ARM® Cortex™-A9 processors

r0 releases

Software Developers Errata Notice

ARM UAN 0005D (ID032315)

ARM Cortex-A9 processors
Software Developers Errata Notice

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
18 June 2012	A	Non-confidential	First release for r0 releases, limited distribution
20 September 2012	B	Non-confidential	Second release for r0 releases
18 February 2013	C	Non-confidential	Third release for r0 releases
25 March 2015	D	Non-confidential	Fourth release for r0 releases

Proprietary Notice

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents

ARM Cortex-A9 processors Software Developers Errata Notice

Chapter 1
Introduction

This chapter introduces the errata notices for ARM Cortex™-A9 processors, for r0 releases.

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 1-1 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.

1.3 Errata summary

Table 1-2 lists all the errata described in this document. The Status column shows any errata that are new or updated in the current issue of the document. An erratum is shown as updated if there has been any change to the text of the erratum Description, Conditions, Implications or Workaround. Fixed errata are not shown as updated, unless the erratum text has changed.

Table 1-2 List of errata

Status	ID	Area	Cat	Rare	Summary of erratum
-	571622	Prog	A	-	SPSR state corruption is possible where a serial instruction precedes a state-changing predictable branch
-	754319	Prog	A	-	A sequence of cancelled Advanced-SIMD or VFP stores might deadlock
-	754320	Prog	A	-	A cancelled Advanced-SIMD or VFP load multiple of more than 8 beats might deadlock
-	548877	Prog	A	Rare	Three-way coherency deadlock
-	745320	Prog	A	Rare	A Floating Point write following a failed conditional read might write corrupted data
-	761319	Prog	A	Rare	Ordering of read accesses to the same memory location might be uncertain
-	523319	Prog	B	-	Watchpoint hit on LDM/VLDM treated as Uncacheable might cause core deadlock
-	571620	Prog	B	-	Automatic data prefetcher might deadlock in case of hazards between the two possible strides
-	571771	Prog	B	-	Memory barriers might not ensure ordering against some broadcast cache maintenance operations
-	643767	Prog	B	-	CPSR flag corruption when a FMSTAT follows a flushed FCMP
-	643923	Prog	B	-	Debug can be entered even when SPIDEN is low
-	716049	Prog	B	-	Processor state corruption on simultaneous external debug request and state-changing branch execution
-	720789	Prog	B	-	TLBIASIDIS and TLBIMVAIS operations might broadcast a faulty ASID
-	751469	Prog	B	-	Overflow in PMU Counters might not be detected
-	751472	Prog	B	-	An interrupted ICIALLUIS operation might prevent the completion of a following broadcast operation
-	751476	Prog	B	-	Missed watchpoint on the second part of an unaligned access crossing a page boundary
-	754327	Prog	B	-	With no automatic Store Buffer drain, visibility of written data requires an explicit memory barrier
-	764369	Prog	B	-	Data or unified cache line maintenance by MVA fails on Inner Shareable memory
-	782773	Prog	B	-	Updating a translation entry to move a page mapping might erroneously cause an unexpected translation fault

Table 1-2 List of errata (continued)

Status	ID	Area	Cat	Rare	Summary of erratum
-	794073	Prog	B	-	Speculative instruction fetches with MMU disabled might not comply with architectural requirements
-	794074	Prog	B	-	A write request to Uncacheable, Shareable normal memory region might be executed twice, possibly causing a software synchronisation issue
-	571618	Prog	B	Rare	PLI or PLD before interrupted CP15 operation might cause deadlock
-	578565	Prog	B	Rare	PLI or PLD causing a two-level PTW and Uncacheable LDM might deadlock
-	751473	Prog	B	Rare	Under very rare circumstances, automatic data prefetcher might cause deadlock or data corruption
-	761320	Prog	B	Rare	Full cache line writes to the same memory region from at least two processors, might deadlock the processor
New	845369	Prog	B	Rare	Under very rare timing circumstances, transitioning into streaming mode might create a data corruption
-	548867	Prog	C	-	Debug entry can be faulty while in Jazelle state
-	548871	Prog	C	-	Spurious abort reported on an SWP following an aborted Strongly Ordered access
-	571467	Prog	C	-	No exit from WFE when in NS state because AMO/IMO/IFO does not mask the CPSR A/I/F bits
-	571619	Prog	C	-	Imprecise abort reported on a PLD
-	571768	Prog	C	-	TLB invalidate by MVA all ASID should not invalidate locked entries
-	571773	Prog	C	-	SWP might interfere with a Cacheable Page Table Walk request, causing deadlock
-	605225	Prog	C	-	Possible double allocation in the cache when a PLD and a SWP are performed on the same address
-	605226	Prog	C	-	Domain fault on a cache maintenance operation by MVA might cause the operation to be performed on the wrong cache line
-	605230	Prog	C	-	DBGACK and DBGDONE signals might remain asserted after a CPU reset
-	643718	Prog	C	-	Priority given to breakpoints over MMU fault
-	643719	Prog	C	-	LoUIS bit field in CLIDR register is incorrect
-	643769	Prog	C	-	Missing reset on DBGCLAIMCLR register
-	643921	Prog	C	-	Writing 0 to DRCR[1] and DRCR[0] should be ignored
-	643922	Prog	C	-	Software lock writes are not ignored when performed from an external debugger
-	693170	Prog	C	-	In debug state, IRQ/FIQ can interrupt some CP15 maintenance operation
-	693171	Prog	C	-	DBGDSCR internal and external views are inverted

Table 1-2 List of errata (continued)

Status	ID	Area	Cat	Rare	Summary of erratum
-	693172	Prog	C	-	Data abort does not interrupt Non-Blocking execution from ITR
-	719331	Prog	C	-	Data prefetcher can cause a processor deadlock when executing a WFI
-	719332	Prog	C	-	Uncacheable SWP or SWPB instruction can be corrupted by the Data Prefetcher
-	721147	Prog	C	-	TLBIMVAA and TLBIMVAAIS operations might not invalidate all required TLB entries when using multiple page sizes
-	721958	Prog	C	-	TLBIMVAA operations might not invalidate all required TLB entries when multiple page sizes are used
-	725631	Prog	C	-	ISB is counted in Performance Monitor events 0x0C and 0x0D
-	729817	Prog	C	-	MainID register alias addresses do not map to the Debug interface
-	729818	Prog	C	-	In debug state, next instruction is stalled when SDABORT flag is set, instead of being discarded
-	740663	Prog	C	-	Event 0x68 / PMUEVENT[9:8] might be inaccurate
-	743623	Prog	C	-	Bad interaction between a minimum of seven PLDs and one Non-Cacheable LDM, can lead to deadlock
-	743625	Prog	C	-	A coherent ACP request might interfere with a non-cacheable SWP/SWPB from the processor, potentially causing deadlock
-	743626	Prog	C	-	An imprecise external abort received while the processor enters WFI may cause a processor deadlock
-	751471	Prog	C	-	DBGPCSR format is incorrect
-	751480	Prog	C	-	Conditional failed LDREXcc can set the exclusive monitor
-	756421	Prog	C	-	Sticky Pipeline Advance bit cannot be cleared from debug APB accesses
-	757119	Prog	C	-	Some *Unallocated memory hint* instructions generate an Undefined Instruction exception instead of being treated as NOP
-	761321	Prog	C	-	MRC and MCR are not counted
-	764319	Prog	C	-	Read accesses to DBGPRSR and DBGOSLSR might generate an unexpected Undefined Instruction exception
-	771221	Prog	C	-	PLD instructions might allocate data in the Data Cache regardless of the Cache Enable bit value
-	771224	Prog	C	-	Visibility of Debug Enable access rights to enable/disable tracing is not ensured by an ISB
-	775419	Prog	C	-	PMU event 0x0A (exception return) might count twice the LDM PC ^ instructions with base address register write-back
-	795769	Prog	C	-	“Write Context ID” event is updated on read access
-	799770	Prog	C	-	DBGPRSR Sticky Reset status bit is set to 1 by the CPU debug reset instead of by the CPU non-debug reset

Chapter 2
Errata Descriptions

This chapter includes the errata descriptions for ARM Cortex™-A9 processors, for r0 releases.

2.1 Category A

This section describes the Category A errata.

2.1.1 (571622) SPSR state corruption is possible where a serial instruction precedes a state-changing predictable branch

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category A

Fault Status: Present in: r0p0. Fixed in r0p1.

Description

Under some rare conditions, precise instruction sequences involving a predicted Branch changing the state from ARM to Thumb, or from Thumb to ARM, occurring at the same time as an external interrupt, might result in the SPSR corruption.

Conditions

1. One of the following instructions executes:
   a. Conditional failed SWI
   b. Conditional failed SMI
   c. Conditional failed Undef
   d. MSR SPSR.
2. This instruction precedes a non-serial instruction, which is not a taken branch.
3. The second or the third instruction after the serial instruction is a branch which changes state, and this branch is predicted. This means that the branch has already been executed at least once, and that the state change is from ARM to Thumb, or from Thumb to ARM.
4. An interrupt is received.

If the interrupt occurs while the predicted branch is in the decoder stage of the Cortex-A9 pipeline, the SPSR can update using the wrong state.

Implications

Because the SPSR state is corrupted, the return from the interrupt is performed in the wrong state, resulting in the execution of faulty code.

The most common situation in which this errata might occur is as a result of a conditional VFP or Advanced SIMD instruction which becomes UNDEFINED because the VFP or Advanced SIMD functionality is disabled because of the FPSCR or CPACR settings. This can occur, for example, as a result of the use of Lazy Context Switching by an Operating System.

Workaround

The best workaround is to avoid placing in the second or third position after a serial instruction any predictable branch instruction which changes the core state. Inserting NOPs between the instructions would solve the issue.

An alternative workaround is to disable branch prediction if the code contains some predictable branches which change the core state. The implication of this workaround would probably be a very significant decrease in the performance of the processor.

2.1.2 (754319) A sequence of cancelled Advanced-SIMD or VFP stores might deadlock

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore
Fault Type:	Programmer Category A
Fault Status:	Present in: All r0, r1, r2 and r3 revisions. Fixed in r4p0, and FPU/MPE revision 0x4. For all revisions, an ECO fix is available in the MPE logic. All designs having implemented the ECO are identifiable by reading FPSID.Revision=`0x4`.

Description

A store is *cancelled* in the following circumstances:

- if it fails its condition-code
- if it generates a precise abort
- if it forms part of speculative execution which is subsequently discarded as a result of the processor identifying a branch miss-predict scenario.

A store is *queued* after the integer core has determined the address of the transaction. A store can optionally not be queued if the integer core can determine that the store will be cancelled while the address computation is being performed, for example if the store is conditional and the CPSR flags are already ready.

A store is considered *ready* when the values of the operands to be stored have been computed. The order in which a sequence of stores become ready can differ from program order if the operation required to compute a result for an earlier store takes longer than that of the result required for a store that occurs later in the program order.

A store is considered *committed* after it is ready and can no longer be cancelled; in other words, after the value to be stored is known, the instruction passes any condition-code check, and any branch-prediction is known to be correct.

An Advanced-SIMD (Neon) store, or a VFP store executed as part of a Neon code sequence, can cause deadlock in the following circumstances:

1. The sequence results in the first store in the instruction order becoming ready and committed later than a set of following stores.
2. The subsequent stores become queued so that the second store is cancelled, a third store is committed, and a final store is cancelled.

Conditions

Group A is the set of Advanced-SIMD (Neon) or VFP stores whose data fits within a single 64-bit aligned 64-bit memory location, and which are issued by the Neon unit in a single cycle. This group contains all addressing mode forms of:

1. Single-precision 32-bit VFP stores:
   ```
   VSTR.32  St,[Rn,...]
   ```
2. Neon single D-register structure stores with alignment specifiers equal to 64-bits:
   ```
   VST1.8 {Dd},[Rn@64]
   VST1.16 {Dd},[Rn@64]
   VST1.32 {Dd},[Rn@64]
   VST1.64 {Dd},[Rn@64]
   ```
3. Neon indexed single 8-bit element store from one lane:
   ```
   VST1.8 {Dd[<index>]},[Rn]
   ```

4. Neon indexed single larger than 8-bit element store from one lane with an alignment specifier:
   ```
   VST1.16 {Dd[<index>]},[Rn@16]
   VST1.32 {Dd[<index>]},[Rn@32]
   ```
5. Neon indexed multi-element store of no more than 8-bytes with an alignment specifier:
   ```
   VST2.8  {Dd[<index>],Dd2[<index>]},[Rn@16]
   VST2.16 {Dd[<index>],Dd2[<index>]},[Rn@32]
   VST2.32 {Dd[<index>],Dd2[<index>]},[Rn@64]
   VST4.8  {Dd[<index>],Dd2[<index>],Dd3[<index>],Dd4[<index>]},[Rn@32]
   VST4.16 {Dd[<index>],Dd2[<index>],Dd3[<index>],Dd4[<index>]},[Rn@64]
   ```

The conditions for the erratum arise only when the following sequence occurs:

1. The Advanced-SIMD (Neon) extension is present, enabled and active (see below).
2. A group-A Neon or VFP store is queued, but remains not ready.
3. A Neon or VFP store is queued and cancelled, occupying only one 64-bit slot.
4. A group-A Neon or VFP store is queued and committed.
5. A Neon or VFP store is queued and cancelled.
6. The Neon or VFP store in (2) becomes ready and is committed.

Neon is active if an Advanced SIMD (Neon) instruction has been decoded more recently (including speculatively) than an instruction from any of the following categories:

1. A VFP data-processing instruction
2. A VFP-to-VFP register move (including moving a register to itself as a NOP)
3. A VFP condition code transfer (`VMRS APSR_nzcv, FPSCR`, that is FMSTAT)
4. A transfer (VMSR) from an integer register to a VFP system register
5. FLDMX/FSTMX (deprecated)

A VFP load/store instruction, or move between VFP and integer registers, has no effect on whether Neon is active. In particular, VFP/Neon register save/restore sequences typically occurring in context switches have no effect on whether Neon is active, unless they use the deprecated FSTMX/FLDMX forms.

