated by another processor, when cross-halting is used.

Conditions

The following conditions must occur:

- A CTI trigger output fires.
- The CTI CTIINTACK register is used to acknowledge the trigger output, by writing a one to the bit corresponding to that trigger output.
- The same trigger output fires again before the corresponding bit in the CTIINTACK register is cleared.

Implications

Trigger outputs might be missed:

- In the case of a debug-originated interrupt that uses CTIINTACK to clear the interrupt, events other than the first event might not cause an interrupt to occur.
- In the case of a cross-halting debug request, after the first time a processor halts and restarts, it might halt without halting other processors with it.

Workaround

This is a workaround for tools vendors.

When the CTIINTACK register is written with a nonzero value, it must be immediately written to again with the value zero. This prevents any future events on the corresponding trigger output from being acknowledged.

If this workaround is used, there remains a race condition, whereby a trigger output occurring between the two register writes might be lost. This is in general not significant, because the timing of trigger outputs and the timing of register writes are not highly correlated, and if the trigger output had occurred before the first register write, then it would also have been lost.

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: r0p4

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 in 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.

798181: Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

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 this might not occur in the following cases:

- The exclusive monitor is physical address based. Moving a memory region in physical memory might cause unexpected results, including multiple threads acquiring the same lock.
- Hazarding logic to guarantee read after read ordering to the same address is physical address based. If a memory region is moved in physical memory ordering violations might occur.
- The DSB instruction might complete before all memory transactions to the invalidated translation are globally observed.

The first two conditions might lead to unexpected ordering and Load-Exclusive/Store-Exclusive instructions behaving in an unexpected way. The third condition 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 memory transactions have completed.

Note: This erratum matches bugs #5405, #5407, #5428, #5429 in the ARM internal JIRA database.

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:

1) exclusive monitors in the system might be using the old translation
   - this might lead to unexpected Load-Exclusive/Store-Exclusive behavior.
2) the read after read hazard logic might be using the old translation
   - this might lead to a load instruction returning older data than a previous load instruction that accessed the same virtual memory location.
3) outstanding loads or 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

To ensure the completion of a TLB invalidate, software must first follow the translation table invalidation steps specified in the ARM Architecture Reference Manual:

1) Modify the translation tables in memory as required.
2) Execute a DSB to ensure observation of the stores to the translation tables.
3) Execute one or more TLB invalidate instructions targeting the affected memory regions.

4) Execute a DSB to ensure the TLB invalidate instructions from step 3 have been propagated to all processors in the system.

Next, on the processor that is performing the translation table maintenance, it must also perform these two additional steps:

5) Execute a dummy TLBIMVAIS, meaning a TLBIMVAIS to any address with any ASID.
6) Execute a DSB instruction to ensure the dummy TLB invalidate instruction has been propagated to all processors in the system.

Finally, identify any Cortex-A15 processors in the system that are currently using the translation context of the region being invalidated. For global regions this will be all Cortex-A15 processors not powered down or in reset. For non-global regions this will be all Cortex-A15 processors that are currently using the same translation context (typically ASID and VMID) as the modified translation table entries. On each of those processors, ensure that the following occur (in any order):

- execution of a CLREX instruction
- execution of a DMB or DSB instruction

Once the above sequence is complete all uses of the old translation in the system will removed, and any memory transactions to the old translation will be globally observed.

To meet the final requirement, the processor performing the translation table maintenance must execute a CLREX and a DMB or DSB instruction.

The easiest way to meet the final requirement on other processors in the system is to send an inter-processor interrupt to each required processor. Most interrupt handlers will execute a CLREX instruction (as required by the ARM Architecture on a context switch). The interrupt handler must execute a DMB instruction and then signal completion to the processor doing the translation table maintenance.

Note: this workaround must be implemented on any ARM Architecture processor that is executing TLB invalidate instructions that may affect a Cortex-A15 processor. This would include translation table maintenance being performed on a Cortex-A7 processor when a Cortex-A15 processor is present in the system.

Note2:

In r3p2 and earlier versions with REVIDR[4]= 0, the full workaround is required.

In r3p2 and earlier versions with REVIDR[4]=1, REVIDR[9]=0, only the portion of the workaround up to the end of step 6 is required.

In r3p2 and earlier versions with REVIDR[4]=1, REVIDR[9]=1, no workaround is required.

In r3p3, if REVIDR[9]=0, only the portion of the workaround up to the end of step 6 is required.

In r3p3, if REVIDR[9]=1, no workaround is required..

799270: Writing ACTLR.SMP when the L2 cache has been idle for an extended period may not work correctly

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If the L2 cache logic clock is stopped because of L2 inactivity, setting or clearing the ACTLR.SMP bit might not be effective. The bit is modified in the ACTLR, meaning a read of the register returns the updated value. However the logic that uses that bit retains the previous value.

Note: This erratum matches bug #5417 in the ARM internal JIRA database.

Conditions

1) The L2 cache block has been idle for 256 or more cycles with no memory requests from any core, no external snoops, and no ACP requests.

2) A CPU executes an “MCR p15,0,r0,c1,c0,1” instruction (write the ACTLR register) that modifies ACTLR[6].

Implications

The ACTLR.SMP bit controls whether that CPU should receive distributed virtual memory (DVM) requests (instruction cache maintenance, BTB maintenance, or TLB maintenance commands) from other ARM processors in the system. The processor documentation requires that:

- The only time this bit can be set to 1 is at boot time before enabling the MMU.
- The only time this bit can be cleared to zero is during a specific power-down sequence described in the TRM.

If the errata conditions occur when the ACTLR.SMP bit is being set at boot, the instruction cache or TLB could become incoherent, as that CPU would not receive necessary DVM requests.

If the errata conditions occur when the ACTLR.SMP bit is being cleared during a CPU power-down sequence, the L2 logic may send a DVM request to that CPU when it is powered down or in reset. That CPU would not respond to the request and the system would deadlock.

Note: This erratum does not apply to the r3 versions of Cortex-A15 MPCore processor due to clock gating changes implemented for power savings in those revisions.

Workaround

To avoid this erratum, software must ensure that the L2 logic has been used within the previous 256 cycles before modifying the ACTLR.SMP bit. As specified in the TRM, the only times when this bit can be modified are during boot before the MMU has been enabled, or during a specified reset or power down sequence. When modifying this bit, software must ensure that interrupts are disabled (which may require executing in Secure mode), and then do a memory read to a memory location with Non-cacheable, Strongly-ordered, or Device memory attributes. If the MMU is disabled, all memory will appear Strongly-ordered. A dependency must be created between the returning load data and the MCR instruction that sets the ACTLR.SMP bit, as shown in the code below.

Code sequence for setting the ACTLR.SMP bit:

```
; the following code must be executed with all interrupts disabled
; r1 must contain the value of an Non-cacheable, SO, or Dev memory location or register
;    (typically a memory mapped register with no read side effects would be used)
mrc p15,0,r0,c1,c0,1    ; read current value of ACTLR
orr r0,r0,#0x40         ; set SMP bit (ACTLR[6])
ldr r1, [r1]            ; read a device register (location guaranteed not to hit
                         ; the L1 cache)
and r1,r1,#0            ;
orr r0,r0,r1            ; create dummy dependency between dummy load and MCR to write SMP
```

```
MCR p15,0,r0,c1,c0,1    ; Write CP15 ACTLR
ISB
DSB
```

Code sequence for clearing the ACTLR.SMP bit:

```
; the following code must be executed with all interrupts disabled
; r1 must contain the value of an Non-cacheable, SO, or Dev memory location or register
;    (typically a memory mapped register with no read side effects would be used)
mrc p15,0,r0,c1,c0,1    ; read current value of ACTLR
bic r0,r0,#0x40         ; set SMP bit (ACTLR[6])
ldr r1, [r1]            ; read a device register (location guaranteed not to hit
                        ; the L1 cache)
and r1,r1,#0            ;
orr r0,r0,r1            ; create dummy dependency between dummy load and MCR to write SMP
MCR p15,0,r0,c1,c0,1    ; Write CP15 ACTLR
ISB
DSB
```

804622: Extension register load to SO/Dev address combined with erroneous BLX can stall core

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

An extension register load instruction (ExLd1) is executed at instruction address Z. The memory location being loaded by ExLd1 is in Strongly Ordered or Device (SO/Dev) memory. This extension register load to SO or Dev memory triggers a special mechanism that includes a pipe flush and re-execution of ExLd1 in a special ordering mode.

Before the ExLd1 pipe flush occurs, two branch instructions (B1 and B2) are issued. B1 either branches back to Z but incorrectly switches to Thumb state (if currently in ARM state), or branches to a misaligned address near Z (if currently in Thumb state). This misalignment or incorrect instruction set state causes the two bytes at Z to be decoded as a branch (B2) instead of as a load. This bad branch B2 is speculatively executed and loads the branch target buffer (BTB) with information indicating instruction address Z is a branch.

The ExLd1 pipe flush then occurs, and address Z is refetched. The BTB now incorrectly predicts address Z as a branch. When the incorrect prediction is detected the instruction is discarded and the pipe flushed. However, this bad-branch pipe flush exits the special ordering mode needed by ExLd1. When Z is refetched, the attempted re-execution of ExLd1 again triggers the SO/Dev pipe flush and the sequence repeats.

An interrupt or other exception permits normal execution to resume in the exception handler and other threads. However, upon return to the original thread the same repeated flushes might occur.

Note: This erratum matches bug #5449 in the ARM internal JIRA database.

Conditions

1) An extension register load instruction (ExLd1) at address Z to Device or Strongly Ordered memory

2) An erroneous branch, either:

- In ARM instruction set state a branch targeting address Z, but incorrectly switching to Thumb instruction set state, OR
- In Thumb instruction set state, a branch targeting a 32 bit instruction at (Z-2) overlapping the ExLd1.

Implications

If the conditions are met, the core will stop executing instructions in the current thread. An interrupt or other external exception will break the stall, but upon returning from the interrupt the same sequence might deadlock again. The effect would be that the affected thread is never able to make forward progress.

This erratum is not expected to be common. Extension register loads to Strongly Ordered or Device memory are rare in normal applications, and the 'bad branch' should not occur in executable code as the results would be unpredictable. However, if an extension register load was followed by an unpredictable branch over a block of data it is possible that early bytes in the block of data could be decoded as the bad branch back to the extension register load.

Workaround

If a thread is unable to make forward progress and the next instruction to be executed is an extension register load to Strongly Ordered or Device memory that meets the erratum requirements, ensure that there is a predictable branch between the load and the potential bad branch. If there is a data section soon after an extension register load to SO/Dev memory where the data might be decoded as a branch, with no intervening predictable branch, placing a dummy "branch to self" at the head of the data section would avoid the erratum.

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: r0p4

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.

830321: Cortex-A15 might falsely trigger a watchpoint exception on a CLREX instruction

Category B
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r0p4

Description

Cortex-A15 might falsely trigger a watchpoint exception on a CLREX instruction.

Note: This erratum matches bug #5402 in the ARM internal JIRA database.

Configurations affected

All configurations are affected.

Conditions

1) The watchpoint is set to match unprivileged instruction.
2) CLREX is executed. The irrelevant virtual address info from a previous execution happens to match the virtual address as being programmed in Watchpoint Value Register.

Note that this erratum will not occur if the watchpoint is not configured to trigger in the security state in which CLREX being executed. Also, this erratum will not occur if a CLREX is executed in Hyp mode (PL2) and the watchpoint is not configured to trigger in Hyp mode.

Implications

Cortex-A15 might falsely trigger a watchpoint exception on a CLREX instruction.