Neon may be known to be inactive if there have been no Neon instructions decoded (including on any possible speculative paths) since the most recent instruction from one of the above VFP categories.

As an example:

```
LDR       r5,[r4]

CMP       r5,r5         ; delayed update of CPSR flags

VMUL.U32  d0,d0,d0      ; slow operation updating d0 (s0 and s1)

VSTR      s0,[r0]

...

VSTRNE    s2,[r1]       ; cancelled

...

VSTR      s3,[r2]       ; committed

...

VSTRNE    s4,[r3]       ; cancelled
```

Note that it is not a requirement for the Neon or VFP store instructions to be consecutive in the program. It is possible to separate these using a number of ARM register based instructions, branches or any other sequence not provided in the workarounds below. In addition the cancelled stores can reside in speculated branch shadows, and might not form part of a simple sequential execution of the same code.

This erratum is timing sensitive and is influenced by the relative cycle timings of the CPSR flags, branch prediction results, loads, stores, and both Neon and non-Neon related instructions becoming ready, all of which might not be predictable.

Implications

Only implementations of Cortex-A9, including the Multimedia-Processing Engine (MPE) / Neon unit, are affected by this erratum. Execution of a code sequence that stimulates this erratum ultimately results in deadlock. Neon stores that are committed after the erratum is triggered, but before deadlock occurs, might use the correct address but store out-of-sequence Neon data.

Workaround

This erratum does not affect implementations that do not have Neon, or implementations that operate with Neon disabled. You can use the ASEDIS bit to disable the Neon functionality and avoid this erratum, while still retaining VFP floating-point capabilities.

You can enable software that uses Neon to work around this erratum by inserting any of the following:

- One or more Neon or floating-point instructions between conditions 2 and 3.
- Two or more Neon or floating-point instructions between conditions 3 and 4.
- One or more Neon or floating-point load instructions between conditions 3 and 4.
- One or more Neon or floating-point instructions between conditions 4 and 5.

When performing code insertion, take care not to re-create an alternative sequence susceptible to this erratum.

You can enable software using Neon to work around this erratum by substituting the instructions in Group A with those not from Group A as follows:

1. You can correct instructions that contain alignment specifiers by substitution with the identical instruction minus the alignment specifier.
2. VSTR instructions with zero immediate offset may be code substituted a single word VSTM, for example, `VSTR s3,[r0]` can be replaced with `VSTM r0,{s3}`.

Note

Note that this code substitution does not provide equivalents for single-lane variant of VST1.8 or for VSTR instructions with non-zero offsets. For these instructions, code insertion remains the preferred corrective action.

Suitable instructions for code insertion include, `VORR d0,d0,d0` for Neon intensive code and `VMOV.F32 s0,s0` for VFP intensive code. Alternatively, you can insert an additional `VSTR.F64` to a scratch location.

Alternatively, software that does not use Neon can avoid the need to apply the workaround by ensuring that it does not enter the code (including by way of an exception return) with Neon active. For exception returns to software that is not using Neon, you can ensure this by having a `VMOV.F32 s0,s0`, or other VFP instruction, that prevents Neon being active (as described above) in the final basic block before the exception return, and by ensuring that the bit pattern immediately following the exception return instruction does not correspond to an instruction that makes Neon active.

2.1.3 (754320) A cancelled Advanced-SIMD or VFP load multiple of more than 8 beats might deadlock

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore

Fault Type: Programmer Category A

Fault Status: Present in: All r0, r1, r2 and r3 revisions. Fixed in r4p0, and FPU and MPE revision 0x4

For r0, r1, r2 and r3 revisions, an ECO fix is available in the MPE logic. All designs having implemented the ECO are identifiable by reading FPSID.Revision=`0x4`.

Description

A *beat* is required for each 64-bit aligned chunk of 64-bits required to service a load. This erratum affects VLDM and VPOP instructions that require more than 8 beats only. This includes all Advanced-SIMD or VFP load multiple instructions which load:

- greater-than 8 double-word D-register or 16 single-word S-registers from a 64-bit aligned address
- greater-than 7 double-word D-register or 15 single-word S-registers from a less than 64-bit aligned address.

A load is *cancelled* if it fails its condition code, or if it forms part of speculative execution which is subsequently discarded because the processor identifies a branch miss-predict scenario.

A speculative Advanced-SIMD or VFP load multiple of more than eight beats speculatively executed by the Neon unit executed as part of a Neon code sequence, but subsequently cancelled because of miss-speculation, can result in deadlock if followed by a committed Advanced-SIMD or VFP load.

Conditions

1. The Advanced-SIMD (Neon) extension is present and enabled.
2. A Neon or VFP load multiple of more than 8-beats is speculatively issued
3. A Neon or VFP load is issued and not cancelled.
4. The Neon or VFP load in (2) is cancelled.

As an example:

```
CMP    r0,r0; ensure EQ condition passes

BEQ    1      ; conditional branch to VLDR

VLDM   r0,{d0-d9} ; cancelled due to branch miss-predict

...

1:       VLDR   s0,[r1]  ; executed and not cancelled
```

This erratum is timing sensitive and is influenced by the relative cycle timings of the CPSR flags, branch prediction results, loads, stores, and the Neon, VFP and integer instructions becoming ready, all of which might not be predictable.

Implications

This erratum affects only implementations of Cortex-A9 that include the Multimedia-Processing-Engine (MPE) or Neon unit. Execution of a code sequence that stimulates this erratum might result in deadlock.

Workaround

This erratum does not affect implementations without Neon, or those that operate with Neon disabled. You can use the ASEDIS bit to disable the Neon functionality and avoid this erratum, while still retaining VFP floating-point capabilities.

Software that operates with the Advanced-SIMD (Neon) extension enabled and executing Neon software can work around this erratum by splitting load multiples that are capable of generating more than 8-beats into smaller loads.

Note that compliance to the AAPCS ensures that, on exit from a public interface, this erratum will not affect a callee restore of the form `VPOP {d8-d15}` because of the AAPCS requirement for 64-bit stack-alignment.

2.2 Category A (Rare)

This section describes the Category A rare errata.

2.2.1 (548877)Three-way coherency deadlock

Status

Affects: Product Cortex-A9 MPCore.

Fault Type: Programmer Category A (Rare)

Fault Status: Present in: r0p0. Fixed in r0p1.

Description

Under rare circumstances, where a cache line is shared by three coherent agents, the snooping mechanisms within the processor and the Snoop Control Unit can deadlock.

Conditions

1. The system contains either:
    a. three or more CPUs in an SMP configuration, OR
    b. two CPUs in an SMP configuration and the ACP port is making coherent requests.
2. A memory location this WB cacheable and marked as Shareable is shared between the processors.
3. The Outstanding access capability is enabled. This is the default.

Implications

A system with three or more coherent agents might occasionally deadlock.

Workaround

A software workaround is possible by disabling the outstanding access capability by setting bit [12] in the undocumented RESERVED register CP15, 0, C15, C0, 1.

Disabling this has a performance impact by reducing the amount of memory system parallelism.

2.2.2 (745320) A Floating Point write following a failed conditional read might write corrupted data

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category A (Rare)
Fault Status:	Present in: All r0, r1 and r2 revisions* Fixed in r3p0, and FPU/MPE revision 0x3

* For r0, r1 and r2 revisions, an ECO fix is available in the FPU or MPE logic. You can read FPSID.Revision=`0x3` to identify all designs that have implemented the ECO.

Description

Under certain conditions specific to the Cortex-A9 micro-architecture, a floating point write operation which executes a few cycles after a failed conditional read might write corrupted data to the destination address.

Conditions

The erratum is present only in processor configurations where a Data Engine is present (either the FPU, or the Neon SIMD engine which automatically includes the FPU).

The processor needs to execute a conditional load instruction which fails its condition check.

An FPU write operation occurs (VSTR/VSTM/VPUSH/FSTMX) within a few cycles following the conditional load instruction.

Under specific timing conditions, the failed conditional read might interfere with the floating point write, possibly causing a data corruption for the written data.

Implications

The erratum might cause a data corruption on the floating point data.

Workaround

For manual programming, or if the compiler offers the capability, a possible software workaround is to prevent the use of conditional loads in programs which make use of the floating point unit.

There is no practical software workaround if it is not possible to avoid the offending sequence.

2.2.3 (761319) Ordering of read accesses to the same memory location might be uncertain

Status

Affects:	Product Cortex-A9 MPCore
Fault Type:	Programmer Category A (Rare)
Fault Status:	Present in: All revisions Open

Description

The ARM architecture and the general rules of coherency require reads to the same memory location to be observed in sequential order.

Because of some internal replay path mechanisms, the Cortex-A9 can see one read access bypassed by a following read access to the same memory location, thus not observing the values in program order.

Conditions

The erratum requires a Cortex-A9 MPCore configuration with two or more processors or more.

The erratum can occur only on a processor working in SMP mode, on memory regions marked as Normal Memory Write-Back Shared.

Implications

The erratum causes data coherency failure.

Workaround

The majority of multi-processing code examples follow styles that do not expose the erratum. Therefore, this erratum occurs rarely and is likely to affect only very specific areas of code that rely on a read-ordering behavior.

There are two possible workarounds for this erratum:

- The first possible workaround is to use LDREX instead of standard LDR in volatile memory places that require a strict read ordering.
- The alternative possible workaround is the recommended workaround for tool chains integration. It requires insertion of a DMB between the affected LDR that requires this strict ordering rule.

For more information about integrating the workaround inside tool chains, see the Programmer Advice Notice related to this erratum, *ARM UAN 0004A*.

2.3 Category B

This section describes the Category B errata.

2.3.1 (523319) Watchpoint hit on LDM/VLDM treated as Uncacheable might cause core deadlock

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: r0p0. Fixed in r0p1.

Description

A watchpoint hit on an LDM or VLDM treated as Uncacheable might cause the Load-Store Unit and the Bus Interface Unit to go out of synchronization, resulting in deadlock of the Data Side memory system.

The LDM or VLDM is treated as Uncacheable in the following cases:

1. The LDM or VLDM occurs while the Data Cache is Off.
2. The LDM or VLDM is targeting a memory region marked as Strongly Ordered, Device, Normal Memory Non-Cacheable or Normal Memory Write-Through.
3. The LDM or VLDM is targeting a memory region marked as Shareable Normal Memory Write-Back, and the CPU is in AMP mode.

Conditions

1. A watchpoint is set and enabled on an address.
2. The watchpoint is programmed so that a load at this address hits, that is, the watchpoint is to be taken on loads only, or on both loads and stores.
3. An access to this address is treated as Uncacheable, which occurs when any one of the following conditions is met:
    a. The address is in a memory region marked as Strongly Ordered, Device, Normal Memory Non-Cacheable or Normal Memory Write-Through
    b. The address is in a memory region marked as Normal Memory Write-Back, and the CPU is in AMP mode
    c. The Data Cache of the CPU is not enabled.
4. SPIDEN, SUIDEN and DBGEN settings permit the watchpoint to be taken.
5. An LDM or VLDM occurs on an address range which includes the watchpoint address, with all correct access rights to generate a watchpoint hit.
6. The watchpoint hit does not hit on an address located between the LDM or VLDM start address and the next 64-bit boundary address.

As examples:

1. A VLDM starting at address 0x0 generating a watchpoint hit at address 0x0 to 0x7 does not create any issue.
2. A VLDM starting at address 0x0 generating a watchpoint hit at address 0x8 or above causes the issue.
3. A VLDM starting at address 0x7 generating a watchpoint hit at address 0x7 does not create any issue.
4. A VLDM starting at address 0x7 generating a watchpoint hit at address 0x8 or above causes the issue.

Implications

This erratum means that you should not set read watchpoints on Non-Cacheable Normal, Write-Through Normal, Device and Strongly-Ordered memory regions if there is any possibility that these memory regions are accessed using LDM or VLDM instructions.

Similarly, you should not set read watchpoints on Shareable Write-Back Normal memory regions when the CPU is in AMP mode, or on any memory location when the CPU Data Cache is not enabled, if there is any possibility that these memory regions are accessed using LDM or VLDM instructions.

Workaround

You can avoid this erratum by programming the watchpoint to be taken only on store instructions.

You can also avoid this erratum by not setting any watchpoint on a memory region which can be accessed with LDM or VLDM instructions generating Uncacheable accesses. This workaround decreases the observability of this region.

2.3.2 (571620) Automatic data prefetcher might deadlock in case of hazards between the two possible strides

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

The erratum can occur only when the automatic data prefetcher is enabled, that is, when bit[2] in the CP15 Auxiliary Control register is set.

Under some rare circumstances, a conflict on the same address between the two prefetch strides, or between one prefetch stride and a CP15 cache maintenance operation, might cause the erratum to occur.

Implications

In the uniprocessor version, or in the multiprocessor version with a single CPU and no ACP, the erratum might create a deadlock, or might corrupt the data by omitting to evict one cache line.

In the multiprocessor version of the processor, with more than two processors, or with one processor and the ACP, the processor does not deadlock. However, some data might be corrupted because of a cache line not being evicted. This is the same as the uniprocessor case.

Workaround

Because of this erratum, it is not possible to use the data automatic prefetcher reliably. The only workaround is to not enable the automatic data prefetcher. The degraded performance that arises from this workaround can be visible on some applications. However, it is possible to compensate for this by using PLD instructions to perform the prefetch.

2.3.3 (571771) Memory barriers might not ensure ordering against some broadcast cache maintenance operations

Status

Affects: Product Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: r0p0. Fixed in r0p1.

Description

In SMP mode, under some circumstances the DMB might not ensure ordering of the effects of some data cache operations with respect to explicit loads and stores.