Workaround

- The watchpoint handler should look at the instruction that triggered the watchpoint. If it is CLREX, it should simply return from the handler.
- If user mode watchpoints are enabled, the kernel should not execute a CLREX in any situation where an abort cannot be tolerated.
- If a CLREX is required in a code sequence that cannot tolerate a precise abort, the CLREX should be replaced with a dummy STREX or user mode watchpoints should be disabled.

836969: Code sequence continuously hitting the L1 cache can block snoop

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4.

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: r0p4

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.

773319: Unaligned page boundary crossing load hit by TLB invalidate can stall until next interrupt

Category B Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

An unaligned cacheable writeback load instruction is being executed. It crosses a page table boundary. Both halves of the load hit in the TLB, and the data required from the lower of the pages misses in the cache. A cache line fill is issued to the L2 cache or to memory. The data required from the upper page misses the cache, but does not issue a cache line fill due to congestion.

While that fill is outstanding, a TLB invalidate is received that knocks out the TLB entry associated with the upper page, but does not affect the lower page. After that TLB invalidate is processed, the cache line fill returns. A new page table request must now be issued to the L2TLB or memory for the upper page.

A snoop request is then received by the L1 data cache targeting the just returned cache line fill. This snoop will not be processed until a the TLB request for the upper page is complete.

If the upper page miss requires a memory read and that memory read is unable to complete until the snoop completes, the processor may stall.

If another TLB maintenance operation is received by the A15 CPU just after the above snoop and before the page miss can be issued, the upper page miss may be prevented from issuing until the snoop completes and the processor may stall.

In either case, the A15 CPU will be unable to respond to the snoop or complete the load until the next branch flush, event flush, or interrupt.

Conditions

1) Unaligned load request that crosses a page table boundary
2) The lower page accessed by the load is to writeback shared memory
3) Both pages hit in the L1 DTLB
4) The required data from the lower page misses in the cache and issues a line fill
5) The required data from the upper page misses in the cache but does not issue a fill
6) A TLB invalidate command from a different CPU is received by the L1 DTLB
7) This TLB invalidate affects the upper page, but not the lower page (The TLB entry for the lower page must stay valid for the erratum to occur)
8) Data returns for the cache line fill
9) A snoop request is received for the cache line that just returned
10) Either:
    1) The page table walk for the just invalidated upper page requires a memory access that stalls in the memory system dependent on the above snoop, or
    2) A TLB maintenance operation from another CPU is received just after the above snoop command

Implications

If the above conditions occur, the CPU will be unable to complete either the unaligned load or the snoop request until the next exception (branch flush, event flush, or interrupt) occurs. No data corruption will occur. The next branch flush, fault, or interrupt will resolve the issue and execution will continue normally.

This should be very rare in real systems and should only have a temporary performance impact when it occurs. A targeted TLB invalidate that affects only one of the two pages being accessed by an active process, combined with the extremely rare timing sequence above, will make this extremely unlikely to be seen in a running system.

Workaround

There is no workaround.

804969: Extension register loads and stores can livelock core

Category B Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In this erratum description, extension register means a register in the Advanced SIMD and floating-point register bank.

An Advanced SIMD or floating-point data processing instruction (OP1) can stall due to a constant stream of later Advanced SIMD or floating-point data processing instructions being speculatively issued and cancelled. These later operations are issued and cancelled because speculative extension register loads that provide operands for the later instructions are issued and cancelled. The loads are cancelled because they hazard against an extension register store to Strongly Ordered or Device memory. The extension register store cannot complete because the store data is being supplied by the stalled operation (OP1).

Under very specific timing conditions, it is possible for OP1 to be stalled indefinitely.

Note: This erratum matches bug #5451 in the ARM internal JIRA database.

Conditions

1) Advanced SIMD or VFP data processing operation (OP1).
2) An extension register store (ST1) to Strongly Ordered or Device memory, dependent on OP1 results.
3) Three or more extension register loads after ST1 (LD1, LD2, LD3).
4) Two data processing operations dependent on each of LD1, LD2, LD3 (six operations in total).
5) Very specific timing.

Implications

In general, if all of the conditions are hit OP1 cannot to make forward progress. An interrupt will not break the stall. Any inner shared TLB invalidate instruction (e.g. TLBIMVAIS) to any address issued by another ARM core in the system will break the stall.

Extension register stores to Strongly Ordered or Device memory are not expected to be common in user applications.

This condition is currently only seen in contrived (non-useful) validation code. The erratum has not been seen in real applications.

Workaround

Avoid extension register stores to Strongly Ordered or Device memory.

2.7. Category C

766174: ID_PFR0 Jazelle extension support incorrectly encoded

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In the ID_PFR0 register, bits[11:8] identify the type of Jazelle extension support provided. Cortex-A15 currently has a value of 0x0 there (Jazelle not supported). This is a static register read by software to identify processor features that are available. The value should be 0x1 (Support for Jazelle extension, without clearing of JOSCR.CV on exception entry). Cortex-A15 does not support direct byte-code execution, but it does support the trivial implementation of the Jazelle extension.

Note: This erratum matches bugs #5126 in the ARM internal JIRA database.

Conditions

Reading ID_PFR0

Implications

Software relying on that register will believe that Cortex-A15 does not support Jazelle extensions.

Workaround

Any software relying on that value to determine whether Jazelle support is available should check the MIDR to determine if it is running on a Cortex-A15. If it is running on a Cortex-A15, it can assume that the trivial Jazelle support is available.

766419: Benign op instead of UNDEFINED in certain unused opcodes

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

The ARM architecture specifies that some unallocated instruction encodings around the CLREX, DSB, DMB, and ISB instructions are illegal encodings that should take an UNDEFINED instruction exception. Cortex-A15 treats these as UNPREDICTABLE and, to avoid logic/speed-paths, treats those encodings the same as the nearby CLREX/DSB/DMB/ISB instructions.

The effect will be that, if you execute one of these unallocated encodings, you will get a CLREX or barrier instead of an UNDEFINED instruction exception.

```
--CLREX--
32'b1111 0101 1111 xxxx  xxxx xxxx x0xx xxxx  will be treated as CLREX
32'b1111 0101 x111 xxxx  xxxx xxxx 10xx xxxx  will be treated as CLREX
32'b1111 0101 x111 xxxx  xxxx xxxx x01x xxxx  will be treated as CLREX
32'b1111 0101 x111 xxxx  xxxx xxxx x0x0 xxxx  will be treated as CLREX
--DSB--
32'b1111 0101 1111 xxxx  xxxx xxxx x100 xxxx will be treated as DSB
32'b1111 0101 x111 xxxx  xxxx xxxx 1100 xxxx will be treated as DSB
--DMB--
32'b1111 0101 1111 xxxx  xxxx xxxx x101 xxxx will be treated as DMB
32'b1111 0101 x111 xxxx  xxxx xxxx 1101 xxxx will be treated as DMB
--ISB--
32'b1111 0101 1111 xxxx  xxxx xxxx x11x xxxx will be treated as ISB
32'b1111 0101 x111 xxxx  xxxx xxxx 111x xxxx will be treated as ISB
32'b1111 0101 x111 xxxx  xxxx xxxx x111 xxxx will be treated as ISB
```

Note: This erratum matches bugs #5134 in the ARM internal JIRA database.

Conditions

1) executing in ARM mode
2) execute one of the above opcodes that should trigger an UNDEFINED instruction exception

Implications

Historically in the ARM architecture these encodings were UNPREDICTABLE and have only recently been made UNDEFINED. If malicious code happens to execute one of these encodings, the effects are benign. The barriers have no architectural state impact. The CLREX will clear the local monitor, but had the encoding triggered an UNDEFINED instruction exception, the exception handler would have been required to clear the local monitor anyway.

The reason these are UNDEFINED is that the architecture may subsequently re-purpose them, thus there is an assumption that an end user should not be relying on the UNDEFINED behavior (this is what the permanently UNDEFINED space is for).

Workaround

None.

768529: Debug version of ID_PFR0 Jazelle extension support incorrectly encoded

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In the ID_PFR0 register, bits[11:8] identify the type of Jazelle extension support provided. The correct value for Cortex-A15 should be 0x1 (Support for Jazelle extension, without clearing of JOSCR.CV on exception entry). However all r0 version (r0p0,r0p1,r0p2,r0p3) had 0x0 (Jazelle not supported). This is discussed in Erratum 766174. The encoding was corrected for r1p0.

However, the debug logic has a separate version of the ID_PFR0 register for access by external debug reads. This debug version of the ID_PFR0 register still has the 0x0 encoding for r1p0. This means that debug reads of the ID_PFR0 register will see an incorrect value.

Note: This erratum matches bugs #5158 in the ARM internal JIRA database.

Conditions

Debug read of ID_PFR0

Implications

Debug software could read a value of the ID_PFR0 register that doesn't match that value seen by software executing on A15.

Workaround

Ignore the Jazelle support encoding in the debug register version of ID_PFR0.

768532: Mapping the GIC memory mapped region as cacheable can result in unpredictable behavior or a deadlock

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If the physical address page that maps to the A15 internal GIC memory mapped registers is configured as Writeback cacheable in the page tables, a memory access to this region will generate an error and the requested line will not be placed into the L1 data cache. Unless the requesting instruction is flushed (branch or event flush) an external abort will be generated. The L1 data cache will, however, continue to evict the existing line in the data cache that was targeted for eviction by the failing cache line read. The L2 cache does not expect this eviction. This places the L2 logic in an illegal state.

If there is a snoop for the victim line while the eviction is being processed in the L2, A15 will behave unpredictably and could deadlock.

Note: This erratum matches bugs #5180 in the ARM internal JIRA database.

Conditions

1) A15 configuration that includes the internal GIC
2) Page table maps the memory region assigned to the GIC memory mapped registers as WB cacheable
3) Memory access (load or store) made to that page
4) Snoop from another master (inside or outside of A15) targets the L1 victim of that memory access

Implications

Incorrect page tables mapping the GIC to cacheable could cause unpredictable behavior on A15. In general, an external data abort would be detected and the offending process can be killed. However, if the requests that caused the illegal accesses are speculative and later flushed, no abort would be reported.

If a snoop occurs to the victim line with the correct timing relative to the eviction of that line, the part could deadlock.

In addition, if the snoop collision with the victim of the illegal GIC access occurs, A15 will be operating in unvalidated space. It is possible that data in the L1 or L2 cache could become corrupted. We have not been able to find such a case, but have difficulty proving it could not happen.

Workaround

Mapping the physical address region for the GIC to cacheable memory in all page tables avoids this erratum.

A hypervisor mapping the physical memory for the GIC registers to SO or Dev in the second stage translation will prevent a Guest OS from causing this behavior.

770320: Single bit ECC error can cause cache maintenance operation to violate memory ordering

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If a single bit ECC error occurs in the L1 data cache of A15, the L1 will make a request to the L2 logic to trigger an eviction of that line from the L1 data cache in order to correct the error. This background eviction ignores the normal memory ordering rules on cache maintenance operations. However, if there is a cache maintenance operation in flight at the time the ECC error is discovered, it is possible that this cache maintenance operation will pass a store in the machine to the same address as the cache maintenance operation. This will violate the ARM memory ordering requirements.

Note: This erratum matches bug #5223 in the ARM internal JIRA database.

Conditions

1) Store to address A has been executed, has not yet been pushed to the L1 cache or L2 cache
2) DCCMVA or DCCIMVA is executed to address A after the store to address A in program order
3) ECC error detection logic is built into the L1 cache and enabled
4) Single bit ECC error is detected in the L1
5) The DCCMVA or DCCIMVA completes and prepares to go to the memory system after the ECC error is detected and before the ECC error implicit cache maintenance operation is issued
6) Software was depending on the cache being empty or clean at the end of the DCCMVA/DCCIMVA and will fail if the store data was not pushed out

Implications

The DCCMVA or DCCIMVA could pass the store to the same address. This could leave dirty data in the cache where software would expect the cache to be clean or invalid.