The DMB ensures ordering of the effects of data cache operations with respect to explicit loads and stores for the effects of the data cache operation occurring on the processor executing the DMB. It does not ensure ordering of the effects of a data cache operation with respect to explicit loads and stores when the effects of the data cache operation are meant to occur on any other processor except the one that is executing the DMB.

The data cache operations affected by this erratum are:

- Clean data (or unified) cache line by MVA to the Point of Coherency: DCCMVAC
- Clean and invalidate data (or unified) cache line by MVA to the Point of Coherency DCCIMVAC.

Similarly, in SMP mode, under some circumstances a DSB might complete before the completion of a data cache operation.

The DSB still completes, after all the Cache operations that the executing processor issued before the DSB complete on the processor that is executing both the operation and the DSB. The DSB might complete before a Cache operation issued by the executing processor before the DSB completes on any other processor except the one executing the operation and the DSB, if there are other processors affected by both the DSB and the operation.

Implications

This erratum means that the memory barrier is not handled correctly, which might result in reading corrupted data.

Workaround

The workaround is to execute one additional cache clean operation, being either a DCCMVAC or a DCCIMVAC, with an associated MVA marked as Inner Shared, before executing the barrier (which could be either a DMB or a DSB). The execution of a DCCMVAC or a DCCIMVAC with an address marked as Inner Shared before the barrier will effectively enforce a DSB, hence offering the effect of the next barrier.

2.3.4 (643767) CPSR flags might be corrupted when a FMSTAT follows a flushed FCMP

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

Under some rare conditions, described below, an FMSTAT instruction executed after a flushed FCMP instruction can see the wrong flags in the FPSCR. This results in corruption of the CPSR flags.

Conditions

1. A load or store instruction executes.
2. A sequence of at least three Floating-Point instructions executes, including at least one FCMP instruction, and one FDIV or FSQRT instruction.
3. A mispredicted branch instruction executes. This branch does not depend on the first load/store instruction.
4. An FCMP instruction following the mispredicted branch executes.
5. The mispredicted branch is detected (that is, its condition code is resolved, and the branch appears to be mispredicted), so a new instruction stream executes, while the mispredicted branch and the FCMP instruction following it are flushed in the pipeline.
6. In this new instruction stream, an FMSTAT instruction executes.

Under some timing conditions specific to the Cortex-A9 micro-architecture, the FMSTAT instruction can execute before the second FCMP instruction is flushed. As a consequence, the FMSTAT copies the N, Z, C and V flags produced by the flushed FCMP, instead of those of the first FCMP.

Note

For structural reasons, compilers are unlikely to generate the failing code sequence. Compilers usually generate the FMSTAT instruction in pair with an FCMP: they first perform the comparison, before moving the flags back to the CPSR so that they can be acted upon by a conditional branch.

Implications

Due to the erratum, the CPSR N, Z, C and V flags might become corrupted, which might cause incorrect program execution.

Workaround

The only workaround for this erratum is to ensure that no isolated FMSTAT instruction is present in the code, so that an FMSTAT instruction is always coupled with a preceding FCMP instruction. This is usually the case for compiled code.

2.3.5 (643923) Debug can be entered even when SPIDEN is low

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

When the core switches from user to privilege mode, there is a one-cycle window during which debug can be entered even with the SPIDEN pin driven low.

Conditions

1. The core is in Secure state.
2. User Debug is enabled.
3. SPIDEN pin is driven low.
4. The core switches from user to privilege state.
5. A debug event occurs.

Implications

The implication for this erratum is that the debug state can be entered in regions of code where it is not allowed, which might represent a security hole in the design.

Workaround

The software workaround for this erratum is to set the SUIDEN bit low. This prevents entry to debug even when in User state.

2.3.6 (716049) Processor state can be corrupted on external debug request simultaneous with a state-changing branch execution

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All r0 and r1 revisions Fixed in r2p0.

Description

If the Cortex-A9 enters into debug state because of an external debug request, or because of a Halt Request from the DBGDRCR, and if this debug state entry happens in the next few cycles following the execution of a state-changing branch instruction (for example, BXJ or BLX), then the CPSR.T and CPSR.J bit might become corrupted.

If the CPSR.T and CPSR.J bits are corrupted, the instruction set state bit of the processor is wrong. This causes a program malfunction on exit from debug state.

Conditions

1. The core executes code which switches between different instruction set states (ARM / Thumb / Jazelle / ThumbEE).
2. Debug mode is set to Halt mode.
3. Invasive debug is enabled.
4. One of the following Halting Debug Event occurs:
   a. An external debug request occurs (EDBGRQ input pin is asserted).
   b. DBGDRCR.HaltRequest (bit[0]) bit is written.

Because invasive debug is enabled, the Cortex-A9 enters into debug halt state following the Halting debug event. Under some timing conditions specific to the Cortex-A9 micro-architecture, if the debug entry happens following the execution of a branch which switches between different instruction sets, then the CPSR.T and CPSR.J bits might become corrupted.

Note that the corruption happens if entering debug when the instruction immediately following the switching branch is not available to the decoder stage. This is quite likely to happen when the target code of the branch executes for the first time, because the code loading will probably miss in the L1 Instruction Cache and wait for the line-fill to complete. Another condition under which the target instruction might not be available to the decoder is when the branch is mispredicted, or not predicted, which requires a pipeline flush and refill.

In practice, because of the erratum, the CPSR.T and CPSR.J bits are forced to their old value (the value before execution of the state-changing branch) when entering debug.

When exiting debug state, the CPSR.T and CPSR.J bits are still corrupted, and the processor starts executing the new code using the old instruction set state.

Implications

This erratum usually results in a system malfunction, because the program cannot resume its execution properly on exit from debug.

Workaround

For a debugger, one workaround is to program a mismatch breakpoint to enter debug, instead of the usual way of programming a Halt Request. This breakpoint would hit on the first PC which does not match the programmed address in the BVR. That is:

1. BVRn = don't care.
2. BCRn = `0x00400007`:
    a. BCRn[22:20] (BVR meaning) = `0b100` (Unlinked Instruction Virtual Address mismatch).
    b. BCRn[15:14] (Security state control) = `0b00` (both Secure and Non-secure state).
    c. BCRn[8:5] (Byte address select) = `0b0000` (match on no bytes).
    d. BCRn[2:1] (Privileged mode control) = `0b11` (any mode).
    e. BCRn[0] (Enable) = `0b1`.

Because of its implementation on Cortex-A9, this breakpoint is active only after a mispredicted branch, a non-predicted branch, an exception entry/exit, or an ISB. This is compliant with the Architecture which requires such instruction to make writes to cp14 and cp15 registers architecturally visible.

So, in the case where the CPU stalls in a forever loop which only contains correctly predicted branch, without any visible interrupt, the above breakpoint is never valid and never halts the CPU. In this case, if the debugger does not see the CPU halted after a reasonable timeout, it can fall-back to a standard Halt Request.

The workaround described works in most single-core environments, but is not applicable to all systems. This workaround does not work for implementations where an on-chip resource triggers the external halt request. For example, this can be the case in a single-core environment with triggering from the trace unit, or in a multi-core environment with triggering from another core. In such environments, different alternative workarounds are possible:

1. Switch-off hardware cross-trigger support, and work with the software cross-trigger solution only. This inevitably leads to a large skid between the different processors halting.
2. The implementer can ensure that the CTI interrupt trigger out signals are routed to the on-chip interrupt controller. In this case, the Operating System (or other software) would need to cope with cross-trigger.

All workarounds described above consist in avoiding occurrence of the failure. Other workarounds have been investigated, with a view to recovering from the failing state after the erratum occurs, but there is no reliable solution in this case.

2.3.7 (720789) TLBIASIDIS and TLBIMVAIS operations might broadcast a faulty ASID

Status

Affects:	Product Cortex-A9 MPCore.
Fault Type:	Programmer Category B
Fault Status:	Present in: All r0 and r1 revisions Fixed in r2p0.

Description

A faulty ASID can be sent to the other CPUs for the broadcast CP15 TLB maintenance operations TLBIASIDIS and TLBIMVAIS. As a consequence of this erratum some TLB entries, which should be invalidated, are not. This results in an incoherency in the system page tables.

Conditions

1. MP Configuration with two or more CPUs.
2. At least two CPUs are working in SMP mode (ACTLR.SMP=1).
3. CP15 TLB maintenance operations broadcasting is enabled in at least one CPU (ACTLR.FW=1).
4. The CPU with ACTLR.FW=1 executes one of the following instructions:
   a. TLBIASIDIS (Invalidate TLB entry by ASID Inner Shareable).
   b. TLBIMVAIS (Invalidate TLB entry by MVA Inner Shareable).
5. Another CPU with ACTLR.SMP=1 contains an entry matching the broadcast ASID (for the TLBIASIDIS operation), or matching both the MVA and the ASID (for the TLBIMVAIS operation).
6. This other CPU performs a request which hits in the stale TLB entry, which should have been invalidated, instead of loading the new page entry probably defined for this ASID.

Under the conditions described, the CPU executing the maintenance operation should broadcast it to the other CPUs together with the appropriate parameters, including the ASID.

Because of this erratum, the ASID being broadcast to the other CPUs is not the correct one.

As a consequence, the entries in other CPUs, which should be invalidated, are not invalidated because the ASID does not match. Instead, some other entries with the faulty ASID can be invalidated.

Implications

Having some TLB entries erroneously invalidated does not have any significant impact. It does not create any corruption in the system and results only in a minor degradation in performance, which is probably negligible if the page table entries are cacheable in L1.

However, the TLB entries which should have been invalidated, and which are not, cause the system page table entries to be incoherent. One of the CPUs hitting in the stale, corrupted TLB entries, can then cause an indeterminate system malfunction.

Workaround

In the case of the TLBIASIDIS operation, the workaround is to replace this operation by a TLBIALLIS operation, which invalidates the entire TLB regardless of the ASID value.

In the case of the TLBIMVAIS operation, the simplest workaround is to replace the faulty TLBIMVAIS by a TLBIMVAAIS, which invalidates the TLB for all matching MVA regardless of the ASID. However, this workaround applies only when it is certain that the MVA is not mapped in page table entries that have different page sizes. See erratum 721147 for more information about this issue. If this workaround is not applicable, then it is necessary to replace the faulty TLBIMVAIS with a TLBIALLIS.

The side effect of both proposed workarounds is that they invalidate more TLB entries than originally required. This should result in a small performance penalty compared with the original code.

In the case where page table entries are marked as Inner Cacheable, it is very likely that the page entry remains in the Data Cache, even if it is no longer present in the main TLB. Therefore, in most cases, a hit in the Data Cache backs up the main TLB miss, which means any performance degradation in the system is insignificant.

In the case where the page table entries are not cacheable in L1, the performance degradation might be significantly higher.

2.3.8 (751469) Overflow in PMU Counters might not be detected

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore
Fault Type:	Programmer Category B
Fault Status:	Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

Overflow detection logic in the Performance Monitor Counters is faulty, and under certain timing conditions the overflow might remain undetected. In this case, the Overflow Flag Status register (PMOVSR) does not update as it should, and there is no interrupt reported on the corresponding PMUIRQ line.

Implications

PMU overflow detection is not reliable.

Workaround

The workaround for this erratum is to set two PMU counters to count the same event, and to explicitly offset them by 1 at the start of the count. The following sequence achieves this:

1. Disable PMU count.
2. Set up Counter0 to value N.
3. Set up Counter1 to value (N+1).
4. Enable PMU count.

This ensures that at least one of the two counters detects the overflow.

Usually, both counters trigger. Therefore, if using this workaround it is necessary for the software to reset both counters when the first one triggers.

2.3.9 (751472) An interrupted ICIALLUIS operation might prevent the completion of a following broadcast operation

Status

Affects:	Product Cortex-A9 MPCore
Fault Type:	Programmer Category B
Fault Status:	Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

In an MPCore configuration with two or more processors working in SMP mode with maintenance operation broadcast enabled, if a processor is interrupted while executing an ICIALLUIS operation, and performs another broadcast maintenance operation during its Interrupt Service Routine, then this second operation might not execute on the other processors in the cluster.

Conditions

The erratum requires an MPCore configuration with two or more CPUs working in SMP mode.

One processor has interrupts, Cache, and TLB maintenance broadcast enabled (ACTLR.FW=1'b1). This processor executes an ICIALLUIS (Invalidate All Instruction Caches Inner Shareable to Point of Unification). This instruction executes on the processor, and is broadcast to other processors in the MPCore cluster.

The processor then receives an interrupt (IRQ or FIQ), which interrupts the ICIALLUIS operation.

During the Interrupt Service Routine, the processor executes any other Cache or TLB maintenance operation which is also broadcast to other processors in the MPCore cluster.

The erratum occurs if the other processors in the cluster receive this second maintenance operation before they complete the first ICIALLUIS operation, because the other processors do not execute the second maintenance operation. This is because there is no stacking mechanism for acknowledge answers between the processors, so that the originating processor interprets the acknowledge request sent to signify the completion of the ICIALLUIS as an acknowledgement for the second maintenance operation.

Implications

Because of the erratum, the processor might hold corrupted entries in the Cache or in the TLB, causing indeterminate failures in the system.

Workaround

A software workaround for this erratum is to set bit[11] in the undocumented Diagnostic Control register placed in CP15 c15 0 c0 1.

You can write this bit in Secure state only, by using the following Read/Modify/Write code sequence:

```
MRC p15,0,rt,c15,c0,1

ORR rt,rt,#0x800

MCR p15,0,rt,c15,c0,1
```

When it is set, this bit prevents the interruption of CP15 maintenance operations.

There is unlikely to be any visible impact on system performance when using this software workaround.

2.3.10 (751476) Missed watchpoint on the second part of an unaligned access crossing a page boundary

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore
Fault Type:	Programmer Category B
Fault Status:	Present in: All revisions Open

Description

Under rare conditions, a watchpoint might be undetected if it occurs on the second part of an unaligned access that crosses a 4K page boundary and misses in the µTLB for the second part of its request.