This erratum is likely to be rare enough to ignore. ECC errors are inherently very rare. Cache maintenance operations occurring very close to earlier stores to the same address are likely to only exist in specialized code. The odds of a store/CMO/ECC error occurring at the same point in time is very low. In general this erratum can be ignored.

When the erratum conditions do occur, the effect will be silent data corruption. A correctable error will be reported in the syndrome registers, but there will be no way to determine that the rare boundary conditions lined up to allow data corruption. However, because the alignment of events is very rare, this is not expected to add significantly to the silent error rate.

Workaround

In general, this issue should be ignored, other than to understand that affected A15 versions do not have perfect reporting or recovery from single ECC errors.

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: r0p4

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.

774570: Fault Status bit in register DBGDSCR is implemented as sticky

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

Fault status bit[9] in debug register DBGDSCR might be improperly set for synchronous data aborts in debug state in some cases.

Note: This erratum matches bug #5278 in the ARM internal JIRA database.

Conditions

1) Core is currently in debug state.
2) If either of the following occur:
    1) Second stage synchronous data abort occurs in Non Secure PL0 or PL1 mode when external debugger issues an instruction through DBGITR debug register, or
    2) Synchronous data abort other than second stage synchronous data abort occurs in Non Secure PL0 or PL1 mode when external debugger issues an instruction through DBGITR debug register with HCR.TGE bit set to 1.
3) FS bit in debug register DBGDSCR bit gets set due to this synchronous data abort as described above.
4) FS bit is not cleared by external debugger by explicitly writing 1’b0 to bit[9] of DBGDSCR.
5) If any of the following occur:
    1) Another synchronous data abort other than second stage synchronous data abort occurs in Non Secure PL0 or PL1 mode when external debugger issues another instruction through DBGITR debug register with HCR.TGE bit set to 0, or
    2) Another synchronous data abort occurs in Non Secure PL2 mode when external debugger issues another instruction through DBGITR debug register, or
    3) Another synchronous data abort occurs in Secure state when external debugger issues another instruction through DBGITR debug register.

Implications

Read of debug register DBGDSCR fault status bit[9] returns incorrect value as 1 when it should be 0.

Workaround

Whenever the external debugger reads DBGDSCR.FS bit as 1 following a synchronous data abort caused by an instruction issued through DBGITR register, the external debugger should explicitly write 1’b0 to fault status bit[9] of DBGDSCR.

774571: Sampling registers DBGCIDSR and DBGVIDSR can return incorrect value

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

Context ID sampling register DBGCIDSR and Virtualization ID sampling register DBGVIDSR can return incorrect values in some cases.

Note: This erratum matches bug #5279 in the ARM internal JIRA database.

Conditions

1) Non invasive debug authentication status currently allows sample based profiling.
2) Debug software access lock status is set as indicated by SLK, bit[1] of DBGLSR.
3) PC sampling register DBGPCSR is read by external sampling agent through external debug interface.
4) Context ID or Virtualization ID has changed and ISB or exception entry or exception return has occurred and PC sampling register DBGPCSR is read by spurious software running on the current cpu or another cpu in the system using memory mapped debug interface before DBGCIDSR and DBGVIDSR registers are read by external sampling agent through external debug interface.

Implications

Reads of debug registers DBGCIDSR and/or DBGVIDSR might return a value which is not consistent with PC value sampled by external agent.

Workaround

This issue can occur only when a spurious read to DBGPCSR is made by software running on a cpu in the system. One possible workaround is to set up a translation table for the system register map such that it is not possible for software to access Cortex-A15 debug register space.

775622: Debug version of Cache Type Register (CTR) IminLine field are incorrectly encoded

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

Debug version of CTR.IminLine, register bits[3:0], are incorrectly encoded as 4'b0100 instead of 4'b0011 when IMINLN input of Cortex A15 is LOW.

Note: This erratum matches bug #5304 in the ARM internal JIRA database.

Conditions

This erratum requires both of the following conditions to be met.

1) The IMINLN input of Cortex A15 is LOW. This input is sampled only during reset.

2) The debug register 833 at offset 0xD04, which is an alias of the CP15 Cache Type Register CTR, is read.

Implications

This should not create any issues with an external debugger because the actual value encoded in the register has no effect on hardware operation. CTR.IminLine is used by software to determine stride for cache maintenance operations while operating on a range of addresses. The processor uses the IMINLN input to generate this value. When IMINLN is HIGH,CTR.IminLine is encoded as 4'b0100 indicating 64 bytes. When IMINLN is LOW, CTR.IminLine is encoded as 4'b0011 indicating 32 bytes. This signal does not affect internal instruction cache line size or operation of the cache maintenance commands in hardware. Cortex-A15 hardware always uses 64 bytes as the minimum instruction cache line size.

Workaround

Ignore the CTR.IminLine value in the debug register reads of register 833, the Cache Type Register (CTR). If required, an external debugger can halt the processor and read the Cache Type Register (CTR) using a CP15 instruction to determine the value used by software.

777769: ICache parity error may not be corrected for NC code

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

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.

777771: Ordering of read accesses to the same memory location may not be ensured in the case of an unaligned load

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

An unaligned 64-byte boundary crossing load hits a cache line currently being returned from memory in one of the L1 fill buffers and returns its data to a register, executing out of order ahead of load instructions that are earlier in the program. One or both of the cache lines touched by the load are then lost from the L1 data cache (due to eviction or snoop invalidate). Another master modifies the lost cache line or lines. One of the earlier load instructions (which need not be unaligned) executes, reads the modified data, and returns it to a register. A15 should detect that the earlier load has returned more recent data (in violation of the ARM memory ordering model) and flush the younger load. In this case it does not correctly detect the hazard.