The erratum requires a previous conditional instruction which accesses the second 4KB memory region (=where the watchpoint is set), which misses in the µTLB, and which is condition failed. The erratum also requires that no other µTLB miss occurs between this conditional failed instruction and the unaligned access, which implies that the unaligned access must hit in the µTLB for the first part of its access

Implications

A watchpoint does not trigger when it should.

Workaround

The erratum might occur in the case when a watchpoint is set on any of the first 3 bytes of a 4KB memory region, and unaligned accesses are not being faulted.

The workaround is then to set a guard watchpoint on the last byte of the previous page, and to deal with any false positive matches if they occur.

2.3.11 (754327) With no automatic Store Buffer drain, visibility of written data requires an explicit memory barrier

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All r0 and r1 revisions Fixed in r2p0

Description

The Cortex-A9 Store Buffer does not have any automatic draining mechanism. Any written data might consequently remain in this buffer, invisible to the rest of the system.

If an external agent continues to poll this memory location, waiting to see the update of the written data to make any further progress, then a system livelock might occur.

Conditions

The erratum can happen only on Normal Memory regions.

The following scenario is an example which can exhibit the erratum, where CPU0 might loop infinitely, waiting for the notification from CPU1 which remains in CPU1 Store Buffer:

- CPU0 has the spinlock locked, and is waiting in a loop for CPU1

```
1:      LDR      R0, [R1]
        CMP      R0, #1
        BNE      1
```

- CPU1 runs with interrupts disabled:

```
        MOV      R0, #1
        STR      R0, [R1] @ Notify CPU1
1:      LDREX    R1, [R2] @ try to acquire spinlock
        CMP      R1, #1
        STREXEQ  R1, R0, [R2]
        CMPEQ    R1, #1
```

Implications

Because of the erratum, a livelock situation might occur in the system.

Workaround

The workaround for this erratum is to insert a DMB after the notification write access, to ensure its visibility to any external agent.

2.3.12 (764369) Data or unified cache line maintenance by MVA fails on Inner Shareable memory

Status

Affects:	Product Cortex-A9 MPCore
Fault Type:	Programmer Category B
Fault Status:	Present in: All revisions. Open

Description

Under certain timing circumstances, a data or unified cache line maintenance operation by MVA that targets an Inner Shareable memory region might fail to propagate to either the Point of Coherency or to the Point of Unification of the system.

As a consequence, the visibility of the updated data might not be guaranteed to either the instruction side, in the case of self-modifying code, or to an external non-coherent agent, such as a DMA engine.

Conditions

The erratum requires a Cortex-A9 MPCore configuration with two or more processors, working in SMP mode, with the broadcasting of CP15 maintenance operations enabled.

The following scenario shows how the erratum can occur:

1. One CPU performs a data or unified cache line maintenance operation by MVA targeting a memory region which is locally dirty.
2. A second CPU issues a memory request targeting this same memory location within the same time frame.

A race condition can occur, resulting in the cache operation not being performed to the specified Point of Unification or Point of Coherence.

The erratum affects the following maintenance operations:

- DCIMVAC: Invalidate data or unified cache line by MVA to PoC
- DCCMVAC: Clean data or unified cache line by MVA to PoC
- DCCMVAU: Clean data or unified cache line by MVA to PoU
- DCCIMVAC: Clean and invalidate data or unified cache line by MVA to PoC.

The erratum can occur when the second CPU performs any of the following operations:

- A read request resulting from any Load instruction; the Load might be a speculative one
- A write request resulting from any Store instruction
- A data prefetch resulting from a PLD instruction; the PLD might be a speculative one.

Implications

Because it is uncertain whether execution of the cache maintenance operation propagates to either the Point of Unification or the Point of Coherence, stale data might remain in the data cache and not become visible to other agents that should have gained visibility on it.

Note that the data remains coherent on the L1 Data side. Any data read from another processor in the Cortex A9 MPCore cluster, or from the ACP, would see the correct data. In the same way, any write on the same cache line from another processor in the Cortex-A9 MPCore cluster, or from the ACP, does not cause a data corruption resulting from a loss of either data.

Consequently, the failure can only impact non-coherent agents in the systems. This can be either the instruction cache of the processor, in the case of self-modifying code, or any non-coherent external agent in the system like a DMA.

Workaround

Two workarounds are available for this erratum.

The first workaround requires the three following elements to be applied altogether:

1. Set bit[0] in the undocumented SCU Diagnostic Control register located at offset 0x30 from the PERIPHBASE address.

   Setting this bit disables the *migratory bit* feature. This forces a dirty cache line to be evicted to the lower memory subsystem, which is both the Point of Coherency and the Point of Unification, when it is being read by another processor.

   Note that this bit can be written, but is always Read as Zero.

2. Insert a DSB instruction before the cache maintenance operation.

   Note that, if the cache maintenance operation executes within a loop that performs no other memory operations, ARM recommends only adding a DSB before entering the loop.

3. Ensure there is no false sharing (on a cache line size alignment) for self-modifying code or for data produced for external non-coherent agent such as a DMA engine.

   For systems which cannot prevent false sharing in these regions, this third step can be replaced by performing the sequence of DSB followed by Cache maintenance operation twice.

Note that even when all three components of the workaround are in place, the erratum might still occur. However, this would require some extremely rare and complex timing conditions, so that the probability of reaching the point of failure is extremely low. This, and the fact that the erratum requires an uncommon software scenario, explains why this workaround is likely to be a reliable practical solution for most systems.

To ARM's knowledge, no failure has been observed in any system when all three components of this workaround have been implemented.

For critical systems that cannot cope with the extremely low failure risks associated with the above workaround, a second workaround is possible which involves changing the mapping of the data being accessed so that it is in a Non-Cacheable area. This ensures that the written data remains uncached. This means it is always visible to non-coherent agents in the system, or to the instruction side in the case of self-modifying code, without any need for cache maintenance operation.

2.3.13 (782773) Updating a translation entry to move a page mapping might erroneously cause an unexpected translation fault

Status

Affects: product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All r0, r1, r2 and r3 revisions. Fixed in r4p0

Description

Under certain conditions specific to the Cortex-A9 micro-architecture, a write operation that updates a Cacheable translation table entry might cause both the old and the new translation entry to be temporarily invisible to translation table walks, thus erroneously causing a translation fault.

Conditions

The erratum requires the following conditions to happen:

1. The processor has its Data Cache and MMU enabled.
2. The TTB registers are set to work on Cacheable descriptors memory regions.
3. The processor is updating an existing Cacheable translation table entry, and this write operation hits in the L1 Data Cache.
4. A hardware translation table walk is attempted. The hardware translation table walk can be due to either an Instruction fetch, or to any other instruction execution that requires an address translation, including any load or store operation. This hardware translation walk must attempt to access the entry being updated in condition 2, and that access must hit in the L1 Data Cache.

In practice, this scenario can happen when an OS is changing the mapping of a physical page. The OS might have an existing mapping to a physical page (the old mapping), but wants to move the mapping to a new page (the new mapping). To do this, the OS might:

1. Write a new translation entry, without cancelling the old one. At this point the physical page is accessible using either the old mapping or the new mapping.
2. Execute a DSB instruction followed by an ISB instruction pair, to ensure that the new translation entry is fully visible.
3. Remove the old entry.

Because of the erratum, this sequence might fail because it can happen that neither the new mapping, nor the old mapping, is visible after the new entry is written, causing a Translation fault.

Implications

The erratum causes a Translation fault.

Workaround

The recommended workaround is to perform a clean and invalidate operation on the cache line that contains the translation entry before updating the entry, to ensure that the write operation misses in the Data Cache. This workaround prevents the micro-architectural conditions for the erratum from happening. Interrupts must be temporarily disabled so that no interrupt can be taken between the maintenance operation and the translation entry update. This avoids the possibility of the interrupt service routine bringing the cache line back in the cache.

Another possible workaround is to place the translation table entries in Non-Cacheable memory areas, but this workaround is likely to have a noticeable performance penalty.

Note that inserting a DSB instruction immediately after writing the new translation table entry significantly reduces the probability of hitting the erratum, but is not a complete workaround.

2.3.14 (794073) Speculative instruction fetches with MMU disabled might not comply with architectural requirements

Status

Affects:	product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B
Fault Status:	Present in: All revisions. Open

Description

When the MMU is 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 read accesses 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.

A Cortex-A9 processor usually operates with both the MMU and branch prediction enabled. If the processor operates in this condition for any significant amount of time, the *Branch Target Address Cache* (BTAC) will contain branch predictions. If the MMU is then disabled, but branch prediction remains enabled, these stale BTAC entries can cause the processor to violate the rules for speculative fetches.

Configurations affected

This erratum affects all configurations of the processor.

Conditions

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

1. MMU and branch prediction are enabled.
2. Branches are executed.
3. MMU is disabled, and branch prediction remains enabled.

Implications

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

Workaround

The recommended workaround is to invalidate all entries in the BTAC, by executing an *Invalidate Entire Branch Prediction Array* (BPIALL) operation followed by a DSB, before disabling the MMU.

Another possible workaround is to disable branch prediction when disabling the MMU, and keep branch prediction disabled until the MMU is re-enabled.

2.3.15 (794074) A write request to Uncacheable, Shareable normal memory region might be executed twice, possibly causing a software synchronisation issue

Status

Affects: product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All revisions. Open

Description

Under certain timing circumstances specific to the Cortex-A9 microarchitecture, a write request to an Uncacheable, Shareable Normal memory region might be executed twice, causing the write request to be sent twice on the AXI bus. This might happen when the write request is followed by another write into the same naturally aligned doubleword memory region, without a DMB between the two writes.

The repetition of the write usually has no impact on the overall behaviour of the system, unless the repeated write is used for synchronisation purposes.

Configurations affected

The erratum affects all configurations of the processor.

Conditions

The erratum requires the following conditions:

1. A write request is performed to an Uncacheable, Shareable Normal memory region.
2. Another write request is performed into the same naturally doubleword aligned memory region. This second write request must not be performed to the exact same bytes as the first store.

A write request to Normal memory region is treated as Uncacheable in the following cases:

- The write request occurs while the Data Cache is disabled.
- The write request is targeting a memory region marked as Normal Memory Non-Cacheable or Cacheable Write-Through.
- The write request is targeting a memory region marked as Normal Memory Cacheable Write-Back and Shareable, and the CPU is in AMP mode.

Implications

This erratum might have implications in a multi-master system where control information is passed between several processing elements in memory using a communication variable, for example a semaphore. In such a system, it is common for communication variables to be claimed using a Load-Exclusive/Store-Exclusive, but for the communication variable to be cleared using a non-Exclusive store. This erratum means that the clearing of such a communication variable might occur twice. This might lead to two masters apparently claiming a communication variable, and therefore might cause data corruption to shared data.

A scenario in which this might happen is:

```
MOV r1,#0x40; address is double-word  aligned, mapped in
                  ; Normal Non-cacheable Shareable memory
Loop:LDREXr5, [r1,#0x0]; read the communication variable
CMP r5, #0  ; check if 0
STREXEQ r5, r0, [r1]; attempt to store new value
CMPEQ r5, #0; test if store succeeded
BNE Loop; retry if not
DMB        ; ensures that all subsequent accesses are observed when
```

```
; gaining of the communication variable has been observed
; loads and stores in the critical region can now be performed
MOV r2,#0
MOV r0, #0
DMB         ; ensure all previous accesses are observed before the
; communication variable is cleared
STR r0, [r1]; clear the communication variable with normal store
STR r2, [r1,#0x4]
; previous STR might merge and be sent again, which might
; cause undesired release of the communication variable.
```

This scenario is valid when the communication variable is a byte, a half-word, or a word

Workaround

There are several possible workarounds:

- Add a DMB after clearing a communication variable:

  ```
  STR r0, [r1]; clear the communication variable
          DMB         ; ensure the previous STR is complete
  ```

  Also any IRQ or FIQ handler must execute a DMB at the start to ensure as well the clear of any communication variable is complete.

- Ensure there is no other data using the same naturally aligned 64-bit memory location as the communication variable:

  ```
                  ALIGN 64
  communication_variable DCD 0
  unused_data DCD 0
  LDR r1,= communication_variable
  ```

- Use a Store-Exclusive to clear the communication variable, rather than a non-Exclusive store.

2.4 Category B (Rare)

This section describes the Category B rare errata.

2.4.1 (571618) PLI or PLD before interrupted CP15 operation might cause deadlock

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B (Rare)
Fault Status:	Present in: r0p0. Fixed in r0p1.

Description

If a PLI or PLD operation executes before an interruptible CP15 operation, and a CP15 interruption occurs before the PLI or PLD completes, then the interruption affects the PLI or PLD instead of the CP15 operation.

The following CP15 maintenance operations are interruptible:

- Invalidate entire instruction cache, inner shareable (c7,0,c1,0)
- Invalidate all instruction cache to PoU (c7,0,c5,0)
- Invalidate TLB entries on ASID match, inner shareable (c8,0,c3,2)
- Invalidate TLB entries by ASID (c8,0,c7,2).

Conditions

1. A PLI or PLD executes before an interruptible CP15 maintenance operation.
2. An interrupt occurs before the PLI or PLD operation completes, that is, while the PLI or PLD is waiting for the memory system to return the prefetched cache line.

Implications

A deadlock can occur on any CP15 operation following the interruptible CP15 operation.

Workaround

This erratum has the following workarounds:

1. Insert a memory barrier before the CP15 interruptible operations in case PLI or PLD are used in the code.

   This is the preferred workaround because it has only a small impact on performance.
2. Do not use PLI and PLD.
3. Set the appropriate bits in the undocumented RESERVED register CP15, 0, C15, C0, 1 to treat PLI and PLD as NOPs.
   - bit[24] disables the PLI
   - bit[20] disables the PLD.