If this erratum is hit these loads are observed out of program order by the master doing the stores. This violates the ARM memory ordering model.

Note: This erratum matches bug #5328 in the ARM internal JIRA database.

Conditions

1) Unaligned load executes out of order ahead of loads that are earlier in program order
2) The unaligned load touches two 64-byte aligned regions and is one of the following:
   - ldrh or ldr
   - ldm that is not 8-byte aligned
   - vld* to 32bit Si registers that is not 8-byte aligned
   - vld* to 64bit Di registers that is not 16-byte aligned
3) At least one of the lines read by the unaligned load is evicted from the L1 data cache (snooped or replaced)
4) The evicted line is modified by another master
5) One of the earlier loads now executes, brings in the modified line, and returns one of the modified bytes

Implications

In general, when one master is modifying a memory location and another is reading from it, the accesses will be synchronized with the appropriate use of semaphores or locks such that the write to memory is complete before the read is executed. Such code will not be affected by this erratum.

However, certain lock-free synchronization techniques depend on the memory order property that earlier loads will not return more recent data for a given byte in memory than later loads. If an unaligned 64-byte boundary crossing variable is used in lock-free programming it may not work correctly.

This is expected only to happen in hand-written assembly code. In a compiler, variables that are used for lock-free synchronization must be marked as ‘volatile’ to avoid normal compiler reordering. Current compilers will not place volatile variables in an unaligned memory location and will not generate code that will be affected by this erratum.

Workaround

Whenever possible, use aligned memory locations for any memory reads that will rely on in order observation by other masters. Volatile variables in the ARM and GNU compilers will meet this requirement for volatile variables.

If there is an unaligned memory read (as defined in condition 2 above) that must be observed in order with earlier reads, place a DMB instruction before it. This will correctly enforce the observation order.

777772: Device LDM may stall if memory type changed asynchronously from Device to Normal

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

A LDM/VLDM or extension register element load instruction begins execution. This load will be broken up into multiple micro-ops of 8 or 16 bytes. The first few load micro-ops hit the TLB and return translations with the memory type Strongly Ordered or Device. Before the rest of the micro-ops in the load instruction can complete, a TLB invalidate instruction from another master removes that TLB entry. The LDM therefore triggers a new tablewalk. This tablewalk returns a translation with a memory type of Normal. The CPU will be unable to complete the load. The CPU will stall until the next interrupt. If this occurs in the highest level of privilege with interrupts disabled, the CPU may deadlock.

Note: This erratum matches bug #5329 in the ARM internal JIRA database.

Conditions

1) A load instruction of greater than 8-bytes is executed
2) The current translation for the targeted page in memory is Strongly Ordered or Device
3) A TLB invalidate from another CPU is received in the middle of executing the large load instruction
4) The tablewalk for the invalidated page returns a Normal memory page (NC, write-through, or write-back)

Implications

If all the conditions occur, the CPU will stall until the next interrupt.

This is not expected to be an issue in any real systems. An OS will not remap a page from an SO/Device type to normal memory while an active process is accessing that memory.

If in a rare case the OS does shift an actively accessed page from Strongly Ordered/Device to a Normal (NC/WT/WB) memory type, it is possible that the thread accessing the page may stall until the next interrupt. As this is expected to occur extremely infrequently or never, the impact will be low.

Workaround

When remapping a virtual address page from Strongly Ordered or Device to a Normal memory type, the OS should first remap the page from the original memory type to an invalid page, execute a DSB to ensure it is complete, and then remap the page as Normal memory. This will avoid this issue.

777774: DCCMVAC/DCCSW colliding with ReadOnce from ACE could cause data corruption

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

A cache maintenance instruction (DCCSW to the level 2 cache or DCCMVAC) is executed to a line that is dirty in the A15 L1 data cache or L2 unified cache. In a coherent ACE system, this will cause A15 to generate an ACE WriteClean operation pushing out the dirty data while retaining the line in the cache in Unique Clean state. The cache state is updated and the dirty data is placed in a buffer to be sent over the ACE write channel.

A ReadOnce command for the same line is then received over the AC snoop interface. The L2 cache detects that the WriteClean is in the buffer and sends the dirty data out over the CDATA channel in response to the snoop, marking the data as dirty. A15 has no completed the snoop and the line is in UniqueClean state. The interconnect does not yet finish writing the dirty data back to memory, however.

A write from an A15 CPU or ACP then modifies that line in the L2 cache, changing its state to UniqueDirty. The line is then replaced in the L2 cache and a WriteBack is issued on the ACE write channel.

It is possible that this WriteBack will arrive at memory before the memory update generated from the CDATA snoop response. If this occurs, the most recent WriteBack data will be overwritten by the older CDATA data. This will lead to stale data in memory.

To avoid the above case, the ACE protocol states that a UniqueDirty line can't transition to UniqueClean in response to a ReadOnce. A15 violates this restriction.

Note: This erratum matches bug #5334 in the ARM internal JIRA database.

Conditions

1) Cache line in Unique Dirty state (Modified) in A15 L1 data or L2 cache
2) DCCSW or DCCMVAC instruction executed to that line
3) Before the WriteClean command is issued to the bus, a ReadOnce occurs for that line
4) The dirty data is sent on the CDATA channel in response to the snoop, but stalls in the system
5) That cache line is further modified by A15
6) That cache line is cleaned or replaced in the A15 L2 and generates a WriteBack or WriteClean
7) The WriteBack or WriteClean completes to memory before the stalled CDATA channel response above completes

Implications

If the above conditions are met, the latest data from A15 to that line will be overwritten by the stale data in the CDATA response. This is expected to be very uncommon or impossible in real systems, however.

A ReadOnce command will be received by A15 if a non-caching master is making a read request for that line. This would tend to imply that cache coherence for that memory location is being handled by the ACE hardware coherence mechanism.