Workaround #2 and #3 are equivalent in terms of functionality. There is unlikely to be a significant impact on performance in the PLI case. However, by not having functional PLD there might be a significant reduction in performance on optimized code routines such as a `memcopy()`.

2.4.2 (578565) PLI or PLD causing a two-level PTW and Uncacheable LDM might deadlock

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B (Rare)
Fault Status:	Present in: r0p0. Fixed in r0p1.

Description

The second access of a two-level uncacheable page table walk might conflict with an uncacheable LDM access, causing a data corruption or a deadlock.

Conditions

1. A PLI or PLD instruction executes, and causes an uncacheable two-level page table walk.
2. An LDM to an uncacheable memory region executes before the second uncacheable page table walk access to get the level2 descriptor completes.

An LDM to an uncacheable memory region is an LDM performed under one of the following conditions:

1. The access occurs while the Data Cache is Off.
2. The access targets a memory region marked as Strongly Ordered, Device, Normal Memory Non-Cacheable or Normal Memory Write-Through.
3. The access targets a memory region marked as Shareable Normal Memory Write-Back, and the CPU is in AMP mode.

The page table walk is uncacheable if it occurs under one of the following conditions:

1. The page table walk occurs while the Data Cache is Off.
2. The CP15 TTBR (Translation Table Base Register) being used for the page table walk has bit[6] and bit[0] set, specifying Non-Cacheable or Write-Through attributes.
3. The CP15 TTBR (Translation Table Base Register) being used for the page table walk has bit [1] set, specifying a Shared page table walk access, and the CPU is in AMP mode.

Implications

Under the conditions described, the uncacheable page table walk might conflict with the LDM accesses and cause a deadlock.

Workaround

The ideal workaround is to avoid meeting the conditions for the erratum to occur:

1. Make all page table walks cacheable. This is already a preferred configuration, for performance reasons.
2. If it is not possible to use the first workaround option, avoid using LDM to uncacheable memory regions.

If it is not possible to avoid these conditions, the following functionally equivalent workarounds are possible. However, these might cause a visible reduction in performance:

1. Do not use PLI and PLD in the executed code.
2. Set the appropriate bits in the undocumented RESERVED register CP15, 0, C15, C0, 1 treat PLI and PLD as NOPs. Bit 24 disables the PLI, bit 20 disables the PLD.

There is unlikely to be a significant reduction in performance with PLI disabled. However, disabling the PLD might reduce performance significantly on optimized code routines such as a memcopy().

2.4.3 (751473) Under very rare circumstances, automatic data prefetcher might cause deadlock or data corruption

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category B (Rare)
Fault Status:	Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

Under very rare timing circumstances, the automatic data prefetcher might cause address hazard issues, possibly leading to a data corruption or a processor deadlock.

Conditions

This erratum can occur only when the Data Cache and MMU are enabled in the following cases:

- on all memory regions marked as Write-Back Non-Shared, when the data prefetcher in L1 is enabled (ACTLR[2]=1'b1), regardless of the ACTLR.SMP bit
- on all memory regions marked as Write-Back Shared when the data prefetch hint in L2 is enabled (ACTLR[1]=1'b1) and when the processor is in SMP mode (ACTLR.SMP=1'b1).

Implications

When this erratum occurs, a data corruption or a processor deadlock can occur.

Workaround

The workaround for this erratum is to not enable the automatic data prefetcher by keeping ACTRL[2:1]=2'b00, which is the default value on exit from reset.

Although this feature might show significant performance gain on a few synthetic benchmarks, it usually has no impact in real systems. Therefore, this workaround is unlikely to cause any visible impact on final products.

2.4.4 (761320) Full cache line writes to the same memory region from at least two processors might deadlock the processor

Status

Affects: Product Cortex-A9 MPCore.

Fault Type: Programmer Category B (Rare)

Fault Status: Present in: All r0, r1, r2 and r3 revisions. Fixed in r4p0

Description

Under very rare circumstances, full cache line writes from at least two processors on cache lines in hazard with other cache accesses might cause arbitration issues in the SCU, leading to processor deadlock.

Configurations affected

This erratum affects the configurations of the processor with three or more active coherent agents, which is either:

- Two or more processors if the ACP is present.
- Three or more processors.

Conditions

To trigger the erratum, at least three agents need to be working in SMP mode, and accessing coherent memory regions.

Two or more processors need to perform full cache line writes, to cache lines which are in hazard with other access requests in the SCU. The hazard in the SCU happens when another processor, or the ACP, is performing a read of or a write to the same cache line.

The following example describes one scenario that might cause this deadlock:

- CPU0 performs a full cache line write to address A, then a full cache line write to address B
- CPU1 performs a full cache line write to address B, then a full cache line write to address A
- CPU2 performs read accesses to addresses A and B

Under certain rare timing circumstances, the requests might create a loop of dependencies, causing a processor deadlock.

Implications

When the erratum happens, it leads to system deadlock.

It is important to note that any scenario leading to this deadlock situation is uncommon. It requires two processors writing full cache lines to a coherent memory region, without taking any semaphore, with another processor or the ACP accessing the same lines at the same time, meaning that these latter accesses are not deterministic. This, combined with the extremely rare microarchitectural timing conditions under which the defect can happen, explains why the erratum is not expected to cause any significant malfunction in real systems.

Workaround

This erratum can be worked round by setting bit[21] of the undocumented Diagnostic Control Register to 1. This register is encoded as CP15 c15 0 c0 1.

The bit can be written in Secure state only, with the following Read/Modify/Write code sequence:

```
MRC p15,0,rt,c15,c0,1
```

```
ORR rt,rt,#0x200000
MCR p15,0,rt,c15,c0,1
```

When this bit is set, the "direct eviction" optimization in the Bus Interface Unit is disabled, which means this erratum cannot occur.

Setting this bit might prevent the Cortex-A9 from utilizing the full bandwidth when performing intensive full cache line writes, and therefore a slight performance drop might be visible.

In addition, this erratum cannot occur if at least one of the following bits in the Diagnostic Control Register is set to 1:

- bit [23] - Disable Read-Allocate mode
- bit [22] - Disable Write Allocate Wait mode

2.4.5 (845369) Under very rare timing circumstances, transitioning into streaming mode might create a data corruption

Status

Affects:	Product Cortex-A9 MPCore.
Fault Type:	Programmer Category B (Rare)
Fault Status:	Present in: All revisions. Open

Description

Under very rare timing circumstances, a data corruption might occur on a dirty cache line that is evicted from the L1 Data Cache due to another cache line being fully written.

Configurations affected

This erratum affects configurations with either:

- One processor if the ACP is present
- Two or more processors

Conditions

The erratum requires the following conditions:

- The CPU contains a dirty line in its data cache.
- The CPU performs at least four full cache line writes, one of which causes the eviction of the dirty line.
- Another CPU, or the ACP, performs a read or write operation on the dirty line.

The defect requires very rare timing conditions to reach the point of failure.

These timing conditions depend on the CPU micro-architecture and are not controllable in software:

- The CPU must be in a transitional mode that might be triggered by the detection of the first two full cache line writes.
- The evicted line must remain stalled in the eviction buffer, which is likely to be caused by congested write traffic.
- The other coherent agent, either another CPU in the cluster or the ACP, must perform its coherency request on the evicted line while it is in the eviction buffer.

Implications

The erratum might lead to data corruption.

Workaround

This erratum can be worked round by setting bit[22] of the undocumented Diagnostic Control Register to 1. This register is encoded as CP15 c15 0 c0 1.

The bit can be written in Secure state only, with the following Read/Modify/Write code sequence:

```
MRC p15,0,rt,c15,c0,1
ORR rt,rt,#0x00400000
MCR p15,0,rt,c15,c0,1
```

When this bit is set, the processor is unable to switch into Read-Allocate (streaming) mode, which means this erratum cannot occur.

Setting this bit could possibly result in a visible drop in performance for routines that perform intensive memory accesses, such as memset() or memcpy(). However, the workaround is not expected to create any significant performance degradation in most standard applications.

2.5 Category C

This section describes the Category C errata.

2.5.1 (548867) Debug entry can be faulty while in Jazelle state

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: r0p0. Fixed in r0p1.

Description

Under certain conditions, the debug entry when the processor is in Jazelle state might be ignored or might be faulty.

Conditions

1. Debug is enabled.
2. The core is in Jazelle state.
3. A debug entry event occurs.

If these conditions exist, debug state should be entered, together with some Jazelle context registers being saved, to correctly resume the program execution on exit from debug.

Because of this erratum, debug entry might not occur (for example, some breakpoints might not be caught), or the Jazelle stack state (R5 register) might not be saved properly. This prevents the Java bytecode execution resuming properly on exit from debug.

Implications

The debug of Jazelle applications is limited on the processor.

Workaround

There is no workaround for this erratum.

2.5.2 (548871) Spurious abort reported on an SWP following an aborted Strongly Ordered access

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: r0p0. Fixed in r0p1.

Description

A spurious abort can be reported on an SWP or SWPB instruction if the SWP/SWPB is performed to an uncached memory region, and if the SWP/SWPB is the next memory access following an aborted memory access to a Strongly Ordered memory region

Conditions

1. SWP/SWPB instruction support is enabled, that is, bit[10] in the System Control register is set.
2. An access occurs to a Strongly Ordered memory region.
3. This access causes an external abort.
4. The next memory access instruction is an SWP or an SWPB to an uncacheable memory region.

An access to an uncacheable memory region, as described above, is an access performed under one of the following conditions:

1. The access is performed while the Data Cache is Off.
2. The access targets a memory region marked as Strongly Ordered, Device, Normal Memory Non-Cacheable or Normal Memory Write-Through.
3. The access targets a memory region marked as Shareable, and the CPU is in AMP mode.

Under such conditions, a spurious abort is reported on the SWP/SWPB instruction.

Implications

The CPU has to recover from the spurious abort reported on the SWP, which also causes the FSR to update falsely.

Workaround

The workaround is to use an LDREX/STREX mechanism instead of the SWP/SWPB instructions.

Note

The SWP/SWPB instructions are deprecated, and are trapped to UNDEFINED by default in the Cortex-A9.

2.5.3 (571467) No exit from WFE when in NS state because AMO/IMO/IFO does not mask the CPSR A/I/F bits

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: r0p0. Fixed in r0p1.

Description

A processor in Non-Secure state might not wake up from WFE because of the A/I/F bit masking the Abort/IRQ/FIQ event regardless of the AMO/IMO/IFO values.

Implications

A deadlock can occur if the processor cannot wake up from WFE by another method.

Workaround

A hardware workaround for this erratum is to tie the external EVENTI pin high. This prevents entry to WFE state. Although this workaround does not create any functional issue, it might increase the overall power consumption in the system by disabling all WFE state power savings.

If the hardware workaround is not possible, the only software workaround is to not use WFE.

2.5.4 (571619) Imprecise abort reported on a PLD

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: r0p0. Fixed in r0p1.

Description

If a PLD accesses a memory region which aborts, the abort might be falsely reported to the core when it should not.

Implications

A spurious imprecise abort is reported to the core.

Workaround

If the software abort handler can cope with spurious imprecise abort, and can resume the program where it has been interrupted when the abort occurred, then no workaround is required.

If not, the workaround is to set bit[20] in the undocumented RESERVED register CP15, 0, C15, C0, 1 to treat PLD as a NOP. This might have a significant performance impact on optimized code such as a memcopy().

2.5.5 (571768) TLB invalidate by MVA all ASID should not invalidate locked entries

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

The TLB Invalidate by MVA all ASID operations should only invalidate unlocked entries in the TLB.

Because of the erratum, even the locked entries matching the MVA are invalidated in the TLB.

Implications

The implication of the erratum is that some entries in the lockdown region of the TLB might be evicted when they should not, causing further TLB miss, and page table walks, if the entry is accessed afterwards.

Workaround

The erratum might not create any functional issue, but only a performance issue on the invalidated TLB entry the next time it is accessed. This means that in some systems it is not necessary to apply a workaround.

If the entry must absolutely not be invalidated from the lockdown TLB region, the workaround is to replace the Invalidate TLB by MVA all ASID operation by Invalidate TLB by MVA and by ASID operations, for all the ASID using this MVA entry (but not for the MVA and ASID in the TLB lockdown entry).

2.5.6 (571773)SWP might interfere with a Cacheable Page Table Walk request, causing deadlock

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: r0p0. Fixed in r0p1.

Description

Under rare conditions, a SWP instruction that executes while a Cacheable Page Table Walk request is pending on the AXI bus might conflict, and cause a deadlock.

Conditions

1. SWP/SWPB instruction support is enabled, that is, bit[10] in the System Control register is set.
2. A request is done, which misses in the µTLB and main TLB, and consequently requires a page table walk to occur. This page table walk is cacheable, and misses in the Data Cache, causing an external line-fill request on the AXI bus.
3. Before this external PTW line-fill request completes, an SWP instruction is being executed.

Note

The request that causes the page table walk can have different possible sources, either from the Instruction side, from Data side (Load or Store), from CP15 operations, or from CP15 operations broadcast from another CPU.

Under very rare circumstances, the SWP and the page table walk might conflict, causing a deadlock of the Cortex-A9.

Implications

If all conditions exist for the erratum to occur, this may cause a system deadlock.

Workaround

The workaround is to use LDREX/STREX instead of SWP.

Note

SWP are deprecated, and are trapped to UNDEFINED by default in the Cortex-A9.

2.5.7 (605225) Possible double allocation in the cache when a PLD and a SWP are performed on the same address

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r0 revisions. Fixed in r1p0.

Description

Under very rare circumstances, a bad hazard detection involving two or more PLDs and a SWP or SWPB at the same address as the second PLD, might cause a double allocation in the cache, possibly resulting in data corruption.

Conditions

1. SWP/SWPB are enabled
2. At least 2 PLDs are being issued.
3. A SWP or SWPB is executed at the same address of the last PLD.