If a DCCMVAC instruction (Data Cache Clean by MVA to point of coherence) is executed for a given memory location, there is software cache maintenance occurring. Typically software would be pushing data out of the L1/L2 caches so it can be seen by an external master that is not participating in ACE hardware coherence.

A DCCSW instruction is generally used only as part of loop cleaning all dirty data out of the caches in preparation for powering down A15.

This erratum can only occur if a ReadOnce occurs concurrent with one of the above software clean instructions. In the case of the DCCSW power down sequence, it is possible that a non-caching master would generate a ReadOnce while a DCCSW is executing. However, even if this occurs, during the core powerdown operation software will typically have the caches disabled and will be making no further modifications to those lines. There will be no second modification of the line to generate the second WriteBack/WriteClean required by the race condition.

In the case of software communicating with an external peripheral through a cached memory buffer, this erratum is expected to be rare for two reasons. First, generally only software or hardware coherence will be used for a given memory location, not both. This means that getting a ReadOnce and a DCCMVAC to the same location concurrently

would not be expected. Second, software will generally modify a section of memory and then clean that memory from the caches to be used by the non-hardware-coherent peripheral. While that peripheral is using the data, it will be unusual for software on the A15 to further modify that memory location. Because of this, the second WriteBack/WriteClean required to generate the race condition will be unlikely.

DCCMVAU (clean to point of unification), often used for self-modifying code, will not trigger this erratum.

Workaround

Do not execute DCCSW or DCCMVAC instructions to cache lines that might be accessed by an ACE ReadOnce command while A15 is actively modifying it. Replace any DCCSW that are not part of a powerdown sequence with DCCISW (clean/invalidate). If cache clean by MVA (DCCMVAC) must be done to a region of memory being accessed by a coherent non-caching peripheral, do a clean/invalidate by MVA (DCCIMVAC) instead.

780121: PTM might not acknowledge a trace flush request when cpu is in WFI.

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

When a trace sink requests a flush of a trace source by asserting AFVALID, the trace source normally acknowledges the flush request by asserting AFREADY after all the trace in FIFO has been output.

Under certain conditions, the PTM might not acknowledge a trace flush request even after all the trace has been output.

Note: This erratum matches bug #5346 in the ARM internal JIRA database.

Conditions

The following sequence must occur:

1) A processor power on reset or debug trace reset request

2) The PTM is enabled

3) The PTM is later disabled

4) A WFI or WFE instruction is executed

5) The processor stops its internal clocks

6) The processor is still powered up

In the above scenario after step 4, PTM will de assert AFREADYM temporarily for few processor clock cycles and few trace clock cycles. If the processor clocks are stopped before AFREADYM is asserted again, then any new flushes will not be acknowledged until the processor clocks are restarted.

AFREADYM will be correctly asserted during WFI in any of the following cases:

- The processor is powered down and clamps are activated
- The PTM is reset
- The PTM was never enabled out of trace reset
- The PTM is not disabled
- The PTM is disabled during WFI

Implications

If a trace sink initiates a flush request and then stops on the flush completion, then due to this erratum the flush never completes and the trace sink will not stop.

If the PTM is connected to a CoreSight trace funnel, then if a flush request is initiated the funnel will wait for the all trace sources to acknowledge the flush before continuing. This erratum might cause the funnel to stop accepting new trace from all other trace sources, preventing any further trace capture.

Workaround

There are two workarounds:

- If the PTM is connected to a trace funnel, after disabling the PTM, the trace analyzer must disable the corresponding trace slave port in trace funnel. This will make the trace funnel acknowledge the flush immediately.
- Instead of disabling the PTM, the PTM should be configured to generate no more trace, but must remain enabled.

784419: An unaligned load crossing a 4K boundary between SO/Dev memory and WB memory can stall the core until the next interrupt

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If an unaligned load crosses a 4k page boundary between two pages where the lower page is mapped to Strongly-ordered or Device memory and the upper page is mapped to Normal, Write-Back Cacheable memory, in certain rare cases the core may stall until the next interrupt. A memory transaction that crosses a boundary between these page types is architecturally UNPREDICTABLE and should not occur in any real code. However, if this were to occur at the highest privilege level with interrupts disabled, it could deadlock that CPU.

Note: This erratum matches bug #5355 in the ARM internal JIRA database.

Conditions

1) A load instruction is executed that accesses bytes from two different 4k pages
2) The lower page is Device or Strongly-ordered memory type
3) The upper page is Normal Write-Back Cacheable memory, that is also Read-Allocate, Write-Allocate, or both
4) The load instruction is one of the following:
   - LDRH or LDR
   - an LDM that is not 8-byte aligned
   - a VLD* to 32bit Sd registers that is not 8-byte aligned
   - a VLD* to 64bit Dd registers that is not 16-byte aligned

Implications

If the above conditions occur, it is possible that in rare cases the core will stall until the next interrupt. If this occurs in a situation where no interrupt will occur, the CPU deadlocks. Examples of where a deadlock might occur are if there is no timer interrupt in the system, or the conditions occur at the highest privilege level with interrupts disabled.

Workaround

Do not execute loads that cross between Strongly-ordered or Device memory and Normal memory pages. Architecturally, such a load is UNPREDICTABLE.

784469: CTI Authentication Status register is incorrect

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

The AUTHSTATUS register is a read-only register in the CTI that reports the debug level supported by the CTI and the current status of the debug level.

The CoreSight Architecture Specification specifies bits [3:0] in the AUTHSTATUS register as below:

- [3:2] Non-Secure Non-Invasive Debug
- [1:0] Non-Secure Invasive Debug

For each of these fields, the value of the status bits as returned by the CTI and their meanings are specified as below:

Value Description

2'b10 Functionality disabled

2'b11 Functionality enabled

In the CTI each pair of bits ([3:2] and [1:0]) in the AUTHSTATUS register currently read:

- When functionality is disabled - 2'b01

but should read (as per table above):

- When functionality is disabled - 2'b10

The bits are swapped.

Condition 1

AUTHSTATUS[1:0] - Non-secure Invasive Debug

- DBGEN input to the CTI is LOW
- AUTHSTATUS register is read

Condition 2

AUTHSTATUS[3:2] - Non-secure non-Invasive Debug

- NIDEN input to the CTI is LOW
- DBGEN input to the CTI is LOW
- AUTHSTATUS register is read

Implications

The status of the debug level supported by the CTI as returned by the AUTHSTATUS register read is incorrect. The masking of trigger inputs and outputs using DBGEN and NIDEN is not affected by this erratum. The return of an incorrect value might lead to incorrect operation of debug tools.

Workaround

This is a workaround for users and tools vendors. When reading the AUTHSTATUS register, swap the bits in the affected fields and interpret the read data accordingly.

803620: Near zero Advanced SIMD fused multiply add may be incorrectly flushed to zero.

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

In very rare cases, a very small single-precision Advanced SIMD fused multiply add (VFMA/VFMS) that should have the result 2^-125 (two times the smallest nonzero number supported in Advanced SIMD) will instead be flushed to zero. Only fused multiply adds that should round to this very small value are affected, and not even all of those are affected.

Note: This erratum matches bug #5438 in the ARM internal JIRA database.

Conditions

1) Advanced SIMD FMA instruction is executed.
2) Correct result of the fused multiply add is 2^-125.
3) The product and the augend have opposite signs.
4) The fraction for the sum is exactly zero with a carry out.

Implications

In the rare case that the erratum conditions are hit, the result will be zero rather than 2^-125. Advanced SIMD already rounds any number less than 2^-126 to zero. It is not expected that this erratum will have a noticeable effect in typical applications.

Because Advanced SIMD already rounds subnormals to zero, it is expected that most code that cannot tolerate the rounding of extremely small numbers to zero will be using the VFP versions of these instructions (which are not affected by the erratum).

Workaround

This erratum only applies to the Advanced SIMD version of the VFMA/VFMS. The VFP versions of VFMA and VFMS return the correct result. Code that cannot tolerate the rounding of very small results to zero should use the VFP versions of VFMA and VFMS

803670: Unaligned load accessing two cache lines could return uncorrected data in case of single bit ECC error in the L2 cache

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

If a load micro-operation accesses memory in two cache lines, misses the L1 cache for both cache lines, and hits a single-bit ECC error in the L2 for the first cache line, uncorrected data can be returned.

If ECC is not configured or is not enabled, the erratum is not possible.

Note: This erratum matches bug #5423 in the ARM internal JIRA database.

Conditions

1) An unaligned load is executed that touches two 64-byte aligned regions and is one of the following:
   - LDRH or LDR
   - An LDM that is not 8-byte aligned
   - VLD* to 32bit Si registers that is not 8-byte aligned
   - VLD* to 64bit Di registers that is not 16-byte aligned
2) Both 64-byte cache lines miss in the L1 cache.
3) The lower cache line hits the L2 cache with a single bit ECC error in the data array.
4) At the point the bad data returns all previous loads in the machine have completed.

Implications

If the erratum conditions occur, uncorrected data can be returned for the load and no abort is signalled. The L2 memory error syndrome register is correctly updated to reflect a single bit ECC error. The data will be corrected in the L2 cache and future loads will return correct data.

Although unaligned loads that cross 64-byte aligned boundaries can occur, they are expected to be much less common than aligned transactions.

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.

803671: The ACE ARDOMAIN field may be incorrect on some Instruction or Tablewalk prefetch requests

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4

Description

The A15 will prefetch cache lines that are expected to be needed in the near future based on current instruction fetches and translation table walks. In certain rare cases, a prefetch request for a table walk cache line might use the shareability attributes from a recent instruction fetch request, or an instruction prefetch might use the shareability attributes from a recent table walk. If this occurs and the prefetch request misses in the L2 cache, a cache line read request is issued on the ACE AR channel with an incorrect ARDOMAIN field. This might lead to incorrect snooping behavior in the ACE interconnect.

If a prefetch read is issued with higher shareability than it should have (outer-shareable rather than inner-shareable or non-shareable, inner-shareable rather than non-shareable), there will be no functional impact, although very rarely an unnecessary snoop might be issued. However, if a prefetch read is issued with lower shareability than it should have, there could be a coherence issue. The prefetch request might be issued as ReadNoSnoop and not correctly snoop other caches in the system.

Note: This erratum matches bug #5441 in the ARM internal JIRA database.

Configurations Affected

ACE Systems only

Conditions

1) Instruction prefetch or table walk descriptor prefetch are enabled, because either:
   - L2PFR[8:7] is not zero
   - OR L2PFR[10] is zero
2) Instruction fetch requests (IF request) and translation table walk descriptor fetches (TBW request) are both being issued,
3) The IF requests and TBW requests have different shareability attributes.
4) Rare timing boundary conditions occur.

Implications

In general, this is not expected to be a problem in most systems. In a non-ACE system, the errata cannot occur (because A15 snoops all L1 caches for all cacheable requests regardless of shareability). In an ACE system, it is expected that code and translation tables will exist in cacheable memory regions with consistent shareable attributes (either outer-shareable or inner-shareable).

Workaround

To avoid this erratum, use the same shareability attribute (non-shareable, inner-shareable, or outer-shareable) for all translation table memory and executable instruction memory. This is the preferred workaround.

Alternatively, the erratum can be avoided by disabling both instruction prefetch and table walk descriptor prefetch (by setting L2PFR[10] = 1 and L2PFR[8:7] = 0). This may cause some performance degradation depending on the application.

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: r0p4.

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: r0p4.

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: r0p4.

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: r0p4.

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 ordering on a load forwarding from a store crossing a 16-byte boundary

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r0p4.

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: r0p4.

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: r0p4.

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: r0p4.

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 (DBGDSC.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.