Under very rare circumstances, the SWP or SWPB might interfere with the PLD and cause a double allocation in the cache. The result of this is a double hit on future cache requests on this line.

Implications

The erratum can cause data corruption.

Workaround

There are two possible workarounds:

1. Do not use SPW/SWPB. Note that SWP/SWPB are deprecated, and are trapped to UNDEFINED by default in the Cortex-A9. Use LDREX/STREX instead.
2. Do not use SWP/SWPB and PLD targeting the same address.

2.5.8 (605226) Domain fault on a cache maintenance operation by MVA might cause the operation to be performed on the wrong cache line

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

The ARM architecture specifies that in the case of a cache maintenance operation by MVA which causes a permission fault or a domain fault, the cache maintenance operation must still be performed, because the VA translation is correct.

Because of the erratum, in the case of a cache maintenance operation by MVA causing a domain fault, the VA translation is not performed correctly, and the cache maintenance operation is performed on the wrong cache line.

Note

The erratum only occurs with domain faults. There is no issue with access fault or permission fault.

Implications

The erratum possibly affects two different cache lines:

1. The cache line that the cache maintenance operation targets remains untouched.
2. Another cache line might be erroneously invalidated.

In the best case, the erratum might have no noticeable effect.

In the worst case, data corruption might occur.

Workaround

Domain faults are a deprecated feature. Do not use them.

If you cannot avoid using domain permissions, then there is no easy software workaround for this erratum.

2.5.9 (605230) DBGACK and DBGDONE signals might remain asserted after a CPU reset

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r0 revisions. Fixed in r1p0.

Description

The processor has two different reset inputs to be able to reset separately the core functional logic and the debug logic.

The DBGACK and DBGDONE debug signals are wrongly reset by the debug reset, instead of by the core functional reset.

If the CPU is in debug state, the DBGACK and DBGDONE outputs are asserted. If the core functional reset is asserted, the core is expected to exit from debug, and the DBGACK and DBGDONE outputs should go low.

The core exits from debug as expected, but because of the erratum, the DBGACK and DBGDONE outputs might remain asserted.

Conditions

1. The CPU is in debug state, with the DBGACK and DBGDONE outputs being asserted.
2. A core functional reset is applied, and the core debug reset is not.

Implications

The system identifies that DBGACK and DBGDONE are asserted and might act as if the CPU is still in debug state, when it is not.

Workaround

Do not assert the functional core reset without the debug reset while the CPU is in debug state.

2.5.10 (643718) Priority given to breakpoints over MMU fault

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

When an instruction has a breakpoint and an MMU fault programmed on it, the MMU fault should be taken after the instruction fetch completes.

Because of the erratum, the breakpoint might be taken instead of the MMU fault.

Conditions

1. A breakpoint is set at a given address.
2. The MMU is enabled, and programmed so that a MMU fault is generated at this given address The instruction at this address is fetched and executed.

Implications

Because of the erratum, the breakpoint is taken, and debug state is entered, before the MMU fault code resolution executes.

There is no other expected implication for the erratum.

Workaround

Workaround for the debugger:

The debugger can observe this problem because, on entering debug state, reading the address at the PC returns an abort.

To emulate the correct architecture behavior, the debugger can:

1. unset the breakpoint at the instruction (A)
2. set vector catch on the Prefetch Abort vector
3. set the PC=A and restart.

The target fetches the instruction at A, generating a Prefetch Abort and halt at the Prefetch Abort vector. The debugger can then:

1. unset vector catch
2. reset the breakpoint at A
3. set PC=PAbort_vector and restart.

The OS handles the prefetch abort, and then returns to A, re-triggering the breakpoint.

2.5.11 (643719) LoUIS bit field in CLIDR register is incorrect

Status

Affects:	Product Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

CLIDR[23:21] is the LoUIS bit field: Level of Unification Inner Shareable.

This field should be 3'b001 in the Cortex-A9 MPCore version, indicating that the L1 caches of the Cortex-A9 MPCore are before the point of unification for the Inner Shareable shareability domain, and as such, they need to be cleaned or invalidated when cleaning or invalidation to the point of unification for the Inner Shareable shareability domain.

However, this value is 3'b000 in the Cortex-A9 MPCore, falsely indicating that the L1 caches of the Cortex-A9 MPCore do not need to be cleaned or invalidated in this case.

Implications

Any software code relying on this LoUIS value to determine which levels in the memory hierarchy need to be cleaned and invalidated in case of clean and invalidate operation to the point of unification for the Inner Shareable shareability domain will conclude that the L1 caches do not require the operation.

If the clean and invalidate operation is not performed, this can lead to data corruption.

Workaround

Instead of directly reading the LoUIS value in the CLIDR register, the software needs to read the CPU ID, and consider that LoUIS is 3'b001 if the CPU ID corresponds to a Cortex-A9 MPCore processor.

2.5.12 (643769) Missing reset on DBGCLAIMCLR register

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

On exit from reset, the DBGCLAIMCLR register should be read as 0. This might not be the case, because this register is not reset in the design.

Implications

Any debug tool that relies on this register to be 0 on exit from reset might fail.

Workaround

If the DBGCLAIM functionality is required, first write to DBGCLAIMCLR once in order to reset the DBGCLAIM register before using it.

2.5.13 (643921) Writing 0 to DRCR[1] and DRCR[0] should be ignored

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

Writing 0 to bit [0] and bit[1] of the DRCR registers should be ignored. However, because of the erratum, the write is performed.

Implications

The debug entry (bit[0]) and debug exit (bit[1]) pending request flags can be cleared because of the write being executed, when it should not be cleared.

This erratum is not likely to create any issue in a real implementation because such writes should never occur.

Workaround

There is no workaround for this erratum.

2.5.14 (643922) Software lock writes are not ignored when performed from an external debugger

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 revisions. Fixed in r1p0.

Description

Accesses from an external debugger (that is, with PADDRDBG[31]==1) to the DBGLAR and DBGLSR register should be ignored (write ignored, read as zero).

Because of the erratum, the DBGLSR register can be updated because the write to the DBGLAR register is not ignored, even when it comes from an external debugger.

Implications

The DBGLSR register is updated when it should not be updated.

Please note that even though it is updated, the DBGLSR register is still read as zero, as expected, when the read is performed by an external debugger.

Workaround

There is no workaround for this erratum.

2.5.15 (693170) In debug state, IRQ/FIQ can interrupt some CP15 maintenance operation

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r0 and r1 revisions Fixed in r2p0.

Description

The ARM architecture states that *IRQ and FIQ exceptions are disabled and not taken in Debug state*. Because of the erratum, if an interrupt (IRQ or FIQ) occurs while the processor is in Debug state, and the interrupt is not disabled explicitly by setting the DBGDSCR.INTdis, then any interruptible CP15 maintenance operation can be interrupted instead of completing.

Conditions

1. The processor is in Debug state (Debug Halt mode).
2. DBGDSCR.INTdis bit (bit [11]) is zero.
3. The processor executes an interruptible CP15 operation though the ITR. The IRQ or the FIQ pin is asserted, and the corresponding I or F bit in the CPSR is not set so that the interrupt can be taken

Under these conditions, the CP15 operation should complete because the IRQ or FIQ should be ignored while the processor is in Debug state. However, because of the erratum, the CP15 operation can be interrupted by the IRQ or FIQ, and, if it is interrupted, the operation does not complete.

The erratum affects four interruptible CP15 operations:

1. Instruction Cache Invalidate all Normal.
2. Instruction Cache Invalidate all Inner Shareable.
3. Invalidate TLB entry on ASID match Normal.
4. Invalidate TLB entry on ASID match Inner Shareable.

Implications

Because the corresponding maintenance operation might not complete as it should do, the code sequence executed by the debugger through the ITR might not run reliably.

Workaround

To ensure the IRQ and FIQ are fully ignored while the processor is in Debug state, the debugger has to explicitly set bit[11] in the DBGDSCR (Interrupt Disable bit) on debug entry whenever it uses any of the four interruptible CP15 operations mentioned above. In these cases it must restore this bit to its previous value before exiting debug.

2.5.16 (693171) DBGDSCR internal and external views are inverted

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r0 and r1 revisions Fixed in r2p0.

Description

The decoding of the internal and external views of the DBGDSCR register is inverted. Because of this inversion, the side-effects affecting the DBGSCR registers do not comply with the ARM architecture, and can cause corruption of the TXfull_l and RXfull_l values.

Conditions

1. The processor is in Debug state.
2. The DCC (Debug Communications Channel) mechanism is used

Implications

Because of the erratum:

- The TXfull_l value in the DBGDSCR might not be the last value of TXfull read from DBGDSCRext.
- The RXfull_l value in the DBGDSCR might not be the last value of RXfull read from DBGDSCRext.

The point of these latched versions of the signals is to ensure that the value of the flags last seen by the debugger is the one used by the hardware to prevent overwrite. For the DCC this allows a debugger to send a sequence of commands to an ICE and pick apart the results later; for example, if it sends “read DSCR; read DTRTX” as a pair of commands and later finds DSCR.TXfull is 0, it knows that even if the software writes to DTRTX between processing those two APB commands, the processor ignores the read of DTRTX so that value can be discarded.

Because the values of TXfull_l and RXfull_l are not fully reliable, the DCC mechanism can appear lossy, with no way for the debugger to detect this loss.

Workaround

Software executing in privileged modes can avoid the problem by using the DSCRext encoding of the CP14 instruction instead of the DSCRint encoding in such checks. Such software would obviously exhibit the same problem on a processor which does not have this erratum, so the workaround is totally specific to the versions of the processor being affected by this erratum.

Software executing in User mode cannot use the same workaround because the instruction would be decoded as UNDEFINED, in accordance with the ARM architecture. No proper workaround is possible in this case.

2.5.17 (693172) Data abort does not interrupt Non-Blocking execution from ITR

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r0 and r1 revisions Fixed in r2p0.

Description

When in Debug state, the processor can execute instructions which are fed by the debugger through the ITR register. There are three modes of operation in this case: Stall mode, Non-blocking mode, and Fast mode.

If an instruction executed by the ITR generates a synchronous Data Abort, the SDABORT_l flag (bit[6]) is set in the DBGDSCR register. When this SDABORT_l flag is set, the following instruction executed by the ITR should behave differently depending on the mode:

1. In Non-Blocking and Stall modes, the instruction should not execute. The DBGDSCR.InstrCompl_l flag (bit[24]), indicating whether the instruction execution is complete, should not be set.
2. In Fast mode, the SDABORT_l flag should be ignored. The following instruction should complete as normal, and the DBGDSCR.InstrCompl_l flag should be set after the instruction execution completes.

Because of the erratum, the behaviors described above are inverted, so that:

1. In Non-Blocking and Stall modes, the instruction completes as normal even when the DBGDSCR.SDABORT_l flag is set. Consequently, the DBGDSCR.InstrCompl_l flag is set after the instruction execution completes.
2. In Fast mode, the DBGDSCR.SDABORT_l is wrongly taken into account, and the instruction does not execute. The DBGDSCR.InstrCompl_l flag remains at zero.

Conditions

1. The processor is in debug state.
2. The processor executes instructions from the ITR register.
3. One instruction executed from the ITR register generates a Synchronous Data Abort. This causes the SDABORT_l flag, bit[24], to be set in the DBGDSCR register.
4. A second instruction executes after the one which caused the Synchronous Data Abort.

Implications

Because of the erratum, the debugger might behave incorrectly if it executes a code sequence involving an instruction which generates a Synchronous Data Abort.

Workaround

After executing an instruction which can possibly generate a Synchronous Data Abort, the debugger has to explicitly read the DBGDSCR register to check whether a Data Abort occurred, so it can execute (or not execute) the following instruction, depending on which mode the ITR is running in.

2.5.18 (719331) Data prefetcher can cause a processor deadlock when executing a WFI

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: r0p0 and all r1 revisions. Fixed in r2p0.

Description

A WFI execution causes the processor to wait while it achieves a stable state before cutting the clock. The processor waits for all pending AXI transactions to complete before cutting the clock.

Because of this erratum, an automatic prefetch request can be issued in the same cycle as when the clock is cut. That is, an external linefill request is issued, which cannot receive its data back because of the clock being cut.

Conditions

1. The L1 Data Prefetcher is enabled (bit[2] in the CP15 Auxiliary Control register).
2. The processor performs some load or store requests, or PLD, or Page Table Walk, which trigger the Data Prefetcher to non-coherent memory regions (Shared=0).
3. The processor executes a WFI instruction.

Note

The L2 Prefetch Hint Enable, bit[1] in the CP15 Auxiliary Control register in revisions, which exists only since the r1p0 revision, can still be enabled without causing any issue.

Implications

The ARM Architecture recommends that a DSB executes before a WFI to ensure that all memory accesses that are executed before the WFI are visible to other observers before the processor is suspended. This erratum is avoidable by following this recommendation.

If the erratum occurs, the processor issues a linefill request for which it cannot receive any data back, leading to a potential system corruption or deadlock.

Workaround

1. The workaround for this erratum is to prevent a prefetch request from being issued when the processor wants to enter WFI, which you can achieve by inserting a DSB before the WFI instruction, as recommended in the ARM Architecture.
2. An alternative solution is to disable the L1 Data Prefetcher before executing the WFI instruction, by setting ACTLR.DP==1 (bit[2] in the Auxiliary Control register).
3. If it is not possible to modify the code before the WFI, then the automatic L1 Data Prefetcher must remain disabled in the Cortex-A9.

   This is the case on exit from reset.

2.5.19 (719332) Uncacheable SWP or SWPB instruction can be corrupted by the Data Prefetcher

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: r0p0 and all r1 revisions. Fixed in r2p0.

Description

An uncacheable SWP or SWPB instruction can be corrupted because of a faulty interaction with the Data Prefetcher. This causes a faulty SWP or SWPB transaction to be issued on the external AXI bus.

Conditions

1. SWP and SWPB instructions need to be enabled, that is, SCTLR.SW bit (bit[10] in the System Control register) needs to be set. By default, this is not the case, because SCTLR.SW=0 on exit from reset.
2. The automatic Data Prefetcher must be enabled:
   a. In r0p0, this means that ACTLR.DP==1 (bit[2] in the CP15 Auxiliary Control register)
   b. In r0 revisions, the data prefetcher is disabled in hardware, which means the erratum is not present in these revisions
   c. In r1p0 and r1p1, this means that either ACTLR.L1PrefetchEnable==1 (bit[2]) or ACTLR.L2PrefetchHintEnable==1 (bit[1]).
3. An uncacheable SWP or SWPB instruction executes.

The SWP or SWPB is uncacheable in the following cases:

1. The SWP or SWPB occurs while the Data Cache is Off.
2. The SWP or SWPB targets a memory region marked as Strongly Ordered, Device, Normal Memory Non-Cacheable or Normal Memory Write-Through.
3. The SWP or SWPB targets a memory region marked as Shareable Normal Memory Write-Back, and the CPU is in AMP mode.

Under some additional timing conditions specific to the Cortex-A9 micro-architecture, not controllable by software, the SWP or SWPB can wrongly interfere with a Data prefetch request, causing a faulty access on the AXI bus.

Implications

Because of the erratum, the SWP or SWPB transaction is corrupted, which causes a faulty request on the AXI bus leading to different possible malfunctions in the system.

Workaround

1. SWP and SWPB instructions are deprecated in the ARMv7 architecture. Use exclusive accesses (LDREX/STREX) instead.
2. If you cannot avoid using SWP/SWPB, for example in the case of legacy code, then do not enable the automatic Data Prefetcher in the Cortex-A9. Note that the Data Prefetcher is disabled on exit from reset.
   - In r0p0, this is controlled by ACTLR.DP, bit[2], which must remain at 0.
   - In r0p1 and r0p2, the prefetcher is disabled in hardware, so the erratum is not present.
   - In r1px, both Prefetch Enable bits (ACTLR.L1PrefetchEnable and ACTLR.L2PrefetchHintEnable) must remain at 0.

2.5.20 (721147) TLBIMVAA and TLBIMVAAIS operations might not invalidate all required TLB entries when using multiple page sizes

Status

Affects:	Product Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0 and r1 revisions Fixed in r2p0.

Description

A TLBIMVAA operation should invalidate all TLB entries with a matching MVA, regardless of the page size. Because of the erratum, the operation executes only for entries on one single page size. So, in the case when the same MVA is present several times in the main TLB for different page sizes, some entries are not invalidated although they should be, potentially leading to a corruption in the page table entries.

In the MPCore version of the product, the broadcast version of the instruction, TLBIMVAAIS, is also affected by the erratum. In this case, each CPU invalidates the TLB entry in its own main TLB which matches the last page size it used, and does not invalidate other TLB entries.

Conditions

1. MMU is on.
2. The main TLB contains at least:
   a. One entry for a given MVA with a given page size
   b. Another entry for the same MVA with a different page size.
3. A TLBIMVAA or TLBIMVAAIS operation executes.

Under such conditions, only one of the two entries in the TLB is invalidated, because the operation is performed on one single page size. Which page size is affected by the operation solely depends on the internal state of the processor, and is not controllable in software.

In the case of an MPCore system, with the TLBIMVAAIS operation, each CPU in the cluster is affected by the erratum if it also contains multiple entries for the same MVA with different page sizes.

Implications

Some entries in the main TLB which should be invalidated are not, causing a corruption in the page table entries.

Workaround

The workaround for this erratum is to replace the TLBIMVAA maintenance operation with the TLBIALL operation, and to replace the TLBIMVAAIS operation with the TLBIALLIS operation, invalidating all entries in the main TLB instead of invalidating all entries with a matching MVA.

This workaround has a performance impact, because it invalidates more TLB entries than required. However, if the page table descriptors are marked as Inner cacheable, then the performance impact is likely to be insignificant, because the L1 data cache then backs up the main TLB.

2.5.21 (721958) TLBIMVAA operations might not invalidate all required TLB entries when multiple page sizes are used

Status

Affects: product Cortex-A9

Fault Type: Programmer Category Cat C

Fault Status: Present in: all r0 and r1 revisions. Fixed in r2p0.

Description

A TLBIMVAA operation should invalidate all TLB entries with a matching MVA, regardless of the page size. Because of the erratum, the operation is only executed for entries on one single page size. So, if the same MVA is present several times in the main TLB for different page sizes, some entries are not invalidated although they should be, potentially leading to a corruption in the page table entries.

Conditions

1. MMU is on.
2. The main TLB contains at least:
    a. One entry for a given MVA with a given page size
    b. Another entry for the same MVA with a different page size
3. A TLBIMVAA operation is executed

Under such conditions, only one of the two entries in the TLB is invalidated, because the operation is performed on one single page size. Which page size is affected by the operation solely depends on the internal state of the processor, and is not controllable in software.

Implications

Some entries in the main TLB which should be invalidated are not, leading to a corruption in the page table entries.

Workaround

The workaround for this erratum is to replace the TLBIMVAA maintenance operation with the TLBIALL operation, invalidating all entries in the main TLB instead of invalidating all entries with a matching MVA.

This workaround has a performance impact, because it invalidates more TLB entries than necessary. However, if the page table descriptors are marked as Inner cacheable, then the performance impact is likely to be insignificant, because the L1 data cache then backs up the main TLB.

2.5.22 (725631) ISB is counted in Performance Monitor events 0x0C and 0x0D

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions Open

Description

The ISB is implemented as a branch in the Cortex-A9 micro-architecture.

This implies that events `0x0C` (software change of PC) and `0x0D` (immediate branch) are asserted when an ISB occurs, which is not compliant with the ARM Architecture.

Implications

The count of events `0x0C` and `0x0D` are not completely precise when using the Performance Monitor counters, because the ISB is counted together with the real software changes to PC (for `0x0C`) and immediate branches (`0x0D`).

The erratum also causes the corresponding PMUEVENT bits to toggle in case an ISB executes.

- PMUEVENT[13] relates to event `0x0C`
- PMUEVENT[14] relates to event `0x0D`.

Workaround

You can count ISB instructions alone with event `0x90`.

You can subtract this ISB count from the results you obtained in events `0x0C` and `0x0D`, to obtain the precise count of software change of PC (`0x0C`) and immediate branches (`0x0D`).

2.5.23 (729817) MainID register alias addresses do not map to the Debug interface

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions Open

Description

The ARM Debug Architecture specifies registers 838 and 839 as *Alias of the MainID register*. They should be accessible using the APB Debug interface at addresses `0xD18` and `0xD1C`.

The two alias addresses are not implemented in Cortex-A9. A read access at either of these two addresses returns 0, instead of the MIDR value.

Note

Read accesses to these two registers using the internal CP14 interface are trapped to UNDEFINED, which is compliant with the ARM Debug architecture. Therefore the erratum only applies to the alias addresses using the external Debug APB interface.

Implications

If the debugger, or any other external agent, tries to read the MIDR register using the alias addresses, it will get a faulty answer (`0x0`), which can cause indeterminate errors in the debugger afterwards.

Workaround

The workaround for this erratum is to always access the MIDR at its original address, `0xD00`, and not to use its alias address.

2.5.24 (729818) In debug state, next instruction is stalled when SDABORT flag is set, instead of being discarded

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions Open.

Description

When the processor is in debug state, an instruction written to the ITR after a Load/Store instruction that aborts gets executed on clearing the SDABORT_l, instead of being discarded.

Conditions

- Debugger has put the extDCCmode bits into Stall mode
- A previously issued load/store instruction has generated a synchronous Data Abort (for example, an MMU fault)
- For efficiency, the debugger does not read DBGDSCRext immediately, to see if the load/store has completed and has not aborted, but writes further instructions to the ITR, expecting them to be discarded if a problem occurs
- The debugger reads the DBGDSCR at the end of the sequence and discovers the load/store aborted
- The debugger clears the SDABORT_l flag (by writing to the Clear Sticky Aborts bit in DBGDRCR).

Under these conditions, the instruction that follows in the ITR might execute instead of being discarded.

Implications

Indeterminate failures can occur because of the instruction being executed when it should not. In most cases, it is unlikely that the failure will cause any significant issue.

Workaround

There are a selection of workarounds with increasing complexity and decreasing impact. In each case the impact is a loss of performance when debugging:

1. Do not use stall mode.
2. Do not use stall mode when doing load/store operations.
3. Always check for a sticky abort after issuing a load/store operation in stall mode (the cost of this probably means workaround number 2 is a preferred alternative).
4. Always check for a sticky abort after issuing a load/store operation in stall mode before issuing any further instructions that might corrupt an important target state (such as further load/store instructions, instructions that write to “live” registers such as VFP, CP15).

2.5.25 (740663) Event 0x68 / PMUEVENT[9:8] might be inaccurate

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

Event `0x68` counts the total number of instructions passing through the Register Rename pipeline stage.

Under certain conditions, some branch-related instructions might pass through this pipeline stage without being counted. As a consequence, event `0x68` might be inaccurate, with a lower value than expected.

The event is also reported externally on PMUEVENT[9:8], which suffers from the same inaccuracy.

Conditions

- Events are enabled.
- One of the PMU counters is programmed to count event 0x68 - number of instructions passing through the Register Rename stage. Alternatively, an external component counts, or relies on, PMUEVENT[9:8].
- A program executes, which contains one of the following instruction:
  - A Branch immediate, without Link
  - An ISB instruction
  - An HB instruction, without Link and without parameter, in ThumbEE state
  - An ENTERX or LEAVEX instruction, in Thumb or ThumbEE state.
- The program executed causes some stalls in the processor pipeline.

Under certain timing conditions, specific to the Cortex-A9 micro-architecture, a cycle stall in the processor pipeline might hide the instructions mentioned above, resulting in a corrupted count for event `0x68`, or a corrupted value on PMUEVENT[9:8] during this given cycle. If the hidden instruction appears in a loop, the count difference can be significant.

As an example, consider the following loop:

```
loop     subs            r0, #1

         vadd.f32 q1, q1, q0

         bne             loop
```

The loop contains three instructions, so the final instruction count should (approximately) be three times the number of executed loops. In practice, the `vadd` causes a pipeline stall after a few loop iterations. This stall hides the branch instruction (`bne loop`), so that only two instructions are counted per loop, and the final count appears as twice the number of executed loops instead of three times this number.

Implications

The implication of this erratum is that the values of event `0x68` and PMUEVENT[9:8] are imprecise, and cannot be relied upon.

Workaround

No workaround is possible to achieve the required functionality of counting precisely how many instructions are passing through the Register Rename pipeline stage.

2.5.26 (743623) Bad interaction between a minimum of seven PLDs and one Non-Cacheable LDM, can lead to deadlock

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore

Fault Type: Programmer Category C

Fault Status: Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

Under very rare circumstances, a deadlock can occur in the processor when handles a minimum of seven PLD instructions, shortly followed by one LDM to an uncacheable memory location.

The LDM is treated as Uncacheable in the following cases:

1. The LDM occurs while the Data Cache is Off.
2. The LDM targets a memory region marked as Strongly Ordered, Device, Normal Memory Non-Cacheable, or Normal Memory Write-Through
3. The LDM targets a memory region marked as Shareable Normal Memory Write-Back, and the CPU is in AMP mode.

Conditions

The code sequence that exhibits this erratum requires at least seven PLDs shortly followed by one LDM to an uncacheable memory region. The erratum occurs when the LDM appears on the AXI bus before any of the seven PLDs, which can possibly happen when the first PLD misses in the µTLB. In this case it needs to perform a TLB request, which may not be serviced immediately because the main TLB is already performing a Page Table Walk for another resource (for example, Instruction side), or because the PLD request to the main TLB is missing and causing a Page Table Walk.

Also note that the conditions above are not sufficient to recreate the failure, because additional rare conditions on the internal state of the processor are necessary to trigger the errata.

Implications

The erratum might create a processor deadlock. However, the conditions which are required for this to occur are extremely unlikely to occur in real code sequences.

Workaround

The primary workaround is to avoid using the offending code sequence, that is, do not use uncacheable LDM at the same time as an intensive use of PLD instructions.

If not possible, another workaround for this erratum is to set bit[20] in the undocumented Control register, which is placed in CP15 c15 0 c0 1.

This bit needs to be written with the following Read/Modify/Write code sequence:

```
MRC     p15,0,r0,c15,c0,1

ORR     r0,r0,#0x00100000

MCR     p15,0,r0,c15,c0,1
```

Setting this bit causes all PLD instructions to be treated as NOPs, with the consequence that code sequences usually using the PLDs, such as the `memcpy()` routine, might suffer from a visible performance drop.

2.5.27 (743625) A coherent ACP request might interfere with a non-cacheable SWP/SWPB from the processor, potentially causing deadlock

Status

Affects:	Product Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

Under very rare circumstances, a faulty address hazard checking in the SCU might cause a coherent ACP request to badly interfere with a Non-Cacheable SWP/SWPB request performed by the CPU.

Conditions

This erratum can only occur in a Cortex-A9 MPCore product containing one single processor, with the ACP present. The processor must work in coherent mode, with ACTLR.SMP=1, and with its data cache enabled.

The processor also needs to have enabled SWP instructions, by setting ACTLR.SWP=1. SWP and SWPB instructions are strongly deprecated by ARM, and in the Cortex-A9 these instructions are disabled by default.

To reproduce the failure, the following events are required:

1. The processor performs a write in a cache line A, marked as coherent (Write-Back Shared).
2. This line exits the processor, either because the processor has switched into Read-Allocate mode, so that the write is immediately performed externally, or because the cache line is naturally evicted, replaced by another line, or cleaned.
3. The processor then performs another write in this cache line A, which brings the data back in dirty state in its data cache.
4. Finally, the processor executes a SWP or SWPB instruction to a Non-Cacheable region (Strongly Ordered, Device, or Normal Memory Non-Cacheable).

The offending sequence also requires the ACP to perform a coherent request in cache line A. In r0p0, r0p1 and r0p2, the ACP request is coherent if AxUSER[0]=1. Since r1p0 revision, the ACP request is coherent if AxUSER[0]=1 and AxCACHE[1]=1.

Under these conditions, and if the CPU has not performed any other write since cache line A was written externally, then a faulty address hazard might be detected in the SCU, which prevents all current transactions completing.

Implications

Under certain timing conditions, specific to the Cortex-A9 MPCore micro-architecture, the erratum can lead to processor deadlock.

Workaround

The recommended workaround is to use LDREX/STREX instead of SWP/SWPB instructions, which are deprecated in the ARMv7 architecture.

In case SWP and SWPB cannot be suppressed from the code, an alternative workaround is to perform a dummy write to a Non-Shared, Non-Cacheable memory location before executing the SWP or SWPB, which would clear the faulty hazard checking.

2.5.28 (743626) An imprecise external abort received while the processor enters WFI may cause a processor deadlock

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All r0, r1 and r2 revisions Fixed in r3p0

Description

An imprecise external abort received while the processor is ready to enter into WFI state might cause a processor deadlock.

Explicit memory transactions can be completed by inserting a DSB before the WFI instruction, but this does not prevent memory accesses generated by:

- Previously issued PLD instructions
- Page table walks associated with previously issued PLD instructions or as a result of the PLE engine.

If an external abort returns as a result of one of these memory accesses after executing a WFI instruction, the processor can deadlock.

Implications

In case the non-explicit memory request receives an external imprecise abort response while the processor is ready to enter into WFI state, the processor might deadlock.

In practical systems, it is not likely that these memory transactions will generate an external abort, because external aborts are usually a sign of significant corruption in the system.

Workaround

The workaround for this erratum is to protect all memory regions which can return an imprecise external abort with the correct MMU settings, to prevent any external aborts.

2.5.29 (751471) DBGPCSR format is incorrect

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions Open

Description

With respect to the DBGPCSR register, the ARM Architecture specifies that:

- DBGPCSR[31:2] contain the sampled value of bits [31:2] of the PC.
- The sampled value is an instruction address plus an offset that depends on the processor instruction set state.
- DBGPCSR[1:0] contain the meaning of PC Sample Value, with the following permitted values:
  - `0b00` ((DBGPCSR[31:2] << 2) – 8) references an ARM state instruction
  - `0bx1` ((DBGPCSR[31:1] << 1) – 4) references a Thumb or ThumbEE state instruction
  - `0b10` IMPLEMENTATION DEFINED.

  This field encodes the processor instruction set state, so that the profiling tool can calculate the true instruction address by subtracting the appropriate offset from the value sampled in bits [31:2] of the register.

In Cortex-A9, the DBGPCSR samples the target address of executed branches (but possibly still speculative to data aborts), with the following encodings:

- DBGPCSR[31:2] contain the address of the target branch instruction, with no offset.
- DBGPCSR[1:0] contains the execution state of the target branch instruction:
  - `0b00` for an ARM state instruction
  - `0b01` for a Thumb state instruction
  - `0b10` for a Jazelle state instruction
  - `0b11` for a ThumbEE state instruction

Implications

The implication of this erratum is that the debugger tools must not rely on the architected description for the value of DBGPCSR[1:0], nor remove any offset from DBGPCSR[31:2], to obtain the expected PC value.

Subtracting 4 or 8 from the DBGPCSR[31:2] value would lead to an area of code which is unlikely to have been recently executed, or which might not contain any executable code.

The same might be true for Thumb instructions at half-word boundaries, in which case PC[1]=1 but DBGPCSR[1]=0, or ThumbEE instructions at word boundaries, with PC[1]=0 and DBGPCSR[1]=1.

In Cortex-A9, because the DBGPCSR is always a branch target (= start of a basic block to the tool), the debugger should be able to spot many of these cases and attribute the sample to the right basic block.

Workaround

The debugger tools can find the expected PC value and instruction state by reading the DBGPCSR register, and consider it as described in the Description section.

2.5.30 (751480) Conditional failed LDREXcc can set the exclusive monitor

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.		
Fault Type:	Programmer Category C		
Fault Status:	Present in:	All r0, r1 and r2 revisions	Fixed in r3p0

Description

A conditional LDREX might set the internal exclusive monitor of the Cortex-A9 even when its condition fails.

Implications

The implication of the erratum is that a subsequent STREX might succeed when it should not. Therefore, the memory region protected by the exclusive mechanism can be corrupted if another agent accesses it at the same time.

Workaround

The workaround for this erratum is to not use conditional LDREX together with non-conditional STREX.

- The erratum cannot trigger unless using conditional LDREX.
- If using conditional LDREX, the associated STREX should also be conditional using the same condition. This means that, even if the exclusive monitor is set by the condition failed LDREX, the following STREX will not execute because it will be condition failed too. For most situations this will naturally be the case anyway.

2.5.31 (756421) Sticky Pipeline Advance bit cannot be cleared from debug APB accesses

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

The Sticky Pipeline Advance bit is bit[25] of the DBGDSCR register. This bit enables the debugger to detect whether the processor is idle. This bit is set to 1 every time the processor pipeline retires one instruction.

A write to DBGDRCR[3] clears this bit.

The erratum is that the Cortex-A9 does not implement any debug APB access to DBGDRCR[3].

Implications

Because of the erratum, the external debugger cannot clear the Sticky Pipeline Advance bit in the DBGDSCR. In practice, this makes the Sticky Pipeline Advance bit concept unusable on Cortex-A9 processors.

Workaround

There is no practical workaround for this erratum.

The only possible way to reset the Sticky Pipeline Advance bit is to assert the nDBGRESET input pin on the processor, which obviously has the side effect to reset all debug resources in the concerned processor, and any other additional Coresight components nDBGRESET connects to.

2.5.32 (757119) Some *Unallocated memory hint* instructions generate an Undefined Instruction exception instead of being treated as NOP

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open.

Description

The ARM Architecture specifies that ARM opcodes of the form `11110 100x001 xxxx xxxx xxxx xxxx xxxx` are *Unallocated memory hint (treat as NOP)* if the core supports the MP extensions, as the Cortex-A9 does.

The errata is that the Cortex-A9 generates an Undefined Instruction exception when bits [15:12] of the instruction encoding are different from `4'b1111`, instead of treating the instruction as a NOP.

Implications

Because of the erratum, an unexpected Undefined Instruction exception might be generated.

In practice, this erratum is unlikely to cause any significant issue because such instruction encodings are not supposed to be generated by any compiler, nor used by any handcrafted program.

Workaround

The workaround for this erratum is to modify the instruction encoding with bits[15:12]=`4'b1111`, so that the Cortex-A9 treats the instruction properly as a NOP.

If it is not possible to modify the instruction encoding as described, the Undefined Instruction exception handler has to cope with this case, and emulate the expected behavior of the instruction, that is, it must do nothing (NOP), before returning to normal program execution.

2.5.33 (761321) MRC and MCR are not counted in event `0x68`

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open.

Description

Event `0x68` counts the total number of instructions passing through the register rename pipeline stage. The erratum is that MRC and MCR instructions are not counted in this event.

The event is also reported externally on PMUEVENT[9:8], which suffers from the same defect.

Implications

The implication of this erratum is that the values of event `0x68` and PMUEVENT[9:8] are imprecise, omitting the number of MCR and MRC instructions. The inaccuracy of the total count depends on the rate of MRC and MCR instructions in the code.

Workaround

No workaround is possible to achieve the required functionality of counting precisely how many instructions are passing through the register rename pipeline stage when the code contains some MRC or MCR instructions.

2.5.34 (764319) Read accesses to DBGPRSR and DBGOSLSR might generate an unexpected Undefined Instruction exception

Status

Affects:	Product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All revisions. Open

Description

CP14 read accesses to the DBGPRSR and DBGOSLSR registers generate an unexpected Undefined Instruction exception when the DBGSWENABLE external pin is set to 0, even when the CP14 accesses are performed from a privileged mode.

Implications

Because of the erratum, the DBGPRSR and DBGOSLSR registers are not accessible when DBGSWENABLE=0.

This is unlikely to cause any significant issue in Cortex-A9 based systems because these accesses are mainly intended to be used as part of debug over powerdown sequences, and the Cortex-A9 does not support this feature.

Workaround

The workaround for this erratum is to temporarily set the DBGSWENABLE bit to 1 so that the DBGPRSR and DBGOSLSR registers can be accessed as expected.

There is no other workaround for this erratum.

2.5.35 (771221) PLD instructions might allocate data in the Data Cache regardless of the Cache Enable bit value

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Category C

Fault Status: Present in: All r0, r1, r2 and r3 revisions Fixed in r4p0

Description

PLD instructions prefetch and allocate any data marked as Write-Back (either Write-Allocate or Non-Write-Allocate, Shared or Non-Shared), regardless of the processor configuration settings, including the Data Cache Enable bit value.

Implications

Because of this erratum, unexpected memory cacheability aliasing is created which might result in various data consistency issues.

In practice, this erratum is unlikely to cause any significant issue. The Data Cache is likely to be enabled as soon as possible in most systems, and not dynamically modified. Therefore, this erratum is likely to impact only boot-up code. This code is usually carefully controlled and does not usually contain any PLD instruction while Data Cache is not enabled.

Workaround

If this erratum impacts a system, a software workaround is available which is to set bit [20] in the undocumented Control register, which is placed in CP15 c15 0 c0 1.

This bit needs to be written with the following Read/Modify/Write code sequence:

```
MRC p15,0,r0,c15,c0,1

ORR r0,r0,#0x00100000

MCR p15,0,r0,c15,c0,1
```

Setting this bit causes all PLD instructions to be treated as NOPs, with the consequence that code sequences that usually use the PLDs, such as the memcpy() routine, might suffer from a visible performance drop. Therefore, if this workaround is applied, ARM strongly recommends restricting its use to periods of time where the Data Cache is disabled.

2.5.36 (771224) Visibility of Debug Enable access rights to enable/disable tracing is not ensured by an ISB

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

According to the ARM architecture, any change in the Authentication Status Register should be made visible to the processor after an exception entry or return, or an ISB.

Although this is correctly achieved for all debug-related features, the ISB is not sufficient to make the changes visible to the trace flow. As a consequence, the **WPTTRACEPROHIBITEDn** signal(s) remain stuck to their old value up to the next exception entry or return, or to the next serial branch, even when an ISB executes.

A serial branch is one of the following:

- Data processing to PC with the S bit set (for example, MOVS pc, r14)
- LDM pc ^

Implications

Because of the erratum, the trace flow might not start, nor stop, as expected by the program.

Workaround

To work around the erratum, the ISB must be replaced by one of the events causing the change to be visible. In particular, replacing the ISB by a MOVS PC to the next instruction will achieve the correct functionality.

2.5.37 (775419) PMU event 0x0A (exception return) might count twice the LDM PC ^ instructions with base address register write-back

Status

Affects:	product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All revisions. Open

Description

The LDM PC ^ instructions with base address register write-back might be counted twice in the PMU event 0x0A, which is counting the number of exception returns.

The associated PMUEVENT[11] signal is also affected by this erratum, and might be asserted twice by a single LDM PC ^ with base address register write-back.

Implications

Because of the erratum, the count of exception returns is imprecise. The error rate depends on the ratio between exception returns of the form LDM PC ^ with base address register write-back and the total number of exceptions returns.

Workaround

There is no workaround to this erratum.

2.5.38 (795769) “Write Context ID” event is updated on read access

Status

Affects:	product Cortex-A9, Cortex-A9 MPCore.
Fault Type:	Programmer Category C
Fault Status:	Present in: All revisions. Open

Description

When selected, the Write Context ID event (event `0x0B`) of the *Performance Monitoring Unit* (PMU) increments a counter whenever an instruction that writes to the Context ID register, CONTEXTIDR, is architecturally executed. However, this erratum means that an instruction that reads the Context ID register also updates this counter.

Configurations affected

This erratum affects all configurations of the processor.

Conditions

The erratum can happen under the following conditions:

1. A PMU counter is enabled, by setting the PMCNTENSET.P*x* bit to 1 (*x* identifies a single event counter, and takes a value from 0 to 7).
2. The “Write Context ID” event is mapped to this selected PMU counter:
   a. The chosen PMU counter is selected, by setting PMSELR.SEL to x (the same value as in condition 1).
   b. The "Write Context ID" event is mapped to this selected PMU, by setting PMXEVTYPER.evtCount to `0x0B`.
3. The PMU is enabled, by setting the PMCR.E bit to 1.
4. A read access occurs to the CONTEXTIDR.

In this situation the PMU updates the counter when it should not.

Implications

The erratum affects the accuracy of the “Write Context ID” event, and its associated PMUEVENT[12] output signal.

Workaround

There is no workaround for this erratum.

2.5.39 (799770) DBGPRSR Sticky Reset status bit is set to 1 by the CPU debug reset instead of by the CPU non-debug reset

Status

Affects: product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

DBGPRSR.SR, bit [3], is the Sticky Reset status bit. The ARM architecture specifies that the processor sets this bit to 1 when the non-debug logic of the processor is in reset state.

Because of this erratum, the Cortex-A9 processor sets this bit to 1 when the debug logic of the processor is in reset state, instead of when the non-debug logic of the processor is in reset state.

Configurations affected

The erratum affects all configurations of the processor.

Implications

Because of the erratum:

- DBGPRSR.SR might not be set to 1 when it should, when the non-debug logic of the processor is in reset state.
- DBGPRSR.SR might be set to 1 when it should not, when the debug logic of the processor is in reset state.

In both cases, the DBGPRSR.SR bit value might be corrupted, which might prevent the debug logic from correctly detecting when the non-debug logic of the processor has been reset.

Workaround

There is no workaround to this erratum.